

# Fullfillment 가상 서비스 환경 구축

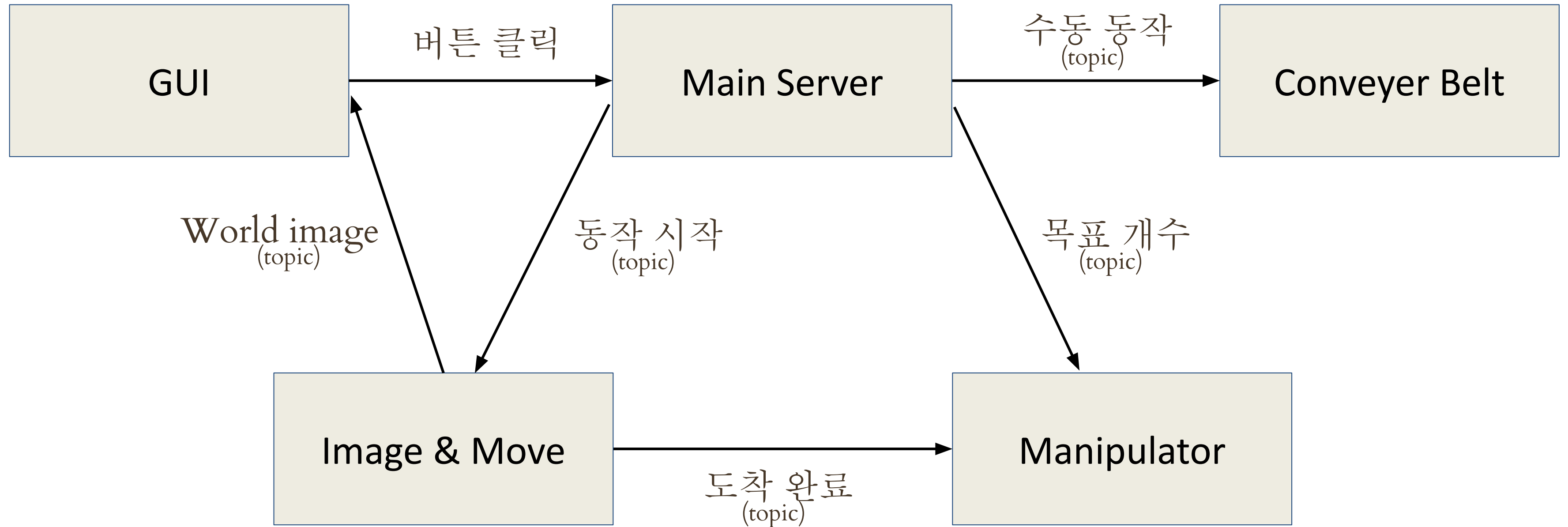
D-2 조

김우정 복권근 장세환 최현성

# 순서

1. 아키텍처 소개
2. 이동 알고리즘
3. 매니플레이터 알고리즘
4. 챌린지 포인트
5. 시연

# 아키텍처

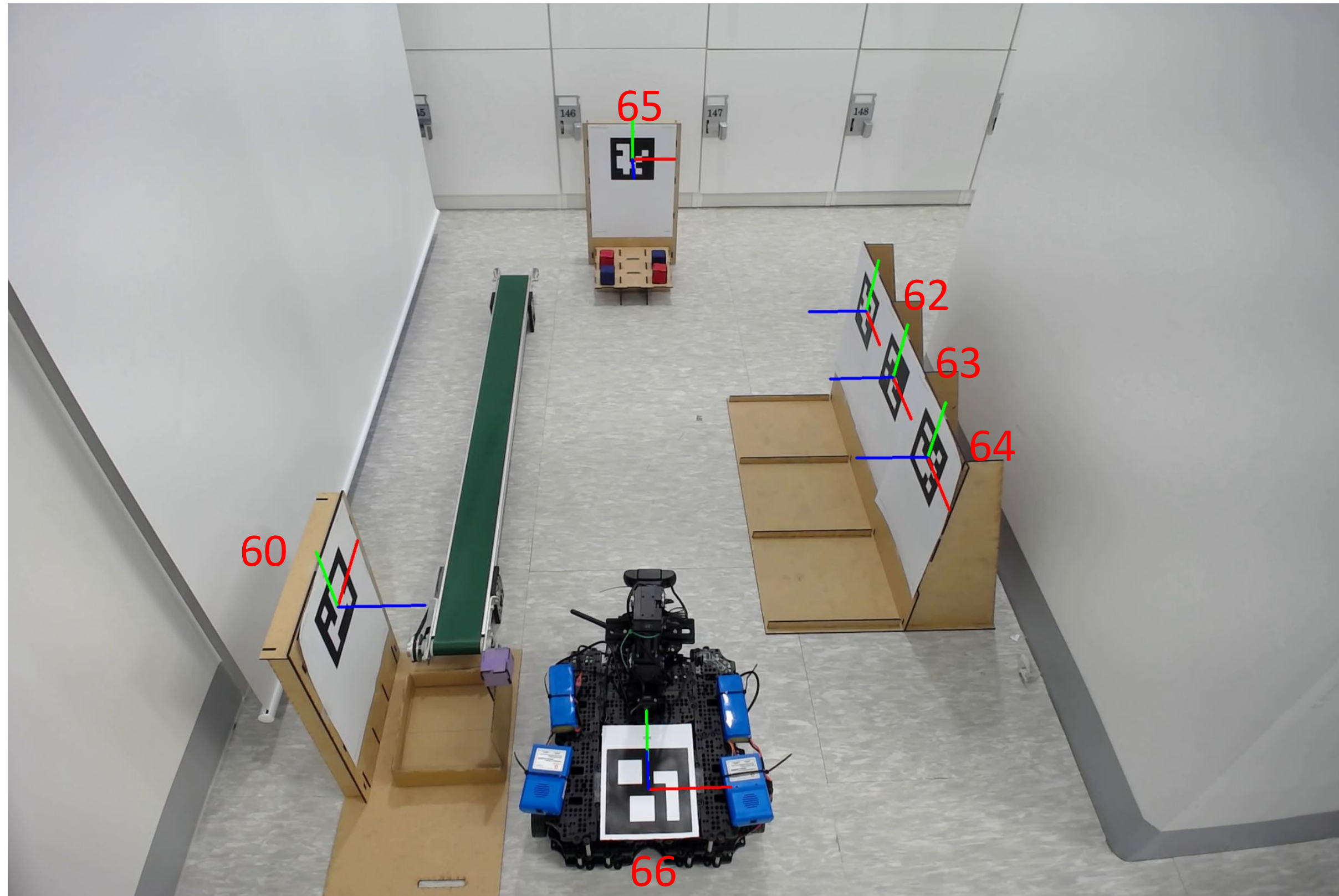


# 이동 알고리즘

---

1. 1프레임 이용 아코 마커 사이 간격 측정
  2. 자세 보정, 비상정지 스레드 동작
  3. 시작 토픽 수신 전까지 대기
  4. 각 단계별 임무 수행
-

