Perception, Grasping, and Assembly Robot Based on Embedded Systems and Deep Learning

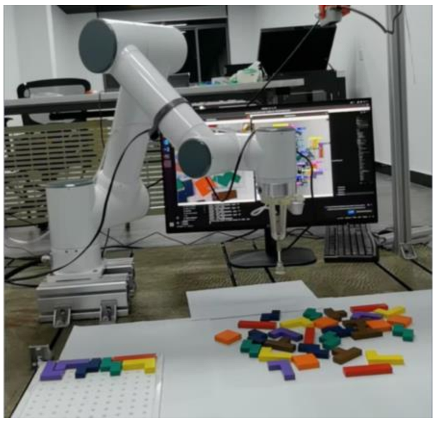
Team Members: Yuyang Qin, Lihao Pan, Chenlu Sheng
Supervisor: Shan Liu

Research Background

Robotic arms are widely used in industrial production in sectors such as automotive, electronics, food, and pharmaceuticals. They can perform repetitive, dangerous, and complex tasks, improving production efficiency and quality while reducing labor costs and risks. Currently, automation robotic arms are commonly used for tasks that are repetitive and relatively simple, such as assembly, welding, and painting. Their control systems typically involve pre-written programs or simple feedback control based on sensor input. These robots generally lack the ability to perceive and make decisions, requiring manual setup and monitoring of their workflows. The rapid development of IoT (Internet of Things) technology and artificial intelligence has provided new tools to explore and develop smarter robotic arms.

In practical applications of assembly robots, the accuracy of operation, autonomy of the assembly process, and safety of the workflow are crucial. Therefore, algorithms for target recognition, the movement of parts from their initial position to the target position, and decision-making during the assembly process are essential. Additionally, ensuring the robot can safely complete assembly tasks in a complex environment with obstacles is also important.

Professor Shan Liu’ s research project, ″Target Recognition, Grasping, and Tetris Assembly Robots,″ combined with the Intelligent Robotics Creative Competition project, proposed the task of assembling Tetris blocks. This project focuses on iteratively developing a more accurate, robust, and efficient system based on the task of assembling Tetris blocks.



The tasking view of our robotic arm

Main Work

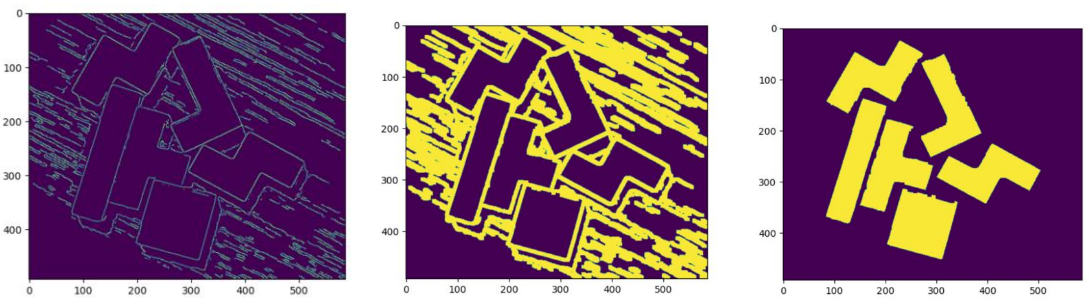
First-Generation Robotic Arm:

The first-generation Tetris robot used traditional machine vision methods for Tetris block segmentation, recognition, and positioning. The PD (Proportional-Derivative) algorithm was employed for planning the assembly scheme, and the operation was completed using the Elephant Robot Platform.

In traditional machine vision, the first-generation robot used Gaussian blur and Canny edge detection, followed by dilation to obtain multiple connected components. Based on the area consistency of Tetris blocks, the robot generated candidate connected components for recognition and segmentation. For each segmented block, the minimum rotation matrix algorithm was used to determine the tilt angle, center point, and dimensions (width and height), which were then used for classification, grasping points, and rotation angles.

For assembly planning, the robot employed an algorithm modified from Pierre Dellacherie’ s method to plan the placement of Tetris blocks. This algorithm calculated the value of each potential placement based on factors such as the number of ″gaps″ created and the number of lines that could be cleared. The robot placed the current Tetris block at the position with the highest value to implement the plan. The visual system provided information about the current Tetris block, generated the target position and rotation angle, and fed it back to the control module to complete the assembly.

For robot operation, a checkerboard was used for camera calibration. After placing the Tetris blocks, the total number of blocks was input, and the system automatically recognized the placed blocks and began assembling them. The system dynamically generated the assembly plan and supported multiple batches of block placement.



