University of Tehran

Faculty of Engineering

School of Electrical Engineering

Mechatronics

Final Project Report

:Title

Arranging Colored Blocks in The Production Line Based On Image Processing Using UR10 ROBOT

August 2021

Table of Contents

Page No Title

Abstract

In this project, we intend to use the UR10 6 degrees of freedom robot in the CoppeliaSim simulation environment to pick and place colored cubes. In the simulation, cubes with random colors are placed on the conveyor belt. The robot drops the cubes according to their color into the special box of each color. At the beginning of the work, we obtain the working space of the robot in the environment in which it is located with the help of CoppeliaSim and MATLAB software. To remove the colored cubes, we need to know the coordinates of their center, this is done by image processing on the images obtained from the simulation environment camera with the help of the OpenCV library in Python. To control the robot, we need to solve inverse kinematics equations; For this purpose, we obtain the equations related to direct kinematics according to the DH parameters of the robot, considering the complexity of the inverse kinematic relationships, we perform the calculations numerically using direct kinematic equations. In the MATLAB software, the Python code related to image processing is called and the coordinates of the cubes are obtained as the output of the Python code in the MATALB environment, then according to the relationships we obtained and the numerical solution using the Quasi-algorithm Newton, the angle of each joint of the robot to go to the desired position is obtained. Then, these angles are sent to the CoppeliaSim environment, and using the PID controller, the final operator of the robot goes to the desired position and the Pick and Place operation is performed.

Key words

Pick & Place robot, UR10 robot, inverse and direct kinematics, image processing with OpenCV, CoppeliaSim

1- Introduction

Today, the use of picking and placing robots in the environment of factories and productions is very widespread; The use of these robots greatly speeds up the process of removing and placing parts in another place, thus increasing the speed of production; In other words, these robots perform repetitive actions that are sometimes tiring and exhausting for humans with more speed and accuracy, and also reduce costs and increase efficiency. These robots usually have an advanced image processing system that can detect the location, color and other characteristics of different parts and in applications such as; putting together different parts, Packing different parts in a box or placing them on a pallet. Checking the production parts and identifying the defective parts and separating them from the rest of the products. And ... He used them. Different types of robots can be used for Pick and Place applications, including robots such as: robotic arm, Cartesian robot, Delta robot, Scara robot, which are made in different sizes according to their application. and are exploited. Among the mentioned robots, Robotic Arm robots are widely used in the industry to perform various operations including Pick and Place operations because it is a series robot with 6 degrees of freedom (3 degrees of translational freedom and 3 rotational degrees of freedom) which, while being easy to control, can bear a significant payload and also occupies a small space, but at the same time has a large working space, compared to parallel robots, they are more difficult to control. And it is more expensive, and they also occupy a larger space, and... for these reasons, compared to other robots, Robotic Arms are more common in the industry for the mentioned application.

Figure 1. SCARA robot packaging and palletizing in the IC production line

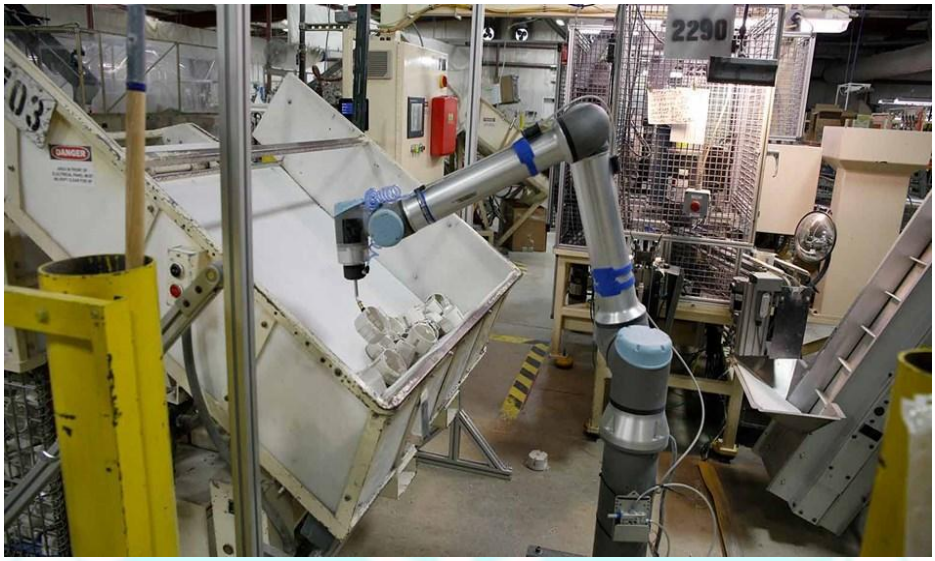


Figure 2. A UR10 robot from Universal Robots A/S performing Pick & Place operations

In this project, we used a UR10 robot, which is a type of robotic arm, and using CoppeliaSim software, Edu version, we simulated the production line environment of a factory, which produces cubic parts (with dimensions slightly larger than the cube) Rubik's standard 3x3) with 3 different colors (red, blue and green) move on a conveyor belt and the robot recognizes the location and color of each cube by processing the image and puts it in the box corresponding to each cube. is thrown; For the image processing part, we used the powerful OpenCV library, Python version, and also used MATLAB software and specifically the Robotics Toolbox developed by Mr. Peter Croke [5] to control the robot and calculate its inverse kinematics. The general procedure was that MATLAB was an intermediary between CoppeliaSim (simulation environment) and Python (OpenCV) and the Robotics toolbox (robot control tools). All these tools and software are used in the Windows environment, of course, due to the fact that they are multi-platform, they can be easily used in other operating systems as well. In order to achieve our goal in this project, we needed to use different image processing algorithms, which are preferably described in the next section, to identify the color, location, and rotation of the cubes on the conveyor belt; Also, direct and inverse kinematic equations were needed to control the robot's movement and routing, which was using the DH parameters of the UR10 robot. To control the location (angle) of the robot's joints from

We used a PID controller and specifically, due to the absence of noise and the ideality of the simulation environment, the use of a P controller was sufficient in this project; We note that most of the techniques and algorithms used in this project were learned in class during the semester and we saw the practical application of our theoretical learning in this project. Next, the details related to the preparation of the simulation environment, finding the robot's workspace, image processing and finding the position and rotation of the cubes, as well as the kinematic model of the robot and its control are given in the next section.

2- Background and Methods

2.1 Simulation Environment:

One of the main parts of the project is the simulation environment, and to start the work, the simulation environment must be prepared first. This environment is made up of different components, first we name its components and then we explain each one briefly; The final designed environment is as follows.

The components of the environment are as follows:

1. Part Producer: This part of the environment has the task of producing cubes.
2. Conveyor: This is the part of the conveyor that moves the cubes towards the robot.
3. Vision Sensor: It is a camera that is used to observe the cubes and recognize their color.
4. Deletion Box: The boxes in which the cubes are supposed to be dropped by color.
5. UR10: The robot used in this project to separate the cubes.
6. Baxter Vacuum Cup: It is a gripper used to hold the cubes.

We examine each of the mentioned sections in detail:

1. Part Producer: According to the algorithm, this part produces cubes with the same dimensions but with three different colors, red, blue, and green, and different positions on the conveyor belt and choosing the color and position randomly. be Considering that 3 or more cubes with the same color may be produced in a row in this process, the algorithm is designed to produce a maximum of 2 duplicate colors in a row, and the reason is simply that the robot can perform for all colors. observed and checked in the separation process. In front of the conveyor belt, the point where the cubes stop for the robot to pick them up, there is a sensor that stops the production of cubes when a cube is in front of the sensor. It is also possible to change the speed of cube parts production by changing the shapeDropFrequency variable.

