TRIPLE-K COMPANY

환경 미화 로봇 시스템

• 김 강 경

- Object Detection Labeling & Learing
- Object Tracking Mechanism
- Local Navigation

• 오 건

- Turtlebot Modeling
- Manipulator Modeling
- ROS Confirm

• 전 수 현

- GUI Server
- Way Point Coordinate
- AMR Mechanism
- PPT

• 최 유 민

- Custom Map Design
- Custom Map Mapping
- PPT Confirm

Cleaning Bot

- 인건비 절감

  반복 업무를 대체해 인력 효율성을 높이고 인건비를 절감

- 환경 개선 효과

  로봇이 빠르고 정확하게 쓰레기를 수거해 깨끗한 공원과 거리 유지

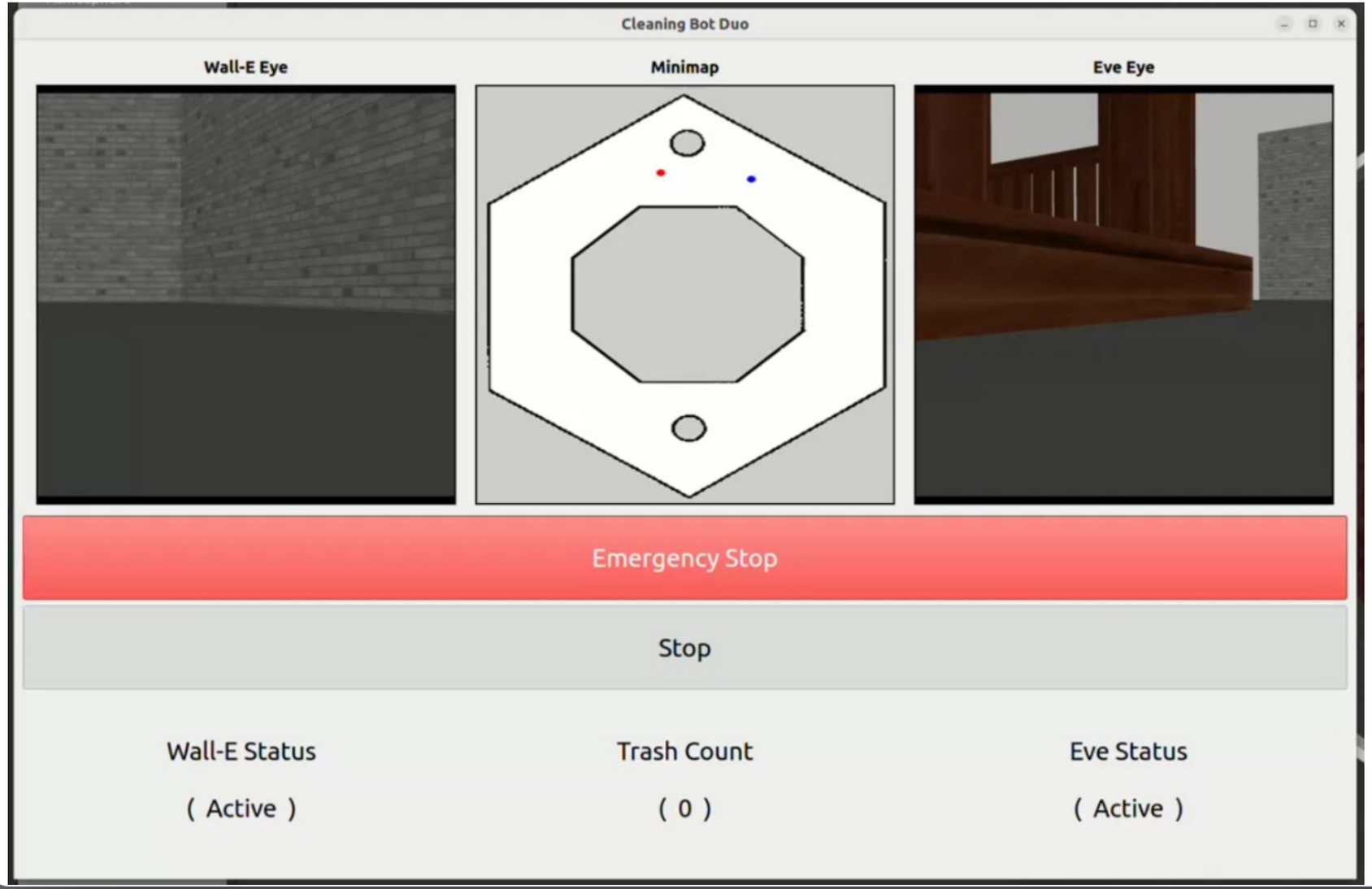
- 24시간 운영 가능

