# Boss Baby

D-2 조 김우정 복권근 장세환 최현성

# 순서

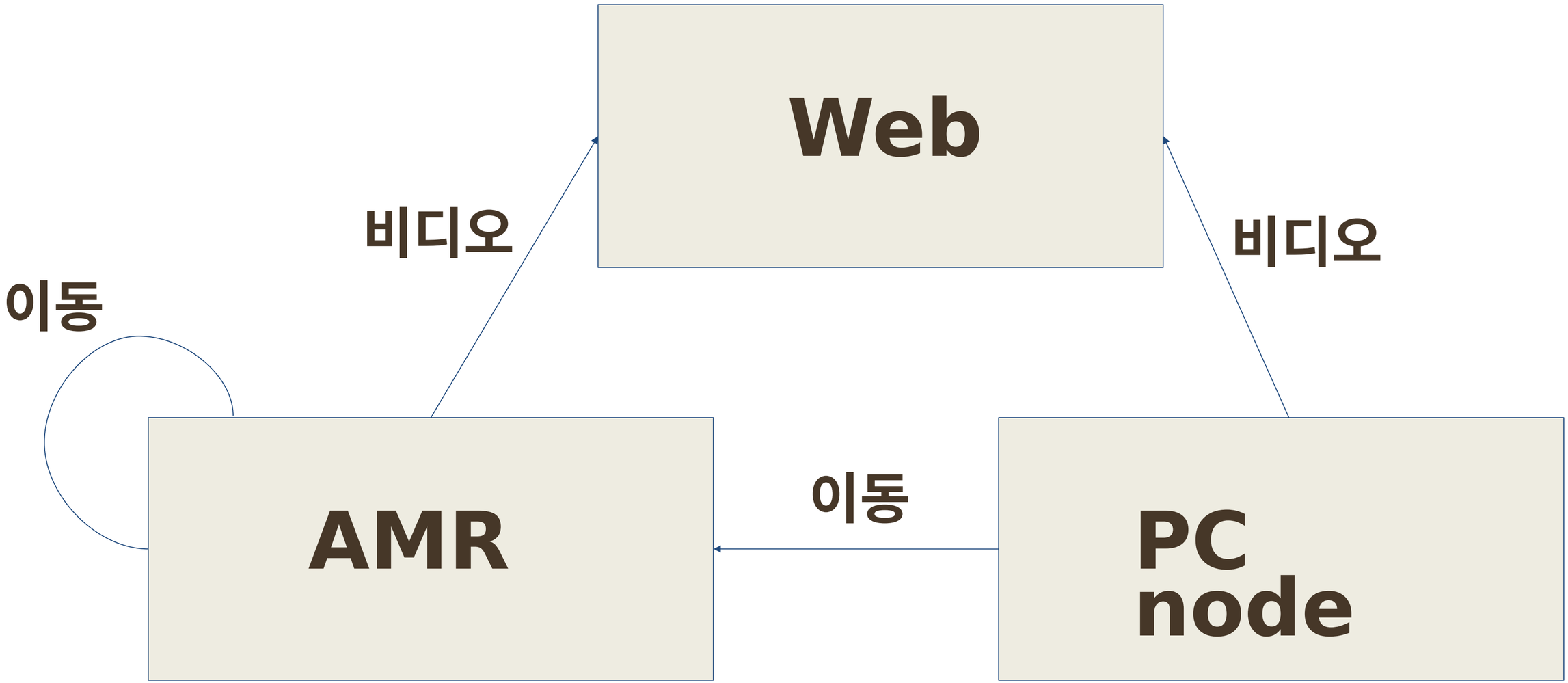
1. Solution OverView
2. 메인 기능(코드 리뷰) 및 어려웠던 점
3. 프로젝트 발전 방향
4. 배운점
5. 프로젝트 기여

# OverView

# 메인 기능

1. 아기와 위험 물체 객체 인식
2. DB와 AMR 수동 조작
3. 아기 따라다니기
4. 위험 물체와 아기 사이 가로 막기
5. 안전 구역 이탈 아기 가로 막기

# 아키텍쳐



# 아기와 위험 물체 객체 인식

```python
# 퍼블리셔 설정
self.publisher_image = self.create_publisher(
    CompressedImage,
    'amr_camera_image',
    10
)
self.publisher_detection = self.create_publisher(
    String,
    'detected',
    10
)
```

```python
baby/test
# 타이머 설정
self.timer_image = self.create_timer(0.2, self.publish_image)
self.timer_detection = self.create_timer(1, self.publish_detection)
```

