

Virtual Service Environment Practice for Fulfillment

OUTLINE

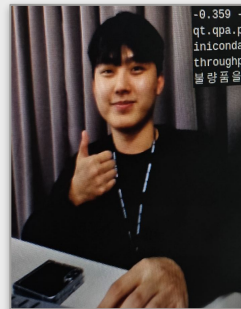
- 
1. Project Overview
 2. Project Structure
 3. Implementation Features
 4. Development Challenges
 5. Conclusion
 6. Development Experience Sharing and Q&A

Introduction

Members



윤민식



정승환



장준하

| Project Overview

Objectives of the Project

- 작업 자동화
- 프로토타입 개발
- 디지털 팩토리 구현
- ROS2 기반 설계

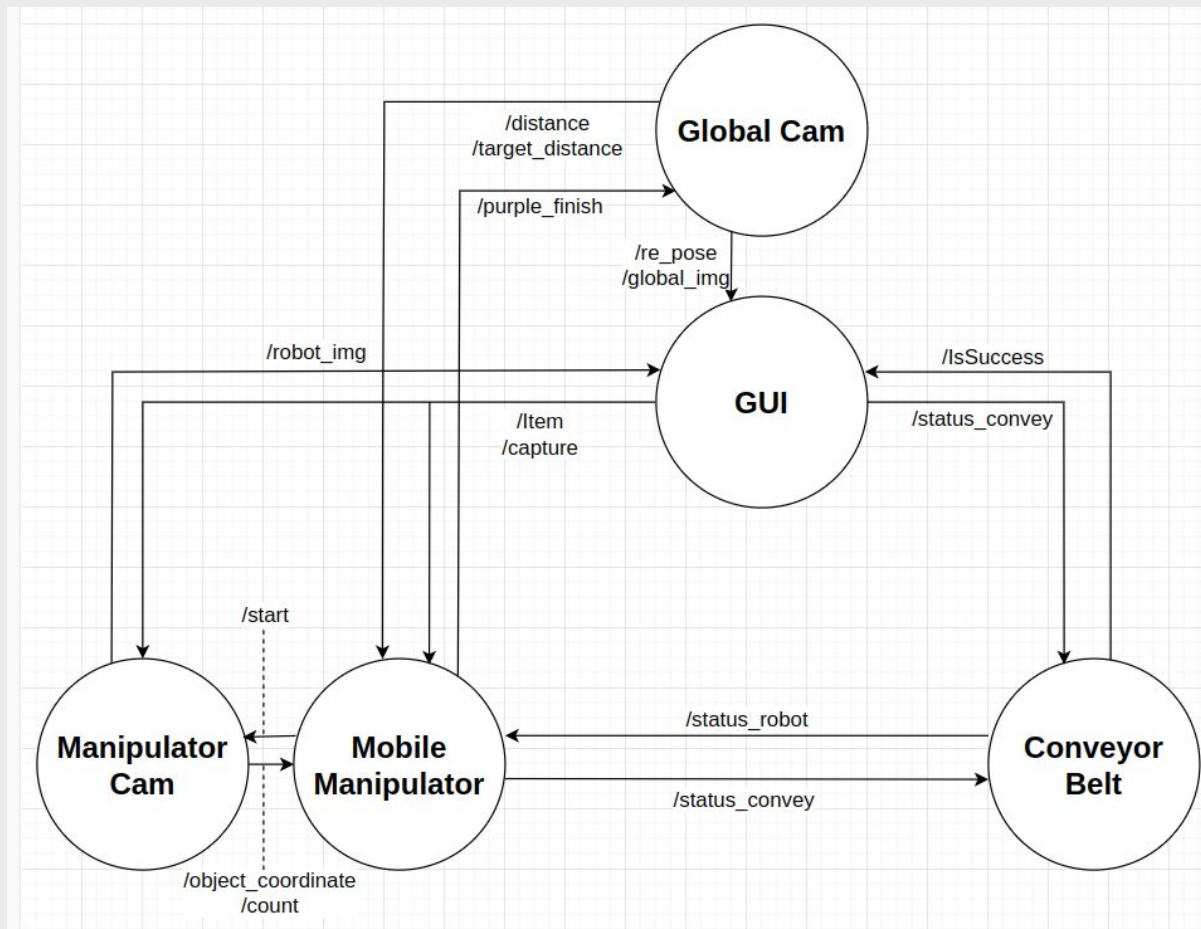
| Project Overview

Key Features and Workflow

- 자동화 로봇 프로토타입
- 가상 환경 구축
- 모듈형 ROS2 노드 설계
- 테스트 프레임워크

Project Structure

Node Design



| Project Structure

Message Types

/status_convey

/distance

/Item

/count

/status_robot

/IsSuccess

/target_distance

/purple_finish

/re_pose

/capture

/start

String

/global_img

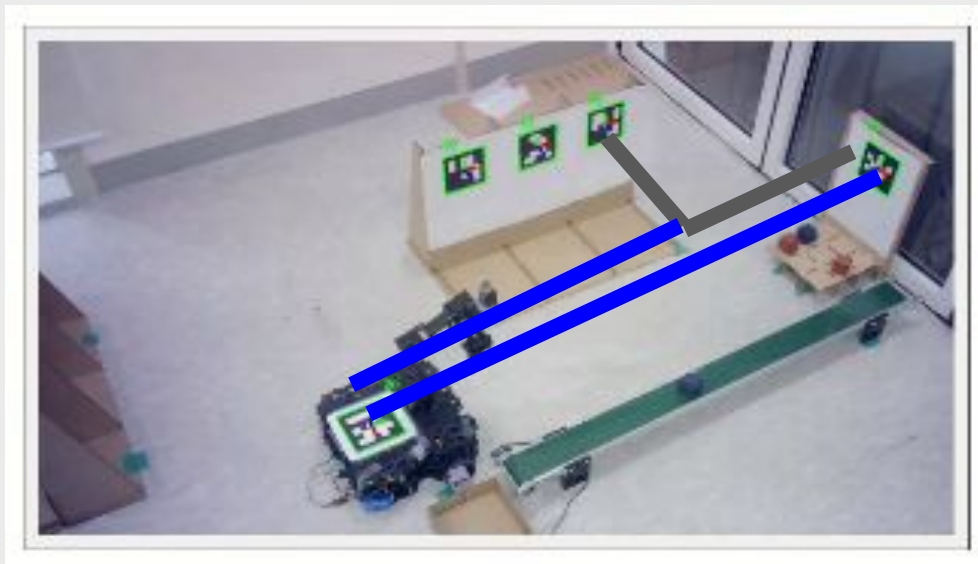
/robot_img

**Compressed
Image**

/object_coordinate

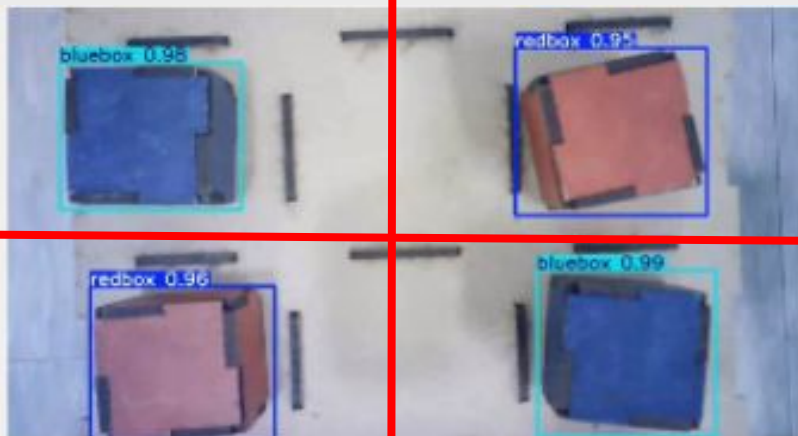
Point

Logic



```
[INFO] [1735629294.589357044] [pick_and_place]: distance:0.22873370020443684
[INFO] [1735629294.628049911] [pick_and_place]: 터틀봇 전진
[INFO] [1735629294.628972525] [pick_and_place]: distance:0.2166954160826171
