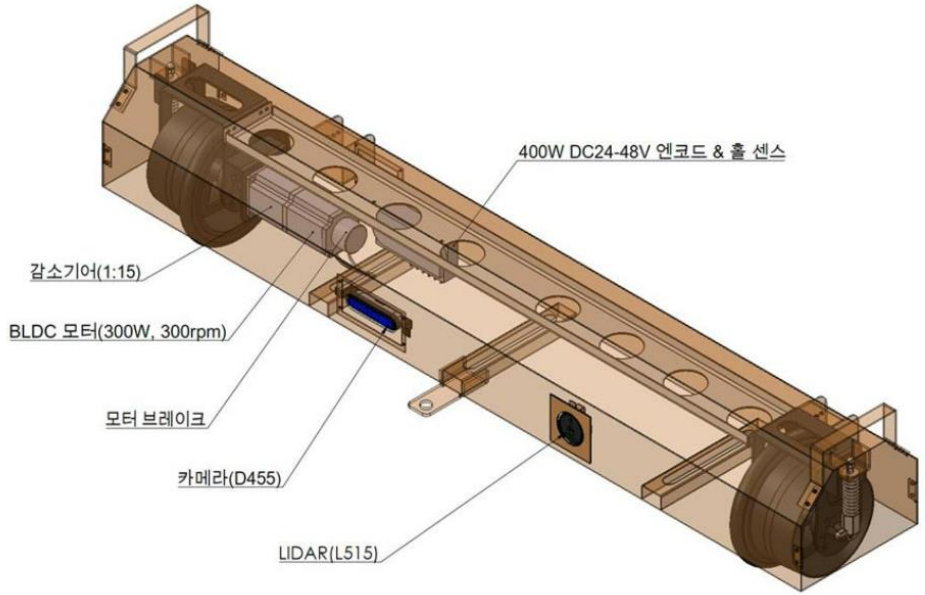
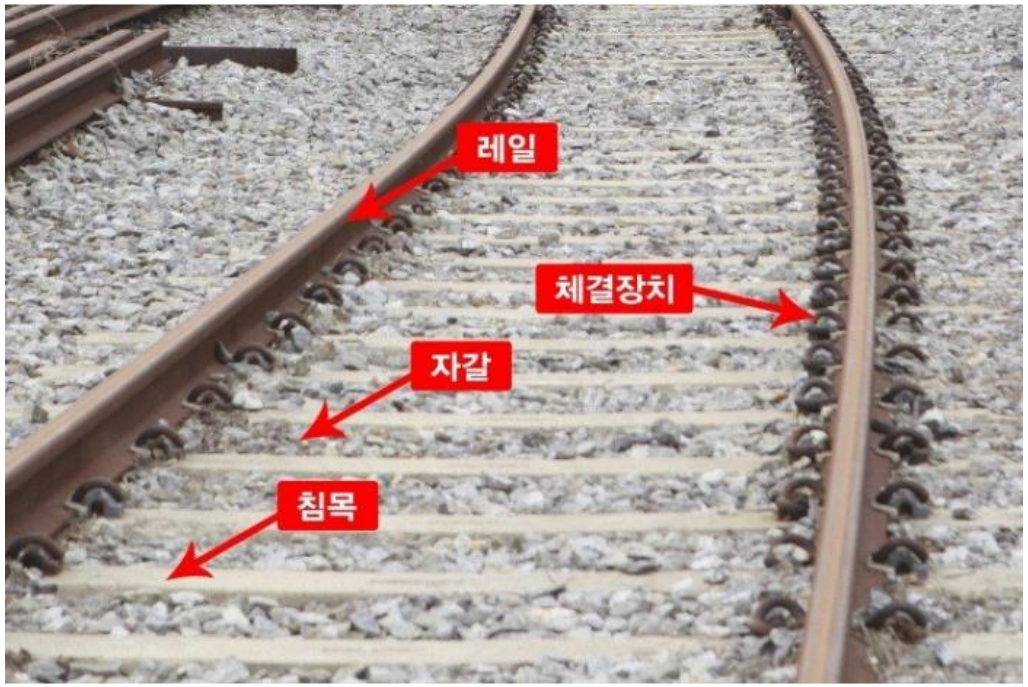
01. 레일로봇 프로젝트 소개





➢ 자율 궤도 운행 이송로봇 개발

- ✓ 한국 철도 기술 연구원에서 개발 진행중인 로봇
- ✓ 철도 선로 작업자의 산업재해 예방과 열악한 작업 환경 개선을 위해 인공지능 기반의 자율 궤도 운행 이송로봇 기술
- ✓ 오픈 소스 기반의 로봇 운영체제(ROS,Robot Operating System) 으로 프로그램 개발 진행

출처 : http://www.ctman.kr/paper/news/print.php?newsno=25867 (자율 궤도 운행 이송로봇 관련 기사)
https://m.post.naver.com/viewer/postView.nhn?volumeNo (철도기술연구원 블로그 – 궤도 설명)

02. 로봇 필요성 및 기능 소개

➢ 선로 작업자의 작업 환경 개선

- ✓ 철도 현장 특성 상 무거운 도구와 장비가 많음 -> 무거운 중량물 운반에 활용 가능
- ✓ 로봇이 스스로 이동하여 열차 접근 및 선로 감시와 경보
- ✓ 선로 시설물에 대한 복합 점검 지원 및 유지보수 데이터 관리 가능 -> 로봇의 데이터를 DB 연동, 관제시스템 활용

➢ 선로 작업자의 산업 재해 예방

- ✓ 선로 작업자 사고의 위험도 평가에 따르면 열차와 선로 작업자의 접촉 시나리오가 가장 높음
- ✓ 작업자 통보 강화, 열차 감시자 배치 의무화, 작업자 착각 오류 예방
  -> 추가적인 인력과 시간 투입, 작업계획 수립 시 추가 안전 활동에 많은 시간과 예산이 소요되어 한계점
- ✓ 따라서 한정적인 인력과 시간으로 선로 작업 시 산업 재해 예방을 위해 로봇 활용

➢ 경제적 및 운영 효율성

- ✓ 고속철도 선로 등 주요 노선에 대해서는 자동으로 선로 점검이 진행 (선로 검측차 및 장비)
  -> 선로 검측 차량 구입 대당 가격은 약 41억원 수준, 충분한 수의 차량 보유 어려움 -> 수작업 선로 점검
- ✓ 선로 검측 차량보다 저렴하며 활용성 높은 로봇을 사용해 효율성 극대화

03. 로봇에 사용된 딥러닝 설명

➢ 작업자 추종, 자율 복합 점검 지원

- ✓ 카메라, 라이다, 센서 등을 통해 수집된 데이터를 기반으로 학습 데이터 구축
- ✓ Object Detection Model 인 YOLOv5를 활용하고 학습하여 작업자를 인식 및 장애물 탐지
- ✓ 작업자 인식 및 일정 간격을 유지하며 추종 -> 하드웨어 제어는 ROS 사용 (Python 기반) , depth camera 사용
- ✓ 자율 복합 점검은 Rail-Way Segmentation을 추출 및 학습 후 이상 탐지 개념

➢ ROS 기반 궤도 자율 주행

- ✓ SLAM(Simultaneous Localization And Mapping) : 지도를 그림과 동시에 로봇의 위치를 추정
- ✓ Localization : 센서,지도,위치,경로 등을 사용해 위치 파악
- ✓ Navigation : 원하는 경로를 찾는 방법 (목표 지점 설정 -> 해당 지점까지 자율 주행)

