

ODK Fulfillment

-chapter 1: 개발자 사장님과 사원들-



오동규



김교희

박병주

김유찬

박영건



STAGE

1. 시나리오

2. GUI

3.튜닝전

4.문제점

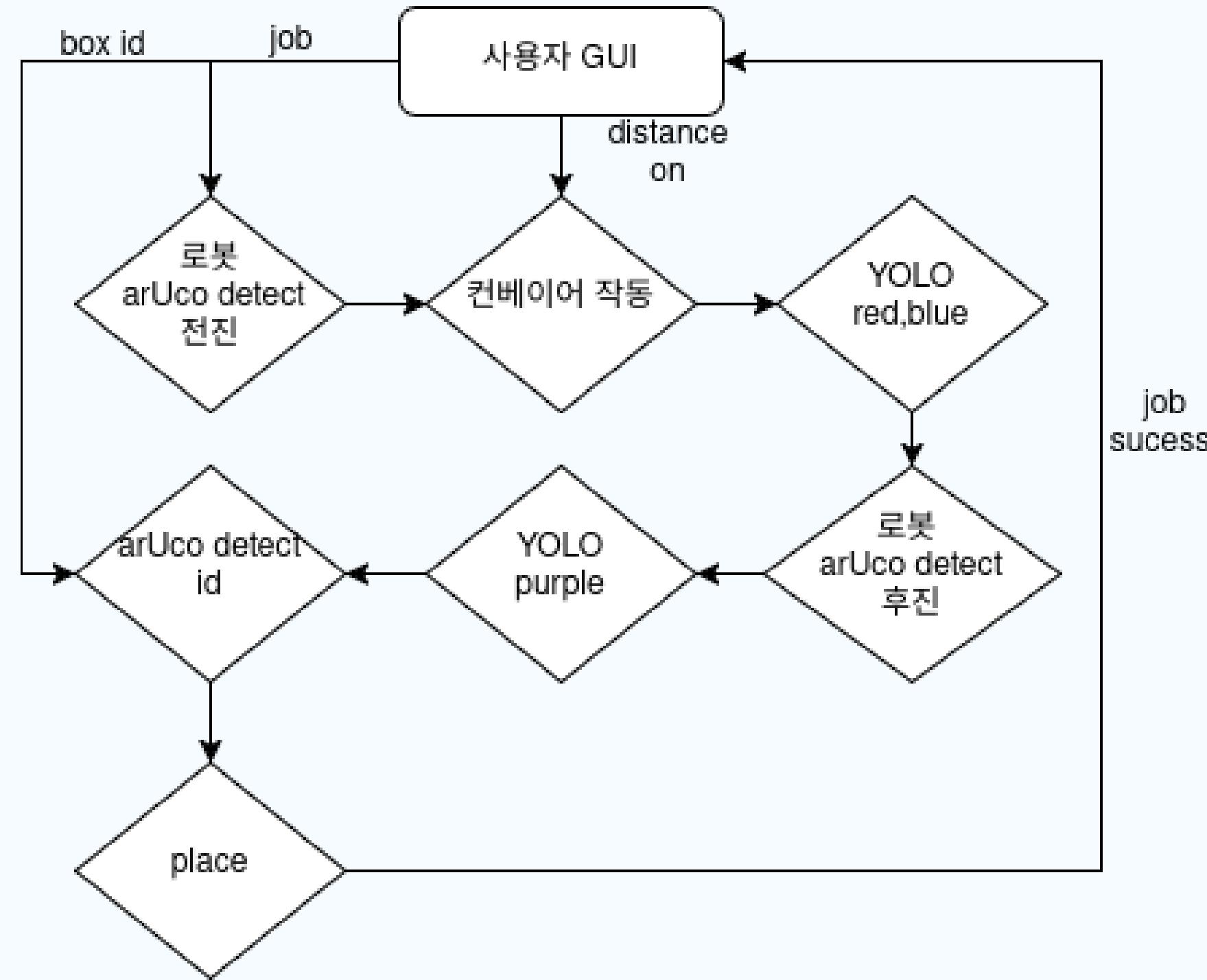
5. 코드 설명

6.YOLO

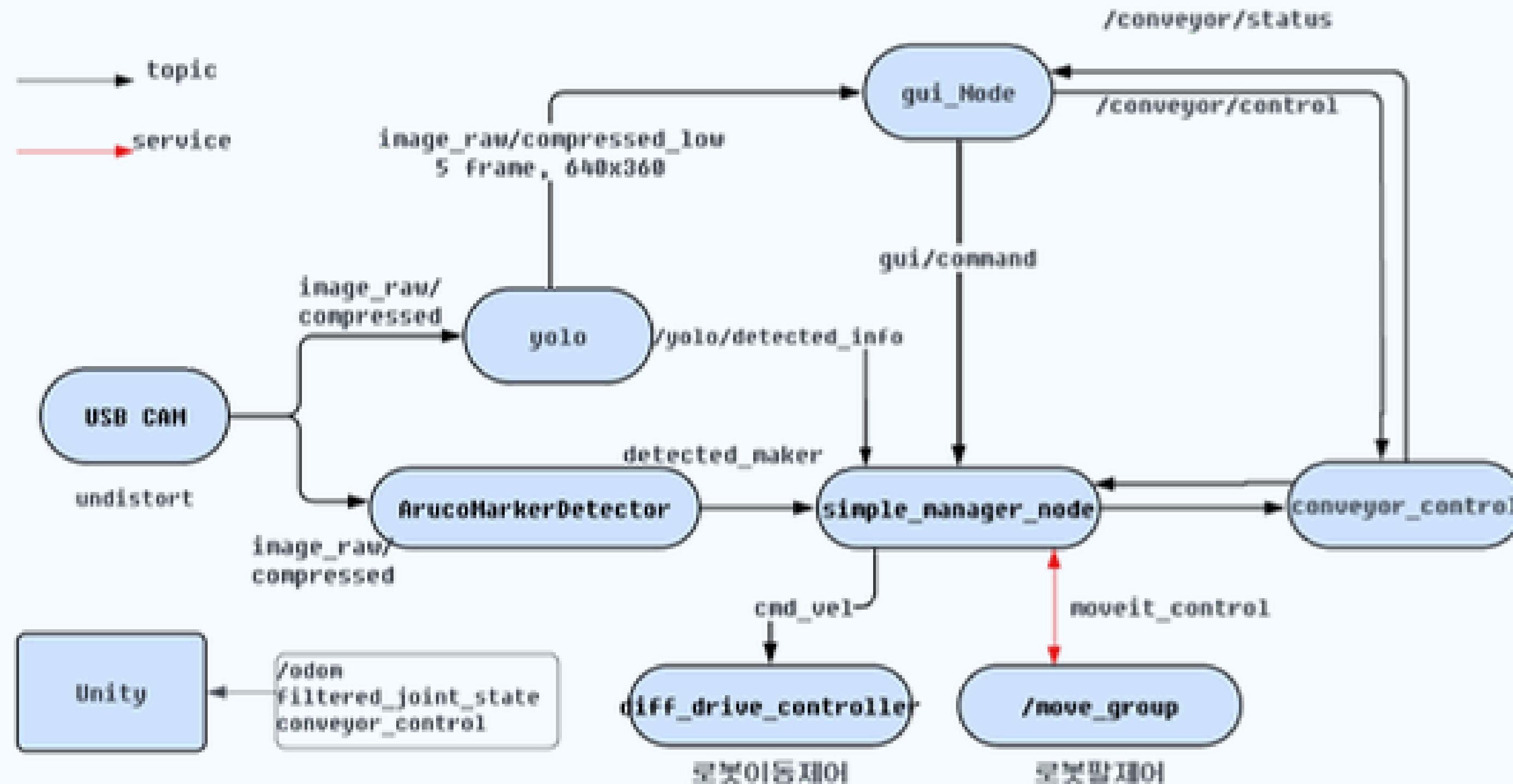
7.튜닝후

8. Unity

시나리오



노드구성



GUI

1



2

	pub	sub
1		image_raw/ compressed_low
2	conveyor/ control	conveyor/status
3	conveyor/ control gui/command	