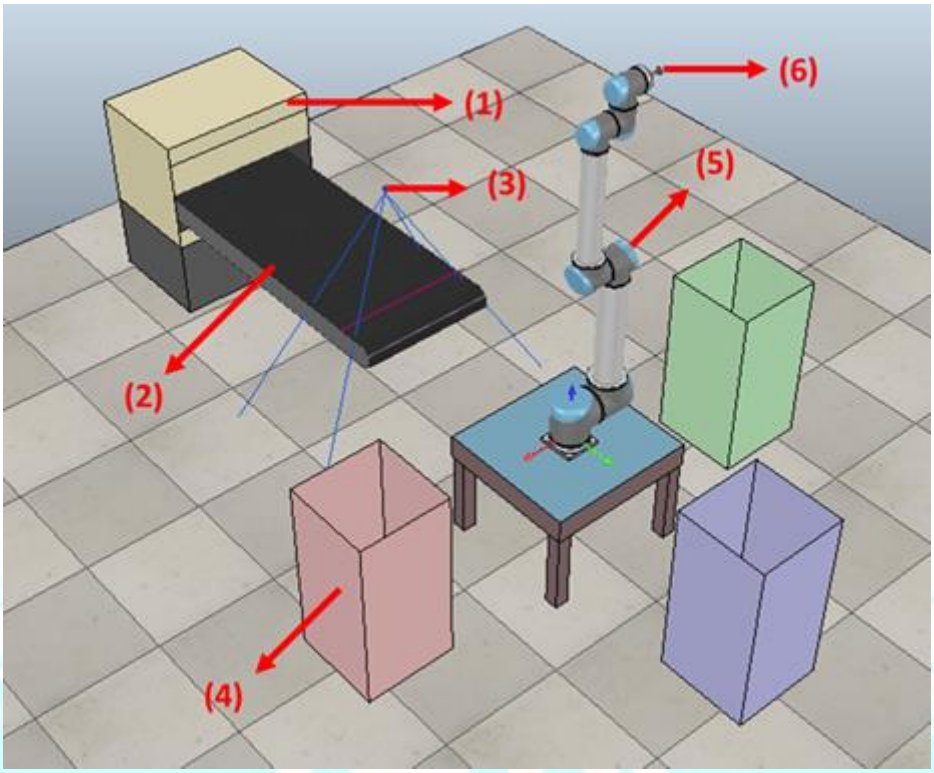


Figure 3. The simulations environment

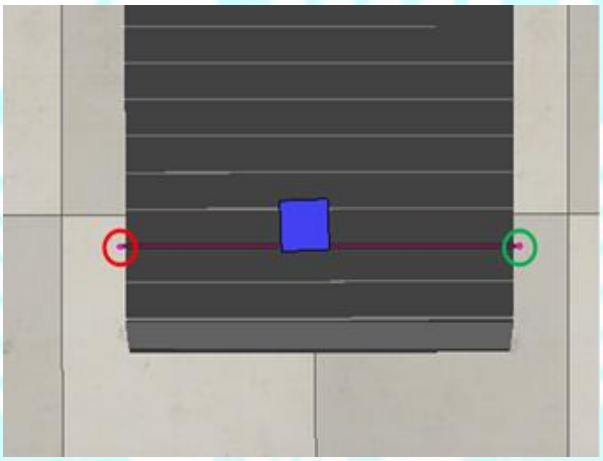


Figure 4. Conveyor and Part Producer sensors

2. Conveyor: It is a conveyor belt on which the produced cubes are placed and moves towards the robot with a specific speed that can be changed. Its dimensions can also be changed. On the side of the conveyor belt that is close to the robot, there is a sensor parallel to the sensor of the previous part, which, when a cube is placed in front of it, reduces the speed of the conveyor belt to zero so that the position of the cube remains constant. In the figure below, a cube is placed in front of the sensors, in which case the sensors are in flashing mode, and in the figure below, the sensor shown in red

corresponds to the Part Producer and the sensor shown in green. It is related to the conveyor belt.

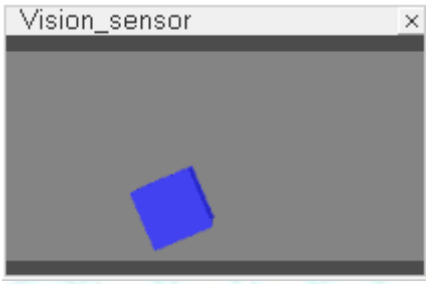


Figure 5. Sample image seen by Vision Sensor

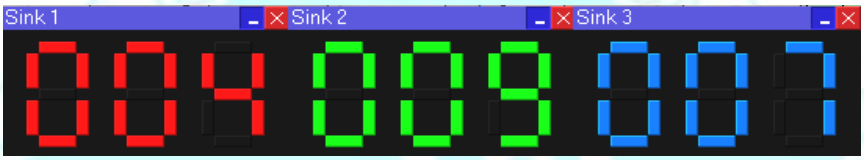


Figure 6. Counters

3. Vision Sensor: It is a sensor that is placed in a place where the cubes are visible after the conveyor stops, and its image is saved by MALTAB, and then we recognize its coordinates and color by OpenCV. The quality of this sensor can be changed and in this project we considered 256x128 quality. An example of the image that this sensor sees, which can be seen in the software environment, is given below.

4. Deletion Box: There are boxes for separating cubes. First, the color of the cube must be recognized and then the robot is directed to the corresponding box. There are three boxes for green, red and blue colors, and the color of each box is the same. There is a proximity sensor in each box, which adds a number to the counter by dropping the cube. In the figure below, you can see an example of the counter, which after some time of simulating and separating the cubes, numbers like the figure below are displayed.

5. UR10: The robot used in this project, whose simulation example is available in the software. This robot has 6 axes and its gripper can also be changed. The real sample of this robot is also available, its weight is 33.5 kg and its payload is 10 kg, its range is 1300 mm and its accuracy is 0.1 mm. to be

Figure 7. real UR10

Figure 8. Image of Baxter Vacuum Cup gripper and an example of this gripper in reality

6. Baxter Vacuum Cup: It is a type of gripper that can hold the object by using air suction for the robot to lift it. The advantage of using this gripper is that the rotation of the object around the axis perpendicular to its surface is not important. If the desired object has a suitable surface, the object can be lifted by sticking the gripper. In the picture below, a gripper is used and an example of it can be seen in reality. Note: This used gripper can be changed to other grippers as well, but changes must be made in the program code according to the used gripper; According to the need, other grippers can also be used, because the object that is to be moved may not be moved by the suction cup, for example, it has fluff or... for this reason, for example, another version of the project with a gripper. There is another one called RG2, whose image is shown below.

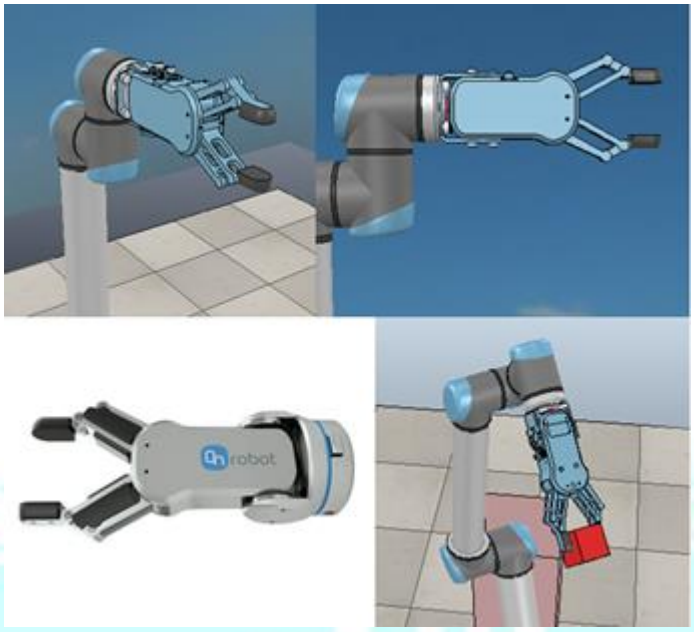
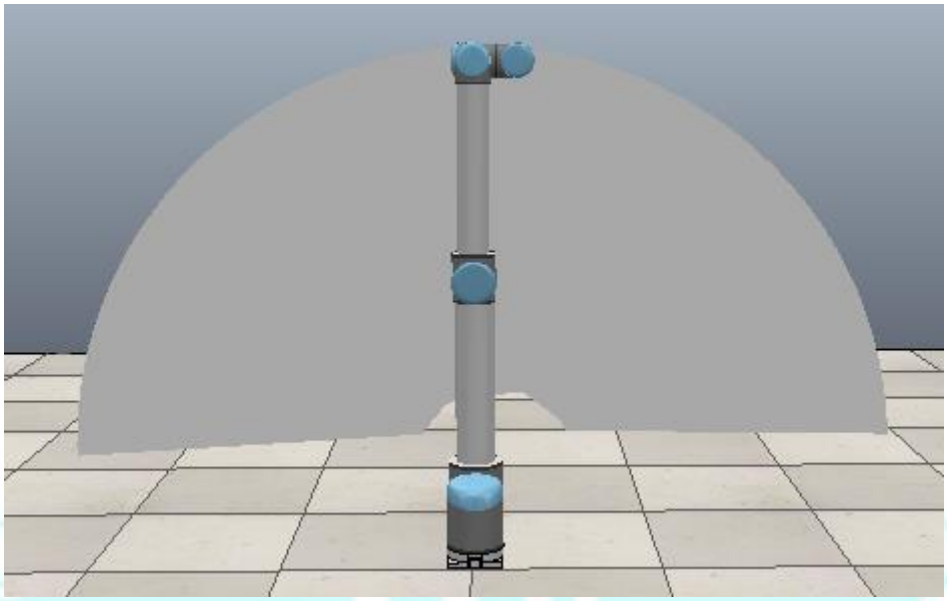