  로봇은 시간 제약 없이 작업 가능, 공공 환경의 지속적인 관리 실현

- 환경 보호 인식 개선

  로봇 활용으로 시민들에게 환경 보호와 청결의 중요성을 자연스럽게 알림

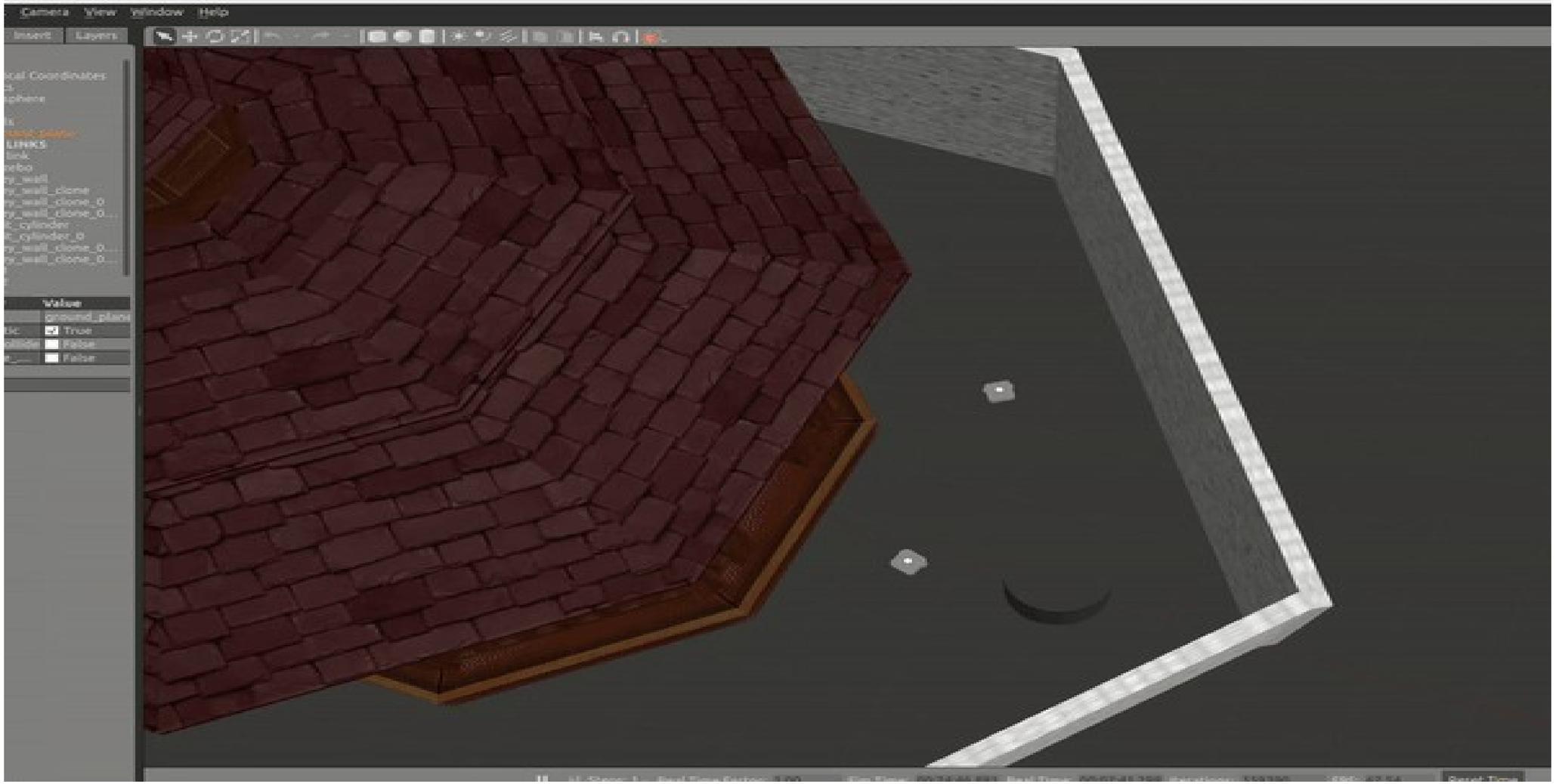
- GUI



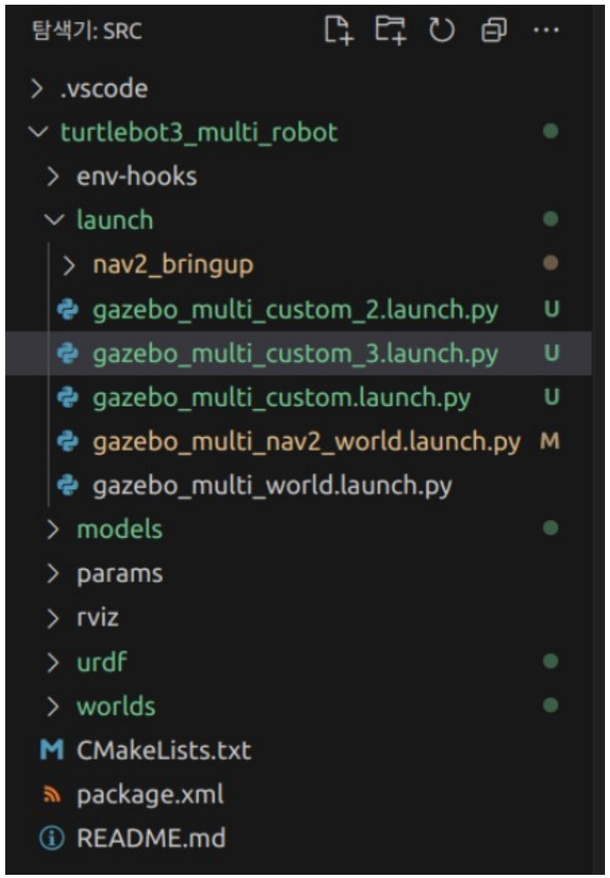
- Wall-E

- EVE

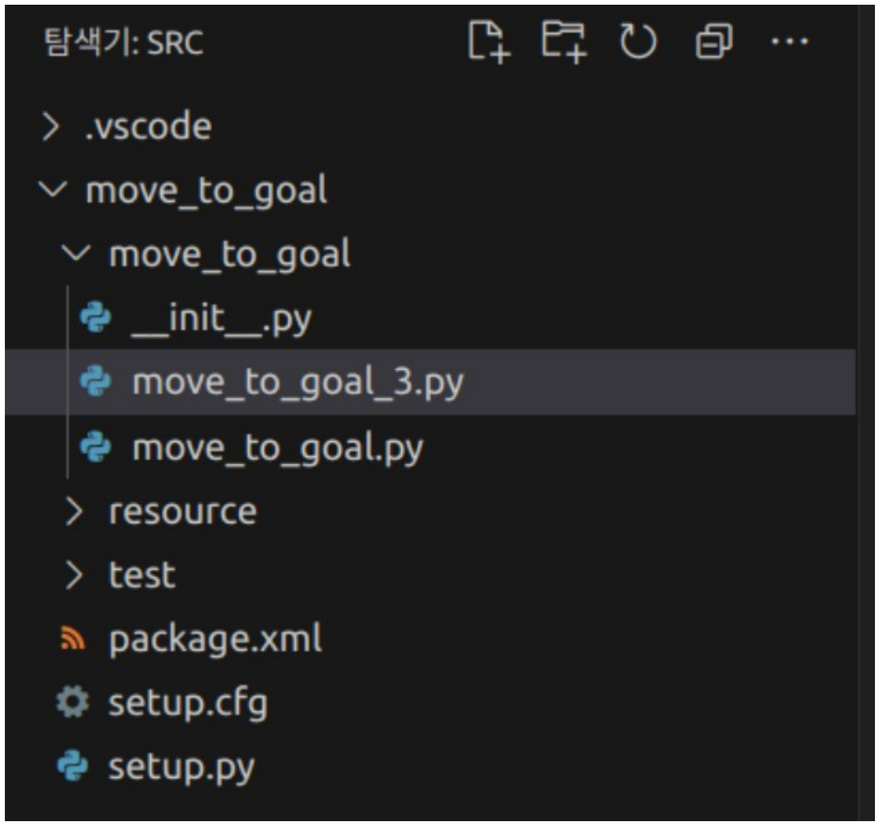
- Bot Tracking System



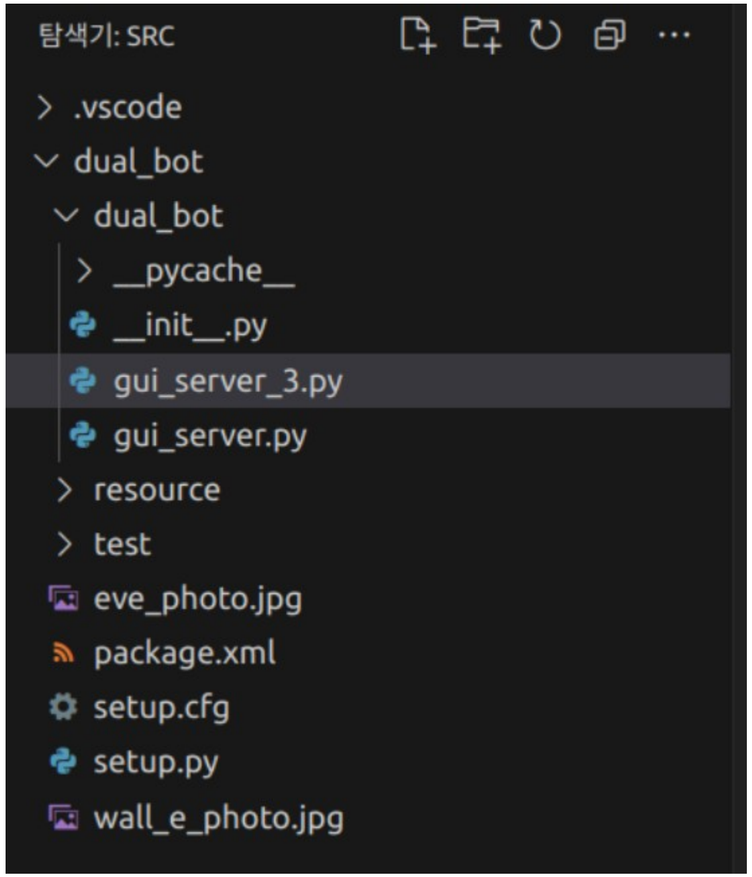
- turtlebot3_multi_robot (gazebo, rviz)



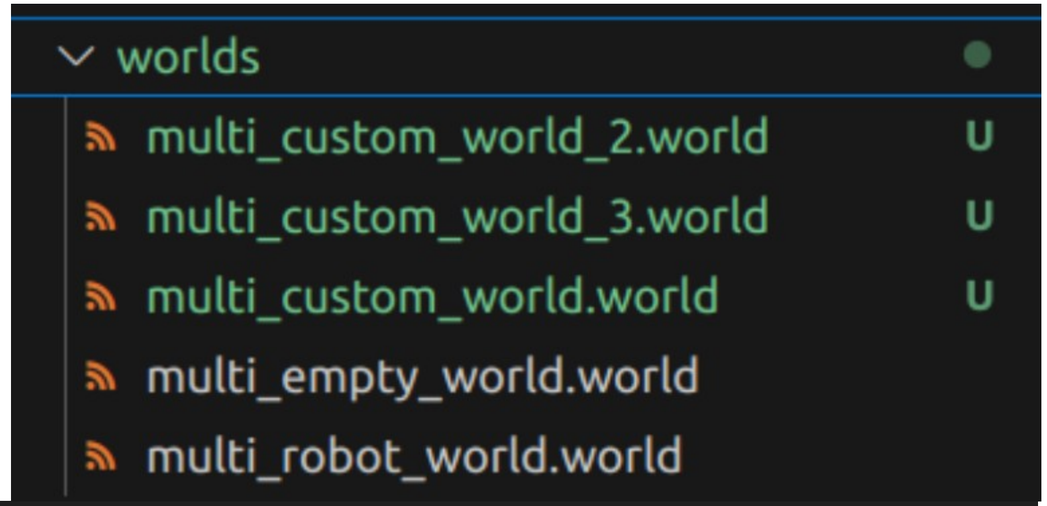
- move_to_goal (amr)



- dual_bot (gui)



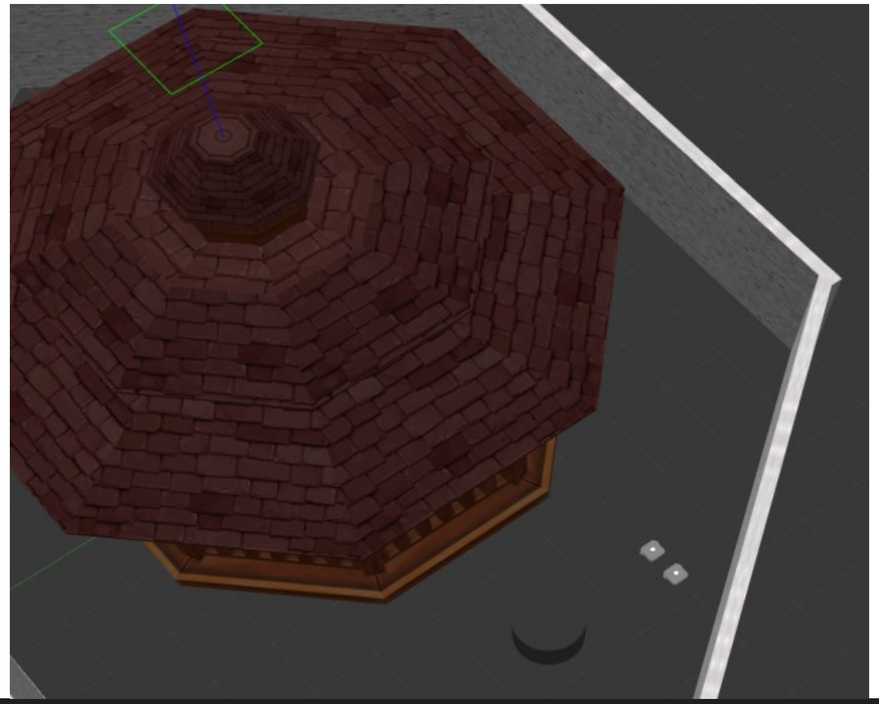
- turtlebot3_multi_robot (gazebo, rviz)