딸
까

튜닝 전

joint

```
<group_state name="camera_home" group="arm">
    <joint name="joint1" value="0"/>
    <joint name="joint2" value="0.0698131700797731"/>
    <joint name="joint3" value="-0.5759586531581287"/>
    <joint name="joint4" value="2.0420352248333654"/>
</group_state>
```

```
<group_state name="box_back_put" group="arm">
    <joint name="joint1" value='1.583068173097' />
    <joint name="joint2" value="0.523599"/>
    <joint name="joint3" value="-0.698132"/>
    <joint name="joint4" value='0.785398'/>
</group_state>
```

basket to aruco

```
# 마지막 바구니 놓을 때, marker_id 받아오는걸로 수정 필요!!!
if self.state == 'CHECK':
    print(marker.id)
    marker_id = marker.id
    x_position = marker.pose.pose.position.x
    if marker_id == self.goal and abs(x_position) <= 0.05:
        print("find")
        self.publish_cmd_vel(0.0)
        self.final_task()
```

전진

```
def execute_forward_task(self, current_z_position):
    # 전진 작업: 30cm까지 전진 후 멈추고, 작업을 진행
    if self.aruco_marker_found and self.aruco_pose:
        self.get_logger().info("Executing forward task...")
        # 목표 z축 위치를 30cm로 설정
        if current_z_position > 0.3:
            self.publish_cmd_vel(0.05)
        elif current_z_position > 0.25:
            self.publish_cmd_vel(0.025)
```

후진

```
def execute_backward_task(self, current_z_position):
    # 후진 작업: 1m만큼 후진하고 다시 Aruco marker를 확인
    if self.aruco_marker_found and self.aruco_pose:
        self.get_logger().info("Executing backward task...")
        # 목표 z축 위치를 30cm로 설정
        if current_z_position < 0.98:
            self.publish_cmd_vel(-0.05)
```

yolo_arm_controll

```
pose_array = self.append_pose_init(0.137496 - self.yolo_y + 0.05, 0.00 - self.yolo_x ,0.122354)

response = arm_client.send_request(0, "", pose_array)
arm_client.get_logger().info(f'Response: {response.response}')

pose_array = self.append_pose_init(0.137496 - self.yolo_y + 0.05, 0.00 - self.yolo_x ,0.087354 )
```

purple_arm_control

```
pose_array = self.append_pose_init(0.0101294- self.yolo_x ,-0.2800000 ,0.205779 - self.yolo_y + 0.06)

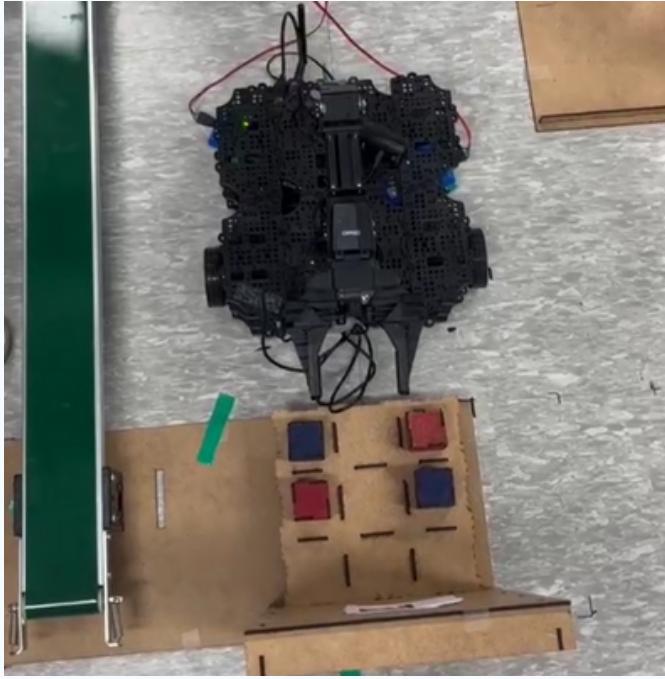
response = arm_client.send_request(3, "", pose_array)
arm_client.get_logger().info(f'Response: {response.response}')

response = arm_client.send_request(9, "")
arm_client.get_logger().info(f'Response: {response.response}')

pose_array = self.append_pose_init(0.0101294 - self.yolo_x,-0.3100000 ,0.205779 - self.yolo_y + 0.06 )
```