Figure 9. Image of RG2 gripper in the simulation environment and its real example

To use this gripper, the challenge we have is to be able to detect the amount of rotation of the object so that the gripper can pick up the object correctly. has less; Of course, we note that adding this feature was not one of the goals of this project, and just to show the possibility of using the robot in different applications and according to the needs, with minimal changes, we prepared a version of the project with this gripper

2.2 Robot Workspace:

One of the important parts to control the robot is to know the working space of the robot. In order to perform the Pick and Place operation, we need to know what positions the gripper robot can go to, so that we can properly give it the necessary commands. For this purpose, we examined the robot in two modes, only on the ground and in the work environment. The working space of the robot is obtained using simulation in the CoppeliaSim and MATLAB environment, which we discuss in the discussion section in detail about how to obtain it.



Figre 10. The robot and its workspace in the simulation environment

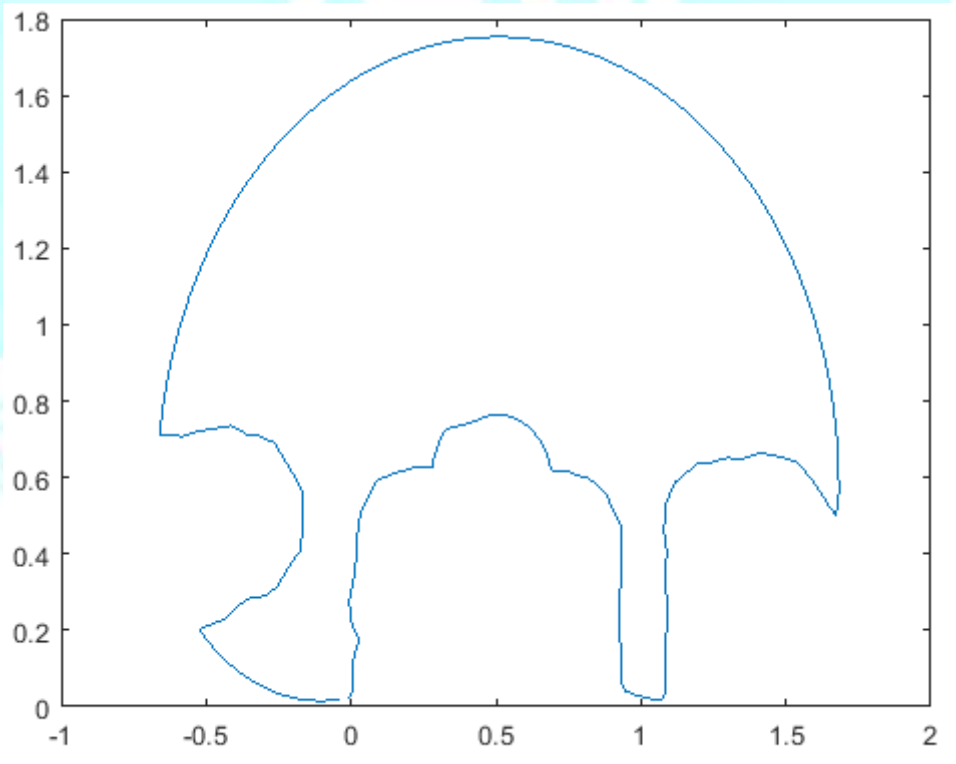


Figure 11. Robot workspace in Pick & Place operation environment

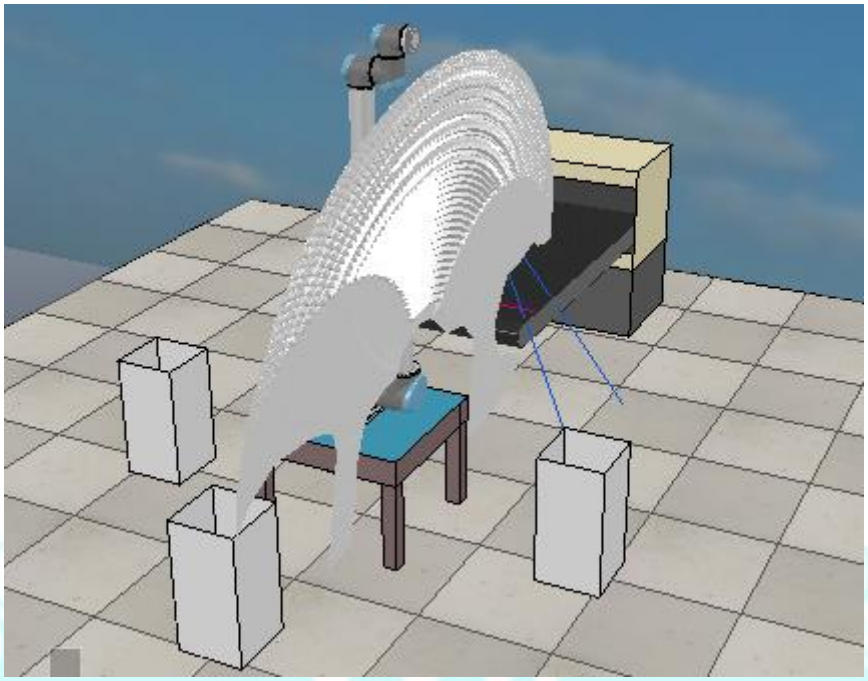


Figure 12. The image of the robot in its operating environment and workspace

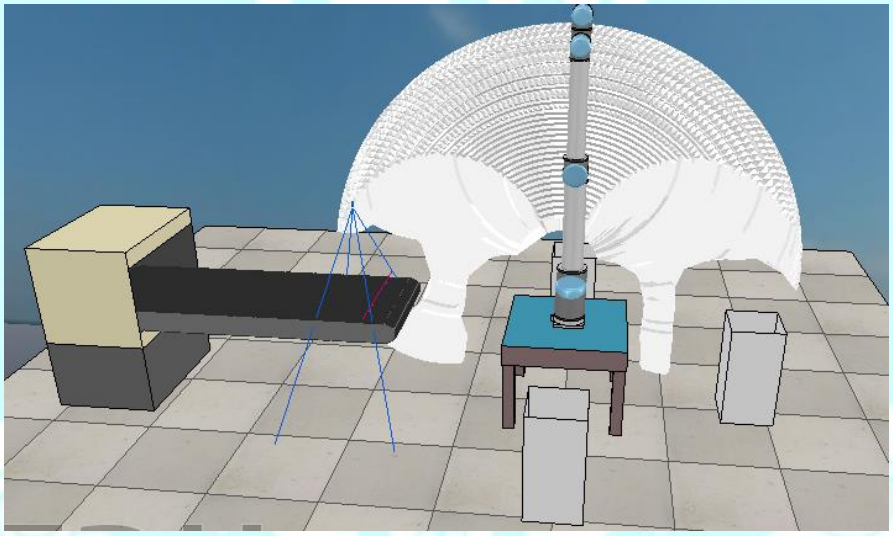


Figure 13. The image of the robot in its operating environment and workspace

In the above images, the white parts are the robot's workspace, and the robot's gripper can be placed in these places without the robot having unwanted contact with other objects.

2.3 Image Proccessing:

Using the moments of a binary image, we can get its central coordinates. The moment of M_10 for one binary matrix is equivalent to the sum of coordinates in the x direction of all non-zero domains. Similarly, the moment of M_01 for a binary matrix is equivalent to the sum of coordinates in the y direction of all non-zero terms; So by dividing these moments by the number of points, we can get the average coordinates of all points. So it is concluded that these obtained coordinates are the center of gravity of the image, which will be obtained from the average coordinates of all its points, which is obtained from the following equation

$$\bar{x} = \frac{M_{10}}{M_{00}} \; ; \bar{y} = \frac{M_{01}}{M_{00}}$$

Moments up to order 3 are accessible through cv:Moments. Thus, it is necessary to first apply a series of filters on the original photo and convert it into a binary photo so that we can extract its moments and thus have access to its other characteristics such as the center of gravity.

