Author

- 22000167 Kim Inyeop

github link
https://github.com/Kiminyeop-cpu/Robot-Automation/tree/main

Demo link
IAIA Robot Automation

1. Background

1-1. Introduction

With the rapid growth of the electric vehicle (EV) market, the secondary battery industry has emerged as a key strategic sector. It is projected that by 2030, electric vehicles will account for more than 40% of the global automotive market, leading to a significant increase in battery production demand. However, conventional battery cell assembly and disassembly processes remain highly dependent on manual labor, resulting in limitations in productivity, process consistency, and quality control. In particular, as the importance of battery recycling and safe disassembly increases, there is a growing demand for intelligent automation systems capable of handling precise and complex operations beyond simple assembly tasks.

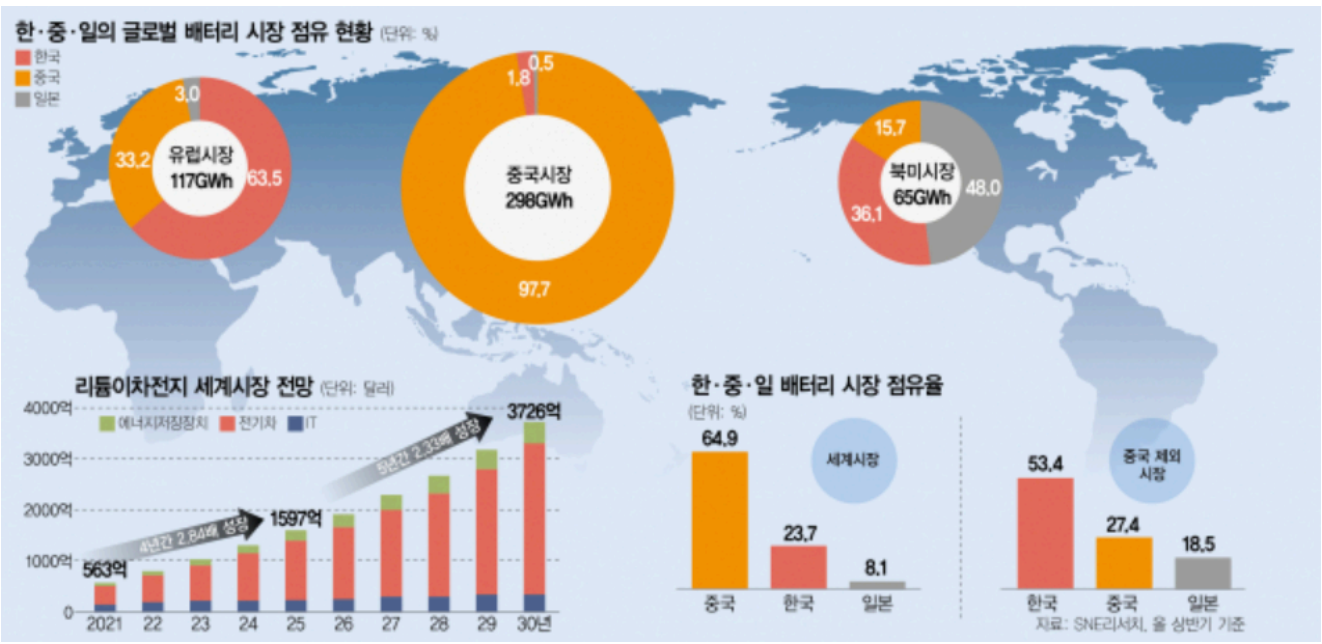

Figure 1. Battery Market Share

This project aims to address these challenges by developing an **automated battery cell winding process using industrial robots integrated with machine vision and artificial intelligence technologies**. By combining robotic manipulation, sensor-based detection nodes, state diagram–based control logic, and camera-based recognition of battery cells and winding components, the project seeks to transform a labor-intensive process into a high-speed, high-precision automated system. Through this approach, the feasibility of improving productivity, reliability, and safety in battery manufacturing and recycling processes is investigated.

1-2. Problem Statement

1-2.1. Project Objectives

The main objectives of this project are as follows:

1. **Implementation of Automated Assembly and Disassembly of Battery Cell Winding Processes**
   To automate the assembly and disassembly of battery cells and winding components using industrial robots, thereby minimizing human error and improving repeatability.
2. **Development of a Machine Vision–Based Recognition System**
   To detect battery cell positions, orientations, and conditions (normal or defective) using camera-based vision systems and AI models, and to integrate the recognition results into the robot control system.
3. **Design of Robot Control Logic and State Diagrams**
   To design structured state diagrams that manage process flow and ensure stable coordination between sensor detection nodes and robotic actions.
4. **Verification of Process Reliability and Performance**
   To experimentally evaluate the success rate, accuracy, and operational stability of the automated system and assess its applicability to real industrial environments.

1-2.2. Expected Outcomes

The expected outcomes of this project are as follows:

1. **Improved Productivity and Operational Efficiency**
   By replacing manual labor with robotic automation, continuous operation beyond traditional working hours becomes possible, significantly increasing overall productivity.
2. **Enhanced Accuracy and Quality Consistency**
   The automated system is expected to achieve a high success rate in assembly and disassembly tasks, while the machine vision system enables reliable identification of normal and abnormal battery cells, excluding specific inclusion cases.
3. **Reduced Labor Dependency and Improved Workplace Safety**
   Automating repetitive and potentially hazardous battery handling processes reduces worker

exposure to risk and lowers dependence on skilled manual labor.

4. **Scalability for Battery Recycling and Future Smart Factory Applications**
   The proposed system can be extended to battery disassembly and recycling processes, providing a technological foundation for advanced smart factory and sustainable battery production systems.

1-3. Evaluation Index

Process	Target
UR5 Robot Operation	- Bracket detection success rate ≥ 98% - Cell box detection success rate = 100% - Pick and Place success rate ≥ 95% - Stopper linkage success rate = 100% - Assembly/Disassembly process completion time ≤ 1 minute
Indy10 Robot Operation	- Cell orientation detection success rate = 100% - Foreign object detection success rate ≥ 95% - Defective cell detection success rate ≥ 95% - Stopper linkage success rate = 100% - Winding assembly/disassembly process completion time ≤ 1 minute
Defect Detection	- Defect detection success rate (cylindrical surface) ≥ 95% - Defect detection success rate (flat surface) ≥ 80%

2. Requirement

Software(In Process Awareness-Based Pick and Place Automation Part, ur5e)

- OS: Linux Ubuntu 20.04
- ROS: ROS Noetic
- Additional Packages Required
  - OpenCV
  - rospy
  - numpy
  - moveit_ros_planning
- Additional Ubuntu Utility Programs
  - VS code
  - terminator

    To build overall software setting, follow the link for extra guideline.

Hardware

- Cell & Wind(Cell Radius: 57mm, Bracket Radius: 36mm)

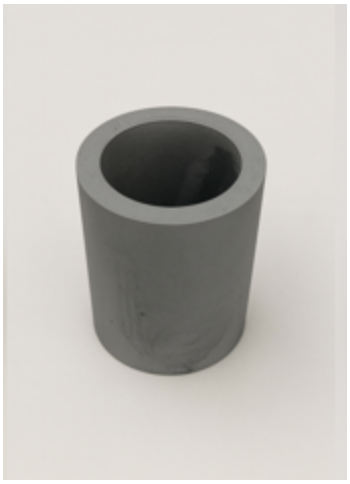

Figure 2A. Cell

Figure 2B. Wind

- Bracket(Radius: 60mm)

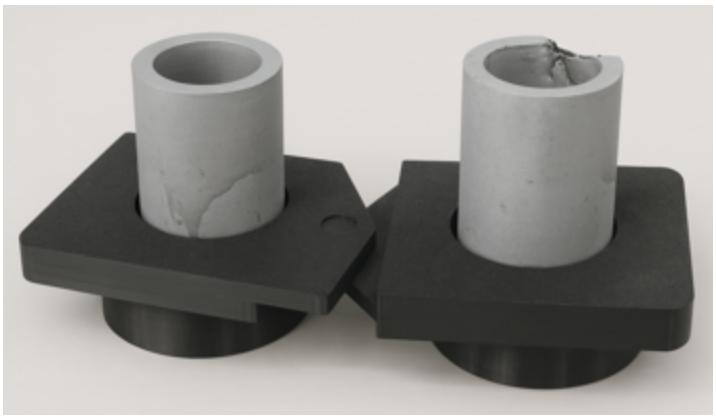

Figure 3. Bracket

- Cell Box(Width & Length: 60mm)

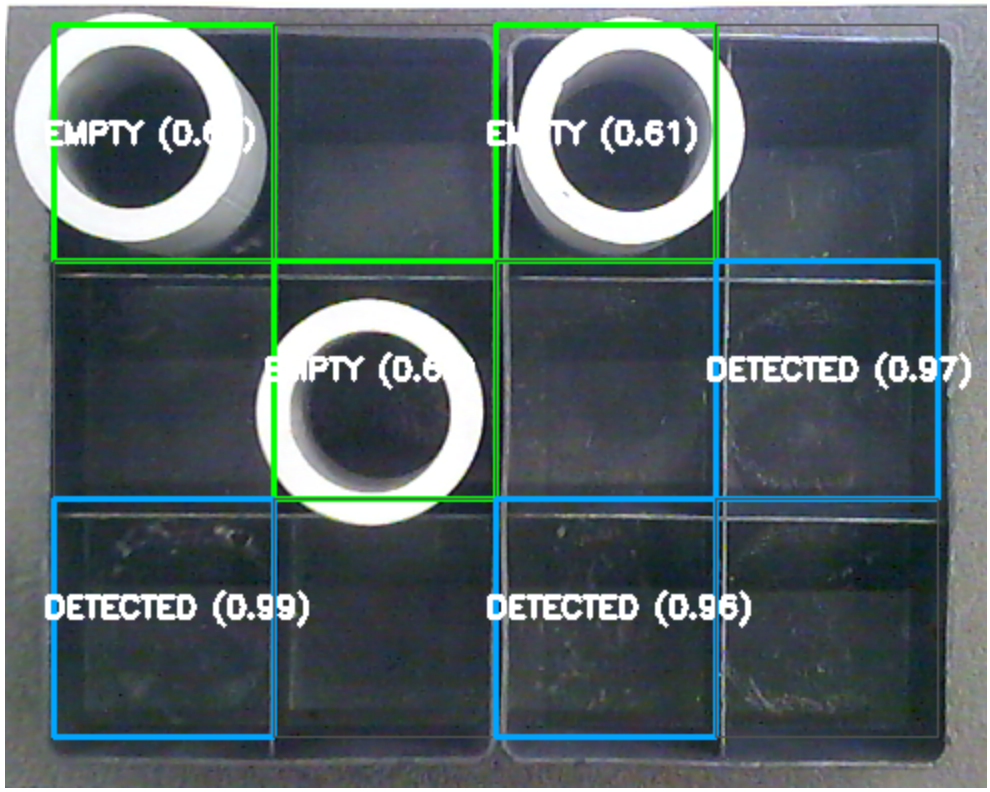

Figure 4. Cell Box

3. Tasks

3-1. Assembly/Disassembly Process Recognition

- Recognizes whether a bracket contains a cell.
- Recognizes cells/composites in the cell box.
- Calculates coordinates for picking from the cell box and placing on the bracket.
- If a cell is missing from the cell box, the disassembly process switches. If a composite is missing, the assembly process switches.

3-2. Pick and Place Process Description

1. The ur5e robot approaches the bracket and checks for the presence of a cell.
2. If a cell is present, it places the cell in the appropriate cell box. If not, it picks the appropriate cell from the cell box.
3. During pick and place, the ur5e robot processes the image and calculates relative coordinates (pixel to millimeter) based on the camera, allowing the gripper to reach the correct location.
4. The robot automatically determines and switches between the assembly and disassembly processes based on the presence of a cell in the cell box.

4. Algorithm

4-1. Logic Design

Based on the tasks listed above, the system's flowchart is drawn as below:

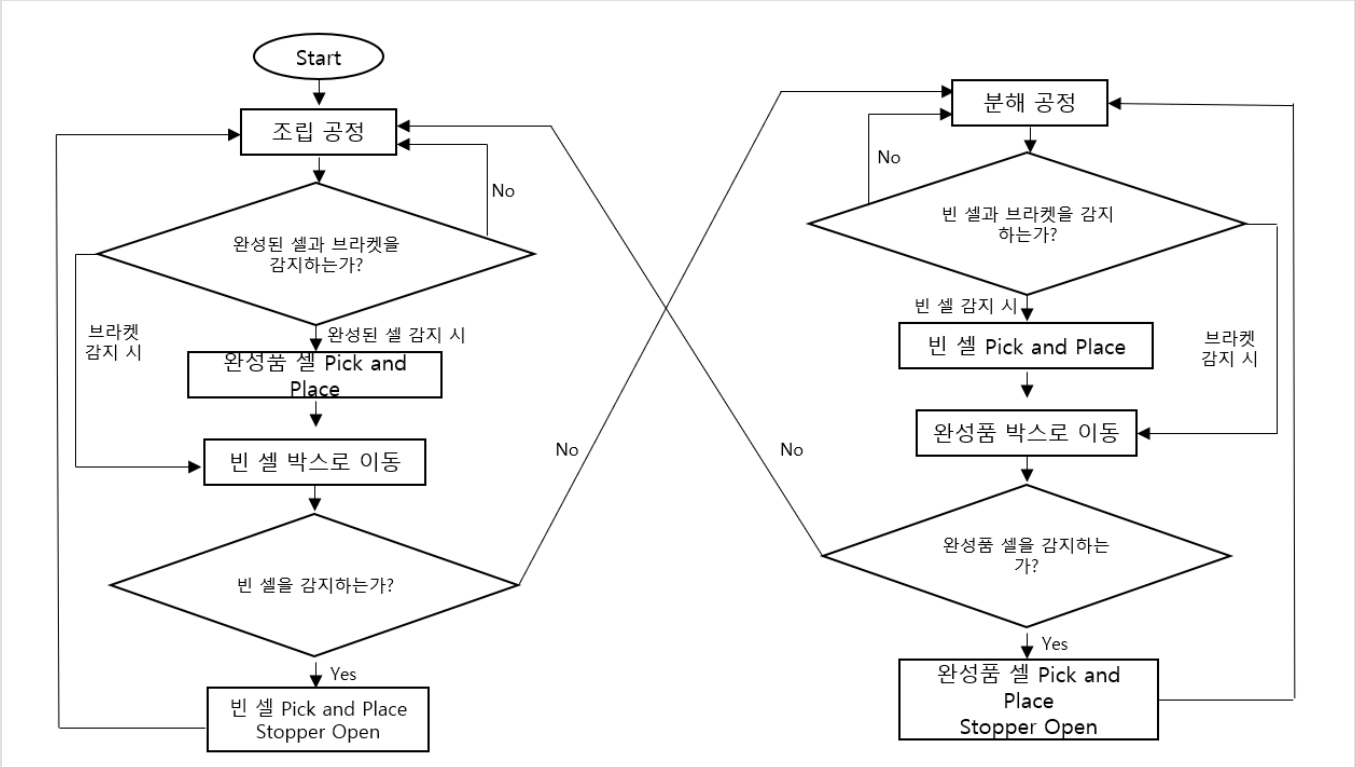

Figure 5. State Diagram

4-1.1. Assembly Process

- **State 1**: First Bracket Detection (State 2 if Composite is detected, State 5 if Bracket is detected)
- **State 2**: Place Composite in Cell Box, then State 3
- **State 3**: Second Bracket Detection (State 4 if Composite is detected, State 6 if Bracket is detected)
- **State 4**: Place Composite in Cell Box, then State 5
- **State 5**: Pick a cell from the cell box, place it on the bracket, then State 6 (if no cell is in the cell box, then State 101 (Disassembly Process))
- **State 6**: Pick a cell from the cell box, place it on the bracket, then reset to State 1 (if no cell is in the cell box, then State 201 (Disassembly Preparation Process))

4-1.2. Disassembly Process

- **State 101**: First Bracket Recognition (State 102 for cell detection, State 105 for bracket detection)
- **State 102**: Place the cell in the cell box, then State 103
- **State 103**: Second bracket recognition (State 4 for cell detection, State 106 for bracket detection)
- **State 104**: Place the cell in the cell box, then State 5
- **State 105**: Pick the composite from the cell box, then place it on the bracket, then State 106 (if there is no composite in the cell box, then State 1 (assembly process))
- **State 106**: Pick the composite from the cell box, then place it on the bracket, then reset to State 101 (if there is no composite in the cell box, then State 301 (assembly preparation process))

4-1.3. Disassembly Preparation Process

- **State 201**: First bracket recognition (if there is a composite, place it in the cell box, then State 101 (disassembly process) and State 202 for bracket detection)
- **State 202**: Detect a composite in the second bracket, place it in the cell box, and switch to State 101 (disassembly process).

4-1.4. Assembly Preparation Process

- **State 301**: First bracket recognition (if a cell is detected, place it in the cell box, then switch to State 1 (assembly process). If a bracket is detected, switch to State 302.)
- **State 302**: Detect a cell in the second bracket, place it in the cell box, and switch to State 1 (assembly process).

4-2. Image Processing

The following image processing was used to identify the battery cell and assembly/disassembly process status.

1. yellow_bracket_detection_node.py

- This sensor node detects the bracket holes in the yellow bracket.
  Bracket (yellow circle) detection

```
mask = cv2.inRange(hsv, [20,120,120], [32,255,255])
circles = cv2.HoughCircles(mask, cv2.HOUGH_GRADIENT,
                           dp=1.2, minDist=190,
                           minRadius=210, maxRadius=430)
x, y, r = max(circles, key=lambda c: c[2])
```

- HSV-based yellow segmentation followed by Hough Circle circular bracket detection.
- Since the brackets are of constant size, false detections can be eliminated within the radius range.

EMA Filter

```
x = (1-a)*x_prev + a*x
y = (1-a)*y_prev + a*y
```

- Removing frame-to-frame positional blur with EMA filters

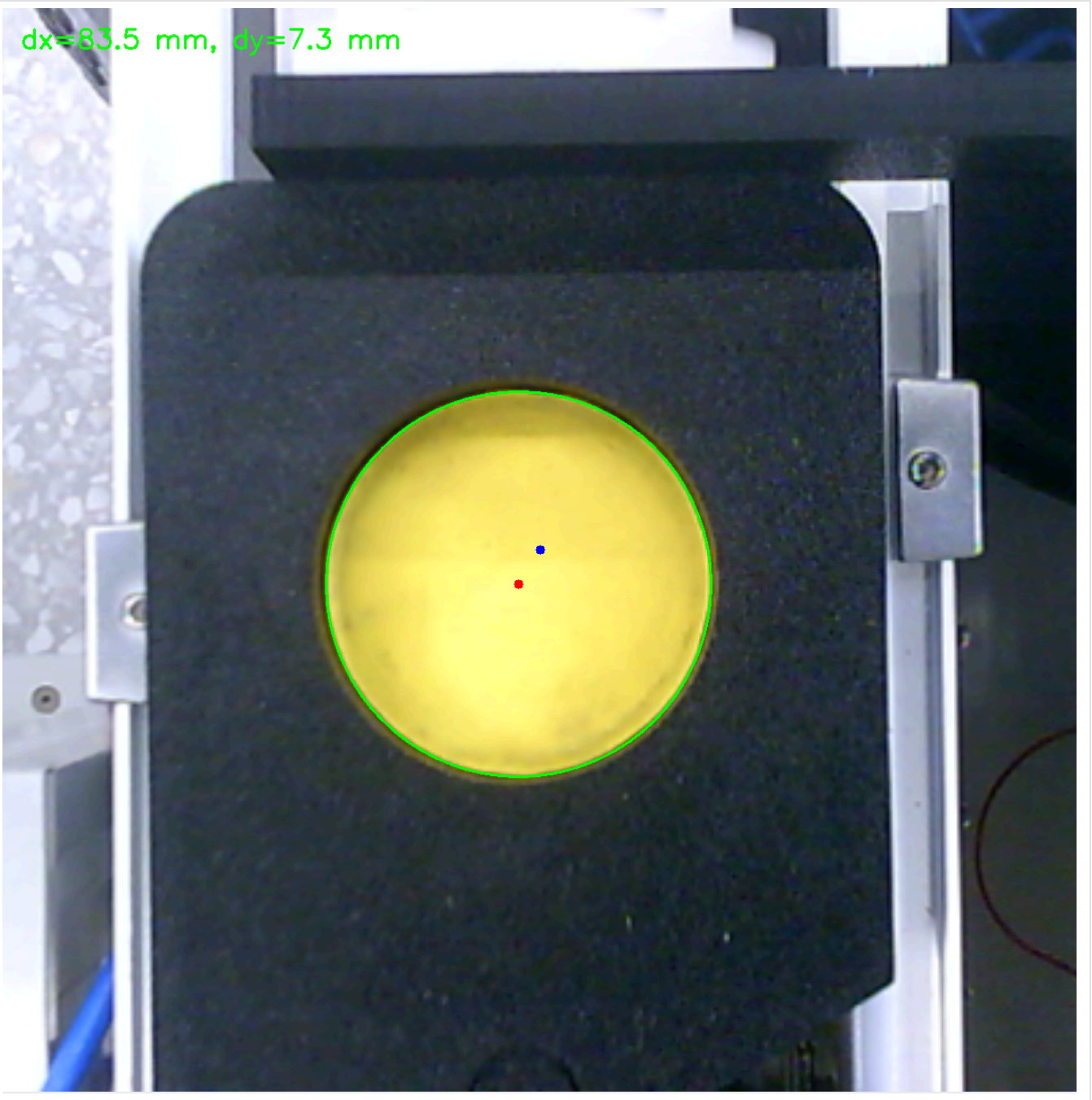

Figure 6. Bracket Detection

2. filled_cellbox_detection_node.py

- This detection node detects cells in an empty cell box.
  Cell box brightness detection technique (classical image processing)

```
bright_mask = np.all(cell_roi >= 215, axis=2)
ratio = np.sum(bright_mask) / cell_area
```

- This detection technique is a classical computer vision method that uses RGB color space-based threshold brightness detection.
- Pixels within each cell region whose R, G, and B channel values all exceed a threshold are defined as bright pixels. If the area ratio of these pixels exceeds a threshold, the cell is determined to be detected.