튜닝 전 - Picture

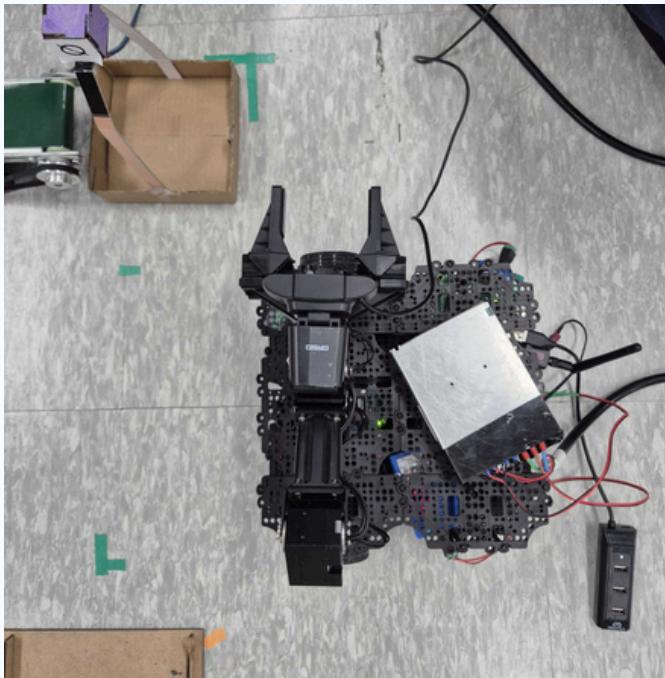
전진



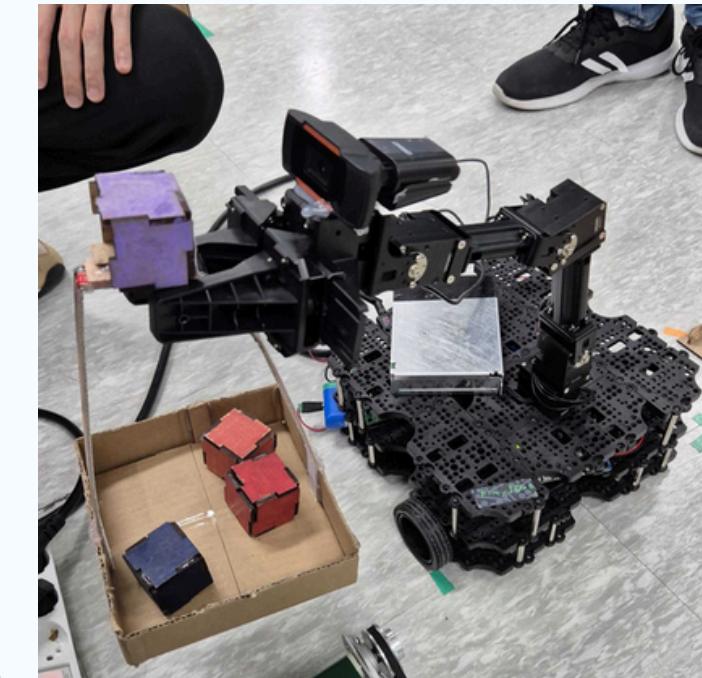
yolo_arm



후진



purple_arm



문제점

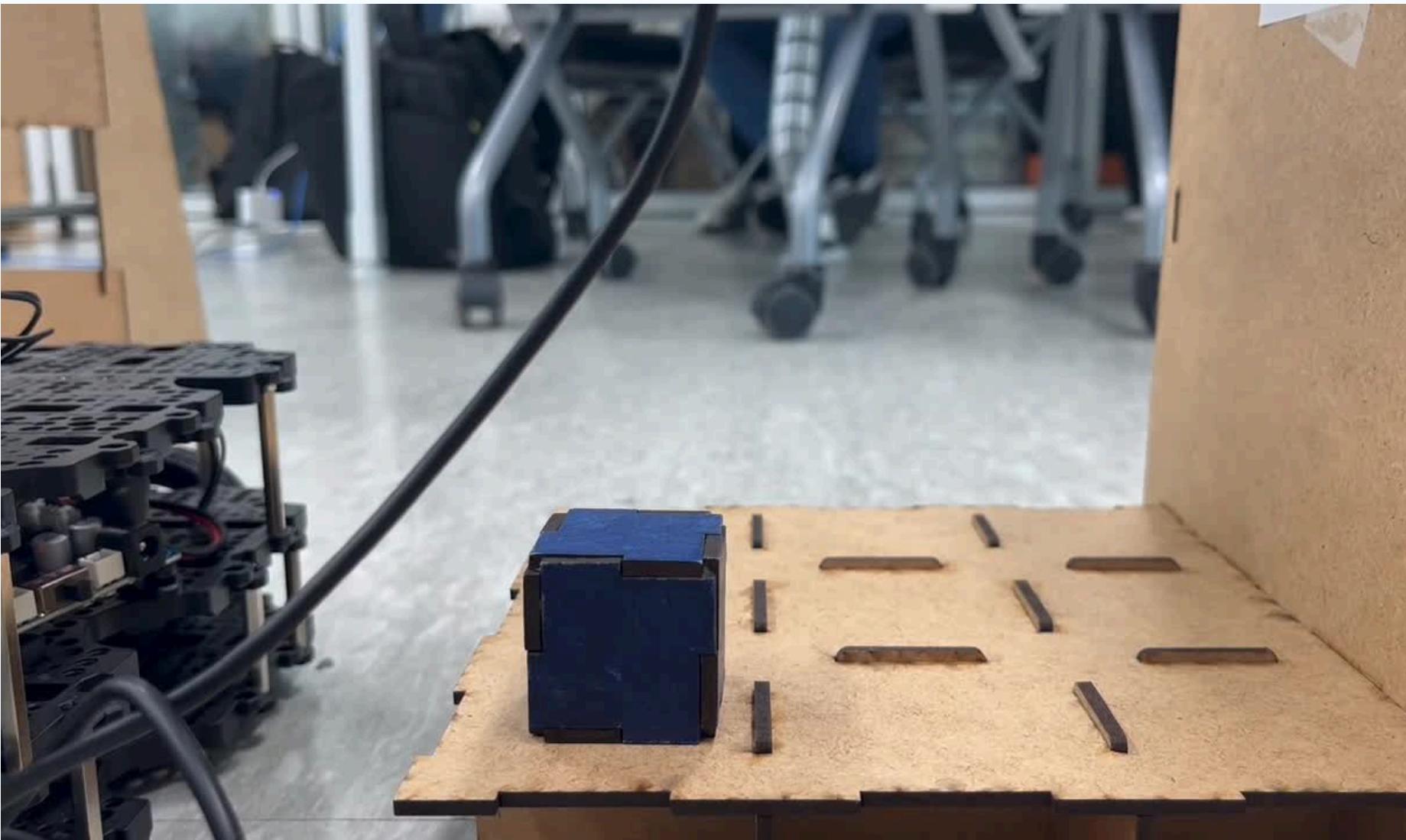
right_high로
이동 불가

right_high에서
위치 이탈

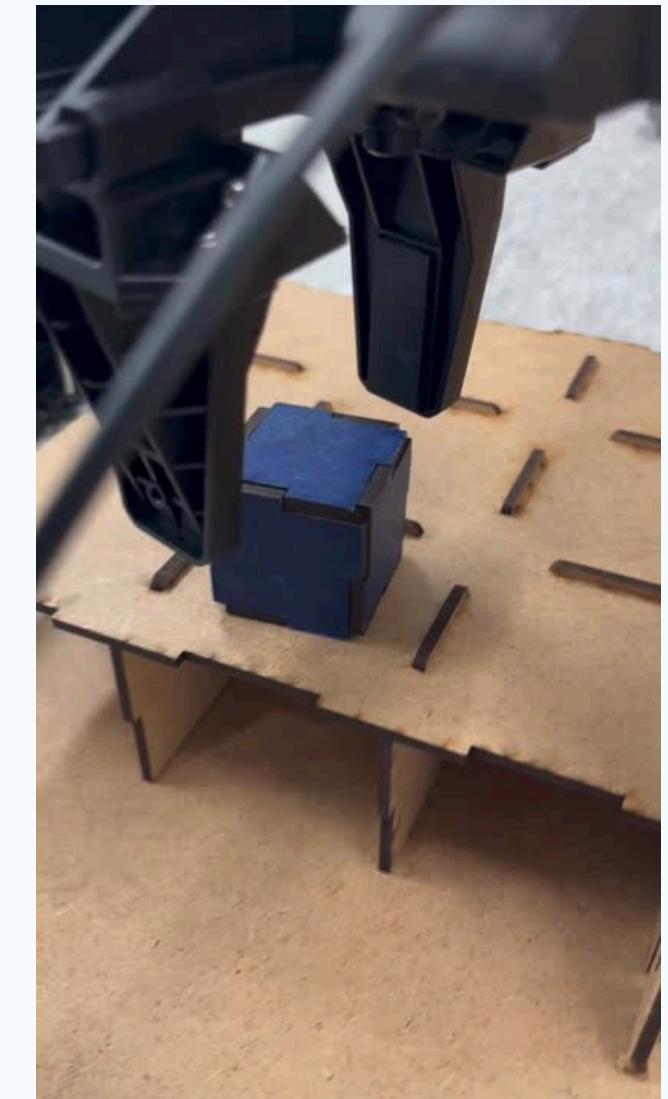
박스 위치
이중 접근

문제점 - Video

1. 미끄럼으로 인한 위치 이탈



2. 이중 접근



코드설명_1

gui_listener_callback()