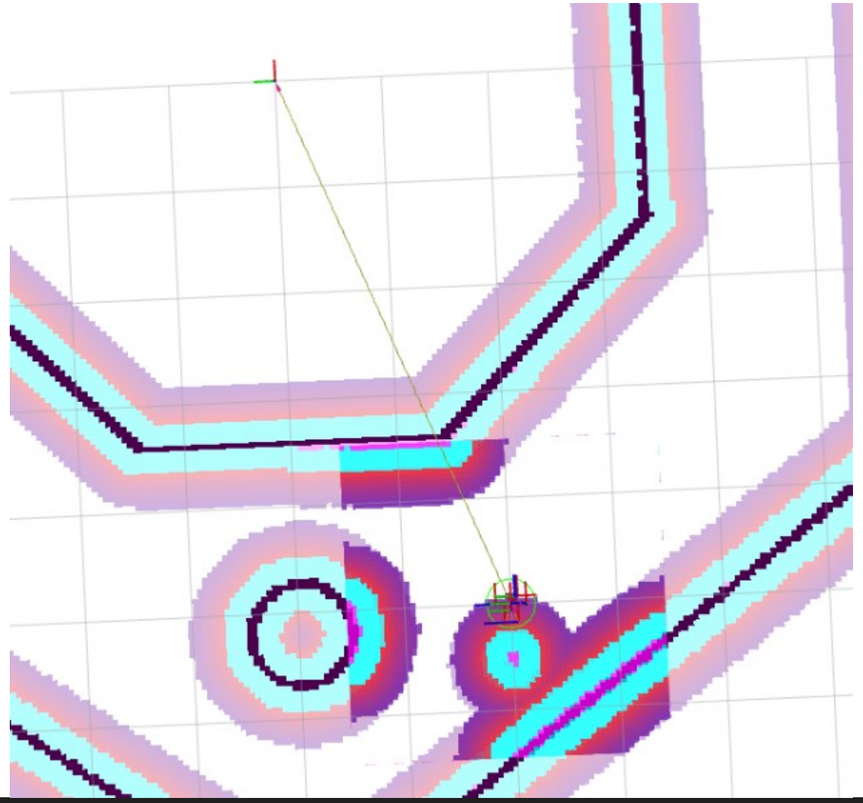



```
world = os.path.join(
    get_package_share_directory('turtlebot3_multi_robot'),
    'worlds', 'multi_custom_world_3.world')
```

```
map_server=Node(package='nav2_map_server',
    executable='map_server',
    name='map_server',
    output='screen',
    parameters=[{'yaml_filename': '/home/jsh/map3/map.yaml'},],
    remappings=remappings)
```

・turtlebot3_multi_robot (gazebo, rviz)

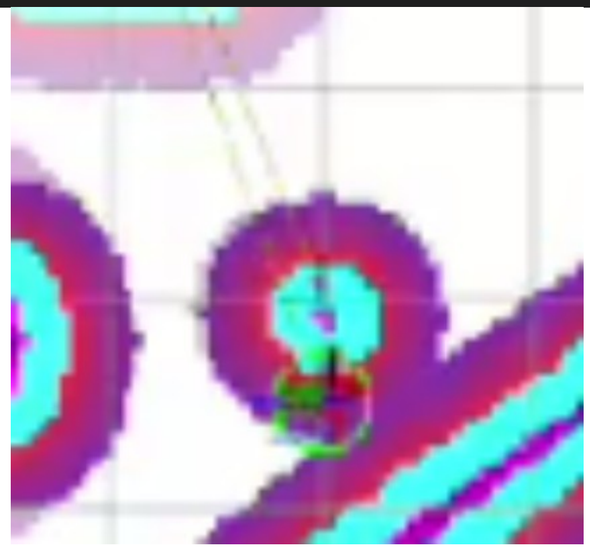




```
# Names and poses of the robots
robots = [
    {'name': 'tb1', 'x_pose': '-5.0', 'y_pose': '-2.0', 'z_pose': '0.01'},
    {'name': 'tb2', 'x_pose': '-5.5', 'y_pose': '-2.0', 'z_pose': '0.01'},
    ]
```

- turtlebot3_multi_robot (gazebo, rviz)

```
# Start rviz nodes and drive nodes after the last robot is spawned
for robot in robots:

    namespace = [ '/' + robot['name'] ]

    # Create a initial pose topic publish call
    if namespace[0] == '/tb1':
        message = '{header: {frame_id: map}, pose: {pose: {position: {x: 0.0, y: 0.0, z: 0.1}, orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}}, }
    else:
        message = '{header: {frame_id: map}, pose: {pose: {position: {x: -0.5, y: 0.0, z: 0.1}, orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}},
```



- move_to_goal (amr)

```
way_point_coordinates = [
    {"way_point_number": 1, "x": -0.0266, "y": -0.0388, "yaw": -0.785},
    {"way_point_number": 2, "x": 2.6459, "y": -2.4964, "yaw": -0.0},
    {"way_point_number": 3, "x": 7.6131, "y": -2.4085, "yaw": 0.785},
    {"way_point_number": 4, "x": 9.7074, "y": -0.0979, "yaw": 0.0},
    {"way_point_number": 5, "x": 9.6343, "y": 4.0883, "yaw": 2.355},
    {"way_point_number": 6, "x": 7.831, "y": 6.3066, "yaw": 3.14},
    {"way_point_number": 7, "x": 2.3753, "y": 6.4383, "yaw": -2.355},
    {"way_point_number": 8, "x": -0.0653, "y": 3.6761, "yaw": -1.57}
]
```

```
class MoveToGoal(Node):
    def __init__(self, namespace=''):
        super().__init__('move_to_goal_node')

        # 네임스페이스를 추가하여 NavigateToPose 액션 서버에 연결
        self._client = ActionClient(
            self,
            NavigateToPose,
            f'{namespace}/navigate_to_pose'  # 네임스페이스를 동적으로 할당
        )
        self.namespace = namespace
        self.current_goal_complete = False
        self.start_mission = False
        self.stop_requested = False
```

```
# 첫 번째 로봇 위치 추적 (tb1/amcl_pose)
if namespace == 'tb2':
    self.leader_pose_x = 0.0
    self.leader_pose_y = 0.0
    self.leader_yaw = 0.0
    self.create_subscription(
        PoseWithCovarianceStamped,
        '/tb1/amcl_pose',
        self.leader_pose_callback,
        10
    )
```

- move_to_goal (amr)

```python
# Start/Stop Toggle 구독
self.create_subscription(
    String,
    'start_stop_toggle',
    self.start_stop_callback,
    10
)

# Emergency Stop 구독
self.create_subscription(
    String,
    'emergency_stop',
    self.emergency_stop_callback,
    10
)
```

```python
def leader_pose_callback(self, msg):
    self.leader_pose_x = msg.pose.pose.position.x
    self.leader_pose_y = msg.pose.pose.position.y
    quaternion = msg.pose.pose.orientation
    self.leader_yaw = R.from_quat([
        quaternion.x, quaternion.y, quaternion.z, quaternion.w
    ]).as_euler('xyz')[2]
```

```python
# 좌표 설정
goal_msg.pose.pose.position.x = coordinates["x"]
goal_msg.pose.pose.position.y = coordinates["y"]

# Yaw를 기반으로 쿼터니언 계산
yaw = coordinates["yaw"]
quaternion = R.from_euler('z', yaw).as_quat()
goal_msg.pose.pose.orientation.x = quaternion[0]
goal_msg.pose.pose.orientation.y = quaternion[1]
goal_msg.pose.pose.orientation.z = quaternion[2]
goal_msg.pose.pose.orientation.w = quaternion[3]
```

- move_to_goal (amr)

```python
def send_goal(self, way_point_number=None, follow_tb1=False):
    # 목표 위치 설정
    goal_msg = NavigateToPose.Goal()
    goal_msg.pose.header.frame_id = 'map'
    goal_msg.pose.header.stamp = self.get_clock().now().to_msg()

    # Waypoint 좌표 가져오기
    if not follow_tb1:
        coordinates = next(
            (coord for coord in way_point_coordinates if coord["way_point_number"] == way_point_number), None
        )
    else:  # tb2 follows tb1
        coordinates = {
            "x": self.leader_pose_x - 0.75 * cos(self.leader_yaw),
            "y": self.leader_pose_y - 0.75 * sin(self.leader_yaw),
            "yaw": self.leader_yaw
        }
```

