Course design for Automation Systems Robotics: Camera-based Pick-&-Place

Teacher:

Prof. Dr.-Ing. Thomas Wich

Group 4

Team member

(Alphabetical order by initials):

Chen yuan

Jin bingyi

Pu xinchen

Zhao wenjie

Date:01.2022

Context

1. Introduction

1.1. Background

At present, robots have been widely used in industrial production. Compared with industrial deterministic environments, the research on the application of mobile robot robotic arms in uncertain environments is not yet mature. Due to the multidisciplinary fields involving vision processing, real-time control, and robot arm kinematics, the study of robot arm control based on machine vision is still one of the difficult areas in mobile robotics research.

Germany which has a major historical industrialization level has always been in the leading position in the world in robot technology after decades of exploration and research. A variety of home robots((Figure1.1.1), equipped with stereoscopic color cameras, laser scanners, etc., can use image sensors to identify daily necessities and complete autonomous grabbing and delivery services.

Figure 1.1. 1 source: https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other

As for the image sense, there are two main types which are charge-coupled device (CCD) and the active-pixel sensor (CMOS sensor). Nowadays, the most common use of the camera is the binocular camera because the traditional monocular camera can only get two-dimensional information about the environment. The works applied nowadays are so complex that we can only achieve by using binocular cameras simultaneously to collect environmental information through two parallel cameras, and then obtains the depth information of the environment through image processing technology and three-dimensional geometric principles.

When it comes to camera calibration methods, there are traditional camera calibration, computer self-calibration, and active vision camera calibration methods. The simplest way is to use the traditional method which calibrates by using the existing subject and camera. But it depends on the accuracy of the equipment significantly.

As a result, we want to implement the simplest application of these techniques which is to use the robot to implement a method for camera-supported gripping of a Lego Duplo brick. It is the most common situations to meet with the mobile robot.

A pick-and-place robot is a type of industrial robot that is used for handling and placing products on a production line. They are typically used in high-volume manufacturing settings, where they can quickly and accurately place products onto conveyor belts or other production equipment. The most common types of such robot are Gantry robots, Articulated robots, SCARA robots, Delta robots and so on. They have different advantages and disadvantages. For example, the Delta robots have great flexibility that they can be applied in many packaging situations. A basic robot is shown below.

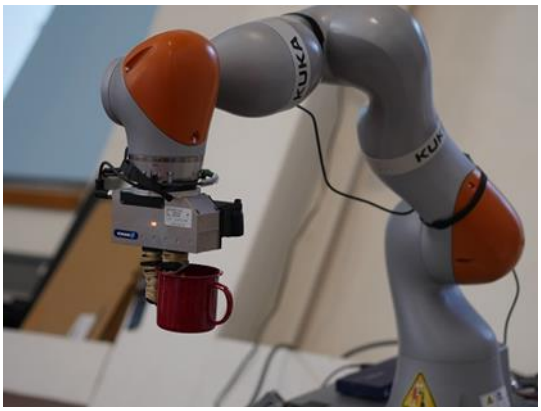

Figure 1.1. 2 source: https://electronics360.globalspec.com/images/assets/757/12757/DON_system_and_Kuka_robot_grasp_a_cup.JPG

1.2. organization of the subject

According to the research process, the chapters of this paper are arranged as follows:

Chapter 1 introduces the background of the subject and the structure of this paper.

Chapter 2 introduces the research content and the structure of the project for the implementation object.

Chapter 3 introduces the key technologies of the mobile robot arm control system based on machine vision, and provides the theoretical basis for the experimental sessions.

Chapter 4 outlines the basic preparations through the concept and requirements of this topic.

Chapter 5 establishes the description of the robotic arm joint coordinate system according to the robotic arm configuration using the D-H representation and carries out the robotic arm motion planning in the joint space; develops the vision-based robotic arm grasping object strategy and verifies through experiments whether the robotic arm meets the demand for fast and accurate semi-autonomous grasping of target objects.

Chapter 6 performs field tests on the target, debugging, analysis and improvement of each step.

Chapter 7 summarizes the work accomplished in this paper and provides an outlook on future directions.

2. Objection

2.1. Main research content

According to the demand for accurate grasping of objects by mobile robotic arm, the group has studied the camera calibration, visual ranging, target detection and tracking, robotic arm motion planning and other contents involved in the control system, and designed and implemented a mobile robot robotic arm control system based on machine vision. The main research contents of this paper are as follows:

- Camera calibration: Due to the camera production process and other trapped factors camera is usually a non-ideal imaging device, there are distortions and other interference in the imaging process, the error improvement by grasping the prism length of the block.
- Monocular ranging: According to the experimental conditions of this paper, the monocular ranging algorithm is selected to obtain the three-dimensional spatial coordinates of the target, and the ground ranging model is established using the camera imaging model, and its feasibility is verified using experiments.
- Target detection and tracking: In view of the specific color characteristics of the robot arm, this paper uses the color histogram as the target image feature in target tracking, and also applies the particle filtering algorithm to target tracking, and analyzes its working principle in detail.
- Robotic arm motion planning: According to the robotic arm configuration studied in this paper, a six-joint robotic arm joint coordinate system description is established using the D-H representation, and forward and inverse kinematic analysis is performed, based on which the motion planning of the robotic arm joint space is realized, semi-autonomous grasping based on machine vision: On the basis of monocular ranging and target detection, combined with robotic arm motion planning, the semi-autonomous grasping of regular objects is realized.

In this paper, we use Tinkerbot Braccio, webcam (Logitech C270) and Lego Duplo brick to fulfill the simple implementation. we use a monocular camera as the vision sensor for the robot arm, and study the problems of visual ranging, target detection and tracking, and motion

planning involved in the semi-autonomous grasping of the robot arm, and realize the robot arm control system of the mobile robot based on monocular vision. The other two methods are often used in some special cases which requires the track of movements for the camera.

2.2. Project implementation process

Our group firstly designs the overall scheme of the robotic arm control system, then calibrates the camera based on the camera imaging model using the traditional calibration method of the camera; verifies the feasibility of monocular vision ranging through experiments; realizes fast and accurate target tracking using particle filtering algorithm; describes the coordinate relationship and solves the forward and reverse kinematics according to the robotic arm configuration using the D-H representation, and completes the robotic arm motion planning in the joint space. The robot arm motion planning is done in the joint space.

The group members communicate through Gitlab, program in python, and write robotic arm motion commands. Finally, the target detection and tracking and object grasping experiments were completed, and the grasping experiment results showed that the control system could grasp the target object quickly and accurately semi-autonomously.

3. State of the art

3.1. Image recognition

Machine vision is the use of machines instead of the human eye to make measurements and judgments. Machine vision system refers to the machine vision products (i.e., image ingestion device, divided into CMOS and CCD two kinds) will be ingested target into image signals, transmitted to the special image processing system, according to the pixel distribution and brightness, color and other information, into digital signals; image system to these signals for various operations to extract the characteristics of the target, and then according to the results of the discriminatory to control the field equipment Action.

Image recognition has gone through three main stages: recognition of textual information, recognition of digitized information, and recognition of objects. After these three stages of continuous development, image recognition gives full play to its own characteristics and advantages, and gradually expands to various fields and combines with various industry technologies. The main applications and development directions of image recognition technology are character recognition, machine vision recognition, graphic image recognition, biomedicine, etc.

3.2. Robotic arm vision servo system

The principle of machine vision-based robotic arm control is to control the motion of the robotic arm based on the two-dimensional image information of the environment acquired by the camera, and different ways of using the camera correspond to different control system structures and control algorithms.

According to the installation position of the camera on the robot, the robotic arm control system can be divided into eye-in-hand and eye-to-hand systems. Eye on the hand that the camera is mounted on the end of the robot arm actuator; eye on the hand that the camera is mounted in a position with a good viewing angle relative to the end of the robot arm actuator. According to the type of feedback information of the robotic arm control system, that is, the choice of day marker feature type can be divided into position-based visual servo system (PBVS, also known as 3D visual servo system), image-based visual servo system (IBVS, also known as 2D visual servo) and hybrid visual servo system

(HVS, also known as 2.5D visual servo). The basic model of the machine vision-based robotic arm control system is the error function between the current target eigenvalue and the desired target eigenvalue.

The position-based vision control system takes the difference between the current pose information of the end-effector or target and the desired pose information obtained from the camera image as the control quantity, thus constituting a position-based robotic arm control system. The input variables and error-driven variables of the controller of the position-based robotic arm control system are described in a right-angle coordinate system.

The position-based vision control system uses 3D positional parameters as features of the target, and the system structure is shown in Figure 4-a-1.

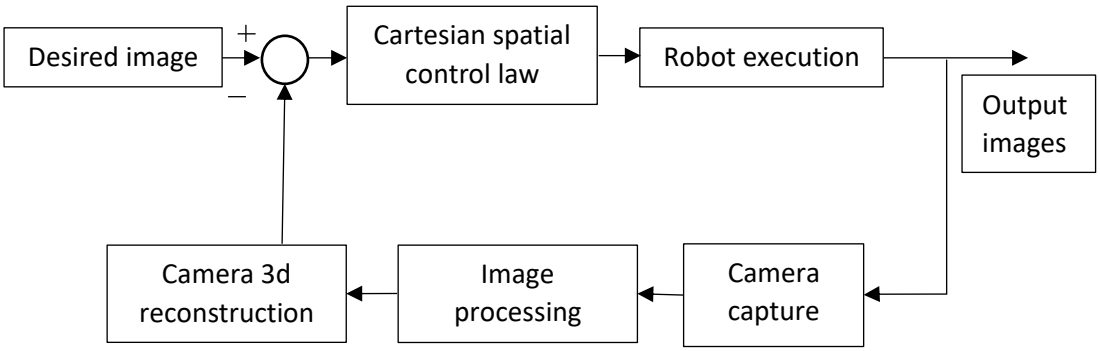

Figure 3.2. 1 position-based vision control system

The image feature-based robotic arm control system uses two-dimensional image information as the target object feature, and its control system structure is shown in Fig 4-a-2.