# 이동 알고리즘





# 이동 알고리즘

```
# 카메라 매트릭스와 왜곡 계수 (캘리브레이션 후 얻은 값)
cameraMatrix = np.array([[1.38992675e+03, 0.00000000e+00, 1.09573766e+03],
                          [0.00000000e+00, 1.39126023e+03, 7.02526462e+02],
                          [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])

distCoeffs = np.array([0.13879985, -0.35371541, -0.00368127, -0.00311165, 0.22808276])

# 아루코 마커 감지
cap = cv2.VideoCapture(2)

if not cap.isOpened():
    print("캠을 열 수 없습니다.")
    exit()

ret, frame = cap.read()

if not ret:
    print("프레임을 읽을 수 없습니다.")

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_5X5_100)
parameters = aruco.DetectorParameters_create()
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)

# 마커의 위치와 회전 추정
markerLength = 0.1 # 마커의 실제 크기 (미터 단위)
rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners, markerLength, cameraMatrix, distCoeffs)

# 벡터 값 저장
self.id = ids
self.rvec = rvec
self.tvec = tvec

# 값 출력
self.get_logger().info(f"Marker ID: {self.id}")
self.get_logger().info(f"Rotation Vector: {self.rvec}")
self.get_logger().info(f"Translation Vector: {self.tvec}")

# 주어진 회전 벡터와 이동 벡터
for i, id_value in enumerate(self.id):
    id_num = id_value[0] # id_value가 리스트로 감싸져 있으므로 첫 번째 값 추출
    self.rotation_vectors[id_num] = np.array(self.rvec[i][0]) # 해당 rvec 값을 np.array 형식으로 저장
```

# 이동 알고리즘

```
# 66번 마커의 회전 벡터를 회전 행렬로 변환
R_66, _ = cv2.Rodrigues(self.rotation_vectors[66])

# 66번 마커를 기준으로 상대적 좌표 계산
relative_position_60_from_66 = R_66.T @ (self.translation_vectors[60] - self.translation_vectors[66])
relative_position_62_from_66 = R_66.T @ (self.translation_vectors[62] - self.translation_vectors[66])
relative_position_63_from_66 = R_66.T @ (self.translation_vectors[63] - self.translation_vectors[66])
relative_position_64_from_66 = R_66.T @ (self.translation_vectors[64] - self.translation_vectors[66])
relative_position_65_from_66 = R_66.T @ (self.translation_vectors[65] - self.translation_vectors[66])

# 매니퓰레이터 위치 보정
self.y_60location = relative_position_60_from_66[1] - 0.14
self.y_62location = relative_position_62_from_66[1] - 0.15
self.y_63location = relative_position_63_from_66[1] - 0.15
self.y_64location = relative_position_64_from_66[1] - 0.15

# 작업 테이블 보정 + 로봇 크기 보정
self.y_65location = relative_position_65_from_66[1] - 0.24 - 0.2
```



# 이동 알고리즘

---

```
# 스레드로 실시간 프레임 읽고 자세 보정
self.vector_thread = threading.Thread(target=self.read_frame)
self.vector_thread.start()

# 0단계
while self.step == 0:
    self.get_logger().info("0. 대기중")
    rclpy.spin_once(self)

# 비상정지 스레드
self.stop_thread = threading.Thread(target=self.stop)
self.stop_thread.start()
```



# 이동 알고리즘

```
# 주어진 회전 벡터와 이동 벡터
for i, id_value in enumerate(self.id):
    id_num = id_value[0] # id_value가 리스트로 감싸져 있으므로 첫 번째 값 추출
    self.rotation_vectors[id_num] = np.array(self.rvec[i][0]) # 해당 rvec 값을 np.array 형식으로 저장

self.angle = self.rotation_vectors[66][1]
msg = Twist()

if self.angle < -0.05: # 오른쪽 회전
    msg.angular.z = -0.1
    #self.get_logger().info(f"회전 보정: {self.angle}오른쪽 회전")

elif self.angle > 0.05: # 왼쪽 회전
    msg.angular.z = 0.1
    #self.get_logger().info(f"회전 보정: {self.angle}왼쪽 회전")

else:
    msg.angular.z = 0.0
    #self.get_logger().info(f"회전 보정 완료{self.angle}")

self.move_publisher.publish(msg)
```

# 이동 알고리즘

---

```
def stop(self):  
    self.get_logger().info("비상 정지 대기")  
    while self.stop_check == 0:  
        rclpy.spin_once(self)  
    rclpy.shutdown()
```

# 이동 알고리즘

```
# 1단계
self.sleep_thread = threading.Thread(target=self.sleep_task, args=(self.y_65location * 10,))
self.sleep_thread.start()

while self.step == 1:
    self.get_logger().info("1. 작업물 접근")
    self.move_forward()

# 2단계
arrive_msg = Int32()
arrive_msg.data = 1
self.arrive_publisher.publish(arrive_msg)

self.sleep_temp_thread = threading.Thread(target=self.sleep_task, args=(80,))
self.sleep_temp_thread.start()

while self.step == 2:
    self.get_logger().info("2. 상자 컨베이어벨트 옮기기")
```

```
def sleep_task(self, wait_time):
    self.get_logger().info(f"Sleeping for {wait_time}sec in a separate thread...")
    threading.Event().wait(wait_time)

    # 다음 단계 진입 확인 변수
    self.step += 1
```



# 이동 알고리즘

```
# 0단계
while self.step == 0:
    self.get_logger().info("0. 대기중")
    rclpy.spin_once(self)

# 비상정지 스레드
self.stop_thread = threading.Thread(target=self.stop)
self.stop_thread.start()

# 1단계
self.sleep_thread = threading.Thread(target=self.sleep_task, args=(self.y_65location * 10,))
self.sleep_thread.start()

while self.step == 1:
    self.get_logger().info("1. 작업을 접근")
    self.move_forward()

# 2단계
arrive_msg = Int32()
arrive_msg.data = 1
self.arrive_publisher.publish(arrive_msg)

self.sleep_temp_thread = threading.Thread(target=self.sleep_task, args=(80,))
self.sleep_temp_thread.start()

while self.step == 2:
    self.get_logger().info("2. 상자 컨베이어벨트 옮기기")

# 3단계
self.sleep_thread = threading.Thread(target=self.sleep_task, args=((self.y_65location - self.y_60location) * 10,))
self.sleep_thread.start()

while self.step == 3:
    self.get_logger().info("3. 바구니 이동")
    self.move_backward()

# 4단계
arrive_msg.data = 2
self.arrive_publisher.publish(arrive_msg)

self.sleep_thread = threading.Thread(target=self.sleep_task, args=(17,))
self.sleep_thread.start()

while self.step == 4:
    self.get_logger().info("4. 상자 확인 및 잡기")

# 5단계
# 목표지점 62
self.sleep_thread = threading.Thread(target=self.sleep_task, args=((self.y_62location - self.y_60location) * 10,))
# 목표지점 63
self.sleep_thread = threading.Thread(target=self.sleep_task, args=((self.y_63location - self.y_60location) * 10,))
# 목표지점 64
self.sleep_thread = threading.Thread(target=self.sleep_task, args=((self.y_64location - self.y_60location) * 10,))
self.sleep_thread.start()

while self.step == 5:
    self.get_logger().info("5. 목표 지점 이동")
    self.move_forward()

# 6단계
arrive_msg.data = 3
self.arrive_publisher.publish(arrive_msg)

self.sleep_thread = threading.Thread(target=self.sleep_task, args=(5,))
self.sleep_thread.start()

while self.step == 6:
    self.get_logger().info("6. 목표지점 바구니 옮기기")

self.get_logger().info("<< 작업 완료 >>")
```



# UI 설정

로그인

사용자명:

비밀번호:

로그인

Control\_tower

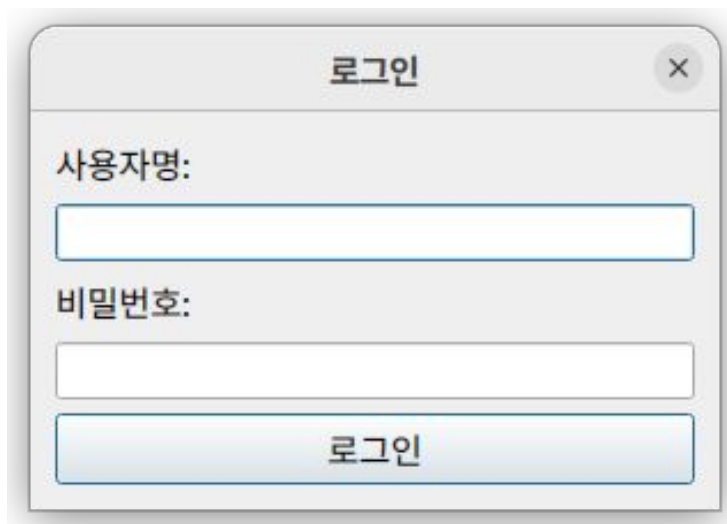
worldbot1bot2

작업 소요 시간 :작업 상태 : 대기 중

Job1Job2Job3시작비상 정지일시정지재가동리셋

작업 목록 :컨베이어 시작컨베이어 정지학습 데이터 수집

# UI 설정



```
class LoginDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("로그인")
        self.setGeometry(100, 100, 300, 150)

        layout = QVBoxLayout()

        self.username_label = QLabel("사용자명:")
        self.password_label = QLabel("비밀번호:")

        self.username_input = QLineEdit(self)
        self.password_input = QLineEdit(self)
        self.password_input.setEchoMode(QLineEdit.Password) # 비밀번호는 숨김 처리

        self.login_button = QPushButton("로그인")
        self.login_button.clicked.connect(self.handle_login)

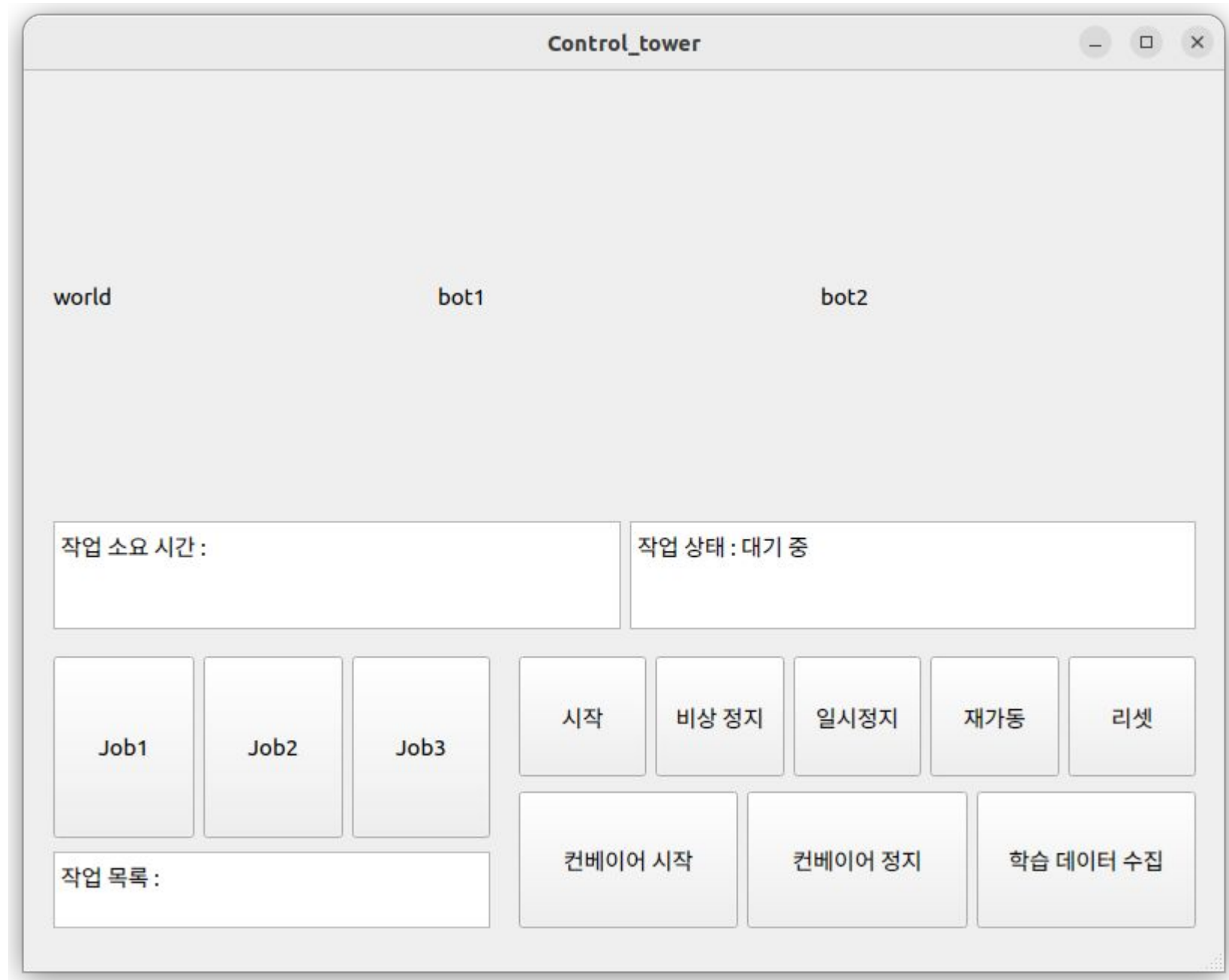
        layout.addWidget(self.username_label)
        layout.addWidget(self.username_input)
        layout.addWidget(self.password_label)
        layout.addWidget(self.password_input)
        layout.addWidget(self.login_button)

        self.setLayout(layout)

    def handle_login(self):
        username = self.username_input.text()
        password = self.password_input.text()

        # 간단한 로그인 인증 (사용자명: 'user', 비밀번호: 'password' 예시)
        if username == "1" and password == "1":
            self.accept() # 로그인 성공
        else:
            self.reject() # 로그인 실패
```

# UI 설정



```
# 시작 버튼 클릭 이벤트 연결
self.start.clicked.connect(self.send_message_to_service)
self.start.clicked.connect(self.on_start_button_click)
# 정지 버튼 클릭 이벤트 연결
self.stop.clicked.connect(self.on_stop_button_click)
# 일시정지 버튼 클릭 이벤트 연결
self.pause.clicked.connect(self.on_pause_button_click)
# 계속 버튼 클릭 이벤트
self.resume.clicked.connect(self.on_resume_button_click)
# 리셋 버튼 클릭 이벤트
self.reset.clicked.connect(self.on_reset_button_click)

self.constart.clicked.connect(self.constart_button_click)
self.constop.clicked.connect(self.constop_button_click)
self.data.clicked.connect(self.data_button_click)
```



# UI 설정 - 시작 버튼

Job1	Job2	Job3
RED: 2, BLUE: 1		
Job1	Job2	Job3
RED: 1, BLUE: 2		
Job1	Job2	Job3
RED: 1, BLUE: 1		

sorkaksema@naver.com	28분 전
긴급정지	
긴급정지	49분 전
긴급정지	
긴급정지	51분 전
긴급정지	
긴급정지	51분 전
긴급정지	
아두이노 연결 실패	

```
def call_service(self, service_name, job_list):
    while not self.client.wait_for_service(timeout_sec=1.0):
        self.get_logger().info(f"Service {service_name} not available, waiting...")
    request = StartTime.Request()
    request.job_list = job_list
    future = self.client.call_async(request)
    rclpy.spin_until_future_complete(self, future)
    response = future.result()

    if response:
        self.get_logger().info(f"Service Response: {response.status}")
    else:
        self.get_logger().warn("No response received from service.")
    return response
```