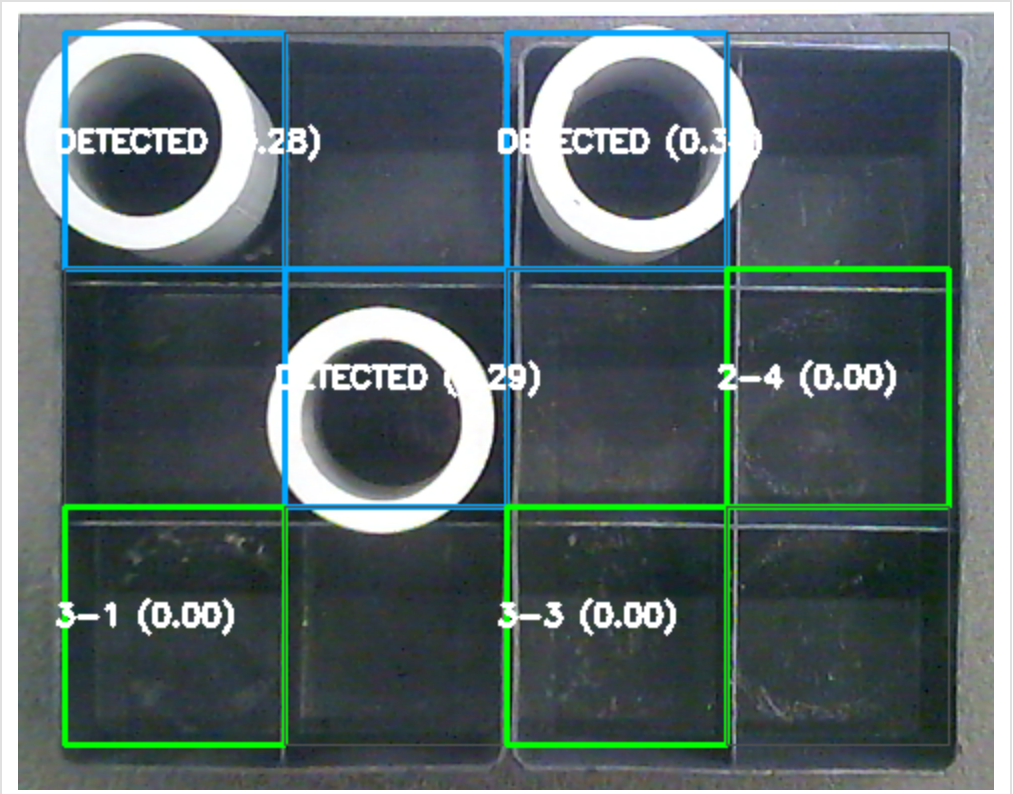

Figure 7. Cell Detection in Cellbox

3. empty_composite_cellbox_detection_node.py

- This detection node detects empty boxes in the composite cell boxes.
  Cell detection technique (classical image processing)

```
mean_intensity = np.mean(cell_region)
if mean_intensity < 90:
```

```
    detected = True
```

- This technique is a classical computer vision method that uses grayscale average intensity-based threshold detection.
- If the mean intensity of each grid cell area falls below a threshold, the cell is considered dark and is then detected as a location for a composite cell.

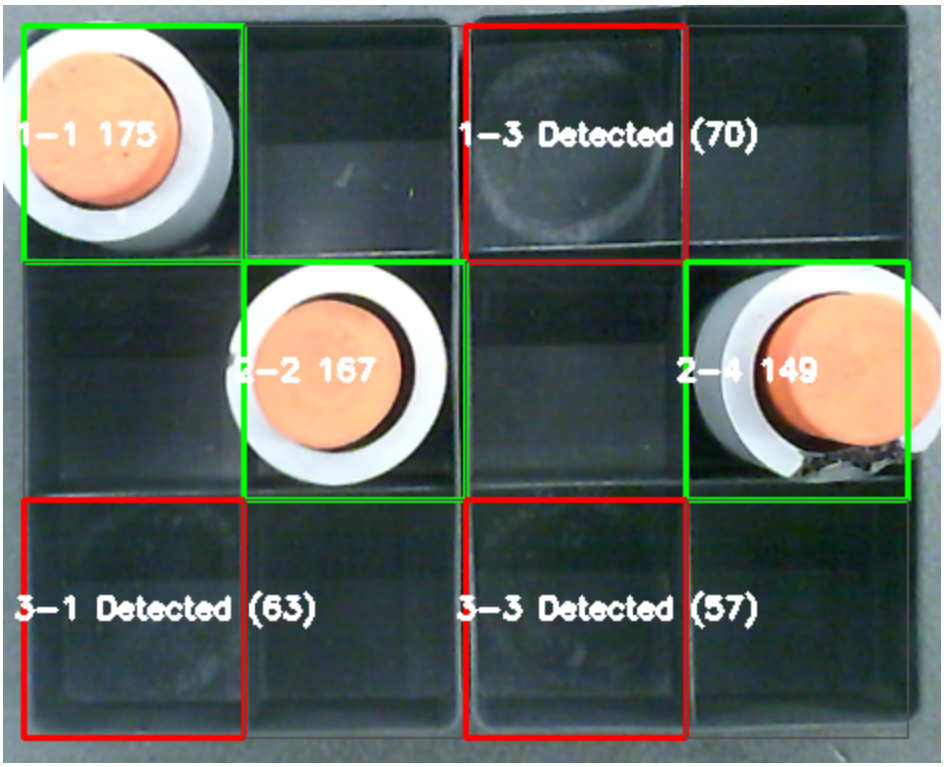

Figure 8. Empty Composite Cell in Cellbox

4. filled_composite_cellbox_detection_node.py

- This detection node detects composite cells in the finished cell box.
  Composite cell detection core logic

```
orange_mask = cv2.inRange(cell_roi,
                          (65,110,220), (185,220,255))
ratio = np.sum(orange_mask > 0) / cell_area
```

- This node is a classical computer vision technique that uses threshold color detection based on the RGB (BGR) color space.
- If the percentage of pixels within each cell that fall within the orange range exceeds a certain threshold (≥5%), then the cell is considered detected.

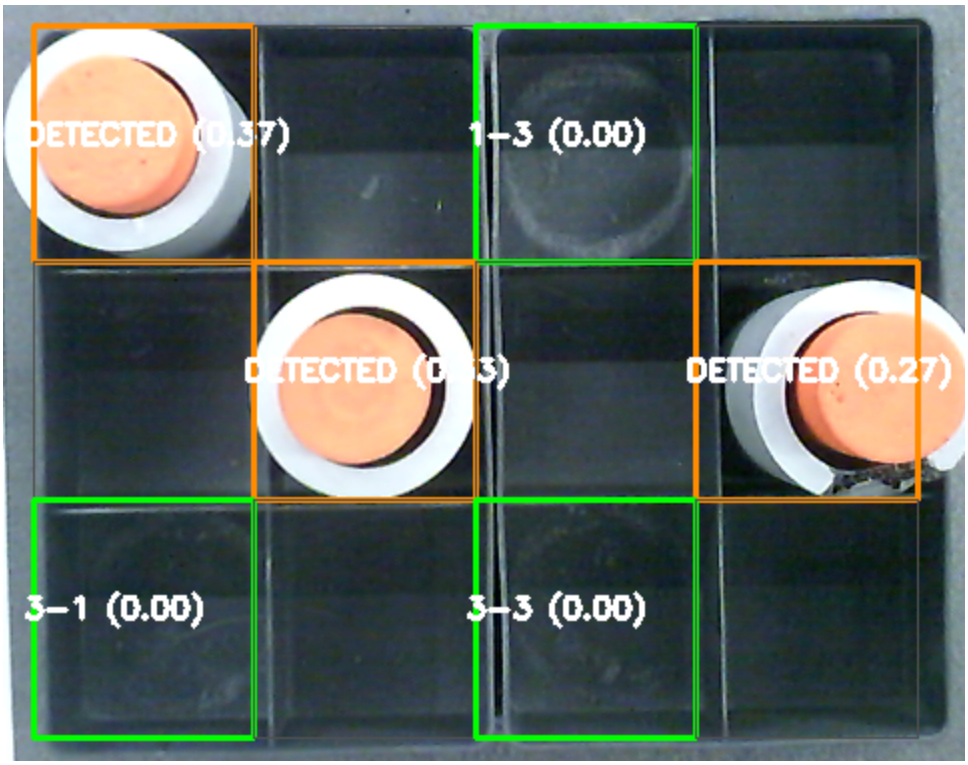

Figure 9. Filled Composite Cell in Cellbox

**5. empty_cell_cellbox_detection_node.py

- This detection node detects empty spaces in empty cell boxes.
  Empty space detection core logic

```
black_mask = cv2.inRange(hsv_roi, lower_black, upper_black)
```

- It is a threshold-based classical image processing technique that determines the presence of a black object based on the ratio of pixels with low brightness (V) in the HSV color space.

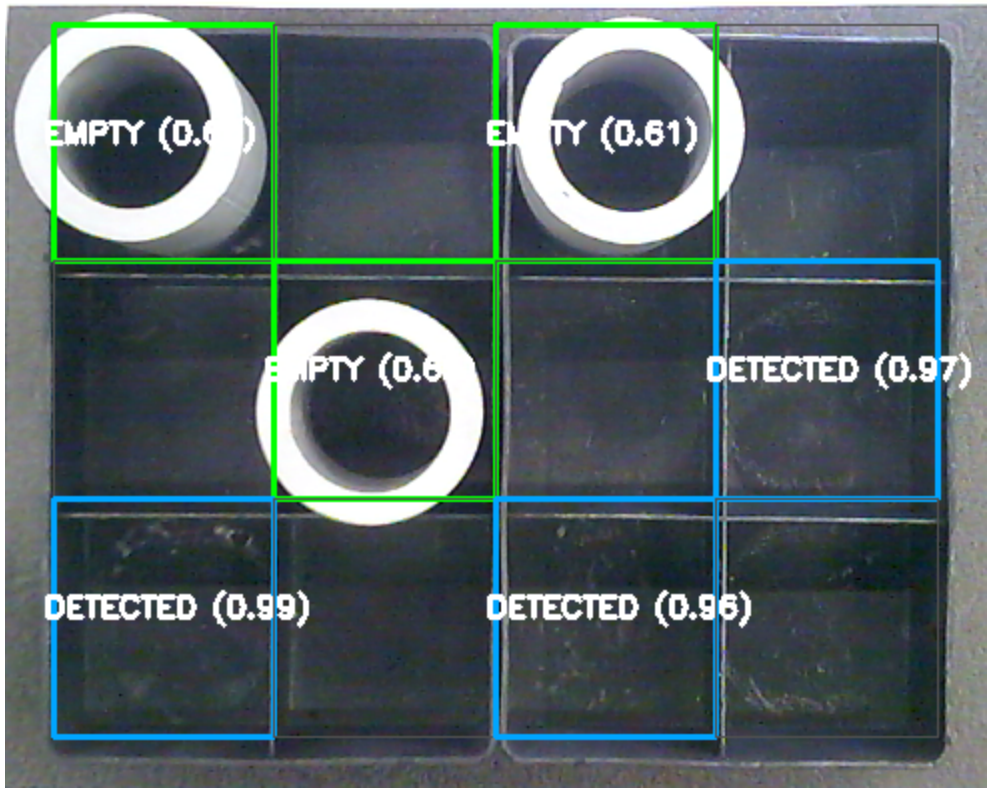

Figure 10. Empty Cell in Cellbox

6. winding_decect_node.py

- This sensor node detects the wind.
  Wind detection core logic

```
circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, ...)
orange_ratio = orange_pixels_inside_circle / circle_area
```

- It is a classical image processing technique that detects circular objects using grayscale-based Hough Circle Transform and determines them as valid objects only when the proportion of orange pixels inside the detected circle exceeds a threshold.

7. composite_detection.py

- This detection node detects the composite cell.
  Composite cell detection core logic

```
circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, ...)
```

- It is a classical image processing technique that detects circular objects by applying Hough Circle Transform to a grayscale threshold image within the central ROI, and determines

them as valid locations only when the center distance condition is satisfied.

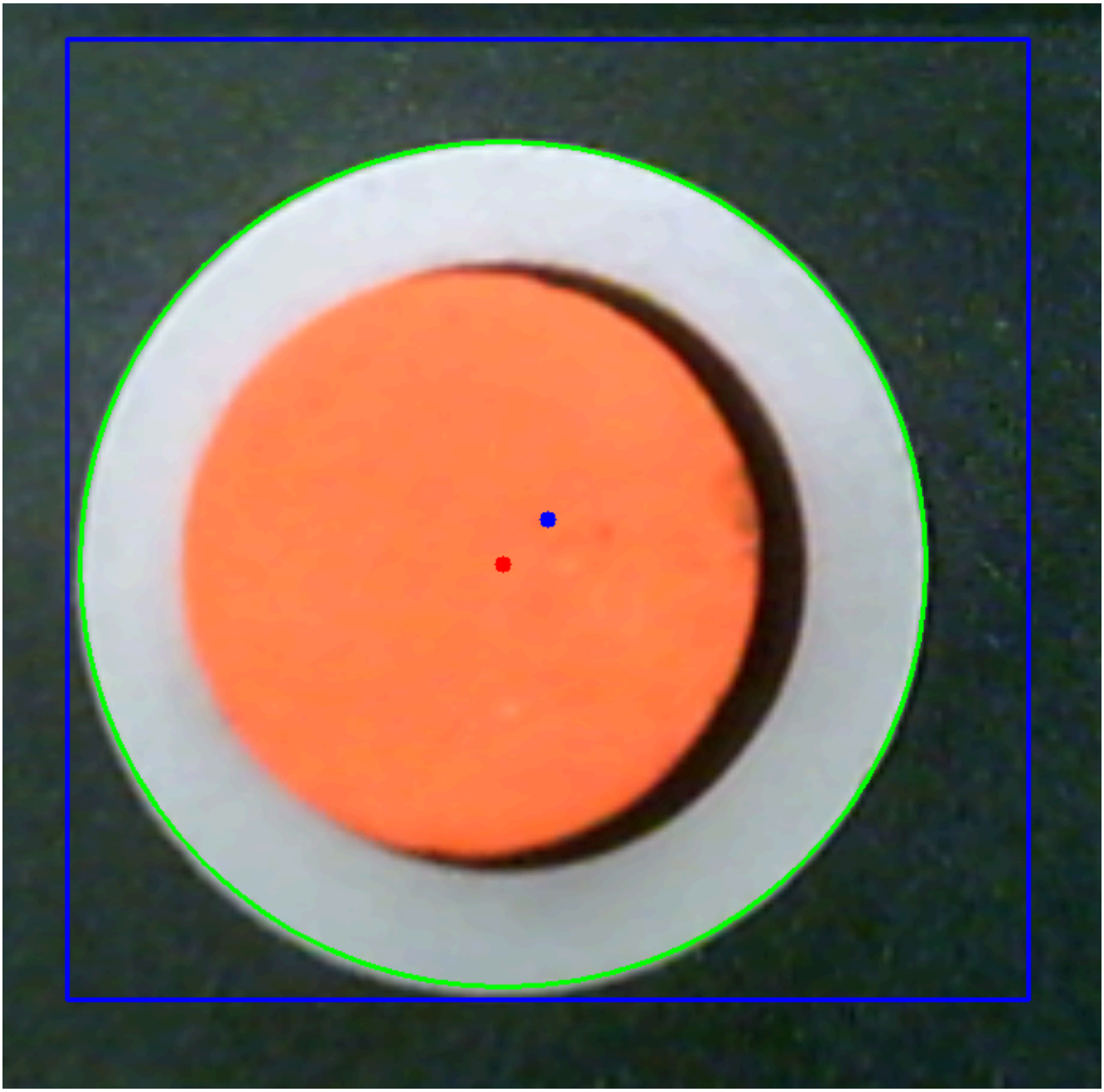

Figure 11. Composite Detection

**8. cell_detection.py

- This detection node detects empty cells.
  Empty cell detection core logic

```
circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, ...)
```

- It is a rule-based classical image processing technique that detects circular objects by applying Hough Circle Transform to grayscale threshold images, and judges them as pure

gray objects only when the orange/yellow ratio inside the circle is below the threshold.

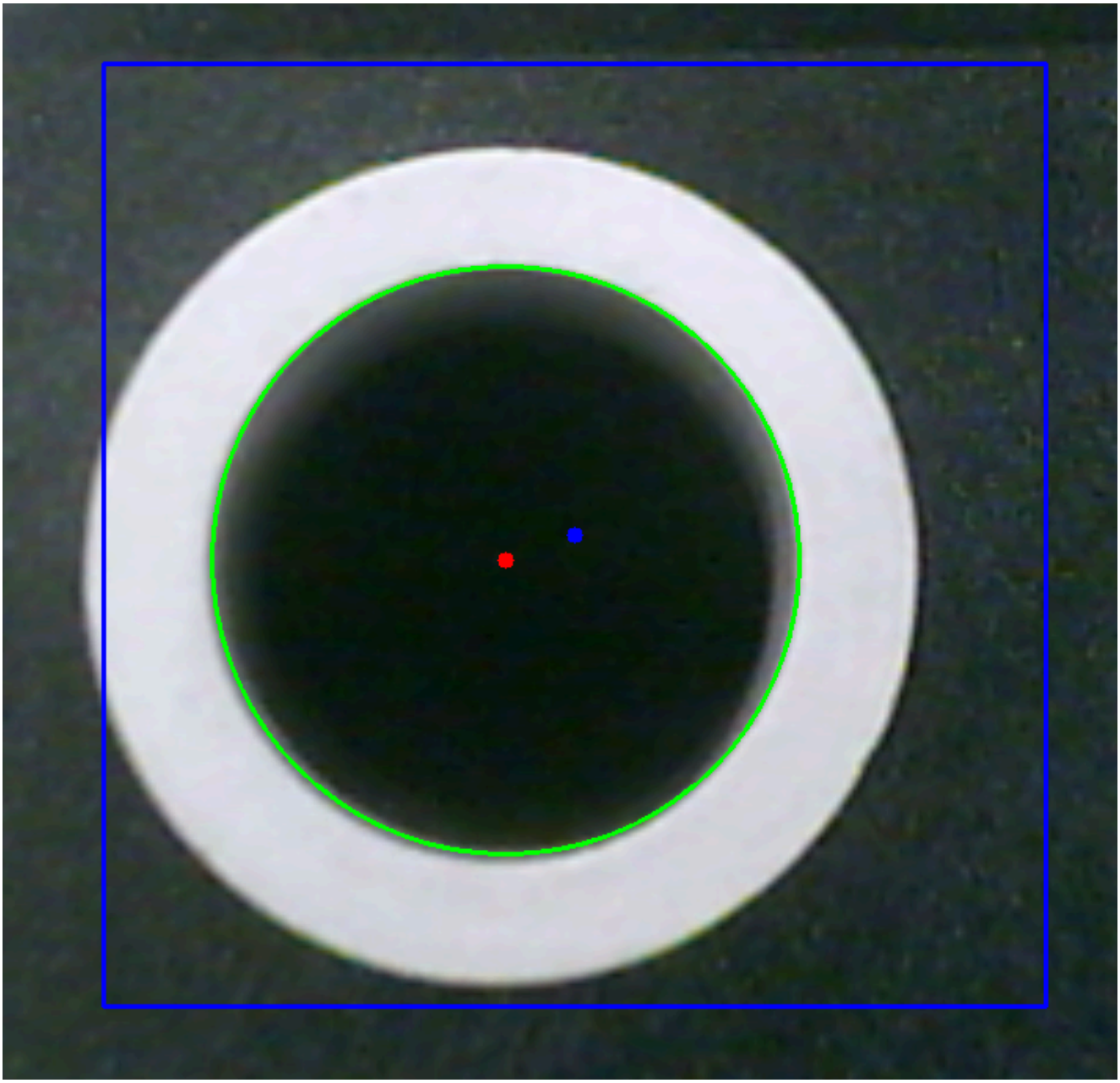

Figure 12. Cell Detection

4-3. Pick and Place Algorithm(Manipulation)

1. Joint-based locomotion (MoveJ)

```
def goJ(self, arr_deg, label=""):
    ok = self.ur5e.go_to_joint_abs(self.jdeg(arr_deg))
    rospy.sleep(1.5)
    return ok
```

- Meaning: Move the UR5e relative to absolute joint coordinates using a predefined array of joint angles.
- Usage: Stable pose transitions, such as observation poses, grid access, and return poses.

2. Cartesian movement based on relative coordinates (XY/Z separation)

```
self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
self.ur5e.go_to_pose_rel([0.0, 0.0, -0.07], [0, 0, 0])
```

- Meaning: Relative movement relative to the current TCP
- Features: After moving in the XY plane, only the Z axis is lowered/raised → Minimizes collisions
- Usage: Vision-based fine-position correction pick-and-place

3. Pick Basic Sequence

```
# XY 이동
self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0,0,0])
# Z 하강
self.ur5e.go_to_pose_rel([0.0, 0.0, -0.07], [0,0,0])
# 그립 ON
self.grip_on_6s()
# Z 상승
self.ur5e.go_to_pose_rel([0.0, 0.0, 0.115], [0,0,0])
```

- Meaning: Typical industrial robot pick sequence
- Feature: Reuse of the same structure for all pick operations

4. Grid-based Pick Cycle (Reusable Blocks)

```
def pick_cycle(self, pose_dict, cell_key, grip_mode="off"):
    self.goJ(p["start"])
    self.goJ(p["pick"])
    if grip_mode == "on": self.grip_on_6s()
    else: self.grip_off_6s()
    self.goJ(p["start"])
```

- Meaning: Cell name (e.g., "2-3") → Automatically execute the corresponding joint pose.
- Usage:
  - Move to the grid containing the cell.
  - Later, calculate the relative coordinates and execute Pick or Place.

5. State-based robot motion transition (FSM)

```
if self.state == 1: self.state_1()
elif self.state == 5: self.state_5()
elif self.state == 101: self.state_101()
```

- Meaning: Manage robot motion scenarios as states based on detection results.
- Features:
  - Assembly Process
  - Disassembly Process

This system consists of an industrial robot control logic that combines joint-based movement and Cartesian relative movement and performs pick-and-place operations through an FSM structure.

5. Results(Process Awareness-Based Pick and Place Automation Part, UR5e)

5-1 Results

The test was performed a total of 100 times, and the process time was measured from the moment the stopper contacted the bracket until the stopper opened after the process was completed. The results are as follows.

Process	Goal	Result
Detection	- Bracket detection success rate ≥98% - Cell box detection success rate =100%	- Bracket Detection Success Rate = 100% - Cell Box Detection Success Rate = 100%
Pick and Place	- Pick and Place Success Rate ≥95%	- Pick and Place Success Rate = 97%
Stopper linkage and process completion time	- Stopper Interlock Success Rate = 100% - Assembly/Disassembly Process Completion Time: Within 1 Minute	- Stopper Interlock Success Rate = 100% - Assembly/Disassembly Process Completion Time: 1 minute 35 seconds

5-2 Discussion

1. While the pick-and-place operation met the target performance indicator of a success rate of over 95%, the battery cell assembly and disassembly process requires precise analysis,

as even a single failure can impact the overall process quality. The reason is that the `go_to_joint_rel()` function used in relative coordinate-based control sets the positioning error tolerance to 0.003, allowing for a maximum error of approximately 3 mm. However, indiscriminately reducing this tolerance value improves positioning accuracy, but there is a trade-off: increased path calculation time and overall process delays. Therefore, more detailed adjustments to the tolerance value are necessary to simultaneously satisfy the required process accuracy and computational efficiency.

2. The target process time of less than one minute was not met, requiring further analysis in this area. The Cartesian technique, which directly specifies the spatial coordinates (X, Y, Z) of the robot and moves it, has no risk of sign inversion of the posture (yaw, pitch, roll) and can secure high precision because the path is calculated by dividing it into intervals of about 2 mm. However, it has the limitation of relatively slow process speed. On the other hand, the Euler technique, which directly controls the posture (yaw, pitch, roll) of the end-effector, has fast movement speed, but a serious problem has been observed in which the sign inversion of the posture occurs in certain situations and the robot posture suddenly becomes distorted, which acts as a clear trade-off. In addition, a delay of about 1.5 seconds occurred in the process of calling the detection node in a callback manner, which is believed to be caused by the ROS communication structure itself rather than the delay caused by `rospy.sleep()`.
3. Since the image processing was performed based on the lighting standard of NTH 115, detection and pick and place may become more inaccurate when exposed to other lighting environments.
4. If neither the Cell nor the Composite box contains a single cell, the ur5e robot will continue to switch between process states. Therefore, at least one cell must be present in each box.
5. The Cell box must have a Cell in its proper location, and the Composite box must have a Composite Cell in its proper location. Otherwise, the cell will not be recognized.

6. Conclusion

By introducing a process-aware pick and place automation system, operating hours can be significantly extended compared to existing human-driven processes, enabling continuous 24-hour operation. While the cycle time was somewhat long at 1 minute and 35 seconds, and only two cells were picked and placed per cycle, this project demonstrated the feasibility of fully unmanned pick and place, suggesting potential savings in labor and operating costs.
Future research will focus on shortening the overall process execution time and enhancing the precision of image processing detection to ensure a near-zero pick and place failure rate.