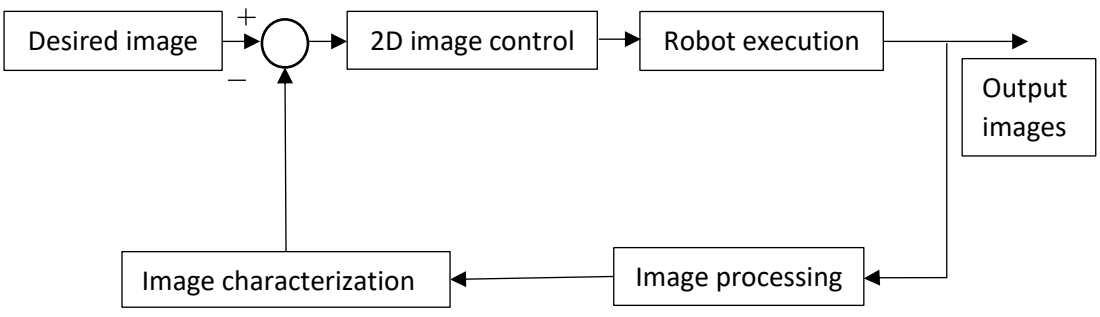

Figure 3.2. 2 feature-based robotic arm control system

The image-based robotic arm control system compares the image feature values acquired by the camera with the expected image feature values, and maps the image feature differences to the robotic arm joint space control quantities to form a closed-loop image-based feedback control system. A key problem in the image-based robotic arm control system is the calculation of the image Jacobian matrix, i.e., the relationship matrix between the variation of the robotic arm end-effector in Cartesian space and the variation of the image features in image space. By inverting the Jacobian matrix, the control law of the robotic arm control system can be obtained, and then the position of the robotic arm end-effector can be controlled to obtain the expected image feature information.

3.3. Corner detection and examples

Two algorithms for corner detection, Harris corner feature extraction and FAST corner feature extraction, are introduced below.

Harris corner point detection is a first-order derivative matrix detection method based on image grayscale. The main idea of the detector is local self-similarity/autocorrelation, i.e., the similarity of an image block within a certain local window to an image block within a window after a tiny movement in each direction. Within the neighborhood of pixel points, the derivative matrix describes the variation of the data signal. It is assumed that if a block region is moved in any direction within the neighborhood of a pixel point, and if the intensity changes drastically, the pixel point at the change is a corner point. The defined Harris matrix is as follows,

$$M(x,y)=\sum_w \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix} = \begin{bmatrix} \sum_w I_x(x,y)^2 & \sum_w I_x(x,y)I_y(x,y) \\ \sum_w I_x(x,y)I_y(x,y) & \sum_w I_y(x,y)^2 \end{bmatrix} = \begin{bmatrix} A & C \\ C & B \end{bmatrix}$$

Figure 3.3. 1 source: https://blog.csdn.net/weixin_41033536/article/details/82970511

The corner response value R of the Harris matrix is calculated to determine whether it is a corner point or not. Its calculation formula is,

$$R = detM - \alpha(traceM)^2$$

Figure 3.3. 2 source: https://blog.csdn.net/weixin_41033536/article/details/82970511

where det and trace are operators of determinant and trace, and are constants taking values from 0.04 to 0.06. When the corner point

response value is greater than the set threshold and is a local maximum in the neighborhood of the point, the point is treated as a corner point.

The FAST algorithm based on accelerated segmentation test can extract corner point features quickly. The algorithm determines whether a candidate point is a corner point based on the fact that if there is a consecutive pixel on the circumference of a discrete Bresenham circle with a pixel point as the center and a radius of 3 pixels, under the condition of a given threshold, if the gray value of the pixel on the circumference is greater than or less than. For the above definition, we can do the detection in a fast way without comparing all points on the circumference. First, we compare the pixel value relationships of the top, bottom, left and right four points, and at least three points with pixel gray value greater than or less than are candidates, and then we can further make a complete judgment.

4. Concept and requirements of this work

4.1. HSV color recognition

HSL and HSV are the two most common color models represented by cylindrical coordinates, which re-imagine the RGB model and thus can be visually more intuitive than the RGB model.

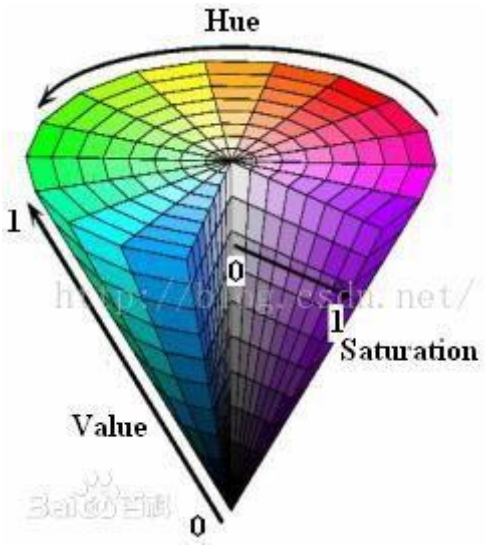

Figure 4.1. 1 source: https://blog.csdn.net/kakiebu/article/details/79476235

Generally, the effective processing of images in color space is performed in HSV space, and then a strict range needs to be given for the corresponding HSV components in the fundamental colors, and the following is the blurring range.

Color	R	G	B	H	H_2	C	C_2	V	L	I	Y'_{601}	S_{HSV}	S_{HSL}	S_{HSI}
	1.000	1.000	1.000	n/a	n/a	0.000	0.000	1.000	1.000	1.000	1.000	0.000	0.000	0.000
	0.500	0.500	0.500	n/a	n/a	0.000	0.000	0.500	0.500	0.500	0.500	0.000	0.000	0.000
	0.000	0.000	0.000	n/a	n/a	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	0.000	0.000	0.0°	0.0°	1.000	1.000	1.000	0.500	0.333	0.299	1.000	1.000	1.000
	0.750	0.750	0.000	60.0°	60.0°	0.750	0.750	0.750	0.375	0.500	0.664	1.000	1.000	1.000
	0.000	0.500	0.000	120.0°	120.0°	0.500	0.500	0.500	0.250	0.167	0.293	1.000	1.000	1.000
	0.500	1.000	1.000	180.0°	180.0°	0.500	0.500	1.000	0.750	0.833	0.850	0.500	1.000	0.400
	0.500	0.500	1.000	240.0°	240.0°	0.500	0.500	1.000	0.750	0.667	0.557	0.500	1.000	0.250
	0.750	0.250	0.750	300.0°	300.0°	0.500	0.500	0.750	0.500	0.583	0.457	0.667	0.500	0.571
	0.628	0.643	0.142	61.8°	61.5°	0.501	0.494	0.643	0.393	0.471	0.581	0.779	0.638	0.699
	0.255	0.104	0.918	251.1°	250.0°	0.814	0.750	0.918	0.511	0.426	0.242	0.887	0.832	0.756
	0.116	0.675	0.255	134.9°	133.8°	0.559	0.504	0.675	0.396	0.349	0.460	0.828	0.707	0.667
	0.941	0.785	0.053	49.5°	50.5°	0.888	0.821	0.941	0.497	0.593	0.748	0.944	0.893	0.911
	0.704	0.187	0.897	283.7°	284.8°	0.710	0.636	0.897	0.542	0.596	0.423	0.792	0.775	0.686
	0.931	0.463	0.316	14.3°	13.2°	0.615	0.556	0.931	0.624	0.570	0.586	0.661	0.817	0.446
	0.998	0.974	0.532	56.9°	57.4°	0.466	0.454	0.998	0.765	0.835	0.931	0.467	0.991	0.363
	0.099	0.795	0.591	162.4°	163.4°	0.696	0.620	0.795	0.447	0.495	0.564	0.875	0.779	0.800
	0.211	0.149	0.597	248.3°	247.3°	0.448	0.420	0.597	0.373	0.319	0.219	0.750	0.601	0.533
	0.495	0.493	0.721	240.5°	240.4°	0.228	0.227	0.721	0.607	0.570	0.520	0.316	0.290	0.135

Figure 4.1. 2 https://blog.csdn.net/kakiebu/article/details/79476235

The model of HSV (hue, saturation, value) color space corresponds to a subset of cones in the cylindrical coordinate system, with the top face of the cone corresponding to V=1. It contains the three faces R=1, G=1, and B=1 of the RGB model, represented by the brighter colors. The color H is given by the rotation angle around the V-axis. Red corresponds to an angle of 0°, green to an angle of 120°, and blue to an angle of 240°. In the HSV color model, each color differs from its complementary color by 180° . The saturation S takes values from 0 to 1, so the radius of the top surface of the cone is 1. The color field represented by the HSV color model is a subset of the CIE chromaticity diagram, and colors with a saturation of 100 percent in this model generally have a purity of less than 100 percent. At the apex of the cone (i.e., the origin), V=0, H and S are undefined and represent black. At the top center of the cone, S=0, V=1, H is undefined and represents white. From this point to the origin, it represents gray with diminishing brightness, i.e., gray with different shades of gray. For these points, the values of S=0,H are not defined. It can be said that the V-axis in the HSV model corresponds to the main diagonal in the RGB color space. The color on the circumference of the top surface of the cone, V=1 and S=1, is a pure color.