- move_to_goal (amr)

```python
def goal_response_callback(self, future, way_point_number):
    goal_handle = future.result()
    if not goal_handle.accepted:
        self.get_logger().warning(f"Goal for Way Point {way_point_number} was rejected.")
        return

    self.get_logger().warning(f"Goal for Way Point {way_point_number} was accepted.")
    result_future = goal_handle.get_result_async()
    result_future.add_done_callback(lambda f: self.goal_result_callback(f, way_point_number))

def cancel_goal_response(self, future):
    cancel_result = future.result()
    if cancel_result.goal_cancelled:
        self.get_logger().warning("Goal successfully cancelled.")
    else:
        self.get_logger().warning("Failed to cancel the goal.")

def goal_result_callback(self, future, way_point_number):
    result = future.result().result
    if result:
        self.get_logger().warning(f"Successfully arrived at Way Point {way_point_number}.")
    else:
        self.get_logger().warning(f"Failed to reach Way Point {way_point_number}.")
    self.current_goal_complete = True

def wait_for_goal_completion(self):
    while not self.current_goal_complete:
        rclpy.spin_once(self, timeout_sec=0.1)
```

- move_to_goal (amr)

```
move_to_goal_tb1 = MoveToGoal(namespace='tb1')
move_to_goal_tb2 = MoveToGoal(namespace='tb2')

try:
    while True:
        # Start 신호를 받을 때까지 대기
        move_to_goal_tb1.get_logger().info("Waiting for Start signal...")
        while not move_to_goal_tb1.start_mission:
            rclpy.spin_once(move_to_goal_tb1, timeout_sec=0.1)

        # 현재 tb1의 목표를 저장하여 중복 목표 전송 방지
        current_goal_tb1 = None

        # Waypoints 1번부터 8번까지 무한으로 반복
        while move_to_goal_tb1.start_mission:
            for way_point_number in [1, 2, 3, 4, 5, 6, 7, 8]:
                # tb1 목표 전송 (중복 전송 방지)
                if current_goal_tb1 != way_point_number:
                    move_to_goal_tb1.send_goal(way_point_number)
                    current_goal_tb1 = way_point_number
```

- move_to_goal (amr)

```
            # tb1이 목표로 이동하는 동안 tb2가 실시간으로 따라감
            while not move_to_goal_tb1.current_goal_complete:
                # tb2의 목표를 갱신하지 않고 tb1의 뒤를 따라가도록 설정
                move_to_goal_tb2.send_goal(follow_tb1=True)
                rclpy.spin_once(move_to_goal_tb1, timeout_sec=0.05)
                rclpy.spin_once(move_to_goal_tb2, timeout_sec=0.05)

            # Stop 신호를 받으면 남은 경로를 돌고 루프 종료
            if move_to_goal_tb1.stop_requested:
                break

        if move_to_goal_tb1.stop_requested:
            break

    # Stop 신호를 받으면 남은 경로를 수행 후 1번 Waypoint로 복귀
    current_index = [1, 2, 3, 4, 5, 6, 7, 8].index(current_goal_tb1)
    remaining_waypoints = [1, 2, 3, 4, 5, 6, 7, 8][current_index + 1:]

    for way_point_number in remaining_waypoints + [1]:
        move_to_goal_tb1.send_goal(way_point_number)  # tb1의 목표 전송

        # tb1이 목표로 이동하는 동안 tb2가 실시간으로 따라감
        while not move_to_goal_tb1.current_goal_complete:
            move_to_goal_tb2.send_goal(follow_tb1=True)  # tb1의 뒤를 따라 이동
            rclpy.spin_once(move_to_goal_tb1, timeout_sec=0.05)
            rclpy.spin_once(move_to_goal_tb2, timeout_sec=0.05)

    # Stop 신호를 받은 이후 루프 종료를 보장
    move_to_goal_tb1.start_mission = False
    move_to_goal_tb1.stop_requested = False
```

- dual_bot (gui)

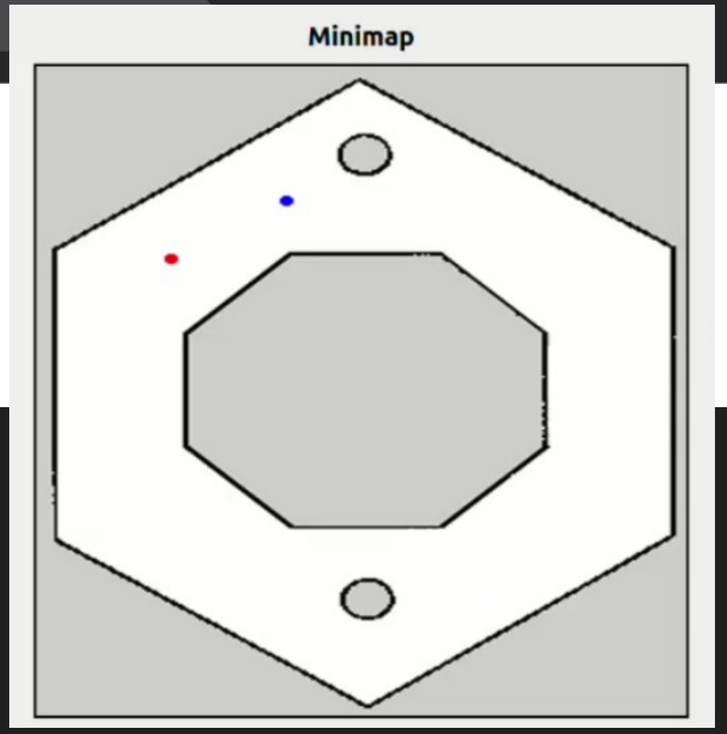
```
# publisher
self.pub_emergency_stop = self.node.create_publisher(
    String,
    "emergency_stop",
    10
)
self.pub_start_stop_toggle = self.node.create_publisher(
    String,
    "start_stop_toggle",
    10
)
```

```
# subscriber
self.sub_wall_e_eye_image = self.node.create_subscription(
    Image,
    "/tb1/camera/image_raw",
    self.wall_e_eye_image_callback,
    10
)
self.sub_global_eye_wall_e = self.node.create_subscription
    PoseWithCovarianceStamped,
    "/tb1/amcl_pose",
    self.amcl_pose_callback_wall_e,
    10
)
self.sub_global_eye_eve = self.node.create_subscription(
    PoseWithCovarianceStamped,
    "/tb2/amcl_pose",
    self.amcl_pose_callback_eve,
    10
)
self.sub_eve_eye_image = self.node.create_subscription(
    Image,
    "/tb2/camera/image_raw",
    self.eve_eye_image_callback,
    10
)
```

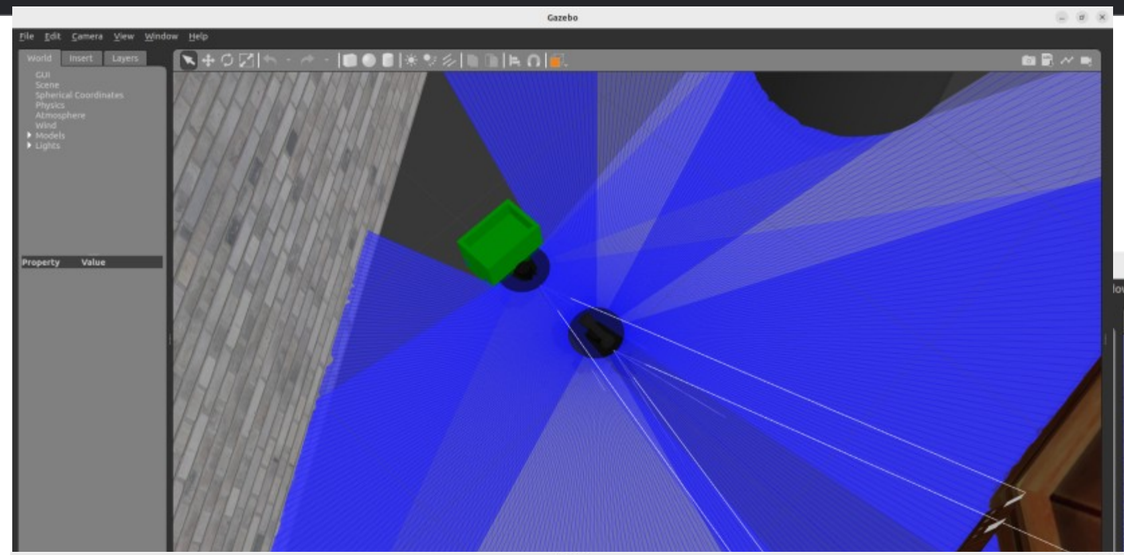
- dual_bot (gui)

