DOOSAN ROKEY
BOOTCAMP

스마트 공장 자동화 시스템 프로토타입 개발

with ArUco + YOLOv8

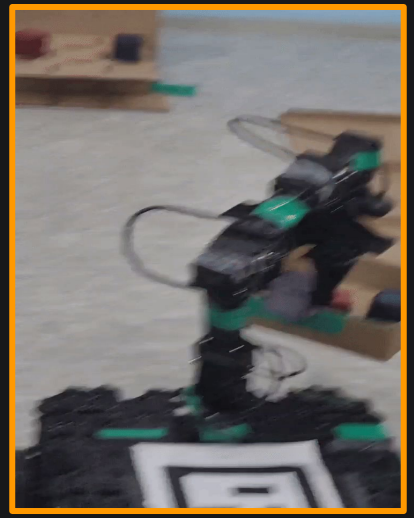
TEAM
A - 5
남기중 신동준 김정한 장효진

List of Content

Introduction

ArUco marker :
로봇의 실시간 global 위치 추정,
목표 지점의 위치값 추정, 기준점을
통한 상대 위치

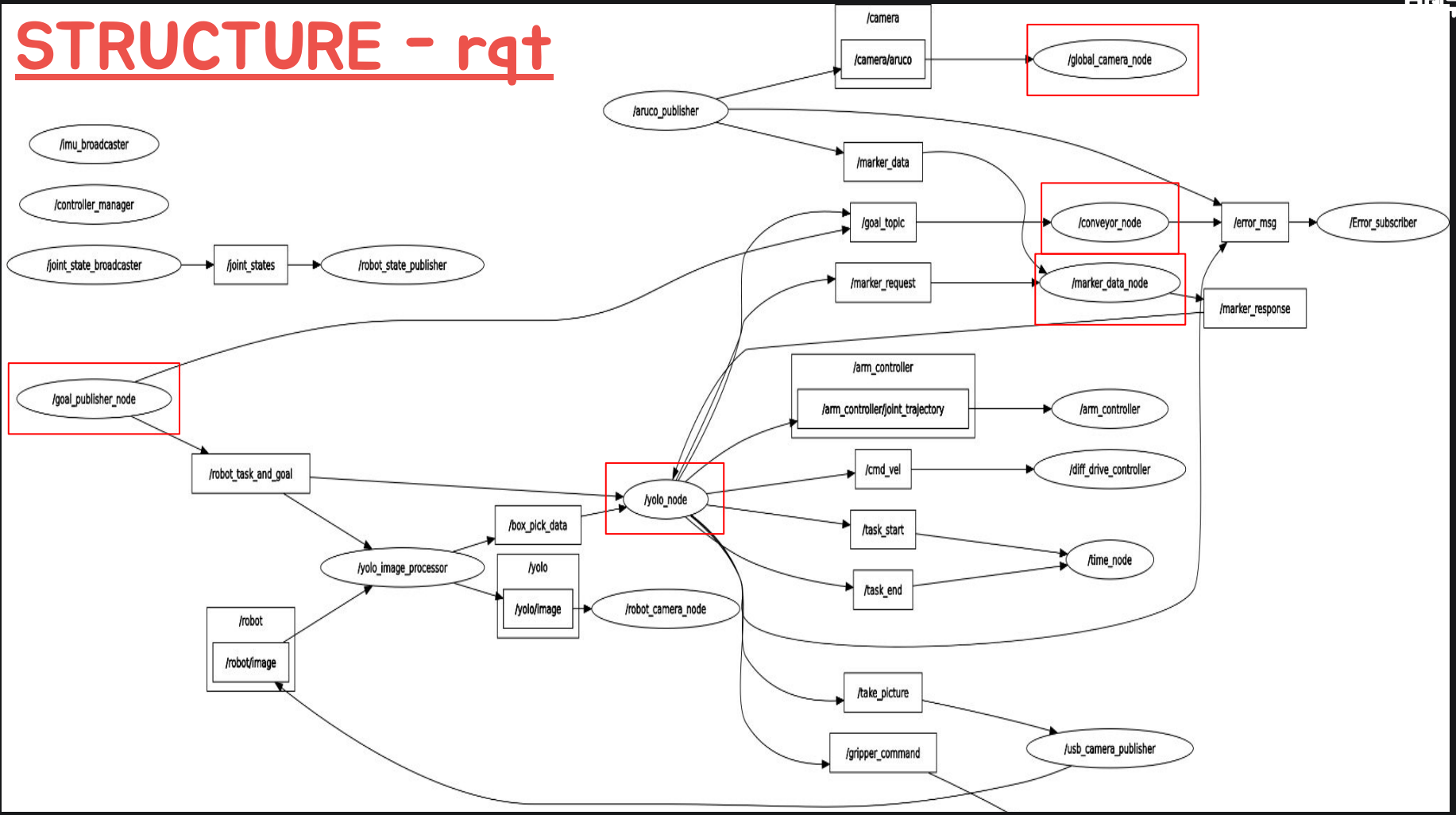
Open_manipulator +
Turtlebot(Waffle):
mobile manipulator , grip
boxes & carrier