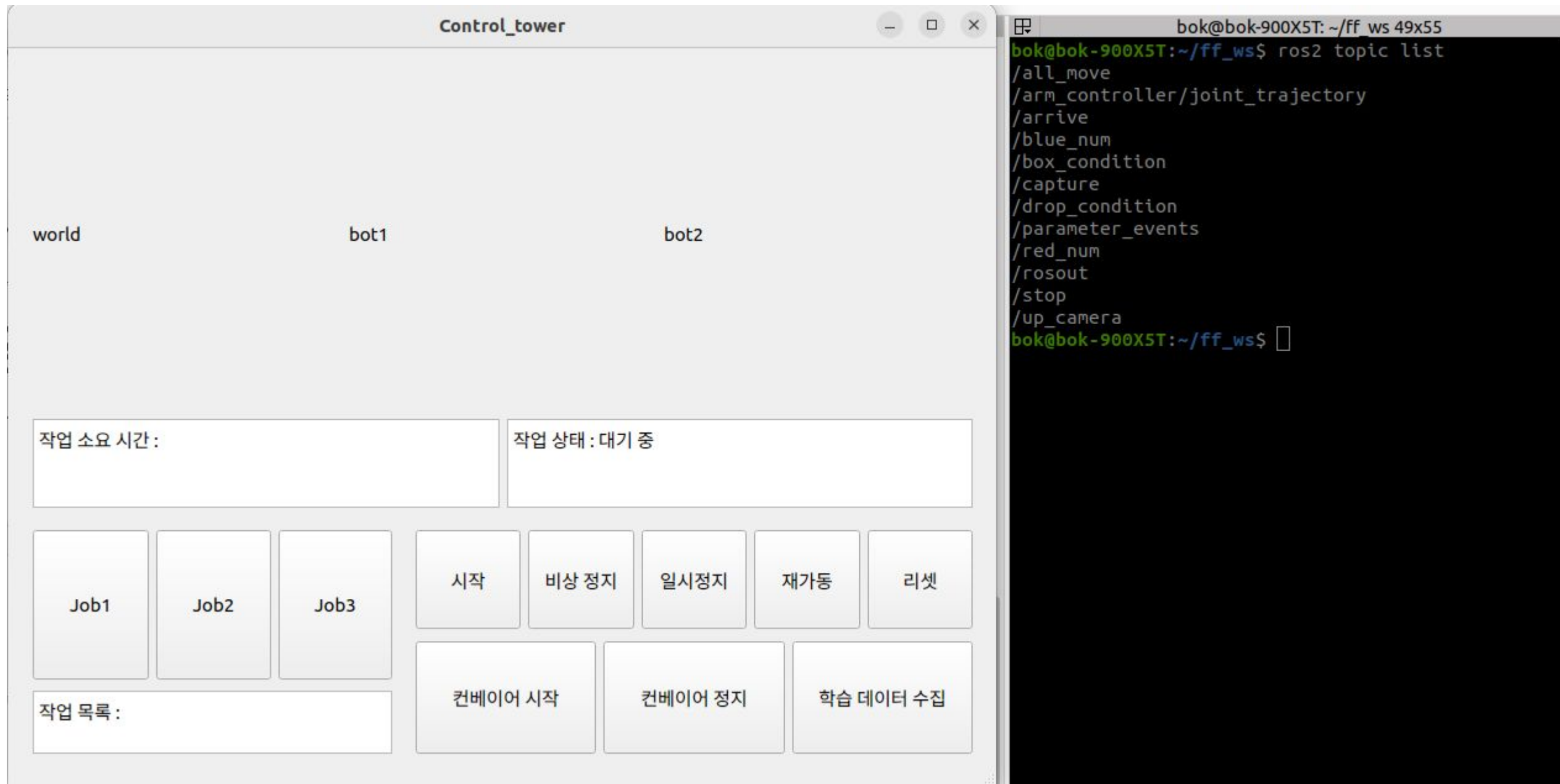
```
def check_arduino(self):
    ports = [port.device for port in serial.tools.list_ports.comports()]
    try:
        if self.arduino is None or not self.arduino.isOpen():
            # 연결이 없거나, 연결이 끊어진 경우
            print("아두이노 연결 시도 중...")
            self.arduino = serial.Serial(self.arduino_port, self.baudrate, timeout=1)
            print('연결 완료!')

            # 연결이 정상인 경우
            if self.arduino.isOpen():
                self.connection = 1

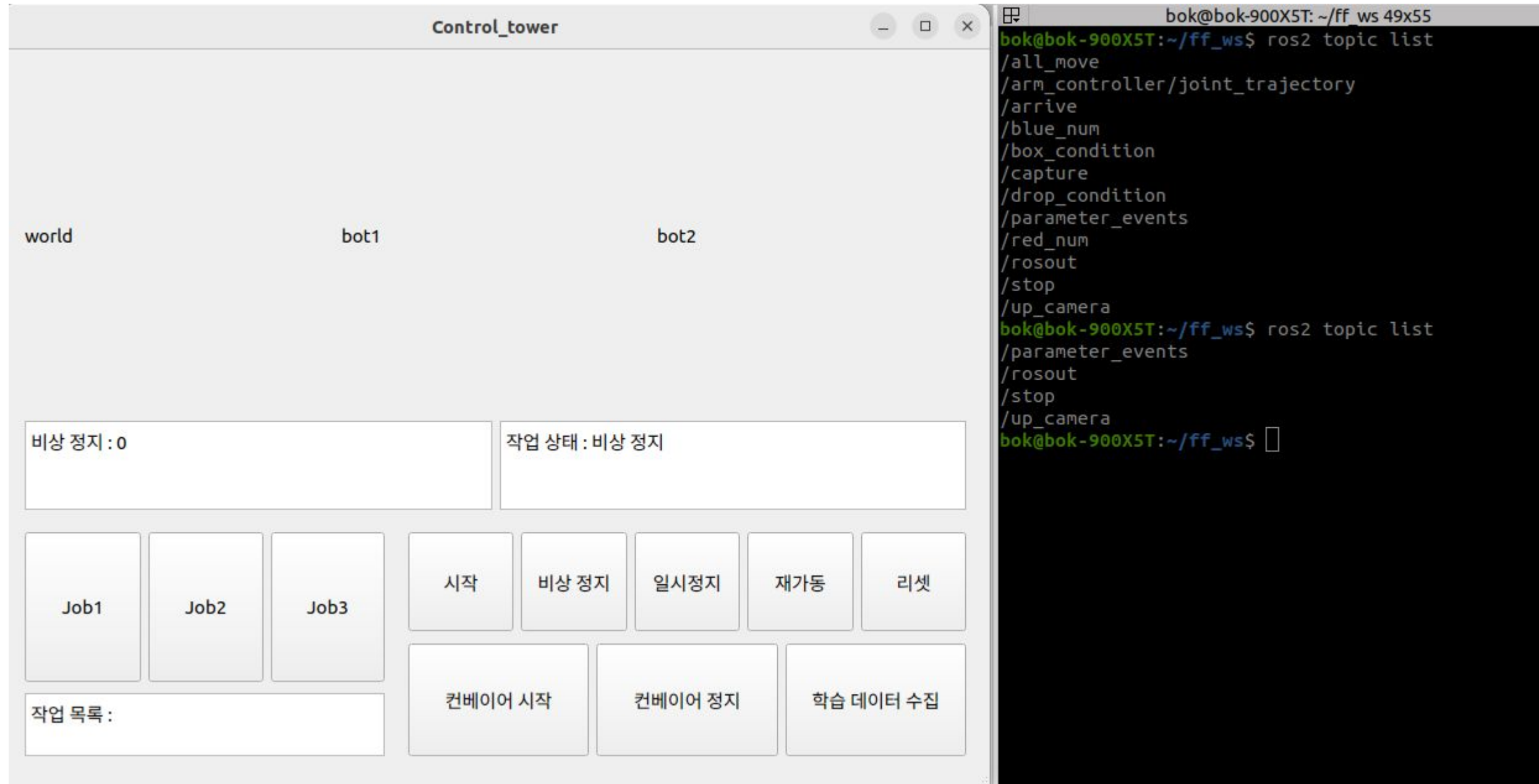
        if '/dev/ttyACM0' not in ports:
            # 연결 실패 처리
            self.send_email("sorkaksema@naver.com", "XPLW2BFKH76J", "sorkaksema@naver.com")
            print('연결 실패')
            self.arduino = None
    except Exception as e:
        print(f'오류 사항 : {e}')
```



