Project Report

Industrial Robotics 2024:

Camera-based Pick-&-Place

**Group 2**

**Member:**

Lu Shize

Gan Liming

Chen Ke

Li Junhao

**Content**

[1. Introduction 3](#_Toc156080943)

[2. Goal 3](#_Toc156080944)

[2.1. Description of subtasks 3](#_Toc156080945)

[3. State of art 5](#_Toc156080946)

[3.1. Python and key libraries 5](#_Toc156080947)

[3.2. Denavit-Hardenberg process 7](#_Toc156080948)

[3.3. Forward Kinematics 7](#_Toc156080949)

[3.4. Inverse Kinematics 8](#_Toc156080950)

[4. Concepts and Requirement 8](#_Toc156080951)

[4.1. Concept 8](#_Toc156080952)

[4.1.1. Kinematic 8](#_Toc156080953)

[4.1.2. Image Processing 9](#_Toc156080954)

[4.2. Requirement 9](#_Toc156080955)

[4.2.1. Kinematic 9](#_Toc156080956)

[4.2.2. Image Processing 9](#_Toc156080957)

[5. Implementation 10](#_Toc156080958)

[5.1. Kinematic 10](#_Toc156080959)

[5.1.1. Inverse Kinematic 10](#_Toc156080960)

[5.1.2. Grip Process 12](#_Toc156080961)

[5.2. Image Processing 14](#_Toc156080962)

[5.3. Coordinate Transformation 16](#_Toc156080963)

[6. Tests and Verification 19](#_Toc156080964)

[6.1. Boundary Test 19](#_Toc156080965)

[6.2. Grasping Test 21](#_Toc156080966)

[7. Summary 25](#_Toc156080967)

[8. Project Responsibilities 25](#_Toc156080968)

# Introduction

The primary aim of the project is to develop a system using an industrial robot (Tinkerbot Braccio) equipped with a webcam (Logitech C270) to implement a camera-supported gripping mechanism for a Lego Duplo brick. We use several tools to complete the task:

1. Tinkerbot Braccio
2. Webcam Logitech C270
3. LEGO brick
4. Laptop
5. Cardboard to define the moving area and the location of the robot

# Goal

## Description of subtasks

In order to achieve this goal, six subtasks need to be managed, each of which contributes to the overall success of the project. Following are overview of them.

**Task 1: World Coordinate System**

Create a common world coordinate system that aligns the camera and the robot. This coordination ensures that the information obtained from the camera can be accurately translated into the robot's movements.

**Task 2: Image Processing**

Program image processing algorithms using a self-chosen Python library (e.g. OpenCV) to recognize the Lego Duplo brick. This involves analyzing images from the webcam to identify the coordinate and rotation angle of the object. Three parameters helping locate the brick are generated in this step.

**Task 3: Kinematics Setup**

Sketch the kinematic chain of robot and determine Denavit-Hardenberg parameters. Calculate the coordinate of Tool Connector Point in the world coordinate with base as origin. (Forward Kinematics). Obtain each joint angle of the robot by using parameters identified in image processing step (Inverse Kinematics). This is crucial for accurate control of the robot's movements.

**Task 4: Python Program for Gripping**

Develop a Python program to control the robot for gripping the Lego Duplo brick based on the information obtained through image processing. It contains translating the recognized object's position into robot arm movements.

**Task 5: Hand-eye Calibration**

Perform simple hand-eye calibration to fine-tune the relationship between the camera and the robot. This is important for accurate spatial mapping and coordination.

**Task 6: Test Execution and Evaluation**

Execute gripping tests using the implemented system and evaluate the success rate of the gripping mechanism. This includes testing the reliability and accuracy of the developed solution.

# State of art

To realize the target, several tools and key concepts are involved which are, including but not limited to, Python with its libraries (OpenCV, SymPy and NumPY), Denavit-Hardenberg process, Forward Kinematics and Inverse Kinematics. For these tools and concepts, we’ll give further introduction and explanation below.

## Python and key libraries

In this project, we use Python as coding language to script the program controlling robot movements, image processing and coordinate transformation. Python's popularity in robotics stems from its readability, ease of learning, and extensive libraries. Its scalability suits projects of varying sizes, while cross-platform compatibility ensures flexibility. The open-source community fosters collaboration, vital in robotics' dynamic landscape. Python's versatility enables seamless integration with other languages and systems. These features demonstrate python’s exclusive advantages. And in this program, three of python’s libraries play vital roles, which are OpenCV, SymPy and Numpy.

**OpenCV**

I**ntroduction:** OpenCV, or Open-Source Computer Vision Library, is a powerful open-source computer vision and image processing library widely employed in robotics, machine learning, and computer vision applications. Developed in C++ and optimized for real-time operations, OpenCV also provides bindings for Python, making it accessible to a broad range of developers.

**Function in the program:** It is utilized for object recognition tasks using the webcam. OpenCV provides functions and tools to capture, process, and analyze images, enabling the program to identify and locate the Lego Duplo brick. Its robust capabilities in image manipulation contribute significantly to the program's ability to interpret visual data and inform subsequent robotic actions, ensuring accurate and effective pick-and-place operations.

**SymPy**

I**ntroduction:** SymPy, a symbolic mathematics library for Python, is dedicated to performing algebraic computations symbolically. It allows developers to work with mathematical expressions in their symbolic form, making it a valuable tool for tasks involving algebraic equations, calculus, and symbolic mathematics.

**Function in the program:** It assists in handling and manipulating symbolic expressions, facilitating the calculation of joint angles and positions required for precise control of the robotic arm. SymPy's capabilities in symbolic mathematics contribute to the accurate and efficient formulation of mathematical relationships crucial for instructing the robot's movements during the gripping and pick-and-place operations.

**NumPy**

I**ntroduction:** NumPy, short for Numerical Python, is a powerful open-source library in Python designed for numerical computations. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy is a fundamental tool in scientific computing, data analysis, and machine learning applications.