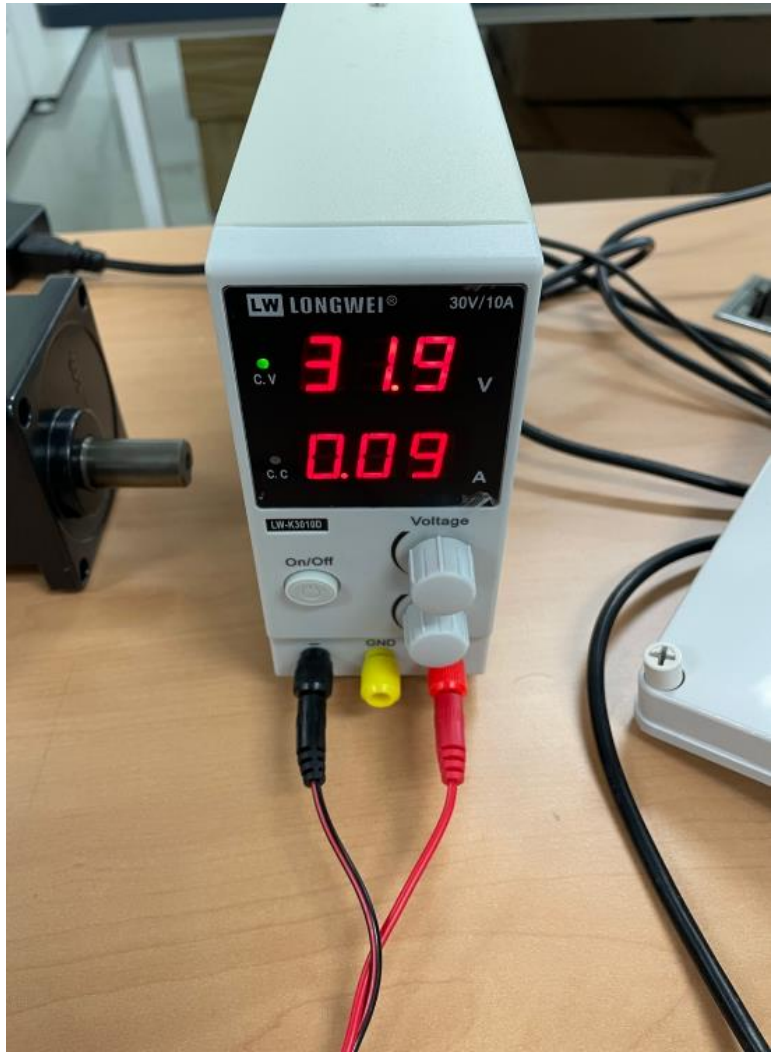
➢ 향후 계획

- ✓ 로봇 성능 개선 및 지정 위치 자율 이동 주행, 열차 접근 등 이상 상황 인식
- ✓ 선로 탐지 기술과 함께 시설물 검측 장비와 작업 도구 등 다양한 임무 모듈 탑재
- ✓ 자율 복합 점검 연계 인터페이스와 데이터 수집,처리,전송 기술 개발 연구 및 이를 통한 데이터 분석 진행
- ✓ 실제 환경과 유사한 시뮬레이션 환경 개발

출처 : https://github.com/yeonsoo98/railload_robot (개인 깃허브)

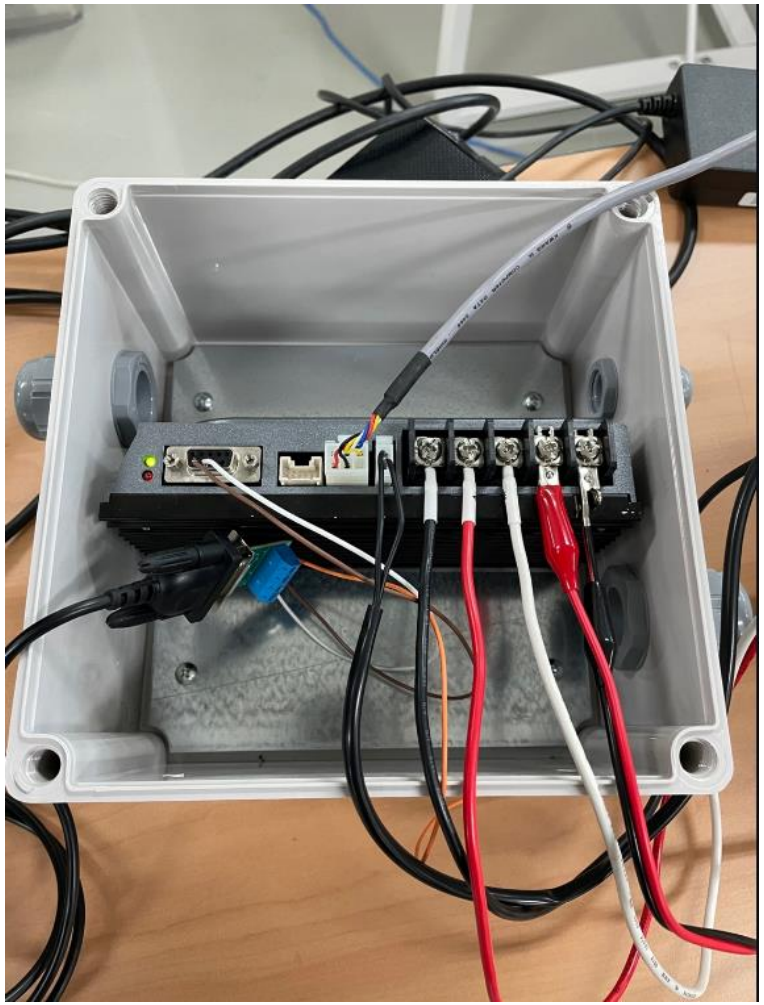
레일로봇 구성


ument (no answer)
rument (no answer)
rument (no answer)
CTRL-C to quit
currently: speed 0.5 turn 1.0
Run CTest