# 아기와 위험 물체 객체 인식

```python
def publish_image(self):
    """카메라 프레임을 캡처, YOLO로 처리 후 퍼블리시."""
    ret, frame = self.capture.read()
    if ret:
        frame = cv2.resize(frame, (640, 480))  # 프레임 크기 축소
        results = self.model(frame)
        self.detection_info = self.draw_bounding_boxes(frame, results)
        self.publish_compressed_image(frame)

def publish_compressed_image(self, frame):
    """이미지를 압축하여 퍼블리시."""
    encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 50]
    _, compressed_frame = cv2.imencode('.jpg', frame, encode_param)
    compressed_image_msg = CompressedImage()
    compressed_image_msg.header.stamp = self.get_clock().now().to_msg()
    compressed_image_msg.header.frame_id = 'camera'
    compressed_image_msg.format = 'jpeg'
    compressed_image_msg.data = compressed_frame.tobytes()
    self.publisher_image.publish(compressed_image_msg)
```

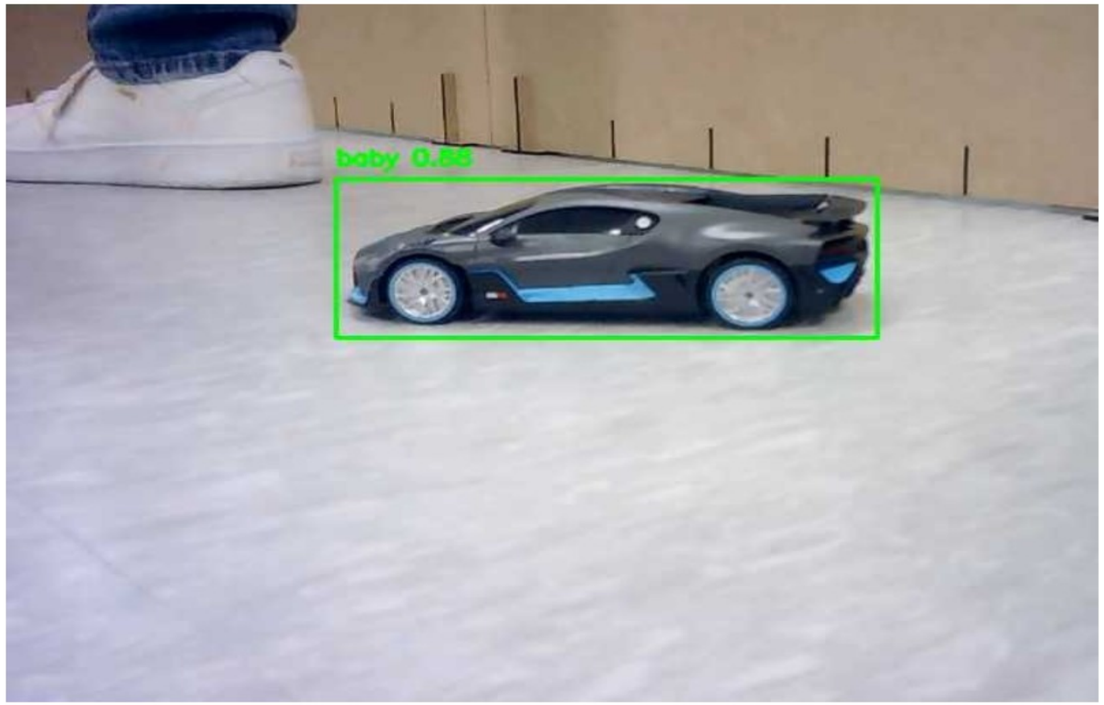
# 아기와 위험 물체 객체 인식

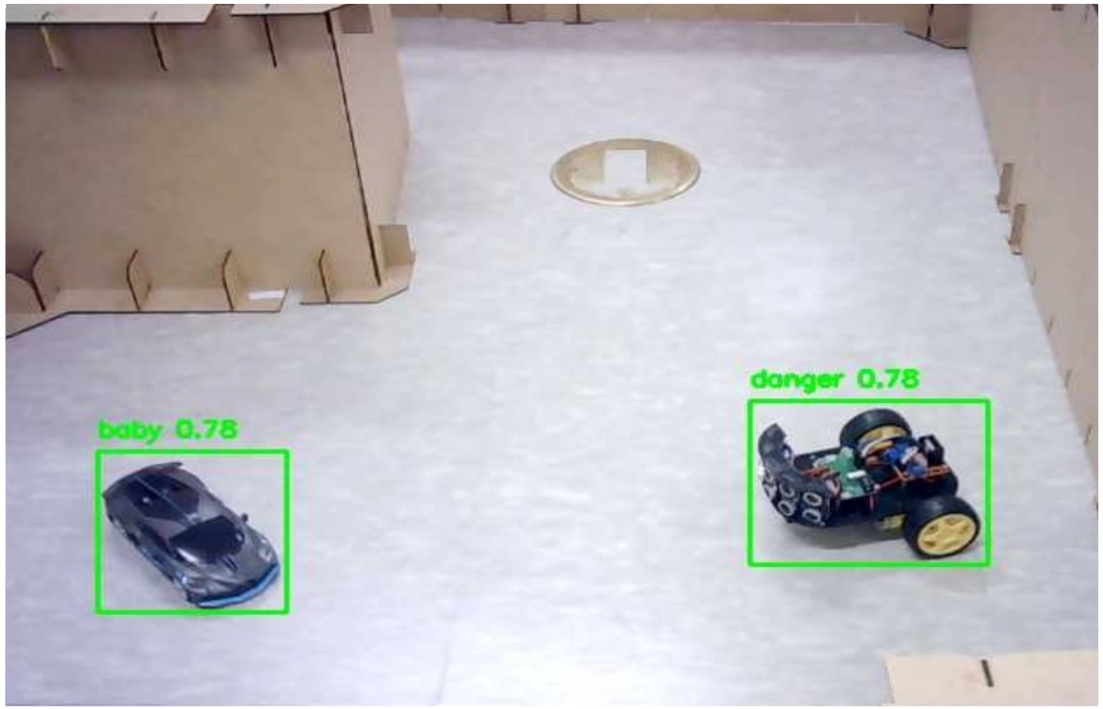
```python
def draw_bounding_boxes(self, frame, results):
    """YOLOv8 결과를 기반으로 각 클래스별로 신뢰도가 가장 높은 객체만 표시."""
    class_best_boxes = {}
    self.baby_detected = False

    for result in results[0].boxes.data.tolist():
        x1, y1, x2, y2 = map(int, result[:4])
        score = result[4]
        class_id = int(result[5])
        class_name = self.model.names[class_id]

        if class_id not in class_best_boxes:
            class_best_boxes[class_id] = {"score": score, "bbox": (x1, y1, x2, y2), "class_name": class_name}
        elif score > class_best_boxes[class_id]["score"]:
            class_best_boxes[class_id] = {"score": score, "bbox": (x1, y1, x2, y2), "class_name": class_name}

    detection_info = ""
    for class_id, best_box in class_best_boxes.items():
        x1, y1, x2, y2 = best_box["bbox"]
        score = best_box["score"]
        class_name = best_box["class_name"]

        if class_name == 'baby':
            self.baby_detected = True
            self.baby_bounding_box = {"x_center": (x1 + x2) / 2, "y_center": (y1 + y2) / 2, "x1": x1, "y1": y1, "x2": x2, "y2": y2}
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(frame, f"{class_name} {score:.2f}", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        detection_info += f"{class_name} ({score:.2f})\n"