**Function in the program:** It is used to perform matrix operations and mathematical calculations required for tasks like inverse kinematics. NumPy's efficiency in array manipulations ensures the swift and accurate implementation of mathematical operations, contributing to the overall success of the program in guiding the robotic system during gripping and manipulation tasks.

## Denavit-Hardenberg process

The Denavit-Hartenberg (DH) process is a systematic method for modeling the kinematics of robotic arms. It involves assigning coordinate frames to each joint, along with parameters describing the transformation between successive frames. This process simplifies the representation of complex robotic systems, aiding in the calculation of joint angles and positions.

In this project, the DH process is crucial for configuring the inverse kinematics of the Tinkerbot Braccio. By assigning coordinate frames and defining transformations through DH parameters, the program establishes a mathematical model of the robot's kinematics. This model is then used to calculate the precise joint angles necessary for the robot to grip the Lego Duplo brick accurately. The DH process streamlines the representation of the robotic arm's geometry, facilitating precise control and coordination during pick-and-place operations.

## Forward Kinematics

Forward Kinematics is a fundamental concept in robotics that involves determining the position and orientation of the end-effector (such as a gripper) based on the joint angles of a robotic system. It describes how the robot's joints contribute to the overall position and orientation of its end-effector in space.

It is implemented by following steps:

1. Establishing the DH Parameters
2. Defining the Transformation Matrices
3. Building the Homogeneous Transformation Matrix
4. Extracting Position and Orientation
5. Updating Robot's Pose

## Inverse Kinematics

Inverse Kinematics is a critical concept in robotics that involves determining the joint angles required for a robotic system to achieve a specific position and orientation of its end-effector. Unlike forward kinematics, which calculates the end-effector's pose from joint angles, inverse kinematics works in the reverse direction, calculating the joint configurations needed to achieve a desired end-effector position.

It is implemented by following steps:

1. Specifying End-Effector Position and Orientation
2. Formulating the Position Vector
3. Solving for Joint Angles
4. Validating Solutions
5. Updating Robot's Configuration

# Concepts and Requirement

## Concept

## Kinematic

**Forward Kinematic**: This concept aims to determine the position and orientation of the gripper based on the known joint angles and link lengths. It calculates how the individual joints contribute to the overall pose of the gripper in the robot's workspace.

**Inverse Kinematic**: In contrast, inverse kinematics mainly focus on achieving the default gripper position and orientation by determining the calculated joint angles. It involves solving a set of mathematical equations to find the joint configurations that result in the desired spatial coordinates.

## Image Processing

Firstly, the computer obtains the image through the camera. Next, the image is processed through Gaussian Blur, HSV conversion and erosion. Then, the color of the brick will be filtered out by setting certain color parameters. Moreover, the pixel coordinate and the angle of the brick can be acquired through camera. Finally, the pixel coordinate will be converted into the world coordinate, and pass the value to the inverse kinematic.

## Requirement

## Kinematic

In this project, there are some requirements that need to be completed to ensure the correct operation of the kinematic model.

* The position and orientation of robot arm and camera should be fixed.
* There should be enough light to ensure the correct recognition of the brick.
* The position of the brick should in the workspace of the gripper.

## Image Processing

* The camera must look vertically down to the working area.
* The height of the camera should be adjusted to fit the height of the chair in the classroom.
* The program can only detect green bricks, and it cannot detect brick of other colors.
* There should be no other things in the camera’s vision except the brick and the working background (cardboard in our group).
* The background should not be of color that is close to the color of the bricks.
* The lighting in the environment should be adjusted to eliminate the possible shadow, which may cause the identification of the bricks to have some errors.

## Implementation

## Kinematic

This part mainly aims to generate the orientations of joints according to the coordinate of the brick which obtained by the camera.

## Inverse Kinematic

In order to finish the upper task, we create a function called ‘**BraccioInverse(w)**’. In this function, we mainly did some calculation based on equations given in class. **(w)** is a array including 6 parameters. w[1,2,3] is the X,Y,Z position of the gripper and w[4,5,6] is the rotation angle around x, y, z axis. Also, we put angles of each joint together to gain a six-dimension array called **q**.

|  |  |
| --- | --- |
| 图示, 维恩图  描述已自动生成 | 图示  描述已自动生成 |
| Figure 1. q vector | Figure 2. kinematic model |

Before starting the calculation, we should measure that some constant parameters concerning the lengths between the robot joints that need to be used in the transformation.

|  |
| --- |
| 图片包含 表格  描述已自动生成 |
| Figure 3. lengths between the robot joints |

1. Firstly we calculate the q[1], the angle of rotation of the base, based on equation1.

|  |  |
| --- | --- |
| 图片包含 徽标  描述已自动生成 |  |
| Equation 1. | Figure 4. code to get q1 |

1. And then, we need to calculate q[3] in order to get q[2]. There are some prefix values (q234, b1, b2, b2) are needed to calculate q[3].

|  |
| --- |
|  |
|  |
|  |
| 画里面的卡通人物  低可信度描述已自动生成 |
| 文本  中度可信度描述已自动生成 |
| Figure 5. equations |
|  |
|  |
|  |
|  |
|  |
| Figure 6. code to calculate q[3] |

1. After that, the value of q[2], q[4], and q[5] is easy to calculate.

|  |
| --- |
|  |
| 卡通画  中度可信度描述已自动生成 |
| 图示, 示意图  描述已自动生成 |
| Figure 7. equations |
|  |
|  |
|  |
| Figure 8. code to calculate q[2], q[4], q[5] |

1. Finally, we add the offset to each value.

|  |
| --- |
| 文本  低可信度描述已自动生成 |
| Figure 9. add offset |

## Grip Process

Until now, we have already get the needed angle of each joint. Then, we need to generate the grip code. Firstly, we wrote a function called ‘**genBraccioString(q, tcpAngular, offset)**’ to generate command string and determine whether the rotation angle exceeds the range.

|  |
| --- |
| 文本  描述已自动生成 |
| Figure 10. command function |

a. ‘qtemp = q \* (180 / pi)’: Converts the joint angles from radians to degrees.

b. ‘tempCommand = "P" + str(int(qtemp[0])) + "," + ... + ",30\n”’ : Constructs a temporary command string (`tempCommand`) by formatting the joint angles and additional parameters (tcpAngular, offset). The resulting string follows a specific pattern.