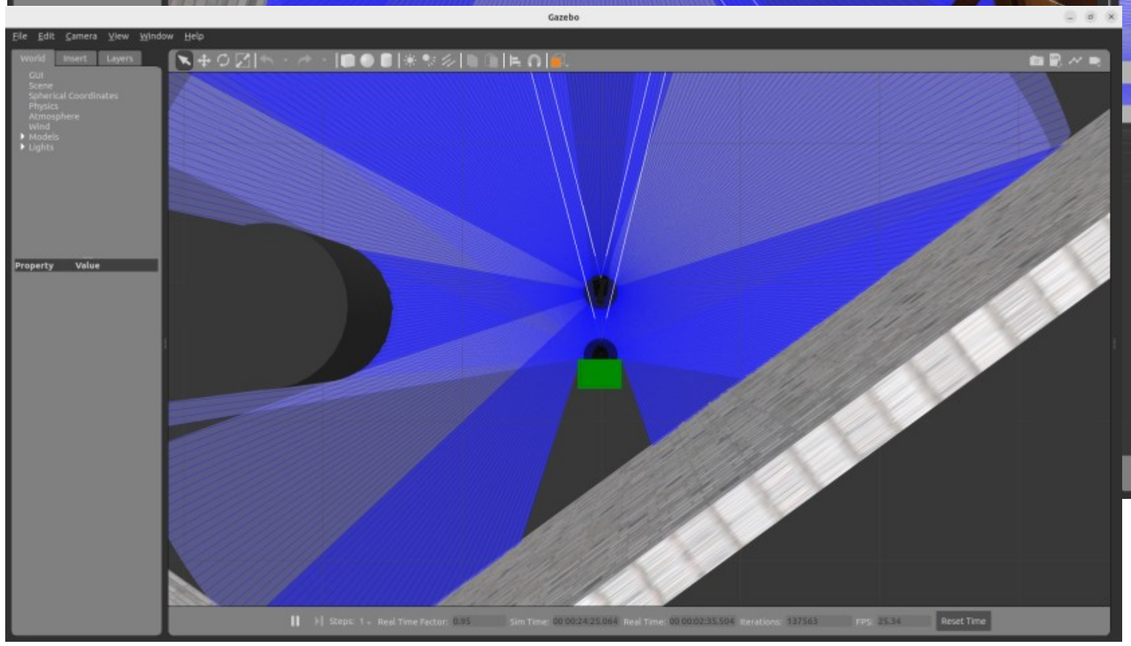
```python
def amcl_pose_callback_wall_e(self, msg):
    self.amcl_pose_x_wall_e = msg.pose.pose.position.x
    self.amcl_pose_y_wall_e = msg.pose.pose.position.y
    self.wall_e_last_update = self.node.get_clock().now()

    if self.wall_e_status != "Active":
        self.wall_e_status = "Active"
        self.wall_e_status_label.setText(f"Wall-E Status\n\n(  {self.wall_e_status}  )")
    self.update_minimap()

def amcl_pose_callback_eve(self, msg):
    self.amcl_pose_x_eve = msg.pose.pose.position.x
    self.amcl_pose_y_eve = msg.pose.pose.position.y
    self.eve_last_update = self.node.get_clock().now()

    if self.eve_status != "Active":
        self.eve_status = "Active"
        self.eve_status_label.setText(f"Eve Status\n\n(  {self.eve_status}  )")
    self.update_minimap()

def check_pose_update(self):
    now = self.node.get_clock().now()
    # Wall-E 상태 확인
    if (now - self.wall_e_last_update).nanoseconds / 1e9 > 2.0:
        if self.wall_e_status != "Inactive":
            self.wall_e_status = "Inactive"
            self.wall_e_status_label.setText(f"Wall-E Status\n\n(  {self.wall_e_status}  )")
    # Eve 상태 확인
    if (now - self.eve_last_update).nanoseconds / 1e9 > 2.0:
        if self.eve_status != "Inactive":
            self.eve_status = "Inactive"
            self.eve_status_label.setText(f"Eve Status\n\n(  {self.eve_status}  )")
```

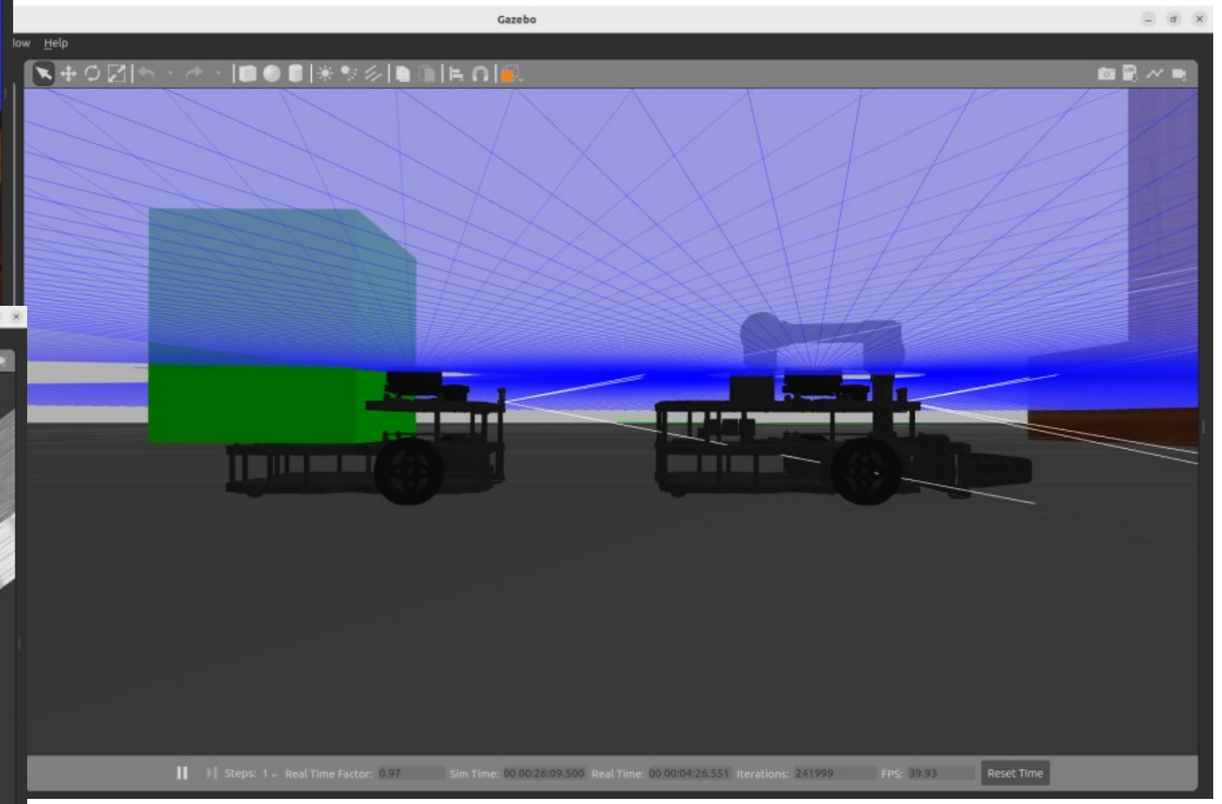
- dual_bot (gui)