    return detection_info
```

```python
def publish_detection(self):
    if self.detection_info:
        self.publisher_detection.publish(String(data=self.detection_info))
```

# DB와 AMR 수동 조작

## ROS 2 Object Detection Stream




### 감지 및 사라짐 이벤트

danger가 2024-12-02 15:00:43에 감지되었습니다.
danger가 2024-12-02 15:00:43에 사라졌습니다.

### 수동 조작

수동 조작 활성화

로그아웃

# DB와 AMR 수동 조작

```python
def create_database(self):
    """SQLite3 데이터베이스와 테이블 생성"""
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS danger_events (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            detect_time TEXT,
            disappear_time TEXT
        )
    """)
    conn.commit()
    conn.close()
```

```python
    # 'danger' 클래스 감지 시 처리
    if class_name == 'danger':
        detected_danger = True
        if not self.danger_detected:
            # 'danger'가 새로 감지된 경우
            self.danger_detected = True
            self.save_detect_time()
        break  # 'danger'를 이미 감지한 경우 추가 처리는 필요 없음

if not detected_danger and self.danger_detected:
    # 'danger'가 감지되지 않았지만 이전에 감지 상태였던 경우
    self.danger_detected = False
    self.save_disappear_time()
```

```python
def save_detect_time(self):
    """danger 감지 시간을 데이터베이스에 저장"""
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    detect_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    cursor.execute("""
        INSERT INTO danger_events (detect_time, disappear_time)
        VALUES (?, ?)
    """, (detect_time, None))
    self.last_danger_id = cursor.lastrowid
    conn.commit()
    conn.close()
    self.get_logger().info(f"danger 감지 시간 저장: {detect_time}")

def save_disappear_time(self):
    """danger 사라진 시간을 데이터베이스에 업데이트"""
    if self.last_danger_id is None:
        return  # 감지된 기록이 없으면 종료

    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    disappear_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    cursor.execute("""
        UPDATE danger_events
        SET disappear_time = ?
        WHERE id = ?
    """, (disappear_time, self.last_danger_id))
    conn.commit()
    conn.close()
    self.get_logger().info(f"danger 사라짐 시간 저장: {disappear_time}")
```