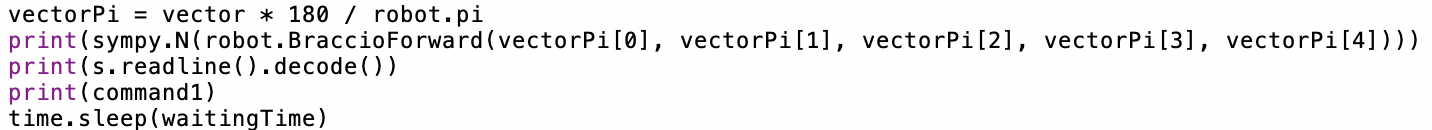
c. The code then checks if the joint angles are within a specified range (0 to 270 degrees) to ensure they are valid for the robotic arm. If any angle is outside this range, the variable ‘go’ is set to False, and the loop breaks.

d. ‘if go:’: If the joint angles are within the valid range, the function sets the final command string to the temporary command string (‘tempCommand’).

e. ‘else:’: If the joint angles are not within the valid range, the function prints "not in range" and returns -1 to indicate an error.

Then, we need to generate some commands to control the gripper to reach position of the brick.

|  |
| --- |
| 文本  描述已自动生成 |
| Figure 11. command1 |



|  |
| --- |
| Figure 12. code to reach the position |

And then we generate command2, 3, 4, and 5 to let the gripper close, lift, go to another place and put down the brick.

|  |
| --- |
| 文本  中度可信度描述已自动生成 |
| Figure 13. command2,3,4,5 |

## Image Processing

The image processing will be separate into four parts, including getting frame, frame filtering and color extraction, brick identification and pixel coordinate calculation, and center plot:



Figure 14. getting frame

**Cap** is a video capture object for the camera. The following code will run only if the video capture object is successfully opened. Frame is the image read from the camera, and ret will be true if the frame is successfully read.

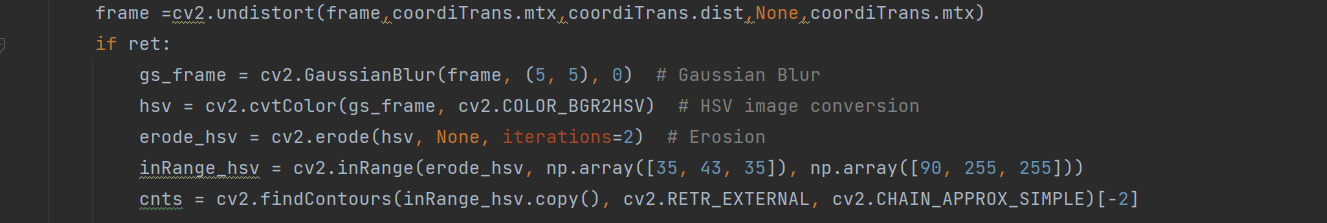


Figure 15. frame filtering and color extraction

The frame will then firstly go through the process to eliminate distortion. **coordiTrans.mtx** is the camera internal reference matrix. **coordiTrans.dist** is the aberration coefficient matrix. Since we do not need the output camera matrix, so it is set to None. Secondly, the undistorted frame is applied the Gaussian blur to reduce the noise and smoothen the image. The width and height of the filter is set to (5,5) and the standard deviation is set to 0. Thirdly, the blurred frame from the BGR color space is converted to the HSV color space. Fourthly, erosion is performed on the HSV image to further remove small unwanted details or noise. The number of times erosion is set to 2. Then, a binary mask is added to select desired color by thresholding the HSV values within a specific range. In this project, the value is set to meet the range of color green. Finally, contours (**cnts**) are found in the binary mask. **cv2.RETR\_EXTERNAL** make sure that it only retrieves the external contour. **cv2.CHAIN\_APPROX\_SIMPLE** is the contour approximation method we applied. It compresses horizontal, diagonal, and vertical segments and leaves only the end points in the contour.

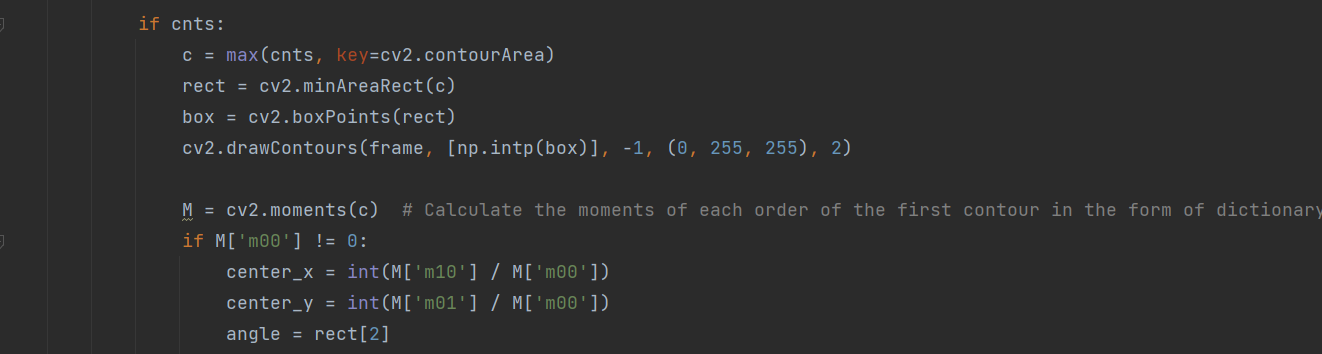


Figure 16. brick identification and pixel coordinate calculation

After extracting the desired shape and color from the frame, the pixel coordinate is needed to determine the world coordinate of the object. The region with the largest area is found in the contour. This region is selected as the region where the square is located. The smallest outer rectangle of the region is then drawn and recorded. **Box** stores the coordinates of the four points of the rectangle. The rectangular contour is drawn on the original image through **cv2.drawContours()**.