[INFO] [1735629294.670255554] [pick_and_place]: 터틀봇 전진
[INFO] [1735629294.671177524] [pick_and_place]: distance:0.21383730742217277
[INFO] [1735629294.713225715] [pick_and_place]: 터틀봇 전진
[INFO] [1735629294.714264701] [pick_and_place]: distance:0.20183763479209327
[INFO] [1735629294.752127561] [pick_and_place]: 터틀봇 전진
[INFO] [1735629294.753920106] [pick_and_place]: distance:0.20748638541096032
[INFO] [1735629294.791636118] [pick_and_place]: 터틀봇 전진
[INFO] [1735629294.792598675] [pick_and_place]: distance:0.21248246994676745
```


Logic



```
def callback_item(self, msg):
    jobs = msg.data.split()
    for job in jobs:
        job_count = job.split(',')
        if job_count[0] == 'red':
            self.get_red_count = int(job_count[1])
        elif job_count[0] == 'blue':
            self.get_blue_count = int(job_count[1])
        elif job_count[0] == 'goal':
            self.goal = int(job_count[1])
    self.total_count = self.get_red_count + self.get_blue_count
```

```
if self.get_blue_count > 0 or self.get_red_count > 0:
    self.get_logger().error("박스가 부족합니다. 실행 불가")
    sys.exit()
else:
    self.move_arm()
```