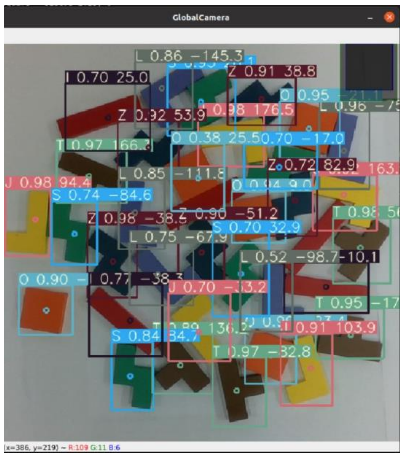
The tetris visual segmentation diagrams

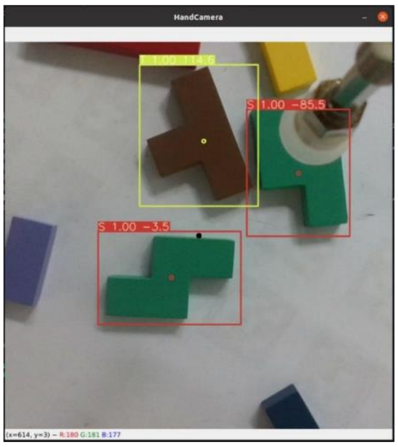
Second-Generation Robotic Arm:

The second-generation Tetris robot built on the first version by using deep learning algorithms to process inputs from the visual sensors. The YOLOv3 algorithm, which performs well in object detection tasks, was improved to also estimate the pose of the Tetris blocks. YOLOv3 is a single-stage object detection network consisting of three parts: Backbone, FPN (Feature Pyramid Network), and Yolo Head. The Backbone is made up of multiple residual blocks stacked together, with a 3x3 convolution layer (kernel_size=3x3, stride=2) between blocks for downsampling the input image. FPN further extracts and fuses features from the three valid layers received from the Backbone.

Hand-eye calibration was introduced, and a two-stage robot servo control system based on global and hand-eye cameras was employed. The first stage involved processing the images captured by the global camera using a deep network to obtain the pixel coordinates of the grasp points of the Tetris blocks. The pixel coordinates were then converted into the robot coordinate system using the camera's intrinsic and extrinsic parameters, allowing the end effector to move to the grasping position on the horizontal plane. In the second stage, a vision-based servo system was used to precisely align the end effector with the grasping center to complete the Tetris block grasping task.

For planning, the robot continued using the algorithm from the first-generation robot. The second-generation robot also introduced the advanced ROS (Robot Operating System) framework, establishing visual nodes (Vision Node), image processing nodes (ImageProcess Node), and control nodes (Robot Node). These nodes were responsible for obtaining image data from the Realsense camera, processing the data and planning the grasping scheme, and sending specific grasping commands to the robot control cabinet. This architecture enabled better real-time performance compared to the first generation.





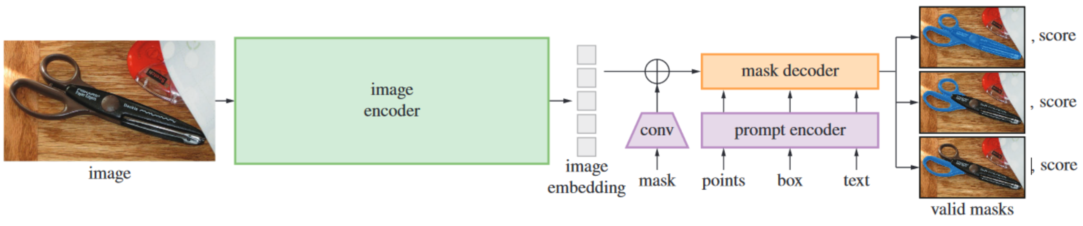
YOLO recognition results of global camera and hand-eye camera

Third-Generation Robotic Arm:
During the development of the third-generation Tetris robot, the Elephant robot was unexpectedly damaged and returned for repairs, and as of the completion of this paper, it had not yet been returned. Therefore, the third-generation solution has not been system-integrated on physical hardware and is presented here as a proposed solution. However, this solution precisely addresses the pain points and challenges of the previous two solutions and represents the focus of our efforts in the second half of the year.

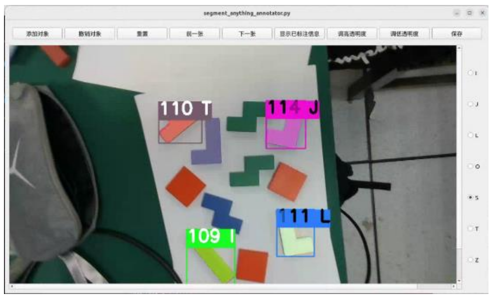
In Tetris block pose estimation and recognition, the first-generation solution required the total number of blocks to be input and lacked real-time recognition. The second-generation solution used YOLO to output the rotation angles of the Tetris blocks, but with limited accuracy. To address this, we combined both methods by using MASK R-CNN for Tetris block recognition and segmentation, followed by the minimum rotation matrix algorithm used in Solution 1 to determine the tilt angles. This approach overcame the robustness and real-time issues of traditional machine vision methods and addressed the inaccuracies in rotation angle recognition from Solution 2.