If the pixel coordinate of the brick is needed, the moments of the contour need to be calculated. Moments are statistical measures of the distribution of pixel intensities in an image. The zeroth order moment is checked to ensure that the value is not equal to zero. The zeroth order moment represents the total area of the contour. This check is important to avoid division by zero. Then, the x-coordinate of the centroid of the contour is calculated. Its value is the average position of all the pixels in the contour along the x-axis. The y-coordinate of the centroid of the contour is calculated in a similar way. Finally, the rotation angle is extracted from the **rect** variable.

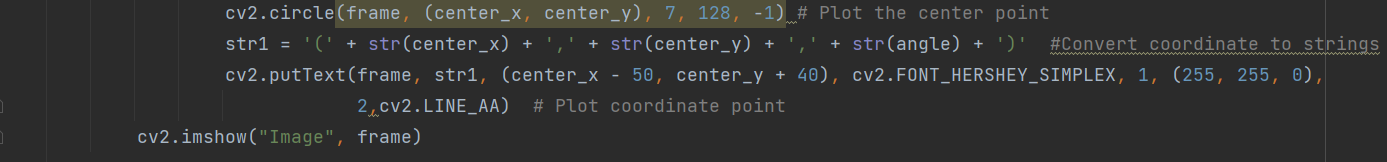


Figure 17. center plot

The last step in image processing is to show the coordinate of the center of the brick on the frame. It makes the testing phase more convenient and efficient.

## Coordinate Transformation

To transform the pixel position we get at the image processing to the position in the world coordinate, we generate two matrices and multiply it left by the pixel position. The equation is given by:

Where T is the Camera Pose Matrix, K is the Intrinsic Matrix of camera and Z is the depth of object in camera.

First, we take multiple (15-25) photos of the calibration board with the camera and use them to calculate the intrinsic and distortion parameters in MATLAB.

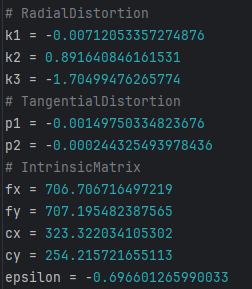
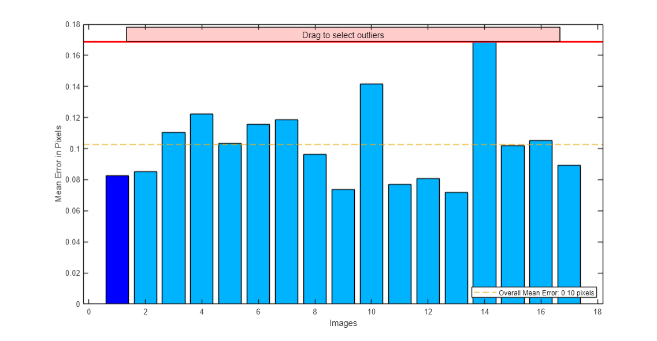
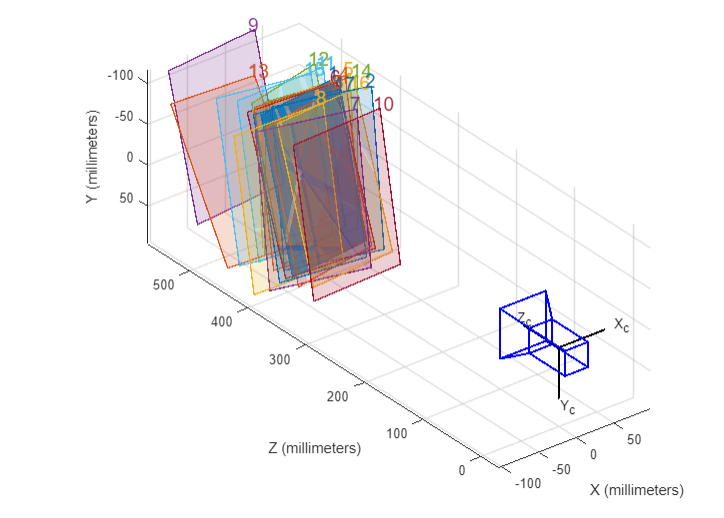
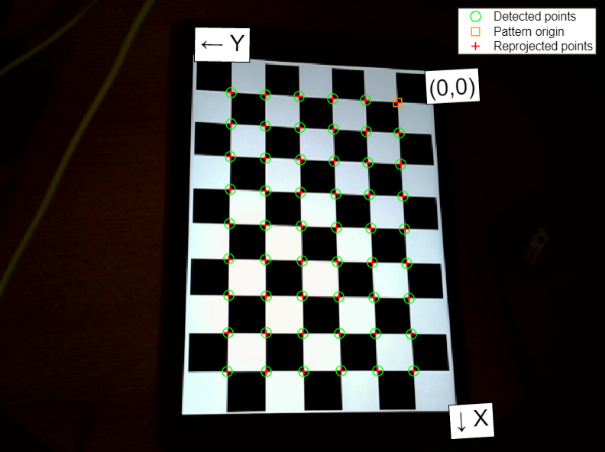


Figure 18.

By using the camera calibration tool in MATLAB, we get the intrinsic matrix and five distortion parameters with 0.1 pixels of overall mean error like the pictures showed above.