LW LONGWEI
30V/10A
C.V
31.9
V
C.C
0.09
A
LW-K3010D
Voltage
On/Off
GND

<모터 구동 >

레일로봇 구성

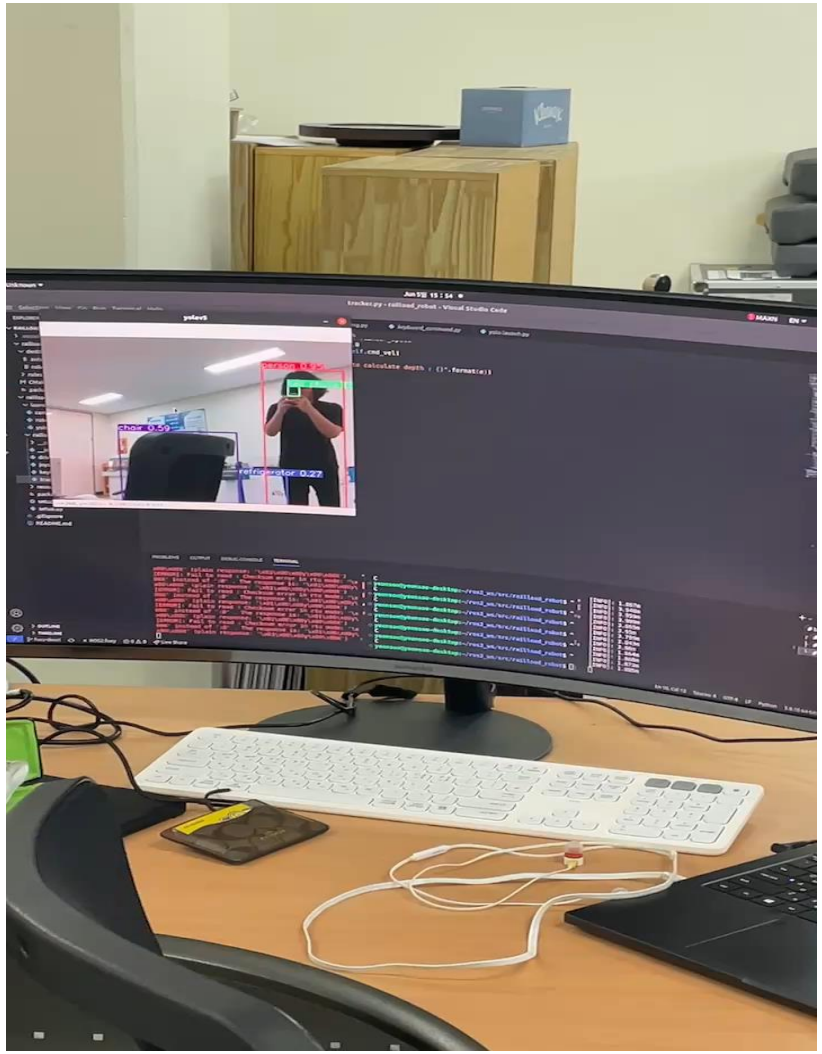


<모터 구동 >

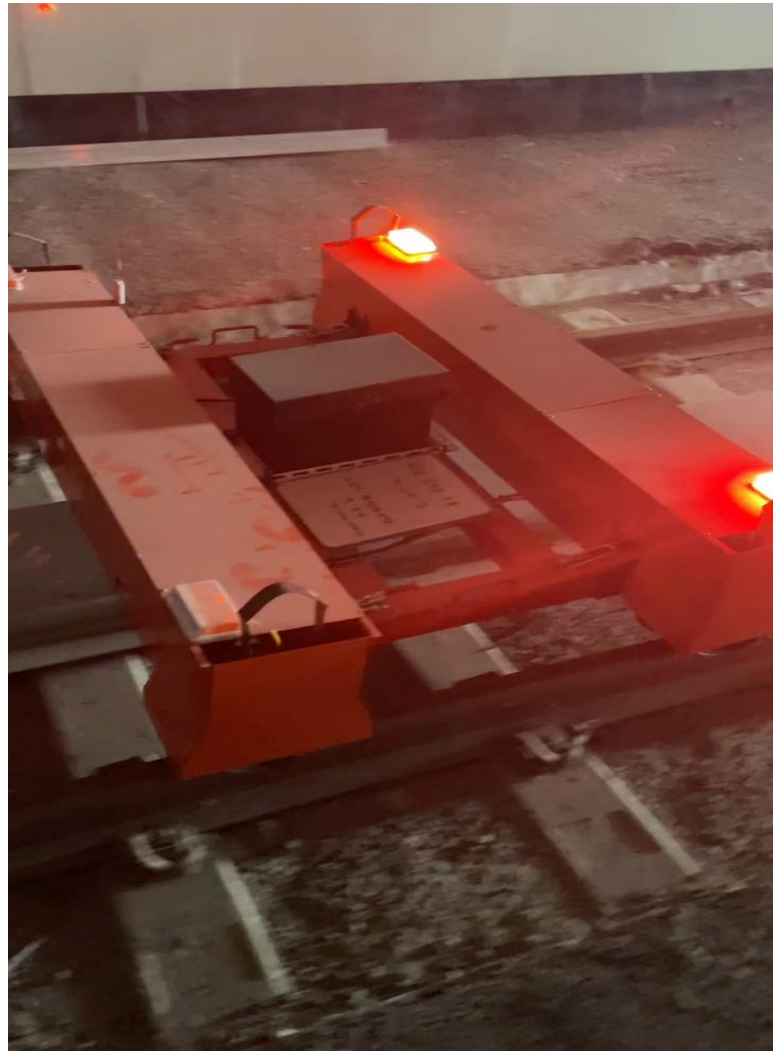
<모터 구동 >

레일 로봇 영상

<모터 구동 >



<사람 추종 >

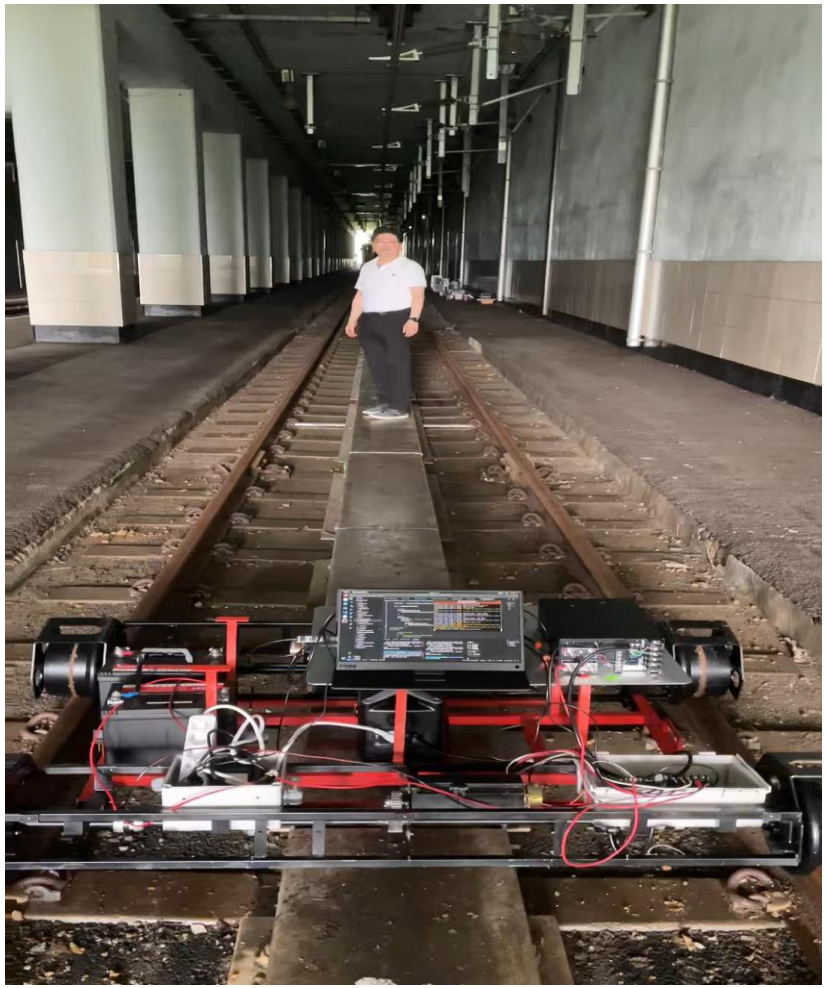


<로봇 주행 >

레일 로봇 영상

<로봇 하드웨어 >

레일 로봇 영상



< 사람 추종 주행 >

< 안전 주행 >

레일로봇 구동 관련 Python Code - Driving

```python
class Driving(Node):
    def __init__(self):
        super().__init__("driving_node")
        qos_profile = QoSProfile(depth=10)

        # motor driver setup
        self.instrument = minimalmodbus.Instrument("/dev/motor_driver", ADDRESS)  # port name, slave address (in decimal)
        self.instrument.serial.baudrate = 115200  # Baud
        self.instrument.serial.bytesize = 8
        self.instrument.serial.parity = serial.PARITY_NONE
        self.instrument.clear_buffers_before_each_transaction = True
        self.instrument.serial.stopbits = 1
        self.instrument.serial.timeout = 0.05 # seconds
        self.instrument.mode = minimalmodbus.MODE_RTU  # rtu or ascii mode

        try:
            self.instrument.serial.write(bytes(COM_MODE))
            time.sleep(0.1)
            self.instrument.serial.write(bytes(SERVO_ON))
            time.sleep(0.1)
            self.instrument.serial.write(bytes(BREAK_OFF))
            time.sleep(0.1)
            self.instrument.serial.write(bytes(RPM_0))
        except Exception as e:
            self.get_logger().error("Fail to initialize : {}".format(e))
            self.destroy_node()
            rclpy.shutdown()

        # subscriber
        self.cmd_vel_sub = self.create_subscription(
            Twist,
            'cmd_vel',
            self.cmd_vel_callback,
            10
        )
        self.input_rpm_sub = self.create_subscription(
            Int16,
            'input_rpm',
            self.input_rpm_callback,
            10
        )

        # publisher
        self.req_rpm_publisher = self.create_publisher(Int32, 'req_rpm', qos_profile)
        self.cur_rpm_publisher = self.create_publisher(Int32, 'cur_rpm', qos_profile)
```

```python
    # info timer
    def info_timer_callback(self):
        try:
            # publish current rpm
            cur_rpm = Int32()
            cur_rpm.data = self.instrument.read_register(registeraddress=0x03, number_of_decimals=0, functioncode=4, signed=False)
            self.cur_rpm_publisher.publish(cur_rpm)

            # publish required rpm
            req_rpm = Int32()
            req_rpm.data = self.instrument.read_register(registeraddress=0x02, number_of_decimals=0, functioncode=4, signed=False)
            self.req_rpm_publisher.publish(req_rpm)
        except Exception as e:
            self.get_logger().error("Fail to read : {}".format(e))

    # robot cmd_vel callback
    def cmd_vel_callback(self, msg):
        try:
            if msg.linear.x > 0.0:
                self.instrument.serial.write(bytes(RPM_P0300))
            elif msg.linear.x < 0.0:
                self.instrument.serial.write(bytes(RPM_N0300))
            else:
                self.instrument.serial.write(bytes(RPM_0))
        except Exception as e:
            self.get_logger().error("Fail to write : {}".format(e))

    def input_rpm_callback(self, msg):
        try:
            if msg.data == 300:
                self.instrument.serial.write(bytes(RPM_P0300))
            elif msg.data == 500:
                self.instrument.serial.write(bytes(RPM_P0500))
            elif msg.data == 700:
                self.instrument.serial.write(bytes(RPM_P0700))
            elif msg.data == 1000:
                self.instrument.serial.write(bytes(RPM_P1000))
            elif msg.data == 0:
                self.instrument.serial.write(bytes(RPM_0))
            elif msg.data == -1000:
                self.instrument.serial.write(bytes(RPM_N1000))
        except Exception as e:
            self.get_logger().error("Fail to write : {}".format(e))

def main(args=None):
    rclpy.init(args=args)
    driving = Driving()
    rclpy.spin(driving)

    # Destroy the node explicitly
    driving.instrument.serial.close()
    driving.destroy_node()
    rclpy.shutdown()
```

레일로봇 구동 관련 Python Code –joystick & keyboard

```python
class JoystickCommand(Node):
    def __init__(self):
        super().__init__("joystick_command_node")
        qos_profile = QoSProfile(depth=10)


        # publisher
        self.joystick_command = Int16()
        self.joystick_command_publisher = self.create_publisher(Int16,'joystick_command', qos_profile)

        # subscriber
        self.joy_sub = self.create_subscription(
            Joy,
            'joy',
            self.joy_callback,
            10
        )

        self.get_logger().info("init Joystick Command node ! ")

    def joy_callback(self, msg):
        """
        Command ID
        0 => A + B
        1 => A + Y
        ...
        """

        if msg.buttons[0] and msg.buttons[1]:
            self.get_logger().info("joystick command 0")
            self.joystick_command.data = 0
            self.joystick_command_publisher.publish(self.joystick_command)

        elif msg.buttons[0] and msg.buttons[3]:
            self.get_logger().info("joystick command 1")
            self.joystick_command.data = 1
            self.joystick_command_publisher.publish(self.joystick_command)

def main(args=None):
    rclpy.init(args=args)
    joystick_command = JoystickCommand()

    try:
        rclpy.spin(joystick_command)
    except KeyboardInterrupt:

        joystick_command.get_logger().info("Keyboard interrupt, shutting down")
        joystick_command.destroy_node()
        rclpy.shutdown()

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

```python
def getKey(settings):
    if sys.platform == 'win32':
        # getwch() returns a string on Windows
        key = msvcrt.getwch()
    else:
        tty.setraw(sys.stdin.fileno())
        # sys.stdin.read() returns a string on Linux
        key = sys.stdin.read(1)
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

class KeyboardCommand(Node):
    def __init__(self):
        super().__init__("keyboard_command_node")
        qos_profile = QoSProfile(depth=10)

        # keyboard intterupt setting
        self.settings = termios.tcgetattr(sys.stdin)

        # keyboard input timer
        key_input_period = 0.1
        self.key_input_timer = self.create_timer(key_input_period, self.key_input_callback)

        # publisher
        self.key_input_rpm_publisher = self.create_publisher(Int16,'input_rpm', qos_profile)

        self.get_logger().info("init KeyboardCommand node ! ")

    # keyboard input timer
    def key_input_callback(self):
        self.get_logger().info(msg)
        key = getKey(self.settings)
        if key in RPMBindings.keys():
            self.key_input_rpm_publisher.publish(Int16(data=RPMBindings[key]))


def main(args=None):
    rclpy.init(args=args)
    keyboard_command = KeyboardCommand()

    try:
        rclpy.spin(keyboard_command)
    except KeyboardInterrupt:

        keyboard_command.get_logger().info("Keyboard interrupt, shutting down")
        keyboard_command.destroy_node()
        rclpy.shutdown()