# DB와 AMR 수동 조작

```
// 1초마다 이벤트 정보를 업데이트
function fetchEvents() {
    fetch('/events')
        .then(response => response.json())
        .then(data => {
            const eventContainer = document.getElementById("event-container");
            let disappearMessage = data.disappear_time
                ? `danger가 ${data.disappear_time}에 사라졌습니다.`
                : "물체가 아직 존재합니다."; // disappear_time이 null일 경우 처리

            eventContainer.innerHTML = `
                <h2>감지 및 사라짐 이벤트</h2>
                <p>danger가 ${data.detect_time}에 감지되었습니다.</p>
                <p>${disappearMessage}</p>
            `;
        })
        .catch(error => console.error('Error fetching events:', error));
}

// 페이지 로딩 시, 이벤트 정보를 1초마다 업데이트
setInterval(fetchEvents, 1000);
```

```
sqlite> SELECT * FROM danger_events;
546|2024-12-02 14:35:20|2024-12-02 14:35:37
547|2024-12-02 14:35:37|2024-12-02 14:35:37
548|2024-12-02 14:35:37|2024-12-02 14:35:38
549|2024-12-02 14:35:38|2024-12-02 14:35:39
```

## 감지 및 사라짐 이벤트

danger가 2024-12-02 14:35:38에 감지되었습니다.

danger가 2024-12-02 14:35:39에 사라졌습니다.

# DB와 AMR 수동 조작

수동 조작

수동 조작 활성화

수동 조작

수동 조작 비활성화

```html
<div class="manual-control">
    <h2>수동 조작</h2>
    <button class="toggle-button" id="toggle-manual" onclick="toggleManualControl()">수동 조작 활성화</button>
</div>

<button onclick="logout()">로그아웃</button>

<script>
    let manualControlEnabled = false; // 수동 조작 상태

    // 수동 조작 상태를 전환하는 함수
    function toggleManualControl() {
        manualControlEnabled = !manualControlEnabled; // 상태 전환
        const toggleButton = document.getElementById("toggle-manual");

        if (manualControlEnabled) {
            toggleButton.classList.add("active");
            toggleButton.textContent = "수동 조작 비활성화";
        } else {
            toggleButton.classList.remove("active");
            toggleButton.textContent = "수동 조작 활성화";
        }
    }
```

# DB와 AMR 수동 조작

```javascript
// 수동 명령 전송 함수
function sendCommand(direction) {
    if (!manualControlEnabled) {
        alert("수동 조작이 활성화되어 있지 않습니다.");
        return;
    }

    fetch('/control', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ direction: direction })
    })
    .then(response => response.json())
    .catch(error => console.error('Error sending command:', error));
}
```

```javascript
// 키 이벤트 리스너
window.addEventListener('keydown', function(event) {
    if (!manualControlEnabled) return; // 수동 조작이 비활성화되어 있으면 명령을 보내지 않음

    switch (event.key) {
        case 'w':
            sendCommand('forward');  // W 키: 앞으로
            break;
        case 'x':
            sendCommand('backward'); // x 키: 뒤로
            break;
        case 'a':
            sendCommand('left');     // A 키: 왼쪽
            break;
        case 'd':
            sendCommand('right');    // D 키: 오른쪽
            break;
        case 's':
            sendCommand('stop');     // s 키: 스탑
            break;
        default:
            break;
    }
});
```

# DB와 AMR 수동 조작

```python
def send_command_via_ssh(command):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect('192.168.1.1', username='rokey1', password='rokey1234')  # 비밀번호 필요
    stdin, stdout, stderr = ssh.exec_command(command)
    output = stdout.read().decode()
    ssh.close()
    return output
```

```python
# ROS 2 노드를 별도의 스레드에서 실행하는 함수
def start_ros_node():
    """ROS 2 노드 실행 함수 - rclpy.spin() 사용"""
    global cmd_vel_publisher  # 전역 변수 사용
    rclpy.init()
    image_subscriber_node = ImageSubscriberNode()
    node = rclpy.create_node('manual_control_node')

    # cmd_vel_publisher를 ROS 노드 내에서 생성
    cmd_vel_publisher = node.create_publisher(Twist, '/cmd_vel', 10)

    try:
        rclpy.spin(image_subscriber_node)  # 이벤트 루프 실행
    except KeyboardInterrupt:
        pass
    finally:
        image_subscriber_node.destroy_node()  # 실행 후 노드 종료
        rclpy.shutdown()
```

```python
@app.route('/control', methods=['POST'])
def control():
    global cmd_vel_publisher
    data = request.get_json()  # 요청 본문에서 JSON 데이터 가져오기
    direction = data.get('direction')  # 방향 정보를 추출

    if direction:
        twist = Twist()

        # 방향에 따라 Twist 메시지 설정
        if direction == 'forward':
            twist.linear.x = 0.1  # 전진
        elif direction == 'backward':
            twist.linear.x = -0.1  # 후진
        elif direction == 'left':
            twist.angular.z = 0.2  # 왼쪽 회전
        elif direction == 'right':
            twist.angular.z = -0.2  # 오른쪽 회전
        elif direction == 'stop':
            twist.linear.x = 0.0  # 정지
            twist.angular.z = 0.0  # 회전 정지


        # cmd_vel 토픽에 메시지 발행
        cmd_vel_publisher.publish(twist)
        return jsonify({"message": f"Moved {direction}"}), 200

    return jsonify({"message": "Invalid direction or ROS publisher not initialized"}), 500
```