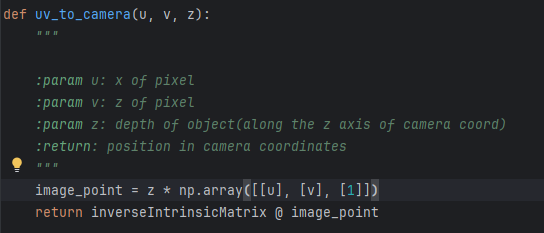


Figure 19.

In the **uv\_to\_camera(u,v,z)** function, we generate the position of the object in the camera coordinate system. During this process, we introduce three parameters, where the first one is the horizontal coordinate of the pixel, the second one is the vertical coordinate of the pixel and the third one is the depth of the object.

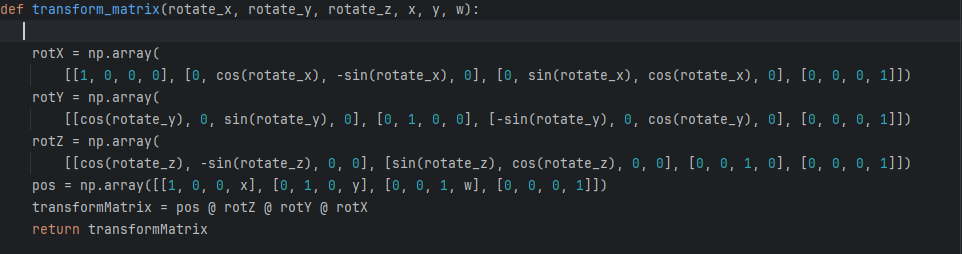


Figure 20.

In the **transform\_matrix(rotate\_x, rotate\_y, rotate\_z, x, y, w)** function, we generate the Camera Pose Matrix T by multiplying three rotation and one transform matrices. We measure the angle of rotation of the camera’s coordinate system around each axis in the world coordinate system by order and the position of the origin point of camera’s coordinate system in the world coordinate system. By entering these values, we get the rotation matrices and transform matrix. Finally, we return the camera pose matrix in this method.

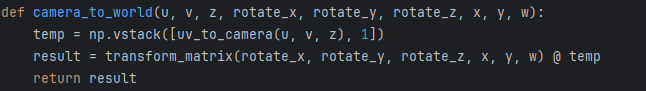


Figure 21.

In the **camera\_to\_world(u,v,z,rotate\_x,rotate\_y,rotate\_z,x,y,w)** function, we integrate the above processes for the actual coordinate transformation in operating robot.

The main difficulty in the whole transformation process is that the measurement value of the camera coordinate system is related to the error directly, which means we must make them accurate as much as possible. So, we need to test many times to acquire the most appropriate values or add some offsets to these values.

# Tests and Verification

## Boundary Test

The boundary test is used to measure the workspace of the robot and to ensure that the robot do not reach out of the workspace. Based on our layout, we conclude that the workspace of our robot shapes like Figure 22