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

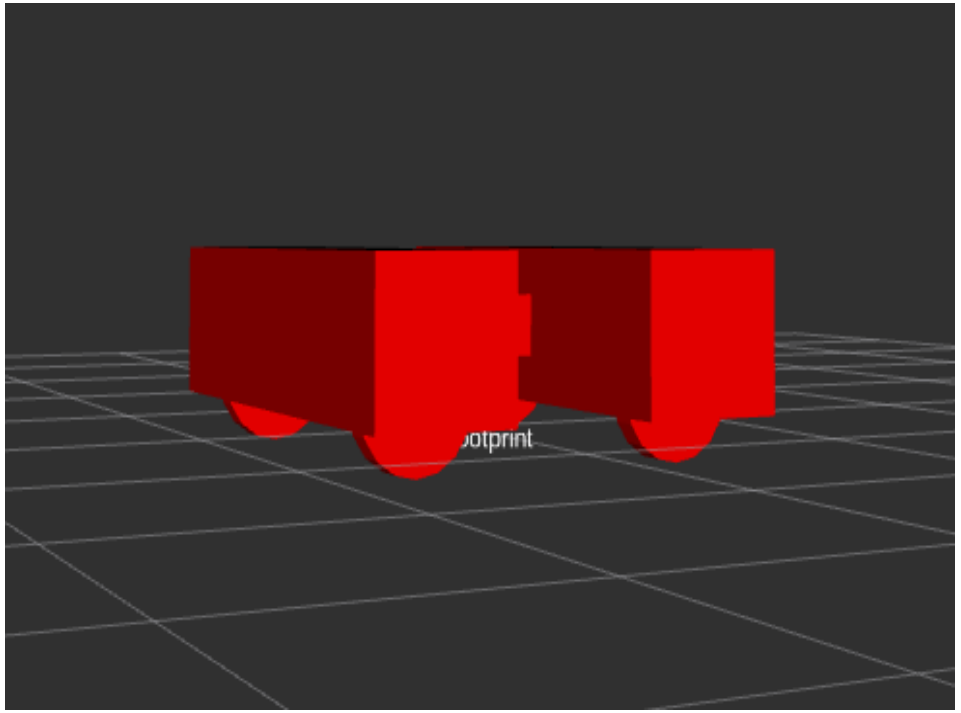
레일로봇 구동 관련 Python Code - Tracker

```python
class Follower(Node):

    def __init__(self):
        super().__init__('follower_node')
        qos_profile = QoSProfile(depth=10)
        self.bridge = CvBridge()

        # Subscriber
        self.bbox_sub = self.create_subscription(
            BoundingBoxes, 'yolov5/bounding_boxes', self.bbox_callback, qos_profile)
        self.depth_sub = self.create_subscription(
            Image, 'camera/aligned_depth_to_color/image_raw', self.depth_callback, qos_profile)
        self.depth_info_sub = self.create_subscription(
            CameraInfo, 'camera/aligned_depth_to_color/camera_info', self.depth_info_callback, qos_profile)

        # Publisher
        self.cmd_vel_pub = self.create_publisher(Twist, 'cmd_vel', qos_profile)
        self.min_depth_pub = self.create_publisher(Float32, 'min_depth', qos_profile)
        self.count_people_pub = self.create_publisher(Int8, 'count_people', qos_profile)

        # Timer
        self.control_timer = self.create_timer(1, self.control_callback)

        # target related
        self.target_box = []
        self.target_class = 'person'
        self.target_depth = []
        self.target_flag = False

        # control related
        self.linear_speed = 0.5
        self.cmd_vel = Twist()

        # depth related
        self.m_fx = 0.0
        self.m_fy = 0.0
        self.depth_array = np.zeros(1)

    def bbox_callback(self, msg):

        for box in msg.bounding_boxes:
            if box.class_id == self.target_class:
                # self.get_logger().info(str(box.class_id) + str(box.id))
                self.target_box.append([box.xmin, box.ymin, box.xmax, box.ymax])
                # self.get_logger().info(str(self.target_box))
                self.target_flag = True
            else:
                self.target_box = []
                self.target_depth = []
                self.target_flag = False

    def depth_info_callback(self, msg):
        # Intrinsic camera matrix for the raw (distorted) images.
        #     [fx  0 cx]
        # K = [ 0 fy cy]
        #     [ 0  0  1]
        if msg.k[0] != 0.0:
            self.m_fx = msg.k[0];
            self.m_fy = msg.k[4];
            self.m_cx = msg.k[2];
```