4.2. Homography

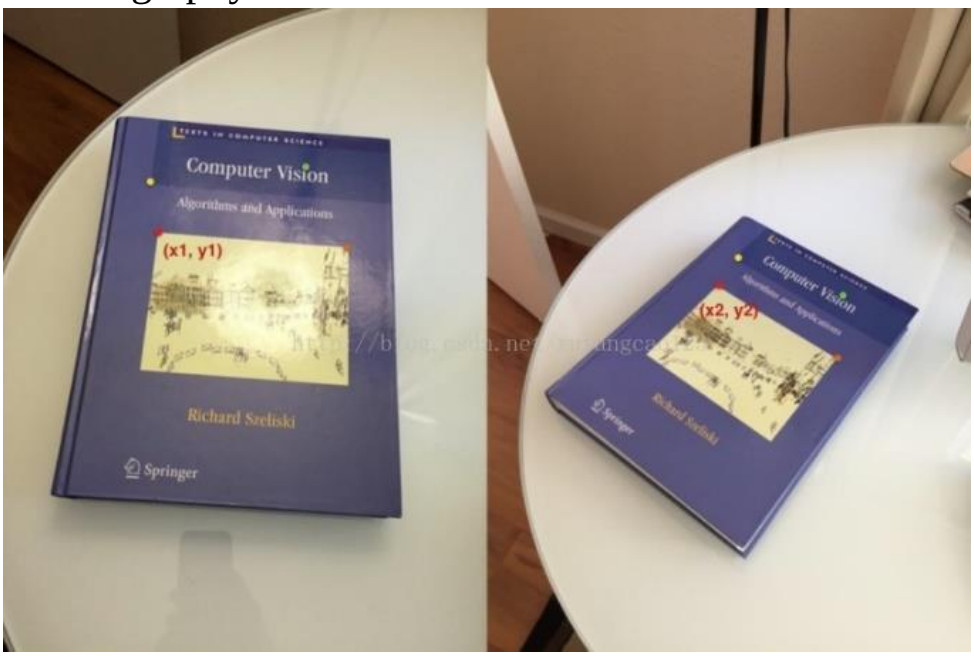

Figure 4.2. 1 source: https://blog.csdn.net/xuyangcao123/article/details/70916767

As shown in the figure 4-b-1, the points of the same color in two images are called Corresponding Points, for example, two red dots are a pair of Corresponding Points. a single correspondence matrix (Homography) is a transformation matrix (3*3) of the mapping relationship from one image to another. It can be expressed by the following equation,

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

Figure 4.2. 2 source: https://blog.csdn.net/xuyangcao123/article/details/70916767

Taking the red dots in the image as an example, the single-responding transformation can be written in the following form,

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

Figure 4.2. 3 source: https://blog.csdn.net/xuyangcao123/article/details/70916767

4.3. Inverse Kinematics

Assuming that the arm has six degrees of freedom, there are six unknown joint angles. According to the inverse kinematic concept, we know the transformation matrix of {6} with respect to {0}.

$${}^0_6T = \left[\begin{array}{ccc|c} & {}^0_6R_{3\times3} & & {}^0P_{6\,org_{3\times1}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]_{4\times4} = \left[\begin{array}{ccc|c} | & | & | & | \\ {}^0\hat{X}_6 & {}^0\hat{Y}_6 & {}^0\hat{Z}_6 & {}^0P_{6\,org} \\ | & | & | & | \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Figure 4.3. 1 source: https://blog.csdn.net/huangjunsheng123/article/details/109314877

The rotation matrix part has nine numbers, indicating three degrees of freedom, indicating that there are six constraints inside, three of which are that X, Y, and Z are all unit vectors with length 1; the other three are that two are perpendicular to each other. The position of the origin of {6} is independent of the three numbers in the {0} coordinate system, so it contains three degrees of freedom. So there are six degrees of freedom in the matrix, a total of 12 equations, six constraints, and six unknowns to solve.

5. Implementation

5.1. Image Processing

5.1.1. Base on BGR

Although the code runs fast, the overall result is not ideal. Overly sensitive to lighting conditions, the square will show different color characteristics under different lighting, which means that basically every time a new environment must recreate the template

i. Simple grayscale binarization

   Early experimental route, it was difficult to choose a threshold for the switch function in the dormitory environment, either there were too many areas in the background that were considered brick, or the outcome could not show the location of the brick, so it was not used.

ii. Color feature extraction

   At the stage when the template matching failed, we try to extract the color features of pink from the RGB map, but the conclusion is that pink does not have many features.

   Proportionally, as the light increases, the green channel basically remains the same, but the red channel decreases and the blue channel increases. The color even flushes blue in high light, while the picture has serious noise in low light. The sweet spot of light is narrow, not very good summary characteristics. And the dependence on the environment will be too high.

   So, we came up with the idea of calculating the BGR offset, specifically by extracting the line profile from the colors of the squares in the picture. then calculate the average value of each color, calculate an offset to this average value and use it to build a grayscale map. This worked fine, but the computational load was too high, which about three seconds per frame.

5.1.2. Base on Contour

At the time, cv.Canny() was used in conjunction with template matching. The reason for abandonment is similar to grayscale. contours under the BGR channel are unexpectedly affected by lighting, presumably because the lighting sensitivity of BGR is inherited to the contour recognition session as the image is taken.

The large noise in the H channel under the HSV channel can seriously interfere with the contour recognition.
The operation method at that time may not be the best. Directly using findContour() and drawContour() in cv has the potential to perform better, but these two require single-channel images. The problem of poor distinction between pink and background under grayscale map will still exist, so it will not be used either.

5.1.3. Base on HSV

The default images are read out as RGB and need to be converted to HSV, which has performance overhead. In the dormitory environment, the H channel can distinguish pinks and other colors very well, but there is very serious large noise and unclear contours. the S and V channel values change basically similarly and can distinguish the top and sides of the square, but poorly with the background.
The distinction between pink and white under the H channel is not good, but it is enough to match. The actual deployment needs to recreate the template.

We finally decided to use the HSV-based image preprocessing method. First, we convert the image from RGB channel to HSV channel, and then perform Gaussian blur to reduce the noise.

5.2. Model Matching

5.2.1. Template match

i. Location match

The principle is by panning the template over the image and calculating the match level between the template and the image. Theoretically it is not suitable for matching with angles. Our approach is to ensure that the object blocks are distinguishable from the background by image preprocessing, and to avoid using binarization to retain enough information for angle matching. Because for objects with angles, we need a not so matching result, so that it can both confirm the matching coordinate position immediately and ensure that a more matching result can appear in the angle matching session as a way to confirm its angle.
Code:

Match the template and angle

Get the coordinates and angle of the bricks in the real world

In [45]:

```
def template_match(photo,template,showInWindow=False):
    height, width,c = template.shape
    results = cv.matchTemplate(photo, template, cv.TM_SQDIFF_NORMED)
    minValue, maxValue, minLoc, maxLoc = cv.minMaxLoc(results)
    minLoc
    #updateValue(minValue)

    return minValue,minLoc
```

Figure 5.2.1. 1 location match

ii. Angle match

The principle of angle match is to rotate the frame with the object position as the center and re-match it, picking the angle with the highest match within 90 degrees, in steps of 5 degrees, if the location match can already determine the object position. Since the brick is rotationally symmetric, it makes no sense to rotate the angle beyond 90 degrees.

Code:

```
def angleMatch(photo,template):
    photoCopy=photo.copy()
    step=5
    angle=0
    h,w,c = photo.shape
    ht,wt,ct=template.shape
    m,center=template_match(photo,template,True)
    m=0.15
    print(m)
    for i in range (0,int(90/step)):
        M = cv.getRotationMatrix2D(center, i*step, 1.0) #12
        rotated = cv.warpAffine(photo, M, (w, h))
        minValue,minLoc=template_match(rotated,template,False)
        print("minValue: "+str(minValue)+", angle: "+str(i*step))
        if minValue<m:
            m=minValue
            angle=i*step
            cv.imshow("bestMatch",rotated)
    comp=calcCompensationVector(minLoc)
    resultPoint1 = (int(center[0]+comp[0]),int(center[1]+comp[1]))
    resultPoint2 = (int(resultPoint1[0] + wt+comp[0]), int(resultPoint1[1] + ht+comp[1]))

    photoCopy=cv.rectangle(photoCopy, resultPoint1, resultPoint2, (0, 0, 255), 2)
    cv.imshow("img", photoCopy)
    k=cv.waitKey(1)
```

Figure 5.2.1. 2

It was used from the beginning to the end. The performance was stable for a long time after the image preprocessing session was in place, but there was a period of failure at a later stage. It was found that there was a relationship with the size of the template and the presence or absence of contours. A mismatched template size can lead to positioning errors, and a template without contours can lead to problems with angle matching. Oversized templates can cause a loss of match with a brick that is supposed to match in height. The gradient change of the brick profile is also important information for angle matching.

5.2.2. Shape match and feature point match

An alternative solution that was studied at a later stage when template matching failed.

Shape matching was very incompatible with the technical route we had in place at the time, and basically everything within the process from image pre-processing to matching had to be reversed. The core is findContour() and drawContour(), so the problem of poor differentiation

in grayscale images still exists. We considered masking the H data in the HSV channel to mask out the background, but the outline given by H itself is rather blurred, and even if it is masked out, there is not necessarily a good distinction between the object block and the large noisy outline in the background, so this solution was abandoned.

5.3. Error Correction

5.3.1. Mechanical error

At first, the mechanical error was corrected by giving an offset on the XY axis, which was not very effective. The reason is that the mechanical error of the robot is essentially in polar coordinates. Later, we realized this problem and made a new set of error correction with multi-angle gripping. The amount of correction needed for the radius of the robot in different working modes at different radii was stored in a table, and the XY coordinates were corrected according to the polar coordinates before the coordinates were passed to the inverse transformation. The result is very good.

5.3.2. Perspective distortion

The image captured by the camera is usually not a rectangle, because the camera cannot guarantee absolute perpendicularity to the ground, while the uncertainty of the position of the bricks makes it impossible to avoid distortion of the view of the bricks.

The more skewed the camera is, the greater the distortion is. Also, considering the effects of lighting, robot height and work area, it is not always possible to set up a camera directly above the work area. Moreover, this would require the relative coordinates of the robot and the camera to be highly controllable, but it is very troublesome to do in practice, so in practice we use the single-response transformation to eliminate the viewpoint distortion.