- GUI에서 받은 데이터를 파싱하여 red, blue, goal등의 목표 설정
- red와 blue의 개수를 합산하여 전체 개수 저장
- run_tasks를 1초 간격으로 실행하는 타이머 생성

aruco_listener_callback()

- 현재 상태가 ARUCO, BACKWARD, CHECK일 때만 동작
- Aruco 마커를 찾고, 해당 마커의 ID와 위치 정보를 저장
- 현재 상태에 따라 특정 작업 수행
 - ARUCO: Aruco 마커의 Z 위치를 기반으로 전진 및 execute_forward_task호출
 - BACKWARD: 후진 및 컨베이어 작동, execute_backward_task호출
 - CHECK: 목표 마커에 도달했는지 확인하고 final_task실행

코드설명_2

yolo_listener_callback()

- 현재 상태가 YOLO 또는 PURPLE일 때만 동작
- YOLO에서 감지된 객체 데이터를 red/blue 박스 또는 purple 바구니의 위치 저장
- YOLO로 감지한 위치를 기반으로 박스/바구니 픽업 수행 및 yolo_arm_controll or purple_arm_control 호출
- self.count가 self.num과 같아지면, 로봇팔을 home2 위치로 이동 후 BACKWARD상태로 변경

execute_aruco_task()

- 상태를 ARUCO로 변경하여 Aruco 탐지를 시작

execute_forward_task()

- Aruco 마커까지 특정 거리 전진
- 특정 거리 도달 시 camera_arm_controll실행 후 YOLO 상태로 변경

execute_backward_task()

- 특정 거리까지 후진 후 box_home_arm_controll실행
- PURPLE 상태로 변경하여 바구니 탐지 시작

camera_arm_controll()

- 로봇팔을 camera_home 위치로 이동

코드설명_3

home2_arm_control()

- 로봇팔을 home2 위치로 이동

box_home_arm_control()

- 로봇팔을 box_home_01 위치로 이동

append_pose_init(x, y, z)

- 주어진 x, y, z 좌표로 PoseArray 생성 및 반환

yolo_arm_control()

- 로봇팔을 열어 박스 퍽업 준비
- YOLO로 탐지된 x, y 좌표를 기반으로 로봇팔 이동
- 박스를 집은 후 home2위치로 이동
- 컨베이어에 박스를 놓고 컨베이어 작동 명령 전송
- self.count 증가 및 yolofind 플래그 초기화

purple_arm_control()

- 컨베이어 정지
- YOLO 탐지된 바구니 위치로 이동
- 로봇팔을 열고 바구니 위치 조정 후 바구니 집기
- 바구니를 특정 위치로 옮긴 후 CHECK상태로 변경

final_task()

- 바구니를 box_back_put 위치로 이동 후 로봇팔 열기
- 상태를 FINISH로 변경

finish_task()

- FINISH 상태에서 노드를 종료하고 rclpy.shutdown()호출

publish_cmd_vel()

- linear_x, angular_z값을 기반으로 cmd_vel토픽에 메시지 발행하여 이동 명령 수행

run_tasks()

- 현재 상태에 따라 작업 실행 (START일 경우 execute_aruco_task실행, FINISH면 finish_task 실행)

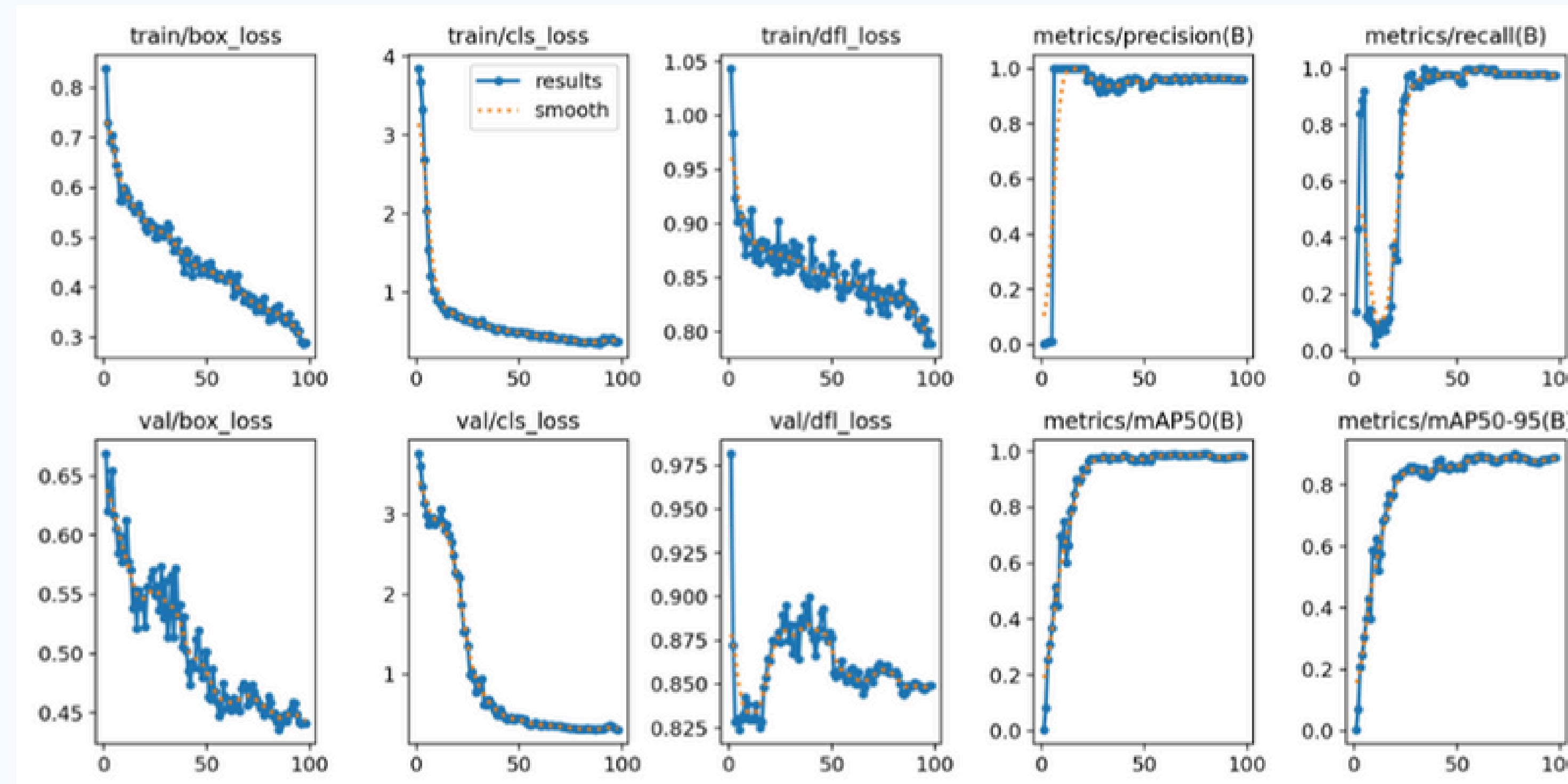
YOLO8



- red, blue, basket, aruco 클래스 별로 분류
- 조량, 주변 상황 등 다양한 환경을 고려하여 다양한 상황에서 사진 촬영 및 학습
- yolo에서 발견한 물체의 중심 좌표를 추출하여 로봇이동