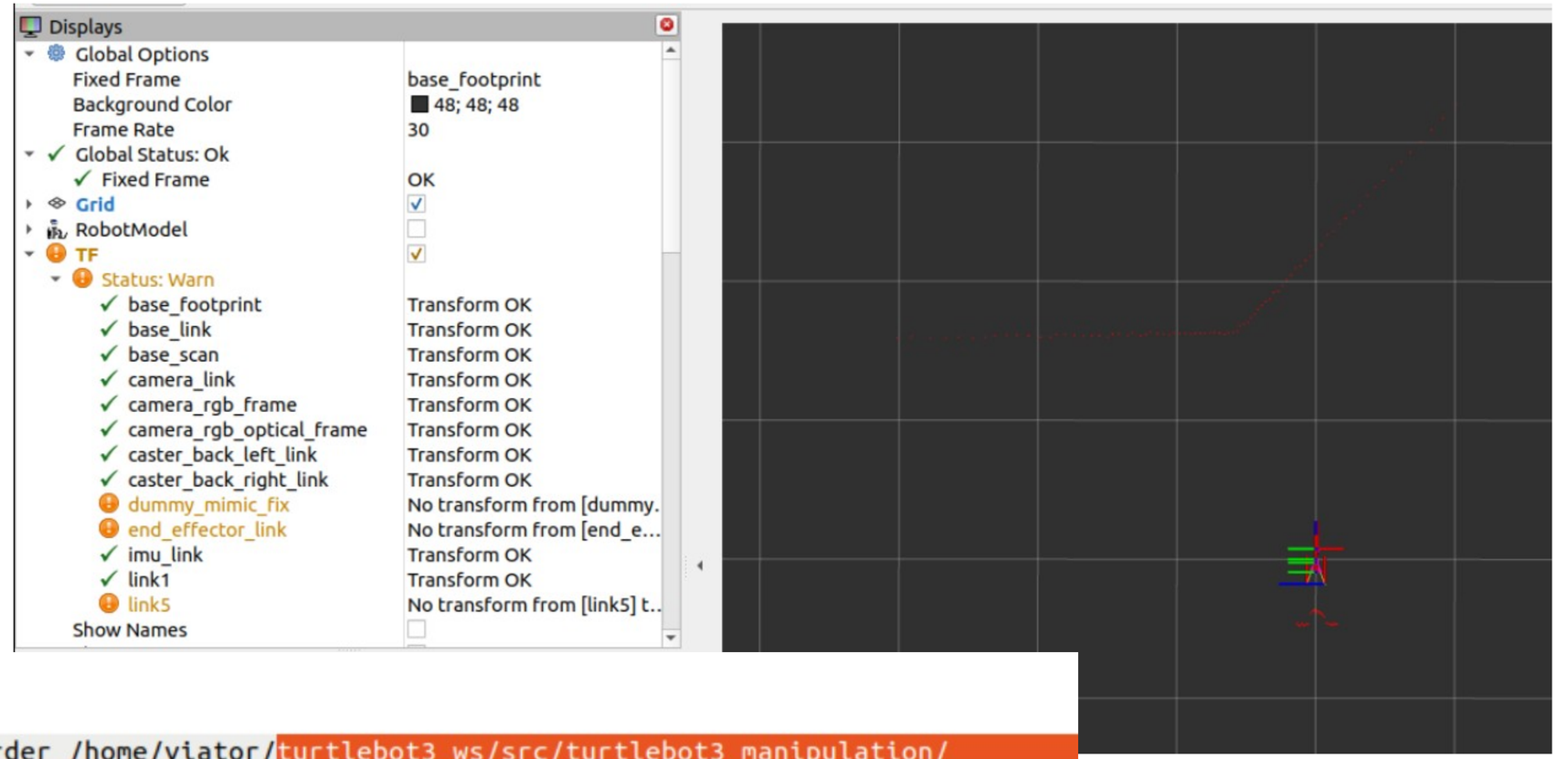



```
def update_minimap(self):
    x_pixel_wall_e = (self.map_origin_x + self.amcl_pose_x_wall_e) / self.resolution
    y_pixel_wall_e = (self.map_origin_y - self.amcl_pose_y_wall_e) / self.resolution
    x_pixel_eve = (self.map_origin_x + self.amcl_pose_x_eve) / self.resolution
    y_pixel_eve = (self.map_origin_y - self.amcl_pose_y_eve) / self.resolution

    image_copy = self.image_rgb.copy()
    draw = PIL.ImageDraw.Draw(image_copy)
    draw.ellipse(
        (x_pixel_wall_e - self.dot_size, y_pixel_wall_e - self.dot_size, x_pixel_wall_e + self.dot_size, y_pixel_wall_e + self.dot_size),
        fill='red'
    )
    draw.ellipse(
        (x_pixel_eve - self.dot_size, y_pixel_eve - self.dot_size, x_pixel_eve + self.dot_size, y_pixel_eve + self.dot_size),
        fill='blue'
    )
    image_rotated = image_copy.rotate(90, expand=True)
    image_resized = image_rotated.resize((500, 500))
    pil_image = image_resized.convert('RGBA')
    data = pil_image.tobytes("raw", "RGBA")
    qimage = QImage(data, 500, 500, QImage.Format_RGBA8888)
    pixmap = QPixmap.fromImage(qimage)
    self.global_eye_display.setPixmap(pixmap)
```


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value

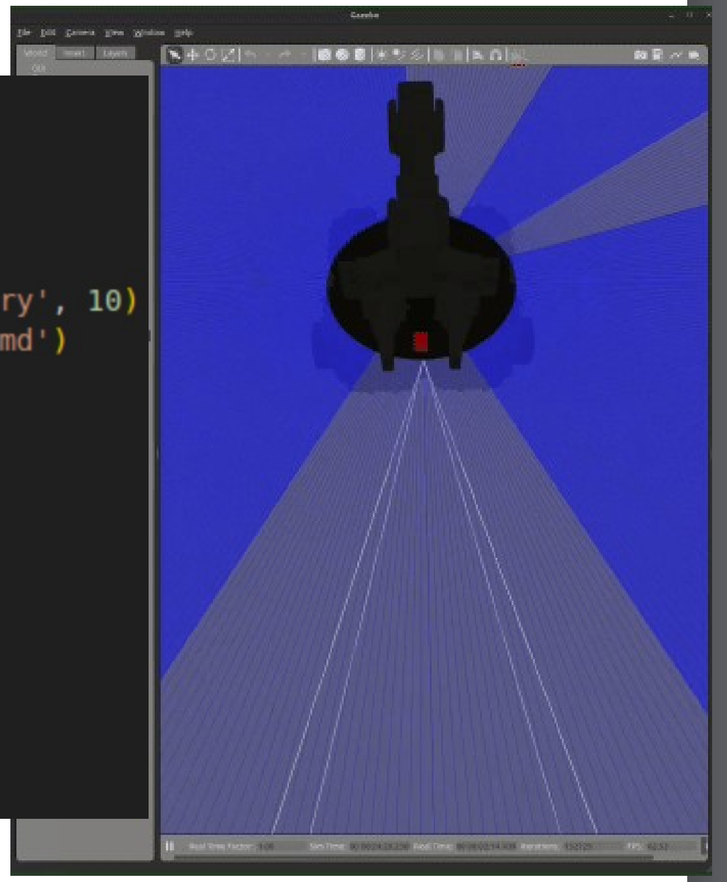

Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 0.95 Sim Time: 00 00:24:25.064 Real Time: 00 00:02:35.504 Iterations: 137563 FPS: 25.34
Reset Time


Gazebo
low Help
Steps: 1 Real Time Factor: 0.97 Sim Time: 00 00:26:09.500 Real Time: 00 00:04:26.551 Iterations: 241999 FPS: 39.93
Reset Time




```
#xacro 파일 urdf 변환
ros2 run xacro xacro --inorder /home/viator/turtlebot3_ws/src/turtlebot3_manipulation/
turtlebot3_manipulation_description/urdf/turtlebot3_manipulation.urdf.xacro -o /home/viator/
Downloads/multitb_ws/src/turtlebot3_multi_robot/urdf/turtlebot3_manipulation.urdf


#URDF->SDF변환
gz sdf -p turtlebot3_manipulation.urdf > turtlebot3_manipulation.sdf
gz sdf -p turtlebot3_with_basket.urdf > turtlebot3_with_basket.sdf
```

```python
class OpenManipulatorWithGripper(Node):
    def __init__(self):
        super().__init__('open_manipulator_with_gripper')

        # Initialize publishers and action clients
        self.joint_publisher = self.create_publisher(JointTrajectory, '/arm_controller/joint_trajectory', 10)
        self.gripper_action_client = ActionClient(self, GripperCommand, 'gripper_controller/gripper_cmd')

        # Constants for manipulator arm
        self.r1 = 130
        self.r2 = 124
        self.r3 = 126
        self.th1_offset = -math.atan2(0.024, 0.128)
        self.th2_offset = -0.5 * math.pi - self.th1_offset

        # Initialize trajectory message
        self.trajectory_msg = JointTrajectory()
        self.trajectory_msg.joint_names = ['joint1', 'joint2', 'joint3', 'joint4']
```