Code:

Compensation Vector

Compensate for the coordinates, as the bricks have thickness and the camera is not perpendicular to the plane

In [5]:

```
def calcCompensationVector(cord):
    xOffset=(cordMax[0][0]-cordMin[0][0])*cord[0]
    yOffset=(cordMax[1][0]-cordMin[1][0])*(1-cord[1])
    #updateValue((xOffset,yOffset))
    return xOffset, yOffset
```

Figure 5.3. 1

5.4. Coordinate calculation

To calculate the coordinates, we use the theoretical basis of inverse kinematics. The relevant code is shown below

Code:

Backward transformation

Specification of the target vector w from 3 spatial coordinates and 3 rotation angles. It is match with the numbering in the documented calculation algorithm. The target vector w

```
def genString(q,claw,speed):
    # function to generate the string for the serial communication with the robot
    # convert radiant to degrees
    q_deg=q
    print(q_deg)

    # generate the robot string
    go = True
    for angle in q_deg:
        #print(angle)
        try: #if something goes wrong, the script won't stop
            if not 0 <= angle <= 180:
                pass
                #print(angle)
                #go=False
                #break
        except:
                pass
                #go=False
                #break
    if go:
        command="P"+str(int(q_deg[0]))+"," \
                +str(int(q_deg[1]))+"," \
                +str(int(q_deg[2]))+","\
                +str(int(q_deg[3]))+","\
                +str(int(q_deg[4]))+","+str(int(claw))+","+str(int(speed))+"\n"
       # print(angles)
        command=bytes(command,encoding='ascii')
    else:
        #print("not in range")
        command = None
    return command
```

Figure 5.4. 1

```
def backward_transformation(w1,w2,w3,w4,w5,w6,isElbowUp):
    d1=80
    a2=125
    a3=125
    a4=85
    d5=160

    q1=atan2(w2,w1)
    q234=atan2(-w4*cos(q1)-w5*sin(q1),-w6)

    b1 = w1*cos(q1) + w2*sin(q1) - a4*cos(q234) + d5*sin(q234)
    b2=d1-a4*sin(q234) - d5*cos(q234)-w3
    q3=acos((b1**2+b2**2-a2**2-a3**2)/(2*a2*a3))
    if isElbowUp==false:
        q3=-q3
    q2 = atan2((a2 + a3*cos(q3))*b2 - a3*b1*sin(q3) , (a2 + a3*cos(q3))*b1 + a3*b2*sin(q3))
    q4 = q234- q2 - q3
    q5=pi*log(sqrt(w4**2+w5**2+w6**2)) + q1
    q5 = q5 % pi
    q1=q1/pi*180-6
    q2=180+q2/pi*180-5
    q3=90+q3/pi*180-7
    q4=90+q4/pi*180
    q5=q5/pi*180

    if q3>180:
        #print("Compensated\n")
        #q2=q2+(q3-180)/2
        q3=180

    result=[math.floor(q1),math.floor(q2),math.floor(q3),math.floor(q4),math.floor(q5)]

    return result
```

Figure 5.4. 2

5.5. Capture Execution

There are six working modes -60 degrees, -45 degrees, -30 degrees, 0 degrees, 30 degrees and 45 degrees. Angle refers to the angle of the claw deviation from the vertical downward direction. The negative sign means facing inward, and the positive sign means facing outward.

Actually have a role is -45 ~ 45 degrees of work mode. -60 degrees can not be used, because this working mode will touch the base of the robot arm.

The maximum angle of 45 degrees is chosen for a reason. q4 to the claw distance is fixed. In order to achieve the maximum working area should make this section as parallel to the ground as possible. The angle between these two points is 45 degrees parallel to the ground. But due

to mechanical errors our robot q2 actually bends less than the real 180 degrees, the actual ideal angle should be slightly smaller than this, but the impact on the work area should not be obvious.

Code:

```
x=location[0]
y=location[1]

x=x
y=90+y
print("x:"+str(x)+", y:"+str(y))
r=(x**2+y**2)**0.5
print("r:"+str(r)+"\n")
rCompensation={-60:-30,-45:-45,-30:-25,0:-10,30:-12,45:45}
hGrabCompensation={-60:30,-45:20,-30:15,0:10,30:20,45:20}

if r<110:
    workMode=-60
elif 110<=r<160:
    workMode=-45
elif 160<=r<210:
    workMode=-30
elif 210<=r<300:
    workMode=0
elif 300<=r<370:
    workMode=30
elif 300<=r<=390:
    workMode=45
else:
    print("!Not in workzone!")
    quit(1)
print("workMode: "+str(workMode))

x*=(r+rCompensation[workMode])/r
y*=(r+rCompensation[workMode])/r
print("x:"+str(x)+", y:"+str(y))
try:
    if workMode!=0:
        q = backward_transformation(x,y,hGrabCompensation[workMode],0,sin(workMode/180*pi),-cos(workMode/180*pi),True)
    else:
        qStandby = backward_transformation(x,y,40,0,sin(workMode/180*pi),-cos(workMode/180*pi),True)
        q = backward_transformation(x,y,20,0,sin(workMode/90*pi),-cos(workMode/90*pi),True)
except TypeError:
    print("!Not in workzone!")
    quit(1)
```

Figure 5.5. 1

The work area is divided into two parts, the first part is a radius of 210 mm to 300 mm area. Within this range, the robot works in 0 degree working mode and can rotate its claws to match the angle of the brick with a very high probability of grasping.

Outside this area, the robot works in other modes in the area smaller than the radius of 390 mm. The claws are tilted in other modes, so it can only grab bricks with a low success rate.

Code:

```
if workMode<0:
    q[4]=90
    qStandby=q.copy()
    qStandby[2]=90
    s.write(genString(qStandby,0,50))
    time.sleep(2)
    s.write(genString(q,0,20))
    time.sleep(3)
    s.write(genString(q,70,30))
    time.sleep(1.5)
elif workMode==0:
    q[4]+=angle
    qStandby[4]+=angle
    s.write(genString(qStandby,0,40))
    time.sleep(2)
    s.write(genString(q,0,40))
    time.sleep(2)
    s.write(genString(q,70,30))
    time.sleep(1.5)
elif workMode>0:
    q[4]=90
    qStandby=q.copy()
    qStandby[3]=0
    s.write(genString(qStandby,0,50))
    time.sleep(1.5)
    s.write(genString(q,0,30))
    time.sleep(3)
    s.write(genString(q,70,30))
    time.sleep(1.5)

s.write(b'P90,85,83,90,90,50,90')
```

Figure 5.5. 2

6. Test and Evaluation

6.1. Site preparation

First, we place the chair on the table and then stick the camera just on the edge of the chair like the following picture. The camera should aim at the space under it with a little angle and be sticked by the tape toughly so that the shoot area by camera will be stable. Any shake by the camera or chair may lead to the deviation of the result.

Figure6.1. 1

Then, we tape the area which can be seen by the camera. We use Ni vision assistant – an application on the laptop to see the picture taken by the camera. We use this application because it can be greatly helpful for us to do the picture processing later. So, we tape this rectangle area on the table according to the picture taken shown in the following picture.

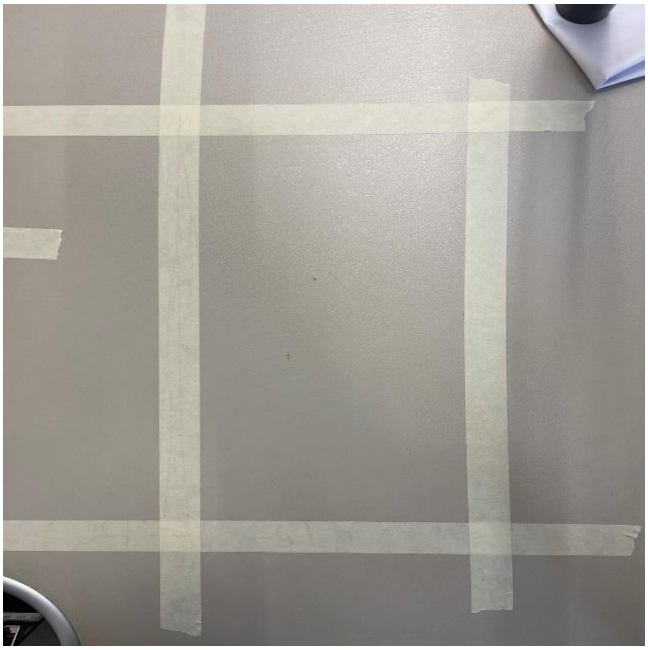

Figure6.1. 2

After that, we place the robot on one prolonged side of the rectangle area. The distance away should be not so far away because of the distance reached by the robot. Moreover, we also firm the bottom of the robot.

Figure6.1. 3

Finally, we plug in the power of robot and connect all the wires to the computer.

6.2. Parameter Setting

Adjust the camera view to ensure that the work area is within the shooting field of view. After determining the camera view, take a picture as a template for the work area. And measure the length of the rectangular working area and enter the length and width as basic parameters into the preparation code.