7. Reference

1. 박예나. (2024년 11월 17일). “배터리 삼국지”… 中 질주 속 K배터리의 난제는. 서울경제. https://www.sedaily.com/NewsView/29X4QWC74S

2. Attia, P. M., Moch, E., & Herring, P. K. (2025). Challenges and opportunities for high-quality battery production at scale. Nature Communications, 16, 611. https://doi.org/10.1038/s41467-025-55861-7
3. Jiang, X., Gramopadhye, A.K., Melloy, B.J. and Grimes, L.W. (2003), Evaluation of best system performance: Human, automated, and hybrid inspection systems. Hum. Factors Man., 13: 137-152. https://doi.org/10.1002/hfm.10031
4. Jo, H., Park, H., Baek, S., & Kang, E. K. (2017). Low back pain in farmers: The association with agricultural work management, disability, and quality of life in Korean farmers. Human Factors and Ergonomics in Manufacturing & Service Industries, 27(3), 156–165. https://doi.org/10.1002/hfm.20699

8. Code

8-1. Structure

```
catkin_ws
    |- build
    |- devel
    |- packages_from_git
    |- src
    |----|- CMakeLists.txt
    |----|- 22000167_Kiminyeop_ur_python
    |----|----|- msg
    |----|----|----| grip_command.msg
    |----|----|----| grip_state.msg
    |----|----|----| object_info.msg
    |----|----|----| pet_info.msg
    |----|----|----| robot_state.msg
    |----|----|- src
    |----|----|----| camera.py
    |----|----|----| cell_detection.py
    |----|----|----| composite_detection.py
    |----|----|----| empty_cell_box_detection_node.py
    |----|----|----| empty_composite_cellbox_detection_node.py
    |----|----|----| filled_cellbox_detection_node.py
    |----|----|----| filled_composite_cellbox_detection_node.py
    |----|----|----| main.py
    |----|----|----| move_group_python_interface.py
    |----|----|----| winding_detect_node.py
    |----|----|----| yellow_bracket_detection_node.py
```

`my_robot_calibration.yaml` will add to the home

8-2. Command

1. If you saved my_robot_calibration.yaml to your home directory, run the following. If the path doesn't exist, you'll need to reset it to match your computer's path.

```
chmod+x/~my_robot_calibration.yaml
```

2. Since the project is based on ROS environment, every command line is done under catkin workspace directory.

```
cd~/catkin_ws
catkin_make
roscore
```

3. To activate the system, the robot should be connected to a computer. Communication protocol is TCP/IP based therefore, IP should be set in proper manner

```
roslaunch ur_robot_driver ur5e_bringup.launch robot_ip:=192.168.0.2
roslaunch ur5e_rg2_moveit_config move_group.launch
```

4. Connect camara in UR5e

```
chmod +x ~/catkin_ws/src/ur_python/src/camera.py
rosrun 22000167_Kiminyeop_ur_python camera.py
```

5. Follow the steps below

- Empty Bracket Detection: This detection node detects yellow bracket holes.

```
chmod +x ~/catkin_ws/src/ur_python/src/yellow_bracket_detection_node.py
rosrun 22000167_Kiminyeop_ur_python yellow_bracket_detection_node.py
```

- Cell Box Cell Detection: This detection node detects cells in empty cell boxes.

```
chmod +x ~/catkin_ws/src/ur_python/src/filled_cellbox_detection_node.py
rosrun 22000167_Kiminyeop_ur_python filled_cellbox_detection_node.py
```

- Empty Completed Box Detection: This detection node detects empty boxes in completed cell boxes.

```
chmod +x
~/catkin_ws/src/ur_python/src/empty_composite_cellbox_detection_node.py
rosrun 22000167_Kiminyeop_ur_python empty_composite_cellbox_detection_node.py
```

- Completed Cell Box Detection: This detection node detects cells in a completed cell box.

```
chmod +x
~/catkin_ws/src/ur_python/src/filled_composite_cellbox_detection_node.py
rosrun 22000167_Kiminyeop_ur_python filled_composite_cellbox_detection_node.py
```

- Empty Cell Box Detection: This detection node detects empty boxes in an empty cell box.

```
chmod +x ~/catkin_ws/src/ur_python/src/empty_cell_cellbox_detection_node.py
rosrun 22000167_Kiminyeop_ur_python empty_cell_cellbox_detection_node.py
```

- Orange Winding Detection: This detection node detects orange windings.

```
chmod +x ~/catkin_ws/src/ur_python/src/winding_detect_node.py
rosrun 22000167_Kiminyeop_ur_python winding_detect_node.py
```

- Completed Cell Detection: This detection node detects completed cells.

```
chmod +x ~/catkin_ws/src/ur_python/src/composite_detection.py
rosrun 22000167_Kiminyeop_ur_python composite_detection.py
```

8. Cell Detection: This detection node detects empty cells.

```
chmod +x ~/catkin_ws/src/ur_python/src/cell_detection.py
rosrun 22000167_Kiminyeop_ur_python cell_detection.py
```

- Run Arduino Stopper
  After installing Arduino 1.8.19, confirm that the board is Arduino Uno in the tool bar and that the port is dev/tty/ACM0. Confirm and upload, then close the Arduino window.

10. Enter VScode and run main.py.

8-3 Troubleshooting

1. If the robot movement doesn't work, it's almost always due to a problem with

```
roslaunch ur_robot_driver ur5e_bringup.launch robot_ip:=192.168.0.2
```

Therefore, undoing and re-running this terminal should resolve the issue.

2. If you run the code

```
chmod +x ~/catkin_ws/src/ur_python/src/camera.py
rosrun 22000167_Kiminyeop_ur_python camera.py
```

the camera may not be connected to the ur5e robot but to the laptop. In this case, adjust the numeric value of `self.cap = cv2.VideoCapture(2)`
from 0 to 5 and then run the program.

8-4 Code

`my_robot_calibration.yaml`

```
kinematics:
  shoulder:
    x: 0
    y: 0
    z: 0.1627122445743823
    roll: -0
    pitch: 0
    yaw: 5.707382499074709e-08
  upper_arm:
    x: 0.0001820110310317555
    y: 0
    z: 0
    roll: 1.571314866107557
    pitch: 0
    yaw: 1.12157791523139e-06
  forearm:
    x: -0.4251703446314444
    y: 0
    z: 0
    roll: 0.0014443205793765 47
    pitch: -0.0004261037597356073
    yaw: 4.642343028998144e-06
  wrist_1:
    x: -0.392646598028507
    y: -0.0004030322688786401
    z: 0.1331063119537431
    roll: 0.003027888244224887
```

```
    pitch: -0.001185722601572754
    yaw: -2.569771029137183e-06
  wrist_2:
    x: -7.071306960874342e-06
    y: -0.09960359635829828
    z: -3.198591184007791e-07
    roll: 1.570799538115874
    pitch: 0
    yaw: 1.304820282092834e-06
  wrist_3:
    x: 1.369362273278849e-05
    y: 0.09958483828527664
    z: 9.521789789182917e-05
    roll: 1.571752475045119
    pitch: 3.141592653589793
    yaw: 3.141592510837143
  hash: calib_14698139022496758595
```

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.0.2)
project(ur_python)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  rospy
  message_generation
  std_msgs
  message_runtime
  moveit_ros_planning
  moveit_ros_planning_interface
  moveit_visual_tools
  geometry_msgs
  image_transport
  cv_bridge
  sensor_msgs
)

find_package (OpenCV REQUIRED)

add_message_files(
```

```
  FILES
  object_info.msg
  robot_state.msg
  grip_command.msg
  grip_state.msg
  pet_info.msg
)

generate_messages(
  DEPENDENCIES
  std_msgs
  geometry_msgs
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a exec_depend tag for each package in
MSG_DEP_SET
##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##     * add a exec_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEP_SET to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEP_SET to
##     catkin_package(CATKIN_DEPENDS ...)
##   * uncomment the add_*_files sections below as needed
##     and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
```

```
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
```

```
#   cfg/DynReconf2.cfg
# )

####################################
## catkin specific configuration ##
####################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects
also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also
need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES ur_python
#  CATKIN_DEPENDS rospy std_msgs
#  DEPENDS system_lib
    LIBRARIES opencv
    CATKIN_DEPENDS
    cv_bridge
    image_transport
    sensor_msgs
    std_msgs
)

###########
## Build ##
###########

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
  ${OpenCV_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/ur_python.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
```

```
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't
collide
# add_executable(${PROJECT_NAME}_node src/ur_python_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following
renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg
someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node
PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables for installation
## See
http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_executables
```

```
.html
# install(TARGETS ${PROJECT_NAME}_node
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
## See
http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_libraries.h
tml
# install(TARGETS ${PROJECT_NAME}
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

##############
## Testing ##
##############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_ur_python.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

`camera.py`

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-
```

```python
import rospy
import cv2
from sensor_msgs.msg import Image                # sensor_msg 패키지로부터 Image type을 import함
from cv_bridge import CvBridge, CvBridgeError    # cv_bridge 라이브러리 : OpenCV 이미지와 ROS 메시지 간의 변환 가능

class CameraNode:

    def __init__(self):
        self.bridge = CvBridge()                 # cv_bridge 객체 생성
        self.image_pub = rospy.Publisher("camera/image_raw",Image,queue_size=1)    # "camera/image_raw"라는 토픽으로 메시지를 publish할 publisher 객체 생성
        self.cap = cv2.VideoCapture(2)           # 카메라 연결을 위한 VideoCapture 객체 생성

    def run(self):
        rospy.init_node('camera_node', anonymous=True)  # 노드 이름 "camera_node"로 초기화
        rate = rospy.Rate(30)                           # 루프 실행 주기 : 30hz
        while not rospy.is_shutdown():                  # ROS가 종료되지 않은 동안
            ret, frame = self.cap.read()                # 카메라로부터 이미지를 읽음
            if ret:                                     # 이미지가 정상적으로 읽혀진 경우
                try:
                    # 읽어들인 이미지를 ROS Image 메시지로 변환하여 토픽으로 publish
                    self.image_pub.publish(self.bridge.cv2_to_imgmsg(frame, "bgr8"))

                except CvBridgeError as e:
                    print(e)                            # CvBridge 변환 예외 처리
            rate.sleep()                                # 지정된 루프 실행 주기에 따라 대기

if __name__ == '__main__':
    try:
        camera = CameraNode()       # CameraNode 객체 생성
        camera.run()                # run 메서드 실행
    except rospy.ROSInterruptException:
        pass
```

cell_detection.py

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np, os, math
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from geometry_msgs.msg import Point

class StableGrayDetectionNode:
    def __init__(self):
        rospy.init_node('stable_gray_detection_node', anonymous=True)
        self.bridge = CvBridge()
        self.detection_pub = rospy.Publisher('/grid1_fallback/result', Point,
queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # ============================================================
        # 🧩 사용자 조정 파라미터
        # ============================================================
        self.ROI_WIDTH = 400
        self.ROI_HEIGHT = 400
        self.TARGET_SIZE = (1200, 1200)
        self.CENTER_PX = (600, 600)
        self.DETECTION_OFFSET = 300

        # 픽셀 → mm 변환
        self.scale_x = 0.13
        self.scale_y = 0.13

        # 카메라 오프셋
        self.CAMERA_OFFSET_X_MM = 88.0
        self.CAMERA_OFFSET_Y_MM = 3.0

        # 반지름 필터
        self.MIN_RADIUS = 170
        self.MAX_RADIUS = 250
        self.CENTER_TOL_MM = 10.0

        # Gray 톤 필터
        self.LOWER_GRAY = 140
        self.UPPER_GRAY = 255

        # Hough 파라미터
```

```
        self.HOUGH_DP = 1
        self.HOUGH_MIN_DIST = 100
        self.HOUGH_PARAM1 = 50
        self.HOUGH_PARAM2 = 20

        # EMA 필터
        self.prev_center = None
        self.prev_radius = None
        self.alpha = 0.35

        # 🟠 주황색 감지 파라미터 (반대 조건)
        self.ORANGE_RATIO_THRESH = 0.03  # 3% 이상이면 무시
        self.ORANGE_MIN = (50, 80, 240)
        self.ORANGE_MAX = (120, 150, 255)

        # 🟡 노란색 감지 파라미터 (너가 지정한 EXACT 값)
        self.YELLOW_RATIO_THRESH = 0.03

        self.lower_yellow1 = np.array([20, 120, 120])
        self.upper_yellow1 = np.array([32, 255, 255])

        self.lower_yellow2 = np.array([0, 0, 0])
        self.upper_yellow2 = np.array([0, 0, 0])

        # ============================================================
        # 📂 카메라 캘리브레이션
        # ============================================================
        calib_path =
"/home/kiminyeop/catkin_ws/src/ur_python/src/dataset_bra2_checkboard3/circlegr
id_calibration_results_8x6_20mm.npz"
        if os.path.exists(calib_path):
            data = np.load(calib_path)
            self.camera_matrix = data["camera_matrix"]
            self.dist_coeffs = data["dist_coeffs"]
            rospy.loginfo(f"📷 Loaded camera calibration from: {calib_path}")
        else:
            rospy.logwarn("⚠️ Calibration file not found — skipping
undistortion.")
            self.camera_matrix, self.dist_coeffs = None, None

        rospy.loginfo("🟣 Stable Gray Detection Node initialized (Gray only,
ignore Orange+Yellow)")

    # ------------------------------------------------------------
    def image_callback(self, msg):
        try:
```

```
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            if self.camera_matrix is not None:
                frame = cv2.undistort(frame, self.camera_matrix,
self.dist_coeffs)

            H, W = frame.shape[:2]
            x1 = int(W/2 - self.ROI_WIDTH/2)
            y1 = int(H/2 - self.ROI_HEIGHT/2)
            cropped = frame[y1:y1+self.ROI_HEIGHT, x1:x1+self.ROI_HEIGHT]
            resized = cv2.resize(cropped, self.TARGET_SIZE,
interpolation=cv2.INTER_LINEAR)

            self.detect_gray_circle(resized)

        except Exception as e:
            rospy.logerr(f"Gray Detection Error: {e}")

    # ------------------------------------------------------------
    def detect_gray_circle(self, frame):
        output = frame.copy()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray_filtered = cv2.inRange(gray, self.LOWER_GRAY, self.UPPER_GRAY)
        gray_blurred = cv2.GaussianBlur(gray_filtered, (9, 9), 2)

        # === HSV 변환 (노란색 검출을 위해 추가)
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        OFFSET = self.DETECTION_OFFSET
        HOUGH_SIZE = 600
        HOUGH_INPUT = gray_blurred[OFFSET:OFFSET + HOUGH_SIZE, OFFSET:OFFSET +
HOUGH_SIZE]

        circles = cv2.HoughCircles(
            HOUGH_INPUT, cv2.HOUGH_GRADIENT,
            self.HOUGH_DP, self.HOUGH_MIN_DIST,
            param1=self.HOUGH_PARAM1, param2=self.HOUGH_PARAM2,
            minRadius=self.MIN_RADIUS, maxRadius=self.MAX_RADIUS
        )

        best_circle_data = None
        min_dist_to_center = float('inf')

        # 🔹 가장 중앙에 가까운 원 선택
        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
```

```
                x_rel, y_rel, r = i[0], i[1], i[2]
                x = x_rel + OFFSET
                y = y_rel + OFFSET
                dx_px = float(x - self.CENTER_PX[0])
                dy_px = float(y - self.CENTER_PX[1])
                dist_px = math.sqrt(dx_px**2 + dy_px**2)
                if dist_px < min_dist_to_center:
                    min_dist_to_center = dist_px
                    best_circle_data = (x, y, r, 1.0)

        # 🔸 결과 처리
        if best_circle_data is not None:
            x, y, r, _ = best_circle_data

            # EMA 안정화
            if self.prev_center is None:
                self.prev_center = np.array([x, y], dtype=float)
                self.prev_radius = r
            else:
                self.prev_center = (1 - self.alpha) * self.prev_center +
self.alpha * np.array([x, y])
                self.prev_radius = (1 - self.alpha) * self.prev_radius +
self.alpha * r

            x, y = self.prev_center
            r = self.prev_radius

            # === 🔴 원 내부 오렌지 비율 계산 ===
            mask = np.zeros(frame.shape[:2], dtype=np.uint8)
            cv2.circle(mask, (int(x), int(y)), int(r), 255, -1)
            orange_mask = cv2.inRange(frame, self.ORANGE_MIN, self.ORANGE_MAX)
            orange_inside = cv2.bitwise_and(orange_mask, orange_mask,
mask=mask)
            orange_ratio = np.sum(orange_inside > 0) / np.sum(mask > 0)
            rospy.loginfo_throttle(1.0, f"🍊 Orange ratio inside circle:
{orange_ratio*100:.2f}%")

            # === 🟡 원 내부 노란색 비율 계산 (너가 지정한 HSV 기준 사용) ===
            yellow_mask1 = cv2.inRange(hsv, self.lower_yellow1,
self.upper_yellow1)
            yellow_mask2 = cv2.inRange(hsv, self.lower_yellow2,
self.upper_yellow2)
            yellow_mask = cv2.bitwise_or(yellow_mask1, yellow_mask2)

            yellow_inside = cv2.bitwise_and(yellow_mask, yellow_mask,
mask=mask)
```

```
            yellow_ratio = np.sum(yellow_inside > 0) / np.sum(mask > 0)
            rospy.loginfo_throttle(1.0, f"🟡 Yellow ratio inside circle:
{yellow_ratio*100:.2f}%")

            # === 픽셀 → mm 변환 ===
            dx_px = float(x - self.CENTER_PX[0])
            dy_px = float(y - self.CENTER_PX[1])
            dx_mm = dx_px * self.scale_x
            dy_mm = dy_px * self.scale_y
            dist_mm = math.sqrt(dx_mm**2 + dy_mm**2)

            # === 카메라 오프셋 보정 ===
            dx_mm_total = -dy_mm + self.CAMERA_OFFSET_X_MM
            dy_mm_total = -dx_mm + self.CAMERA_OFFSET_Y_MM

            # ✅ 회색 원은 감지 + 주황색 OR 노란색 > threshold 이면 무시
            if (
                dist_mm <= self.CENTER_TOL_MM and
                orange_ratio <= self.ORANGE_RATIO_THRESH and
                yellow_ratio <= self.YELLOW_RATIO_THRESH
            ):
                cv2.circle(output, (int(x), int(y)), int(r), (0,255,0), 2)
                cv2.circle(output, (int(x), int(y)), 5, (0,0,255), -1)
                cv2.circle(output, self.CENTER_PX, 5, (255,0,0), -1)

                cv2.putText(output, f"dx={dx_mm:.1f} dy={dy_mm:.1f} (OK gray
only)",
                            (25,45), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,0),
2)
                cv2.putText(output, f"XOFF -> dx={dx_mm_total:.1f} dy=
{dy_mm_total:.1f}",
                            (25,85), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
(255,255,0), 2)

                self.detection_pub.publish(Point(x=float(dx_mm_total),
y=float(dy_mm_total), z=float(r)))
                rospy.loginfo_throttle(1.0, f"✅ Valid Gray (no
orange/yellow): dx={dx_mm_total:.1f}, dy={dy_mm_total:.1f}")

            else:
                rospy.logwarn_throttle(1.0,
                    f"⚠️ Ignored (dist={dist_mm:.1f}mm, "
                    f"orange={orange_ratio*100:.1f}%, yellow=
{yellow_ratio*100:.1f}%)")
                self._publish_fail()
```

```
        else:
            rospy.logwarn_throttle(1.0, "❌ No circle found.")
            self._publish_fail()

        # 시각화
        cv2.rectangle(output, (OFFSET, OFFSET), (OFFSET + HOUGH_SIZE, OFFSET +
HOUGH_SIZE), (255, 0, 0), 2)
        cv2.imshow("Stable Gray Detection (Gray only, Ignore Orange +
Yellow)", output)
        cv2.imshow("Hough Filtered Input", gray_blurred)
        cv2.imshow("Orange Mask Inside Circle", orange_inside)
        cv2.imshow("Yellow Mask Inside Circle", yellow_inside)
        cv2.waitKey(1)

    # ------------------------------------------------------------
    def _publish_fail(self):
        self.detection_pub.publish(Point(x=0.0, y=0.0, z=0.0))

# ------------------------------------------------------------
if __name__ == '__main__':
    try:
        StableGrayDetectionNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        cv2.destroyAllWindows()
```

composite_detection.py

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np, os, math
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from geometry_msgs.msg import Point

class StableGrayDetectionNode:
    def __init__(self):
        rospy.init_node('stable_gray2_detection_node', anonymous=True)
        self.bridge = CvBridge()
        self.detection_pub = rospy.Publisher('/grid2_fallback/result', Point,
queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # ============================================================
```

```
        # 🧩 사용자 조정 파라미터 (Hough에 맞게 조정)
        # ============================================================
        self.ROI_WIDTH = 400
        self.ROI_HEIGHT = 400
        self.TARGET_SIZE = (1200, 1200)
        self.CENTER_PX = (600, 600)
        self.DETECTION_OFFSET = 300

        # 픽셀 → mm 변환
        self.scale_x = 0.13
        self.scale_y = 0.13

        # 카메라 오프셋
        self.CAMERA_OFFSET_X_MM = 88.0
        self.CAMERA_OFFSET_Y_MM = 3.0

        # 반지름 필터 (픽셀)
        self.MIN_RADIUS = 170
        self.MAX_RADIUS = 250

        # 중심 허용 거리 (mm)
        self.CENTER_TOL_MM = 20.0

        # Gray 톤 필터링 범위
        self.LOWER_GRAY = 140
        self.UPPER_GRAY = 255

        # Hough 파라미터
        self.HOUGH_DP = 1
        self.HOUGH_MIN_DIST = 100
        self.HOUGH_PARAM1 = 50
        self.HOUGH_PARAM2 = 20

        # EMA 필터
        self.prev_center = None
        self.prev_radius = None
        self.alpha = 0.35

        # ============================================================
        # 📂 카메라 캘리브레이션
        # ============================================================
        calib_path =
"/home/kiminyeop/catkin_ws/src/ur_python/src/dataset_bra2_checkboard3/circlegr
id_calibration_results_8x6_20mm.npz"
        if os.path.exists(calib_path):
            data = np.load(calib_path)
```

```
            self.camera_matrix = data["camera_matrix"]
            self.dist_coeffs = data["dist_coeffs"]
            rospy.loginfo(f"📷 Loaded camera calibration from: {calib_path}")
        else:
            rospy.logwarn("⚠️ Calibration file not found — skipping
undistortion.")
            self.camera_matrix, self.dist_coeffs = None, None

        rospy.loginfo("🟣 Stable Gray Detection Node initialized (Hough
Transform, Limited ROI)")

    # ---------------------------------------------------------
    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            if self.camera_matrix is not None:
                frame = cv2.undistort(frame, self.camera_matrix,
self.dist_coeffs)

            H, W = frame.shape[:2]
            x1 = int(W/2 - self.ROI_WIDTH/2)
            y1 = int(H/2 - self.ROI_HEIGHT/2)
            cropped = frame[y1:y1+self.ROI_HEIGHT, x1:x1+self.ROI_HEIGHT]
            resized = cv2.resize(cropped, self.TARGET_SIZE,
interpolation=cv2.INTER_LINEAR)

            self.detect_gray_circle(resized)

        except Exception as e:
            rospy.logerr(f"Gray Detection Error: {e}")

    # ---------------------------------------------------------
    def detect_gray_circle(self, frame):
        output = frame.copy()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray_filtered = cv2.inRange(gray, self.LOWER_GRAY, self.UPPER_GRAY)
        gray_blurred = cv2.GaussianBlur(gray_filtered, (9, 9), 2)