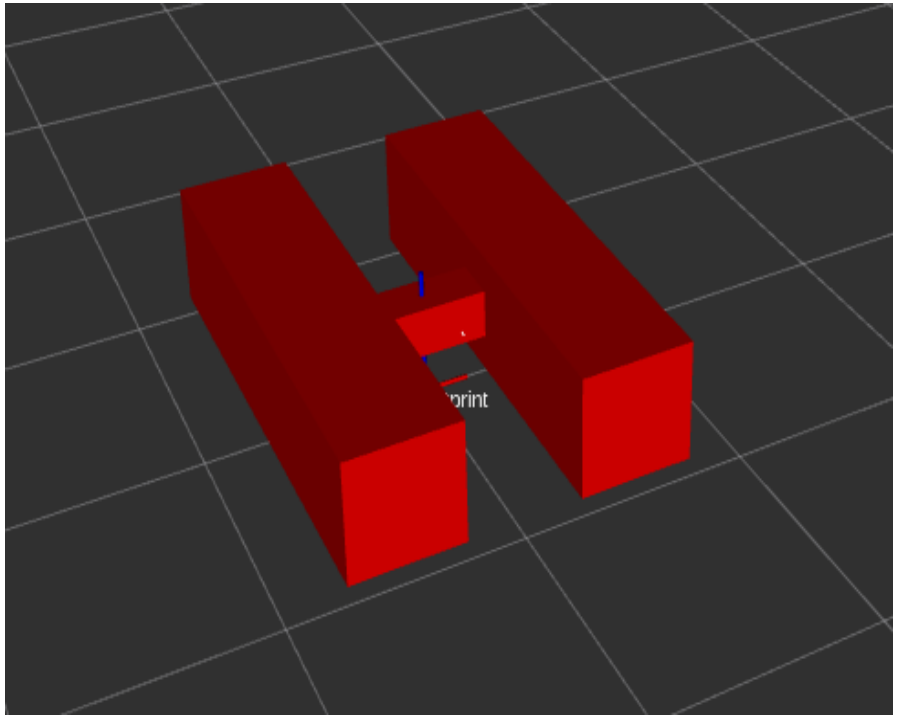
```python
def depth_callback(self, msg):
    self.target_depth = []

    if self.target_flag:
        # ros2 Image msg to numpy array
        depth_image = self.bridge.imgmsg_to_cv2(msg, '32FC1')

        for box in self.target_box:
            x = box[0]
            y = box[1]
            w = box[2] - box[0]
            h = box[3] - box[1]

            # Boxes as large as 30px centered on the center of the box
            x = x + w//2 - 15
            y = y + h//2 - 15
            w = 30
            h = 30

            roi_depth = depth_image[y:y+h, x:x+w]

            inv_fx = 1. / self.m_fx
            inv_fy = 1. / self.m_fy

            n = 0
            sum = 0
            for i in range(0, roi_depth.shape[0]):
                for j in range(0, roi_depth.shape[1]):
                    value = roi_depth.item(i, j)
                    # self.get_logger().info(str(value))
                    if value > 0.0:
                        n = n + 1
                        sum = sum + value
            try:
                point_z = sum / n * 0.001
                point_x = ((x + w/2) - self.m_cx) * point_z * inv_fx
                point_y = ((y + h/2) - self.m_cy) * point_z * inv_fy
                self.depth_array[0] = np.array(math.sqrt(point_x * point_x + point_y * point_y + point_z * point_z))
                mean_depth = float(self.depth_array.mean(axis=0))
                self.target_depth.append(mean_depth)
            except Exception as e:
                self.get_logger().error("Fail to calculate depth : {}".format(e))

    # Publish count_people
    try:
        count_people_msg = Int8()
        count_people_msg.data = len(self.target_depth)
        self.count_people_pub.publish(count_people_msg)
    except Exception as e:
        pass
```

```python
def control_callback(self):
    try:
        min_depth_msg = Float32()
        min_depth_msg.data = min(self.target_depth)
        self.get_logger().info('closest person distance' + str(round(min_depth_msg.data, 3)) + 'm')

        self.min_depth_pub.publish(min_depth_msg)

        # depth range: 0.5m ~ 1.5m
        if 0.5 < min_depth_msg.data < 1.5:
            self.cmd_vel.linear.x = 0.0
            self.cmd_vel.angular.z = 0.0
            self.cmd_vel_pub.publish(self.cmd_vel)
        elif min_depth_msg.data < 0.5:
            self.cmd_vel.linear.x = -self.linear_speed
            self.cmd_vel.angular.z = 0.0
            self.cmd_vel_pub.publish(self.cmd_vel)
        elif min_depth_msg.data > 1.5:
            self.cmd_vel.linear.x = self.linear_speed
            self.cmd_vel.angular.z = 0.0
            self.cmd_vel_pub.publish(self.cmd_vel)

    except Exception as e:
        pass

def main(args=None):
    rclpy.init(args=args)
    follower = Follower()
    rclpy.spin(follower)

    follower.destroy_node()
    rclpy.shutdown()

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

Gazebo 로 구성 진행중인 레일 로봇



<시뮬레이션 모델링 구성 진행 – 옆 >



<시뮬레이션 모델링 구성 진행 – 위 >

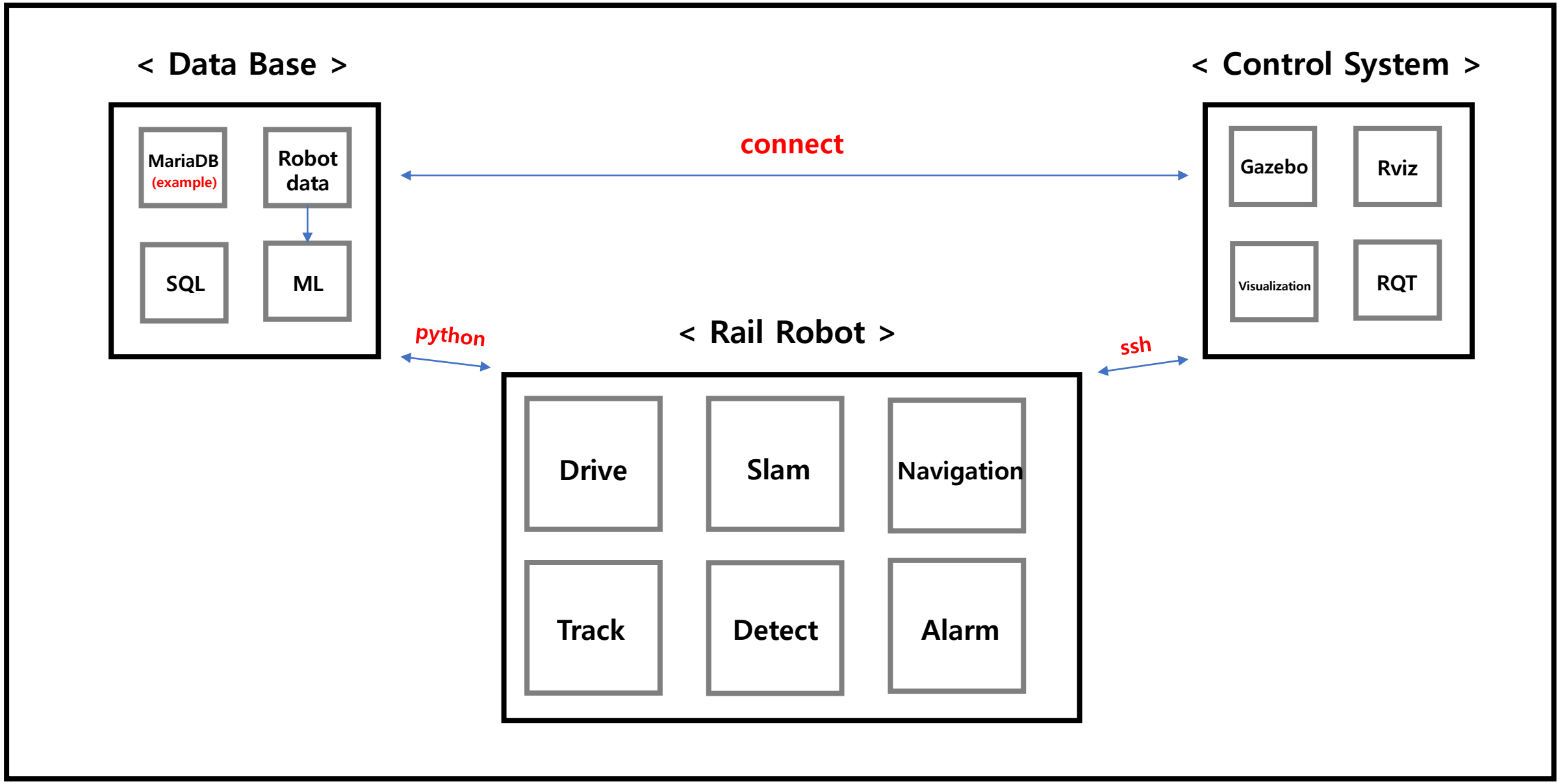
레일로봇 Xacro.URDF Python Code

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
    <xacro:macro name="base" params="mass">
    <xacro:property name="PI" value="3.1415926535897931" />
        <link name="base_footprint"/>

        <link name="base_link" >
            <visual>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <box size="0.5 0.2 0.2"/>
                </geometry>
            </visual>
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <box size="0.5 0.2 0.2"/>
                </geometry>
            </collision>
            <xacro:box_inertial x="0.5" y="2.0" z="0.5" mass="5.0" />
        </link>

        <!-- Front Body -->
        <link name="front_body" >
            <visual>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <box size="0.5 2.0 0.5"/>
                </geometry>
            </visual>
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <box size="0.5 2.0 0.5"/>
                </geometry>
            </collision>
            <xacro:box_inertial x="0.5" y="2.0" z="0.5" mass="${mass}" />
        </link>

        <!-- Rear Body -->
        <link name="rear_body" >
            <visual>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <box size="0.5 2.0 0.5"/>
                </geometry>
            </visual>
            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <box size="0.5 2.0 0.5"/>
                </geometry>
            </collision>
            <xacro:box_inertial x="0.5" y="2.0" z="0.5" mass="${mass}" />
        </link>

        <!-- Joint - base_link ~ front_body -->
        <joint name="base_link_to_front_body" type="fixed">
            <parent link="base_link"/>
            <child link="front_body"/>
            <origin xyz="0.5 0 0" rpy="0 0 0"/>
        </joint>

        <!-- Joint - base_link ~ rear_body -->
        <joint name="base_link_to_rear_body" type="fixed">
            <parent link="base_link"/>
            <child link="rear_body"/>
            <origin xyz="-0.5 0 0.0" rpy="0 0 0"/>
        </joint>

        <!-- Joint - base_link ~ base_footprint -->
        <joint name="base_link_to_footprint" type="fixed">
            <parent link="base_footprint"/>
            <child link="base_link"/>
            <origin xyz="0 0 0.3285" rpy="0 0 0"/>
        </joint>

    </xacro:macro>
</robot>
```