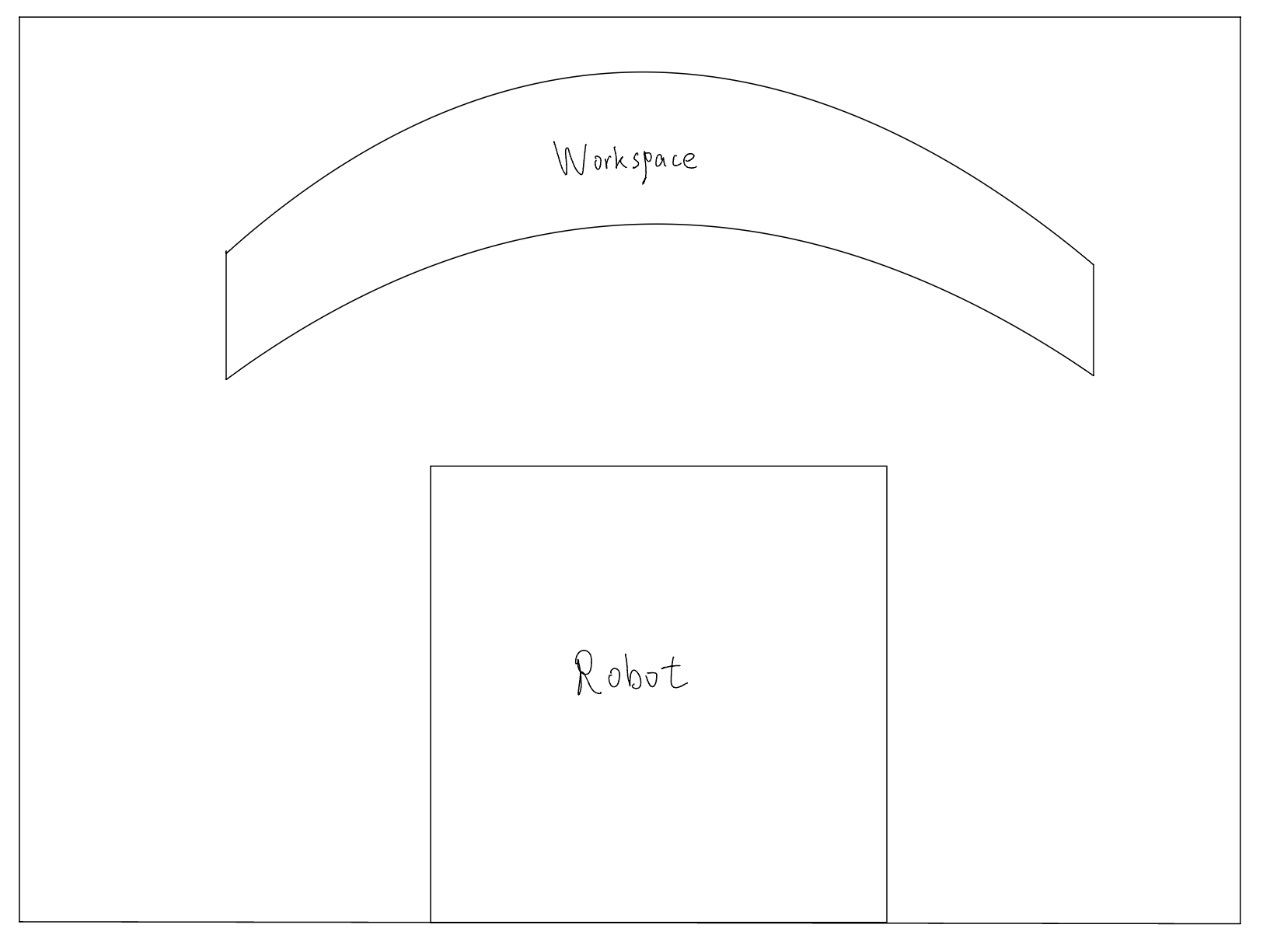


Figure 22. workspace

Theoretically, the workspace is constituted by two aspects: the vision area of the camera and the area that the camera can reach out. Since the vision area of the camera is much larger than the area that the camera can reach out, so we neglect the influence of the vision area of the camera and only use it as the starting point of measuring the real workspace. We firstly use the code to scan the approximate range of the robot. The code is shown below:

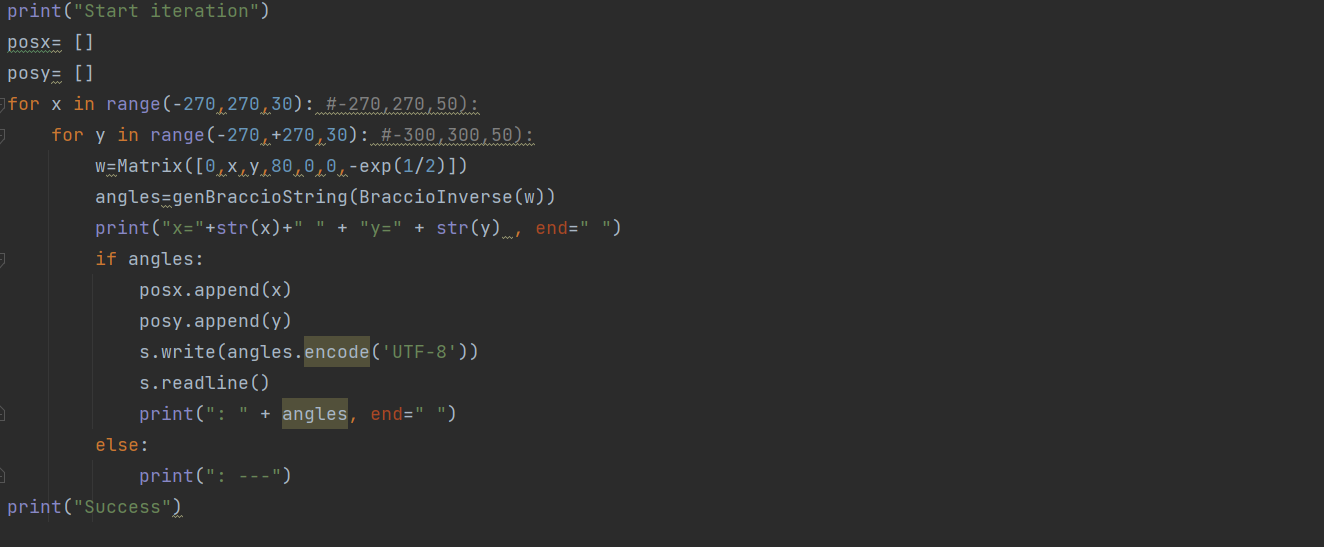


Figure 23. scanning the approximate range of the robot

We then determine 6 exact boundary points including left-upper, left-lower, right-upper, right-lower, center-upper and center-lower points as Figure 24 shows.



Figure 24. boundary points

Finally, we test multiple points and fit a curve to those points as Figure 25 shows.