# 아기 따라다니기

```python
def control_robot(self):
    """아기의 위치와 움직임을 기반으로 로봇 제어."""
    if not self.baby_detected:
        self.move_stop()
        return

    x_center = self.baby_bounding_box['x_center']
    y_center = self.baby_bounding_box['y_center']
    frame_width = 640
    frame_height = 480

    if x_center < frame_width * 0.4:  # 아기가 왼쪽에 있음
        self.get_logger().info("왼쪽으로 회전합니다.")
        self.turn_left()
    elif x_center > frame_width * 0.6:  # 아기가 오른쪽에 있음
        self.get_logger().info("오른쪽으로 회전합니다.")
        self.turn_right()
    elif y_center > frame_height * 0.7:  # 너무 가까움
        self.get_logger().info("아기와 너무 가까워 로봇을 뒤로 갑니다.")
        self.move_backward()
    elif y_center < frame_height * 0.3:  # 너무 멂
        self.get_logger().info("아기와 너무 멀어 로봇이 접근합니다.")
        self.move_forward()
    else:
        self.get_logger().info("로봇이 정지합니다.")
        self.move_stop()
```

# 위험 물체와 아기 사이 가로 막기

up_camera.py

```python
def publish_danger_area(self):
    if self.baby_detected:
        if self.danger_detected:
            transformed_baby_pose = self.transform_to_map_coordinates(self.baby_pose)
            transformed_danger_pose = self.transform_to_map_coordinates(self.danger_pose)
            middle_x = (transformed_baby_pose[0] + transformed_danger_pose[0]) / 2
            middle_y = (transformed_baby_pose[1] + transformed_danger_pose[1]) / 2
            middle_point = Point(x=middle_x, y=middle_y)
            self.publisher_danger_area.publish(String(data="3"))
            self.publisher_trans_coor.publish(middle_point)
        else:
            y_value = self.baby_pose.y
            if y_value < 180:
                status = "1"
            elif y_value > 400:
                status = "2"
            else:
                status = "0"
            self.publisher_danger_area.publish(String(data=status))
    else:
        self.get_logger().info("No baby detected.")
        self.publisher_danger_area.publish(String(data="10"))
```

move.py

```python
def baby_y_callback(self, msg):
    baby_y_value = msg.data  # String 메시지에서 data를 사용
    self.get_logger().info(f'Received baby_y: {baby_y_value}')
    # 이전 값과 다른 경우 목표 취소 및 새 작업 처리
    self.get_logger().info(f'Value changed from {self.previous_baby_y} to {baby_y_value}')
    self.cancel_goal()  # 현재 진행 중인 목표 취소

    # 새로운 baby_y 값에 따른 동작 수행
    if baby_y_value == "10":
        self.send_multiple_goals()  # 다중 목표 전송
    elif baby_y_value == "2":
        self.send_goal(-1.24, -0.02)
        self.is_sending_goals = False
    elif baby_y_value == "1":
        self.send_goal(0.1, -0.629)
        self.is_sending_goals = False
    elif baby_y_value == "3":
        #값이 들어온 경우
        if self.status.x != 0 or self.status.y != 0:
            self.send_goal(self.status.x, self.status.y)
            self.is_sending_goals = False
    else:
        self.cancel_goal()
        self.is_sending_goals = False

    # 이전 값을 현재 값으로 갱신
    self.previous_baby_y = baby_y_value
```

# 위험 물체와 아기 사이 가로 막기

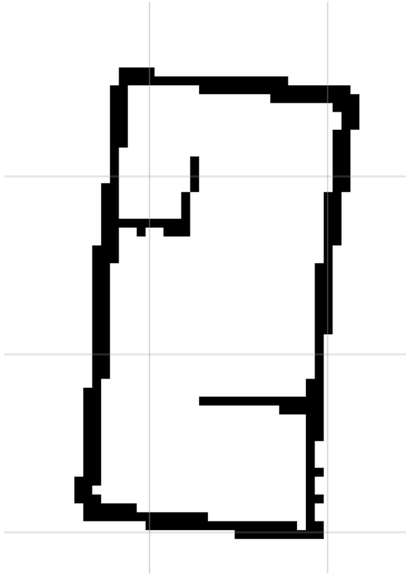
카메라를 통한 물체 인식

```
self.model = YOLO("up_best2.pt")  # YOLOv8 모델 파일 경로
```