2.3.2 Color Space and Color Detection:

RGB color space is not suitable for color recognition because it does not consider parameters such as color intensity and brightness. Although in the simulation environment, because the brightness level is constant, it can obtain the color range in the RGB color space and finally recognize the object, but in reality, due to the constant brightness level and also the color intensity not being the same for all the cubes. Hey, it's hard to find this range. For this reason, we use the HSV color space, which has both the mentioned parameters, and it is easier to find the color range for it. Color space conversion is possible using cv.cvtColor.

Now, by using the cv.inRange method and giving it specified intervals, we have a binary matrix in which pixels with the desired color have a value of 1 and the rest of the pixels have a value of 0. In this way, the object will be recognized with the desired color.

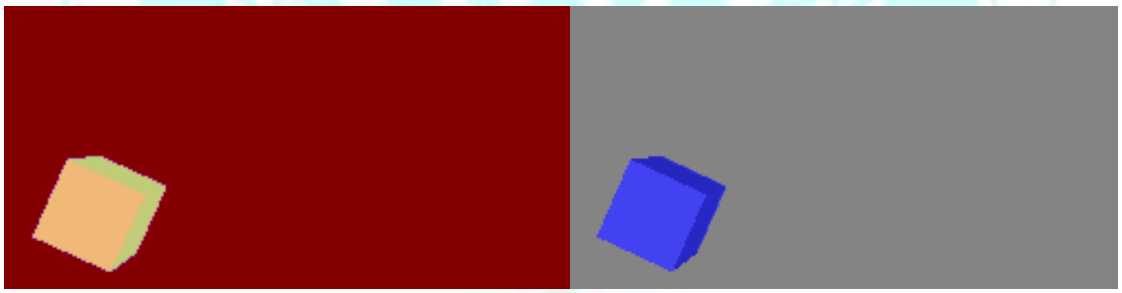


Figure 14. Sample photo taken by the camera and its output in HSV color space

Figure 15. Binary filtered photo with blue color intervals

2.3.3 Filters:

Another thing that can be done to improve the binary image is the use of Morphological Transformations methods. These transformations are simple operators that are applied to the 2D binary image and have a kernel that defines the performance. Although in the simulation environment where noise is not taken into account, this method does not work much, but in reality we need to use this method to eliminate a series of noises in the image. The two basic operators of this method are Dilation and Erosion. In this section, we use the Dilation operator, which, although it does not have a special effect on the output, but leaving it to process the image according to reality, is not without harm. This operator applies a kernel of a certain size to all the pixels of the image, in other words, the kernel is convoluted on the image. If there is even one pixel with a value of one in the neighborhood of a particular pixel (which is the size of the kernel size), that particular pixel will have a value of one. Therefore, when we do not have some corners or even points in the image due to noise, with this method we can eliminate these noises and have a uniform photo. In this way, we have a binary image in which the pixels have a value of 1 in the area where the colored object is located, and the pixels have a value of 0 in the rest of the areas.

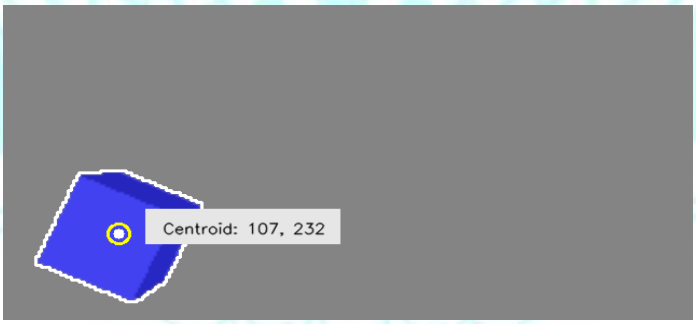


Figure 16. The specified contour and center of gravity of the object

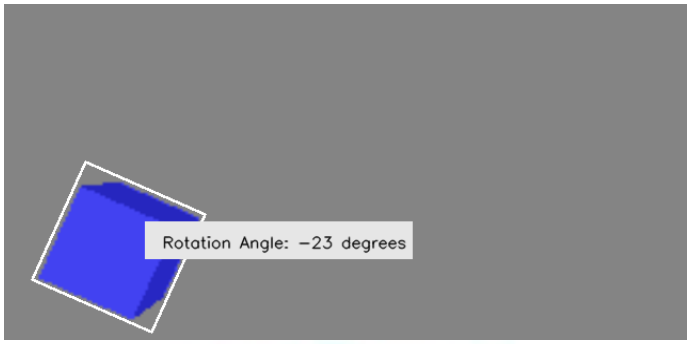


Figure 17. Rectangle with minimum area including object and object rotation detection

2.3.4 Contours

Several cubes of the same color may be placed in one image. Contours are the border coordinates of connected points that allow us to recognize shapes in a binary image, and for this we use the cv.findContours method. Using the explanations of the section related to moments, we first consider the contours that have an acceptable area that determines that this contour is the desired cube. We get the center of gravity of each of the detected contours so that we can have the coordinates of the center. Therefore, having the coordinates of these contours, we choose the nearest cube as the recognized cube. In this way, we were able to get the coordinates of the closest cube with the specified color.

2.3.5 Detection of the rotation angle of the object:

Using the cv.minAeraRect method, we can easily find a rectangle with minimum area that includes the set of points. Since this rectangle has the minimum area, it can have a rotation, and by detecting this rotation, we can guess the rotation of the cubes. Using the cv.boxPoints method, we will access the coordinates of the four corners of the rectangle. By having the coordinates of the vertices of this rectangle, the angle of rotation can be obtained through trigonometric applications.

Table 1. UR 10 DH parameters

i	a_i	b_i	α_i	θ_i
1	0	0.1273	$\frac{\pi}{2}$	θ_1
2	0.612	0	0	θ_2
3	0.5723	0	0	θ_3
4	0	0.163941	$\frac{\pi}{2}$	θ_4
5	0	0.1157	$-\frac{\pi}{2}$	θ_5
6	0	0.0922	0	θ_6

2.4 Robot Control:

To move the final actuator and gripper to the location obtained from the image processing, we need a set of positions (angles) for the joints of the UR10 robot, then the set of angles obtained by MATLAB is sent to the internal PID controller of CoppeliaSim software so that the angles of the joints reach the desired value. we reach (Of course, as mentioned, only the P controller was used in this project in particular.)

2.4.1 Forward kinematics

Since to solve the inverse kinematics of the robot, it depends on the direct kinematics, we will discuss it first: for direct kinematics calculations, we need the DH parameters of the robot, which according to Figure 18 can be obtained in the form of what is given in Table 1.

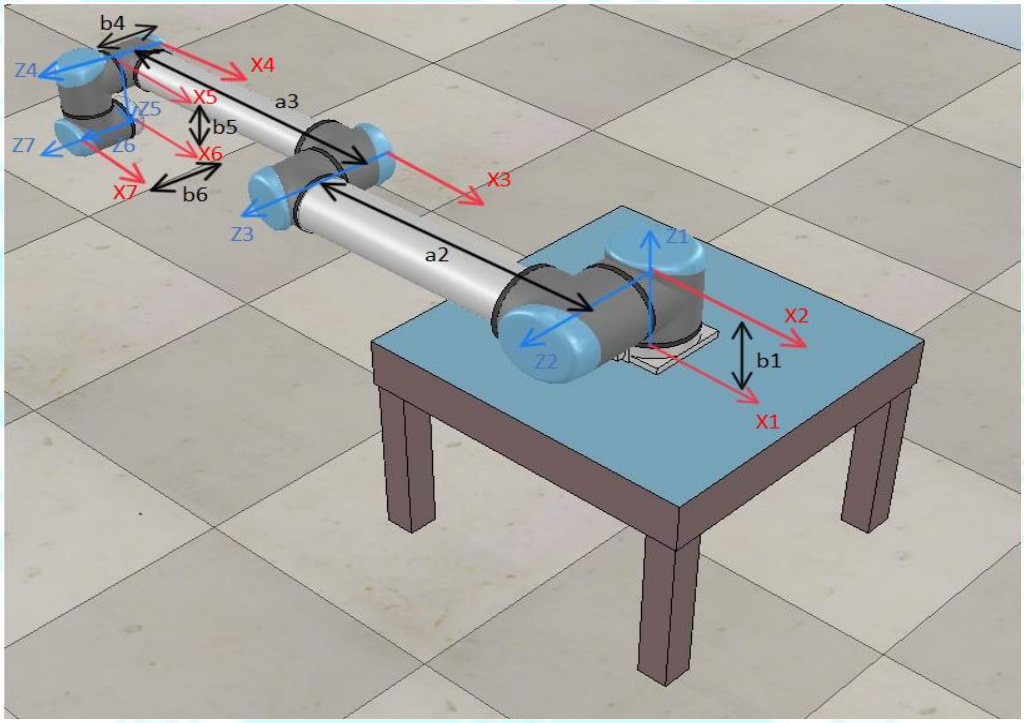


Figure 18. Coordinate devices related to each axis and their distances to find DH parameters