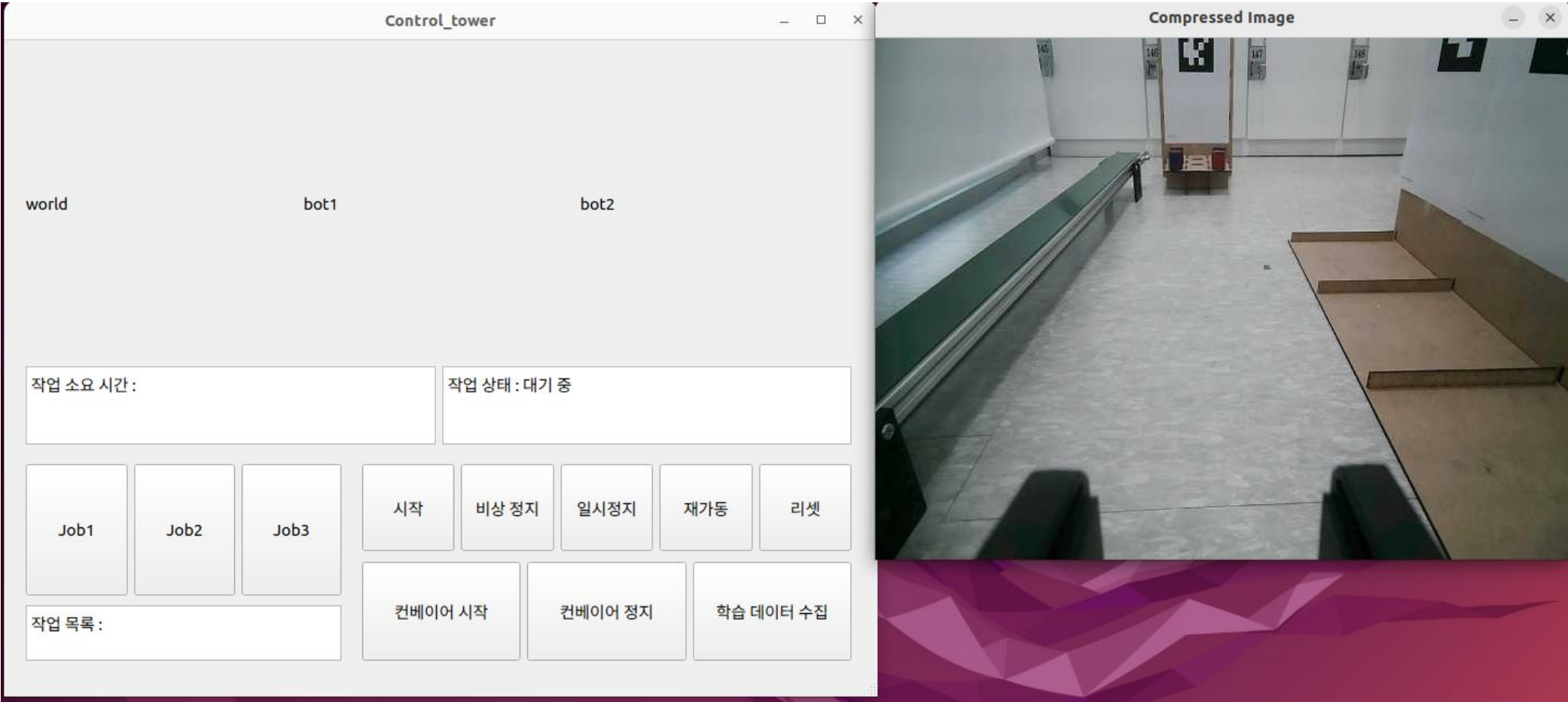
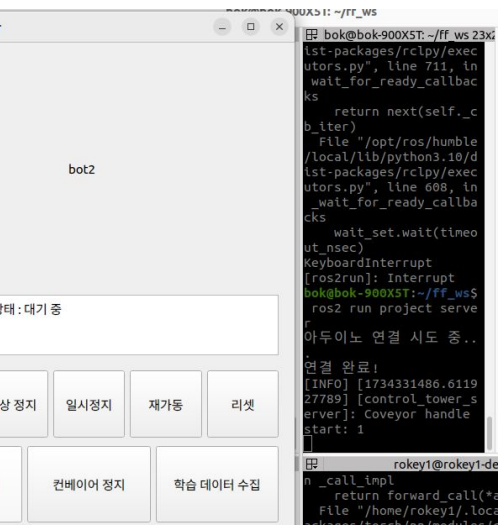
# UI 설정 - 비상정지