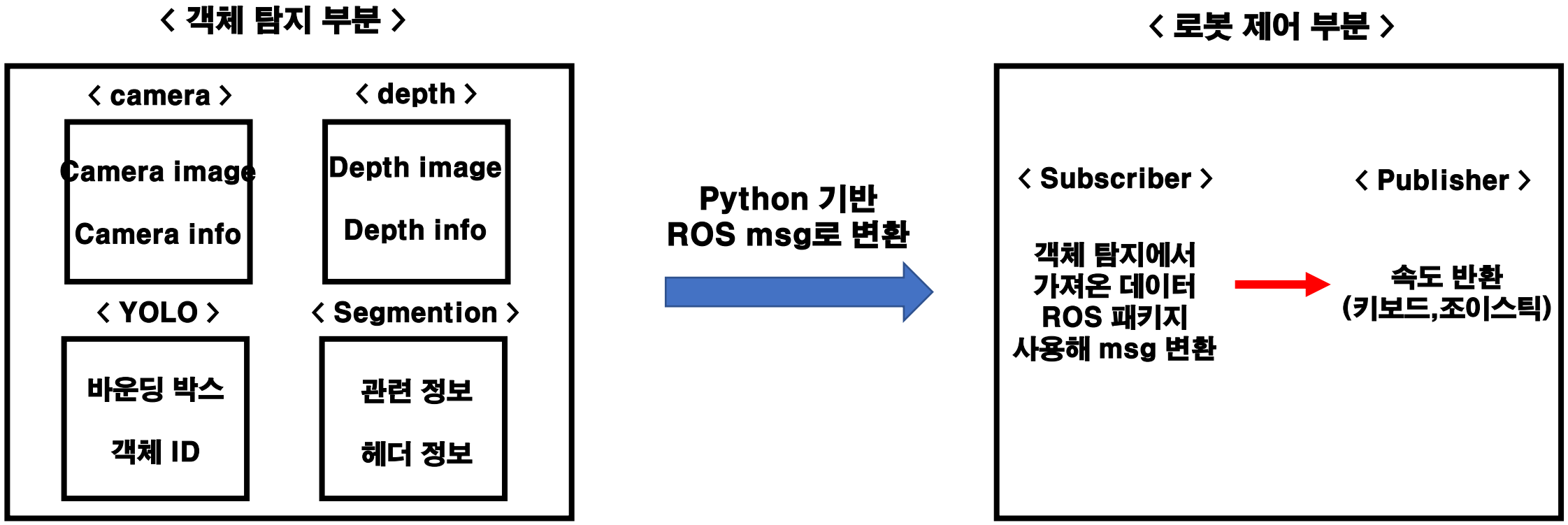
```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">

    <!-- Constants -->
    <xacro:property name="PI" value="3.1415926535897931" />
    <xacro:property name="wheel_length" value="0.1" />
    <xacro:property name="wheel_radius" value="0.2" />
    <xacro:property name="wheel_mass" value="5" />

    <!-- Set up of the Wheel -->
    <xacro:macro name="wheel" params="mass">
        <!-- Wheel1 Link -->
        <link name="wheel1">
            <visual>
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </visual>

            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </collision>

            <xacro:cylinder_inertial radius="${wheel_radius}" length="${wheel_length}" mass="${wheel_mass}" />
        </link>

        <gazebo reference="wheel1">
            <mu1>0.01</mu1>
            <mu2>0.01</mu2>
        </gazebo>

        <!-- Joint - front_body ~ wheel1 -->
        <joint name="wheel1_joint" type="fixed">
            <axis xyz="0 0 1" />
            <parent link="front_body"/>
            <child link="wheel1"/>
            <origin xyz="0.0 0.7 -0.2" rpy="${PI/2} 0 -${PI}" />
        </joint>

        <!-- Wheel2 Link -->
        <link name="wheel2">
            <visual>
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </visual>

            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </collision>

            <xacro:cylinder_inertial radius="${wheel_radius}" length="${wheel_length}" mass="${wheel_mass}" />
        </link>

        <gazebo reference="wheel2">
            <mu1>0.01</mu1>
            <mu2>0.01</mu2>
        </gazebo>
```

```
        <!-- Joint - front_body ~ wheel2 -->
        <joint name="wheel2_joint" type="fixed">
            <axis xyz="0 0 1" />
            <parent link="front_body"/>
            <child link="wheel2"/>
            <origin xyz="0.0 -0.7 -0.2" rpy="${PI/2} 0 -${PI}" />
        </joint>

        <link name="wheel3">
            <visual>
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </visual>

            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </collision>

            <xacro:cylinder_inertial radius="${wheel_radius}" length="${wheel_length}" mass="${wheel_mass}" />
        </link>

        <gazebo reference="wheel3">
            <mu1>0.01</mu1>
            <mu2>0.01</mu2>
        </gazebo>

        <!-- Joint - rear_body ~ wheel3 -->
        <joint name="wheel3_joint" type="fixed">
            <axis xyz="0 0 1" />
            <parent link="rear_body"/>
            <child link="wheel3"/>
            <origin xyz="0.0 0.7 -0.2" rpy="${PI/2} 0 -${PI}" />
        </joint>

        <!-- Wheel4 Link -->
        <link name="wheel4">
            <visual>
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </visual>

            <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                    <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
                </geometry>
            </collision>

            <xacro:cylinder_inertial radius="${wheel_radius}" length="${wheel_length}" mass="${wheel_mass}" />
        </link>

        <gazebo reference="wheel4">
            <mu1>0.01</mu1>
            <mu2>0.01</mu2>
        </gazebo>

        <!-- Joint - rear_body ~ wheel1 -->
        <joint name="wheel4_joint" type="fixed">
            <axis xyz="0 0 1" />
            <parent link="rear_body"/>
            <child link="wheel4"/>
            <origin xyz="0.0 -0.7 -0.2" rpy="${PI/2} 0 -${PI}" />
        </joint>


    </xacro:macro>
</robot>
```

< Rail Robot Architecture >



궤도 이송 로봇 인터페이스

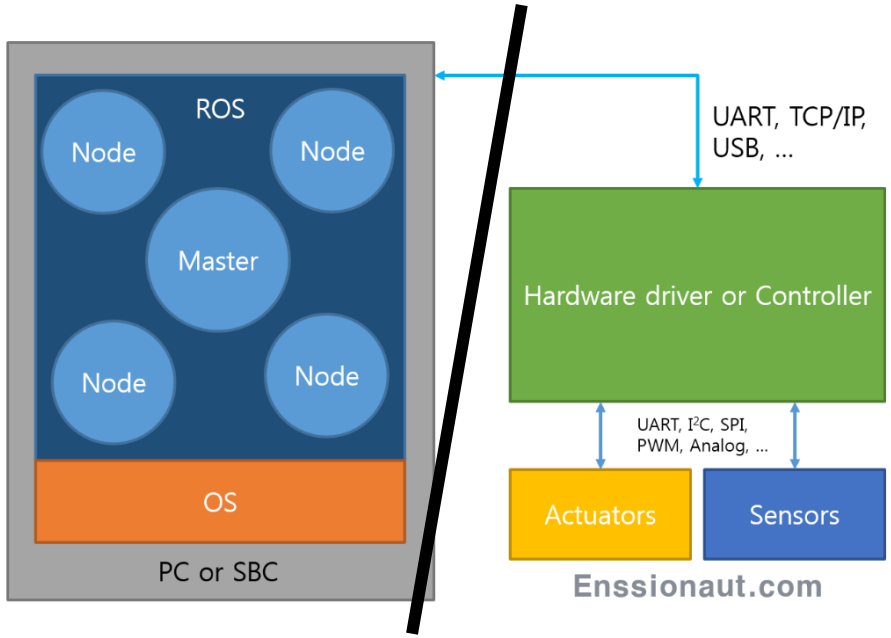


➢ 로봇 제어 부분 관점 기준

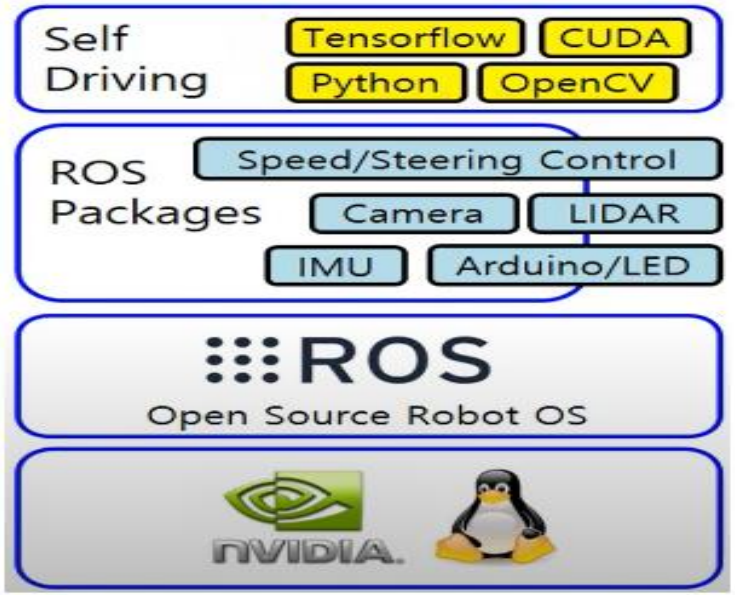
- ✓ Python 기반의 ROS 를 활용해서 이송 로봇 개발 진행 -> ROS의 기본 개념인 Subscriber , Publisher 개념 활용
- ✓ Subscriber에서 객체 탐지 및 다른 분야에서 데이터를 받아 올 때 ROS에서 지원하는 msg 개념으로 받아오면 활용 가능
- ✓ Msg 구조체 확인 및 관련 패키지 다운 (로봇 제어 부분) -> 없을 시 직접 정의 (msg 만들기 -> 해본적 있음, 가능)
  ex) Segmention은 SegmentationMsg 패키지 사용