According to the above table and what we learned in the lesson, the direct kinematic equations will be as follows, and the position vector of the final operator is obtained as follows:

$$\overrightarrow{\boldsymbol{p_{EE}}} = \begin{bmatrix} x_{EE} \\ y_{EE} \\ z_{EE} \end{bmatrix}$$

Where:

$$x_{EE} = 0.163941\,sin(\theta_1) - 0.612\,cos(\theta_2)\,cos(\theta_1) + 0.0922\,cos(\theta_5)\,sin(\theta_1) + 0.5723\,cos(\theta_1)[sin(\theta_2)\,sin(\theta_3) - cos(\theta_2)\,cos(\theta_3)] - 0.0922\,sin(\theta_5)\,cos(\theta_1)\,cos(\theta_2 + \theta_3 + \theta_4) + 0.1157\,cos(\theta_1)\,[cos(\theta_2 + \theta_3)\,sin(\theta_4) + sin(\theta_2 + \theta_3)\,cos(\theta_4)]$$

$$y_{EE} = -0.163941\,cos(\theta_1) + 0.5723\,sin(\theta_1)\,sin(\theta_2)\,sin(\theta_3) - 0.0922\,cos(\theta_1)\,cos(\theta_5) - 0.612\,cos(\theta_2)\,sin(\theta_1) - 0.0922\,sin(\theta_1)\,sin(\theta_5)\,cos(\theta_2 + \theta_3 + \theta_4) + 0.1157\,cos(\theta_2 + \theta_3)\,sin(\theta_4)\,sin(\theta_1) + 0.1157\,sin(\theta_1)sin(\theta_2 + \theta_3)\,cos(\theta_4) - 0.5723\,sin(\theta_1)\,cos(\theta_2)\,cos(\theta_3)$$

$$z_{EE} = 0.1273 - 0.612\,sin(\theta_2) + 0.1157sin(\theta_2 + \theta_3)\,sin(\theta_4) - 0.5723sin(\theta_2 + \theta_3) - 0.0922sin(\theta_5)[cos(\theta_2 + \theta_3)\,sin(\theta_4) + sin(\theta_2 + \theta_3)\,cos(\theta_4)] - 0.1157\,cos(\theta_4)cos(\theta_2 + \theta_3)$$

Also, the end effector rotation is obtained as follows:

$$\boldsymbol{Q_{EE}} = \begin{bmatrix} cos(\theta_6)\,\sigma_3 - cos(\theta_1)\,sin(\theta_6)\,\sigma_1 & -sin(\theta_6)\,\sigma_3 - cos(\theta_1)\,cos(\theta_6)\,\sigma_1 & -cos(\theta_1)\,sin(\theta_5)\,\sigma_4 + cos(\theta_5)\,sin(\theta_1) \\ -cos(\theta_6)\,\sigma_2 - sin(\theta_1)\,sin(\theta_6)\,\sigma_1 & sin(\theta_6)\,\sigma_2 - sin(\theta_1)\,cos(\theta_6)\,\sigma_1 & -sin(\theta_1)\,sin(\theta_5)\,\sigma_4 - cos(\theta_5)\,cos(\theta_1) \\ sin(\theta_6)\,\sigma_4 + cos(\theta_5)\,cos(\theta_6)\,\sigma_1 & cos(\theta_6)\,\sigma_4 - cos(\theta_5)\,sin(\theta_6)\,\sigma_1 & -sin(\theta_1)\,\sigma_1 \end{bmatrix}$$

Where:

$$\sigma_1 = sin(\theta_2 + \theta_3 + \theta_4)$$

$$\sigma_2 = cos(\theta_1)\,sin(\theta_5) - cos(\theta_2 + \theta_3 + \theta_4)\,cos(\theta_5)\,sin(\theta_1)$$

$$\sigma_3 = sin(\theta_1)\,sin(\theta_5) + cos(\theta_2 + \theta_3 + \theta_4)\,cos(\theta_5)\,cos(\theta_1)$$

$$\sigma_4 = cos(\theta_2 + \theta_3 + \theta_4)$$

2.4.2 Inverse Kinematics

To solve the inverse kinematics considering that the robot has 6 degrees of freedom and as can be seen from the direct kinematics equations, solving its inverse kinematics will be complex; In the article [7], he solved the inverse kinematic equations geometrically and analyzed and analyzed its different solutions (in some situations there are 8 solutions for it), but considering that there are different states and conditions that should be should be taken into account and because the desired solution can be easily reached with numerical methods,

and even for some more complex robots, it is practically not possible to solve the inverse kinematics analytically, so the decision to use numerical methods was made. To solve it, we took; In this project, we used the ikunc function of the Robotics toolbox for this purpose, which also uses the MATLAB software optimization toolbox and the fminunc function to solve the inverse kinematics problem, which is a multivariable problem, as follows. that an objective (error) function is defined as follows:

$$Error = sumsqr\ ((\ T^{-1} \times forward\_kinematic(q)\ -\ I) \times \Omega)$$

which is the difference of the direct kinematic answer (resulting from q, the angles obtained so far) and the final executive form. (T is the homogenized matrix indicating the rotational position and our desired spatial position for the final performer, which is given as an input to the ikunc function; also, Ω is a constant coefficient.) Then, using the MATLAB optimization toolbox and fminunc function, iteratively and with the Quasi-Newton algorithm, we find the minimum point of the mentioned error function, and in this way, numerically, the inverse kinematics of the robot for any desired position in the workspace The robot is obtained, and with this answer, we can place the executors in the workspace of the robot in any desired position. We know that in general the inverse kinematics response for a 6 degrees of freedom robot is not unique, here also the final response is determined based on the initial guess, which is (q0) $\vec{}$=0 $\vec{}$ by default, it can also be as a parameter to the ikunc function; At first, we considered this parameter to be the same as the default value of zero, but due to the problems that are described in more detail in the next section, sometimes it did not converge to the correct answer and the control of the robot faced problems. Finally, by changing the default value of the initial guess, the mentioned problem was solved and the UR10 robot could correctly perform the Pick and Place operation that was explained earlier.

2.5 Comparison of RG2 gripper and Baxter Vacuum Cup

In tables 2, 3, and 4, the percentage of success is equivalent to the ratio of the number of cubes dropped into the boxes to the total number of cubes removed from the conveyor belt; In each of these tests, the simulation was run for 5 minutes and during this time the robot performed the Pick & Place operation, in all tests the robot with a suction gripper (Baxter Vacuum Cup) was completely without The problem was that he would remove the cubes and transfer them to the corresponding box, but in some rare cases, due to the relatively low accuracy in detecting the amount of rotation of the object, the RG2 gripper did not fit correctly in the gripper and separated from the gripper when removing or moving. and fell on the ground or even the robot could not lift the cube from the conveyor. Another point that can be observed is that the speed of the suction gripper is about 3 times the speed of the RG2 gripper on average, because when using the RG2, the cubes are removed in two stages, the first stage of the gripper is above the cube and at a distance from It is placed and by adjusting the angle in the second step, it removes the cube and also opens it

And closing it also takes time, and in general, Baxter Vacuum Cup is a more suitable option both in terms of accuracy and speed, except in special applications that cannot be used due to the type of object that is supposed to be moved.