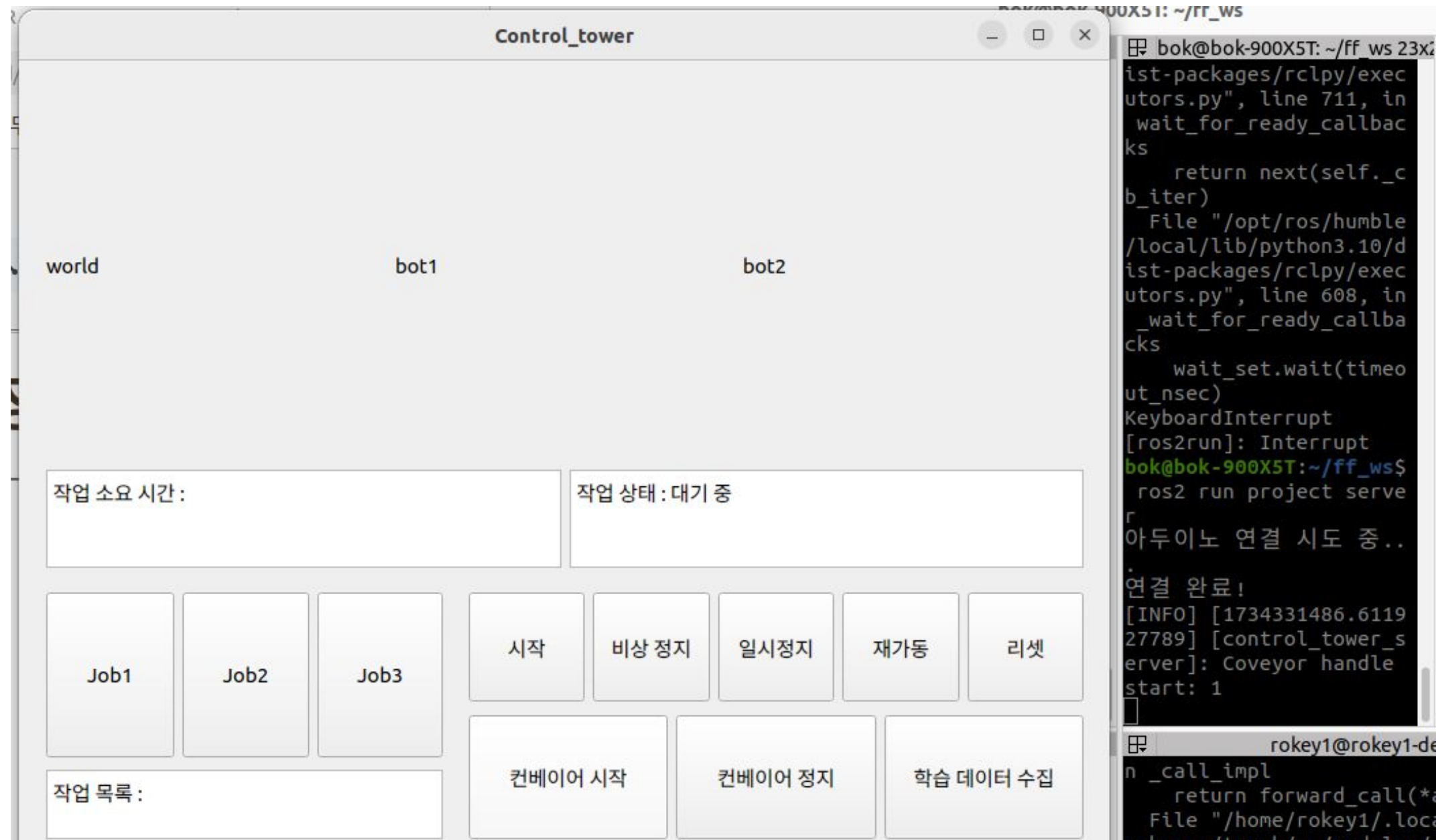
# UI 설정 - 비상정지



# UI 설정 - 학습데이터 수집

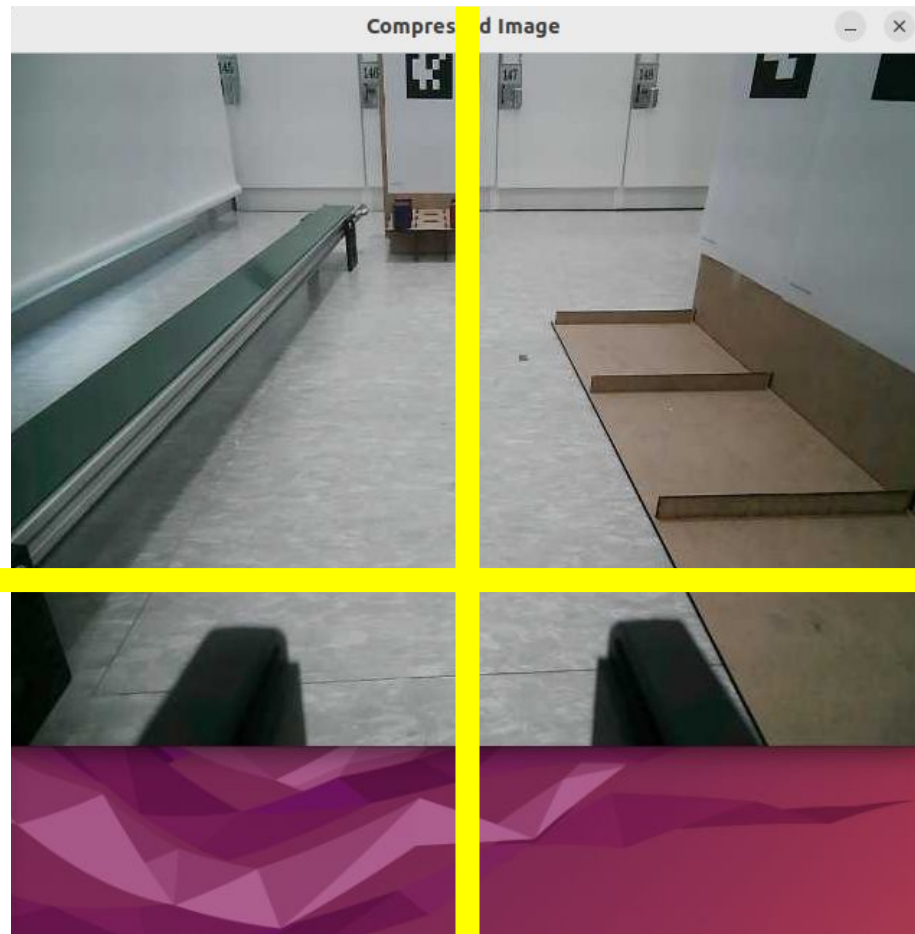


# UI 설정 - 컨베이어 시작



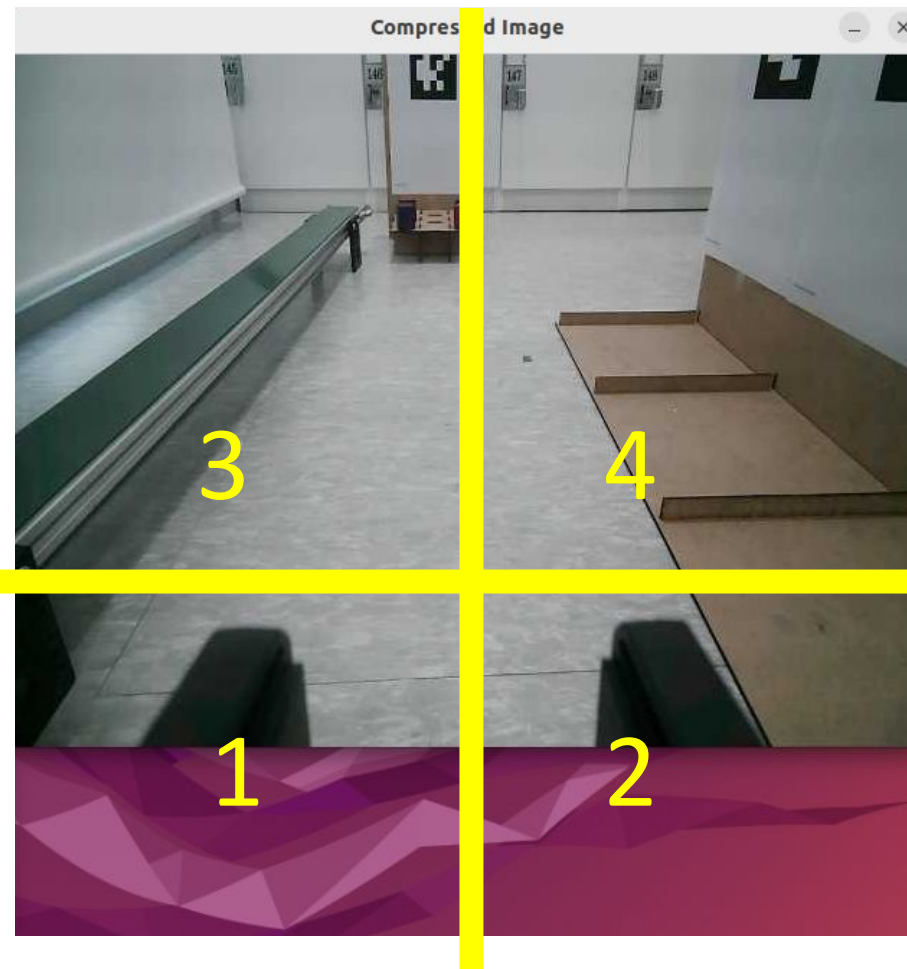
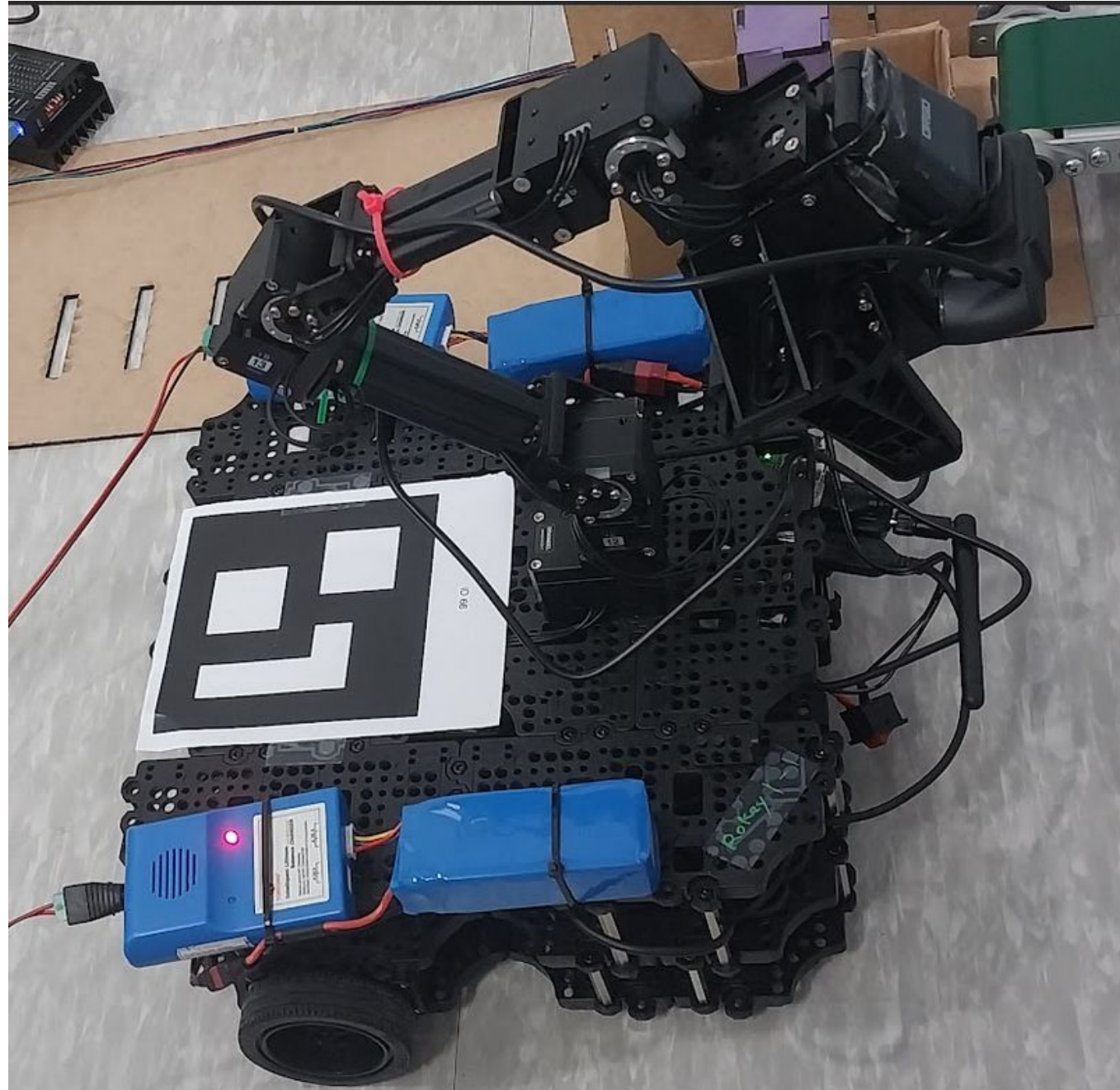