To train the MASK R-CNN model for Tetris block recognition and segmentation, we collected 100 diverse images of Tetris blocks captured from different angles and heights. To make data annotation more efficient, we used Meta AI’s Segment-anything Model, released in 2023, as an auxiliary tool. Using a PyQt5-based annotation interface, we achieved efficient, interactive data labeling. During training, we experimented with different hyperparameters and froze the ResNet50 backbone, ultimately achieving accuracy and recall rates above 98%.

Additionally, in video-based continuous monitoring, occasional Tetris blocks were missed, affecting tracking accuracy. To address this, we introduced the DeepSORT tracker to track Tetris blocks, effectively mitigating issues of frame-level misdetection or false detection.



SAM network structure diagram



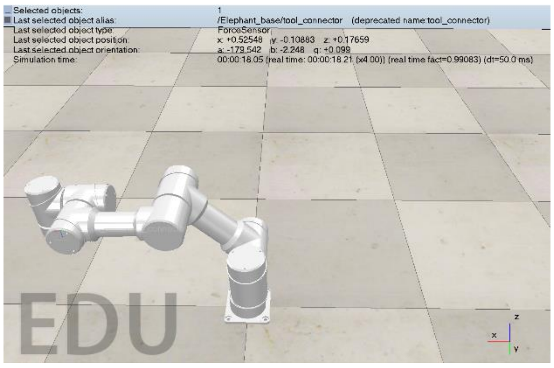
SAM schematic diagram and actual data marking interface

Robot Control Strategy:

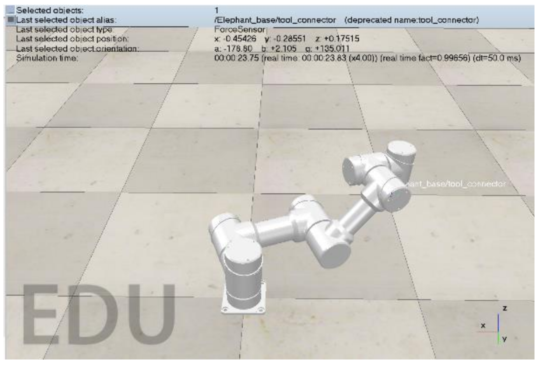
Previously, we used a linear planning method in Euclidean space, transmitting target point information to the Elephant robot control via a network interface. To avoid singularities, we planned an intermediate point. However, this control strategy did not fully utilize the joint velocities and accelerations, which hindered speed improvements. After studying "Robot Modeling and Control," we constructed the DH table for the Elephant robot, performed forward and inverse kinematics calculations, and employed the Slerp method in joint angle space for trajectory planning. This improved the control performance.

Due to the robot's malfunction, we verified the algorithm in the CoppeliaSim simulation environment, achieving satisfactory results. In terms of planning, we modified the original fixed scheme to allow the robot to select the optimal placement position based solely on the current state of the board and the type of Tetris block. We retained the value function calculation method from Pierre

Dellacherie’ s algorithm but adjusted the parameters and weights in the value function. By considering factors like the number of adjacent blocks and the number of connected blocks after placement, the optimal placement position and orientation were determined, ensuring better utilization of the board and significantly improving assembly stability and safety by minimizing restrictive conditions at the target placement positions.



```
Selected objects:                     1
Last selected object alias:           /Elephant_base/tool_connector   (deprecated name:tool_connector)
Last selected object type:            ForceSensor
Last selected object position:        x: +0.52548   y: -0.10883   z: +0.17659
Last selected object orientation:     a: -179.542   b: -2.248   g: +0.099
Simulation time:                      00:00:18.05 (real time: 00:00:18.21 (x4.00)) (real time fact=0.99083) (dt=50.0 ms)
```



```
Selected objects:                     1
Last selected object alias:           /Elephant_base/tool_connector   (deprecated name:tool_connector)
Last selected object type:            ForceSensor
Last selected object position:        x: -0.45426   y: -0.28551   z: +0.17515
Last selected object orientation:     a: -176.60   b: +2.105   g: +135.011
Simulation time:                      00:00:23.75 (real time: 00:00:23.83 (x4.00)) (real time fact=0.99656) (dt=50.0 ms)
```

Coppeliasim simulation diagram

(The two positions are grabbing and placing, and inverse kinematics uses joint angle planning)