변환 행렬 생성

```
self.src_points = np.float32([[27, 238], [604, 199], [56, 428], [522, 408]])
self.dst_points = np.float32([[-0.388, 0.116], [-0.375, -0.892], [-0.945, -0.0156], [-1.03, -0.71]])
self.matrix = cv2.getPerspectiveTransform(self.src_points, self.dst_points)
```

```
def transform_to_map_coordinates(self, pose):
    src_point = np.array([[[pose.x, pose.y]]], dtype=np.float32)
    transformed_point = cv2.perspectiveTransform(src_point, self.matrix)
    return transformed_point[0][0]
```



변환행렬을 통한 아기와 위험 물질 사이의 중간 값 계산 및 중간 지점와 상태 값 퍼블리시

```
if self.baby_detected:
    if self.danger_detected:
        transformed_baby_pose = self.transform_to_map_coordinates(self.baby_pose)
        transformed_danger_pose = self.transform_to_map_coordinates(self.danger_pose)
        middle_x = (transformed_baby_pose[0] + transformed_danger_pose[0]) / 2
        middle_y = (transformed_baby_pose[1] + transformed_danger_pose[1]) / 2
        middle_point = Point(x=middle_x, y=middle_y)
        self.publisher_danger_area.publish(String(data="3"))
        self.publisher_trans_coor.publish(middle_point)
```

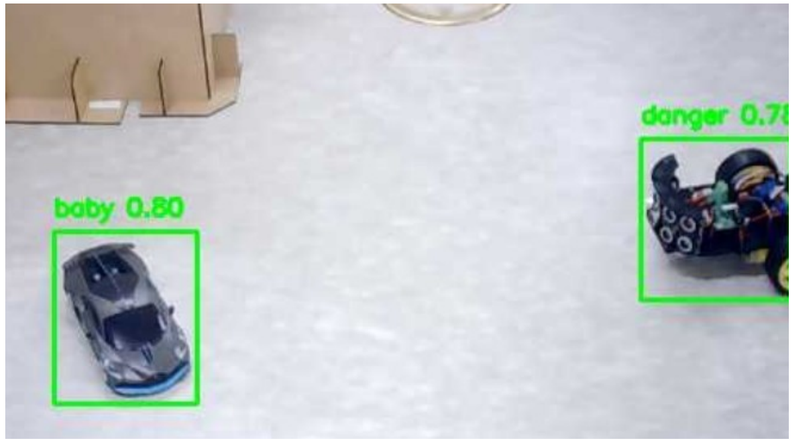
# 위험 물체와 아기 사이 가로 막기

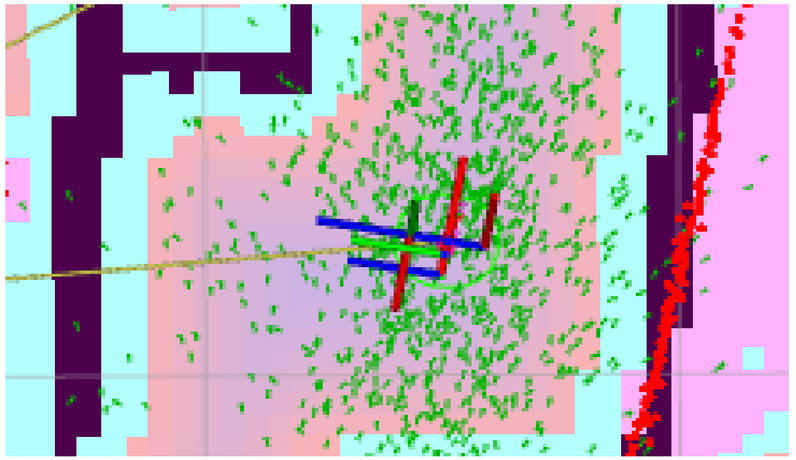
상태값 수신 후 수신한 값을 send_goal시킴

```python
def danger_pos_callback(self, msg):
    self.status = msg  # danger_pos 업데이트
elif baby_y_value == "3":
    #값이 들어온 경우
    if self.status.x != 0 or self.status.y != 0:
        self.send_goal(self.status.x, self.status.y)
        self.is_sending_goals = False
```

```python
def send_goal(self, x, y, yaw=0.0):
    # 새로운 PoseStamped 목표 생성
    waypoint = PoseStamped()
    waypoint.header.frame_id = 'map'
    waypoint.header.stamp = self.get_clock().now().to_msg()

    # 웨이포인트의 위치 설정
    waypoint.pose.position.x = x
    waypoint.pose.position.y = y
    waypoint.pose.position.z = 0.0  # 2D 내비게이션을 가정

    # 웨이포인트의 방향 설정 (예시로 회전하지 않도록 설정)
    waypoint.pose.orientation = Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)

    # NavigateToPose 액션의 목표 메시지 생성
    goal_msg = NavigateToPose.Goal()
    goal_msg.pose = waypoint  # 단일 목표 전송

    # 목표를 비동기적으로 전송
    self._send_goal_future = self.action_client_nav.send_goal_async(goal_msg)
    self._send_goal_future.add_done_callback(self.goal_response_callback)
```