# 카메라 인식



```
def detect_objects_in_zones(self, frame):  
    """구역별로 물체 감지 후 해당 구역 번호와 색상 정보 반환"""  
    # 구역 좌표 설정  
    zones = [  
        1: (0, 370, 312, 480), # 왼쪽 아래  
        2: (312, 370, 640, 480), # 오른쪽 아래  
        3: (0, 0, 312, 370), # 왼쪽 위  
        4: (312, 0, 640, 370) # 오른쪽 위  
    ]
```

# 매니퓰레이터 동작



```
def grip_one(self):
    self.move_arm_to_position(145, 60, 50)
    time.sleep(2)
    self.get_logger().info('1번 자리 그림')
    self.send_gripper_goal(-0.15)

def grip_two(self):
    self.move_arm_to_position(145, -70, 50)
    time.sleep(2)
    self.get_logger().info('2번 자리 그림')
    self.send_gripper_goal(-0.15)

def grip_three(self):
    self.move_arm_to_position(200, 65, 50)
    time.sleep(2)
    self.get_logger().info('3번 자리 그림')
    self.send_gripper_goal(-0.15)
    #1000*90

def grip_four(self):
    self.move_arm_to_position(200, -70, 50)
    time.sleep(2)
    self.get_logger().info('4번 자리 그림')
    self.send_gripper_goal(-0.15)
```



# 카메라 인식

```
results_in_zone = self.model(zone_frame)

# red 또는 blue 물체가 감지되었는지 확인
for box in results_in_zone[0].boxes:
    class_id = int(box.cls)
    confidence = box.conf.item()

    # 물체가 'red' 또는 'blue'일 경우 해당 구역으로 감지
    label = self.model.names[class_id]
    if label in ["red", "blue"] and confidence > 0.7:
        # 이미 해당 색이 감지된 경우 다음 구역으로 넘어감
        if (label == "red" and self.current_red >= self.total_red) or (label == "blue" and self.current_blue >= self.total_blue):
            continue

        detected_info = f"Zone {zone_num} - {label.capitalize()}" # 감지된 구역과 색상 정보를 업데이트

        # 바운딩 박스를 그리기 (물체 감지)
        x1_box, y1_box, x2_box, y2_box = map(int, box.xyxy[0])
        cv2.rectangle(frame, (x1 + x1_box, y1 + y1_box), (x1 + x2_box, y1 + y2_box), (0, 255, 0), 2)
        cv2.putText(frame, f'{label} ({confidence:.2f})',
                    (x1 + x1_box, y1 + y1_box - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

# 감지된 물체가 있으면 더 이상 다른 구역을 확인하지 않음
if detected_info != "None":
    break
```

```
for zone_num in [1, 2, 3, 4]: # 우선순위로 1, 2, 3, 4 순으로 확인
    # 각 구역에서 빨간색과 파란색 감지 여부를 확인하는 조건
    if self.total_red == 0 and self.total_blue == 0:
        self.get_logger().info('Total값을 기다리는중')
        return frame, detected_info
    if (self.current_red >= self.total_red and self.current_blue >= self.total_blue):
        self.total_red = 0
        self.total_blue = 0
        self.publisher_drop.publish(Bool(data=True))
        break # 현재 빨간색과 파란색이 모두 total 값을 초과하면 더 이상 감지하지 않음
```

```
def box_condition(self, msg):
    if not self.ignore:
        try:
            # 구역 및 색상 정보 파싱
            detected_info = msg.data
            if detected_info == 'go_purple':
                return
            # "Zone X - Color" 형식의 문자열 파싱
            else:
                parts = detected_info.split(" - ")
                if len(parts) == 2:
                    self.box_position = int(parts[0].split(" ")[1]) # "Zone X"에서 X 추출
                    self.detected_color = parts[1] # 색상 추출
        except Exception as e:
            self.get_logger().error(f"Failed to parse detected info: {str(e)}")
```

# 매니플레이터 동작

## 박스 인식

```
def box_condition(self, msg):
    if not self.ignore:
        try:
            # 구역 및 색상 정보 파싱
            detected_info = msg.data
            if detected_info == 'go_purple':
                return
            # "Zone X - Color" 형식의 문자열 파싱
        except:
            else:
                parts = detected_info.split(" - ")
                if len(parts) == 2:
                    self.box_position = int(parts[0].split(" ")[1]) # "Zone X"에서 X 추출
                    self.detected_color = parts[1] # 색상 추출
        except Exception as e:
            self.get_logger().error(f"Failed to parse detected info: {str(e)}")
```

## 박스를 집는 과정

```
def arrive_callback(self, msg):
    self.arrive = msg.data
    if self.arrive == 1 and self.arrive_c == 0:
        self.mani_home()
        # 그리퍼 초기화
        self.send_gripper_goal(0.025)
        self.arrive_c += 1
        self.move()
```