| Implementation Features

Advantages

- Manipulator Motion Planning 최적화

로봇 팔이 유연하고 자연스러운 곡선을 그리며 이동

조인트 값만으로 동작을 구현 가능

- 높은 정확도를 가진 ARUCO marker 변환
- red&blue box detection시 0.95 이상의 confidence score

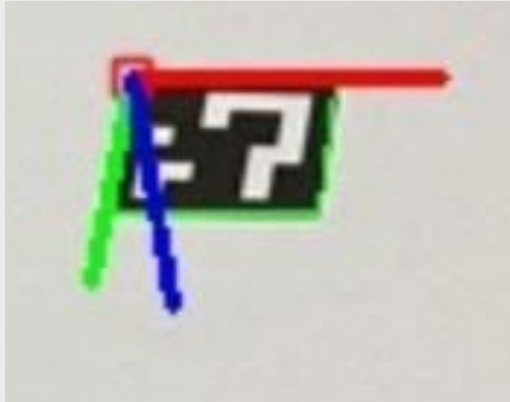
| Implementation Features

Disadvantages

- pick and place시 좌표변환 미사용으로 인한 오차
- purple box detection 실패

| Development Challenges

Most Time-Consuming Aspects & Resolution Process

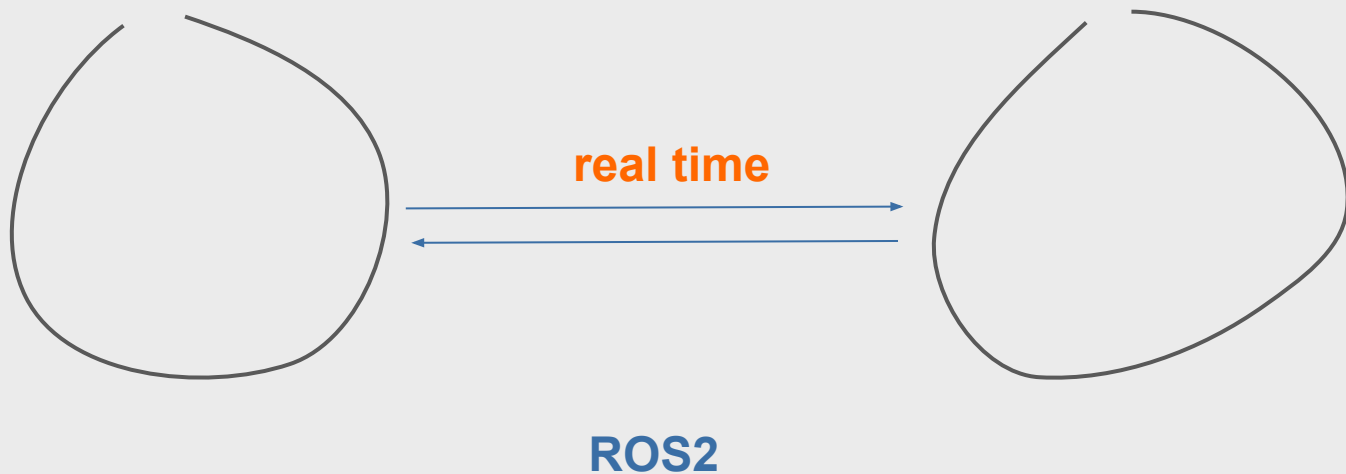


$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

ArUco marker

| Development Challenges

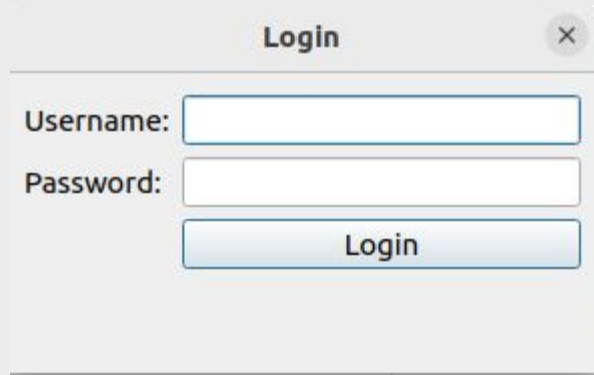
Most Time-Consuming Aspects & Resolution Process



엉성하게 돌아가는 ROS Node 간 실시간 통신

| Requirement

GUI



A screenshot of a 'Login' dialog box. The dialog has a title bar with the word 'Login' and a close button (X). Inside, there are two input fields: 'Username:' and 'Password:'. Below the password field is a 'Login' button.

log in

| Requirement

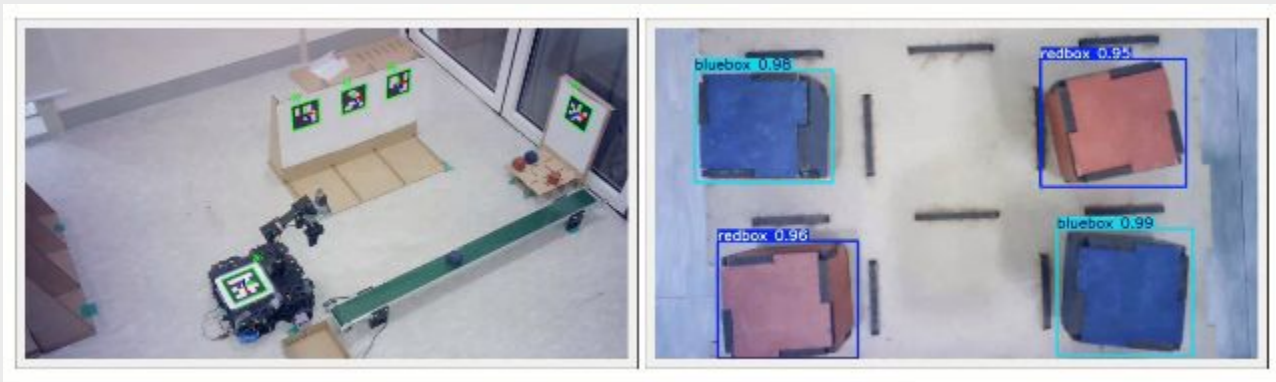
GUI

```
def send_mail(self):  
    connection = smtplib.SMTP("smtp.gmail.com")  
    connection.starttls()  
    connection.login(user=self.email, password=self.pwd)  
    connection.sendmail(  
        from_addr="warning@warning.ing",  
        to_addrs="jang4660893@naver.com",  
        msg="Subject:ERRORRRRRRR!!!\n\nYour conveyorbelt is effeted by something.\n\n\nPlease  
    )  
    connection.close()
```