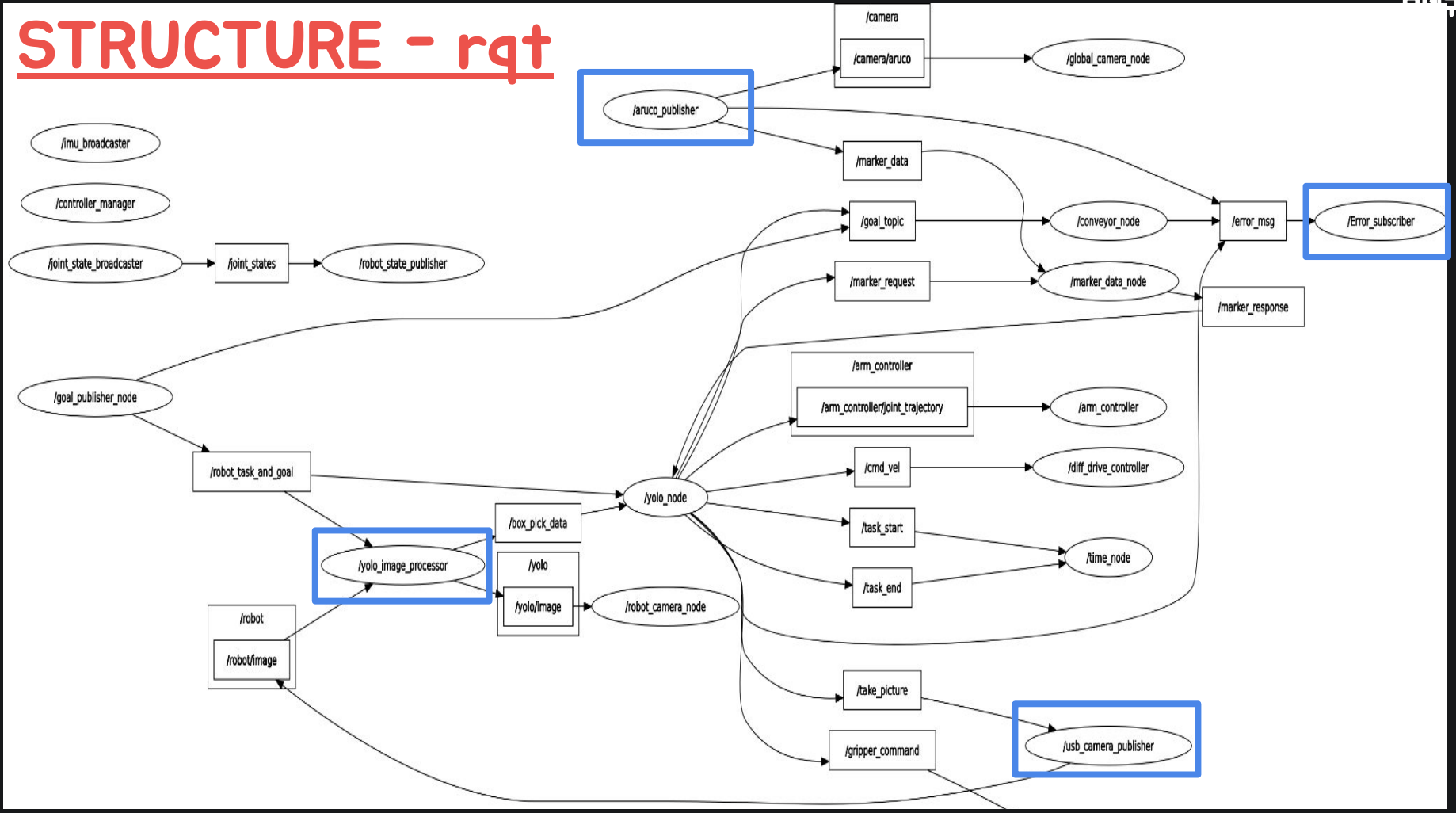
YOLO v8 :
3 종류의 박스 vision기반 추정

Conveyor_belt :
move boxes to carrier,
1.8deg , 3:1 ratio, controlled
by driver + Arduino UNO

STRUCTURE - rqt



STRUCTURE - rqt



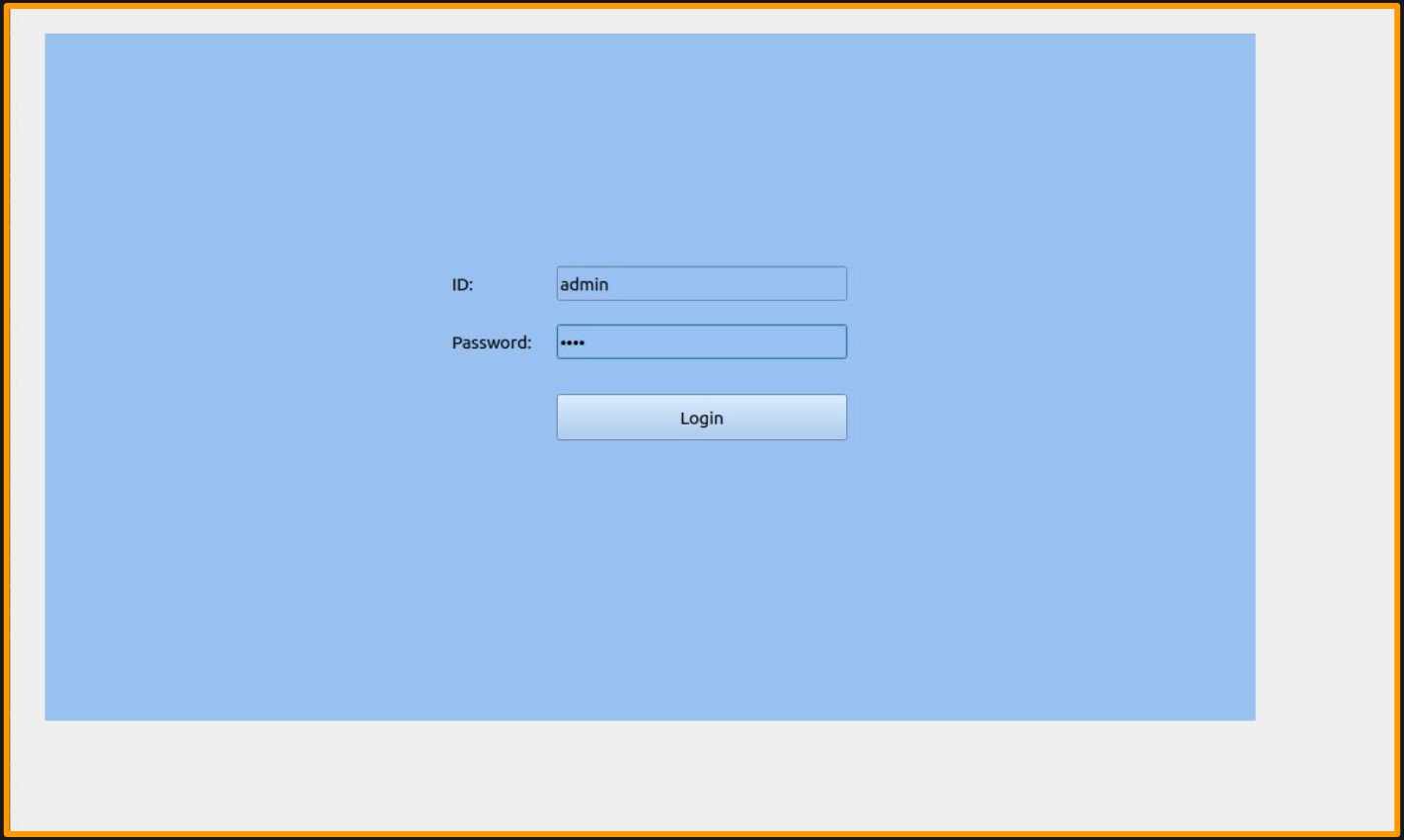
STRUCTURE - train_data

```python
class RobotTrainData(Node):
    def __init__(self):
        super().__init__('train_data')
        self.joint_pub = self.create_publisher(JointTrajectory, '/arm_controller/joint_trajectory', QoSProfile(depth=30))
        self.take_picture_pub = self.create_publisher(Bool, 'take_picture', QoSProfile(depth=10))
        self.image_subscription = self.create_subscription(
            CompressedImage,
            'robot/image',
            self.image_callback,
            QoSProfile(depth=10)
        )
        self.image_received = False
        self.bridge = CvBridge()
        self.image_save_dir = '/home/namsang/turtlebot3_ws/src/test_code/test_code/image5'
        os.makedirs(self.image_save_dir, exist_ok=True)
        self.trajectory_msg = JointTrajectory()
        self.trajectory_msg.header = Header()
        self.trajectory_msg.header.frame_id = ''
        self.trajectory_msg.joint_names = ['joint1', 'joint2', 'joint3', 'joint4']

    def image_callback(self, msg):
        self.image_received = True
        try:
            np_arr = np.frombuffer(msg.data, np.uint8)
            cv_image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
            timestamp = time.strftime("%Y%m%d-%H%M%S")
            file_path = os.path.join(self.image_save_dir, f'image_{timestamp}.jpg')
            cv2.imwrite(file_path, cv_image)
            self.get_logger().info(f"Image saved: {file_path}")
        except Exception as e:
            self.get_logger().error(f"Failed to process compressed image: {e}")

    def publish_take_picture(self):
        self.take_picture_pub.publish(Bool(data=True))
        self.get_logger().info("Published take_picture message.")
        self.wait_for_image()
```