그것이 Yolo 학습이니까'

YOLO8



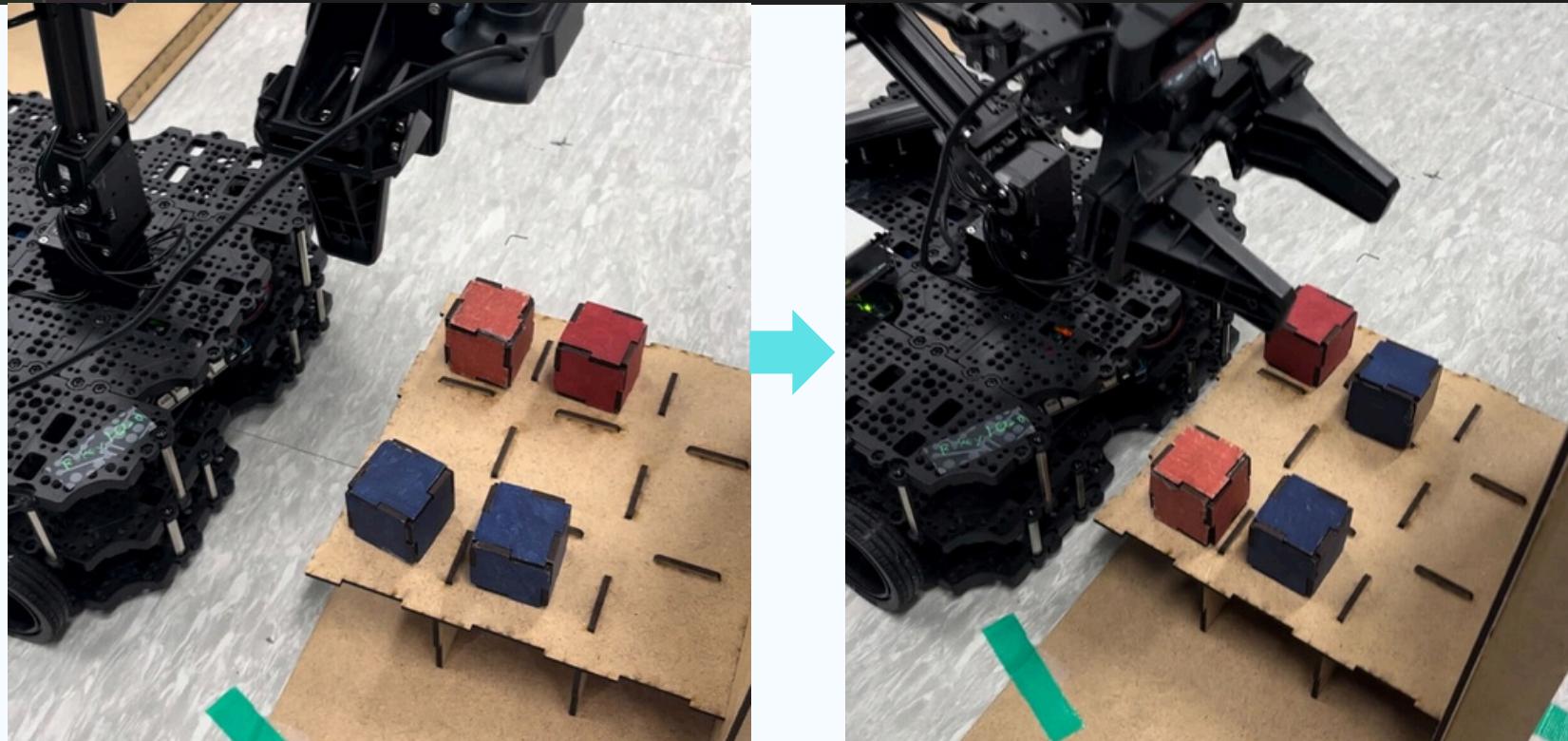
YOLO8



4. 튜닝 - 카메라-마커 거리

1 simple_manager_node.py

```
def execute_forward_task(self, current_z_position, current_x_position):
    # 전진 작업: 30cm까지 전진 후 멈추고 작업 진행
    if self.aruco_marker_found and self.aruco_pose and not self.cancel_tasks:
        self.get_logger().info("전진 작업 수행 중...")
    # 목표: z축 30cm 도달
    if current_z_position > 0.3:
        self.publish_cmd_vel(0.05)
    elif current_z_position > 0.162:
        # 아루코의 x값에 따라 보정 각속도 적용
        x = 0.0
```



튜닝전

- 21 aruco_move 거리 25.6

튜닝후

- 22 aruco_move에서 거리 줌힘
- 메니퓰레이터 좌표 보정

4. 튜닝 - SRDF

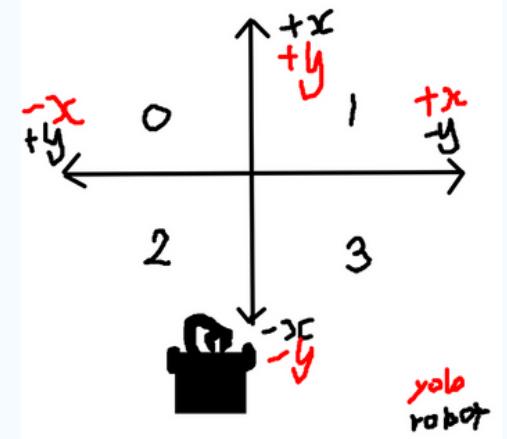
2 SRDF.

```
<group_state name="camera_home" group="arm">
  <joint name="joint1" value="0"/>
  <joint name="joint2" value="-0.3665191429188091"/>
  <joint name="joint3" value="-0.1570796326794896"/>
  <joint name="joint4" value="2.0420352248333654"/>
</group_state>
```

1.camera_home
: 카메라-box 각도

```
<group_state name="box_back_put" group="arm">
  <joint name="joint1" value="1.583068173097"/>
  <joint name="joint2" value="0.401425727958"/>
  <joint name="joint3" value="-0.663225115757"/>
  <joint name="joint4" value="0.209439510239"/>
</group state>
```

