# ROS2 Chessboard and Piece Detector for Robot Arm

This ROS2 package contains a single node, object\_recognizer, designed to work with a robot arm and an Intel RealSense depth camera. The node identifies a chessboard and game pieces (white round, black cross) within a specified region of interest, calculates their 3D positions in the robot's world frame, and publishes this information for use in pick-and-place tasks.

!(https://www.google.com/search?q=https://placehold.co/800x300/4F46E5/FFFFFF%3Ftext%3DRobot%2BArm%2BChess%2BVision)

## Features

* **Chessboard Detection**: Identifies the largest rectangular contour in the ROI as the chessboard.
* **Piece Detection**:
  + Detects white, circular pieces using Hough Circle Transform.
  + Detects black, cross-shaped pieces using color and solidity filtering.
* **Region of Interest (ROI)**: All processing is confined to a user-defined ROI for improved performance and accuracy.
* **2D to 3D Conversion**: Uses depth data from a RealSense camera to convert 2D pixel coordinates into 3D points.
* **Coordinate Transformation**: Transforms 3D points from the camera's frame to the robot's base\_link frame.
* **Self-Contained Calibration**: Includes a hard-coded static transform for hand-eye calibration, removing the need for an external static\_transform\_publisher.
* **Data Publishing**:
  + Publishes the board's 3D center point.
  + Publishes an array of all detected pieces' 3D poses.
  + Publishes TF frames for Rviz visualization.
  + Publishes a debug image showing live detections.

## Prerequisites

* ROS2 Humble Hawksbill
* An Intel RealSense Depth Camera
* realsense-ros package installed and configured.
* cv\_bridge, tf2\_ros, tf2\_geometry\_msgs (typically included in ros-humble-desktop-full).
* OpenCV for Python: pip install opencv-python
* NumPy: pip install numpy

## Setup and Installation

1. **Create a ROS2 Workspace**:  
   mkdir -p ~/ros2\_ws/src  
   cd ~/ros2\_ws
2. **Create the Package**:  
   cd src  
   ros2 pkg create --build-type ament\_python my\_robot\_cv
3. **Add the Node**:
   * Place the robot\_arm\_cv.py script inside the my\_robot\_cv/my\_robot\_cv/ directory.
4. **Configure setup.py**:
   * Edit src/my\_robot\_cv/setup.py to add the entry point for the node.

# Inside the console\_scripts list:  
'object\_recognizer = my\_robot\_cv.robot\_arm\_cv:main',

1. **Build the Package**:  
   cd ~/ros2\_ws  
   colcon build --packages-select my\_robot\_cv

## Configuration

Before running the node, you **must** configure the following parameters inside robot\_arm\_cv.py:

1. **Hand-Eye Calibration**:
   * In the publish\_static\_transform function, update the translation and rotation values to match your robot's specific hand-eye calibration data.

# In publish\_static\_transform()  
t.transform.translation.x = 1.27677 # <-- YOUR VALUE  
t.transform.translation.y = 0.0175114 # <-- YOUR VALUE  
# ... and so on for all 7 values.

1. **Region of Interest (ROI)**:
   * In the \_\_init\_\_ function, adjust the self.roi rectangle [x, y, width, height] to tightly frame the chessboard in your camera's view.

# In \_\_init\_\_()  
self.roi = [100, 50, 600, 400] # [x\_start, y\_start, width, height]

## Usage

1. **Source the Workspace**:
   * In a new terminal, source your workspace's setup file.

source ~/ros2\_ws/install/setup.bash

1. **Launch the Camera**:
   * In the same terminal, start your RealSense camera node.

ros2 launch realsense2\_camera rs\_launch.py

1. **Run the Object Recognizer Node**:
   * In a second terminal (after sourcing the workspace), run the node:

ros2 run my\_robot\_cv object\_recognizer

## Published Topics

* **/object\_recognizer/detected/chessboard**:
  + **Type**: geometry\_msgs/msg/PointStamped
  + **Description**: The 3D coordinates of the chessboard's center point in the base\_link frame.
* **/object\_recognizer/detected/pieces**:
  + **Type**: geometry\_msgs/msg/PoseArray
  + **Description**: An array containing the 3D pose of every detected white and black piece in the base\_link frame.
* **/object\_recognizer/debug\_image**:
  + **Type**: sensor\_msgs/msg/Image
  + **Description**: A live video feed showing the ROI, detected objects, and their calculated world coordinates. View with rqt\_image\_view.

## Published TF Frames

* base\_link -> camera\_link (Static transform from your hard-coded calibration).
* camera\_color\_optical\_frame -> chessboard
* camera\_color\_optical\_frame -> white\_piece\_0, white\_piece\_1, ...
* camera\_color\_optical\_frame -> black\_piece\_0, black\_piece\_1, ...