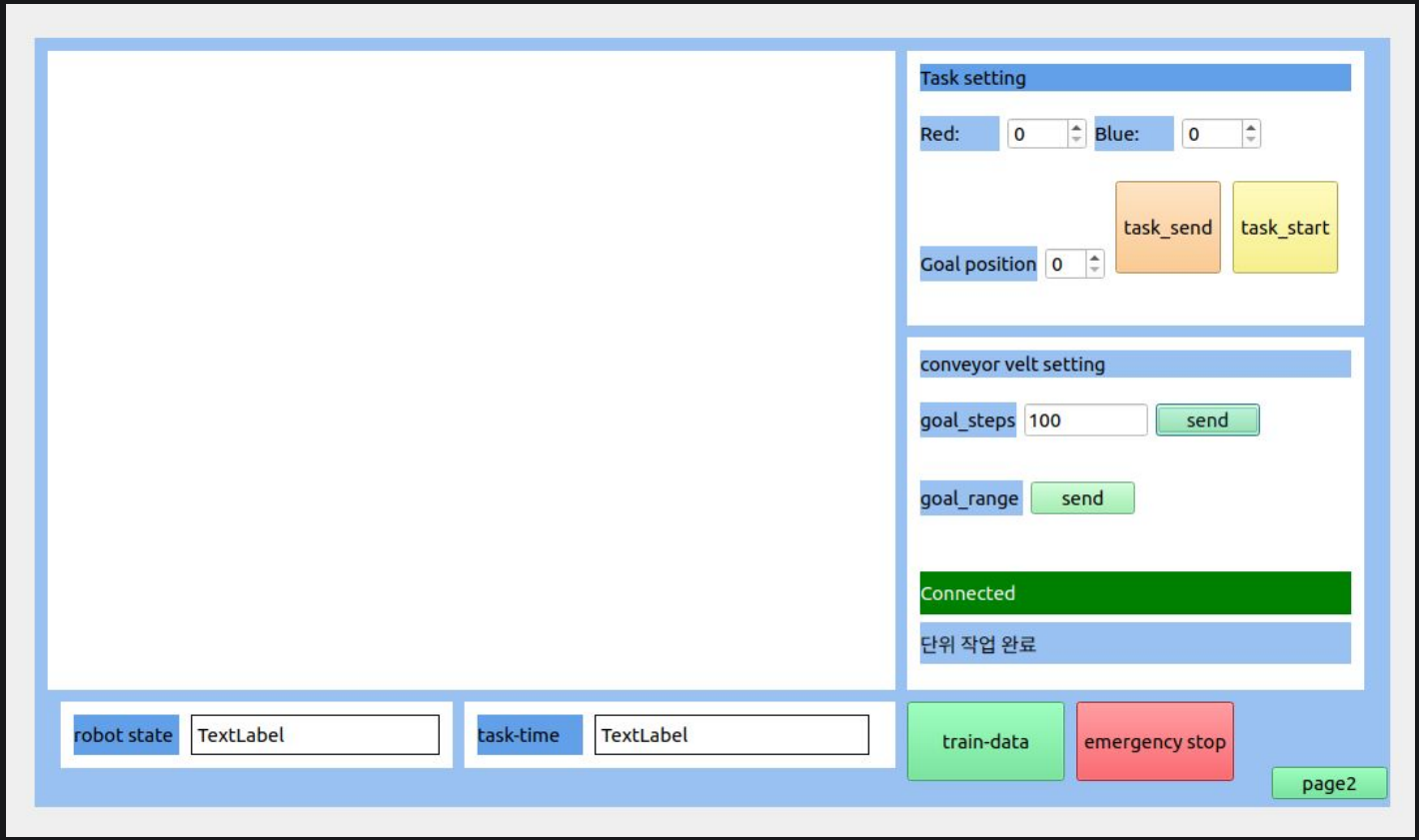
STRUCTURE - train_data

```python
def main(args=None):
    rclpy.init(args=args)
    node = RobotTrainData()

    # Register signal handler for Ctrl+C
    signal.signal(signal.SIGINT, lambda sig, frame: handle_exit(sig, frame, node))

    x_start = 50
    z = 100
    num_cycles = 10

    try:
        for cycle in range(num_cycles):
            x = x_start + cycle * 10
            for step in range(-5, 6):
                y = step * 10
                node.move_arm_to_position(x, y, z)

    except Exception as e:
        node.get_logger().error(f"Error occurred: {e}")
    finally:
        node.disable_arm()
        node.destroy_node()
        rclpy.shutdown()

if __name__ == "__main__":
    main()
[EOF]
```