        OFFSET = self.DETECTION_OFFSET
        HOUGH_SIZE = 600
        HOUGH_INPUT = gray_blurred[OFFSET:OFFSET + HOUGH_SIZE, OFFSET:OFFSET +
HOUGH_SIZE]

        circles = cv2.HoughCircles(
            HOUGH_INPUT, cv2.HOUGH_GRADIENT,
            self.HOUGH_DP, self.HOUGH_MIN_DIST,
```

```
            param1=self.HOUGH_PARAM1, param2=self.HOUGH_PARAM2,
            minRadius=self.MIN_RADIUS, maxRadius=self.MAX_RADIUS
        )

        best_circle_data = None
        min_dist_to_center = float('inf')

        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
                x_rel, y_rel, r = i[0], i[1], i[2]
                x = x_rel + OFFSET
                y = y_rel + OFFSET
                dx_px = float(x - self.CENTER_PX[0])
                dy_px = float(y - self.CENTER_PX[1])
                dist_px = math.sqrt(dx_px**2 + dy_px**2)
                if dist_px < min_dist_to_center:
                    min_dist_to_center = dist_px
                    best_circle_data = (x, y, r)

        if best_circle_data is not None:
            x, y, r = best_circle_data

            # EMA 안정화
            if self.prev_center is None:
                self.prev_center = np.array([x, y], dtype=float)
                self.prev_radius = r
            else:
                self.prev_center = (1 - self.alpha) * self.prev_center +
self.alpha * np.array([x, y])
                self.prev_radius = (1 - self.alpha) * self.prev_radius +
self.alpha * r

            x, y = self.prev_center
            r = self.prev_radius

            # 픽셀 → mm 변환
            dx_px = float(x - self.CENTER_PX[0])
            dy_px = float(y - self.CENTER_PX[1])
            dx_mm = dx_px * self.scale_x
            dy_mm = dy_px * self.scale_y
            dist_mm = math.sqrt(dx_mm**2 + dy_mm**2)

            # 오프셋 적용
            dx_mm_total = -dy_mm + self.CAMERA_OFFSET_X_MM
            dy_mm_total = -dx_mm + self.CAMERA_OFFSET_Y_MM
```

```
            if dist_mm <= self.CENTER_TOL_MM:
                # 시각화
                cv2.circle(output, (int(x), int(y)), int(r), (0,255,0), 2)
                cv2.circle(output, (int(x), int(y)), 5, (0,0,255), -1)
                cv2.circle(output, self.CENTER_PX, 5, (255,0,0), -1)

                # --- ✅ 텍스트 출력 (raw + XOFFSET 적용)
                cv2.putText(output, f"dx={dx_mm:.1f} dy={dy_mm:.1f}",
                            (25,45), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,0),
2)
                cv2.putText(output, f"XOFF -> dx={dx_mm_total:.1f} dy=
{dy_mm_total:.1f}",
                            (25,85), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
(255,255,0), 2)

                # 퍼블리시
                self.detection_pub.publish(Point(x=float(dx_mm_total),
y=float(dy_mm_total), z=float(r)))
                rospy.loginfo_throttle(1.0, f"📍 Valid Circle: dx=
{dx_mm_total:.1f}, dy={dy_mm_total:.1f}")
            else:
                rospy.logwarn_throttle(1.0, f"⚠️ Circle ignored (dist=
{dist_mm:.1f}mm > {self.CENTER_TOL_MM}mm)")
                self._publish_fail()
        else:
            rospy.logwarn_throttle(1.0, "❌ No circle found in Hough area.")
            self._publish_fail()

        # 시각화
        cv2.rectangle(output, (OFFSET, OFFSET), (OFFSET + HOUGH_SIZE, OFFSET +
HOUGH_SIZE), (255, 0, 0), 2)
        cv2.imshow("Stable Gray2 Detection (Hough Transform, Limited ROI)",
output)
        cv2.imshow("Gray2 Hough Filtered Input", gray_blurred)
        cv2.waitKey(1)

    # ------------------------------------------------------------
    def _publish_fail(self):
        self.detection_pub.publish(Point(x=0.0, y=0.0, z=0.0))

# ------------------------------------------------------------
if __name__ == '__main__':
    try:
        StableGrayDetectionNode()
        rospy.spin()
```

```
    except rospy.ROSInterruptException:
        cv2.destroyAllWindows()
```

empty_cell_cellbox_detection_node.py

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from std_msgs.msg import String # 발행할 메시지 타입

class InverseGrayDetectionNode:
    def __init__(self):
        rospy.init_node('gray_detection_node', anonymous=True)
        self.bridge = CvBridge()
        self.detection_pub = rospy.Publisher('/inverse_gray_detection_result',
String, queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # === 그리드 설정 ===
        self.P1, self.P2, self.P3 = (105,77), (547,73), (105,433)
        self.cols, self.rows = 4, 3
        self.active_cells = {(1,1),(1,3),(2,2),(2,4),(3,1),(3,3)}
        self.gray_history = [] # 안정화용

        # === 🚨 [수정] 검정색 픽셀 감지 임계값 (HSV 사용) ===
        # 검정색: V(명도)가 낮음 (V <= 60)
        self.lower_black, self.upper_black = np.array([0, 0, 0]),
np.array([150, 150, 150])

        # 🚨 [수정] 검정색 픽셀의 최소 면적 비율 (물체가 있다고 판단하는 기준)
        self.BLACK_PIXEL_RATIO = 0.77     # 감지 최소 면적 비율 (90% 이상일 때
DETECTED)

        # 📌 우선순위 리스트 (문자열 정렬 순서로 가장 높은 우선순위 결정)
        self.priority_list = sorted([f"{r}-{c}" for r, c in
self.active_cells])

    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            result_frame = self.black_area_detect(frame) # 함수 이름
'black_area_detect'로 변경
```

```
            cv2.imshow("Black Area Grid Cell Detection", result_frame)
            cv2.waitKey(1)
        except Exception as e:
            rospy.logerr(f"Detection Error: {e}")

    def black_area_detect(self, frame):
        display_frame = frame.copy()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) # HSV 변환

        xs = np.linspace(self.P1[0], self.P2[0], self.cols+1)
        ys = np.linspace(self.P1[1], self.P3[1], self.rows+1)
        detected_black = set() # 감지된 '검정색 물체' 셀 목록

        for i in range(self.rows):
            for j in range(self.cols):
                r, c = i+1, j+1
                x1, x2 = int(xs[j]), int(xs[j+1])
                y1, y2 = int(ys[i]), int(ys[i+1])

                # 영역 유효성 검사
                if y2 <= y1 or x2 <= x1:
                    continue

                cell_roi = hsv[y1:y2, x1:x2]
                cell_area = cell_roi.shape[0] * cell_roi.shape[1]

                # 1. HSV 임계값을 사용하여 검정색 픽셀 마스크 생성
                black_mask = cv2.inRange(cell_roi, self.lower_black,
self.upper_black)

                # 2. 검정색 픽셀의 개수 및 비율 계산
                black_pixel_count = np.sum(black_mask == 255)
                current_ratio = black_pixel_count / cell_area if cell_area > 0
else 0

                cell_name = f"{r}-{c}"
                color = (80,80,80) # 기본 회색 (비활성)
                text = f"{cell_name} ({current_ratio:.2f})"

                # 📌 시각화는 원본 BGR 프레임 복사본(display_frame)에 수행
                if (r,c) in self.active_cells:

                    # 🚨 [감지 조건] 검정색 픽셀 비율이 BLACK_PIXEL_RATIO 이상일 때
(DETECTED)
                    if current_ratio >= self.BLACK_PIXEL_RATIO:
                        detected_black.add(cell_name)
```

```
                        color = (255,165,0) # 주황색 (물체 감지)
                        text = f"DETECTED ({current_ratio:.2f})"
                    else:
                        # 🚨 [미감지] 검정색 물체가 없다고 판단 (NOT DETECTED / EMPTY)
                        color = (0, 255, 0) # 녹색 (물체 없음)
                        text = f"EMPTY ({current_ratio:.2f})"

                    # 📌 활성 셀 테두리 및 텍스트 표시
                    cv2.rectangle(display_frame, (x1, y1), (x2, y2), color, 2)
                    cx, cy = int((x1+x2)/2), int((y1+y2)/2)
                    cv2.putText(display_frame, text, (cx - 60, cy),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255), 2)
                else:
                    # 📌 비활성 셀 테두리 표시
                    cv2.rectangle(display_frame, (x1, y1), (x2, y2), color, 1)


        # 📌 5프레임 안정화 및 우선순위가 가장 높은 셀 하나만 발행하는 로직
        # 감지된 '검정색 물체' 셀들의 목록을 우선순위에 따라 정렬
        detected_list = sorted(list(detected_black), key=lambda x: self.priority_list.index(x) if x in self.priority_list else len(self.priority_list))

        is_published = False

        # '검정색 물체'가 감지된 경우
        if detected_list:
            highest_priority_cell = detected_list[0]
            # 히스토리에 (최우선 순위 셀) 문자열을 포함하는 frozenset을 저장
            self.gray_history.append(frozenset({highest_priority_cell}))
        else:
            self.gray_history.append(frozenset()) # 감지된 셀이 없으면 빈 세트 추가

        if len(self.gray_history) > 5: self.gray_history.pop(0)

        # 5프레임 동안 감지된 셀이 존재하고, 그 셀이 모두 동일한 최우선 순위 셀일 경우
        if (len(self.gray_history) == 5 and
            all(h == self.gray_history[0] for h in self.gray_history) and
            len(self.gray_history[0]) == 1):

            # 최우선 순위 셀의 문자열을 추출하여 발행
            result_str = list(self.gray_history[0])[0]
            self.detection_pub.publish(String(result_str))
```

```
            rospy.loginfo(f"✨ Black Object Stable Detection Published
(Highest Priority): {result_str}")
            self.gray_history.clear()
            is_published = True

        # 🚨 [추가] 안정화된 감지에 실패했을 경우 터미널에 로그 출력
        if not is_published:
            rospy.logwarn(f"🚧 Black Object NOT DETECTED (History len:
{len(self.gray_history)}).")

        return display_frame

if __name__ == '__main__':
    try:
        node = InverseGrayDetectionNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
    finally:
        cv2.destroyAllWindows()
```

empty_composite_cellbox_detection_node.py

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from std_msgs.msg import String

class DarkGridDetectionNode:
    def __init__(self):
        rospy.init_node('dark_grid_detection_node', anonymous=True)
        self.bridge = CvBridge()

        self.detection_pub = rospy.Publisher('/darkgrid_detection_result',
String, queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # 📌📌 왜곡 보정 파라미터 (Camera Calibration 결과) 📌📌
        # !!! 경고: 이 값들은 예시입니다. 실제 카메라 캘리브레이션 값으로 반드시 변경해
야 합니다. !!!
        # Camera Matrix (K)
        self.camera_matrix = np.array([
```

```
            [600.0, 0.0, 640.0],    # fx, 0, cx
            [0.0, 600.0, 360.0],    # 0, fy, cy
            [0.0, 0.0, 1.0]
        ], dtype=np.float64)

        # Distortion Coefficients (D): k1, k2, p1, p2, k3...
        self.dist_coeffs = np.array([
            0.1, -0.05, 0.001, 0.001, 0.0
        ], dtype=np.float64)

        # === 감지 파라미터 ===
        self.dark_intensity_thresh = 90
        # 🚨 경고: 이 좌표들은 왜곡 보정된 이미지에 맞춰 다시 측정하고 변경해야 합니다.
🚨
        self.P1, self.P2, self.P3 = (105,77), (547,73), (105,433)
        self.cols, self.rows = 4, 3
        # 로봇이 Place 가능한 셀만 활성화 셀로 정의하는 것이 좋습니다.
        self.active_cells = {(1,1),(1,3),(2,2),(2,4),(3,1),(3,3)}
        self.gray_history = []
        rospy.loginfo("--- 🖤 Dark Grid Detection Node 초기화 완료. ---")

    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")

            # 1. 🚨 왜곡 보정 (Undistortion) 🚨
            processed_frame = frame

            # 파라미터가 유효한지 확인하고 왜곡 보정 수행
            if self.camera_matrix is not None and self.dist_coeffs is not
None:
                h, w = frame.shape[:2]

                # 최적의 새 카메라 행렬 계산
                new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(
                    self.camera_matrix, self.dist_coeffs, (w, h), 1, (w, h)
                )

                # 왜곡 보정 적용
                undistorted_frame = cv2.undistort(frame, self.camera_matrix,
                                                  self.dist_coeffs, None,
new_camera_matrix)

                # 왜곡 보정 후 ROI(관심 영역) 자르기
                x, y, w_roi, h_roi = roi
                if w_roi > 0 and h_roi > 0:
```

```
                    processed_frame = undistorted_frame[y:y+h_roi, x:x+w_roi]
                else:
                    processed_frame = undistorted_frame

            # 2. Dark Grid 감지 실행 (보정된 이미지 사용)
            result_frame = self.dark_detect(processed_frame)
            cv2.imshow("Dark Grid Cell Detection", result_frame)
            cv2.waitKey(1)
        except Exception as e:
            rospy.logerr(f"Dark Grid Detection Error: {e}")

    def dark_detect(self, frame):
        # 왜곡 보정 후 이미지의 크기가 원본과 다를 수 있으므로 그레이 변환
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # --- 그리드 좌표 계산 로직 ---
        xs = np.linspace(self.P1[0], self.P2[0], self.cols+1)
        ys = np.linspace(self.P1[1], self.P3[1], self.rows+1)
        detected = set()

        for i in range(self.rows):
            for j in range(self.cols):
                r, c = i+1, j+1
                x1, x2 = int(xs[j]), int(xs[j+1])
                y1, y2 = int(ys[i]), int(ys[i+1])

                # 범위 체크
                if x1 < x2 and y1 < y2 and x1 >= 0 and y1 >= 0 and x2 <=
frame.shape[1] and y2 <= frame.shape[0]:
                    cell_region = gray[y1:y2, x1:x2]

                    if cell_region.size > 0:
                        # 📌 핵심: 평균 밝기 계산
                        mean_intensity = np.mean(cell_region)
                    else:
                        mean_intensity = 255 # 영역이 비었으면 밝은 것으로 간주

                    if (r,c) in self.active_cells:
                        color = (0,255,0)
                        text = f"{mean_intensity:.0f}"

                        # 📌 감지 로직: 평균 밝기가 임계값 이하일 때 (Dark Grid)
                        if mean_intensity < self.dark_intensity_thresh:
                            detected.add(f"{r}-{c}")
                            color = (0,0,255)
                            text = f"Detected ({mean_intensity:.0f})"
```

```
                        cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
                        cx, cy = int((x1+x2)/2), int((y1+y2)/2)
                        cv2.putText(frame, f"{r}-{c} {text}", (cx-60, cy),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255,255,255), 2)
                    else:
                        cv2.rectangle(frame, (x1, y1), (x2, y2), (80,80,80),
1)

        # --- 안정화 로직 ---
        self.gray_history.append(frozenset(detected))
        if len(self.gray_history) > 3:
            self.gray_history.pop(0)

        if (len(self.gray_history) == 3 and
            all(h == self.gray_history[0] for h in self.gray_history) and
            len(detected) > 0):

            # 🚨🚨🚨 수정된 핵심 로직: 감지된 셀 중 첫 번째 셀만 선택하여 발행 🚨🚨
🚨
            stable_detected_set = self.gray_history[0]

            if stable_detected_set:
                # 감지된 유효한 셀들(예: {"1-1", "2-2"})을 사전순으로 정렬하여 첫 번
째 셀만 선택
                # 이렇게 하면 컨트롤러가 예상하는 단일 키 형식(예: "1-1")이 발행됩니
다.
                result_str = sorted(list(stable_detected_set))[0]

                self.detection_pub.publish(String(result_str))
                rospy.loginfo(f"🖤 Dark Grid Stable Detection Published:
{result_str}")
                self.gray_history.clear()
            # 🚨🚨🚨 수정 끝 🚨🚨🚨

        return frame

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    try:
        node = DarkGridDetectionNode()
        node.run()
    except rospy.ROSInterruptException:
```

```
        pass
    finally:
        cv2.destroyAllWindows()
```

`filled_cellbox_detection_node.py`

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from std_msgs.msg import String # 발행할 메시지 타입

class GrayDetectionNode:
    def __init__(self):
        rospy.init_node('gray_detection_node', anonymous=True)
        self.bridge = CvBridge()
        # ============================================================
        # 📌 토픽 이름
        # ============================================================
        self.detection_pub = rospy.Publisher('/gray_detection_result', String,
queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # === 그리드 설정 ===
        self.P1, self.P2, self.P3 = (105,77), (547,73), (105,433)
        self.cols, self.rows = 4, 3
        # 📌 활성 셀 정의 (순서는 우선순위와 동일): 1-1이 가장 높고, 3-3이 가장 낮습니
다.
        self.active_cell_priority = ["1-1", "1-3", "2-2", "2-4", "3-1", "3-3"]
        self.active_cells = set((int(c.split('-')[0]), int(c.split('-')[1]))
for c in self.active_cell_priority)
        self.gray_history = [] # 안정화용

        # === 새로운 밝은 픽셀 감지 임계값 ===
        self.bright_intensity_thresh = 215 # RGB 채널의 최소값
        self.bright_pixel_ratio = 0.10     # 감지 최소 면적 비율 (10%)

    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            result_frame = self.bright_area_detect(frame)
            cv2.imshow("Bright Area Grid Cell Detection", result_frame)
            cv2.waitKey(1)
```

```
        except Exception as e:
            rospy.logerr(f"Detection Error: {e}")

    def bright_area_detect(self, frame):
        display_frame = frame.copy()

        xs = np.linspace(self.P1[0], self.P2[0], self.cols+1)
        ys = np.linspace(self.P1[1], self.P3[1], self.rows+1)
        detected_cell_names = set() # 감지된 셀 이름 (예: "1-1")

        # 1. 셀 별 밝은 영역 감지
        for i in range(self.rows):
            for j in range(self.cols):
                r, c = i+1, j+1
                cell_name = f"{r}-{c}"
                x1, x2 = int(xs[j]), int(xs[j+1])
                y1, y2 = int(ys[i]), int(ys[i+1])

                if y2 <= y1 or x2 <= x1: continue

                cell_roi = frame[y1:y2, x1:x2]
                cell_area = cell_roi.shape[0] * cell_roi.shape[1]

                # B, G, R 채널 값이 모두 215 이상인 픽셀 찾기
                bright_mask = np.all(cell_roi >= self.bright_intensity_thresh,
axis=2)
                bright_pixel_count = np.sum(bright_mask)
                current_ratio = bright_pixel_count / cell_area if cell_area >
0 else 0

                color = (80,80,80) # 기본 회색 (비활성)
                text = f"{r}-{c} ({current_ratio:.2f})"

                if (r,c) in self.active_cells:
                    color = (0,255,0) # 초록색 (활성 대기)

                    if current_ratio >= self.bright_pixel_ratio:
                        detected_cell_names.add(cell_name) # 감지된 셀 이름 저장
                        color = (255,165,0) # 주황색 (감지됨)
                        text = f"DETECTED ({current_ratio:.2f})"

                    # 활성 셀 테두리 및 텍스트 표시
                    cv2.rectangle(display_frame, (x1, y1), (x2, y2), color, 2)
                    cx, cy = int((x1+x2)/2), int((y1+y2)/2)
                    cv2.putText(display_frame, text, (cx - 60, cy),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255),
```

```
2)
                else:
                    # 비활성 셀 테두리 표시
                    cv2.rectangle(display_frame, (x1, y1), (x2, y2), color, 1)


        # 2. 5프레임 안정화 로직
        self.gray_history.append(frozenset(detected_cell_names))
        if len(self.gray_history) > 5: self.gray_history.pop(0)

        # 5프레임이 채워지고, 모든 프레임에서 같은 결과가 나왔을 때
        if len(self.gray_history) == 5 and all(h == self.gray_history[0] for h
in self.gray_history):

            stable_detected_set = self.gray_history[0]
            result_str = ""

            if len(stable_detected_set) > 0:
                # 📌 3. 감지된 셀 목록에서 우선순위가 가장 높은 셀 하나를 선택
                highest_priority_cell = ""
                for cell_name in self.active_cell_priority:
                    if cell_name in stable_detected_set:
                        highest_priority_cell = cell_name
                        break

                if highest_priority_cell:
                    # 💡 우선순위가 높은 셀 하나만 발행
                    result_str = highest_priority_cell
                    self.detection_pub.publish(String(result_str))
                    rospy.loginfo(f"✨ Bright Area STABLE PRIORITY Detection
Published: {result_str}")
                else:
                    # (이 경우는 이론상 발생하기 어려움: stable_detected_set이 비어
있지 않은데 우선순위 목록에 없음)
                    result_str = ""
                    self.detection_pub.publish(String(result_str))
                    rospy.logwarn(f"--- ⚠️ Bright Area STABLE: No active
priority cell found ({result_str}) ---")

            else:
                # 🚨 아무것도 감지되지 않았을 경우: 빈 문자열 발행
                result_str = ""
                self.detection_pub.publish(String(result_str))
                rospy.loginfo(f"--- ⚠️ Bright Area STABLE Nothing Detected
({result_str}) ---")
```

```
            # 발행 후, 히스토리를 초기화하여 다음 새로운 감지를 대기 (중복 발행 방지)
            self.gray_history.clear()

        return display_frame # 플롯된 프레임을 반환

if __name__ == '__main__':
    try:
        node = GrayDetectionNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
    finally:
        cv2.destroyAllWindows()
```

filled_composite_detection_node.py

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from std_msgs.msg import String

class AssemblyGridDetectionNode:
    def __init__(self):
        rospy.init_node('assembly_grid_detection_node', anonymous=True)
        self.bridge = CvBridge()

        # ============================================================
        # 📌 퍼블리셔 및 서브스크라이버
        # ============================================================
        self.detection_pub = rospy.Publisher('/inverse_grid_detection_result',
String, queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # === 그리드 설정 ===
        self.P1, self.P2, self.P3 = (105,77), (547,73), (105,433)
        self.cols, self.rows = 4, 3
        self.active_cell_priority = ["1-1", "1-3", "2-2", "2-4", "3-1", "3-3"]
        self.active_cells = set((int(c.split('-')[0]), int(c.split('-')[1]))
for c in self.active_cell_priority)
        self.gray_history = []

        # === 🟠 주황색 감지 파라미터 ===
```

```
        # B, G, R 채널 기준 (0~255)
        self.ORANGE_B_MIN, self.ORANGE_B_MAX = 65, 185
        self.ORANGE_G_MIN, self.ORANGE_G_MAX = 110, 220
        self.ORANGE_R_MIN, self.ORANGE_R_MAX = 220, 255
        self.orange_pixel_ratio = 0.05  # 셀 내 주황색이 5% 이상일 때 감지

    # ------------------------------------------------------------
    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            result_frame = self.orange_area_detect(frame)
            cv2.imshow("Assembly Grid Detection (Orange Area)", result_frame)
            cv2.waitKey(1)
        except Exception as e:
            rospy.logerr(f"Detection Error: {e}")

    # ------------------------------------------------------------
    def orange_area_detect(self, frame):
        display_frame = frame.copy()

        xs = np.linspace(self.P1[0], self.P2[0], self.cols+1)
        ys = np.linspace(self.P1[1], self.P3[1], self.rows+1)
        detected_cell_names = set()