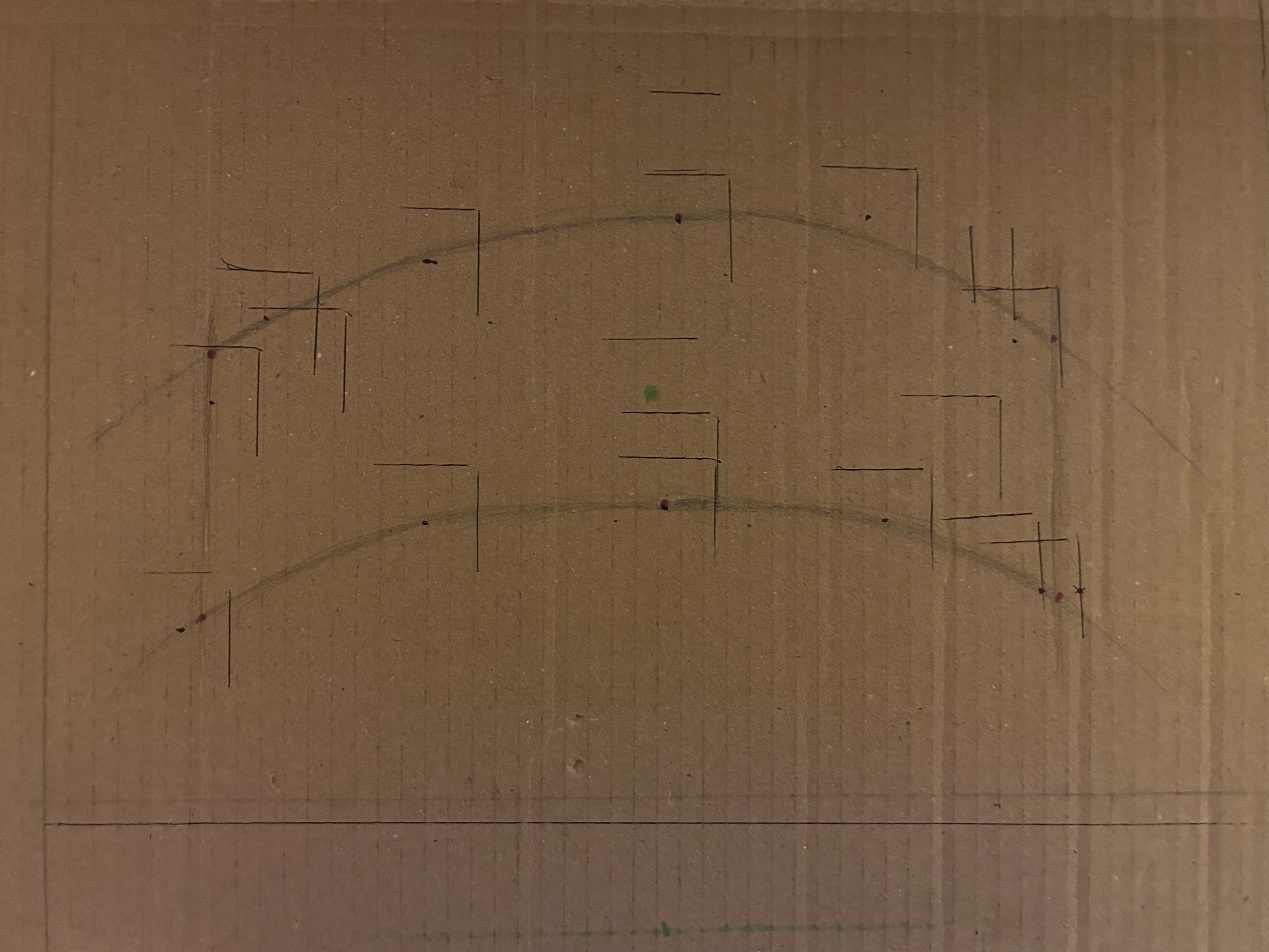


Figure 25. final range

## Grasping Test

We tested the success rate of the gripper inside the workspace area. We select 5 locations, which includes the center area and the boundary area. Layout shows as Figure 26. The green points are the test points.

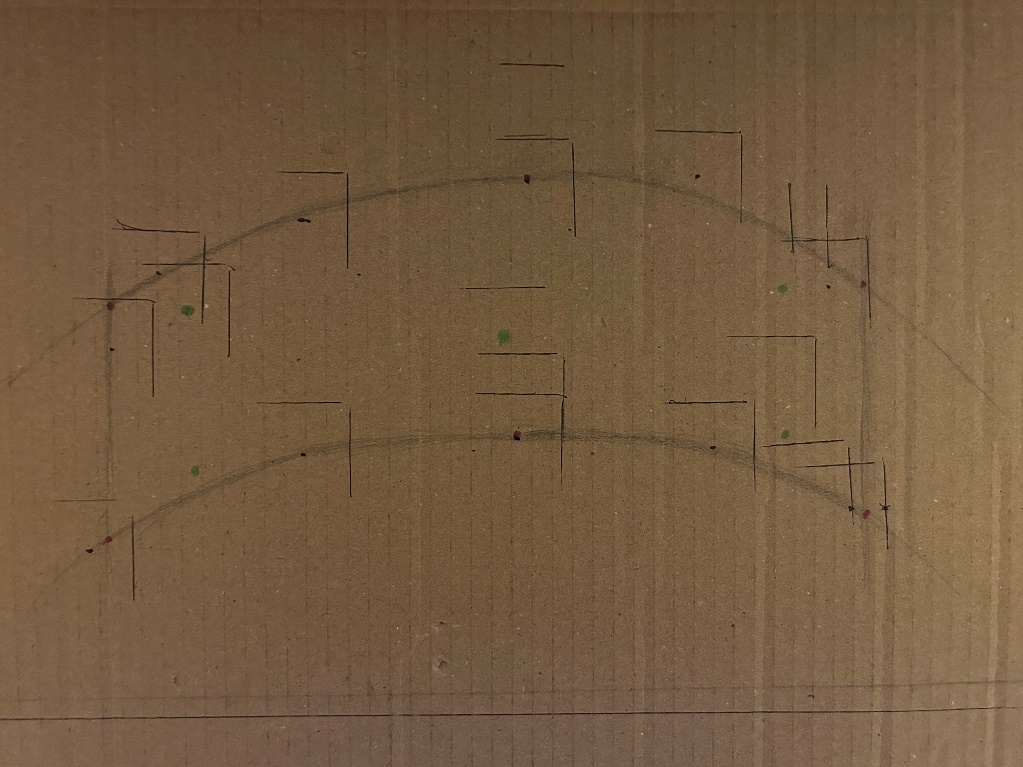


Figure 26. testing points

The test is done by several steps. Firstly, robot and camera are set in their predefined position. Secondly, the camera needs to be adjusted to fit in the vision which is also predefined.

After the program starts to run, the camera will detect the brick. By pressing “Enter”, the robot will start to grab the brick.