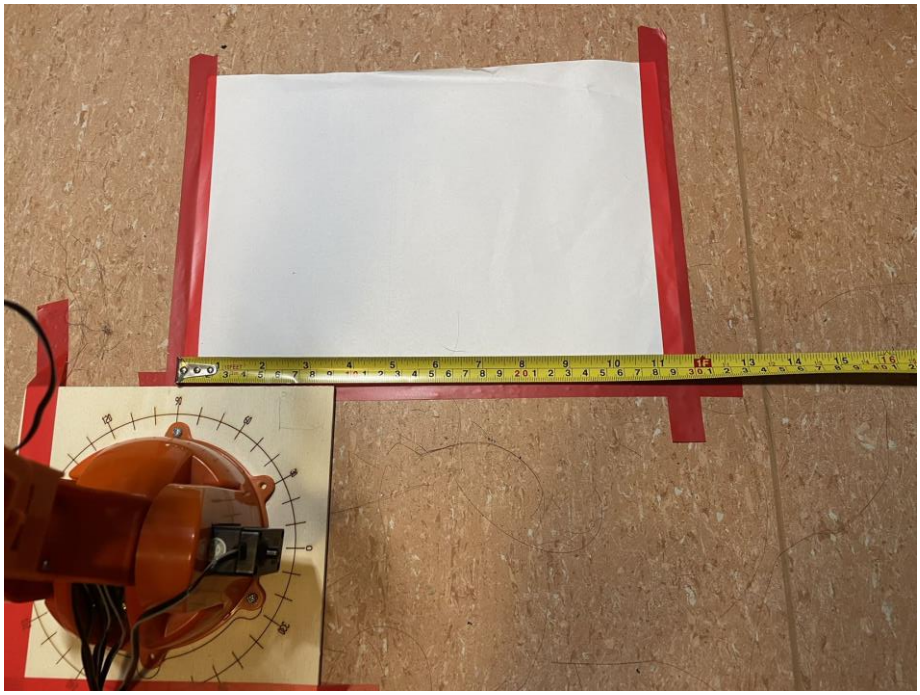

Figure 6.2. 1

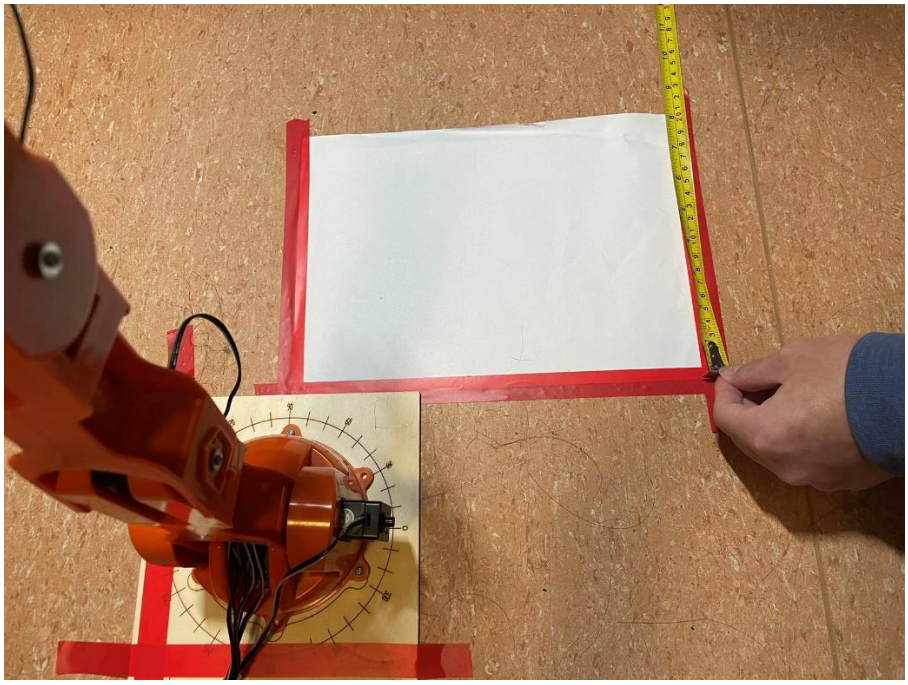

Figure 6.2. 2

Code:

Define the image processing parameter

In [2]:

```
# Width and height for the working area
areaWidth =295
areaHeight =210

# Coordinates of the top left, top right, bottom left and bottom right corners of the image
pts_src =np.array([ (276,261), (838, 248), (239, 661), (866, 645)])
transformedWidth=0
transformedHeight=0
transformedRatio=0

# The prism of the square is a vector from the top to the bottom, with the maximum value at the vertex closest to the viewpoint and the minimum value at the vertex fart
cordMinOrg=np.array([[779],[290],[1]])#Minimum vector starting point
cordMinDst=np.array([[778],[308],[1]])#Minimum vector termination point
cordMaxOrg=np.array([[297],[584],[1]])#Maxmum vector starting point
cordMaxDst=np.array([[308],[591],[1]])#Maxmum vector termination point
```

Figure 6.2. 3 define the processing parameter

Place the target block to the two corners nearest and farthest from the robot arm in the working area and measure its prism length in the vertical direction. The error is calculated by the principle of homography transformation in order to reduce the error in the subsequent calculation process.

Code:

Homography transformation setup

In [4]:

```
# 目标物体中的物体的四个顶点信息
yoffSet=90
pts_dst = np.array([(640+areaWidth, 719-yoffSet), (640, 719-yoffSet), (640+areaWidth, 719-areaHeight-yoffSet), (640, 719-areaHeight-yoffSet)])
# if areaWidth/areaHeight>1.77:
#     transformedWidth=1280
#     transformedRatio=1280/areaWidth
#     transformedHeight=areaHeight*transformedRatio
#     pts_dst = np.array([(0, 0), (transformedWidth, 0), (0, transformedHeight), (transformedWidth, transformedHeight)])
# else:
#     transformedHeight=720
#     transformedRatio=720/areaHeight
#     transformedWidth=areaWidth*transformedRatio
#     pts_dst = np.array([(0, 0), (transformedWidth, 0), (0,transformedHeight), (transformedWidth, transformedHeight)])
homography, status = cv.findHomography(pts_src, pts_dst)

cordMinOrg=homography.dot(cordMinOrg)
cordMinDst=homography.dot(cordMinDst)
cordMaxOrg=homography.dot(cordMaxOrg)
cordMaxDst=homography.dot(cordMaxDst)

cordMinOrg=cordMinOrg/cordMinOrg[2][0]
cordMinDst=cordMinDst/cordMinDst[2][0]
cordMaxOrg=cordMaxOrg/cordMaxOrg[2][0]
cordMaxDst=cordMaxDst/cordMaxDst[2][0]

cordMin=cordMinDst-cordMinOrg
cordMax=cordMaxDst-cordMaxOrg

def homographyTransform(photo):
    h, w, c = photo.shape
    im_out = cv.warpPerspective(photo, homography, (w, h))
    return im_out
```

Figure 6.2. 4 homography transformation setup

Compensation Vector

Compensate for the coordinates, as the bricks have thickness and the camera is not perpendicular to the plane

In [5]:

```
def calcCompensationVector(cord):
    xOffset=(cordMax[0][0]-cordMin[0][0])*cord[0]
    yOffset=(cordMax[1][0]-cordMin[1][0])*(1-cord[1])
    #updateValue((xOffset,yOffset))
    return xOffset, yOffset
```

Figure 6.2. 5 compensation vector

Then the target block is placed in the middle area of the working area, a photo is taken and only the fast part of the target is intercepted and used as the target block matching template.

Figure 6.2. 6

6.3. Image test

We segment an A4-sized white paper every 1.9cm in both vertical and horizontal directions. A total of 14*10 arrays of 1.9cm*1.9cm squares were obtained. The target fast was placed in each array for the image test and capture test.

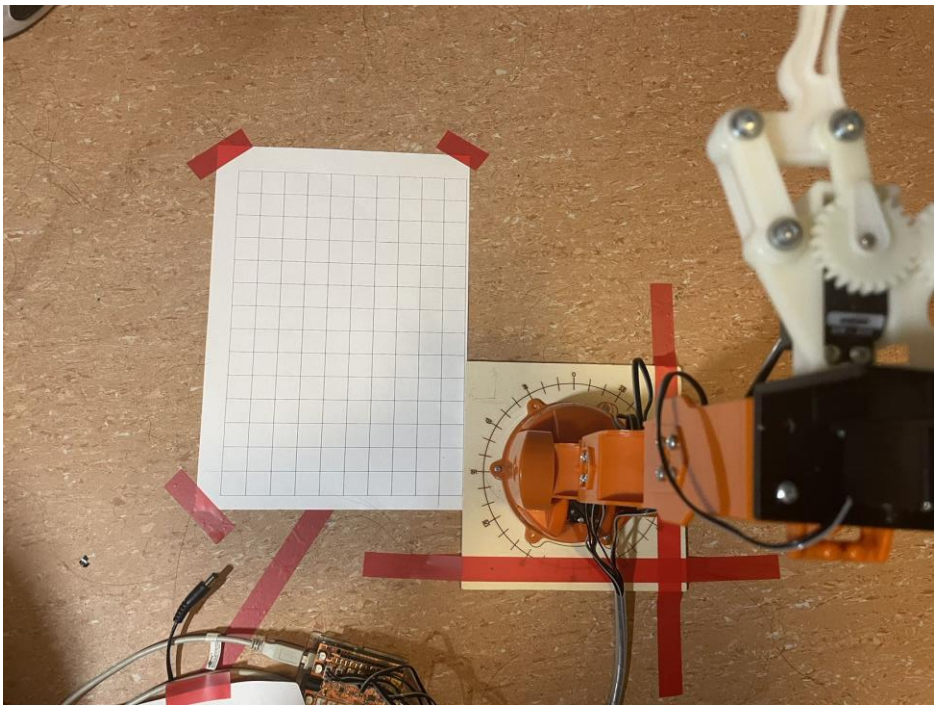

Figure 6.3. 1

After running the graphics code in pycharm, a visualization window will pop up to represent the results of the machine processing.