        # 1️⃣ 셀별 주황색 영역 감지
        for i in range(self.rows):
            for j in range(self.cols):
                r, c = i+1, j+1
                cell_name = f"{r}-{c}"
                x1, x2 = int(xs[j]), int(xs[j+1])
                y1, y2 = int(ys[i]), int(ys[i+1])

                if y2 <= y1 or x2 <= x1:
                    continue

                cell_roi = frame[y1:y2, x1:x2]
                cell_area = cell_roi.shape[0] * cell_roi.shape[1]

                # 주황색 범위 마스크 (BGR)
                b_min, b_max = self.ORANGE_B_MIN, self.ORANGE_B_MAX
                g_min, g_max = self.ORANGE_G_MIN, self.ORANGE_G_MAX
                r_min, r_max = self.ORANGE_R_MIN, self.ORANGE_R_MAX

                orange_mask = cv2.inRange(cell_roi, (b_min, g_min, r_min),
(b_max, g_max, r_max))
                orange_count = np.sum(orange_mask > 0)
```

```
                current_ratio = orange_count / cell_area if cell_area > 0 else
0

                color = (80,80,80)
                text = f"{r}-{c} ({current_ratio:.2f})"

                if (r, c) in self.active_cells:
                    color = (0,255,0)

                    if current_ratio >= self.orange_pixel_ratio:
                        detected_cell_names.add(cell_name)
                        color = (0,140,255)  # 🟠 주황색 감지시 색상 강조
                        text = f"DETECTED ({current_ratio:.2f})"

                    cv2.rectangle(display_frame, (x1, y1), (x2, y2), color, 2)
                    cx, cy = int((x1+x2)/2), int((y1+y2)/2)
                    cv2.putText(display_frame, text, (cx - 60, cy),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255),
2)
                else:
                    cv2.rectangle(display_frame, (x1, y1), (x2, y2), color, 1)

        # 2️⃣ 3프레임 안정화 로직 (EMA처럼 프레임별 안정화)
        self.gray_history.append(frozenset(detected_cell_names))
        if len(self.gray_history) > 3:
            self.gray_history.pop(0)

        if len(self.gray_history) == 3 and all(h == self.gray_history[0] for h
in self.gray_history):
            stable_detected_set = self.gray_history[0]
            result_str = ""

            if len(stable_detected_set) > 0:
                highest_priority_cell = ""
                for cell_name in self.active_cell_priority:
                    if cell_name in stable_detected_set:
                        highest_priority_cell = cell_name
                        break

                if highest_priority_cell:
                    result_str = highest_priority_cell
                    self.detection_pub.publish(String(result_str))
                    rospy.loginfo(f"✨ Assembly Grid STABLE PRIORITY Detection
Published: {result_str}")
                else:
                    result_str = ""
```

```
                    self.detection_pub.publish(String(result_str))
                    rospy.logwarn(f"⚠️ No active priority cell found
({result_str})")
            else:
                result_str = ""
                self.detection_pub.publish(String(result_str))
                rospy.loginfo(f"⚠️ Nothing Detected ({result_str})")

            self.gray_history.clear()

        return display_frame


# ------------------------------------------------------------------
if __name__ == '__main__':
    try:
        node = AssemblyGridDetectionNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
    finally:
        cv2.destroyAllWindows()
```

`move_group_python_interface.py`

```
#! /usr/bin/env python3

import sys
import rospy
import moveit_commander
import moveit_msgs
import geometry_msgs

import tf
import copy
import math

from tf.transformations import *
from geometry_msgs.msg import Quaternion
from moveit_commander.conversions import pose_to_list
from math import pi, tau, dist, fabs, cos

from ur_msgs.srv import SetIO, SetIORequest
from ur_python.msg import robot_state
```

```python
def all_close(goal, actual, tolerance):
    """Compare poses or lists within a tolerance."""
    if type(goal) is list:
        for index in range(len(goal)):
            if abs(actual[index] - goal[index]) > tolerance:
                return False

    elif type(goal) is geometry_msgs.msg.PoseStamped:
        return all_close(goal.pose, actual.pose, tolerance)

    elif type(goal) is geometry_msgs.msg.Pose:
        x0, y0, z0, qx0, qy0, qz0, qw0 = pose_to_list(actual)
        x1, y1, z1, qx1, qy1, qz1, qw1 = pose_to_list(goal)
        d = dist((x1, y1, z1), (x0, y0, z0))
        cos_phi_half = fabs(qx0*qx1 + qy0*qy1 + qz0*qz1 + qw0*qw1)
        return d <= tolerance and cos_phi_half >= cos(tolerance / 2.0)

    return True


class MoveGroupPythonInterface(object):
    """MoveGroupPythonInterface"""

    def ensure_minimal_rotation(self, current_quat, target_quat):
        """Quaternion 부호 반전으로 인한 180° jump 방지"""
        dot = sum(c * t for c, t in zip(current_quat, target_quat))
        if dot < 0.0:
            target_quat = [-t for t in target_quat]
        return target_quat

    def __init__(self, real="real", gripper="gripper"):
        super(MoveGroupPythonInterface, self).__init__()

        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node("move_group_python_interface", anonymous=True)

        self.msg_robot_state = robot_state()
        self.pub_robot_state = rospy.Publisher("robot_state", robot_state,
queue_size=10)

        self.robot = moveit_commander.RobotCommander()
        self.group_name = "manipulator"
        self.manipulator =
moveit_commander.move_group.MoveGroupCommander(self.group_name)

        if real == "real":
```

```
            self.io_handler =
rospy.ServiceProxy('/ur_hardware_interface/set_io', SetIO)
            self.gripper = SetIORequest()
            self.gripper_init()

        self.planning_frame = self.manipulator.get_planning_frame()
        print("============ Planning frame: %s" % self.planning_frame)
        self.eef_link = self.manipulator.get_end_effector_link()
        print("============ End effector link: %s" % self.eef_link)
        self.group_names = self.robot.get_group_names()
        print("============ Available Planning Groups:",
self.robot.get_group_names())
        print("============ Printing robot state")
        print(self.robot.get_current_state())
        print("")
        rospy.sleep(1)

    def move_to_standby(self):
        print("============ Go to 'stand_by' pose")
        self.go_to_joint_abs([tau/4, -tau/4, tau/4, -tau/4, -tau/4, 0.0])
        self.manipulator.go(wait=True)
        self.manipulator.stop()
        print("============ Printing robot state")
        print(self.robot.get_current_state())

    def gripper_init(self):
        self.gripper.fun = 1
        self.gripper.pin = 1
        self.grip_off()

    def grip_on(self):
        self.msg_robot_state.move = 1
        self.pub_robot_state.publish(self.msg_robot_state)
        self.gripper.state = 1
        self.io_handler.call(self.gripper)
        rospy.sleep(1.5)
        self.msg_robot_state.move = 0
        self.pub_robot_state.publish(self.msg_robot_state)

    def grip_off(self):
        self.msg_robot_state.move = 1
        self.pub_robot_state.publish(self.msg_robot_state)
        self.gripper.state = 0
        self.io_handler.call(self.gripper)
        rospy.sleep(1.5)
        self.msg_robot_state.move = 0
```

```
        self.pub_robot_state.publish(self.msg_robot_state)

    def go_to_joint_abs(self, target_joints):
        current_joint = self.manipulator.get_current_joint_values()
        target_joint = copy.deepcopy(current_joint)
        for i in range(6):
            target_joint[i] = target_joints[i]

        self.msg_robot_state.move = 1
        self.pub_robot_state.publish(self.msg_robot_state)
        self.manipulator.go(target_joint, wait=True)
        self.manipulator.stop()
        self.msg_robot_state.move = 0
        self.pub_robot_state.publish(self.msg_robot_state)
        return all_close(target_joint,
self.manipulator.get_current_joint_values(), 0.005)

    def go_to_joint_rel(self, relative_pos):
        current_joint = self.manipulator.get_current_joint_values()
        target_joint = [cj + r for cj, r in zip(current_joint, relative_pos)]
        self.msg_robot_state.move = 1
        self.pub_robot_state.publish(self.msg_robot_state)
        self.manipulator.go(target_joint, wait=True)
        self.manipulator.stop()
        self.msg_robot_state.move = 0
        self.pub_robot_state.publish(self.msg_robot_state)
        return all_close(target_joint,
self.manipulator.get_current_joint_values(), 0.005)

    def go_to_pose_abs(self, absolute_xyz, absolute_rpy):
        current_pose = self.manipulator.get_current_pose().pose
        target_pose = copy.deepcopy(current_pose)
        target_pose.position.x = absolute_xyz[0]
        target_pose.position.y = absolute_xyz[1]
        target_pose.position.z = absolute_xyz[2]
        target_quat = quaternion_from_euler(*absolute_rpy)
        target_pose.orientation = Quaternion(*target_quat)
        self.manipulator.set_pose_target(target_pose)
        self.msg_robot_state.move = 1
        self.pub_robot_state.publish(self.msg_robot_state)
        self.manipulator.go(wait=True)
        self.manipulator.stop()
        self.manipulator.clear_pose_targets()
        self.msg_robot_state.move = 0
        self.pub_robot_state.publish(self.msg_robot_state)
        return all_close(target_pose,
```

```
self.manipulator.get_current_pose().pose, 0.005)

    def go_to_pose_rel(self, relative_xyz, relative_rpy, max_retry=3):
        """
        Move the manipulator to a relative pose (position + orientation)
        with retry and stability.
        relative_xyz: [dx, dy, dz] in meters
        relative_rpy: [dR, dP, dY] in radians
        """

        # ① 현재 pose 읽기
        current_pose = self.manipulator.get_current_pose().pose
        target_pose = copy.deepcopy(current_pose)

        # ② 상대 이동 적용
        target_pose.position.x += relative_xyz[0]
        target_pose.position.y += relative_xyz[1]
        target_pose.position.z += relative_xyz[2]

        # ③ 회전 (쿼터니언 누적 방식)
        current_quat = [
            current_pose.orientation.x,
            current_pose.orientation.y,
            current_pose.orientation.z,
            current_pose.orientation.w,
        ]
        rel_quat = quaternion_from_euler(*relative_rpy)

        # 누적 회전 (상대 회전 * 현재 회전)
        target_quat = quaternion_multiply(rel_quat, current_quat)

        # minimal rotation 보정
        dot = sum(c * t for c, t in zip(current_quat, target_quat))
        if dot < 0.0:
            target_quat = [-q for q in target_quat]
            rospy.logwarn("🌀 Quaternion flipped for minimal rotation")

        target_pose.orientation = Quaternion(*target_quat)

        # 로그 출력
        rospy.loginfo(f"🎯 Δxyz = {relative_xyz}, Δrpy(deg) =
{[round(x*180/pi,3) for x in relative_rpy]}")
        rospy.loginfo(f"📍 From z={current_pose.position.z:.3f} → To z=
{target_pose.position.z:.3f}")

        # ④ Cartesian Path 기반 경로 계획 및 실행 (재시도)
```

```
        waypoints = [target_pose]
        success = False

        for attempt in range(1, max_retry + 1):
            (plan, fraction) = self.manipulator.compute_cartesian_path(
                waypoints,
                0.002,   # eef_step (단위: m)
                True     # jump_threshold
            )
            rospy.loginfo(f"🧭 Attempt {attempt}: Cartesian path fraction=
{fraction:.3f}")

            if fraction > 0.9:
                rospy.loginfo(f"✅ Cartesian path success on attempt
{attempt}")
                self.msg_robot_state.move = 1
                self.pub_robot_state.publish(self.msg_robot_state)

                self.manipulator.execute(plan, wait=True)
                self.manipulator.stop()

                self.msg_robot_state.move = 0
                self.pub_robot_state.publish(self.msg_robot_state)
                success = True
                break
            else:
                rospy.logwarn(f"⚠️ Path incomplete (fraction={fraction:.3f}),
retrying...")
                rospy.sleep(0.5)

        # ⑤ 실패 시 안전 중단
        if not success:
            rospy.logerr("❌ Cartesian path failed after retries — skipping
safely.")
            return False

        # ⑥ 최종 자세 비교
        current_pose = self.manipulator.get_current_pose().pose
        if not all_close(target_pose, current_pose, 0.003):
            rospy.logwarn("⚠️ Final pose deviation detected, check accuracy.")

        return True
```

main.py

```python
#! /usr/bin/env python3
# -*- coding:utf-8 -*-

import sys
import rospy
import math
import numpy as np
from math import pi
import time as pytime
from time import time
import serial

from move_group_python_interface import MoveGroupPythonInterface
from geometry_msgs.msg import Point
from std_msgs.msg import String
from tf.transformations import euler_from_quaternion, quaternion_from_euler
from geometry_msgs.msg import Pose, Quaternion


# ------------------------------------------------------------
# 간단 토픽 래치
# ------------------------------------------------------------
class TopicLatch:
    def __init__(self, ttl=3.0):
        self.ttl = ttl
        self.data = None
        self.stamp = 0.0
    def update(self, msg):
        self.data = msg
        self.stamp = time()
    def alive(self):
        return (time() - self.stamp) <= self.ttl
    def get(self, default=None):
        return self.data if self.alive() else default


class AsmFsmNode:
    def __init__(self):
        # rospy.init_node("asm_fsm_node", anonymous=True)
        self.ur5e = MoveGroupPythonInterface(real="real", gripper="gripper")

        # ✅ 초기 RPY 부호 교정 불필요
        rospy.loginfo("🧭 MoveGroupPythonInterface initialized with stabilized
RPY logic (±0.1° limit + minimal rotation).")

        # ---------- 아두이노 시리얼 연결 ----------
```

```
        try:
            self.arduino = serial.Serial()
            self.arduino.port = '/dev/ttyACM0'
            self.arduino.baudrate = 9600
            self.arduino.timeout = 1

            # ★★★ reset 방지 (포트 오픈 전에 해야 함)
            self.arduino.dtr = False
            self.arduino.rts = False

            self.arduino.open()     # 이제 UNO Reset 절대 안 됨

            rospy.loginfo("🔌 Arduino Serial Connected (/dev/ttyACM) without
reset")
        except Exception as e:
            rospy.logwarn(f"⚠️ Arduino Serial Connection Failed: {e}")
            self.arduino = None


        # 이하 원래 코드
        self.GRIP_HOLD = 2
        self.D2R = pi/180.0
        self.prev_orange = None
        self.ema_alpha = 0.35
        self.ema_stable_thresh_mm = 10.0  # 1mm 이하 변화 시 안정화 간주
        self.ema_min_frames = 2           # 최소 4프레임 유지 필요
        self.ema_stable_count = 0


        # ---- 토픽 래치 ----
        self.red_result        = TopicLatch(ttl=2.0)   # /red_detection/result
(Point)
        self.orange_result     = TopicLatch(ttl=2.0)   #
/orange_detection/result (Point)
        self.darkgrid_result   = TopicLatch(ttl=2.0)   #
/darkgrid_detection_result (String)
        self.gray_result       = TopicLatch(ttl=2.0)   #
/gray_detection_result (String)
        self.inv_gray_result   = TopicLatch(ttl=2.0)   #
/inverse_gray_detection_result (String)
        self.inv_grid_result   = TopicLatch(ttl=2.0)   #
/inverse_grid_detection_result (String)
        self.fallback_circle   = TopicLatch(ttl=2.0)   #
/fallback_circle/result (Point)
        self.grid1_fallback    = TopicLatch(ttl=2.0)   #
/grid1_fallback/result (Point)
```

```
        self.grid2_fallback    = TopicLatch(ttl=2.0)   #
/grid2_fallback/result (Point)


        # ---- 구독 ----
        rospy.Subscriber("/red_detection/result", Point, self.cb_red)
        rospy.Subscriber("/orange_detection/result", Point, self.cb_orange)
        rospy.Subscriber("/darkgrid_detection_result", String,
self.cb_darkgrid)
        rospy.Subscriber("/gray_detection_result", String, self.cb_gray)
        rospy.Subscriber("/inverse_gray_detection_result", String,
self.cb_inv_gray)
        rospy.Subscriber("/inverse_grid_detection_result", String,
self.cb_inv_grid)
        rospy.Subscriber("/fallback_circle/result", Point,
self.cb_fallback_circle)
        rospy.Subscriber("/grid1_fallback/result", Point,
self.cb_grid1_fallback)
        rospy.Subscriber("/grid2_fallback/result", Point,
self.cb_grid2_fallback)


        # ---- 조인트 (deg) ----
        self.ASM_OBS_START_JOINTS          = [298.62, -102.48, 119.54,
-106.94, -89.63, -61.17]
        self.RED_PLACE_POSE_1_JOINTS       = [308.07,  -92.37, 110.96,
-108.60, -89.66, -51.81]
        self.ASM_OBS_GRID_JOINTS           = [243.53,  -74.39,  66.28,
-82.02, -90.00, -116.50]
        self.ASM_RED_DETECT_TRIGGER_JOINTS = [272.92, -118.99, 129.53,
-100.31, -89.71,  -86.91]
        self.OBS_GRID_JOINTS               = [267.86,  -74.47,  66.36,
-81.72, -90.07,  -92.25]
        self.RED_PLACE_POSE_2_JOINTS       = [287.35, -103.92, 128.08,
-108.26, -88.37,  -67.33]

        # === 새로 지정된 Grip-OFF 경유 좌표들 ===
        # A 경로 (state 1/101/201/301에서 사용)
        self.OFF_A1 = [308.44, -89.11, 116.87, -117.93, -89.93, -51.93]
        self.OFF_A2 = [308.44, -90.27, 115.27, -115.26, -89.92, -51.96]
        # B 경로 (state 3/103에서 사용)
        self.OFF_B1 = [288.55, -106.54, 133.50, -116.53, -89.89, -71.82]
        self.OFF_B2 = [288.58, -108.29, 130.91, -112.21, -89.95, -71.79]
        # state 6/106 이후 Grip OFF 위치 동일 (B2)
        self.GRIP_OFF_B2 = self.OFF_B2
        # state 5/105 이후 Grip OFF 위치 (A2)
```

```
        self.GRIP_OFF_A2 = self.OFF_A2

        # ---- 토픽별 포즈 딕셔너리 (새 좌표 반영) ----
        # /gray_detection_result → 조립용 픽 (그레이드 셀)
        self.ASM_GRID_POSES = {
            "1-1":{"start":[273.89, -84.60, 93.19, -98.45, -90.06, -86.15]},
            "1-3":{"start":[273.15, -67.90, 72.17, -94.09, -90.09, -86.80]},
            "2-2":{"start":[266.95, -78.48, 86.06, -97.27, -90.08, -93.06]},
            "2-4":{"start":[267.49, -60.76, 61.65, -90.54, -90.09, -92.57]},
            "3-1":{"start":[259.17, -88.10, 96.91, -98.58, -90.15, -100.80]},
            "3-3":{"start":[261.31, -70.86, 76.25, -95.19, -90.08, -98.61]},
        }

        self.DIS_GRID_POSES = {
            "1-1": {"start": [245.12, -87.75, 96.56, -98.68, -90.19,
-114.91]},
            "1-3": {"start": [249.95, -70.67, 76.04, -95.30, -90.18,
-110.01]},
            "2-2": {"start": [241.81, -77.35, 84.69, -97.10, -90.22,
-118.16]},
            "2-4": {"start": [246.63, -59.80, 60.15, -90.00, -90.18,
-113.43]},
            "3-1": {"start": [232.74, -82.41, 90.72, -98.17, -90.27,
-127.19]},
            "3-3": {"start": [239.69, -65.02, 68.04, -92.84, -90.11,
-120.19]},
        }

        # /darkgrid_detection_result → 빈 그리드 픽(조립 전)
        self.DARKGRID_POSES = {
        "1-1":{"start":[254.13,-89.30,97.65,-98.06,-90.11,-107.77],"pick":
[254.10,-86.06,107.90,-111.54,-90.01,-107.82]},
        "1-3":{"start":[257.45,-71.90,77.20,-95.00,-89.84,-102.30],"pick":
[257.44,-69.45,87.32,-107.60,-89.84,-102.31]},
        "2-2":{"start":[249.57,-80.41,85.10,-94.64,-89.90,-110.18],"pick":
[249.40,-77.17,98.43,-111.19,-89.92,-110.35]},
        "2-4":{"start":[253.53,-62.42,60.22,-87.84,-89.81,-106.28],"pick":
[253.18,-60.95,74.89,-103.90,-89.83,-106.34]},
        "3-1":{"start":[240.32,-86.43,93.50,-96.67,-89.95,-119.46],"pick":
[240.31,-83.03,104.88,-111.49,-89.94,-119.43]},
        "3-3":{"start":[246.35,-69.47,70.17,-90.42,-89.79,-114.60],"pick":
[245.97,-67.14,83.81,-106.32,-89.81,-113.90]},
        }
        # /inverse_gray_detection_result → fallback 상황의 회색 셀 픽
        self._FALLBACK_POSES = {
        "1-1":{"start":[282.64,-80.76,88.80,-97.86,-89.85,-77.24],"pick":
```

```
[282.38,-78.32,97.93,-109.45,-89.81,-77.47]},
        "1-3":{"start":[280.40,-65.60,69.21,-94.72,-90.02,-79.52],"pick":
[280.11,-63.99,78.71,-105.82,-90.02,-79.77]},
        "2-2":{"start":[275.13,-77.00,84.56,-98.48,-89.79,-84.87],"pick":
[275.01,-74.55,94.24,-110.57,-89.77,-84.99]},
        "2-4":{"start":[274.12,-59.33,59.69,-91.05,-89.66,-85.73],"pick":
[273.92,-58.49,70.86,-103.01,-89.54,-85.82]},
        "3-1":{"start":[268.33,-87.22,96.27,-99.81,-89.82,-91.56],"pick":
[268.05,-84.59,106.33,-112.47,-89.92,-91.80]},
        "3-3":{"start":[268.64,-70.52,76.07,-96.50,-89.75,-91.21],"pick":
[268.18,-68.58,86.08,-108.45,-89.77,-91.65]},
        }
        # /inverse_grid_detection_result → 분해공정 픽
        self.INVERSE_POSES = {
            "1-1":{"start":
[253.74,-89.33,99.87,-100.37,-89.86,-108.12],"pick":
[253.73,-84.97,110.70,-115.54,-89.85,-108.10]},
            "1-3":{"start":[257.34,-71.88,79.42,-97.27,-89.95,-102.36],"pick":
[257.34,-68.50,89.85,-111.02,-89.93,-102.33]},
            "2-2":{"start":
[249.35,-79.85,90.38,-100.50,-90.06,-110.40],"pick":
[249.33,-75.75,100.63,-114.85,-89.97,-110.45]},
            "2-4":{"start":[253.09,-62.55,66.25,-93.71,-89.88,-106.70],"pick":
[253.09,-59.97,77.20,-107.24,-89.83,-106.77]},
            "3-1":{"start":
[240.19,-85.99,96.76,-100.47,-89.94,-119.65],"pick":
[240.22,-81.69,107.36,-115.48,-90.01,-119.60]},
            "3-3":{"start":[245.91,-69.16,75.62,-96.42,-89.90,-114.03],"pick":
[245.91,-65.99,86.15,-110.08,-89.90,-113.99]},
        }

        # ---- fallback 안정화 파라미터 ----
        self.fallback_stable = {"x": None, "y": None, "t": 0.0}
        self.FALLBACK_STABLE_TIME = 2.0      # 2초 유지
        self.FALLBACK_TOLERANCE = 20.0       # ±20 px
        self.FALLBACK_MATCH_TOLERANCE = 25.0 # red/orange와 ±25 px

        # ---- 초기 상태 ----
        self.state = 1
        rospy.loginfo("🧠 ASM FSM Node Ready. Start at state 1.")




    def open_stopper(self):
        rospy.logerr(">>> open_stopper() CALLED NOW <<<")
```

```
        if not self.arduino:
            rospy.logwarn("Arduino not connected")
            return

        # 1) Arduino auto-reset 방지 (UNO)
        try:
            self.arduino.setDTR(False)
            self.arduino.setRTS(False)
        except:
            pass