비상상황 발생시 담당자에게 메일
발송

| Requirement

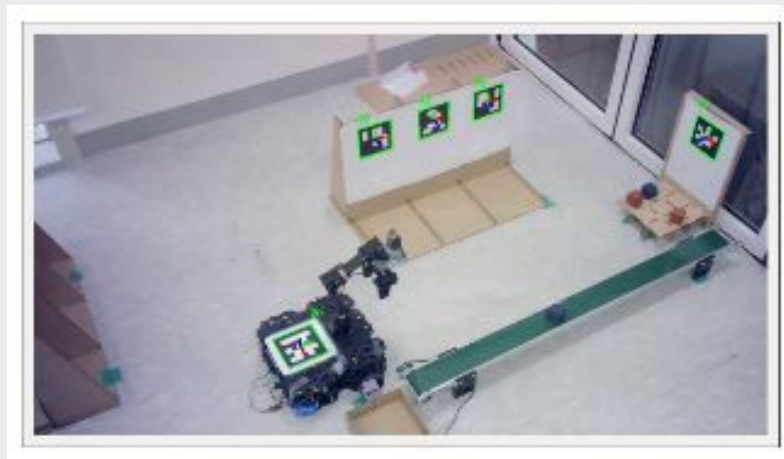
GUI



글로벌 카메라와 yolo 인식 화면
표시

| Requirement

GUI



AuRco 상대 위치와
각도

| Requirement

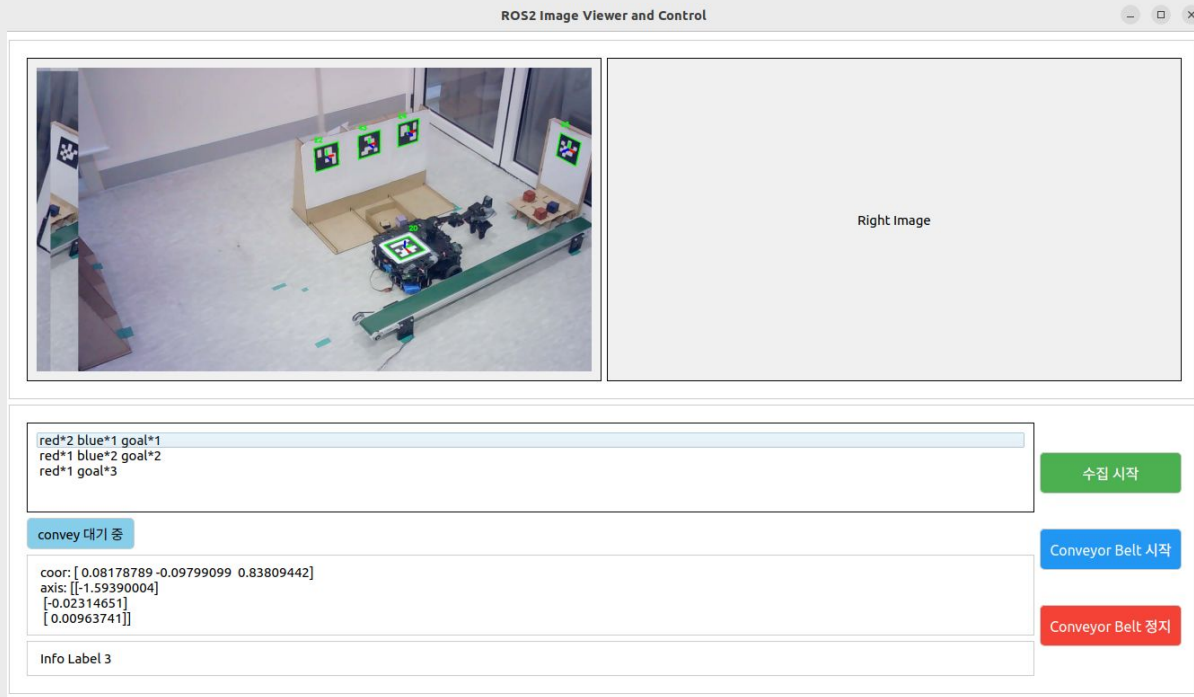
GUI



conveyor 수동 조작 버튼

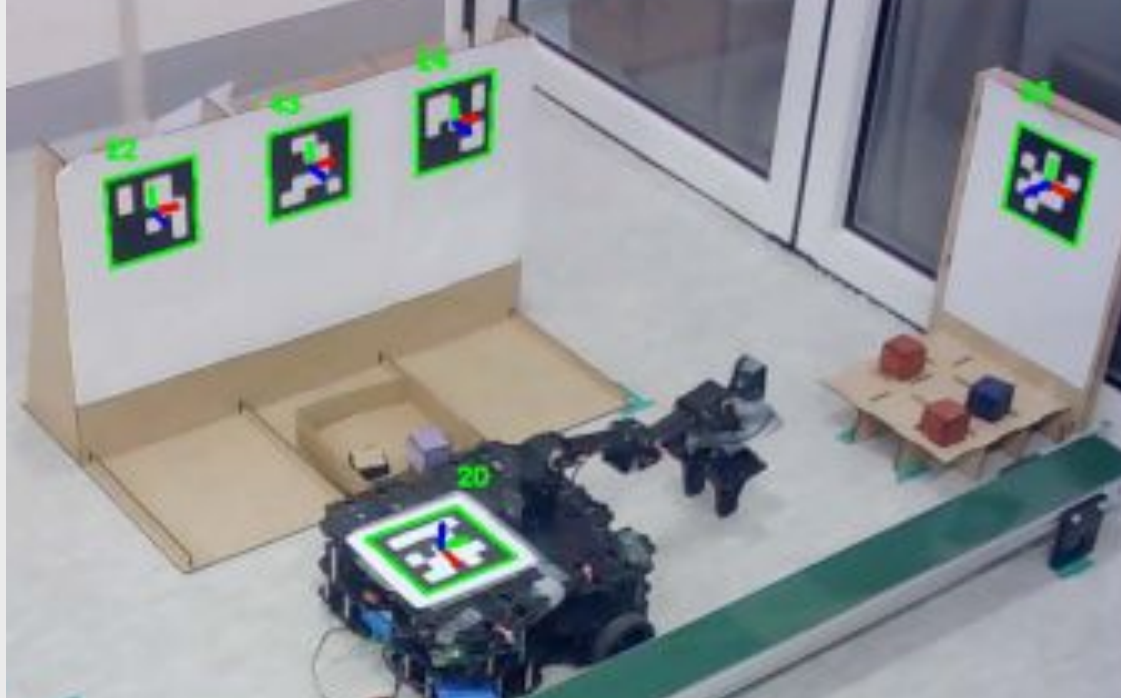
Conclusion

GUI (Image)



Conclusion

ArUco marker



Conclusion

Conveyor belt

```
def moving_signal(self):
    while True:
        try:
            if self.ser.in_waiting > 0:
                rx = self.ser.read().decode('ascii')
                if self.ismoving and rx == '.':
                    self.ismoving = False

                if self.signal == 'start':
                    self.get_logger().info('success')
                    msg1 = String()
                    msg1.data = 'finish'
                    self.pub_start.publish(msg1)

                    msg = String()
                    msg.data = 'success'
                    self.pub_success.publish(msg)


            except OSError:
                self.send_mail()