Code:

Initialize the match template

In [28]:

```
flag,frame = cap.read()
cv.imwrite('template.jpg',homographyTransform(frame), [int( cv.IMWRITE_JPEG_QUALITY), 100])
```

Figure 6.3. 2

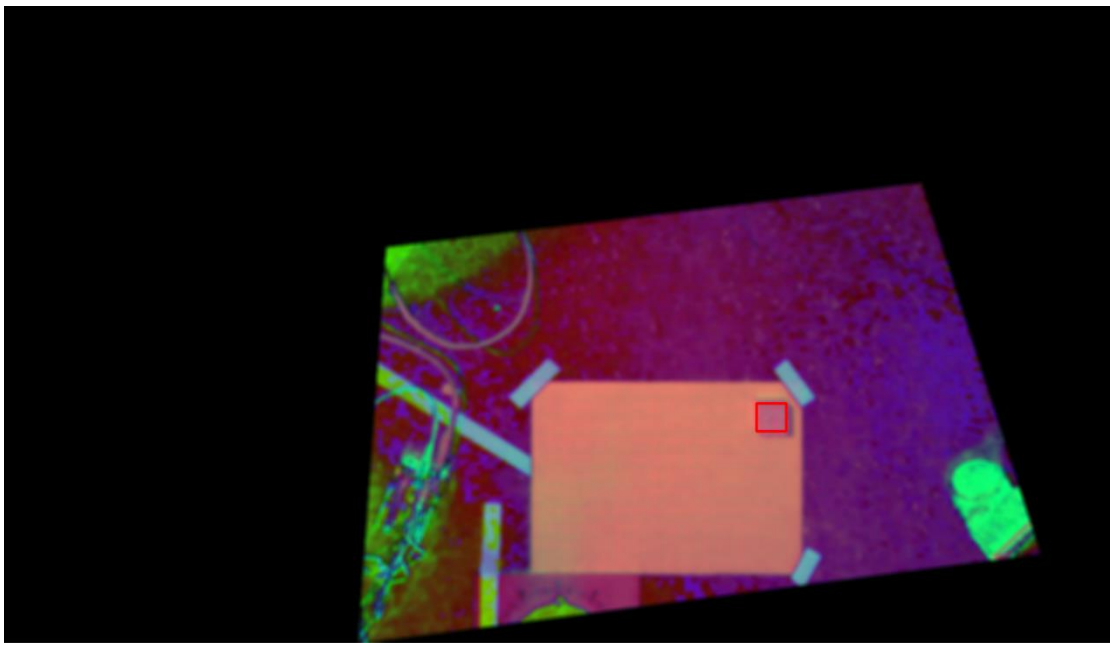

Figure 6.3. 3 image test 1

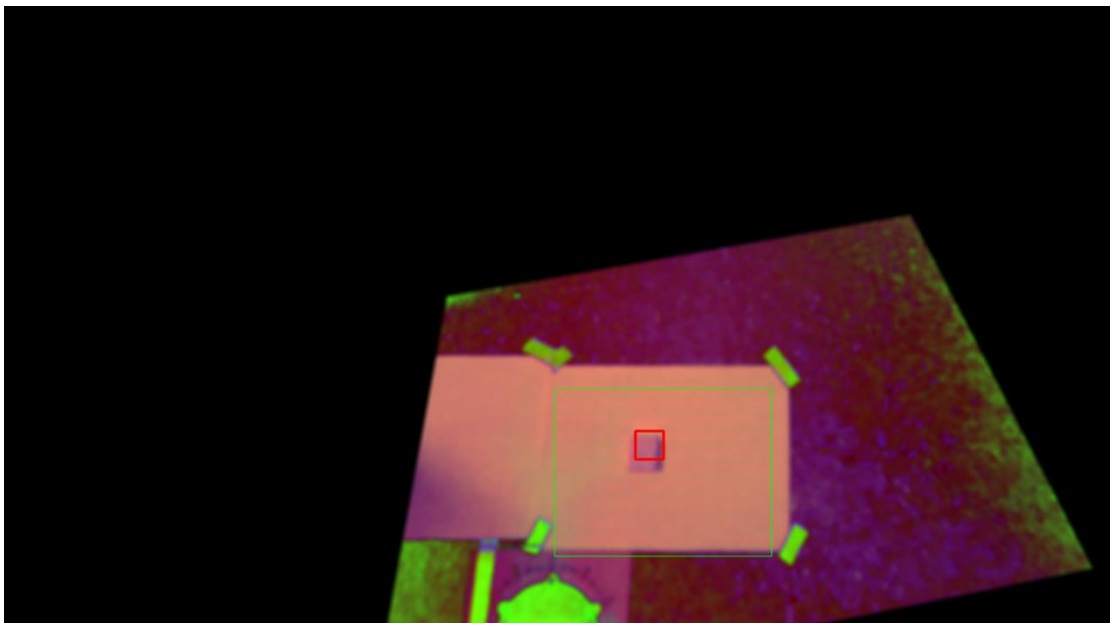

Figure 6.3. 4 image test 2

As you can see in picture 6.3.2, the green box represents the working area of the robot arm, the square array mentioned before. You can compare image 6.3.2 with image 6.3.1 and you will see that the green box perfectly frames the working area. And the red box indicates the position of the target block derived from the calculation. Although the lighting conditions may not be the same for each test, our program is now able to reduce the error to within 10mm for most positions. In some extreme positions, such as directly below the mechanical jaws, or where there is shadow coverage, there may be drift in the calculation results.

The following graph shows the results of the program calculation at different coordinates and angles.

Actual angle at different coordinate				
coordinate / angle	(38,38)	(76,76)	(152,152)	(247,19)
0	0	0	0	0
30	25	20	25	20
45	40	40	40	40

Figure 6.3. 5

The following graph shows the results of the program calculation at different coordinates and the error analysis. The calculated value is subtracted from the actual value, then divided by the maximum value in that direction, and then its absolute value is taken.

X-axis calculated by program		Actual X-axis coordinates													
	mm	0	19	38	57	76	95	114	133	152	171	190	209	228	247
	0	-5	12	33	55	77	95	115	135	154	175	194	213	235	253
	19	-3	19	36	57	74	95	114	133	153	174	195	214	234	254
	38	-2	18	34	57	74	93	113	132	153	173	195	216	234	256
Actual Y-axis coordinates	57	-1	17	35	58	75	94	114	133	152	173	194	214	234	256
	76	-3	15	35	56	74	94	115	133	153	174	194	215	235	255
	95	-4	15	35	57	75	94	114	136	155	173	194	215	235	257
	114	-3	16	35	56	75	94	115	134	155	175	194	214	234	256
	133	-2	15	36	56	74	95	114	133	153	173	195	214	235	256
	152	-2	16	35	57	74	95	114	134	154	174	195	214	234	256
	171	-5	15	36	57	73	95	114	135	154	174	194	214	234	255
	190	-4	15	35	57	74	94	114	135	154	173	194	215	237	256

X-axis deviation percentage		Actual X-axis coordinates													
	mm	0	19	38	57	76	95	114	133	152	171	190	209	228	247
	0	2,02%	2,83%	2,02%	0,81%	0,40%	0,00%	0,40%	0,81%	0,81%	1,62%	1,62%	1,62%	2,83%	2,43%
	19	1,21%	0,00%	0,81%	0,00%	0,81%	0,00%	0,00%	0,00%	0,40%	1,21%	2,02%	2,02%	2,43%	2,83%
	38	0,81%	0,40%	1,62%	0,00%	0,81%	0,81%	0,40%	0,40%	0,40%	0,81%	2,02%	2,83%	2,43%	3,64%
Actual Y-axis coordinates	57	0,40%	0,81%	1,21%	0,40%	0,40%	0,40%	0,00%	0,00%	0,00%	0,81%	1,62%	2,02%	2,43%	3,64%
	76	1,21%	1,62%	1,21%	0,40%	0,81%	0,40%	0,40%	0,00%	0,40%	1,21%	1,62%	2,43%	2,83%	3,24%
	95	1,62%	1,62%	1,21%	0,00%	0,40%	0,40%	0,00%	1,21%	1,21%	0,81%	1,62%	2,43%	2,83%	4,05%
	114	1,21%	1,21%	1,21%	0,40%	0,40%	0,40%	0,40%	0,40%	1,21%	1,62%	1,62%	2,02%	2,43%	3,64%
	133	0,81%	1,62%	0,81%	0,40%	0,81%	0,00%	0,00%	0,00%	0,40%	0,81%	2,02%	2,02%	2,83%	3,64%
	152	0,81%	1,21%	1,21%	0,00%	0,81%	0,00%	0,00%	0,40%	0,81%	1,21%	2,02%	2,02%	2,43%	3,64%
	171	2,02%	1,62%	0,81%	0,00%	1,21%	0,00%	0,00%	0,81%	0,81%	1,21%	1,62%	2,02%	2,43%	3,24%
	190	1,62%	1,62%	1,21%	0,00%	0,81%	0,40%	0,00%	0,81%	0,81%	0,81%	1,62%	2,43%	3,64%	3,64%

Figure 6.3. 6 X-axis calculated by program

Y-axis calculated by program		Actual X-axis coordinates													
		0	19	38	57	76	95	114	133	152	171	190	209	228	247
	0	5	7	6	4	-2	10	13	12	0	-1	0	1	-1	-2
	19	34	29	18	19	18	20	19	17	17	18	18	18	19	18
	38	48	39	39	37	38	38	38	37	38	37	37	40	38	39
	57	67	55	56	57	55	57	56	55	56	57	55	58	57	56
Actual Y-axis coordinates	76	88	74	75	77	75	77	77	75	76	79	74	77	74	76
	95	95	94	95	95	94	95	94	94	95	97	95	96	97	94
	114	115	113	114	114	113	115	115	114	114	116	115	116	115	114
	133	133	133	132	133	132	133	133	132	133	135	133	134	135	132
	152	153	152	152	152	152	154	153	153	153	155	154	154	151	152
	171	174	171	171	171	171	173	172	170	172	172	174	173	172	172
	190	194	192	190	192	188	191	192	190	192	190	192	190	189	191

Y-axis deviation percentage		Actual X-axis coordinates													
	mm	0	19	38	57	76	95	114	133	152	171	190	209	228	247
	0	2,63%	3,68%	3,16%	2,11%	1,05%	5,26%	6,84%	6,32%	0,00%	0,53%	0,00%	0,53%	0,53%	1,05%
	19	7,89%	5,26%	0,53%	0,00%	0,53%	0,53%	0,00%	1,05%	1,05%	0,53%	0,53%	0,53%	0,00%	0,53%
	38	5,26%	0,53%	0,53%	0,53%	0,00%	0,00%	0,00%	0,53%	0,00%	0,53%	0,53%	1,05%	0,00%	0,53%
	57	5,26%	1,05%	0,53%	0,00%	1,05%	0,00%	0,53%	1,05%	0,53%	0,00%	1,05%	0,53%	0,00%	0,53%
Actual Y-axis coordinates	76	6,32%	1,05%	0,53%	0,53%	0,53%	0,53%	0,53%	0,53%	0,00%	1,58%	1,05%	0,53%	1,05%	0,00%
	95	0,00%	0,53%	0,00%	0,00%	0,53%	0,00%	0,53%	0,53%	0,00%	1,05%	0,00%	0,53%	1,05%	0,53%
	114	0,53%	0,53%	0,00%	0,00%	0,53%	0,53%	0,53%	0,00%	0,00%	1,05%	0,53%	1,05%	0,53%	0,00%
	133	0,00%	0,00%	0,53%	0,00%	0,53%	0,00%	0,00%	0,53%	0,00%	1,05%	0,00%	0,53%	1,05%	0,53%
	152	0,53%	0,00%	0,00%	0,00%	0,00%	1,05%	0,53%	0,53%	0,53%	1,58%	1,05%	1,05%	0,53%	0,00%
	171	1,58%	0,00%	0,00%	0,00%	0,00%	1,05%	0,53%	0,53%	0,53%	0,53%	1,58%	1,05%	0,53%	0,53%
	190	2,11%	1,05%	0,00%	1,05%	1,05%	0,53%	1,05%	0,00%	1,05%	0,00%	1,05%	0,00%	0,53%	0,53%

Figure 6.3. 7 Y-axis calculated by program

6.4. Capture Execution

In the square array paper described in the previous step, we place the target blocks at different coordinates for the grasping test. The mechanical jaws can automatically switch the working mode according to different areas to complete the gripping of different working areas.