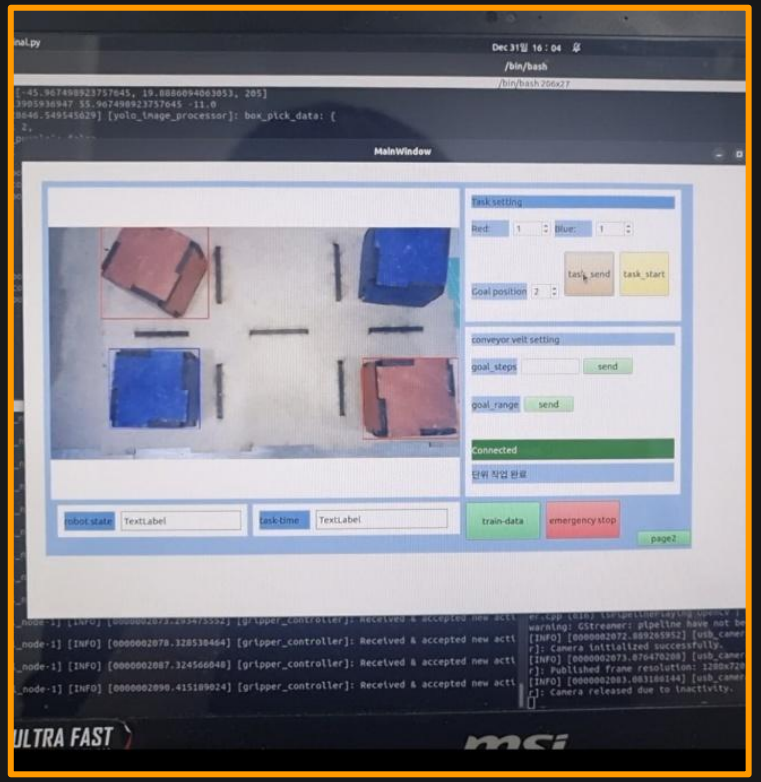
STRUCTURE - GUI

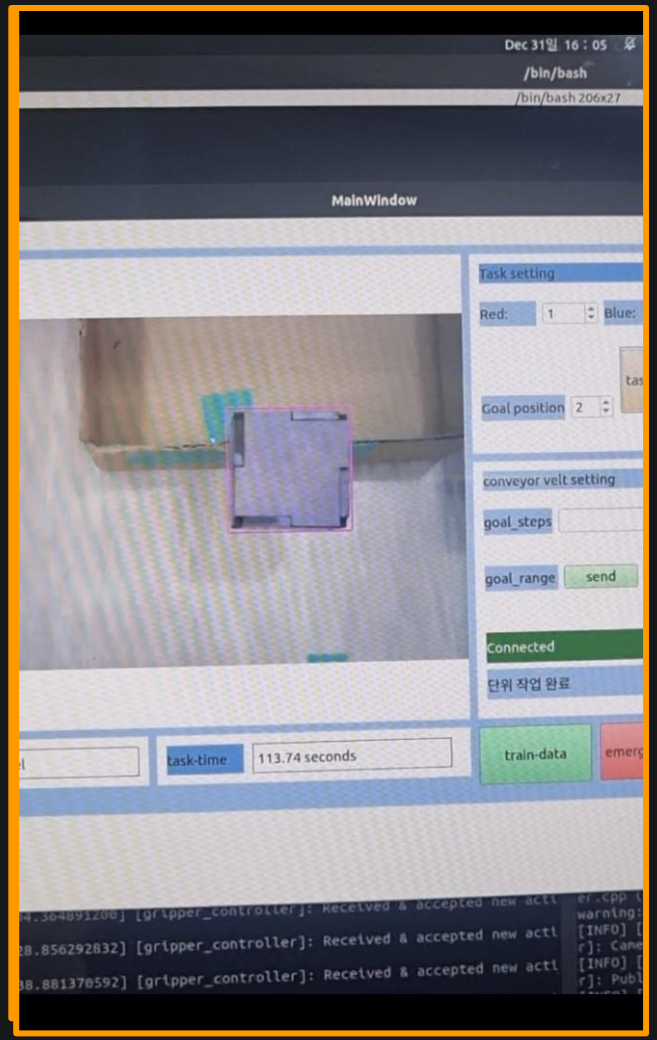


STRUCTURE - GUI

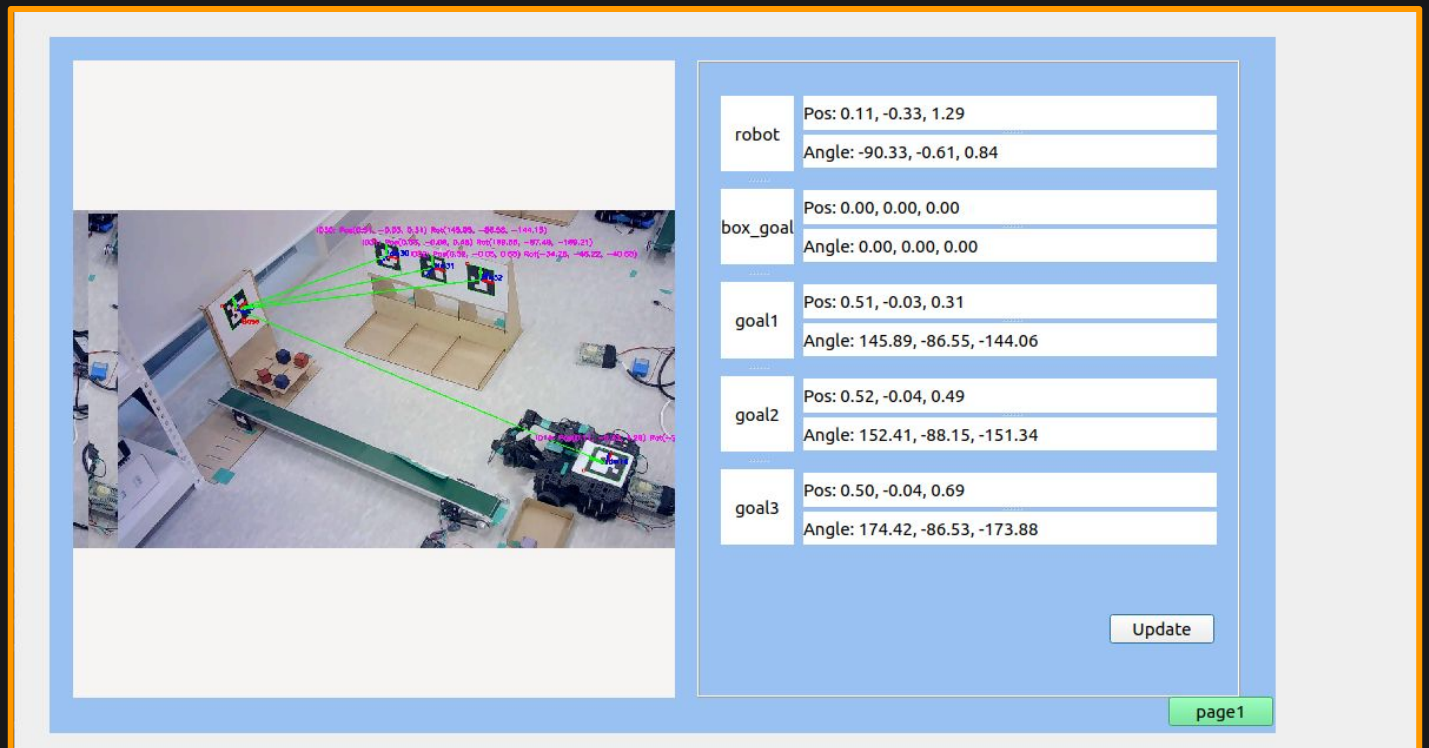


STRUCTURE - GUI





STRUCTURE - GUI

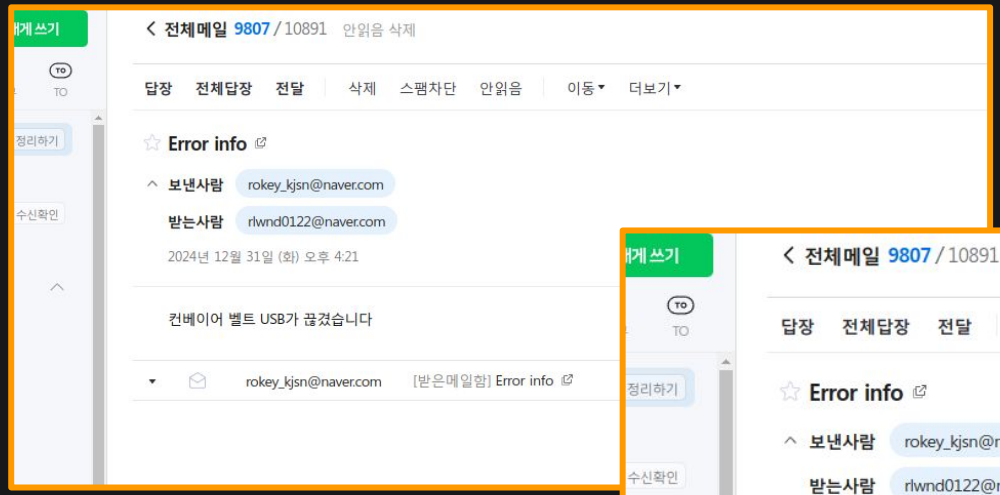


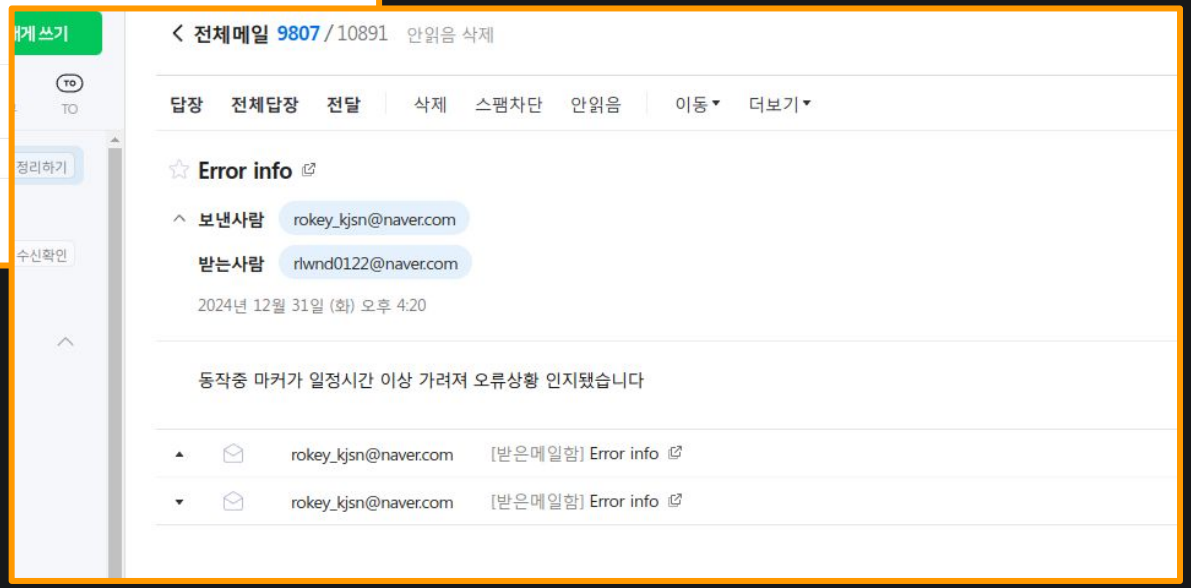
STRUCTURE - Error

```python
#1. 이메일 보내기 기본 예제

import smtplib
from email.mime.text import MIMEText
def send_erroremail(text):
    sendEmail = "rokey_kjsn@naver.com"
    recvEmail = "rlwnd0122@naver.com"
    password = "rokey0R$%"

    smtpName = "smtp.naver.com" #smtp 서버 주소
    smtpPort = 587 #smtp 포트 번호

    # text = "메일 내용"
    msg = MIMEText(text) #MIMEText(text , _charset = "utf8")

    msg['Subject'] ="Error info"
    msg['From'] = sendEmail
    msg['To'] = recvEmail
    print(msg.as_string())

    s=smtplib.SMTP( smtpName , smtpPort ) #메일 서버 연결
    s.starttls() #TLS 보안 처리
    s.login( sendEmail , password ) #로그인
    s.sendmail( sendEmail, recvEmail, msg.as_string() ) #메일 전송, 등
    s.close() #smtp 서버 연결을 종료합니다.

```

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class ErrorSubscriber(Node):
    def __init__(self):
        super().__init__('Error_subscriber')  # Node name
        self.subscription = self.create_subscription(
            String,            # Message type
            'error_msg',        # Topic name
            self.listener_callback,  # Callback function
            10                 # QoS depth
        )
        self.subscription  # Prevent unused variable warning

    def listener_callback(self, msg):
        if msg.data == "conveyor belt error":
            text = "컨베이어 벨트 USB가 끊겼습니다"
            send_erroremail(text)
        elif msg.data == "invalid number of boxes":
            text = "테이블에 박스 개수가 부족합니다"
            send_erroremail(text)
        elif msg.data == "Timeout in reaching target location":
            text = "일정시간 이내에 타켓 위치에 못갔습니다"
            send_erroremail(text)
        elif msg.data == "marker occlusion timeout":
            text = "동작중 마커가 일정시간 이상 가려져 오류상황 인지됐습니다"
            send_erroremail(text)

def main(args=None):
    rclpy.init(args=args)  # Initialize ROS 2 Python client library
    action_subscriber = ErrorSubscriber()

    try:
        rclpy.spin(action_subscriber)  # Keep the node alive
    except KeyboardInterrupt:
        pass
    finally:
        action_subscriber.destroy_node()  # Cleanup the node
        rclpy.shutdown()  # Shutdown ROS 2