2. box_back_put
: 바스켓 내려 놓을 때 높이



4. 튜닝 - box pick 보정

3 simple_manager_node.py

```
elif self.yolo_x < 0 and self.yolo_y < 0:  
    self.yolo_x -= 0.000 # width += 오른쪽  
    self.yolo_y += 0.005 # hight -= 위쪽  
  
print("left high")
```

```
elif self.yolo_x > 0 and self.yolo_y < 0:  
    self.yolo_x += 0.01  
    self.yolo_y += 0.005  
  
print("right high")
```

```
if self.yolo_x < 0 and self.yolo_y > 0:  
    self.yolo_x -= 0.005 # width  
    self.yolo_y += 0.01 # hight  
  
print("left low")
```

```
if self.yolo_x > 0 and self.yolo_y > 0:  
    self.yolo_x += 0.01 # width  
    self.yolo_y += 0.025 # hight  
  
print("right low")
```

4. 튜닝

1. SRDF - 실제 길이와 모델링상 사이즈와의 차이

2. 모터(PID 컨트롤) - 스텝모터와 달리 정확한 각도 제어 불가

3. 환경 변수 - 먼지 등으로 바퀴의 미끌림 => 평행 이동 불가

4. 카메라 왜곡 효과 - 카메라 렌즈에 따라 다른 초점, 주점 등

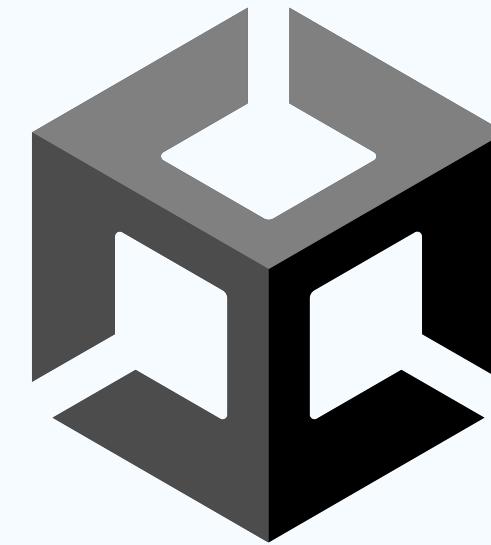


4. 투닝

- `camera_home` => Aruco인식을 위해 카메라 높이를 올리고 바닥과 수직이 되게 수정
- `forward_task` => aruco 마크와의 거리 수정
- `yolo_arm_controll` => 상자 좌표 보정
- `backward_task` => 바구니의 위치까지 후진을 위해 조정
- `purple_arm_controll` => 바구니의 실제 위치를 고려해 수정
- `basket2acuro` => 그리퍼 실제 크기 고려 최종 좌표 높이 삼승
- `box_back_put` => 바구니의 높이 고려



5.