        # 2) 입력 버퍼 비우기
        self.arduino.reset_input_buffer()

        # 3) 명령 즉시 보내기
        self.arduino.write(b'c\n')
        self.arduino.flush()

        rospy.loginfo("[WRITE] c")

        # 4) Arduino 응답 빠르게 읽기 (0.3초) → pytime 사용! (충돌 방지)
        end = pytime.time() + 0.3
        while pytime.time() < end:
            if self.arduino.in_waiting:
                line = self.arduino.readline().decode(errors='ignore').strip()
                rospy.loginfo(f"[ARDUINO] {line}")




    # --------- 콜백 ---------
    def cb_red(self, msg: Point): self.red_result.update(msg)
    def cb_orange(self, msg: Point): self.orange_result.update(msg)
    def cb_darkgrid(self, msg: String): self.darkgrid_result.update(msg)
    def cb_gray(self, msg: String): self.gray_result.update(msg)
    def cb_inv_gray(self, msg: String): self.inv_gray_result.update(msg)
    def cb_inv_grid(self, msg: String): self.inv_grid_result.update(msg)
    def cb_fallback_circle(self, msg: Point): self.fallback_circle.update(msg)
    def cb_grid1_fallback(self, msg: Point): self.grid1_fallback.update(msg)
    def cb_grid2_fallback(self, msg: Point): self.grid2_fallback.update(msg)


    # --------- 헬퍼 ---------
    def jdeg(self, arr_deg):
        return [x*self.D2R for x in arr_deg]
```

```
    def goJ(self, arr_deg, label=""):
        if label: rospy.loginfo(f"➡️ MoveJ: {label}")
        ok = self.ur5e.go_to_joint_abs(self.jdeg(arr_deg))
        rospy.sleep(1.5)
        return ok

    def wait_until_rpy_sign_ok(self, tol_check_rate=0.5):
        while not rospy.is_shutdown():
            try:
                # 현재 pose 읽기
                pose = self.ur5e.manipulator.get_current_pose().pose
                q = pose.orientation
                roll, pitch, yaw = np.degrees(euler_from_quaternion([q.x, q.y,
q.z, q.w]))

                # 부호 확인
                roll_ok  = roll < 0
                pitch_ok = pitch > 0
                yaw_ok   = yaw > 0

                if roll_ok and pitch_ok and yaw_ok:
                    rospy.loginfo(f"✅ RPY OK: [R={roll:.3f}, P={pitch:.3f},
Y={yaw:.3f}]")
                    break
                else:
                    rospy.logwarn(
                        f"⚠️  RPY sign mismatch! [R={roll:.3f}, P={pitch:.3f},
Y={yaw:.3f}] → waiting..."
                    )

            except Exception as e:
                rospy.logwarn(f"⚠️  Failed to read RPY: {e}")

            rospy.sleep(tol_check_rate)

    # ✅ 감지 유효 자세 확인 (OBS1 or OBS2)
    def valid_pose_for_detection(self, tol_deg=1.0):
        """
        현재 UR5e 조인트가 감지 가능한 관찰 자세(OBS1/OBS2)인지 확인.
        tol_deg: 허용 오차 (deg)
        """
        try:
            # 현재 UR5e 조인트 읽기 (deg 변환)
            current_joints = [x / self.D2R for x in
self.ur5e.manipulator.get_current_joint_values()]
```

```
        except Exception as e:
            rospy.logwarn(f"⚠️ Cannot read current joint state: {e}")
            return False

        def is_close(target):
            return all(abs(a - b) <= tol_deg for a, b in zip(current_joints,
target))

        # ✅ OBS1 또는 OBS2 근처에 있을 때만 감지 허용
        valid = (
            is_close(self.ASM_OBS_START_JOINTS) or
            is_close(self.ASM_RED_DETECT_TRIGGER_JOINTS)
        )

        if not valid:
            rospy.loginfo("🚫 Detection ignored (not in OBS1 or OBS2 pose)")
        else:
            rospy.loginfo("✅ Valid detection pose confirmed.")
        return valid


    def grip_on_6s(self):
        self.ur5e.grip_on()
        rospy.sleep(max(0.0, self.GRIP_HOLD - 1.5))

    def grip_off_6s(self):
        self.ur5e.grip_off()
        rospy.sleep(max(0.0, self.GRIP_HOLD - 1.5))

    def point_detected(self, latch: TopicLatch):
        # ✅ OBS1 또는 OBS2에서만 감지 유효
        if not self.valid_pose_for_detection():
            return False
        msg = latch.get()
        return (msg is not None) and (getattr(msg, "x", 0.0) != 0.0)


    def choose_cell_from_string(self, s):
        if not s: return None
        txt = s.strip()
        if "," in txt: txt = txt.split(",")[0].strip()
        if " " in txt: txt = txt.split()[0].strip()
        return txt

    def pick_cycle(self, pose_dict, cell_key, grip_mode="off"):
        if not cell_key or cell_key not in pose_dict:
```

```
            rospy.logwarn(f"⚠️ invalid grid key: {cell_key}")
            return False
        p = pose_dict[cell_key]
        ok  = self.goJ(p["start"], f"{cell_key} start")
        ok &= self.goJ(p["pick"],  f"{cell_key} pick")
        if grip_mode == "on": self.grip_on_6s()
        else:                 self.grip_off_6s()
        ok &= self.goJ(p["start"], f"{cell_key} back")
        return ok

    # --------- fallback 안정 감지 (±20px/2s + red/orange 매칭 ±25px) ---------
    # --------- fallback 안정 감지 (±20px/2s + red/orange 매칭 ±25px) ---------
    def fallback_stable_detected(self):
        if not self.fallback_circle.alive():
            rospy.loginfo("⚠️ No active /fallback_circle/result messages.")
            self.fallback_stable["x"] = None
            return False

        msg_fb = self.fallback_circle.get()
        msg_red = self.red_result.get()
        msg_orange = self.orange_result.get()
        if msg_fb is None:
            rospy.loginfo("⚠️ fallback_circle message is None.")
            return False

        # ✅ (0, 0, 0) 퍼블리시 필터링
        if msg_fb.x == 0.0 and msg_fb.y == 0.0 and msg_fb.z == 0.0:
            rospy.logdebug("⚠️ fallback_circle published (0,0,0) → skip.")
            self.fallback_stable["x"] = None
            return False

        x, y = msg_fb.x, msg_fb.y
        now = time()
        ...


        # 🔸 3️⃣ 첫 감지 → 기준 좌표 등록
        if self.fallback_stable["x"] is None:
            self.fallback_stable = {"x": x, "y": y, "t": now}
            return False

        dx = abs(x - self.fallback_stable["x"])
        dy = abs(y - self.fallback_stable["y"])

        # 🔸 4️⃣ ±20px 이내 이동 유지 확인
        stable_ok = (dx <= self.FALLBACK_TOLERANCE and dy <=
```

```
self.FALLBACK_TOLERANCE)
        time_ok = (now - self.fallback_stable["t"] >=
self.FALLBACK_STABLE_TIME)

        # 🔸 5️⃣ 좌표가 새로 벗어났으면 기준 갱신
        if not stable_ok:
            self.fallback_stable = {"x": x, "y": y, "t": now}
            return False

        # 🔸 6️⃣ red/orange 감지와 ±25px 매칭 여부 확인
        def close_enough(msg_ref):
            if msg_ref is None: return False
            return (abs(x - msg_ref.x) <= self.FALLBACK_MATCH_TOLERANCE and
                    abs(y - msg_ref.y) <= self.FALLBACK_MATCH_TOLERANCE)

        match_ok = close_enough(msg_red) or close_enough(msg_orange)

        # 🔸 7️⃣ 모든 조건 충족 시 감지 성공
        if stable_ok and time_ok and match_ok:
            rospy.loginfo(f"✅ Fallback stabilized & matched (x={x:.1f}, y=
{y:.1f})")
            return True

        return False


    # --------- 메인 루프 ---------
    def spin(self):
        rate = rospy.Rate(10)
        while not rospy.is_shutdown():
            try:
                if   self.state == 1:   self.state_1()
                elif self.state == 2:   self.state_2()
                elif self.state == 3:   self.state_3()
                elif self.state == 4:   self.state_4()
                elif self.state == 5:   self.state_5()
                elif self.state == 6:   self.state_6()
                elif self.state == 101: self.state_101()
                elif self.state == 102: self.state_102()
                elif self.state == 103: self.state_103()
                elif self.state == 104: self.state_104()
                elif self.state == 105: self.state_105()
                elif self.state == 106: self.state_106()
                elif self.state == 201: self.state_201()
                elif self.state == 202: self.state_202()
                elif self.state == 301: self.state_301()
```

```
                elif self.state == 302: self.state_302()
                else:
                    rospy.logwarn(f"❓ Unknown state {self.state} -> 1")
                    self.state = 1
            except Exception as e:
                rospy.logerr(f"💥 Exception in state {self.state}: {e}")
            rate.sleep()

    def state_1(self):
        try:
            # ① 관찰 자세로 이동
            self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")
            rospy.sleep(1.0)

            # ② 감지 결과 확인
            has_red    = self.point_detected(self.red_result)
            has_orange = self.point_detected(self.orange_result)
            msg_orange = self.orange_result.get()

            # ============================================================
            # 🍊 오렌지 감지 시 – 좌표 기반 픽업 절차 실행
            # ============================================================
            if has_orange and msg_orange:
                dx = msg_orange.x
                dy = msg_orange.y

                # 비정상 토픽 값 (픽셀 좌표 등) 필터링
                if abs(dx) > 200 or abs(dy) > 200:
                    rospy.logwarn(f"⚠️ Ignored abnormal orange detection: x=
{dx:.1f}, y={dy:.1f}")
                    return

                # ------------------------------
                # EMA 적용 (좌표 안정화 필터)
                # ------------------------------
                if self.prev_orange is None:
                    self.prev_orange = np.array([dx, dy], dtype=float)
                    self.ema_stable_count = 0
                    rospy.loginfo("📊 EMA 초기화 중...")
                    return

                curr = np.array([dx, dy], dtype=float)
                self.prev_orange = (1 - self.ema_alpha) * self.prev_orange +
self.ema_alpha * curr
                diff = np.linalg.norm(curr - self.prev_orange)
```

```
                if diff < self.ema_stable_thresh_mm:
                    self.ema_stable_count += 2
                else:
                    self.ema_stable_count = 0  # 다시 초기화

                rospy.loginfo_throttle(1.0, f"📈 EMA 안정도: diff={diff:.2f}mm,
count={self.ema_stable_count}")

                # 충분히 안정화되었을 때만 픽업 시작
                if self.ema_stable_count < self.ema_min_frames:
                    rospy.loginfo("⏸ EMA 안정화 대기 중...")
                    return

                rospy.loginfo("✅ EMA 안정화 완료 → 픽업 시작")

                dx_m = self.prev_orange[0] / 1000.0
                dy_m = self.prev_orange[1] / 1000.0

                # 1️⃣ XY 평면 이동
                rospy.loginfo(f"➡️ Move to target: dx={dx_m:.4f} m, dy=
{dy_m:.4f} m")
                self.ur5e.go_to_pose_rel([dx_m, dy_m, 0.0], [0,0,0])
                rospy.sleep(0.3)

                # 2️⃣ Z 하강
                rospy.loginfo("⬇️ Descend 70 mm (Z only)")
                self.ur5e.go_to_pose_rel([0.0, 0.0, -0.07], [0,0,0])

                # 3️⃣ Grip ON
                self.grip_on_6s()

                # 4️⃣ Lift
                self.ur5e.go_to_pose_rel([0.0, 0.0, 0.115], [0,0,0])

                # 5️⃣ 관찰 자세 복귀
                self.goJ(self.ASM_OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")

                # 6️⃣ 다음 상태 전환
                self.state = 2
                rospy.loginfo("➡️ state 2")

            elif has_red and not has_orange:
                self.state = 3
                rospy.loginfo("➡️ state 3")

            else:
```

```
                rospy.loginfo("⏸ state1 hold")

        except Exception as e:
            rospy.logerr(f"💥 Exception in state 1: {e}")



    def state_2(self):
        msg = self.darkgrid_result.get()
        cell = self.choose_cell_from_string(msg.data if msg else "")
        if cell:
            self.pick_cycle(self.DARKGRID_POSES, cell, grip_mode="off")
        else:
            rospy.loginfo("⏸ state2 no grid")
        self.state = 3
        rospy.loginfo("↩️  back to state 3")

    def state_3(self):
        try:
            # ① 관찰 자세로 이동
            self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
            rospy.sleep(0.5)  # 관찰 자세 안정화

            # 감지 결과 읽기
            has_red    = self.point_detected(self.red_result)
            has_orange = self.point_detected(self.orange_result)
            msg_orange = self.orange_result.get()

            # ============================================================
            # 🍊 오렌지 감지 시 — state 1과 동일한 좌표 기반 픽업 절차
            # ============================================================
            if has_orange and msg_orange:
                dx = msg_orange.x / 1000.0   # mm → m 변환
                dy = msg_orange.y / 1000.0

                # 비정상 좌표 필터링
                if abs(msg_orange.x) > 200 or abs(msg_orange.y) > 200:
                    rospy.logwarn(f"⚠️ Ignored abnormal orange detection: x=
{msg_orange.x:.1f}, y={msg_orange.y:.1f}")
                    return

                rospy.loginfo(f"🍊 Orange detected (state3): dx={dx:.4f} m,
dy={dy:.4f} m")

                # 1️⃣ XY 평면 이동
```

```
                rospy.loginfo("➡️ Moving in XY plane first...")
                self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0.0, 0.0, 0.0])
                rospy.sleep(0.3)

                # 2️⃣ Z 하강 -0.07 m
                rospy.loginfo("⬇️ Descend 70 mm (Z only)")
                self.ur5e.go_to_pose_rel([0.0, 0.0, -0.07], [0.0, 0.0, 0.0])

                # 3️⃣ Grip ON
                rospy.loginfo("✊ Gripper ON")
                self.grip_on_6s()

                # 4️⃣ Z 상승 +0.115 m
                rospy.loginfo("⬆️ Lift up 115 mm (Z only)")
                self.ur5e.go_to_pose_rel([0.0, 0.0, 0.115], [0.0, 0.0, 0.0])

                # 5️⃣ 관찰 자세 복귀
                self.goJ(self.ASM_OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")

                # 6️⃣ 다음 상태로
                self.state = 4
                rospy.loginfo("➡️ state 4")

            # ============================================================
            # 🔴 오렌지 감지 X, 빨간 원만 감지됨 → state 5로 전환
            # ============================================================
            elif has_red and not has_orange:
                self.state = 5
                rospy.loginfo("➡️ state 5")

            # ============================================================
            # 🚫 감지 안됐을 때
            # ============================================================
            else:
                rospy.loginfo("⏸ state3 hold")

        except Exception as e:
            rospy.logerr(f"💥 Exception in state 3: {e}")


    def state_4(self):
        msg = self.darkgrid_result.get()
        cell = self.choose_cell_from_string(msg.data if msg else "")
        if cell:
            self.pick_cycle(self.DARKGRID_POSES, cell, grip_mode="off")
        else:
```

```
            rospy.loginfo("⏸ state4 no grid")
        self.state = 5
        rospy.loginfo("➡️  state 5")


    def state_5(self):
        rospy.loginfo("➡️ Enter state_5()")

        # ① 관찰자세로 이동 (그리드 전체 관찰)
        self.goJ(self.OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")
        rospy.sleep(0.5)

        # ② /gray_detection/result 감지 시도
        rospy.loginfo("🕐 Waiting for /gray_detection/result ...")
        gray_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.gray_result.get()
            if msg and msg.data and msg.data.lower() != "none":
                gray_msg = msg
                break
            rospy.sleep(0.1)

        if not gray_msg:
            rospy.logwarn("❌ No gray cell detected → return to
OBS_START_JOINTS → state101")
            self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS (after
gray fail)")
            self.state = 101
            return

        # ✅ 감지 성공 시 셀 이름 결정
        cell = self.choose_cell_from_string(gray_msg.data if gray_msg else "")
        if not cell or cell not in self.ASM_GRID_POSES:
            rospy.logwarn(f"⚠️ Invalid or unknown gray cell: {gray_msg.data}")
            self.goJ(self.ASM_OBS_START_JOINTS, "OBS_START_JOINTS (invalid
gray)")
            self.state = 101
            return

        rospy.loginfo(f"✅ Gray cell detected: {cell}")

        # ③ 해당 셀의 START 자세로 이동
        start_pose = self.ASM_GRID_POSES[cell]["start"]
        self.goJ(start_pose, f"{cell} start (state5)")
```

```
        rospy.sleep(0.3)

        # ④ /grid1_fallback/result 감지 시도
        rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
        fb_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.grid1_fallback.get()
            if msg and (msg.x != 0.0 or msg.y != 0.0):
                fb_msg = msg
                break
            rospy.sleep(0.05)

        if fb_msg:
            dx = fb_msg.x / 1000.0  # mm → m
            dy = fb_msg.y / 1000.0
            rospy.loginfo(f"📍 grid1_fallback offset: dx={dx:.3f} m, dy=
{dy:.3f} m")

            self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
            rospy.sleep(0.3)
            self.ur5e.go_to_pose_rel([0, 0, -0.115], [0, 0, 0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])
        else:
            rospy.logwarn("⚠️ No /grid1_fallback/result → skipping pre-pick")

        # ⑤ RED DETECTION으로 미세 조정
        self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS (red
detection)")
        rospy.loginfo("⏸ Waiting 1.5s for camera stabilization...")
        rospy.sleep(0.5)

        rospy.loginfo("🕐 Waiting for /red_detection/result ...")
        red_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 3.0 and not
rospy.is_shutdown():
            msg = self.red_result.get()
            if msg and msg.x != 0.0 and msg.y != 0.0:
                red_msg = msg
                rospy.loginfo("🕐 Red detected — waiting 1.5s for EMA
stabilization...")
                rospy.sleep(0.5)   # ✅ EMA가 평균값으로 안정화될 시간을 확보
                break
```

```
            rospy.sleep(0.1)

        if not red_msg:
            rospy.logwarn("❌ No /red_detection/result → state101")
            self.state = 101
            return

        dx = red_msg.x / 1000.0
        dy = red_msg.y / 1000.0
        rospy.loginfo(f"📍 Red offset: dx={dx:.3f} m, dy={dy:.3f} m")

        try:
            rospy.loginfo(f"🚀 RELATIVE MOVE START: X={dx:.4f}m, Y={dy:.4f}m,
Z=0.0m")
            self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
            rospy.sleep(0.3)
        except Exception as e:
            rospy.logwarn(f"⚠️ XY offset move failed: {e}")

        # ⑥ 배치 및 복귀
        self.ur5e.go_to_pose_rel([0, 0, -0.04], [0, 0, 0])
        self.grip_off_6s()
        self.ur5e.go_to_pose_rel([0, 0, 0.04], [0, 0, 0])

        # ⑦ 복귀 및 다음 단계
        self.goJ(start_pose, f"{cell} return (state5)")
        self.state = 6
        rospy.loginfo("➡️ Next: state 6")


    # ------------------------------------------------------------
    def state_6(self):
        rospy.loginfo("➡️ Enter state_6()")

        # ① 관찰자세 이동 (그리드 전체 관찰)
        self.goJ(self.OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")
        rospy.sleep(0.5)

        # ② /gray_detection/result 감지 시도
        rospy.loginfo("🕐 Waiting for /gray_detection/result ...")
        gray_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.gray_result.get()
            if msg and msg.data and msg.data.lower() != "none":
```

```
                gray_msg = msg
                break
            rospy.sleep(0.1)

        # ⚠️ 감지 실패 시 → state 201
        if not gray_msg:
            rospy.logwarn("❌ No gray cell detected → return to
OBS_START_JOINTS → state201")
            self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS (after
gray fail)")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 201
            return

        # ✅ 감지 성공 시 셀 이름 결정
        cell = self.choose_cell_from_string(gray_msg.data if gray_msg else "")
        if not cell or cell not in self.ASM_GRID_POSES:
            rospy.logwarn(f"⚠️ Invalid or unknown gray cell: {gray_msg.data}")
            self.goJ(self.ASM_OBS_START_JOINTS, "OBS_START_JOINTS (invalid
gray)")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 201
            return

        rospy.loginfo(f"✅ Gray cell detected: {cell}")

        # ③ 해당 셀의 START 자세로 이동
        start_pose = self.ASM_GRID_POSES[cell]["start"]
        self.goJ(start_pose, f"{cell} start (state6)")
        rospy.sleep(0.3)

        # ④ /grid1_fallback/result 감지 시도
        rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
        fb_msg = None
```

```
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.grid1_fallback.get()
            if msg and (msg.x != 0.0 or msg.y != 0.0):
                fb_msg = msg
                break
            rospy.sleep(0.05)

        if fb_msg:
            # ⚠️ 단위만 m로 변환 – 축변환 금지
            dx = fb_msg.x / 1000.0
            dy = fb_msg.y / 1000.0
            rospy.loginfo(f"📍 grid1_fallback offset: dx={dx:.3f} m, dy=
{dy:.3f} m")

            self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
            rospy.sleep(1.0)
            self.ur5e.go_to_pose_rel([0, 0, -0.115], [0, 0, 0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])
        else:
            rospy.logwarn("⚠️ No /grid1_fallback/result → skipping pre-pick")

        # ⑤ RED DETECTION 기반 미세조정
        self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
        rospy.loginfo("⏸ Waiting 1.5s for camera stabilization...")
        rospy.sleep(0.5)

        rospy.loginfo("🕐 Waiting for /red_detection/result ...")
        red_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 3.0 and not
rospy.is_shutdown():
            msg = self.red_result.get()
            if msg and msg.x != 0.0 and msg.y != 0.0:
                red_msg = msg
                rospy.loginfo("🕐 Red detected – waiting 1.5s for EMA
stabilization...")
                rospy.sleep(0.5)   # ✅ EMA가 평균값으로 안정화될 시간을 확보
                break
            rospy.sleep(0.1)

        # ⚠️ red 감지 실패 시 → state201
        if not red_msg:
```

```
            rospy.logwarn("❌ No /red_detection/result → state201")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 201
            return

        dx = red_msg.x / 1000.0
        dy = red_msg.y / 1000.0
        rospy.loginfo(f"📍 Red offset: dx={dx:.3f} m, dy={dy:.3f} m")

        try:
            rospy.loginfo(f"🚀 RELATIVE MOVE START: X={dx:.4f}m, Y={dy:.4f}m,
Z=0.0m")
            self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
            rospy.sleep(0.3)
        except Exception as e:
            rospy.logwarn(f"⚠️ XY offset move failed: {e}")

        # ⑥ Grip off
        self.ur5e.go_to_pose_rel([0, 0, -0.04], [0, 0, 0])
        self.grip_off_6s()
        self.ur5e.go_to_pose_rel([0, 0, 0.04], [0, 0, 0])

        # ⑦ 종료 및 다음 단계
        rospy.loginfo("✅ state_6 complete → move to state 1")

        ###### ⭐ 여기 추가 ######
        rospy.sleep(1.0)
        self.open_stopper()
        rospy.sleep(1.0)
        ###### ⭐ 여기까지 ######

        self.state = 1


    def state_101(self):
        rospy.loginfo("➡️ Enter state_101()")

        # ✅ 시작 포즈로 이동
        self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")
        rospy.sleep(1.0)
```

```
        has_red = self.point_detected(self.red_result)
        if has_red:
            rospy.loginfo("🔴 Red detected immediately → state 103")
            self.state = 103
            rospy.loginfo("➡️ state 103")
            return   # ← 이후 로직 절대 실행되지 않게