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

STRUCTURE - Error





STRUCTURE - Conveyor


```python
# ConveyorNode class
class ConveyorNode(Node):
    def __init__(self, arduino_port, baud_rate):
        super().__init__('conveyor_node')
        self.subscription = self.create_subscription(Int32, 'goal_topic', self.listener_callback, 10)
        self.error_publisher = self.create_publisher(String, 'error_msg', 10)
        self.arduino_port = arduino_port
        self.baud_rate = baud_rate
        self.connection_status_callback = None
        self.arduino = None
        self.connect_attempts = 0
        self.last_connected_status = False  # Track previous connection status
        self.conveyor_running_callback = None  # Initialize conveyor_running_callback

        # Attempt initial connection
        self.connect_arduino()

        # Set timer to check connection status every 0.5 seconds
        self.timer = self.create_timer(1.0, self.check_connection_status)
        self.timer = self.create_timer(1.0, sel
```

```python
class ConveyorWorker(QThread):
    connection_status_signal = pyqtSignal(bool)  # GUI 업데이트를 위한 연결 상태 신호
    conveyor_running_signal = pyqtSignal(str)  # Signal for Arduino data


    def __init__(self, arduino_port="/dev/ttyACM0", baud_rate=115200):
        super().__init__()
        self.executor = rclpy.executors.SingleThreadedExecutor()
        self.conveyor_node = ConveyorNode(arduino_port, baud_rate)
        self.executor.add_node(self.conveyor_node)

        # 연결 상태 변경 콜백 등록
        self.conveyor_node.connection_status_callback = self.update_connection_status
        self.conveyor_node.conveyor_running_callback = self.conveyor_running_signal.emit

    def update_connection_status(self, is_connected):
        """ConveyorNode에서 받은 연결 상태 변경 신호를 GUI로 전달."""
        self.connection_status_signal.emit(is_connected)

    def run(self):
        try:
            self.executor.spin()
        except Exception as e:
            print(f"Executor error: {e}")
        finally:
            self.executor.shutdown()

    def terminate_execution(self):
        """작업 중지 및 노드 종료."""
        self.conveyor_node.destroy_node()
        self.executor.shutdown()
        self.quit()
        self.wait()
```

```python
    def check_connection_status(self):
        """Periodically check the connection status and emit changes."""
        is_currently_connected = self.is_connected()

        # Emit signal if connection status changes
        if is_currently_connected != self.last_connected_status:
            self.last_connected_status = is_currently_connected
            if self.connection_status_callback:
                self.connection_status_callback(is_currently_connected)

        # Attempt reconnection if disconnected
        if not is_currently_connected:
            self.connect_arduino()
```

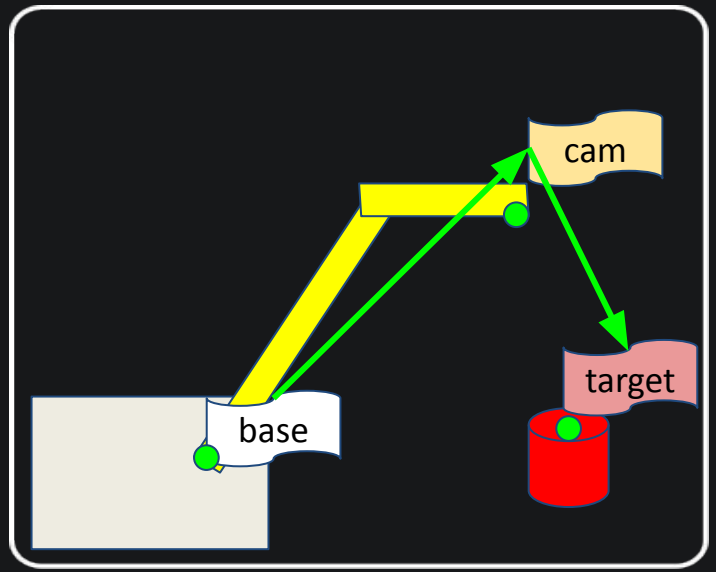
STRUCTURE - Conveyor


```python
class ConveyorWorker(QThread):
    connection_status_signal = pyqtSignal(bool)  # GUI 업데이트를 위한 연결 상태 신호
    conveyor_running_signal = pyqtSignal(str)  # Signal for Arduino data


    def __init__(self, arduino_port="/dev/ttyACM0", baud_rate=115200):
        super().__init__()
        self.executor = rclpy.executors.SingleThreadedExecutor()
        self.conveyor_node = ConveyorNode(arduino_port, baud_rate)
        self.executor.add_node(self.conveyor_node)

        # 연결 상태 변경 콜백 등록
        self.conveyor_node.connection_status_callback = self.update_connection_status
        self.conveyor_node.conveyor_running_callback = self.conveyor_running_signal.emit

    def update_connection_status(self, is_connected):
        """ConveyorNode에서 받은 연결 상태 변경 신호를 GUI로 전달."""
        self.connection_status_signal.emit(is_connected)

    def run(self):
        try:
            self.executor.spin()
        except Exception as e:
            print(f"Executor error: {e}")
        finally:
            self.executor.shutdown()

    def terminate_execution(self):
        """작업 중지 및 노드 종료."""
        self.conveyor_node.destroy_node()
        self.executor.shutdown()
        self.quit()
        self.wait()
```

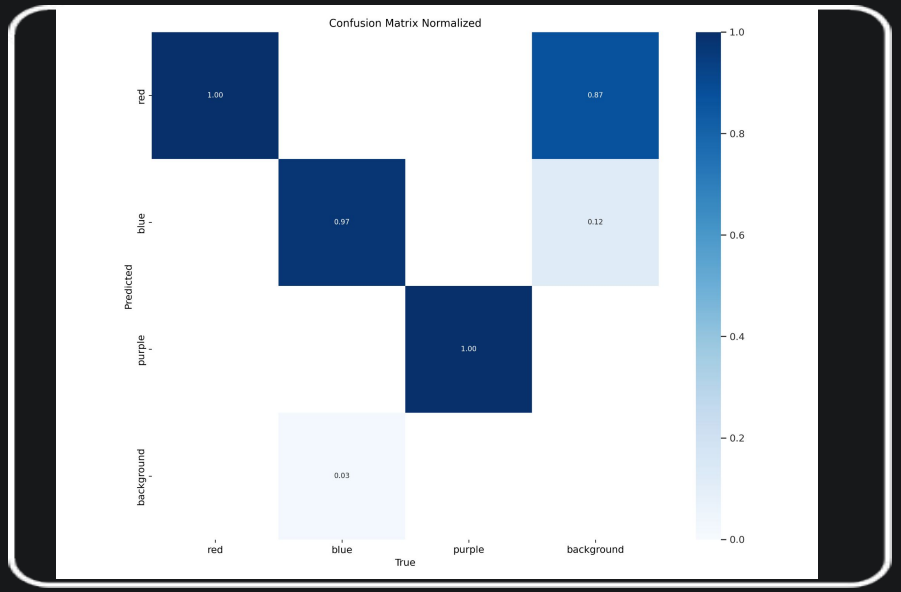
```python
self.conveyor_worker = ConveyorWorker()
self.conveyor_worker.connection_status_signal.connect(self.update_conveyor_status)
self.conveyor_worker.conveyor_running_signal.connect(self.update_conveyor_running_state)
```

```python
def init_ui(self):
    self.ui.login_button.clicked.connect(self.verify_login)
    self.ui.btn_goal_step.clicked.connect(self.send_step_to_conveyor)
    self.ui.btn_goal_range.clicked.connect(self.send_range_to_conveyor)
    self.ui.update_button.clicked.connect(self.enable_other_marker_update)  # 추가
    self.ui.train_data_btn.clicked.connect(self.toggle_train_process)
    self.ui.btn_task_send.clicked.connect(self.send_task_to_goal)
    self.ui.btn_task_start.clicked.connect(self.start_yolo_task)  # YOLO 작업 시작
    self.ui.emegency_stop_btn.clicked.connect(self.handle_emergency_stop)
```