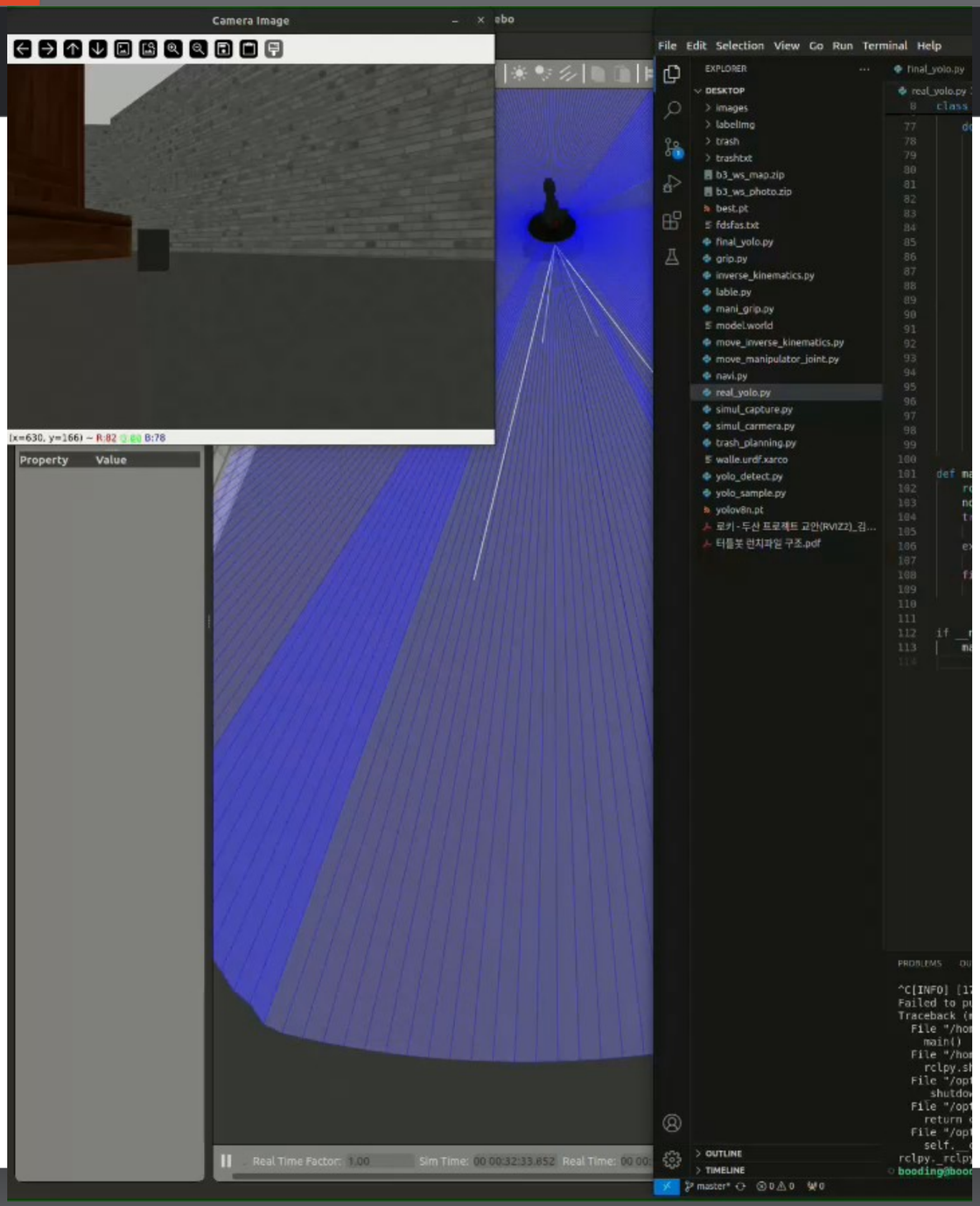


```python
def update_position(self, x, y, z):
    """Calculate joint angles and update trajectory message."""
    try:
        sr1, sr2, sr3, Sxy = self.solv_robot_arm2(x, y, z, self.r1, self.r2, self.r3)
        joint_angles = [Sxy, sr1 + self.th1_offset, sr2 + self.th2_offset, sr3]

        # Update trajectory point
        point = JointTrajectoryPoint()
        point.positions = joint_angles
        point.velocities = [0.0] * 4
        point.time_from_start.sec = 3
        self.trajectory_msg.points = [point]

        # Publish trajectory
        self.joint_publisher.publish(self.trajectory_msg)
        self.get_logger().info(f"Published trajectory with joint angles: {joint_angles}")

    except ValueError as e:
        self.get_logger().error(f"Error calculating joint angles: {e}")
```

```python
def solv_robot_arm2(self, x, y, z, r1, r2, r3):
    """Calculate joint angles for the robot arm to reach the given (x, y, z)."""
    Rt = math.sqrt(x**2 + y**2 + z**2)
    if Rt > (r1 + r2 + r3):
        raise ValueError("Target position is out of the manipulator's reach.")

    St = math.asin(z / Rt)
    Sxy = math.atan2(y, x)
    s1, s2 = self.solv2(r1, r2, Rt)
    sr1 = math.pi / 2 - (s1 + St)
    sr2 = s1 + s2
    sr3 = math.pi - (sr1 + sr2)
    return sr1, sr2, sr3, Sxy

def solv2(self, r1, r2, r3):
    """Calculate angles between links."""
    d1 = (r3**2 - r2**2 + r1**2) / (2 * r3)
    d2 = (r3**2 + r2**2 - r1**2) / (2 * r3)
    s1 = math.acos(d1 / r1)
    s2 = math.acos(d2 / r2)
    return s1, s2
```

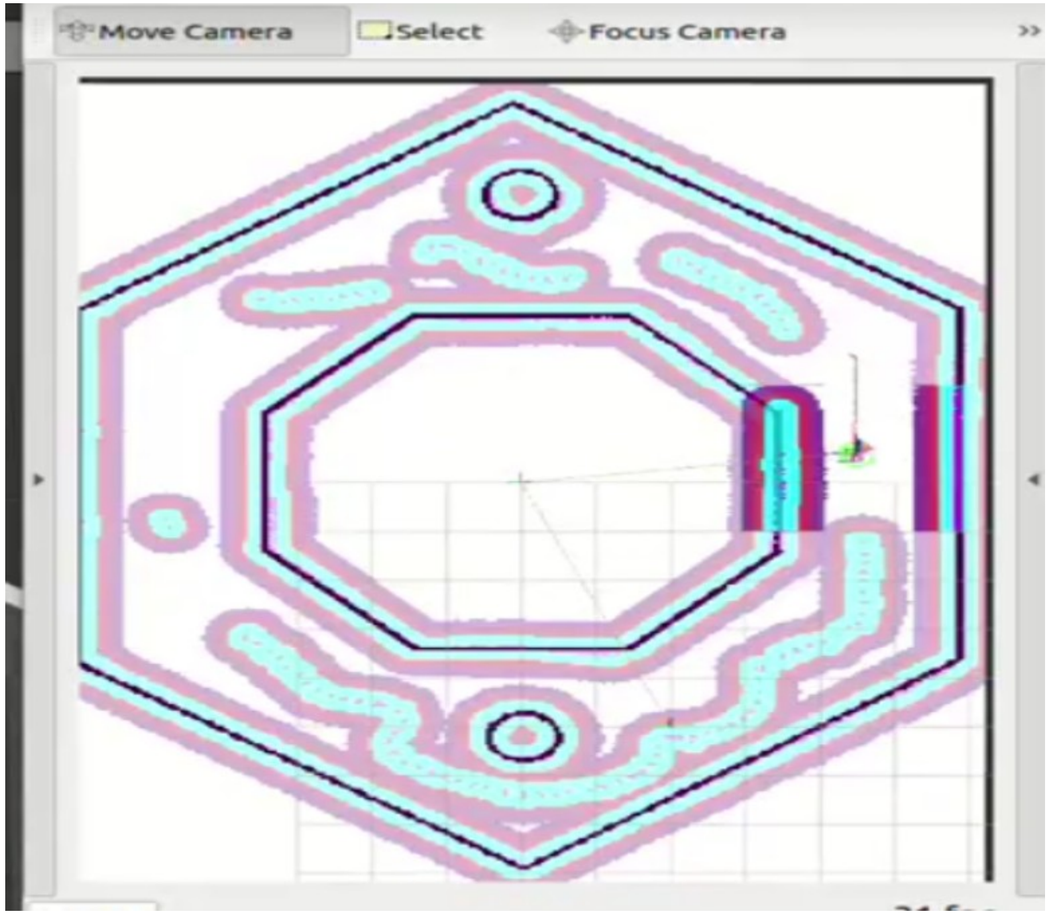
```python
def camera_callback(self, image_msg):
    if not self.is_tracking or self.is_adjusting_distance:
        return  # 트래킹 중이 아니거나 거리 조정 중일 때는 무시

    try:
        # ROS 2 Image -> OpenCV Image
        cv_image = self.bridge.imgmsg_to_cv2(image_msg, "bgr8")

        # YOLOv8 추론
        results = self.model(cv_image, verbose=False)
        boxes = results[0].boxes.data.cpu().numpy()  # numpy 배열로 변환
        self.get_logger().info(f"Detected {len(boxes)} objects.")

        if len(boxes) == 0:
            return  # 감지된 객체가 없으면 처리 중지

        # 첫 번째 객체를 사용 (예: 중앙점 계산)
        x1, y1, x2, y2, conf, cls = boxes[0]
        cx = (x1 + x2) // 2  # 바운딩 박스 중심 x좌표

        # 속도 명령 생성
        twist = Twist()
        twist.linear.x = 0.0  # 트래킹 중에는 선형 이동하지 않음
        image_center = cv_image.shape[1] // 2

        # 방향 계산
        angular_error = (image_center - cx) * 0.001  # 기본 방향 계산
        twist.angular.z = angular_error

        # 정중앙에 위치하면 멈추고 거리 조정 상태로 전환
        if abs(cx - image_center) < 10:  # 10 픽셀 이내
            self.get_logger().info("Object centered. Switching to distance adjustment.")
            self.is_tracking = False  # 트래킹 종료
            self.is_adjusting_distance = True  # 거리 조정 시작
            stop_twist = Twist()
            self.cmd_vel_pub.publish(stop_twist)  # 정지 명령 퍼블리시
            return

        self.get_logger().info(f"Bounding Box Center: {cx}, Image Center: {image_center}, Angular Error: {angular_error}")
        self.get_logger().info(f"Computed Twist - Linear X: {twist.linear.x}, Angular Z: {twist.angular.z}")

        # 이동 명령 저장
        self.current_twist = twist
        self.cmd_vel_pub.publish(twist)

    except Exception as e:
        self.get_logger().error(f"Failed to process image: {e}")
```

```python
def lidar_callback(self, scan_msg):
    if not self.is_adjusting_distance:
        return  # 거리 조정 중이 아니면 무시

    # LiDAR 데이터 처리
    min_distance = min(scan_msg.ranges)  # LiDAR 데이터에서 최소 거리 추출
    self.get_logger().info(f"Minimum distance to object: {min_distance} m")

    twist = Twist()

    # 거리 조정 로직
    if min_distance > self.target_distance:  # 목표 거리보다 멀면 접근
        self.get_logger().info("Too far. Moving closer.")
        twist.linear.x = 0.1  # 천천히 전진
    elif min_distance < self.target_distance:
        self.get_logger().info("Too close. Moving backward.")
        twist.linear.x = -0.1  #
    else:
        self.get_logger().info("Target distance achieved. Stopping.")
        self.is_adjusting_distance = False  # 거리 조정 완료
        twist.linear.x = 0.0  # 정지

    self.cmd_vel_pub.publish(twist)
```