Innovation and Challenges

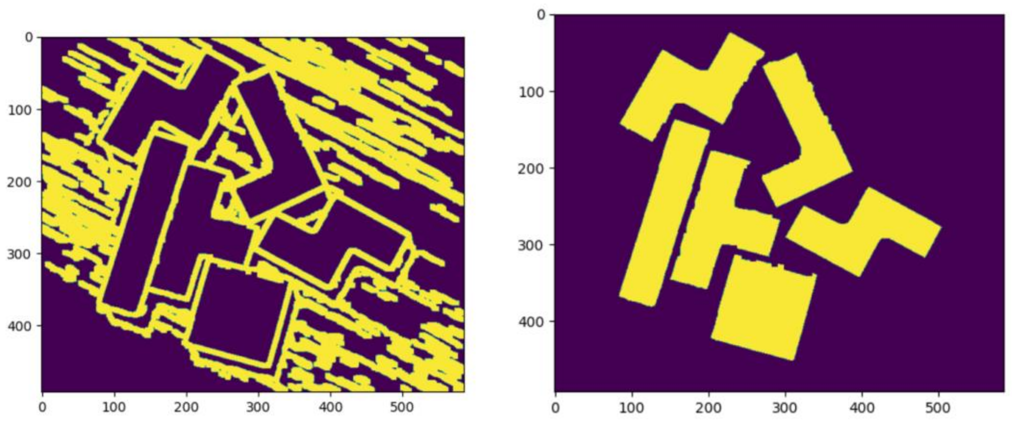
First-Generation Robotic Arm

Tetris Block Segmentation Algorithm:

Due to the uniform color of Tetris blocks and their placement on a white table (which can be easily arranged with a cloth if necessary), edge detection is particularly suitable for segmentation.

The Canny edge detector is a multi-stage edge detection algorithm. Its main steps include Gaussian filtering, gradient calculation, non-maximum suppression, and double-threshold detection and edge linking. The specific algorithm flow can be referenced in the final report. It effectively detects edges in the image while suppressing noise and false edges, yielding accurate edge information.

Although this approach is simple, it has proven to be highly effective in practice. However, it requires that there are no other objects of a similar size to the Tetris blocks within the field of view of the RGB-D camera. Additionally, after each Tetris block is placed, the number of blocks placed must be specified.



Canny edge detection diagram

Block Placement Planning Algorithm:
Tetris block placement requires the robot to output control signals to the control module for assembly, based on the current state of the board and the type of the target block. Depending on the available information, two types of placement plans can be generated:

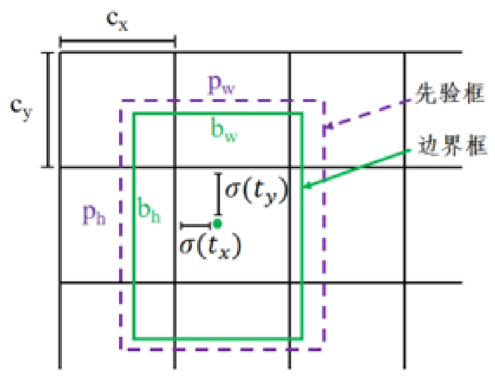
1. **Offline Generation Plan (when all block information is known):** For cases where the types and quantities of all Tetris blocks to be placed are known, an offline algorithm is used to determine the optimal placement strategy through search methods. The results are stored, and after the current block information is input during the visual processing step, the optimal placement position and orientation are output.
2. **Online Generation Plan (when only the current board state is known):** When only the current state of the board and the type of the upcoming block are known, all potential placement positions are evaluated, and the optimal position is selected. The process can be broken down into four steps: preprocessing, obtaining feasible solutions, calculating the value function, and selecting the optimal solution.

Second-Generation Robotic Arm
YOLO-Based Block Boundary Detection:
Detection of different Tetris block types and poses is achieved through a YOLO-based network. The object detection model uses multiple anchor boxes to predict the bounding boxes of objects, as illustrated in the diagram below.

To detect the bounding boxes, the following steps are used to compute the minimum bounding rectangle:

1. Compute the convex hull of the given point set using Graham scan or Andrew’ s algorithm.
2. Use the rotating calipers method to calculate neighboring edges.
3. Calculate projections of points that are parallel and perpendicular to the edges to determine the width and height of the rectangle.
4. Calculate the area and select the minimum value.

In the classification prediction training, since the target consists of 7 different shapes and colors of Tetris blocks, the model uses cross-entropy loss for category prediction training.