Target_position estimation

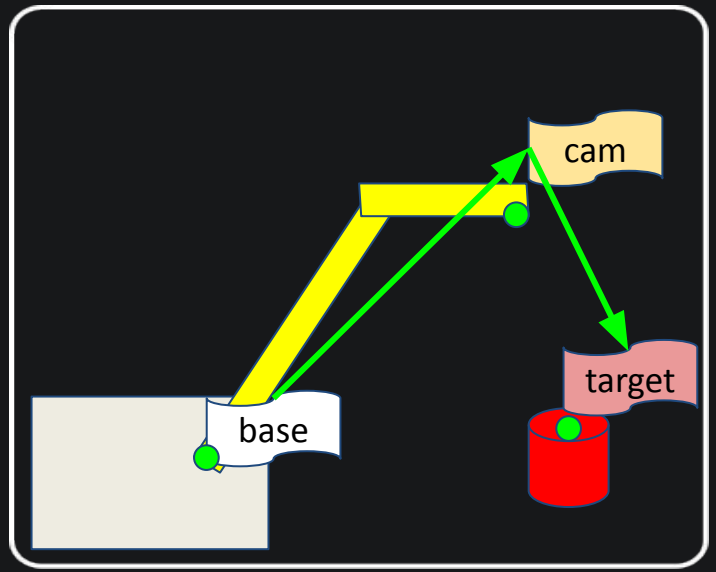


System_sketch



confusion matrix

Yolo recognition



Base -> TCP , 카메라 촬영시 위치 (manipulator pos)
TCP -> CAM , xy-plane: 55mm z: 94mm

CAM -> Target , axis : image center / 우상향으로 진행시 커짐

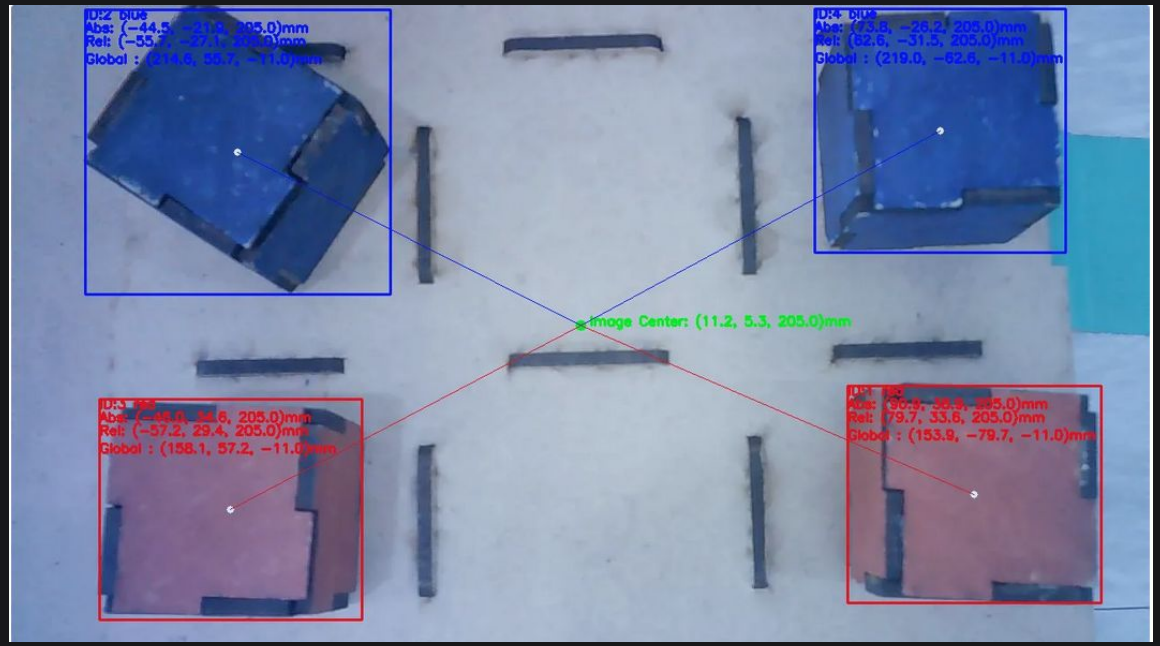
Yolo recognition

TCP -> CAM : 글루건 접착시 생기는 tilt 발생

ex)

```
camera_pos_adjustment =
np.array([-9.0, 31.0, 0.0])
```

what to pick?

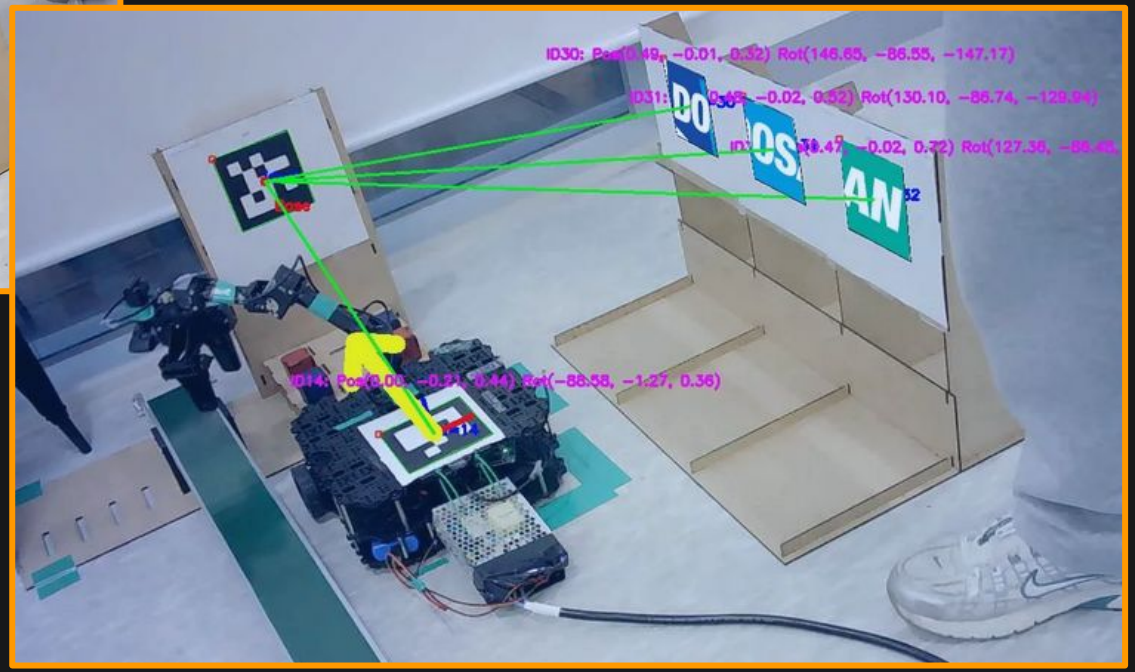


1. yolo(`class ObjectDetector:`를 통해 id,color,global_position 리스트 생성
2. GUI로부터 'robot_task_and_goal' {"color_count": {"red": 2,"blue": 1},"goal_id": 100}
3. `class BoxProcessor:`를 통해 유효성 검증