Table 2. Baxter Vacuum Cup simulation results

Experiment	Speed (cube per minute)	Succuss rate
1	3	100
2	3	100
3	3.2	100
4	3.2	100
5	3.2	100

Table 3. RG2 GRIPPER simulation results

Experiment	Speed (cube per minute)	Succuss rate
1	0.8	100
2	1.2	100
3	1.4	100
4	0.6	75
5	1.2	85

Table 4. Summary and comparison of Tables 2 and 3

Gripper type	Speed (cube per minute)	Succuss rate
Baxter Vacuum Cup	3.12	100
RG2 Gripper	1.04	92

2.6 345 Path Planning:

To change the position from the initial state to the final state, the robot does not travel the straight line between these two points and may reach the final point through a curved path,

which is not necessarily the desired path. If we have a dynamic model of the robot, we can control the robot in such a way that it moves from the initial point to the final point with a smoother speed. But here, the only goal is that the robot does not go to the final point from any path and also moves in the workspace so that it does not get out of control because the robot is controlled by the position mode and the speed and acceleration are not controlled. We assume that we have a dynamic model, and in this case, the position of the joints is such that it is equal to θ_I at the initial time and equal to θ_F at the end, and the path between these two is determined by a polynomial with degree 5. . Therefore, the mentioned polynomial can be obtained by using the following relations:

$$\boxed{s(\tau) = 6\tau^5 - 15\tau^4 + 10\tau^3}$$

$$0 \le s \le 1, \qquad 0 \le \tau = \frac{t}{T} \le 1$$

The angle, angular velocity and angular acceleration at any moment can be calculated according to the mentioned polynomial as follows:

$$\begin{cases} \theta(t) = \theta_I + (\theta_F - \theta_I)s(\tau) \\ \dot{\theta}(t) = (\theta_F - \theta_I)\frac{1}{T}s'(\tau) \\ \ddot{\theta}(t) = (\theta_F - \theta_I)\frac{1}{T^2}s''(\tau) \end{cases}$$

3 Discussion

3.1 Project stages and simulation:

After preparing the simulation environment and making the related settings (such as the settings related to the production frequency of parts, etc., as described in the previous section), we used the RemoteApi provided by the CoppeliaSim software to establish a connection between MATLAB and CoppeliaSim. Also, in order to call the Python functions related to OpenCV image processing from MATLAB software, it was necessary to enter the corresponding module in MATLAB; In Figure 19, there is a flowchart regarding the relationship between these three environments.

After establishing a connection between MATLAB and two other software, it is necessary to obtain the Object Handles to control the objects in the simulation environment (such as commanding the Joints or taking an image from the Vision Sensor or...) finally according to the DH parameters We define the robot that we got, the UR10 robot in MATLAB with the Robotics toolbox, and finally the main simulation loop begins; This loop continues as long as the simulation is in progress, and this is how the robot returns to its initial state (joint angles are all zero) and if the proximity sensor of the conveyor detects that a cube is in front of it, the

image It is saved by MATLAB and then the Python function is called and this function also returns the spatial and temporal position of the cube closest to the robot (if for any reason there are several cubes in the Vision Sensor's viewing angle) to MATLAB; These coordinates are in the Vision Sensor frame, and by converting the coordinates, we will have the coordinates of the mentioned cube in the reference frame of the UR10 robot.

Having the coordinates of the cube and with the help of the Robotics toolbox, we solve the inverse kinematics problem for the coordinates related to the cube to get the desired angles of the joints, finally, these angles are given to the internal PID controller of CoppeliaSim to adjust the angles of the joints to the desired value. arrive and as a result the final operator reaches the place of the cube, finally the cube is removed by the UR10 robot and in this way the Pick operation is performed; Similarly, according to its color, each cube is moved to the coordinates of the box with the same color, and thus the Place operation is performed, and the UR10 robot returns to its initial position and waits for the next cube to perform the operation. Pick & Place remains. (The main simulation loop continues in the same way.)