                self.get_logger().error('cannot connect conveyor belt')
                sys.exit(0)
                break
```

☆ **ERRORRRRRRR!!!** 

^ 보낸사람 ymsix0622@gmail.com

받는사람 (받는데없음)

2024년 12월 31일 (화) 오후 5:51

 영어 → 한국어 번역하기

Your conveyorbelt is effected by something.

Please check your belt.

| Conclusion

Mobile Manipulator

시연으로

대체

| Requirement

GUI

GUI	log in	✓
	비상상황 발생시 등록된 담당자에게 메일로 내용 알림	✓
	글로벌 카메라와 yolo인식 결과 화면 표시	✓
	AuRco 상대 위치와 각도를 표시하시오	✓
	conveyor 수동 조작 버튼	✓

| Requirement

Conveyor belt

Conveyor	정해진 길이만큼 이동	✓
	동작 완료를 외부에서 인식	✓
	동작 중 USB가 뽑혔을 시 인식	✓

| Requirement

AuRco Marker

AuRco	선정한 기준 좌표계 에서의 로봇의 위치	✓
	선정한 기준 좌표계 에서의 로봇의 방향	✓
	글로벌 카메라의 위치가 변경되도 로봇의 위치와 방향이 변화가 없나?	✓
	이동 명령에서 로봇이 일정시간 이내에 타겟 위치에 못가면 오류 발생	
	동작중 마커가 일정시간 이상 가려질 경우 오류상황 인지	

| Requirement

Digital Twin & 연동

Digital Twin	현재 code의 수정없이 yolo 학습 data 수집	✓
	학습된 결과로 로봇팔 동작	
연동	GUI Job 설정 후 로봇이 테이블로 이동한다.	✓
	테이블 앞에 도착 후 yolo 인식 수행함	✓
	yolo 인식 후 해당 박스 잡아서 컨베이어에 올려 놓는다.	✓
	컨베이어 벨트에 모든 박스가 올라온 후 바구니에 박스를 담는다	✓
	바구니를 집어서 골에 옮겨 놓는다.	✓
	비상정지 기능 동작 하는가?	