## 바구니를 집고 옮기는 과정

```
if self.arrive == 2:
    self.get_logger().info('보라박스')
    self.send_gripper_goal(0.025)
    time.sleep(2)
    self.move_arm_to_position(0, 150, 200)
    time.sleep(2)
    self.move_arm_to_position(0, 190, 140)
    time.sleep(2)
    self.send_gripper_goal(-0.15)
    time.sleep(2)
    self.trajectory_msg.points[0].positions[0] += 0.2
    self.joint_pub.publish(self.trajectory_msg)
    time.sleep(1)
    self.move_arm_to_position(0, 40, 230)
    time.sleep(1)
    self.move_arm_to_position(0, -40, 230)

if self.arrive == 3:
    self.get_logger().info('박스 내려놓기')
    self.move_arm_to_position(0, -200, 150)
    time.sleep(2)
    self.send_gripper_goal(0.025)
    time.sleep(2)
    self.move_arm_to_position(0, -100, 200)
    time.sleep(2)
    self.mani_home()
    time.sleep(2)
```



# 매니플레이터 동작

## 박스를 집는 과정

```
def move(self):  
    # box_position이 업데이트될 때까지 기다리기 위한 타이머 설정  
    self.timer = self.create_timer(0.1, self.check_box_position)
```

```
def check_box_position(self):  
    # box_position이 0이 아니고 ignore가 False일 때만 작업 시작  
    if self.box_position != 0 and not self.ignore:  
        self.timer.cancel() # 타이머 종료  
        if self.detected_color == 'Red':  
            self.red+=1  
            self.red_count.publish(Int32(data = self.red))  
            self.get_logger().info(f'published RED count {self.red}')  
        elif self.detected_color == 'Blue':  
            self.blue+=1  
            self.blue_count.publish(Int32(data = self.blue))  
            self.get_logger().info(f'published BLUE count {self.blue}')  
  
        self.ignore = True # Prevent further interference  
        self.get_logger().info(f'좌표 확인 된 곳: {self.box_position}')  
        self.capture_publisher.publish(Int32(data=1))  
        self.capture_publisher.publish(Int32(data=0))  
        if self.box_position == 1:  
            self.grip_one()  
        elif self.box_position == 2:  
            self.grip_two()  
        elif self.box_position == 3:  
            self.grip_three()  
        elif self.box_position == 4:  
            self.grip_four()  
        else:  
            pass
```

```
self.capture_publisher.publish(Int32(data=2))  
self.capture_publisher.publish(Int32(data=0))  
self.get_logger().info(f'자리 확인: {self.box_position}')  
self.trajectory_msg.points[0].positions[1] -= 0.5  
self.joint_pub.publish(self.trajectory_msg)  
time.sleep(2)  
self.drop_con()  
self.mani_home()  
self.box_position = 0  
self.detected_color = " "  
  
self.get_logger().info(f'박스 위치: {self.box_position}   박스 색: {self.detected_color}')  
self.timer = self.create_timer(0.1, self.check_box_position)
```

# 어려웠던 점

## 1. 토픽 대기 상태 구현

```
# 스레드로 실시간 프레임 읽고 자세 보정
self.vector_thread = threading.Thread(target=self.read_frame)
self.vector_thread.start()

# 0단계
while self.step == 0:
    self.get_logger().info("0. 대기중")
    rclpy.spin_once(self)

# 비상정지 스레드
self.stop_thread = threading.Thread(target=self.stop)
self.stop_thread.start()
```



# 어려웠던 점

## 2. 다양한 노드의 활용 실패

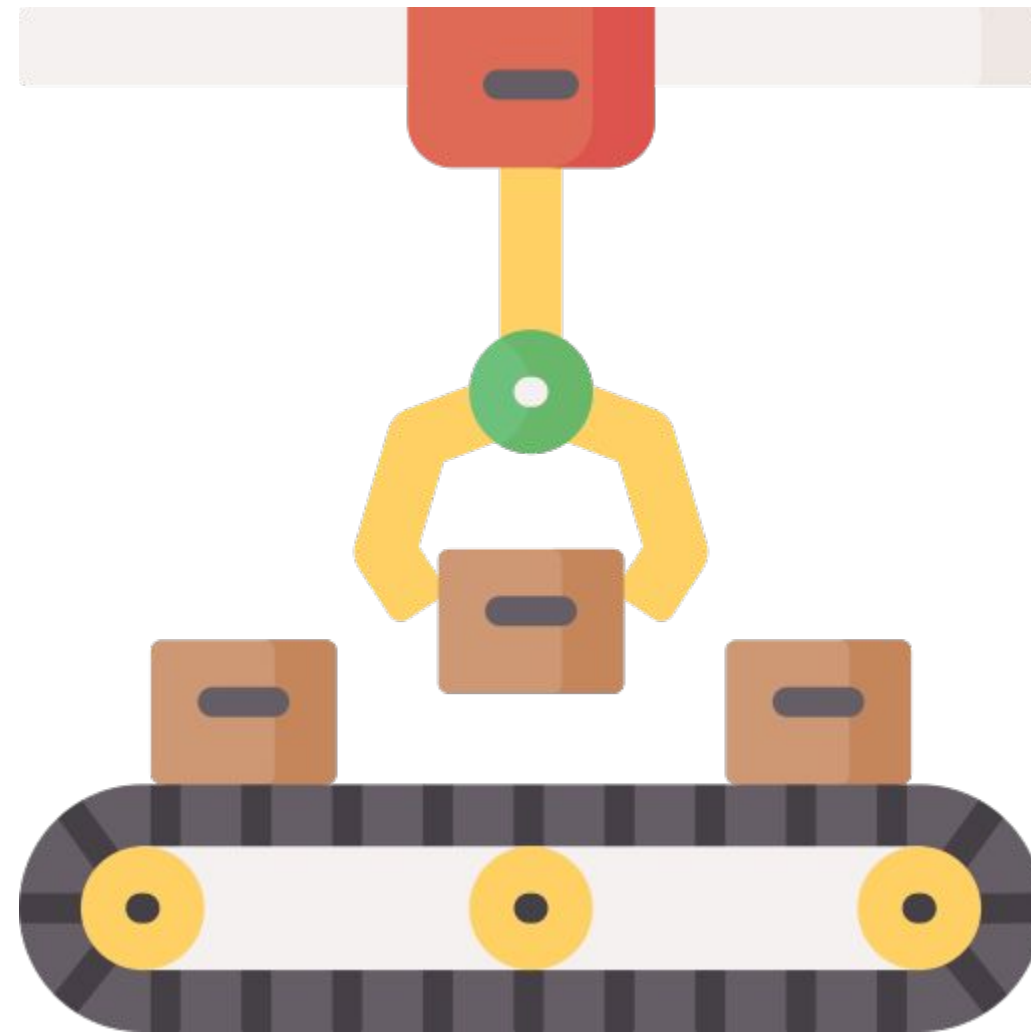
- camera.py
- control\_tower\_server.py
- control\_tower.py
- fix\_rotation.py
- subscription.py

```
self.joint_pub = self.create_publisher(JointTrajectory, '/arm_controller/joint_trajectory', 10)
self.gripper_action_client = ActionClient(self, GripperCommand, 'gripper_controller/gripper_cmd')
self.drop_pub = self.create_publisher(Int32, 'drop_condition', 10)
self.create_subscription(Int32, 'stop', self.stop_callback, 10)
# box_condition 토픽 구독
self.create_subscription(String, 'box_condition', self.box_condition, 10)

self.blue_count = self.create_publisher(Int32, 'blue_num', 10)
self.red_count = self.create_publisher(Int32, 'red_num', 10)
self.create_subscription(Int32, 'arrive', self.arrive_callback, 10)
self.capture_publisher = self.create_publisher(Int32, 'capture', 10)
self.subscription_all_move = self.create_subscription(Int32, 'all_move', self.all_move_callback, 10)
```

# 프로젝트 시연

---





# Thank You

D - 2