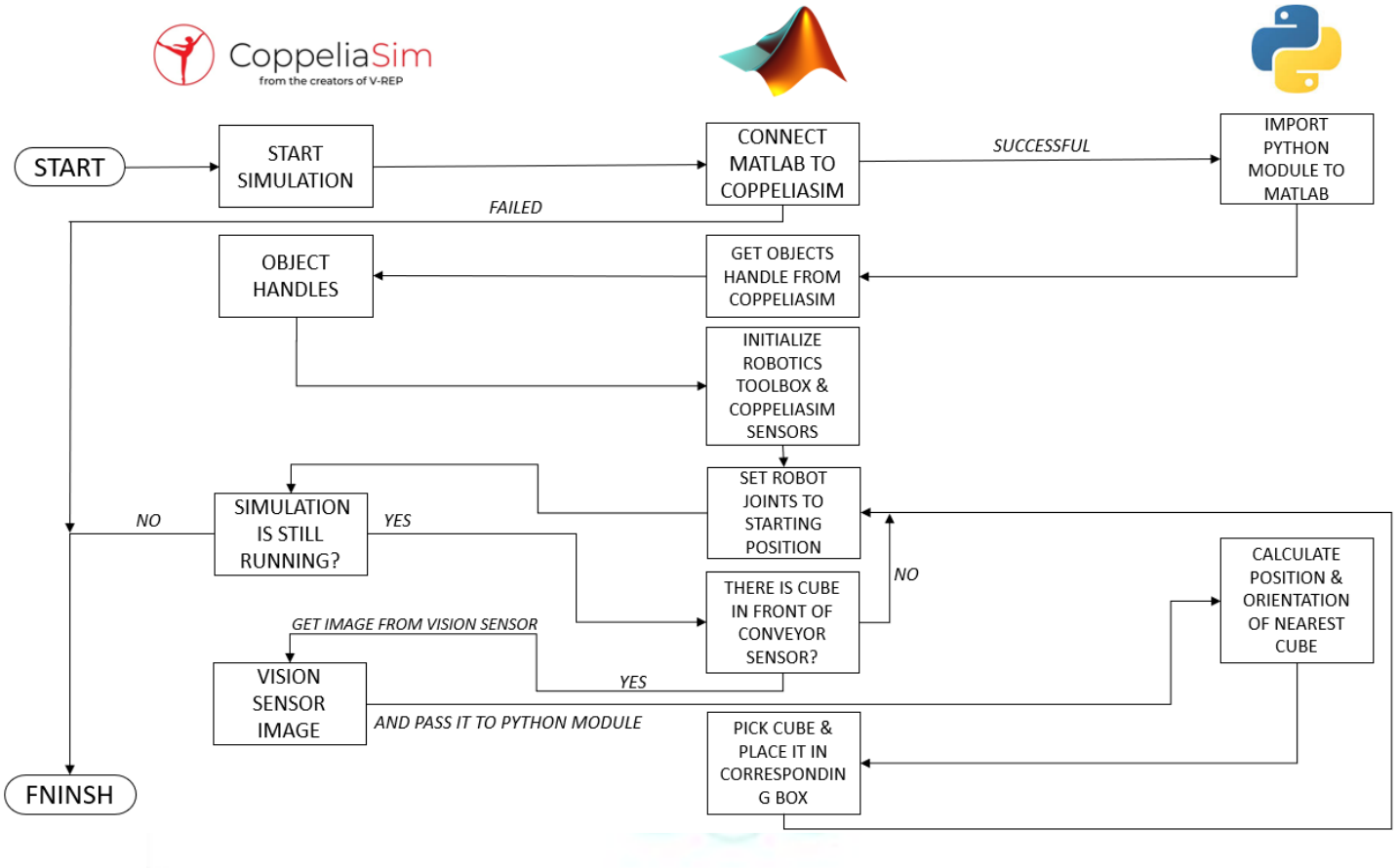


Figure 19. Flowchart of the simulation

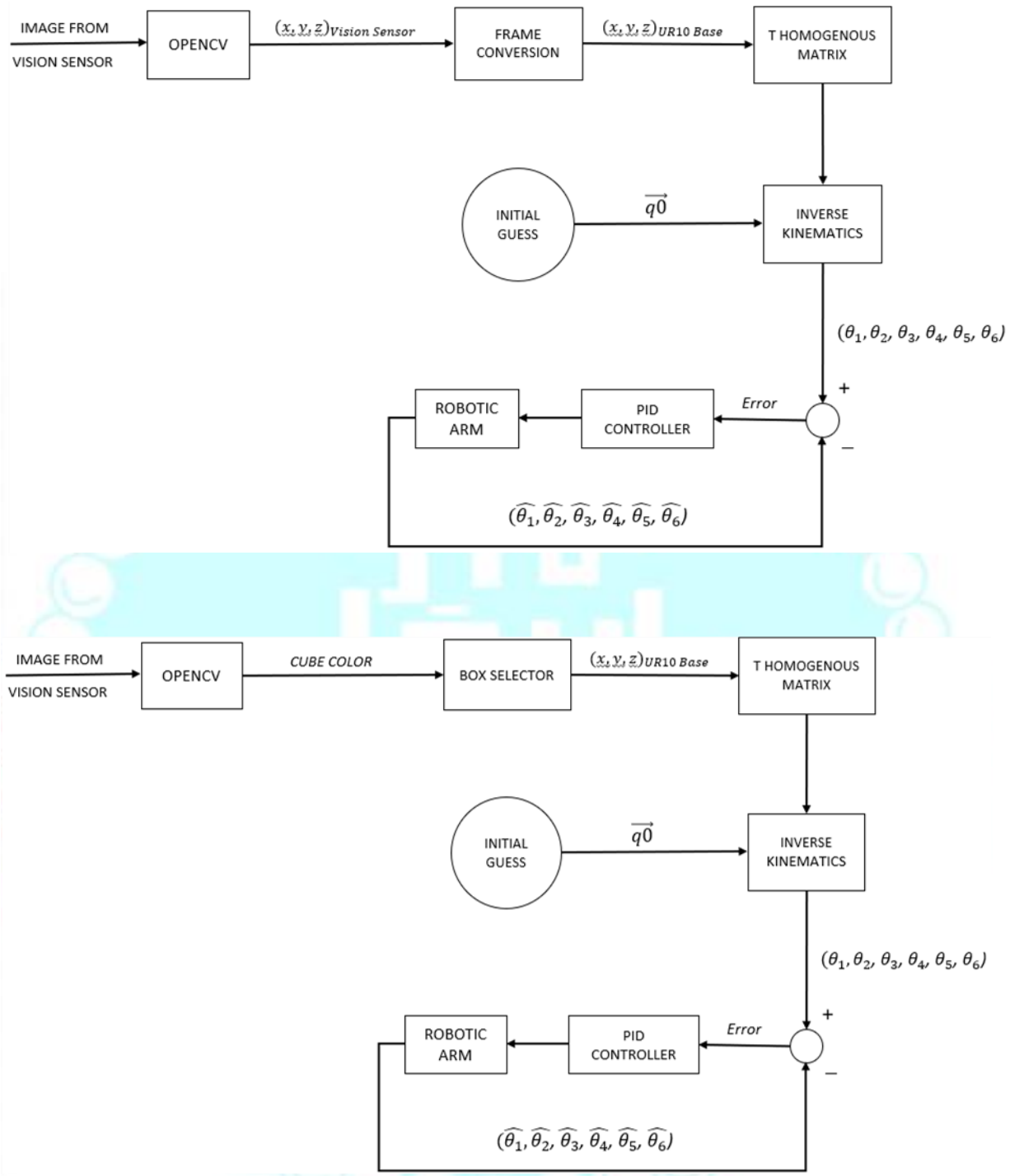


Figure 20. Flowchart related to image processing and joint control in Pick and Place operations

3.2 Challenges

The first challenge we faced was that despite the many features this software has and provides a powerful environment for simulation, its learning curve is a bit complicated; For example, it sometimes has bugs that slow down the progress of the project because, for example, it was not possible to change some settings through the graphical interface of the program, and it was necessary to change the relevant Lua scripts, and on the other hand, we did not want the main parts of the project. depends on the simulation environment, in other words, we did not want the parts related to image processing and robot control to be programmed with Lua script;

These cases initially slowed down the progress of the project, which we did not foresee before the project would be a challenge.

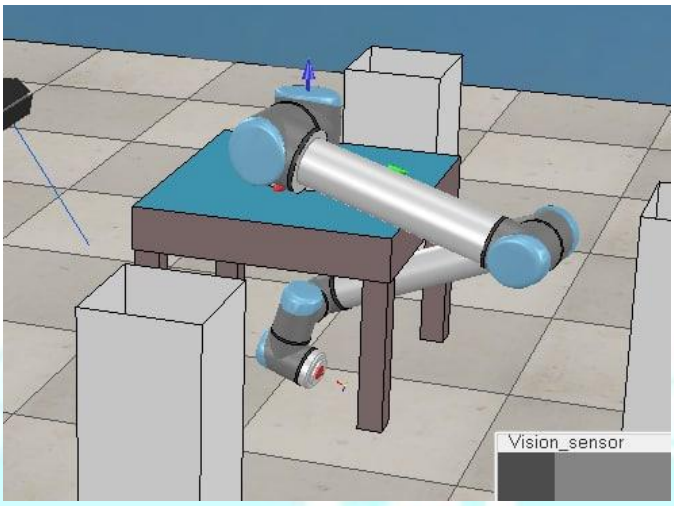


Figure 21. It can be seen that the robot is out of control and its final operator is placed in an undesirable place.

Another challenge that was raised in the previous section was that the numerical solution of inverse kinematics sometimes did not converge to the correct answer (the fminunc function also gave a warning that it did not converge to the correct answer and probably in a the local minimum is located) and it even caused the robot to leave its main working area (area without singularity) and generally the robot went out of control, an example of which is shown in Figure 21.

To solve the mentioned challenge, considering that the location of the boxes as well as the location of the conveyor belt is predetermined and fixed, we realized that this problem can be solved by using a more appropriate initial guess (other than the default value of zero). and by choosing a suitable initial guess, our algorithm converged quickly and with high accuracy (the error function reached its minimum value) and the robot in our desired working area could correctly pick and Place, which was explained in detail earlier.

After solving the above problem, during the test, we realized that the robot would go out of control again and sometimes it would oscillate. It still made the robot uncontrollable at times; At first, we suspected the same problem as before, that maybe the problem is in the inverse kinematics solution part, but finally we realized that the problem is the weight of the cubes because we did not pay enough attention to their weight during the preparation of the simulation, and because Their weight was high and we controlled the robot by kinematic method and by giving the position, it caused the robot to go out of control and become unstable and oscillate in some situations, which was solved by reducing the density and as a result the weight of the cubes. It was also resolved; This problem may seem simple, but because the project was in a simulation environment, it was challenging to find out its cause.

Another control method that was mentioned was the dynamic control method by using the torque control of the motors, in this way we could determine the torque and the final executive force, and in addition, we could use routing so that the robot reaches the object from the desired path. It will reach the desired point with a smoother and smoother acceleration and speed and less impact on the robot and its joints. But getting the dynamic model of the robot was more

complicated and time-consuming, so we decided to use the same kinematic control; But in another version of the project, we implemented 345 routing so that if we want to control the motors in torque mode, we can have a smoother final executive speed and torque. Knowledge of the dynamic characteristics of the robot and routing could somehow eliminate the problems caused by the weight of the cubes as well as out of control. In this version, the path between the initial and final points is divided into several points and follows a 5th degree polynomial. In this way, the robot must be placed in one of these divided points in each time period. The problem of this method was that the process of placing the robot's joints in each of these positions between the first and last point takes a lot of time, and practically, in order to be able to see the movement of the robot, we had to reduce the number of points, which also caused It could be that the movement of the robot is not continuous and therefore in the end we witness a broken and broken movement; We also tried to control the robot by controlling the speed of the joints using the speed obtained from routing 345, but the command that we could use to get feedback from the speed of the joints was not applicable in the RemoteApi provided for MATLAB and with We would encounter errors, and for this reason, this method could not be used, and finally we found the version without routing to control the position in the simulation, better and smoother.

Another challenge that was a problem for the continuation of the work was the pyramidal view of the camera. In fact, the two-dimensional image that the camera gave us did not take into account the height of the object, and because our object had a height, the center that the camera detected was a little further away from the real center. For a better understanding of this issue, Figure 22 is given, where it is clear that the center of the object detected by the camera is distance number 2 with respect to an interface with the main coordinates (which is the floor of the cube), but the main distance that the camera must detect is Distance is number 1. This difference is caused by the two-dimensionality of the processed image, which does not include the height.