        # ------------------------------------------------------------
        # ② 감지 확인 (노이즈 방어 포함)
        # ------------------------------------------------------------
        rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
        fb_msg = None
        recent_points = []
        t0 = rospy.Time.now().to_sec()

        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.grid1_fallback.get()
            if not msg or (msg.x == 0.0 and msg.y == 0.0):
                rospy.sleep(0.05)
                continue

            if abs(msg.x) > 200 or abs(msg.y) > 200:
                rospy.logwarn(f"⚠️ Ignored abnormal fallback detection: x=
{msg.x:.1f}, y={msg.y:.1f}")
                continue

            recent_points.append((msg.x, msg.y))
            if len(recent_points) > 15:
                recent_points.pop(0)

            if len(recent_points) >= 3:
                xs, ys = zip(*recent_points)
                std_x, std_y = np.std(xs), np.std(ys)
                if std_x < 2.0 and std_y < 2.0:  # mm 기준 안정 조건
                    fb_msg = msg
                    rospy.loginfo("✅ Stable fallback detection confirmed.")
                    break

            rospy.sleep(0.05)

        has_red = self.point_detected(self.red_result)
        has_gray_circle = fb_msg is not None

        # ------------------------------------------------------------
        # ❤️ 회색 원 감지 성공 (EMA 기반 안정화 포함)
```

```
        # ------------------------------------------------------------
        if has_gray_circle:
            rospy.loginfo("🩶 Gray circle detected — starting EMA
stabilization check")

            rospy.sleep(0.5)

        # ✅ 최신 감지값으로 보정
        msg = self.grid1_fallback.get()
        if msg and (msg.x != 0.0 or msg.y != 0.0):
            fb_msg = msg
            rospy.loginfo("🔄 Updated fallback position after delay.")

            last_points = []
            stable = False
            start_time = rospy.Time.now().to_sec()

            # EMA 안정화 루프
            while not rospy.is_shutdown():
                msg = self.grid1_fallback.get()
                if msg and (msg.x != 0.0 or msg.y != 0.0):
                    last_points.append((msg.x, msg.y))
                    if len(last_points) > 5:
                        last_points.pop(0)

                    if len(last_points) >= 3:
                        diffs = []
                        for i in range(1, len(last_points)):
                            dx = last_points[i][0] - last_points[i - 1][0]
                            dy = last_points[i][1] - last_points[i - 1][1]
                            diffs.append((dx**2 + dy**2) ** 0.5)
                        avg_move = sum(diffs) / len(diffs)
                        rospy.loginfo_throttle(1.0, f"📉 EMA Δ movement avg=
{avg_move:.2f} mm")

                        if avg_move < 3.0:  # 1mm 이하 변화 → 안정화 완료
                            stable = True
                            break

                if rospy.Time.now().to_sec() - start_time > 6.0:
                    rospy.logwarn("⏱️ EMA stabilization timeout — proceeding
anyway")
                    break

                rospy.sleep(0.1)
```

```
            if stable:
                rospy.loginfo("✅ EMA stabilized — executing pickup")
            else:
                rospy.logwarn("⚠️ Proceeding without full EMA stabilization")

            dx = fb_msg.x / 1000.0
            dy = fb_msg.y / 1000.0
            rospy.loginfo(f"📍 Gray circle offset: dx={dx:.3f} m, dy={dy:.3f}
m")

            rospy.sleep(0.3)

            # XY 이동 → Z 하강 → Grip → 상승
            self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
            rospy.sleep(0.3)
            self.ur5e.go_to_pose_rel([0, 0, -0.07], [0, 0, 0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])

            # 다음 단계
            self.goJ(self.OBS_GRID_JOINTS, "OBS_GRID_JOINTS")
            rospy.sleep(0.3)
            self.state = 102
            rospy.loginfo("➡️ state 102")

        # ------------------------------------------------------------
        # 🔴 빨간 원 감지 시 (EMA 없이 즉시)
        # ------------------------------------------------------------
        elif has_red and not has_gray_circle:
            rospy.loginfo("⚠️ No gray circle, but red detected → state 103")
            self.state = 103
            rospy.loginfo("➡️ state 103")

        # ------------------------------------------------------------
        # 🚫 아무 감지도 없음
        # ------------------------------------------------------------
        else:
            rospy.loginfo("⏸ state101 hold")


    def state_102(self):
        msg = self.inv_gray_result.get()
        cell = self.choose_cell_from_string(msg.data if msg else "")
        if cell:
            self.pick_cycle(self._FALLBACK_POSES, cell, grip_mode="off")
        else:
```

```
            rospy.loginfo("⏸ state102 no inv-gray")
        self.state = 103
        rospy.loginfo("➡️  state 103")

    def state_103(self):
        rospy.loginfo("➡️ Enter state_103()")

        # ✅ 감지 포즈 이동
        self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
        has_red = self.point_detected(self.red_result)
        if has_red:
            rospy.loginfo("🔴 Red detected immediately → state 103")
            self.state = 105
            rospy.loginfo("➡️ state 103")
            return   # ← 이후 로직 절대 실행되지 않게

        # ------------------------------------------------------------
        # ② 감지 확인
        # ------------------------------------------------------------
        rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
        fb_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.grid1_fallback.get()
            if msg and (msg.x != 0.0 or msg.y != 0.0):
                fb_msg = msg
                break
            rospy.sleep(0.05)

        has_red = self.point_detected(self.red_result)
        has_gray_circle = fb_msg is not None  # grid1_fallback 감지 성공 시 True

        # ------------------------------------------------------------
        # 🩶 회색 원 감지 성공 (grid1_fallback + EMA 안정화)
        # ------------------------------------------------------------
        if has_gray_circle:
            rospy.loginfo("🩶 Gray circle detected — starting EMA
stabilization check")

            rospy.sleep(0.5)

        # ✅ 최신 감지값으로 보정
        msg = self.grid1_fallback.get()
        if msg and (msg.x != 0.0 or msg.y != 0.0):
```

```
            fb_msg = msg
            rospy.loginfo("🔄 Updated fallback position after delay.")

            last_points = []
            stable = False
            start_time = rospy.Time.now().to_sec()

            # EMA 안정화 루프
            while not rospy.is_shutdown():
                msg = self.grid1_fallback.get()
                if msg and (msg.x != 0.0 or msg.y != 0.0):
                    last_points.append((msg.x, msg.y))
                    if len(last_points) > 5:
                        last_points.pop(0)

                    if len(last_points) >= 3:
                        diffs = []
                        for i in range(1, len(last_points)):
                            dx = last_points[i][0] - last_points[i - 1][0]
                            dy = last_points[i][1] - last_points[i - 1][1]
                            diffs.append((dx**2 + dy**2) ** 0.5)
                        avg_move = sum(diffs) / len(diffs)
                        rospy.loginfo_throttle(1.0, f"📉 EMA Δ movement avg=
{avg_move:.2f} mm")

                        if avg_move < 3.0:  # 1mm 이하 변화 → 안정화 완료
                            stable = True
                            break

                if rospy.Time.now().to_sec() - start_time > 6.0:
                    rospy.logwarn("⏱️ EMA stabilization timeout — proceeding
anyway")
                    break

                rospy.sleep(0.1)

            if stable:
                rospy.loginfo("✅ EMA stabilized — executing pickup")
            else:
                rospy.logwarn("⚠️ Proceeding without full EMA stabilization")

            dx = fb_msg.x / 1000.0
            dy = fb_msg.y / 1000.0
            rospy.loginfo(f"📍 Gray circle offset: dx={dx:.3f} m, dy={dy:.3f}
m")
```

```
            # 1️⃣ XY 평면 이동
            rospy.loginfo("➡️ Moving in XY plane first...")
            self.ur5e.go_to_pose_rel(
                relative_xyz=[dx, dy, 0.0],
                relative_rpy=[0.0, 0.0, 0.0]
            )

            # 2️⃣ 안정화 대기
            rospy.sleep(0.5)

            # 3️⃣ Z축 하강
            rospy.loginfo("⬇️ Descend 70 mm (Z only)")
            self.ur5e.go_to_pose_rel(
                relative_xyz=[0.0, 0.0, -0.07],
                relative_rpy=[0.0, 0.0, 0.0]
            )

            # 4️⃣ Grip ON
            rospy.loginfo("✊ Gripper ON")
            self.grip_on_6s()

            # 5️⃣ 상승
            rospy.loginfo("⬆️ Lift up 115 mm (Z only)")
            self.ur5e.go_to_pose_rel(
                relative_xyz=[0.0, 0.0, 0.115],
                relative_rpy=[0.0, 0.0, 0.0]
            )

            # 6️⃣ 관찰 자세로 복귀 및 다음 단계 전환
            self.goJ(self.OBS_GRID_JOINTS, "OBS_GRID_JOINTS")
            rospy.sleep(0.3)
            self.state = 104
            rospy.loginfo("↩️  state 102")

        # ------------------------------------------------------------
        # 🔴 회색 원(X), 빨간 원(O) → state 105
        # ------------------------------------------------------------
        elif has_red and not has_gray_circle:
            rospy.loginfo("⚠️ No gray circle, but red detected → state 105")
            self.state = 105
            rospy.loginfo("➡️  state 105")

        # ------------------------------------------------------------
        # 🚫 아무 감지도 없음 → hold
        # ------------------------------------------------------------
        else:
```

```
            rospy.loginfo("⏸ state103 hold")



    def state_104(self):
        msg = self.inv_gray_result.get()
        cell = self.choose_cell_from_string(msg.data if msg else "")
        if cell:
            self.pick_cycle(self._FALLBACK_POSES, cell, grip_mode="off")
        else:
            rospy.loginfo("⏸ state104 no inv-gray")
        self.state = 105
        rospy.loginfo("➡️  state 105")

    def state_105(self):
        rospy.loginfo("➡️ Enter state_105()")

        # ① 관찰 포즈 이동
        self.goJ(self.ASM_OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")
        rospy.sleep(0.5)

        # ② inverse grid 감지 시도
        rospy.loginfo("🕐 Waiting for /inverse_grid_detection_result ...")
        inv_grid_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.inv_grid_result.get()
            if msg and msg.data and msg.data.lower() != "none":
                inv_grid_msg = msg
                break
            rospy.sleep(0.1)

        if inv_grid_msg:
            cell = inv_grid_msg.data.strip()
            rospy.loginfo(f"✅ Inverse grid cell detected: {cell}")

            if cell in self.DIS_GRID_POSES:
                start_pose = self.DIS_GRID_POSES[cell]["start"]
                self.goJ(start_pose, f"{cell} start (inverse grid)")
                rospy.sleep(0.5)

                # ③ grid1_fallback 상대좌표 보정
                rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
                fb_msg = None
                t0 = rospy.Time.now().to_sec()
```

```
                while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
                    msg = self.grid2_fallback.get()
                    if msg and (msg.x != 0.0 or msg.y != 0.0):
                        fb_msg = msg
                        break
                    rospy.sleep(0.05)

                if fb_msg:
                    dx = fb_msg.x / 1000.0
                    dy = fb_msg.y / 1000.0
                    rospy.loginfo(f"📍 grid1_fallback offset: dx={dx:.3f} m,
dy={dy:.3f} m")

                    self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
                    rospy.sleep(0.3)
                    self.ur5e.go_to_pose_rel([0, 0, -0.115], [0, 0, 0])
                    self.grip_on_6s()
                    self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])
                else:
                    rospy.logwarn("⚠️ No /grid1_fallback/result → skipping
pre-pick")

                # ④ ASM_OBS_START_JOINTS로 복귀 후 red detection 수행
                self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")
                rospy.sleep(0.5)

                rospy.loginfo("🕐 Waiting for /red_detection/result ...")
                red_msg = None
                t0 = rospy.Time.now().to_sec()
                while (rospy.Time.now().to_sec() - t0) < 3.0 and not
rospy.is_shutdown():
                    msg = self.red_result.get()
                    if msg and msg.x != 0.0 and msg.y != 0.0:
                        red_msg = msg
                        break
                    rospy.sleep(0.1)

                if red_msg:
                    dx = red_msg.x / 1000.0
                    dy = red_msg.y / 1000.0
                    rospy.loginfo(f"📍 Red offset: dx={dx:.3f} m, dy={dy:.3f}
m")

                    try:
                        rospy.loginfo(f"🚀 RELATIVE MOVE START: X={dx:.4f}m,
```

```
Y={dy:.4f}m, Z=0.0m")
                        self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
                        rospy.sleep(0.3)
                    except Exception as e:
                        rospy.logwarn(f"⚠️ XY offset move failed: {e}")

                    # ⑤ 배치 및 복귀
                    self.ur5e.go_to_pose_rel([0, 0, -0.04], [0, 0, 0])
                    self.grip_off_6s()
                    self.ur5e.go_to_pose_rel([0, 0, 0.04], [0, 0, 0])

                    # ✅ state 전환
                    self.state = 106
                    rospy.loginfo("➡️ state 106")
                else:
                    rospy.logwarn("❌ No /red_detection/result after inverse
pick – skipping placement")
                    self.state = 1
                    rospy.loginfo("↩️ Return to state 1 (no red detection)")

            else:
                rospy.logwarn(f"⚠️ Invalid grid cell name: {cell}")
                self.state = 1
                rospy.loginfo("↩️ Return to state 1 (invalid cell)")

        else:
            rospy.logwarn("❌ No inverse grid detection result received")
            self.state = 1
            rospy.loginfo("↩️ Return to state 1 (no inverse grid)")



    def state_106(self):
        rospy.loginfo("➡️ Enter state_106()")

        # ① 관찰 포즈 이동 (inverse grid 감지용)
        self.goJ(self.ASM_OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")
        rospy.sleep(0.5)

        # ② inverse grid 감지 시도
        rospy.loginfo("🕐 Waiting for /inverse_grid_detection_result ...")
        inv_grid_msg = None
        t0 = rospy.Time.now().to_sec()
        while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
            msg = self.inv_grid_result.get()
```

```
            if msg and msg.data and msg.data.lower() != "none":
                inv_grid_msg = msg
                break
            rospy.sleep(0.1)

        if inv_grid_msg:
            cell = inv_grid_msg.data.strip()
            rospy.loginfo(f"✅ Inverse grid cell detected: {cell}")

            if cell in self.DIS_GRID_POSES:
                start_pose = self.DIS_GRID_POSES[cell]["start"]
                self.goJ(start_pose, f"{cell} start (inverse grid)")
                rospy.sleep(0.5)

                # ③ grid1_fallback 상대좌표 보정
                rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
                fb_msg = None
                t0 = rospy.Time.now().to_sec()
                while (rospy.Time.now().to_sec() - t0) < 2.5 and not
rospy.is_shutdown():
                    msg = self.grid2_fallback.get()
                    if msg and (msg.x != 0.0 or msg.y != 0.0):
                        fb_msg = msg
                        break
                    rospy.sleep(0.05)

                if fb_msg:
                    dx = fb_msg.x / 1000.0
                    dy = fb_msg.y / 1000.0
                    rospy.loginfo(f"📍 grid1_fallback offset: dx={dx:.3f} m,
dy={dy:.3f} m")

                    self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
                    rospy.sleep(0.3)
                    self.ur5e.go_to_pose_rel([0, 0, -0.115], [0, 0, 0])
                    self.grip_on_6s()
                    self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])
                else:
                    rospy.logwarn("⚠️ No /grid1_fallback/result → skipping
pre-pick")

                # ④ ASM_RED_DETECT_TRIGGER_JOINTS로 복귀 후 red detection 수행
                self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
                rospy.sleep(0.5)
```

```
                rospy.loginfo("🕐 Waiting for /red_detection/result ...")
                red_msg = None
                t0 = rospy.Time.now().to_sec()
                while (rospy.Time.now().to_sec() - t0) < 3.0 and not
rospy.is_shutdown():
                    msg = self.red_result.get()
                    if msg and msg.x != 0.0 and msg.y != 0.0:
                        red_msg = msg
                        break
                    rospy.sleep(0.1)

                if red_msg:
                    dx = red_msg.x / 1000.0
                    dy = red_msg.y / 1000.0
                    rospy.loginfo(f"📍 Red offset: dx={dx:.3f} m, dy={dy:.3f}
m")

                    try:
                        rospy.loginfo(f"🚀 RELATIVE MOVE START: X={dx:.4f}m,
Y={dy:.4f}m, Z=0.0m")
                        self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0, 0, 0])
                        rospy.sleep(0.5)
                    except Exception as e:
                        rospy.logwarn(f"⚠️ XY offset move failed: {e}")

                    # ⑤ 배치 및 복귀
                    self.ur5e.go_to_pose_rel([0, 0, -0.04], [0, 0, 0])
                    self.grip_off_6s()
                    self.ur5e.go_to_pose_rel([0, 0, 0.04], [0, 0, 0])

                    ###### ⭐ 여기 추가 ######
                    rospy.sleep(1.0)
                    self.open_stopper()
                    rospy.sleep(1.0)
                    ###### ⭐ 여기까지 ######

                    # ✅ state 전환
                    self.state = 101
                    rospy.loginfo("➡️ state 101")
                else:
                    rospy.logwarn("❌ No /red_detection/result after inverse
pick — skipping placement")

                    ###### ⭐ 여기 추가 ######
                    rospy.sleep(1.0)
                    self.open_stopper()
```

```
                    rospy.sleep(1.0)
                    ###### ⭐ 여기까지 ######

                    self.state = 301
                    rospy.loginfo("↩️ Move to state 301 (no red detection)")

            else:
                rospy.logwarn(f"⚠️ Unknown grid cell name: {cell}")

                ###### ⭐ 여기 추가 ######
                rospy.sleep(1.0)
                self.open_stopper()
                rospy.sleep(1.0)
                ###### ⭐ 여기까지 ######

                self.state = 301
                rospy.loginfo("↩️ Move to state 301 (invalid cell)")

        else:
            # ❌ inverse grid 감지 실패 시
            rospy.logwarn("❌ No inverse grid detection result received")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 301
            rospy.loginfo("↩️ Move to state 301 (no inverse grid)")


    def state_201(self):
        rospy.loginfo("➡️ Enter state_201()")

        # ① 시작 자세
        self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")
        rospy.sleep(3.0)

        # ② 감지
        msg_orange = self.orange_result.get()
        has_orange = self.point_detected(self.orange_result)
        has_red    = self.point_detected(self.red_result)

        # 🍊 오렌지 감지 시 → pick & place → state 202
        if has_orange and msg_orange:
```

```
            dx = msg_orange.x
            dy = msg_orange.y

            # 비정상 좌표 방지
            if abs(dx) > 200 or abs(dy) > 200:
                rospy.logwarn(f"⚠️ Ignored abnormal orange detection: x=
{dx:.1f}, y={dy:.1f}")
                return

            # ============================================================
            # 🔶🔶🔶 ① EMA 초기화 (state_1 그대로)
            # ============================================================
            if self.prev_orange is None:
                self.prev_orange = np.array([dx, dy], dtype=float)
                self.ema_stable_count = 0
                rospy.loginfo("📊 EMA 초기화 중... (state_201)")
                return

            # ============================================================
            # 🔶🔶🔶 ② EMA 적용
            # ============================================================
            curr = np.array([dx, dy], dtype=float)
            self.prev_orange = (1 - self.ema_alpha) * self.prev_orange +
self.ema_alpha * curr

            diff = np.linalg.norm(curr - self.prev_orange)

            if diff < self.ema_stable_thresh_mm:
                self.ema_stable_count += 1
            else:
                self.ema_stable_count = 0

            rospy.loginfo_throttle(
                1.0,
                f"📈 (state201) EMA 안정도: diff={diff:.2f}mm, count=
{self.ema_stable_count}"
            )

            # ============================================================
            # 🔶🔶🔶 ③ EMA 안정화 대기
            # ============================================================
            if self.ema_stable_count < self.ema_min_frames:
                rospy.loginfo("⏸ (state201) EMA 안정화 대기 중…")
                return
```

```
            rospy.loginfo("✅ (state201) EMA 안정화 완료 → 픽업 시작")

            # ============================================================
            # 🔶 EMA 안정된 좌표 사용
            # ============================================================
            dx_m = self.prev_orange[0] / 1000.0
            dy_m = self.prev_orange[1] / 1000.0

            # ================= 원본 로직 그대로 유지 ====================
            self.ur5e.go_to_pose_rel([dx_m, dy_m, 0,], [0,0,0])
            rospy.sleep(0.8)
            self.ur5e.go_to_pose_rel([0,0,-0.07],[0,0,0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0,0,0.115],[0,0,0])

            # Grid 놓기
            self.goJ(self.ASM_OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")
            msg = self.darkgrid_result.get()
            cell = self.choose_cell_from_string(msg.data if msg else "")

            if cell and cell in self.DARKGRID_POSES:
                start_pose = self.DARKGRID_POSES[cell]["start"]
                pick_pose  = self.DARKGRID_POSES[cell]["pick"]
                self.goJ(start_pose, f"{cell} start")
                rospy.sleep(0.5)
                self.goJ(pick_pose, f"{cell} place")
                self.grip_off_6s()
                rospy.sleep(0.5)
                self.goJ(start_pose, f"{cell} return start")
            else:
                rospy.logwarn("❌ No valid dark grid detected.")

            # 다음 단계
            self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
            self.state = 202
            rospy.loginfo("➡️ Transition to state_202")

        # 🔴 빨간색만 감지
        elif has_red and not has_orange:
            rospy.loginfo("🔴 Red only detected → move to red trigger (state
202)")
            self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
            self.state = 202
```

```
        # 🚫 아무 감지도 없음
        else:
            rospy.loginfo("⏸ state201 hold (no orange/red detected)")




    # ============================================================
    # 🍊 state_202: 두 번째 오렌지 pick + grid 놓기 + state101로 복귀
    # ============================================================
    def state_202(self):
        rospy.loginfo("➡️ Enter state_202()")

        msg_orange2 = self.orange_result.get()
        has_orange2 = self.point_detected(self.orange_result)
        has_red2    = self.point_detected(self.red_result)

        # 🍊 두 번째 오렌지 감지 → pick & place → state101
        if has_orange2 and msg_orange2:
            dx = msg_orange2.x / 1000.0
            dy = msg_orange2.y / 1000.0
            rospy.sleep(2.0)
            rospy.loginfo(f"🍊 (Second) Orange detected: dx={dx:.4f}, dy=
{dy:.4f}")

            # 픽업
            self.ur5e.go_to_pose_rel([dx, dy, 0], [0,0,0])
            rospy.sleep(0.8)
            self.ur5e.go_to_pose_rel([0,0,-0.07],[0,0,0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0,0,0.115],[0,0,0])

            # Grid 놓기
            self.goJ(self.ASM_OBS_GRID_JOINTS, "ASM_OBS_GRID_JOINTS")
            msg2 = self.darkgrid_result.get()
            cell2 = self.choose_cell_from_string(msg2.data if msg2 else "")
            if cell2 and cell2 in self.DARKGRID_POSES:
                start_pose2 = self.DARKGRID_POSES[cell2]["start"]
                pick_pose2  = self.DARKGRID_POSES[cell2]["pick"]
                self.goJ(start_pose2, f"{cell2} start")
                rospy.sleep(0.5)
                self.goJ(pick_pose2, f"{cell2} place")
                self.grip_off_6s()
                rospy.sleep(0.5)
                self.goJ(start_pose2, f"{cell2} return start")
            else:
```

```
                rospy.logwarn("❌ No valid dark grid detected in state_202.")