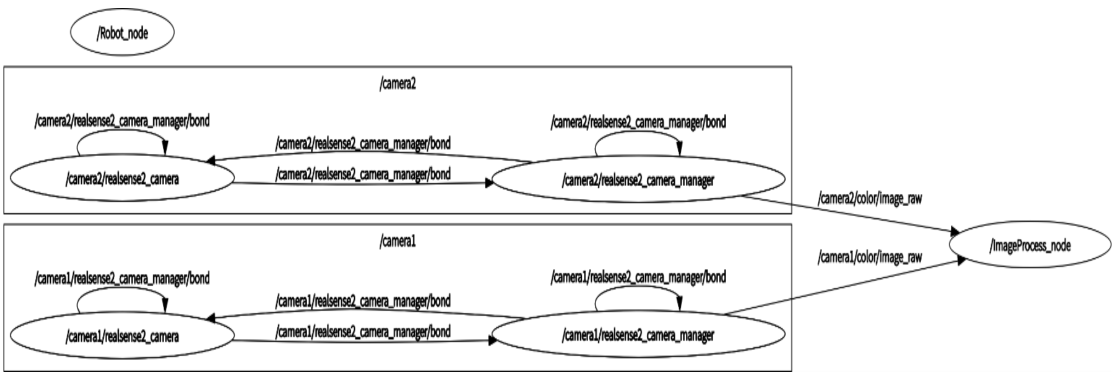


Bounding box detection diagram

Hand-Eye Camera and ROS System Architecture:
To improve the accuracy of block grasping, we introduced the hand-eye calibration algorithm and hand-eye camera. A two-stage robotic arm servo control system was employed, utilizing both a global camera and a hand-eye camera.

In the first stage, a deep network processes the images from the global camera to determine the grasping scheme and the block to be grasped, and the robotic arm moves the end effector to the target grasping position on the horizontal plane. In the second stage, a visual servoing system is used to precisely align the end effector with the grasping center, completing the block grasping task.

Additionally, the second-generation robotic arm introduced the ROS framework, establishing the following nodes for better coordination: Vision Node, ImageProcess Node, and Robot Node. This allowed for improved real-time performance compared to the first-generation robotic arm.


/Robot_node
/camera2
/camera2/realsense2_camera_manager/bond
/camera2/realsense2_camera_manager/bond
/camera2/realsense2_camera_manager/bond
/camera2/realsense2_camera_manager/bond
/camera2/realsense2_camera
/camera2/realsense2_camera_manager
/camera2/color/image_raw
/ImageProcess_node
/camera1/color/image_raw
/camera1
/camera1/realsense2_camera_manager/bond
/camera1/realsense2_camera_manager/bond
/camera1/realsense2_camera_manager/bond
/camera1/realsense2_camera_manager/bond
/camera1/realsense2_camera
/camera1/realsense2_camera_manager

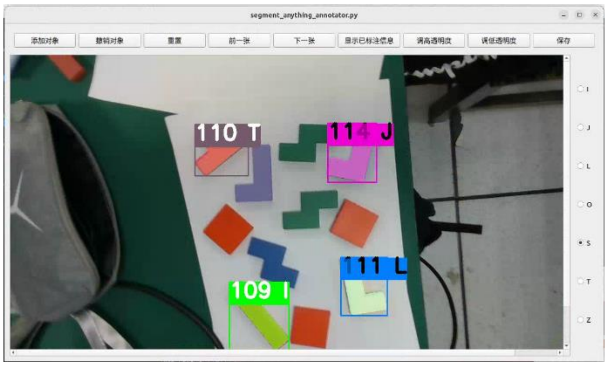
ROS communication architecture of the robotic arm

Third-Generation Robotic Arm

Data Annotation Based on Segment Anything Model (SAM):

The Segment Anything Model (SAM), developed by Meta AI, is a new artificial intelligence model capable of generating masks for any object in any image or video, even those it has never encountered during training. SAM greatly improves the speed and accuracy of data annotation (similar to the Magic Wand tool in Photoshop). Annotating segmentation data no longer requires manually tracing borders with many points (like the Pen tool in Photoshop).

For the third-generation robotic arm, we utilized SAM for dataset annotation. Although SAM offers an open-source annotation tool, it only supports single-image annotation, and switching images requires restarting the software, which is inconvenient. Therefore, we developed a more efficient data annotation platform based on PyQt5, which allowed us to easily annotate segmentation data.



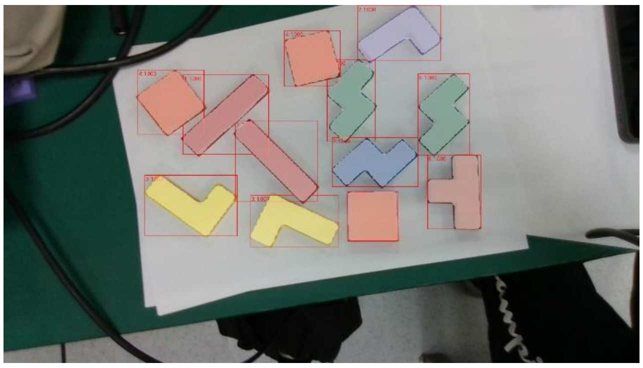
PyQt5 data annotation based on SAM

DeepSORT-Based Block Tracking Strategy:

SORT (Simple Online and Realtime Tracking) is an integrated object tracking framework that combines Faster R-CNN, Kalman filtering, and the Hungarian algorithm to assist in tracking objects through probabilistic prediction. SORT enables fast, accurate, and real-time object tracking.