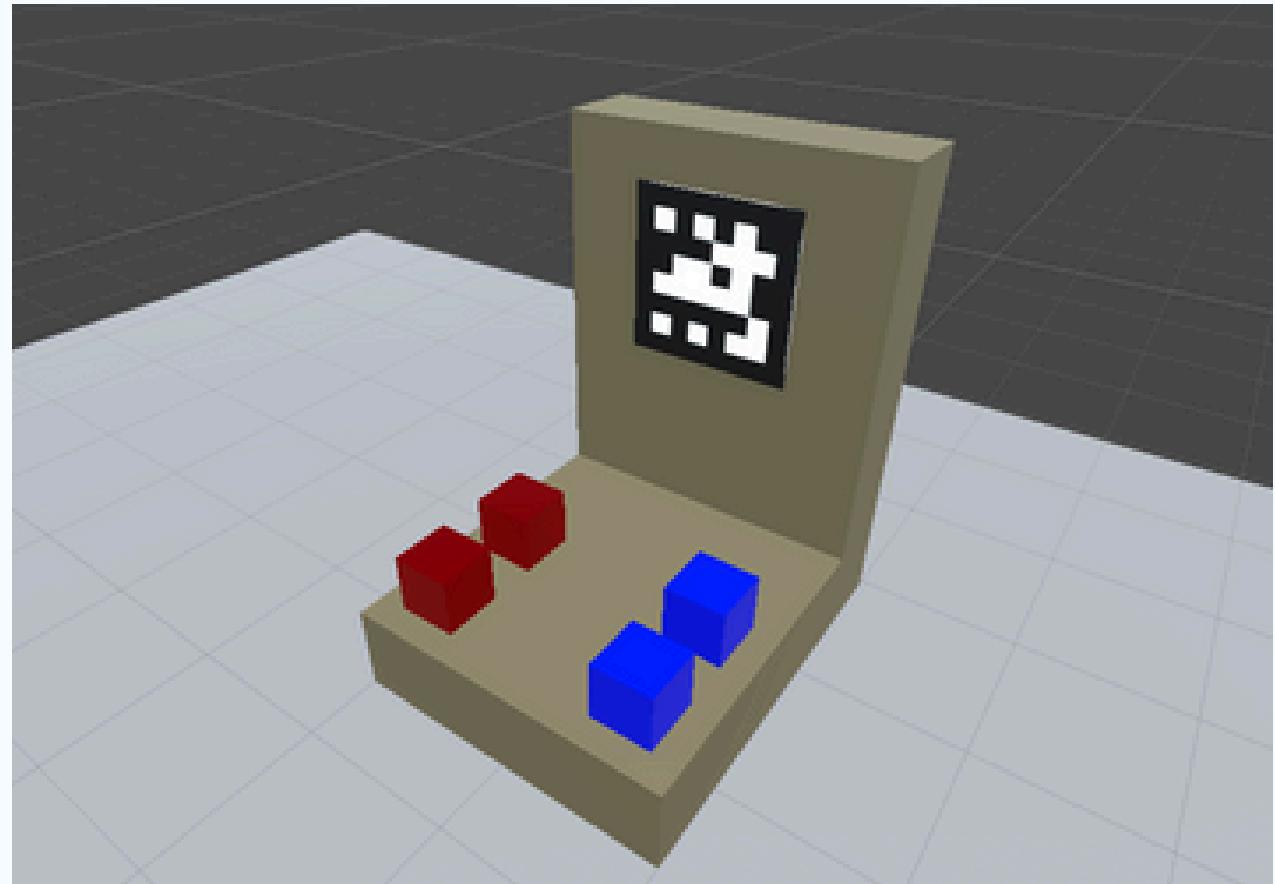


Unity®

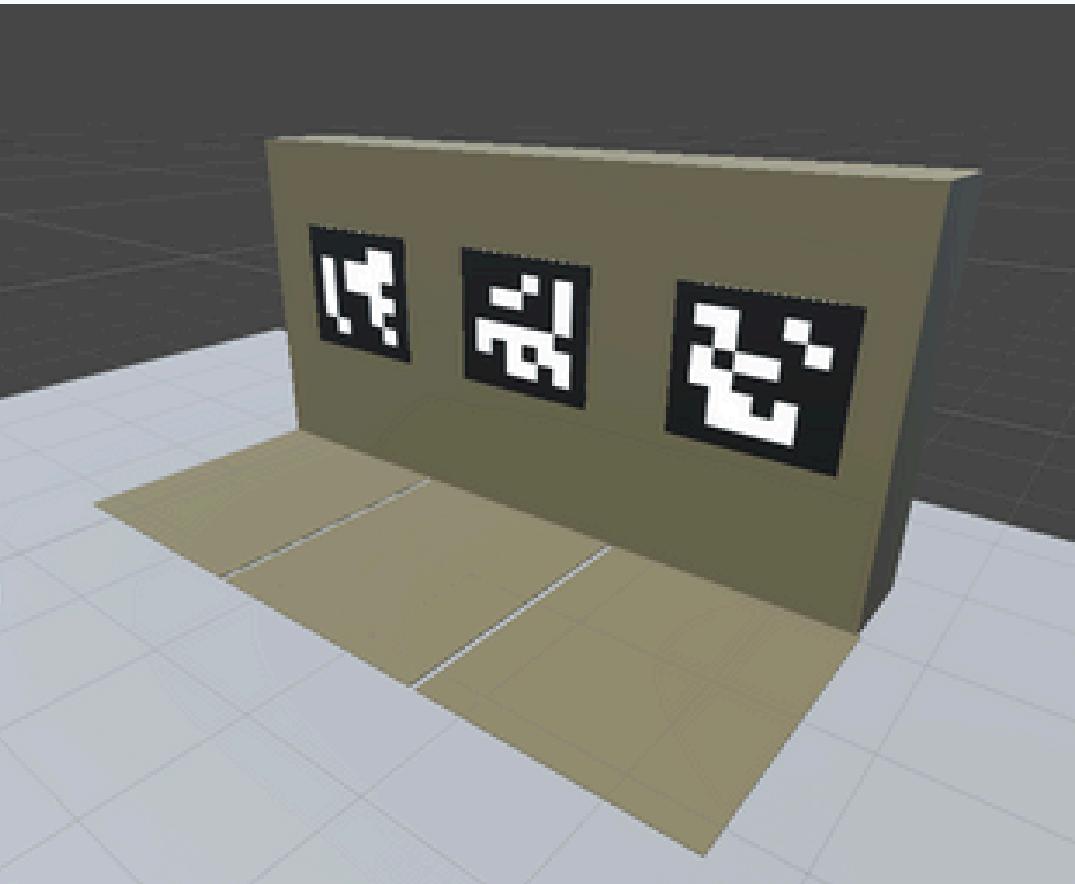
-Patch_note 0.0.2-



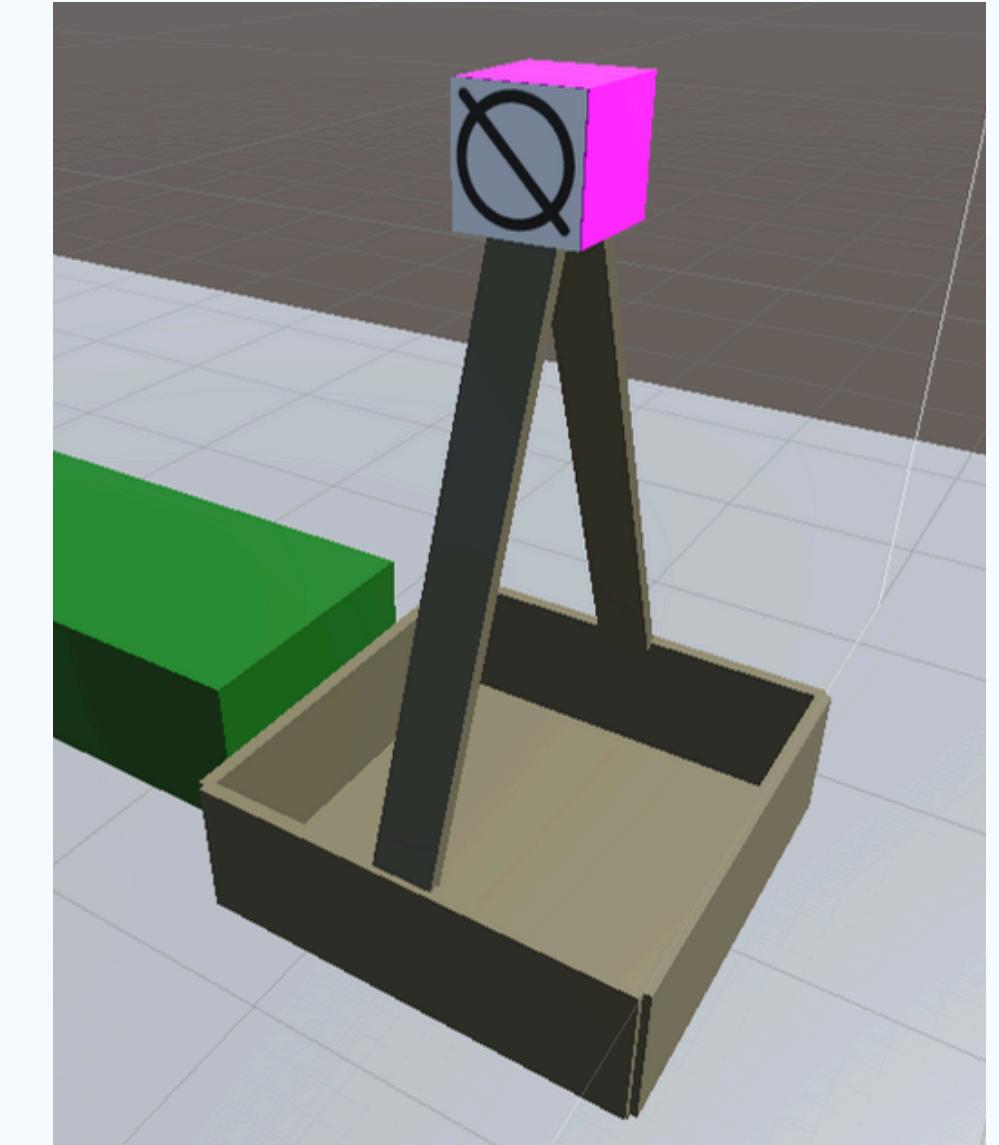
- Patch_1
 - Aurco_marker, Icon



Aurco_4



Aurco_1,
Aurco_2,
Aurco_3