            # 완료 후 state101로 복귀
            self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 101
            rospy.loginfo("✅ Finished second orange → state101")

        # 🔴 빨간색만 감지 → state101로 복귀
        elif has_red2 and not has_orange2:
            rospy.loginfo("🔴 Red only detected → state101")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 101

        # 🚫 아무 감지도 없음 → hold
        else:
            rospy.loginfo("⏸ state202 hold (no orange/red detected)")


# ============================================================
# 🩶 state_301: 첫 번째 회색 원 감지 및 픽업 → grid에 놓기 → state_302 전환
# ============================================================
def state_301(self):
    rospy.loginfo("➡️ Enter state_301()")

    # ① 시작 자세
    self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")
    rospy.sleep(3.0)

    # ② 회색 원 감지 시도 (/grid1_fallback/result)
    rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
    fb_msg = None
    recent_points = []
    t0 = rospy.Time.now().to_sec()
```

```
        while (rospy.Time.now().to_sec() - t0) < 5.0 and not
rospy.is_shutdown():
            msg = self.grid1_fallback.get()
            if not msg or (msg.x == 0.0 and msg.y == 0.0):
                rospy.sleep(0.05)
                continue

            # 🚧 Outlier filter
            if abs(msg.x) > 200 or abs(msg.y) > 200:
                rospy.logwarn(f"⚠️ Ignored abnormal fallback detection: x=
{msg.x:.1f}, y={msg.y:.1f}")
                continue

            # 🚧 Stability filter
            recent_points.append((msg.x, msg.y))
            if len(recent_points) > 15:
                recent_points.pop(0)

            if len(recent_points) >= 4:
                xs, ys = zip(*recent_points)
                std_x, std_y = np.std(xs), np.std(ys)
                if std_x < 3.0 and std_y < 3.0:
                    fb_msg = msg
                    rospy.loginfo("✅ Stable fallback detection confirmed.")
                    break
            rospy.sleep(0.05)

        has_red = self.point_detected(self.red_result)
        has_gray_circle = fb_msg is not None

        # ------------------------------------------------------------
        # 🩶 회색 원 감지 성공 → 픽업 절차 수행
        # ------------------------------------------------------------
        if has_gray_circle:
            dx = fb_msg.x / 1000.0
            dy = fb_msg.y / 1000.0
            rospy.loginfo(f"📍 Gray circle offset: dx={dx:.3f} m, dy={dy:.3f}
m")
            rospy.sleep(2.0)

            # XY 이동
            rospy.loginfo("➡️ Moving in XY plane first...")
            self.ur5e.go_to_pose_rel([dx, dy, 0.0], [0.0, 0.0, 0.0])
            rospy.sleep(0.8)
```

```
            # 하강 → Grip ON → 상승
            self.ur5e.go_to_pose_rel([0, 0, -0.07], [0, 0, 0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])

            # 관찰자세 이동 후 그리드로 이동
            self.goJ(self.OBS_GRID_JOINTS, "OBS_GRID_JOINTS")
            msg = self.inv_gray_result.get()
            cell = self.choose_cell_from_string(msg.data if msg else "")

            if cell and cell in self._FALLBACK_POSES:
                start_pose = self._FALLBACK_POSES[cell]["start"]
                pick_pose = self._FALLBACK_POSES[cell]["pick"]
                self.goJ(start_pose, f"{cell} start (gray)")
                rospy.sleep(0.5)
                self.goJ(pick_pose, f"{cell} pick (gray)")
                rospy.loginfo("🖐 Grip OFF (6s)")
                self.grip_off_6s()
                rospy.sleep(0.5)
                self.goJ(start_pose, f"{cell} return start (gray)")
            else:
                rospy.logwarn("❌ No valid inv-gray grid detected.")

            # ✅ 다음 단계로 전환
            self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
            self.state = 302
            rospy.loginfo("➡️ Move to state_302")

        # 🔴 빨간 원만 감지 → 픽 없이 바로 state_302 전환
        elif has_red and not has_gray_circle:
            rospy.loginfo("🔴 Red detected only → move to
ASM_RED_DETECT_TRIGGER_JOINTS")
            self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")
            self.state = 302
            rospy.loginfo("➡️ state_302")

        # 🚫 아무 감지도 없음
        else:
            rospy.loginfo("⏸ state301 hold (no detection)")


    # ============================================================
    # ❤️ state_302: 두 번째 회색 원 감지 및 픽업 → grid에 놓기 → state_1 복귀
    # ============================================================
```

```python
    def state_302(self):
        rospy.loginfo("➡️ Enter state_302()")

        # ① 시작 포즈 (레드 트리거 위치)
        self.goJ(self.ASM_RED_DETECT_TRIGGER_JOINTS,
"ASM_RED_DETECT_TRIGGER_JOINTS")

        # ② 감지 시도
        rospy.loginfo("🕐 Waiting for /grid1_fallback/result ...")
        fb_msg = None
        recent_points = []
        t0 = rospy.Time.now().to_sec()

        while (rospy.Time.now().to_sec() - t0) < 5.0 and not
rospy.is_shutdown():
            msg = self.grid1_fallback.get()
            if not msg or (msg.x == 0.0 and msg.y == 0.0):
                rospy.sleep(0.05)
                continue

            if abs(msg.x) > 200 or abs(msg.y) > 200:
                rospy.logwarn(f"⚠️ Ignored abnormal fallback detection: x=
{msg.x:.1f}, y={msg.y:.1f}")
                continue

            recent_points.append((msg.x, msg.y))
            if len(recent_points) > 15:
                recent_points.pop(0)

            if len(recent_points) >= 4:
                xs, ys = zip(*recent_points)
                std_x, std_y = np.std(xs), np.std(ys)
                if std_x < 3.0 and std_y < 3.0:
                    fb_msg = msg
                    rospy.loginfo("✅ Stable fallback detection confirmed.")
                    break
            rospy.sleep(0.05)

        has_red = self.point_detected(self.red_result)
        has_gray_circle = fb_msg is not None

        # 🩶 회색 원 감지 성공 시 → 픽업 후 grid 놓기
        if has_gray_circle:
            dx = fb_msg.x / 1000.0
            dy = fb_msg.y / 1000.0
            rospy.loginfo(f"📍 Gray circle offset: dx={dx:.3f} m, dy={dy:.3f}
```

```
m")
            rospy.sleep(2.0)

            self.ur5e.go_to_pose_rel([dx, dy, 0], [0, 0, 0])
            rospy.sleep(0.8)
            self.ur5e.go_to_pose_rel([0, 0, -0.07], [0, 0, 0])
            self.grip_on_6s()
            self.ur5e.go_to_pose_rel([0, 0, 0.115], [0, 0, 0])

            # 관찰자세로 복귀 후 grid 놓기
            self.goJ(self.OBS_GRID_JOINTS, "OBS_GRID_JOINTS")
            msg = self.inv_gray_result.get()
            cell = self.choose_cell_from_string(msg.data if msg else "")

            if cell and cell in self._FALLBACK_POSES:
                start_pose = self._FALLBACK_POSES[cell]["start"]
                pick_pose = self._FALLBACK_POSES[cell]["pick"]
                self.goJ(start_pose, f"{cell} start (gray2)")
                rospy.sleep(0.5)
                self.goJ(pick_pose, f"{cell} pick (gray2)")
                rospy.loginfo("🖐 Grip OFF (6s)")
                self.grip_off_6s()
                rospy.sleep(0.5)
                self.goJ(start_pose, f"{cell} return start (gray2)")
            else:
                rospy.logwarn("❌ No valid inv-gray grid detected
(state302).")

            # ✅ 완료 후 state 1 복귀
            self.goJ(self.ASM_OBS_START_JOINTS, "ASM_OBS_START_JOINTS")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 1
            rospy.loginfo("✅ Finished second gray → state 1")

        # 🔴 빨간 원만 감지 → pick 없이 바로 state 1 전환
        elif has_red and not has_gray_circle:
            rospy.loginfo("🔴 Red only detected → skip pick → state 1")

            ###### ⭐ 여기 추가 ######
            rospy.sleep(1.0)
```

```
            self.open_stopper()
            rospy.sleep(1.0)
            ###### ⭐ 여기까지 ######

            self.state = 1

        # 🚫 아무 감지도 없음 → hold
        else:
            rospy.loginfo("⏸ state302 hold (no detection)")



if __name__ == "__main__":
    node = AsmFsmNode()
    node.spin()
```

winding_detect_node.py

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy, cv2, numpy as np, os, math
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from geometry_msgs.msg import Point

class StableGrayDetectionNode:
    def __init__(self):
        rospy.init_node('stable_gray_detection_node', anonymous=True)
        self.bridge = CvBridge()
        self.detection_pub = rospy.Publisher('/orange_detection/result',
Point, queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # ============================================================
        # 🧩 사용자 조정 파라미터 (변경 금지)
        # ============================================================
        self.ROI_WIDTH = 400
        self.ROI_HEIGHT = 400
        self.TARGET_SIZE = (1200, 1200)
        self.CENTER_PX = (600, 600)
        self.DETECTION_OFFSET = 300
        self.scale_x = 0.115
        self.scale_y = 0.115
        self.CAMERA_OFFSET_X_MM = 87.0
```

```
        self.CAMERA_OFFSET_Y_MM = 2.0
        self.MIN_RADIUS = 250
        self.MAX_RADIUS = 320
        self.CENTER_TOL_MM = 60.0
        self.LOWER_GRAY = 140
        self.UPPER_GRAY = 255
        self.HOUGH_DP = 1
        self.HOUGH_MIN_DIST = 100
        self.HOUGH_PARAM1 = 50
        self.HOUGH_PARAM2 = 20
        self.prev_center = None
        self.prev_radius = None
        self.alpha = 0.2
        self.ORANGE_RATIO_THRESH = 0.03
        self.ORANGE_MIN = (50, 80, 240)
        self.ORANGE_MAX = (120, 150, 255)

        # ============================================================
        # 📂 카메라 캘리브레이션
        # ============================================================
        calib_path =
"/home/kiminyeop/catkin_ws/src/ur_python/src/dataset_bra2_checkboard3/circlegr
id_calibration_results_8x6_20mm.npz"
        if os.path.exists(calib_path):
            data = np.load(calib_path)
            self.camera_matrix = data["camera_matrix"]
            self.dist_coeffs = data["dist_coeffs"]
            rospy.loginfo(f"📷 Loaded camera calibration from: {calib_path}")
        else:
            rospy.logwarn("⚠️ Calibration file not found — skipping
undistortion.")
            self.camera_matrix, self.dist_coeffs = None, None

        rospy.loginfo("🔴 Stable Gray Detection Node initialized (Hough +
Orange area + dx/dy display)")

    # ------------------------------------------------------------
    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
            if self.camera_matrix is not None:
                frame = cv2.undistort(frame, self.camera_matrix,
self.dist_coeffs)

            H, W = frame.shape[:2]
            x1 = int(W/2 - self.ROI_WIDTH/2)
```

```
            y1 = int(H/2 - self.ROI_HEIGHT/2)
            cropped = frame[y1:y1+self.ROI_HEIGHT, x1:x1+self.ROI_HEIGHT]
            resized = cv2.resize(cropped, self.TARGET_SIZE,
interpolation=cv2.INTER_LINEAR)

            self.detect_gray_circle(resized)

        except Exception as e:
            rospy.logerr(f"Gray Detection Error: {e}")

    # ------------------------------------------------------------
    def detect_gray_circle(self, frame):
        output = frame.copy()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray_filtered = cv2.inRange(gray, self.LOWER_GRAY, self.UPPER_GRAY)
        gray_blurred = cv2.GaussianBlur(gray_filtered, (9, 9), 2)

        OFFSET = self.DETECTION_OFFSET
        HOUGH_SIZE = 600
        HOUGH_INPUT = gray_blurred[OFFSET:OFFSET + HOUGH_SIZE, OFFSET:OFFSET +
HOUGH_SIZE]

        circles = cv2.HoughCircles(
            HOUGH_INPUT, cv2.HOUGH_GRADIENT,
            self.HOUGH_DP, self.HOUGH_MIN_DIST,
            param1=self.HOUGH_PARAM1, param2=self.HOUGH_PARAM2,
            minRadius=self.MIN_RADIUS, maxRadius=self.MAX_RADIUS
        )

        best_circle_data = None
        min_dist_to_center = float('inf')

        # 1️⃣ 중앙에 가장 가까운 원 선택
        if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0, :]:
                x_relative, y_relative, r = i[0], i[1], i[2]
                x = x_relative + OFFSET
                y = y_relative + OFFSET
                dx_px = float(x - self.CENTER_PX[0])
                dy_px = float(y - self.CENTER_PX[1])
                dist_px = math.sqrt(dx_px**2 + dy_px**2)
                if dist_px < min_dist_to_center:
                    min_dist_to_center = dist_px
                    best_circle_data = (x, y, r, 1.0)
```

```
        # 2️⃣ 결과 처리
        if best_circle_data is not None:
            x, y, r, _ = best_circle_data

            # EMA 안정화
            if self.prev_center is None:
                self.prev_center = np.array([x, y], dtype=float)
                self.prev_radius = r
            else:
                self.prev_center = (1 - self.alpha) * self.prev_center +
self.alpha * np.array([x, y])
                self.prev_radius = (1 - self.alpha) * self.prev_radius +
self.alpha * r

            x, y = self.prev_center
            r = self.prev_radius

            # 🟠 원 내부 오렌지 비율 계산
            mask = np.zeros(frame.shape[:2], dtype=np.uint8)
            cv2.circle(mask, (int(x), int(y)), int(r), 255, -1)
            orange_mask = cv2.inRange(frame, self.ORANGE_MIN, self.ORANGE_MAX)
            orange_inside = cv2.bitwise_and(orange_mask, orange_mask,
mask=mask)
            orange_ratio = np.sum(orange_inside > 0) / np.sum(mask > 0)
            rospy.loginfo_throttle(1.0, f"🟠 Orange ratio inside circle:
{orange_ratio*100:.2f}%")

            # === 픽셀 → mm 변환 ===
            dx_px = float(x - self.CENTER_PX[0])
            dy_px = float(y - self.CENTER_PX[1])
            dx_mm = dx_px * self.scale_x
            dy_mm = dy_px * self.scale_y
            dist_mm = math.sqrt(dx_mm**2 + dy_mm**2)

            # === 카메라 오프셋 보정 ===
            dx_mm_total = -dy_mm + self.CAMERA_OFFSET_X_MM
            dy_mm_total = -dx_mm + self.CAMERA_OFFSET_Y_MM

            if dist_mm <= self.CENTER_TOL_MM and orange_ratio >
self.ORANGE_RATIO_THRESH:
                cv2.circle(output, (int(x), int(y)), int(r), (0,255,0), 2)
                cv2.circle(output, (int(x), int(y)), 5, (0,0,255), -1)
                cv2.circle(output, self.CENTER_PX, 5, (255,0,0), -1)

                # --- ✅ 화면 표시 부분 수정 (X offset 적용된 값 표시)
                cv2.putText(output, f"dx={dx_mm:.1f} dy={dy_mm:.1f} OR=
```

```
{orange_ratio*100:.1f}%",
                                (25,45), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,0),
2)
                cv2.putText(output, f"XOFF -> dx={dx_mm_total:.1f} dy=
{dy_mm_total:.1f}",
                                (25,85), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
(255,255,0), 2)

                # 퍼블리시
                self.detection_pub.publish(Point(x=float(dx_mm_total),
y=float(dy_mm_total), z=float(r)))
                rospy.loginfo_throttle(1.0, f"✅ Valid Circle + Orange: dx=
{dx_mm_total:.1f}, dy={dy_mm_total:.1f}")
            else:
                rospy.logwarn_throttle(1.0, f"⚠️ Circle ignored (dist=
{dist_mm:.1f}mm, orange={orange_ratio*100:.1f}%)")
                self._publish_fail()
        else:
            rospy.logwarn_throttle(1.0, "❌ No circle found.")
            self._publish_fail()

        # 3️⃣ 시각화
        cv2.rectangle(output, (OFFSET, OFFSET), (OFFSET + HOUGH_SIZE, OFFSET +
HOUGH_SIZE), (255, 0, 0), 2)
        cv2.imshow("Stable Orange Detection (Gray + Orange Filter)", output)
        cv2.imshow("Hough Filtered Input", gray_blurred)
        cv2.imshow("Orange Mask Inside Circle", orange_inside)
        cv2.waitKey(1)

    # ------------------------------------------------------------
    def _publish_fail(self):
        self.detection_pub.publish(Point(x=0.0, y=0.0, z=0.0))

# ------------------------------------------------------------
if __name__ == '__main__':
    try:
        StableGrayDetectionNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        cv2.destroyAllWindows()
```

`yellow_bracket_detection_node.py`

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
```

```python
import rospy, cv2, numpy as np, os, math
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from geometry_msgs.msg import Point

class StableRedDetectionNode:
    def __init__(self):
        rospy.init_node('stable_red_detection_node', anonymous=True)
        self.bridge = CvBridge()
        self.detection_pub = rospy.Publisher('/red_detection/result', Point,
queue_size=1)
        self.image_sub = rospy.Subscriber("/camera/image_raw", Image,
self.image_callback)

        # ============================================================
        # 📂 카메라 캘리브레이션 결과 불러오기
        # ============================================================
        calib_path =
"/home/kiminyeop/catkin_ws/src/ur_python/src/dataset_bra2_checkboard3/circlegr
id_calibration_results_8x6_20mm.npz"
        if os.path.exists(calib_path):
            data = np.load(calib_path)
            self.camera_matrix = data["camera_matrix"]
            self.dist_coeffs = data["dist_coeffs"]
            rospy.loginfo(f"📷 Loaded camera calibration from: {calib_path}")
        else:
            rospy.logwarn("⚠️ Calibration file not found - skipping
undistortion.")
            self.camera_matrix = None
            self.dist_coeffs = None

        # ============================================================
        # ROI 및 화면 설정
        # ============================================================
        self.ROI_WIDTH = 400
        self.ROI_HEIGHT = 400
        self.TARGET_SIZE = (1200, 1200)
        self.CENTER_PX = (600, 600)

        # ============================================================
        # Red HSV 임계값
        # ============================================================
        # Yellow detection (Hue 20~32)
        self.lower_red1 = np.array([20, 120, 120])
        self.upper_red1 = np.array([32, 255, 255])
```

```
        # Not used for yellow, but placeholder to avoid errors
        self.lower_red2 = np.array([0, 0, 0])
        self.upper_red2 = np.array([0, 0, 0])


        # ============================================================
        # Morphology 커널
        # ============================================================
        self.kernel = np.ones((5,5), np.uint8)

        # ============================================================
        # px → mm 변환 비율 (실험값)
        # ============================================================
        self.scale_x = 0.1435  # mm/px
        self.scale_y = 0.1425  # mm/px

        # ============================================================
        # 카메라 오프셋 (Tool0 기준)
        # ============================================================
        self.CAMERA_OFFSET_X_MM = 89.0   # X축 방향 (mm)
        self.CAMERA_OFFSET_Y_MM = 4.0    # Y축 방향 (mm)

        # ============================================================
        # 기울기 보정 (deg)
        # ============================================================
        self.roll_tilt_deg = 0.0   # x축 방향 (좌우)
        self.pitch_tilt_deg = 0.0  # y축 방향 (앞뒤)

        # ============================================================
        # 지수평활 필터 (EMA)
        # ============================================================
        self.prev_center = None
        self.prev_radius = None
        self.alpha = 0.35

        rospy.loginfo("🔴 Stable Red Detection Node initialized (Calibrated +
82 mm offset + tilt correction)")

    # ------------------------------------------------------------
    # 🎥 메인 콜백
    # ------------------------------------------------------------
    def image_callback(self, msg):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")

            # === 📸 왜곡 보정 적용 ===
```

```
            if self.camera_matrix is not None and self.dist_coeffs is not
None:
                frame = cv2.undistort(frame, self.camera_matrix,
self.dist_coeffs)

            # === 중앙 ROI 자르기 ===
            H, W = frame.shape[:2]
            x1 = int(W/2 - self.ROI_WIDTH/2)
            y1 = int(H/2 - self.ROI_HEIGHT/2)
            cropped = frame[y1:y1+self.ROI_HEIGHT, x1:x1+self.ROI_WIDTH]

            # === 1200x1200 확대 ===
            resized = cv2.resize(cropped, self.TARGET_SIZE,
interpolation=cv2.INTER_LINEAR)

            # === 빨간 원 검출 ===
            self.red_detect(resized)

        except Exception as e:
            rospy.logerr(f"Red Detection Error: {e}")

    # ------------------------------------------------------------
    # 🔴 빨간 원 검출
    # ------------------------------------------------------------
    def red_detect(self, frame):
        # --- 영상 안정화 대기 (1 초) ---

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        mask1 = cv2.inRange(hsv, self.lower_red1, self.upper_red1)
        mask2 = cv2.inRange(hsv, self.lower_red2, self.upper_red2)
        mask = cv2.bitwise_or(mask1, mask2)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, self.kernel,
iterations=2)
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, self.kernel,
iterations=1)
        mask = cv2.GaussianBlur(mask, (9,9), 2)

        circles = cv2.HoughCircles(mask, cv2.HOUGH_GRADIENT, dp=1.2,
minDist=190,
                                   param1=100, param2=25, minRadius=210,
maxRadius=430)
        output = frame.copy()

        if circles is not None:
            circles = np.uint16(np.around(circles))[0]
            x, y, r = max(circles, key=lambda c: c[2])
```

```
            # === EMA 안정화 ===
            if self.prev_center is None:
                self.prev_center = np.array([x, y], dtype=float)
                self.prev_radius = r
            else:
                self.prev_center = (1 - self.alpha) * self.prev_center +
self.alpha * np.array([x, y])
                self.prev_radius = (1 - self.alpha) * self.prev_radius +
self.alpha * r

            x, y = self.prev_center
            r = self.prev_radius

            # === 픽셀 → mm 변환 ===
            dx_px = float(x - self.CENTER_PX[0])
            dy_px = float(y - self.CENTER_PX[1])
            dx_mm = dx_px * self.scale_x
            dy_mm = dy_px * self.scale_y

            # === 기울기 보정 ===
            pitch_rad = math.radians(self.pitch_tilt_deg)
            roll_rad  = math.radians(self.roll_tilt_deg)
            dx_mm_corr = dx_mm / math.cos(pitch_rad)
            dy_mm_corr = dy_mm / math.cos(roll_rad)

            # === 오프셋 적용 (카메라 → Tool 기준) ===
            dx_mm_total = -dy_mm_corr + self.CAMERA_OFFSET_X_MM
            dy_mm_total = -dx_mm_corr + self.CAMERA_OFFSET_Y_MM

            # === 시각화 ===
            cv2.circle(output,(int(x),int(y)),int(r),(0,255,0),2)
            cv2.circle(output,(int(x),int(y)),5,(0,0,255),-1)
            cv2.circle(output,self.CENTER_PX,5,(255,0,0),-1)
            cv2.putText(output, f"dx={dx_mm_total:.1f} mm, dy=
{dy_mm_total:.1f} mm",
                        (25,45), cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)

            # === 퍼블리시 ===
            point_msg = Point(x=float(dx_mm_total), y=float(dy_mm_total),
z=float(r))
            self.detection_pub.publish(point_msg)
            rospy.loginfo(f"📍 Published: dx={dx_mm_total:.1f} mm, dy=
{dy_mm_total:.1f} mm, r={r:.1f}")
        else:
            self.detection_pub.publish(Point(x=0.0, y=0.0, z=0.0))
```

```
            rospy.logwarn("❌ Red circle not found, publishing (0,0,0)")

        cv2.imshow("Stable Red Detection (Tilt-corrected + 82 mm offset)",
output)
        cv2.imshow("mask", mask)
        cv2.waitKey(1)


# ------------------------------------------------------------
# 🚀 실행부
# ------------------------------------------------------------
if __name__ == '__main__':
    try:
        StableRedDetectionNode()
        rospy.spin()
    except rospy.ROSInterruptException:
        cv2.destroyAllWindows()
```