To better utilize the information provided by video frames, DeepSORT integrates deep learning-based feature extraction networks to analyze the appearance features of targets. DeepSORT uses a network consisting of two convolutional layers and six residual blocks. The network is trained to effectively extract distinguishing features between pedestrians.

In this experiment, we substituted the pedestrian features in DeepSORT with the features of Tetris blocks and attempted to apply DeepSORT for Tetris block tracking, achieving promising results.

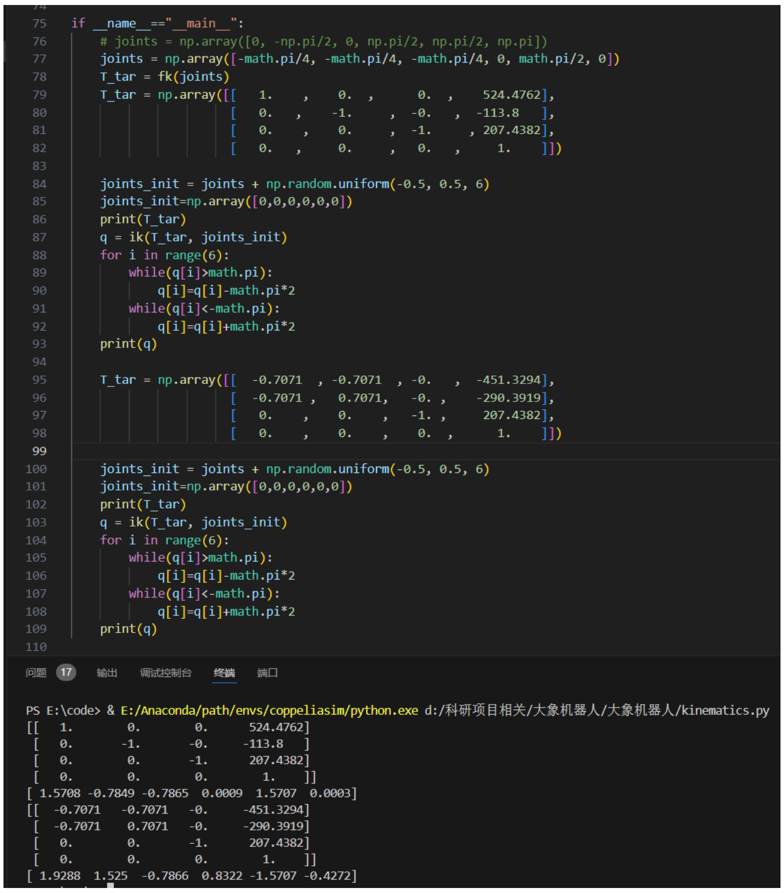


Mask R-CNN output

Robot Pose Calculation and Motion Planning:
For the first-generation robot, we issued Cartesian coordinate control commands using Python functions, primarily through the set_coords() function to move the robotic arm to the target point.

For the improved robot, joint angle planning was added for large-range movements (such as moving from the target block to the placement point after grasping). Due to the malfunction of the robotic arm, we implemented joint angle control through CoppeliaSim simulations. The robot was able to move from the target grasping point to the placement point through joint angle planning, improving the time efficiency for large rotations.

For joint angle planning, we developed an IK (Inverse Kinematics) Solver based on the Jacobian iteration method. By calling the IK Solver’ s Python file, we controlled the joint angles in CoppeliaSim. The inverse kinematics solver was validated by inputting the pose of the target grasping and placement points, calculating the joint angles at both positions, and controlling the robotic arm. The accuracy of the IK Solver was verified through simulations, accounting for the base coordinate offset (x=0, y=0, z=25 mm in CoppeliaSim).



```python
if __name__=="__main__":
    # joints = np.array([0, -np.pi/2, 0, np.pi/2, np.pi/2, np.pi])
    joints = np.array([-math.pi/4, -math.pi/4, -math.pi/4, 0, math.pi/2, 0])
    T_tar = fk(joints)
    T_tar = np.array([[   1.    ,    0.  ,      0.  ,    524.4762],
                      [   0.   ,    -1.     ,  -0.   ,  -113.8    ],
                      [   0.    ,    0.     ,  -1.     , 207.4382],
                      [   0.   ,     0.     ,   0.   ,     1.     ]])

    joints_init = joints + np.random.uniform(-0.5, 0.5, 6)
    joints_init=np.array([0,0,0,0,0,0])
    print(T_tar)
    q = ik(T_tar, joints_init)
    for i in range(6):
        while(q[i]>math.pi):
            q[i]=q[i]-math.pi*2
        while(q[i]<-math.pi):
            q[i]=q[i]+math.pi*2
    print(q)

    T_tar = np.array([[  -0.7071   , -0.7071   , -0.     ,  -451.3294],
                      [  -0.7071 ,   0.7071,   -0. ,    -290.3919],
                      [   0.     ,   0.     ,   -1. ,     207.4382],
                      [   0.     ,   0.     ,   0.  ,      1.     ]])

    joints_init = joints + np.random.uniform(-0.5, 0.5, 6)
    joints_init=np.array([0,0,0,0,0,0])
    print(T_tar)
    q = ik(T_tar, joints_init)
    for i in range(6):
        while(q[i]>math.pi):
            q[i]=q[i]-math.pi*2
        while(q[i]<-math.pi):
            q[i]=q[i]+math.pi*2
    print(q)
```

```
PS E:\code> & E:/Anaconda/path/envs/coppeliasim/python.exe d:/科研项目相关/大象机器人/大象机器人/kinematics.py
[[   1.        0.        0.      524.4762]
 [   0.       -1.       -0.     -113.8   ]
 [   0.        0.       -1.      207.4382]
 [   0.        0.        0.        1.    ]]
[ 1.5708 -0.7849 -0.7865  0.0009  1.5707  0.0003]
[[  -0.7071   -0.7071   -0.     -451.3294]
 [  -0.7071    0.7071   -0.     -290.3919]
 [   0.        0.       -1.      207.4382]
 [   0.        0.        0.        1.    ]]
[ 1.9288  1.525  -0.7866  0.8322 -1.5707 -0.4272]
```