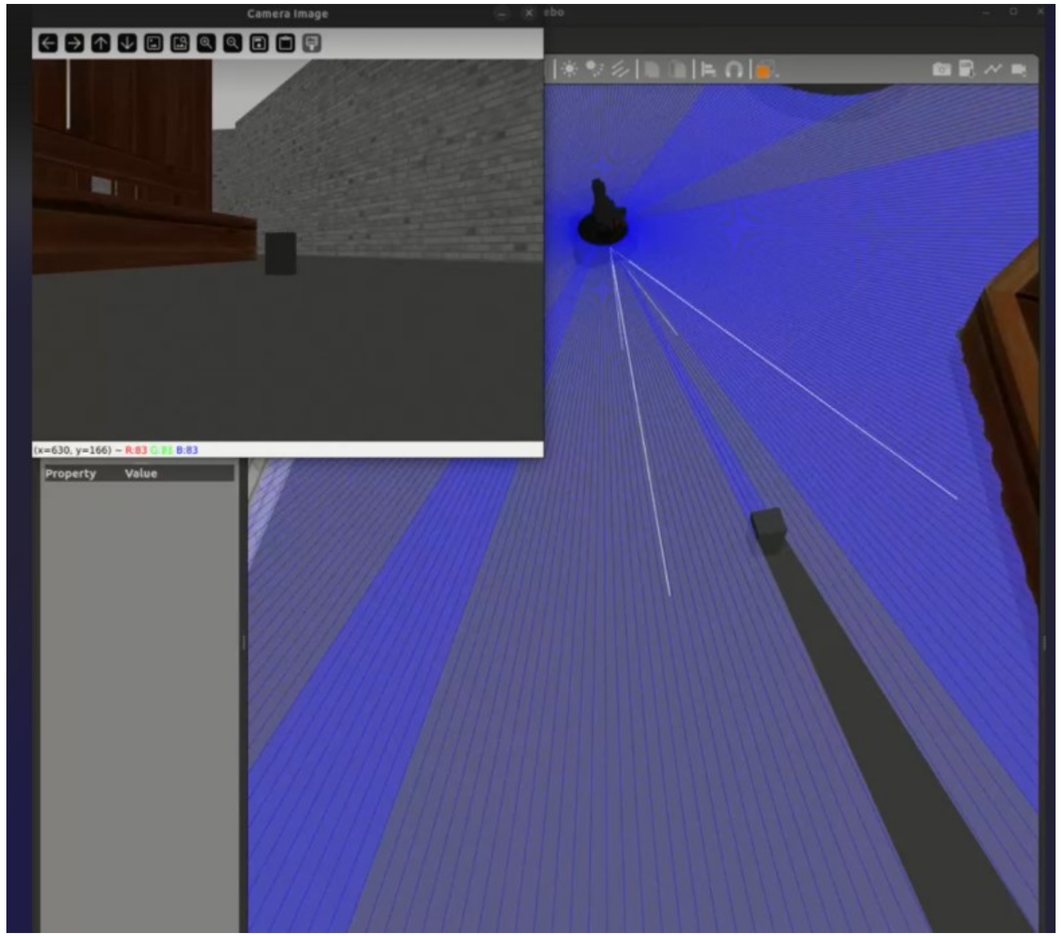

Camera Image
File Edit Selection View Go Run Terminal Help
EXPLORER
DESKTOP
images
labelImg
trash
trashtxt
b3_ws_map.zip
b3_ws_photo.zip
best.pt
fdsfas.txt
final_yolo.py
grip.py
inverse_kinematics.py
lable.py
mani_grip.py
model.world
move_inverse_kinematics.py
move_manipulator_joint.py
navi.py
real_yolo.py
simul_capture.py
simul_carmera.py
trash_planning.py
walle.urdf.xarco
yolo_detect.py
yolo_sample.py
yolov8n.pt
Property
Value
Real Time Factor: 1.00
Sim Time: 00 00:32:33.852
Real Time: 00 00:
OUTLINE
TIMELINE
master*

- 개선 사항

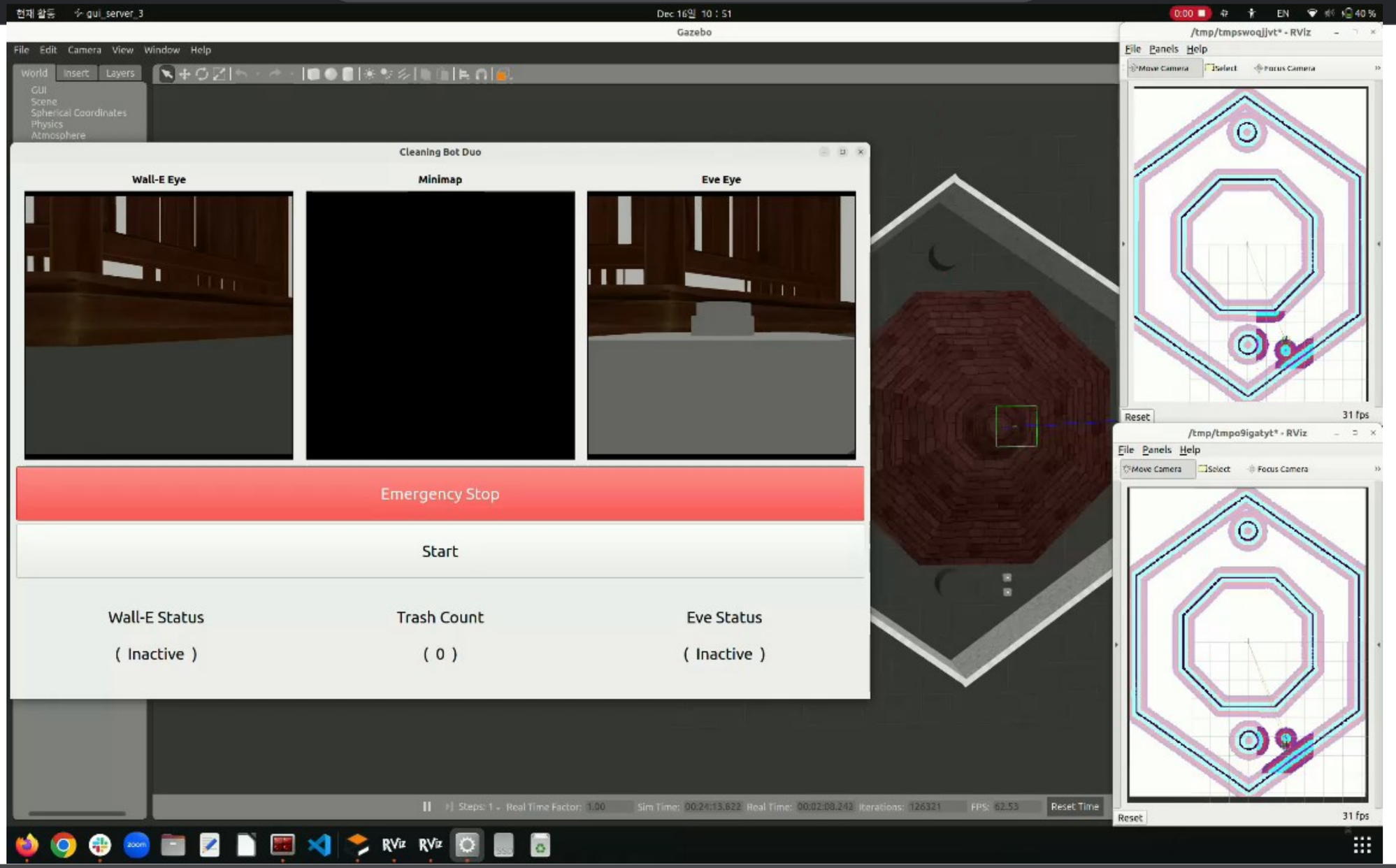
1. Lidar 범위 조정



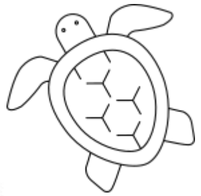
2. YoLo 인식 불안정



시연


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Cleaning Bot Duo
Wall-E Eye
Minimap
Eve Eye
Emergency Stop
Start
Wall-E Status
(Inactive)
Trash Count
(0)
Eve Status
(Inactive)
/tmp/tmpswoqljvt* - RViz
File Panels Help
Move Camera
Select
Focus Camera
Reset
31 fps
/tmp/tmpo9igatyt* - RViz
File Panels Help
Move Camera
Select
Focus Camera
Reset
31 fps

Thank You



B-3
오 건, 김강경, 전수현,
최유민