시뮬레이션 결과



# 안전 구역 이탈 아기 가로 막기

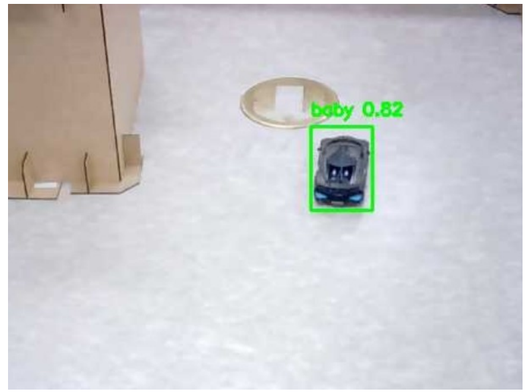
카메라

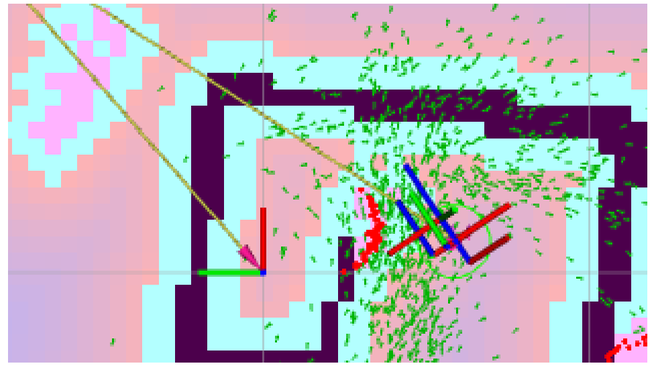
```
y_value = self.baby_pose.y
if y_value < 180:
    status = "1"
elif y_value > 400:
    status = "2"
else:
    status = "0"
self.publisher_danger_area.publish(String(data=status))
```

동작 코드

```
# 새로운 baby_y 값에 따른 동작 수행
if baby_y_value == "10":
    self.send_multiple_goals()  # 다중 목표 전송
elif baby_y_value == "2":
    self.send_goal(-1.24, -0.02)
    self.is_sending_goals = False
elif baby_y_value == "1":
    self.send_goal(0.1, -0.629)
    self.is_sending_goals = False
elif baby_y_value == "3":
    #값이 들어온 경우
    if self.status.x != 0 or self.status.y != 0:
        self.send_goal(self.status.x, self.status.y)
        self.is_sending_goals = False
else:
    self.cancel_goal()
    self.is_sending_goals = False
```

시뮬레이션 결과



# 안전 구역 이탈 아기 가로 막기

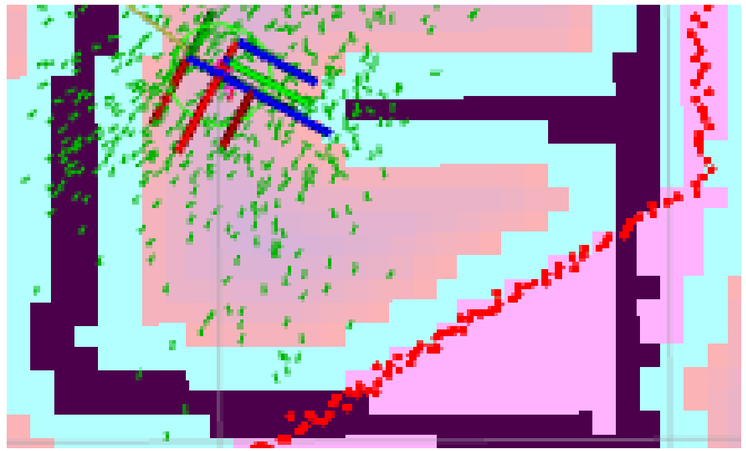
카메라

```
y_value = self.baby_pose.y
if y_value < 180:
    status = "1"
elif y_value > 400:
    status = "2"
else:
    status = "0"
self.publisher_danger_area.publish(String(data=status))
```