ArUco

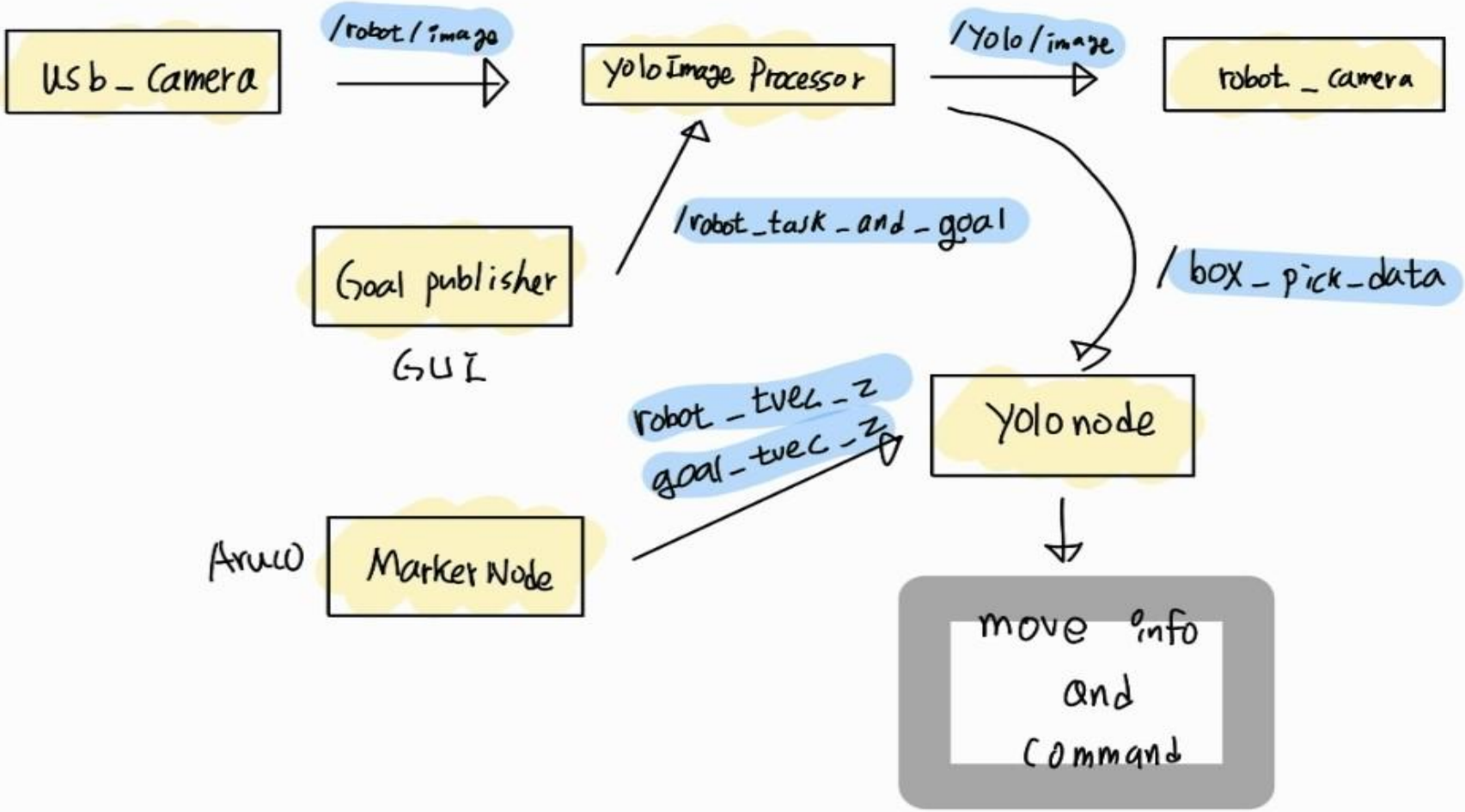
error handler

GUI ArUco Plot



ArUco - yolonode

```python
def request_marker_data(self, goal_id):
    self.pending_goal_id = goal_id  # goal_id를 저장
    msg = Int32()
    msg.data = goal_id
    self.marker_request_pub.publish(msg)
    self.get_logger().info(f"Published marker request for goal_id: {goal_id}")

def handle_marker_response(self, msg):
    try:
        data = msg.data
        if len(data) != 2:
            self.get_logger().error("Invalid marker response format.")
            return

        goal_tvec_z, robot_tvec_z = data
        self.get_logger().info(f"Received goal_tvec_z={goal_tvec_z}, robot_tvec_z={robot_tvec_z}")
        self.distance_to_base = robot_tvec_z
        self.distance_to_goal = robot_tvec_z - goal_tvec_z
    except Exception as e:
        self.get_logger().error(f"Failed to process marker response: {e}")
```


usb_camera
/robot/image
yoloImage Processor
/yolo/image
robot_camera
/robot_task_and_goal
Goal publisher
GUI
/box_pick_data
robot_tvec_z
goal_tvec_z
Yolonode
Aruco
Marker Node
move info
and
command

PROS & CONS

장점

1. 작업/비용 효율성
   - 학습 이미지 자동화로 인한 생산성 향상
   - 초기개발 비용이 적고 빠르게 구축가능

2. YOLO이용 실시간 객체탐지
   - 물체와 카메라 사이의 거리만 알면 범용 활용

3. ArUco기반 위치 추적
   - ArUco 마커로 저렴하고 간단하게 물체의 위치와 방향을 추적, 매니퓰레이터의 정확한 조작에 도움

4. 유연성
   - 다양한 객체나 환경에 활용할수있어, 다른 작업환경에도 유연하게 적용가능

단점

1. 환경제약
   - 카메라 성능과, 아루코마커의 손상이 있을경우나 조명이 일정하지 않는경우, 인식률이 떨어질 수 있음

2. 실시간 이미지 처리 성능 하드웨어 의존
   - 고해상도의 이미지와 연산이 필요 -> 저사양 환경에서 성능 문제가 생길수있음

3. 장애물 회피
   - 정해진 이동경로에 예상치 못한 장애물이 있으면 회피가 힘들 수 도 있음

DIFFICULTIES

1. math domain error

- 로봇이 케리어를 잡을 수 있는 위치와 환경임에도 불구하고 해당 에러로인해 작동하지 않는 문제

2.카메라 이슈

- 주행 중 카메라가 꺼지는 현상이 자주 발생

3. 통신속도

- 반복 주행시 통신 속도 저하로 코드와 동작 확인 어려움

4. Vision

- 전방/좌측에서 잡을때의 TF
- global coordinate의 검증
- calibration 검증

WORK EXPERIENCE

1. Vision_TF (axis)

2. 버전관리의 중요성(day5)

3. 실측의 중요성

4. 의사소통의 중요성

시연영상하이라이트

HIGHLIGHT

짤이나 영상 올리면 될듯!!

PROJECT

Q&A

Thank You

TEAM A - 5

Thank You

TEAM A - 5