This if loop is to determine the working area by coordinates.

```
if r<110:
    workMode=-60
elif 110<=r<160:
    workMode=-45
elif 160<=r<210:
    workMode=-30
elif 210<=r<300:
    workMode=0
elif 300<=r<370:
    workMode=30
elif 370<=r<=390:
    workMode=45
else:
    print("!Not in workzone!")
    quit(1)
print("workMode: "+str(workMode))
```

Figure 6.4. 1 Judgment of working area according to coordinates

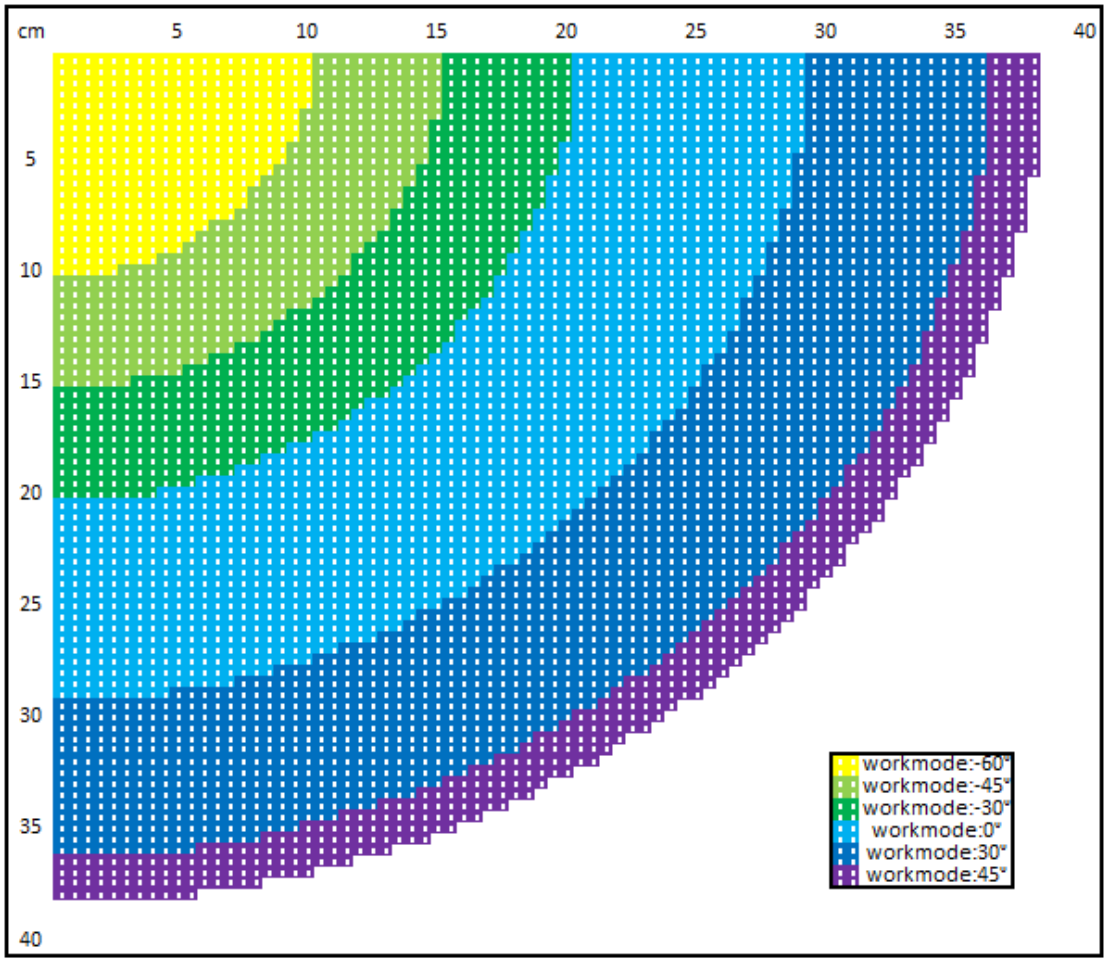

Figure 6.4. 2 working area

The following pictures show the results of different work zones.

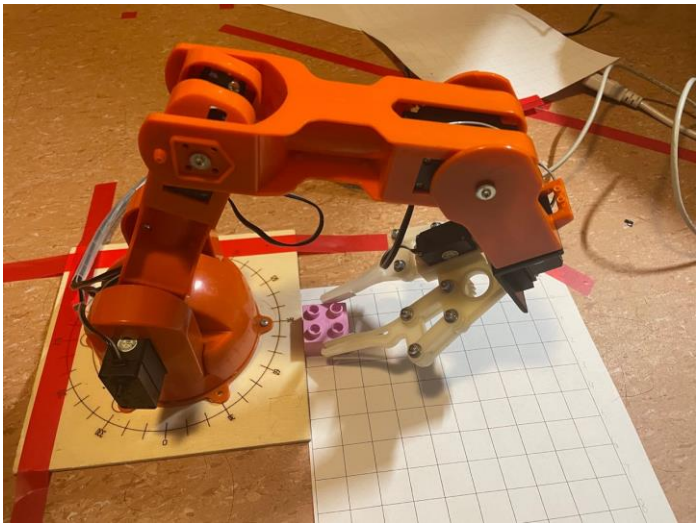

Figure 6.4. 3 -60° working area

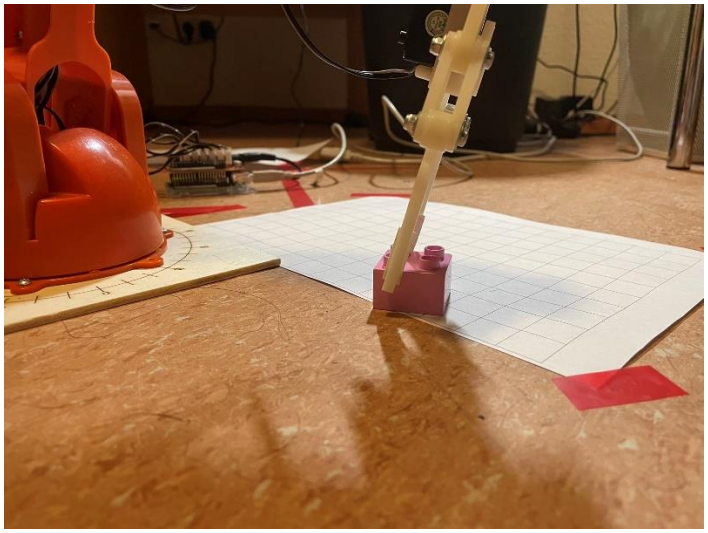

Figure 6.4. 4-45 ° working area

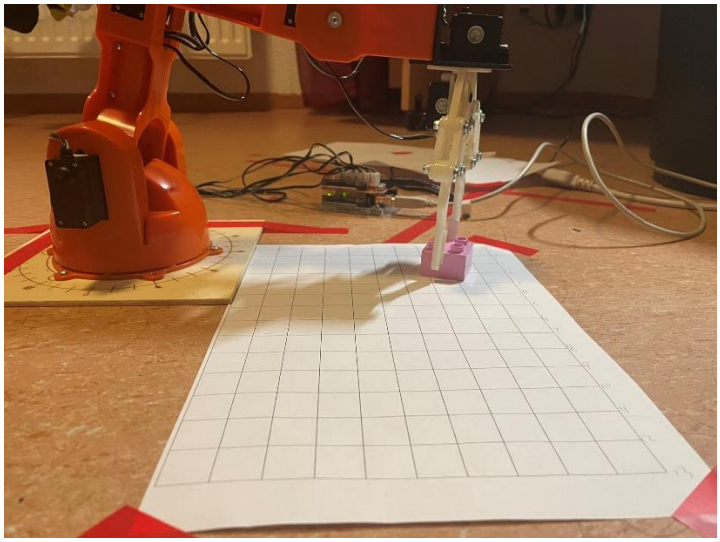

Figure 6.4. 5 0 ° working area

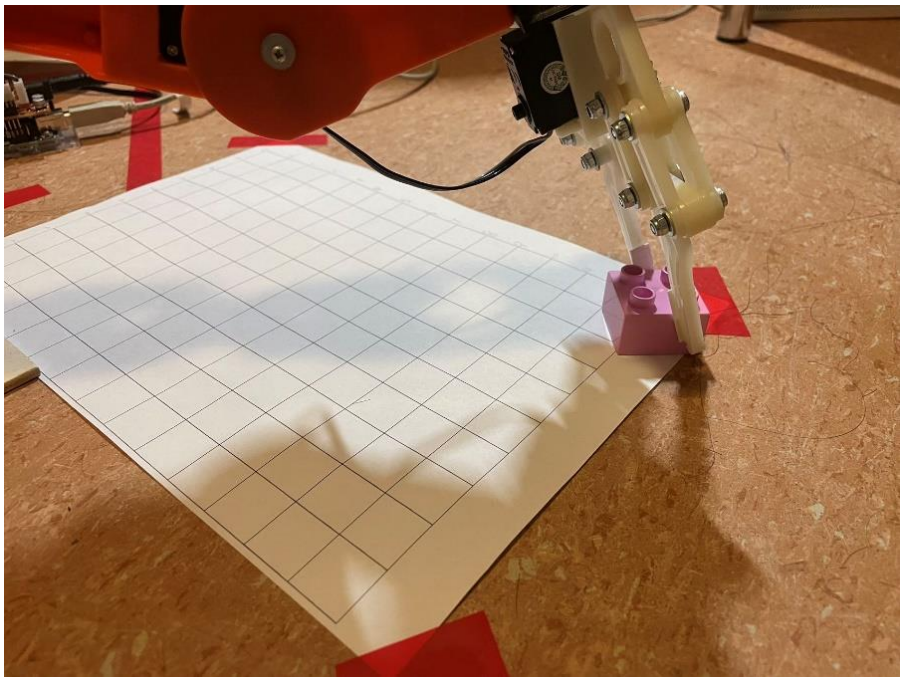

Figure 6.4. 6 45 °working area

6.5. Automatic gripping test

After the robot arm passes the first two tests, the automatic gripping test will be performed. Unlike the capture execution, the coordinates in this test will be passed to the gripping program by the program after the image recognition and the whole process will be executed automatically.