| Requirement

Mobile Manipulator

Manipulator + YOLO	로봇팔로 학습데이터를 자동으로 수집	✓
	YOLO로 인식한 결과를 이용하여 로봇팔을 움직여서 박스를 잡는다	✓
	컨베이어 벨트 위에 박스를 올려 놓는다	✓
	바구니에 박스를 모두 담는다	✓
	테이블에 박스의 개수가 부족하면 오류로 인식한다.	✓
	로봇의 위치를 +1 cm 정도 앞뒤로 이동	✓

Development Experience Sharing & Q&A

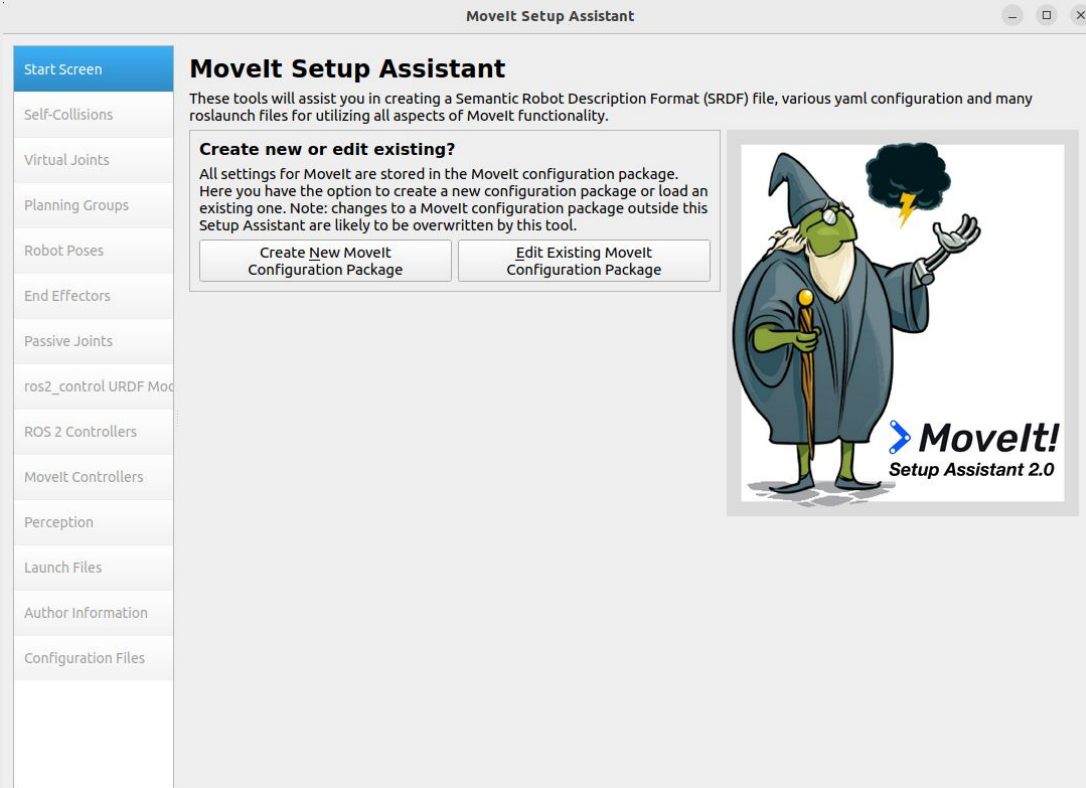
Development Experience Sharing

- MoveIt

로봇의 모션 계획, 조작, 충돌 감지,

시뮬레이션 및 제어를 지원하는 ROS 기반

로봇 모션 플래닝 프레임워크



Development Experience Sharing & Q&A

Development Experience Sharing

- Moveit을 사용하여 설계하기

- 1) 로봇의 URDF / XACRO 설계

- 로봇의 관성 정보를 정확히 정의
- 충돌 범위 및 joint상의 결합 위치 조정
- CAD로 실제 로봇 설계

- 2) Moveit Setup Assistant 사용

- 충돌 범위, Group, End Effector를 지정
- Ros2 Control 설정 = 로봇의 제어 인터페이스 구성

```
<joint name="${prefix}base_joint" type="fixed">
  <parent link="${prefix}base_footprint"/>
  <child link="${prefix}base_link" />
  <origin xyz="0 0 0.010" rpy="0 0 0"/>
</joint>

<link name="${prefix}base_link">
  <visual>
    <origin xyz="-0.064 0 0.0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="${meshes_file_direction}/base.stl" scale="0.001
.001 0.001"/>
    </geometry>
    <material name="light_black"/>
  </visual>

  <collision>
    <origin xyz="-0.064 0 0.047" rpy="0 0 0"/>
    <geometry>
      <box size="0.266 0.266 0.094"/>
    </geometry>
  </collision>

  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="1.3729096e+00"/>
    <inertia ixx="8.7002718e-03" ixy="-4.7576583e-05"
xyz="1.1160499e-04"
            iyy="8.6195418e-03" iyz="-3.5422299e-06"
            izz="1.4612727e-02" />
  </inertial>
</link>

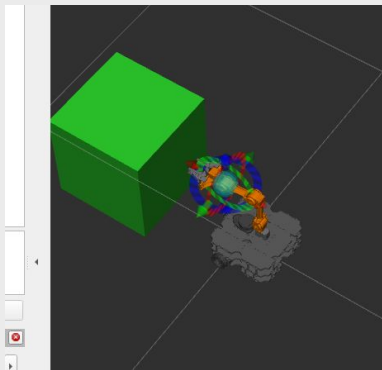
<joint name="${prefix}wheel_left_joint" type="continuous">
  <parent link="${prefix}base_link"/>
  <child link="${prefix}wheel_left_link"/>
  <origin xyz="0.0 0.144 0.023" rpy="-1.57 0 0"/>
  <axis xyz="0 0 1"/>
</joint>
```