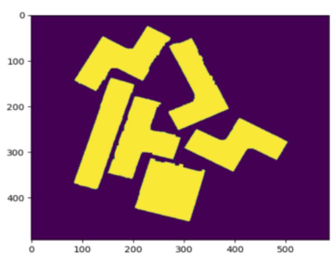
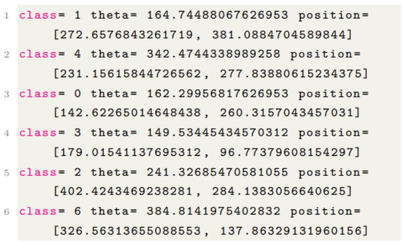
IK Solver based on Jacobi iteration

Results Showcase

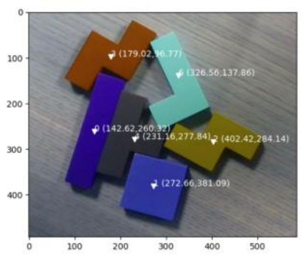
First-Generation Robotic Arm

Traditional Machine Vision-Based Tetris Block Detection:

The input image undergoes Canny edge detection followed by three rounds of dilation. The connected component area is processed to yield the recognition result, with the Tetris block segmentation shown in yellow in the image below. After applying the minimum rotation matrix and class analysis, the final result and image are generated.

```
class= 1 theta= 164.74488067626953 position=
    [272.6576843261719, 381.0884704589844]
class= 4 theta= 342.4744338989258 position=
    [231.15615844726562, 277.83880615234375]
class= 0 theta= 162.29956817626953 position=
    [142.62265014648438, 260.3157043457031]
class= 3 theta= 149.53445434570312 position=
    [179.01541137695312, 96.77379608154297]
class= 2 theta= 241.32685470581055 position=
    [402.4243469238281, 284.1383056640625]
class= 6 theta= 384.8141975402832 position=
    [326.56313655088553, 137.86329131960156]
```