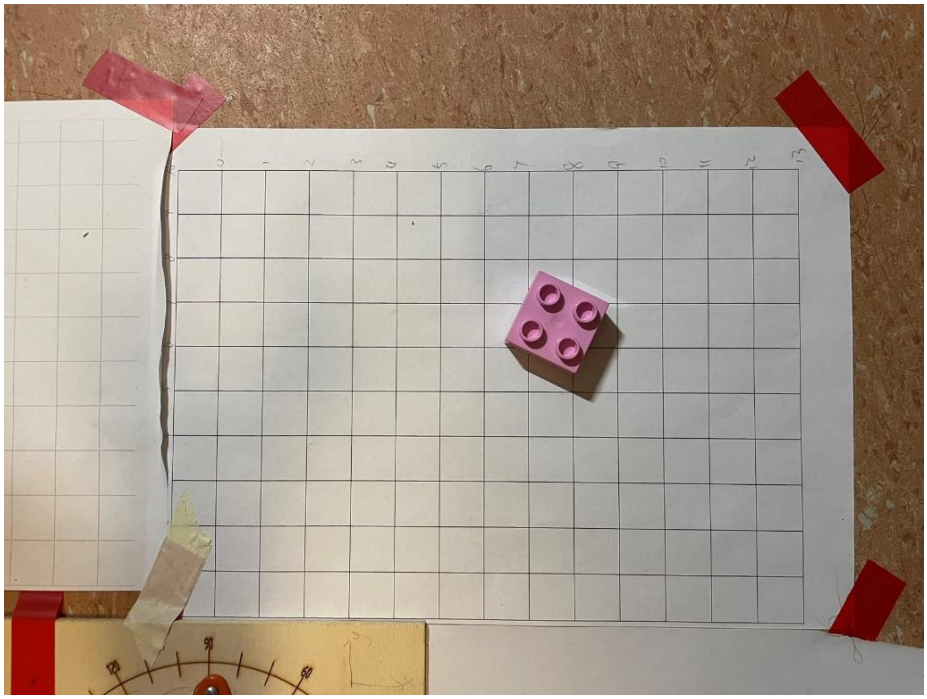

Figure 6.5. 1 position of target block

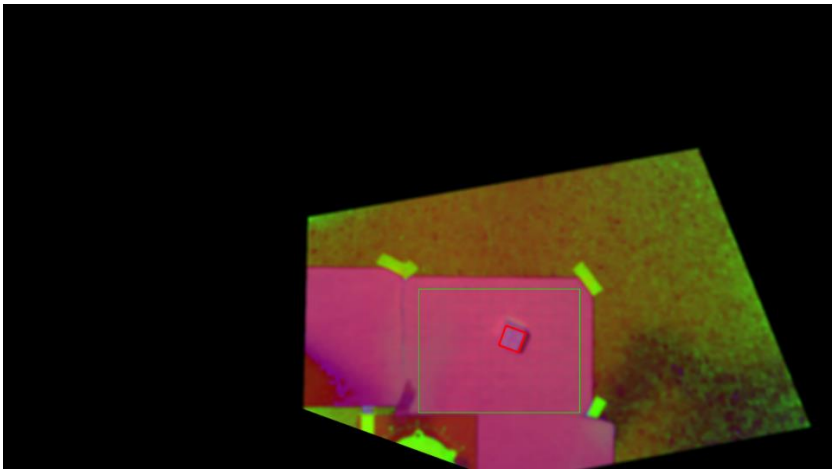

Figure 6.5. 2 image recognition

Get coordinates and angles

```
s.write(b'P90,85,83,90,90,0,90')
template=cv.imread('template.jpg')
template=imagePreprocess(template)
location=0
locCompensated=0
while True:

    flag,frame = cap.read()
    k=cv.waitKey(1)
    if not flag:
        break
    angle,location,locCompensated=angleMatch(imagePreprocess(homographyTransform(frame)),template)
    updateValue((angle,location,locCompensated))
    # if ord(' ') == k:
    time.sleep(3)
    break

cv.destroyAllWindows()
```

```
Realtime Value: (20, (143, 114), (144, 125))
```

Figure 6.5. 3 coordinate output

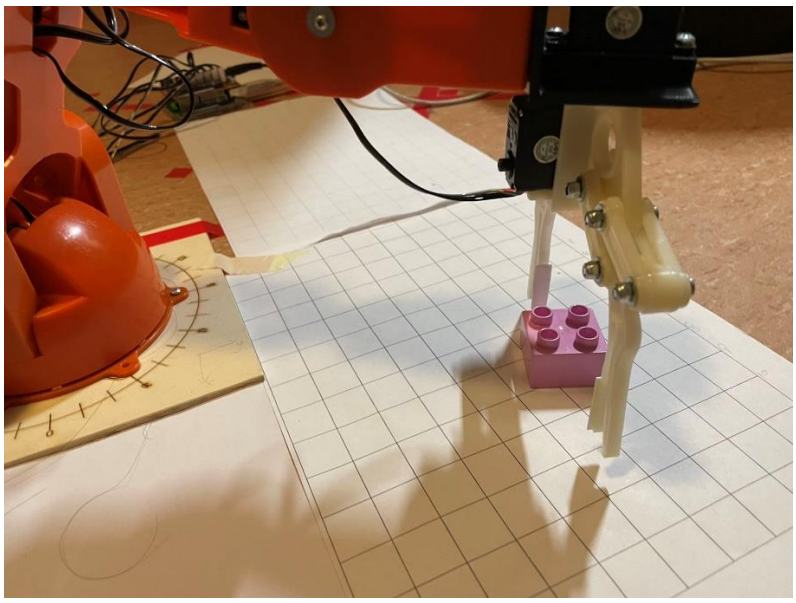

Figure 6.5. 4 Robotic arm gripping process 1

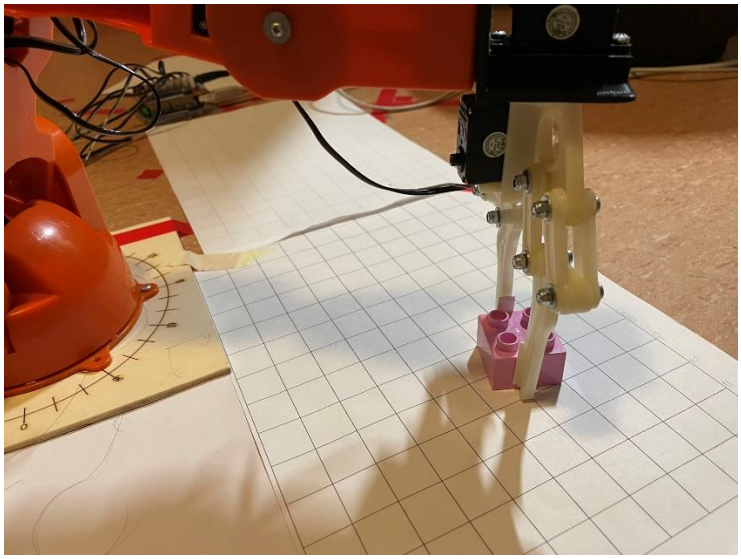

Figure 6.5. 5 Robotic arm gripping process 2

Figure 6.5. 6 Successful execution

It is tested that the program is able to process the image correctly at various places in the working area, change the working mode, and the automatic gripping process of the robot arm can be executed smoothly.

7. Conclusion

In conclusion, we perfectly fulfill the task of our course, which is to accurate grasp the objects by mobile robotic arm. Besides, we also apply some techniques rather novel. For example, the application of homography can free the camera from different angles. The picture shoot after processing is not dependent of the position of the camera.

Secondly, we realize the mechanical error of the robot is because of polar coordinates, then we use a new system to do this mechanical error compensation. The compensation vector can remarkably reduce the error in height of the brick because of the picture taken.

Moreover, our grasping method is done by different angles modes which can also increase the accuracy of our catching. The details are written in the implementation part.

In all, we apparently meet some troubles or small flaws may cause some failure. But finally, we solve all the problems and our accuracy of grasp is relatively high.

This project includes much knowledge in different fields and eventually we can combine all together having a perfect result. Our team cooperates well to complete all tasks.

For the future research, we may use more cutting-edge technology such as using feature point matching while processing the template or having deeper application of HSV. But from now, we have already achieved a great result.

Appendix

https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other

https://electronics360.globalspec.com/images/assets/757/12757/DON_system_and_Kuka_robot_grasp_a_cup.JPG

https://blog.csdn.net/weixin_41033536/article/details/82970511

https://blog.csdn.net/kakiebu/article/details/79476235

https://blog.csdn.net/xuyangcao123/article/details/70916767

https://blog.csdn.net/xuyangcao123/article/details/70916767