Development Experience Sharing & Q&A

Development Experience Sharing

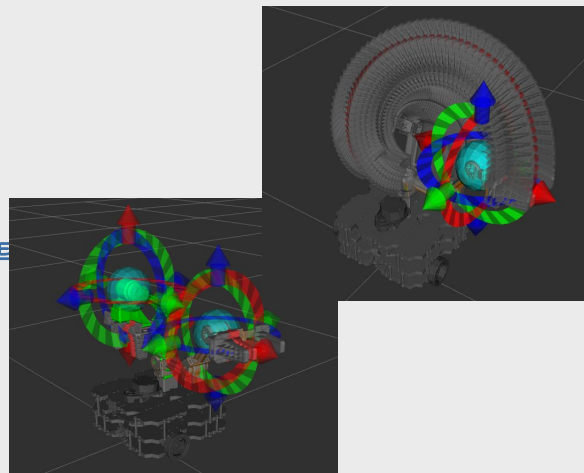
3) Gazebo 및 Rviz 환경 설정

- 현실적인 로봇 동작을 위해 물리 속성 반영
- 외력, 마찰 등 세부 물리값 입력
- Ros2 버전에서는
python \Rightarrow C++ 의 활용도가 높아짐



4) Moveit 실행

- 다양한 Inverse Kinematics를 적용하며 작업에 따른 가장 적합한 알고리즘 선택



| Development Experience Sharing & Q&A

Development Experience Sharing

- 설계의 어려움

실제 환경에 적합한 설계를 구현하려면 **URDF / XACRO**에 대한 높은 숙련도가 필수적이며, 물리 속성을 정밀하게 계산하는 과정 또한 상당히 까다롭습니다.

- 그러나, 로봇 제어 분야에서 큰 도움을 받을 수 있으며, 자신의 로봇을 더욱 정교하고 효율적인 시스템 구현이 가능해집니다.

Development Experience Sharing & Q&A

Development Experience Sharing

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$
$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix}$$
$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Euler angle

- **xyz**
- **zyx**
- **zyz**

·
·

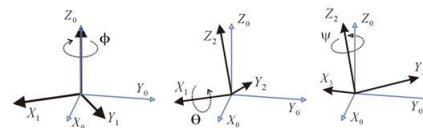
Fixed angle

- **xyz**
- **zyx**
- **zyz**

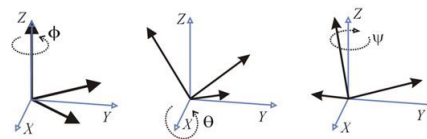
·
·

Euler Angles vs. Fixed Angles

- z-x-z Euler angles: (-60,30,45)



- z-x-z fixed angles: (45,30,-60)



Rotation Matrix

Development Experience Sharing & Q&A

Development Experience Sharing

```
def compute_relative_pose(self, goal_pose, robot_pose):  
    t_goal = goal_pose[0:3]  
    rvec_goal = goal_pose[3:]  
  
    R_goal, _ = cv2.Rodrigues(rvec_goal)  
  
    T_goal = np.eye(4)  
    T_goal[:3, :3] = R_goal  
    T_goal[:3, 3] = t_goal  
  
    T_goal_inv = np.linalg.inv(T_goal)  
  
    t_robot = robot_pose[0:3]  
    rvec_robot = robot_pose[3:]  
    R_robot, _ = cv2.Rodrigues(rvec_robot)  
  
    T_robot = np.eye(4)  
    T_robot[:3, :3] = R_robot  
    T_robot[:3, 3] = t_robot  
  
    T_relative = T_goal_inv @ T_robot  
  
    R_relative = T_relative[:3, :3]  
    t_relative = T_relative[:3, 3]  
  
    rvec_relative, _ = cv2.Rodrigues(R_relative)  
  
    return t_relative, rvec_relative
```

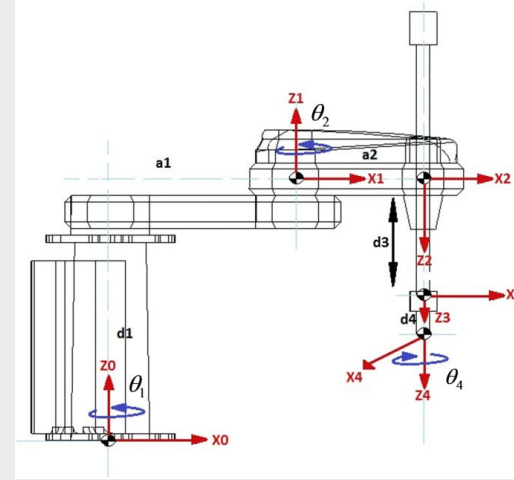
Rotation Matrix

Development Experience Sharing & Q&A

Development Experience Sharing

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation Matrix
Translate Matrix



Scara robot
DH-parameter

Coordinate Transform

Development Experience Sharing & Q&A

Development Experience Sharing

- **Real time** 통신이 어려울 때?