ROS

< Robot Operation System >



< ROS 적용 시스템 동작 구조>

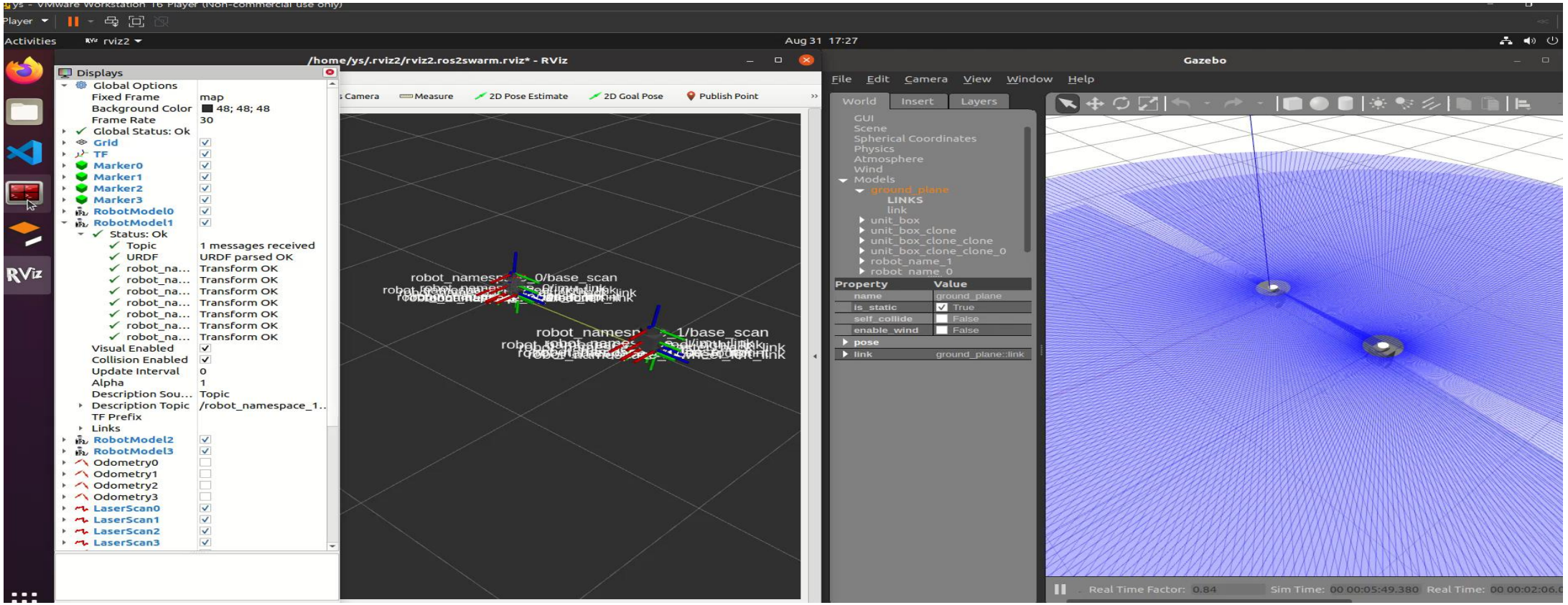


< 자율주행 로봇 소프트웨어 구조>

- ✓ 로봇 응용 프로그램을 구축하는 데 도움이 되는 소프트웨어 라이브러리
- ✓ 하드웨어 플랫폼과 소프트웨어 플랫폼의 연동 (하드웨어에 대한 지식이 부족해도 응용 가능)
- ✓ ROS는 기존의 운영체제 + 로봇 응용 프로그램에 필요한 라이브러리 사용

출처 : https://enssionaut.com/board_robotics/421 , https://20201215.tistory.com/19

Robot – 물류 이송 로봇



✓ **Multi Swarm Robot 운영하기 위한 패키지 개발 중**

✓ **다양한 형태의 주행 가능**

출처 : https://github.com/yeonsoo98/ros2swarm

Robot – 물류 이송 로봇

< 현재 진행 사항 >

- ✓ 물류 창고에서 사용되는 로봇들이 무거운 물품을 더 쉽게 들도록 로봇이 군집을 이뤄 이동하는 것을 목표
- ✓ ROS 2로 구성된 군집 이동 패키지 작성 완료 – Turtlebot3로 주행 완료
- ✓ Wego ST-MINI 로 군집 주행 진행 중

< 향후 계획 >

- ✓ Wego ST-MINI 로 군집 주행 완료
- ✓ 물류 창고를 간단하게 구현해 Robot을 투입
- ✓ 무거운 물품을 들려고 할 때 군집을 이뤄 주행하는 알고리즘 작성

ROS – DB

< MariaDB>

```python
class DBSubscriber(Node):

    def __init__(self):
        # db connection
        self.db_connection()

        # ros init
        super().__init__('db_subscriber')
        self.cmd_vel_subscription = self.create_subscription(
            Twist,
            'robot_namespace_0/cmd_vel',
            self.cmd_vel_callback,
            10)
        self.cmd_vel_subscription  # prevent unused variable warning

        self.front_laser_subscription = self.create_subscription(
            LaserScan,
            'robot_namespace_0/scan',
            self.scan_callback,
            10)
        self.front_laser_subscription

    def db_connection(self):
        """
        @brief  DB연결관련 함수
        """
        self.con = pymysql.connect(host=HOST, user=USER, password=PASSWORD, db=DB, charset='utf8')
        self.cur = self.con.cursor()

    def cmd_vel_callback(self, msg):
        """
        @brief  속도 토픽 콜백
        """
        self.get_logger().info('linear x: "%s"' % msg.linear.x)
        self.get_logger().info('linear y: "%s"' % msg.linear.y)
        self.get_logger().info('angular x: "%s"' % msg.angular.z)

    def scan_callback(self, msg):
        if msg.ranges[180] < 1.0 :
            self.get_logger().warn('front :"%f"' % msg.ranges[180])
            sql = "INSERT INTO robot (robot_namespace_id, speed) VALUES (%s, %s)"
            with self.con:
                with self.con.cursor() as self.cur:
                    self.cur.execute(sql, ('5', '7.1'))
                    self.con.commit()
```

< Python Code >

- ✓ 오픈 소스의 관계형 데이터 관리 시스템인 MariaDB 와 ROS 연동 (MySQL 구조 동일)
- ✓ ROS 에서 나오는 데이터를 DB와 연동 -> 관제 시스템 , 머신러닝 학습 데이터 활용
- ✓ 로봇의 속도 , 로봇의 위치, 위험물 위험 경고 등등 활용 가능

출처 : https://ko.wikipedia.org/wiki/MariaDB , https://github.com/yeonsoo98/db_ws

ROS – DB

< 현재 진행 사항 >

- ✓ ROS에서 나오는 데이터들이 MariaDB로 들어가는 시스템 구축 완료
- ✓ 로봇의 속도, 로봇의 위치, 위험물 위험 경고 등등 활용 가능
- ✓ 시뮬레이션에서 이동하는 로봇들의 데이터 또한 연동 가능

< 향후 계획 >

- ✓ 모인 데이터를 향후 머신러닝에 활용 가능
- ✓ 관제 시스템을 구축 시 데이터를 활용해 시각화 ex) Tableau
- ✓ 앞선 두 로봇 뿐만 아니라 앞으로 다양한 로봇들의 데이터 또한 연동 가능