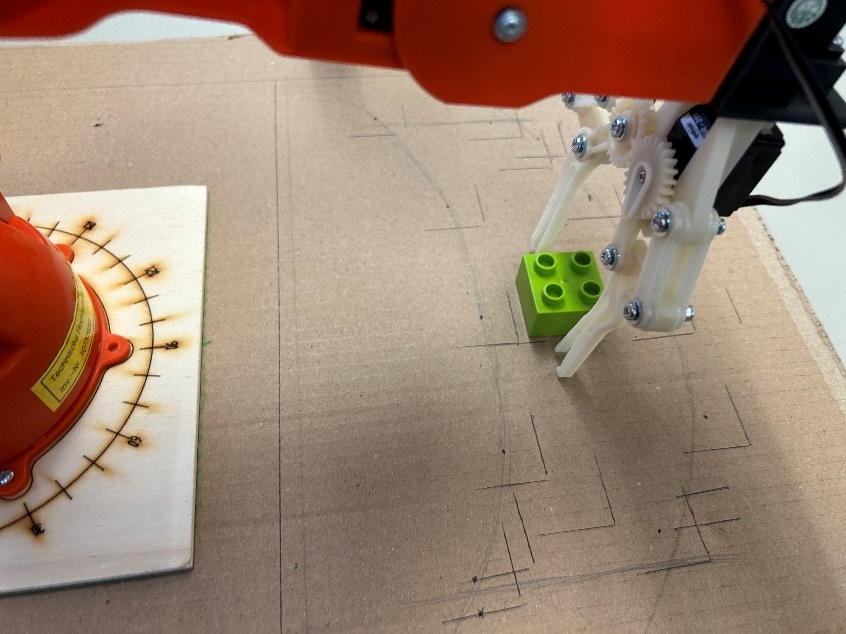


Figure 27. grabbing the brick

After grabbing the brick, the robot will pick it up.



Figure 28. picking brick up

Finally, it will place the brick in predefined location and drop it.

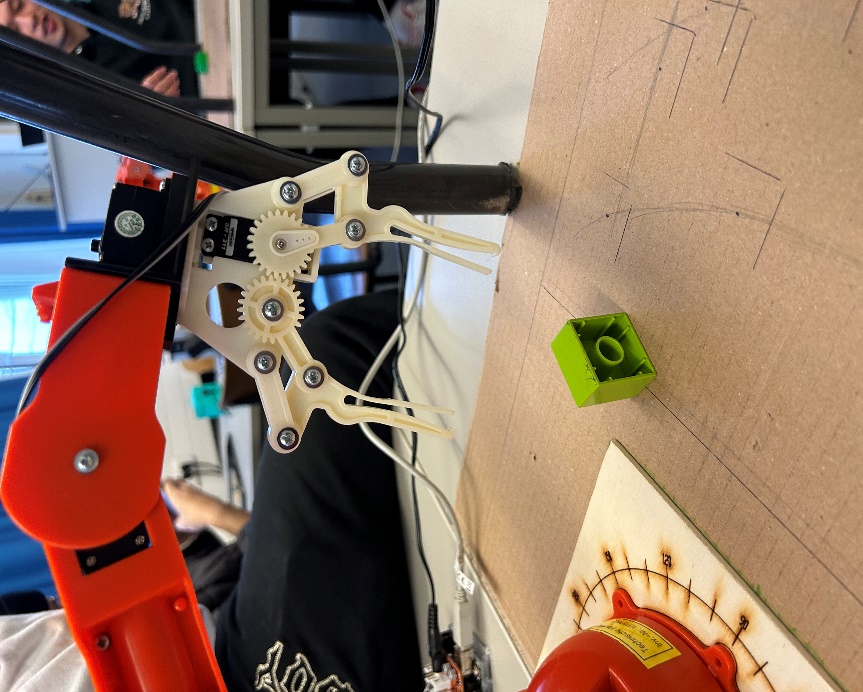


Figure 29. dropping brick

The test result shows as follows:

(1)

|  |  |
| --- | --- |
| Coordinates of the cube | (54,248) |
| Angle of the cube (in degree) | 2.2 |
| Whether the gripper successfully moves directly above the cube | yes |
| Whether the grasp is successfully completed | yes |

(2)

|  |  |
| --- | --- |
| Coordinates of the cube | (11,261) |
| Angle of the cube (in degree) | 53.6 |
| Whether the gripper successfully moves directly above the cube | yes |
| Whether the grasp is successfully completed | no |

(3)

|  |  |
| --- | --- |
| Coordinates of the cube | (-72,266) |
| Angle of the cube (in degree) | -24 |
| Whether the gripper successfully moves directly above the cube | yes |
| Whether the grasp is successfully completed | yes |

(4)

|  |  |
| --- | --- |
| Coordinates of the cube | (-120,240) |
| Angle of the cube (in degree) | 45 |
| Whether the gripper successfully moves directly above the cube | yes |
| Whether the grasp is successfully completed | yes |

(5)

|  |  |
| --- | --- |
| Coordinates of the cube | (-9,285) |
| Angle of the cube (in degree) | 35 |
| Whether the gripper successfully moves directly above the cube | yes |
| Whether the grasp is successfully completed | no |

# Summary

In this project, the team members corporate with each other and finish the main task successfully. Although there are still some drawbacks, such as the inaccurate positioning of the robot arm under some situations, the basic aim of recognizing the brick, grabbing the brick and placing it is obtained. We hope to update our computer vision algorithm in the future to make it more adaptable to fierce environment. Moreover, detecting objects in three dimensions from different view can also help to more accurately determine the location of the brick and increase the success rate of grabbing the brick. We believe tat this project has a promising future.

# Project Responsibilities

* **Lu Shize**

1. Develop the code for object recognition and pixel coordinate.
2. Write the object recognition, pixel coordinate calculation and test part in report.
3. Conduct test to optimize the system.

* **Gan Liming**

1. Develop the code and measure camera parameters for coordinate system transformation.
2. Write the coordinate system transformation part in report.
3. Conduct test to optimize the system.

* **Chen Ke**

1. Develop the code for inverse kinematic and forward kinematic.
2. Write the inverse kinematic and forward kinematic part in report.
3. Conduct test to optimize the system.

* **Li Junhao**

1. Collect and prepare the background information for the project.
2. Write the introduction, goal and state of art part in report.
3. Collect and analyze the test result.

All members contribute to the project, ensuring a collaborative team work.