-> 최대한 실시간 통신을 흉내내는 방향을 선택하자

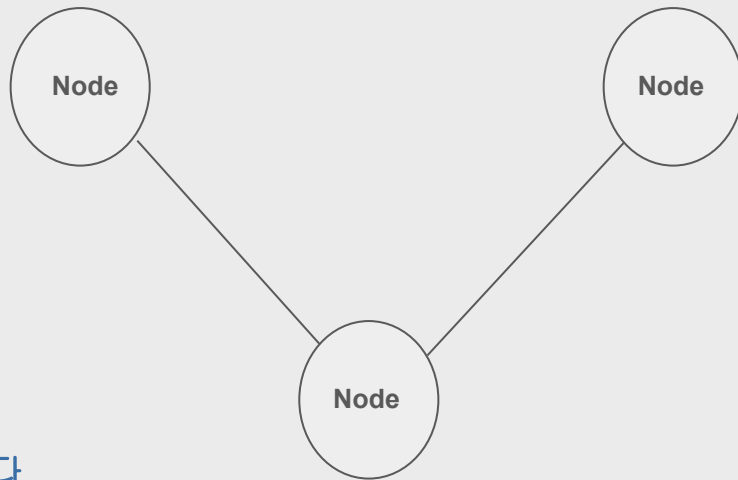
-> 자잘자잘한 **Topic** 메시지를 만들어 다른 노드에서

특정 **Topic** 을 받을 때까지 실행되지 않는 방식을 애용하자..!

-> 많은 **Topic**은 클린 코드에 방해가 될 수 있지만

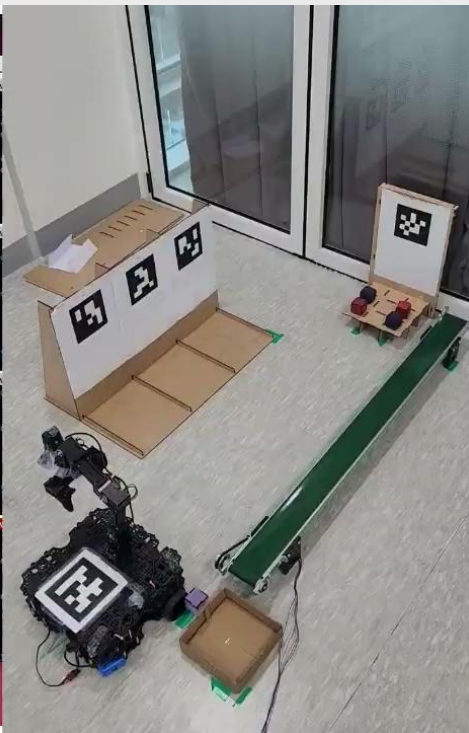
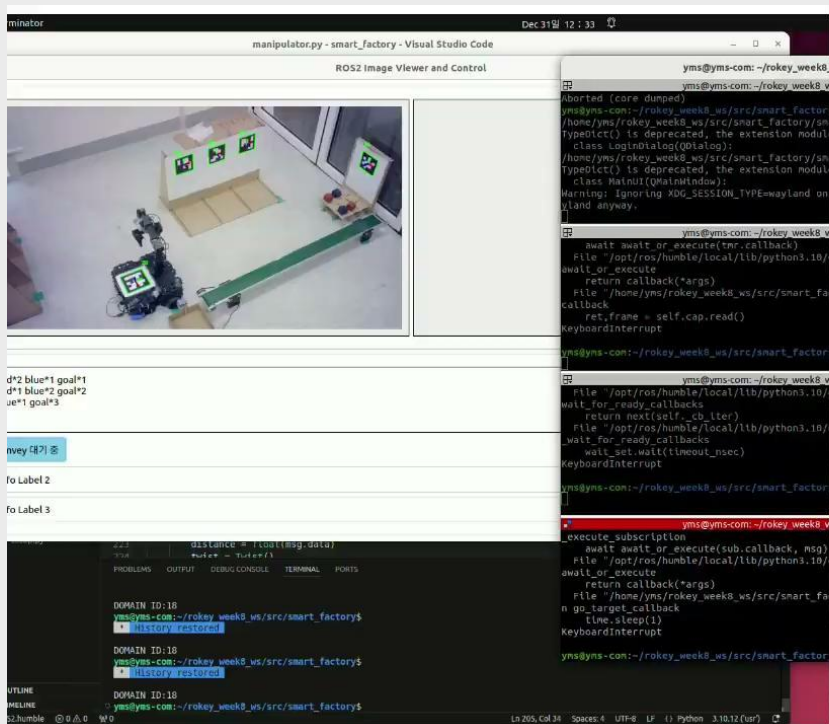
시간이 부족할 땐 새로운 컨트롤 노드를 만들어 통제하는 것보다

훨씬 효율적일 수 있다.



Conclusion

result video



Q&A

감사합니다