동작 코드

```
# 새로운 baby_y 값에 따른 동작 수행
if baby_y_value == "10":
    self.send_multiple_goals()  # 다중 목표 전송
elif baby_y_value == "2":
    self.send_goal(-1.24, -0.02)
    self.is_sending_goals = False
elif baby_y_value == "1":
    self.send_goal(0.1, -0.629)
    self.is_sending_goals = False
elif baby_y_value == "3":
    #값이 들어온 경우
    if self.status.x != 0 or self.status.y != 0:
        self.send_goal(self.status.x, self.status.y)
        self.is_sending_goals = False
else:
    self.cancel_goal()
    self.is_sending_goals = False
```

시뮬레이션 결과

# 정찰 모드(구현 실패)

publisher

```
if self.baby_detected:
    if self.danger_detected:
        transformed_baby_pose = self.transform_to_map_coordinates(self.baby_pose)
        transformed_danger_pose = self.transform_to_map_coordinates(self.danger_pose)
        middle_x = (transformed_baby_pose[0] + transformed_danger_pose[0]) / 2
        middle_y = (transformed_baby_pose[1] + transformed_danger_pose[1]) / 2
        middle_point = Point(x=middle_x, y=middle_y)
        self.publisher_danger_area.publish(String(data="3"))
        self.publisher_trans_coor.publish(middle_point)
    else:
        y_value = self.baby_pose.y
        if y_value < 180:
            status = "1"
        elif y_value > 400:
            status = "2"
        else:
            status = "0"
        self.publisher_danger_area.publish(String(data=status))
else:
    self.get_logger().info("No baby detected.")
    self.publisher_danger_area.publish(String(data="10"))
```


```
else:
    self.get_logger().info("No baby detected.")
    self.publisher_danger_area.publish(String(data="10"))
```

# 정찰 모드(구현 실패)

subscriber

```python
# 새로운 baby_y 값에 따른 동작 수행
if baby_y_value == "10":
    self.send_multiple_goals()  # 다중 목표 전송
```

코드를 제작할 때 수시로 값을 받도록 구성,

waypoint들을 follow하도록 하였지만,

어느 순간 이상으로는 로봇이 지정된 좌표를 받지 못하고,

0번과 1번 인덱스를 반복 이동함을 보였다.

```python
def send_multiple_goals(self):
    # Define waypoints for FollowWaypoints action
    waypoints = [
        (0.307, -0.0595),      # Corner1
        (0.244, -0.705),    # Corner2
        (-0.826, -0.592),   # Corner3
        (-0.507, -0.0219),  # Corner4
        (-1.51, 0.0282),    # Corner5
        (-1.62, -0.542)     # Corner6
    ]

    # 웨이포인트가 이미 전송 중이면 추가적으로 보내지 않음
    if self.is_sending_goals:
        self.get_logger().info("Goals are already being sent. Please wait until they are completed.")
        return

    # Create a list of PoseStamped messages for each waypoint
    poses = []
    for x, y in waypoints:
        waypoint = PoseStamped()
        waypoint.header.frame_id = 'map'
        waypoint.header.stamp = self.get_clock().now().to_msg()
        waypoint.pose.position.x = x
        waypoint.pose.position.y = y
        waypoint.pose.position.z = 0.0
        waypoint.pose.orientation = Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)  # No rotation for now
        poses.append(waypoint)

    # Create the goal message for FollowWaypoints action
    goal_msg = FollowWaypoints.Goal()
    goal_msg.poses = poses

    # Set flag to indicate we're sending multiple goals
    self.is_sending_goals = True

    # Wait for the action server to be available
    self.action_client_waypoints.wait_for_server()

    # Send the goal asynchronously
    self._send_goal_future = self.action_client_waypoints.send_goal_async(goal_msg)
    self._send_goal_future.add_done_callback(self.goal_response_callback)
```

# 프로젝트 발전 방향

1. 아기 따라다닐 때 장애물 회피

2. 다수의 아기 / 다수의 위험 물체

3. 사용자가 원하는 위험 물체 등록

# 배운점

1. 다수의 개발자 >> 다수의 의견

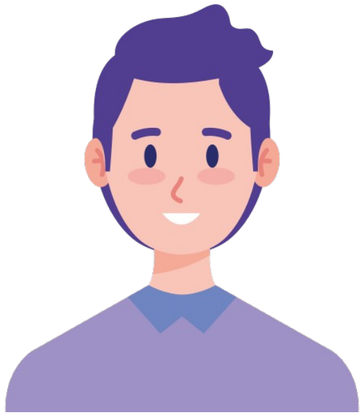
2. 프로젝트 시간 관리

3. 코딩은 중요하지 않다

# Team contribution

김우정

- 아기 따라가기
- 위험 상황 경고 및 로그 열람



복권근

- 아기의 상태에 따른 동작 구현
- flask 기반 설계

장세환

- Project Manager
- Flask 2 monitor
- 아기 따라가기

최현성

- Flask DB 연동
- AMR 수동 조작

# 프로젝트 시연

# Thank You

Boss Baby조