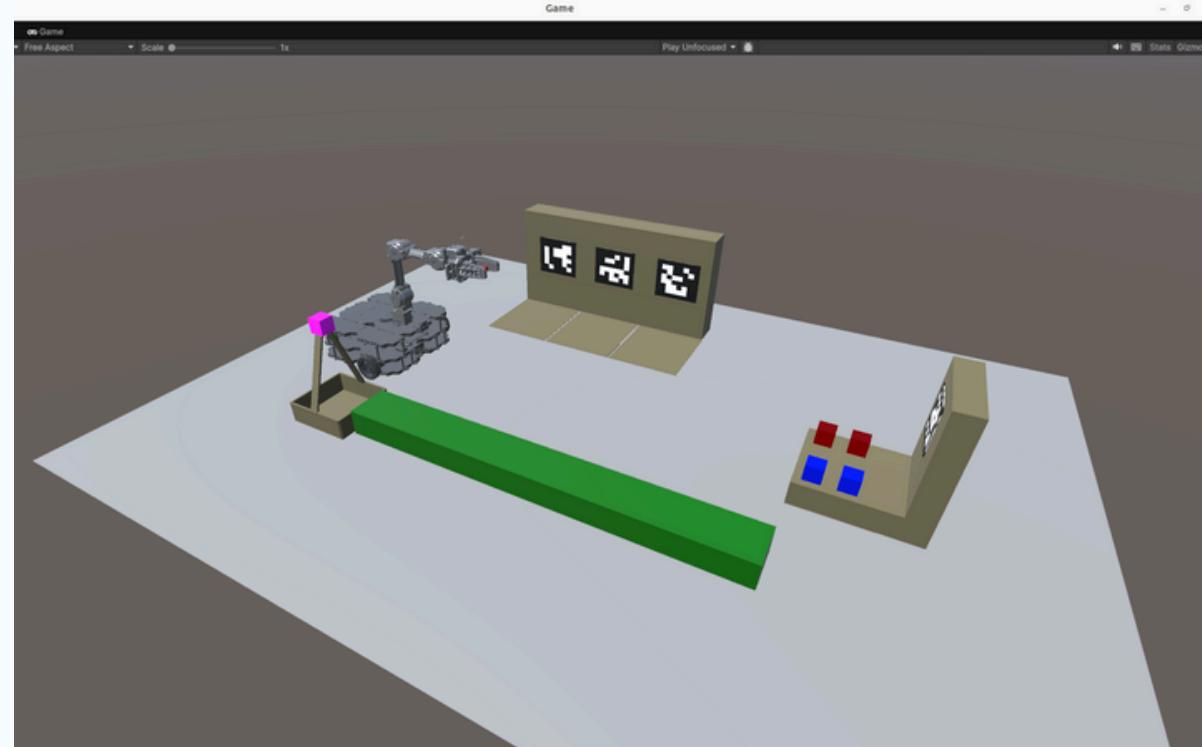


Icon

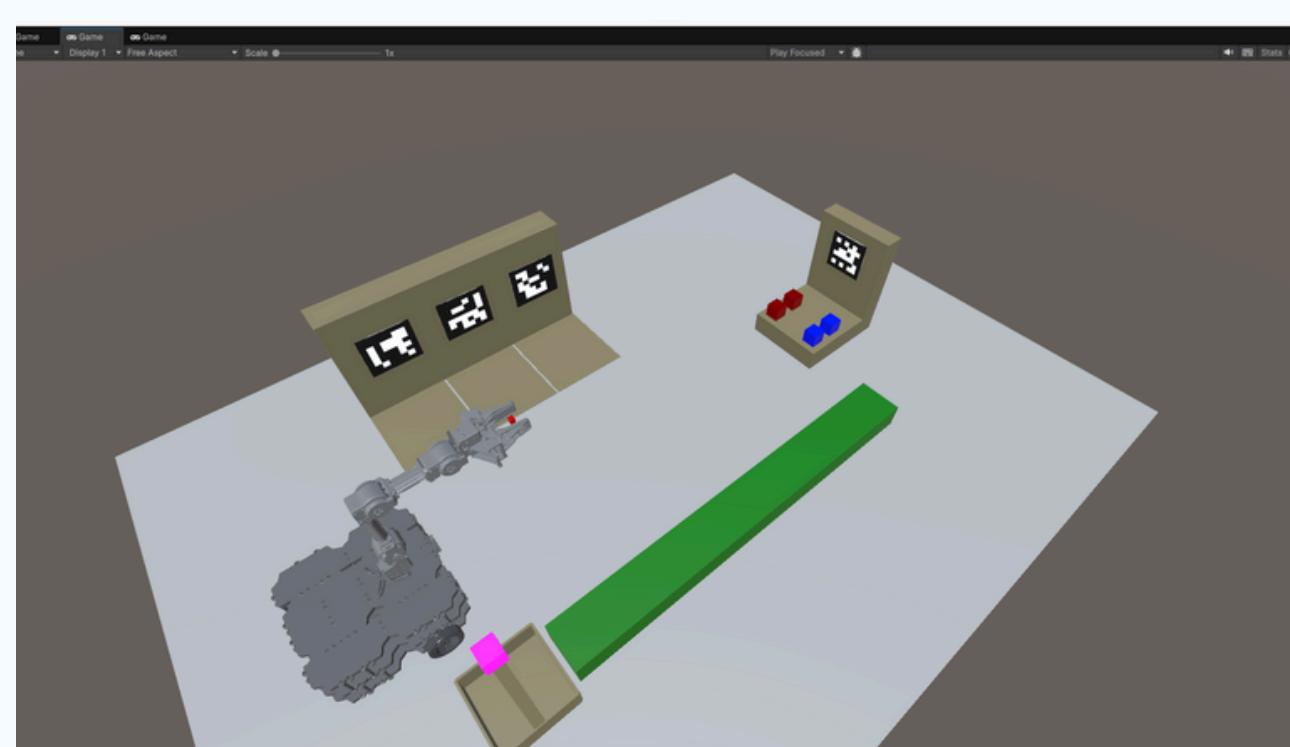
5. Unity®

- Patch_2
 - Camera_View_Point 추가

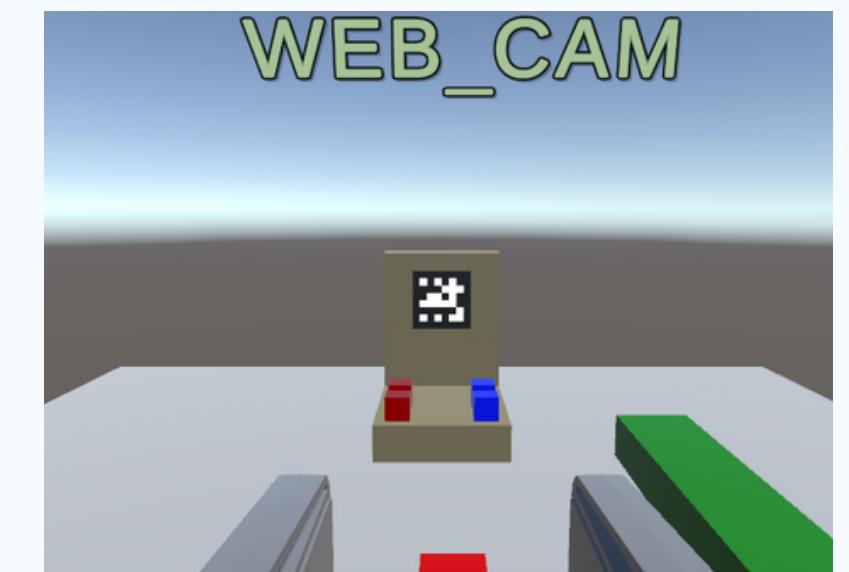
Main_Camera



Camera_2



Web_Cam_View





<- click here!



Thank You