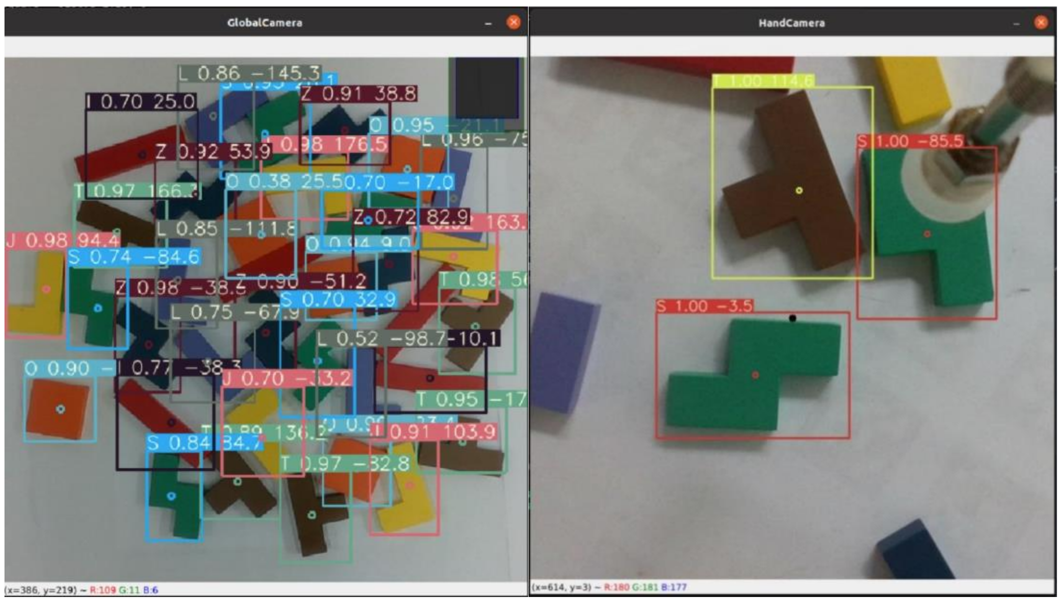


Tetris segmentation, category analysis processing results, recognition results

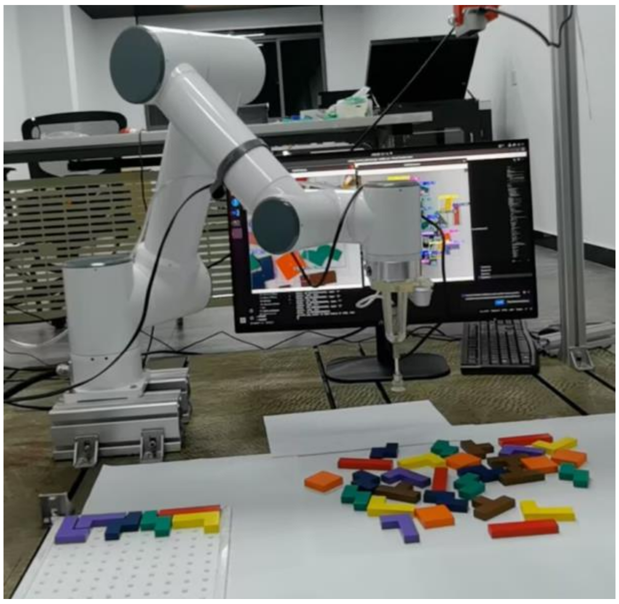
Second-Generation Robotic Arm

YOLO-Based Detection of Tetris Block Categories, Positions, and Poses:

As shown in the image below, YOLO effectively identifies the categories and poses of Tetris blocks. With the global camera's visual processing, the robot can accurately detect the type and pose of each block. The hand-eye camera allows the robotic arm to precisely stop over the Tetris block to be grasped and complete the grasping task.


GlobalCamera
HandCamera
T 1.00 114.6
S 1.00 -85.5
S 1.00 -3.5
(x=386, y=219) ~ R:109 G:11 B:6
(x=614, y=3) ~ R:180 G:181 B:177

YOLO recognition results of global camera and hand-eye camera

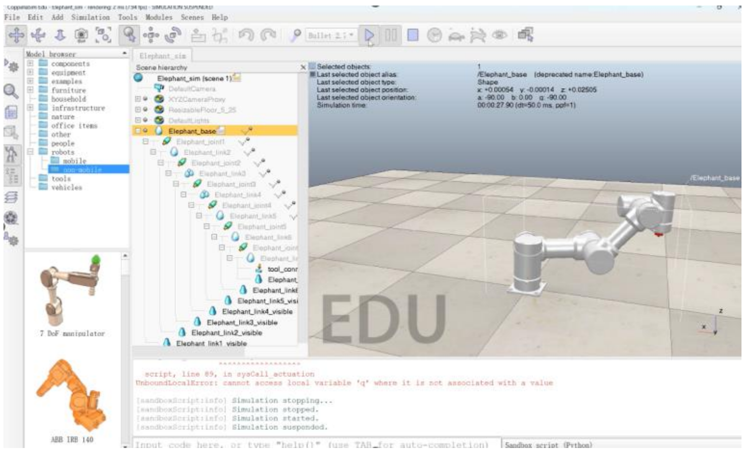


Robot arm working state diagram, including global camera and hand-eye camera

Third-Generation Robotic Arm

IK Solver Based on Jacobian Iteration:

Using SAM for visual recognition and algorithm modeling. Below is snap of the simulation in CoppeliaSim, verifying the accuracy of the IK Solver algorithm.



```
Selected objects:                      1
Last selected object alias:            /Elephant_base   (deprecated name:Elephant_base)
Last selected object type:             Shape
Last selected object position:         x: +0.00054   y: -0.00014   z: +0.02505
Last selected object orientation:      a: -90.00   b: 0.00   g: -90.00
Simulation time:                       00:00:27.90 (dt=50.0 ms, ppf=1)
```

```
      ******************
  script, line 89, in sysCall_actuation
UnboundLocalError: cannot access local variable 'q' where it is not associated with a value

[sandboxScript:info] Simulation stopping...
[sandboxScript:info] Simulation stopped.
[sandboxScript:info] Simulation started.
[sandboxScript:info] Simulation suspended.
```

The screen-shot of the simulation in the Coppliasim