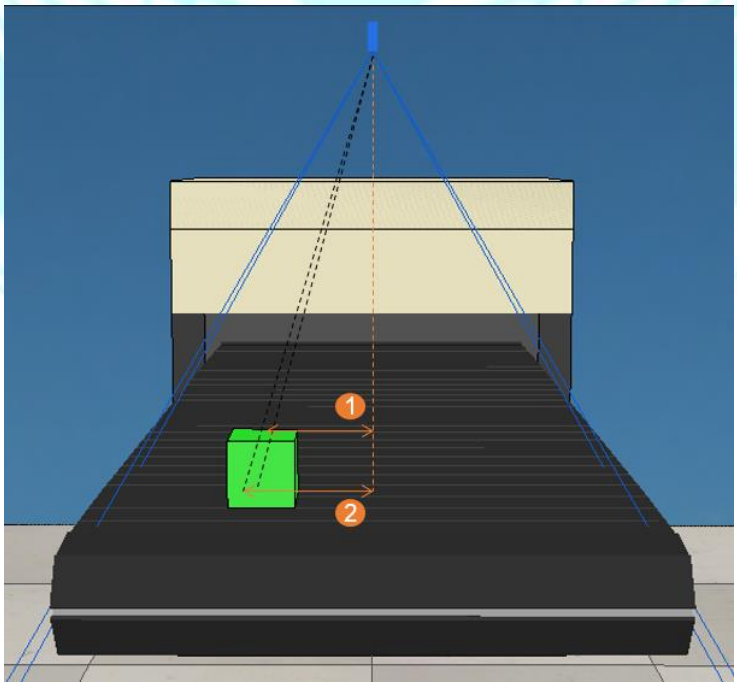


Figure 22. The distance from the center detected by the camera and the actual distance

Also, this caused us to think that the robot has very low accuracy and cannot correctly go to the coordinates that were given to it; Therefore, it created a big challenge for us. To solve this problem, it was only necessary to scale the obtained coordinates with respect to the height of the object and thus have the distance from the real center. This caused the robot to go to the coordinates of the object with much better accuracy.

3.3 How to get the robot workspace:

By writing a script in the section related to the UR10 robot in CoppeliaSim, we put joints 2, 3, 4 and 6, which are related to determining the location of the final executive on a page, in all possible states, in this way, a function called jointSweep We wrote that by taking the number of the joints and their initial position and determining the number of steps, it outputs the positions in which each joint should be placed in each step. The meaning of the number of steps is that, for example, if joint number two is supposed to rotate 360 degrees, and the number of steps for this joint is equal to 100, in each step it rotates 3.6 degrees compared to the previous step. It is clear that the more these steps are, the more points will be covered and the working space obtained will be more accurate. Then, using the output of this function, we change the position of the relevant joints in each step and check whether a part of the robot collides with something or not with the help of the sim.checkCollision function. If there is no collision in this position, we save this point (the current coordinates of the final executive location) as a part of the workspace, and if there is a collision, the robot moves to the next position. And these steps are carried out until all the specified situations have been investigated. Finally, these points in a file with the extension xyz. We save that the first 3 columns are x, y, z of the robot and the next 3 columns represent the angle of the final executive. Next, with the help of MATLAB, draw the workspace of the robot and also the stl file. We produce so that the workspace can be observed in the simulation environment.

Description of the MATLAB code: We load the file that we saved in CoppeliaSim in MATLAB, then with the help of the alphaShape and boundaryFacets commands, we obtain the contour of the workspace and then draw it, then to The help of stlwrite command this workspace in a stl format file. save and import it in the simulation environment.

3.4 Conclusion

In this project, with the help of toolboxes, libraries and software such as MATLAB, OpenCV and CoppeliaSim, we simulated a UR10 robot for Pick and Place application to sort and separate colored blocks based on image processing in the production line, and We managed to control the UR10 robot to perform this operation with high speed and accuracy with the help of its kinematic model; During this project, we controlled the robot by using the knowledge we learned during the semester (such as the DH table of a robot and solving direct and inverse kinematics problems, etc.) and the practical application of these learnings during this project. We observed, in addition to that, we learned a lot in fields such as simulation in the CoppeliaSim environment and how it is related to MATLAB, as well as the relationship between MATLAB and Python and the OpenCV image processing library, etc., and gained a lot of experience in these fields.

One of the most important achievements of this project for us was to face the challenges of a robotics project, that we had to strengthen our software and computer skills and programming, along with that, we had to learn how We were familiar with the mechanism and

mechanics of the robot, as well as the topic of robot control, which is one of the most important topics and we had to face its challenges.

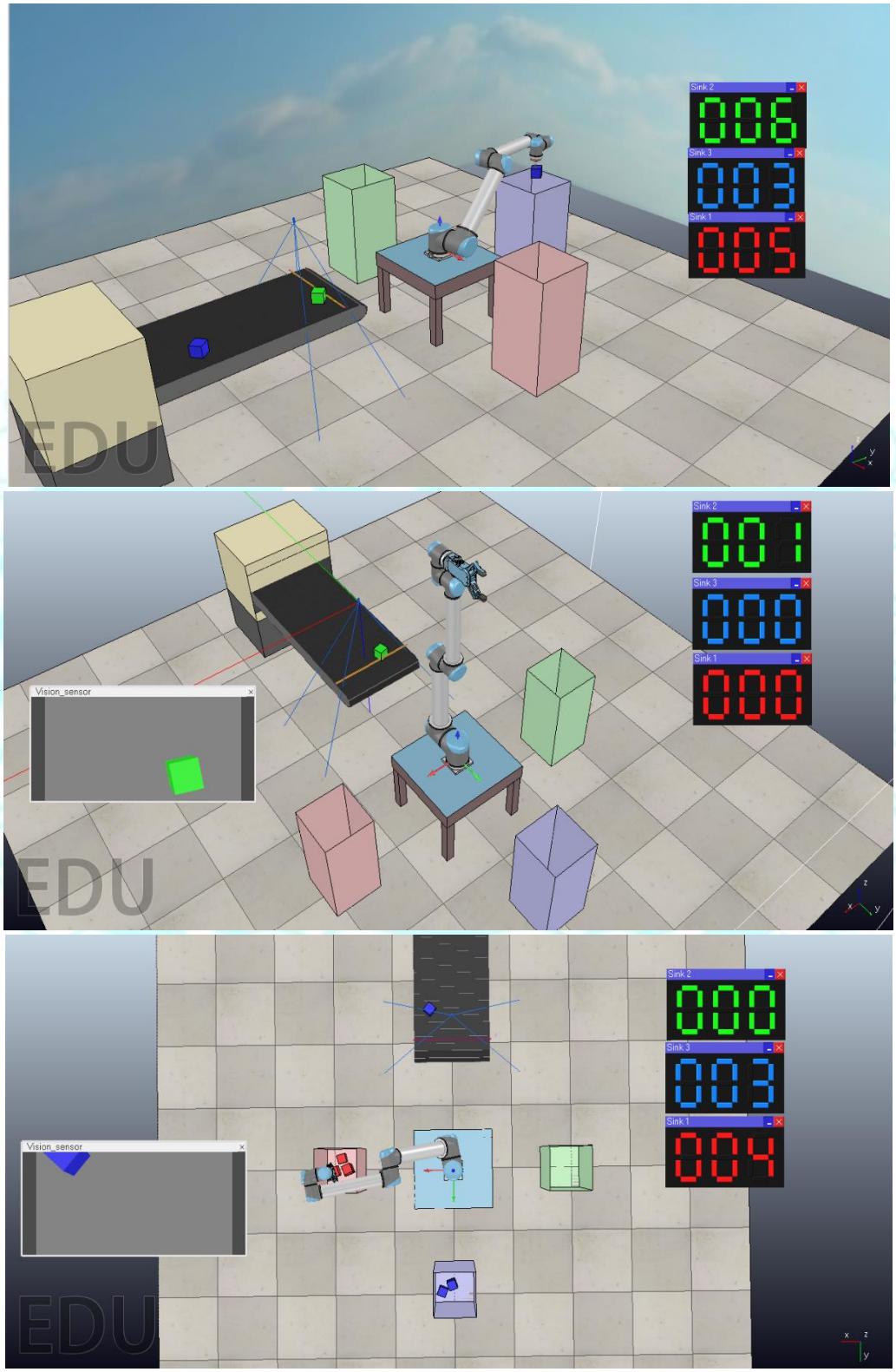


Figure 23. Some pictures of the robot in the mode of Pick & Place operation in the simulation environment

Refrences

1. https://opencv.org/
2. https://www.coppeliarobotics.com/helpFiles
3. Remote API functions (MATLAB) (coppeliarobotics.com)
4. J.Angeles, Fundamentals of Robotic Mechanical Systems, Fourth Edition, New York: Springer, 2014, pp. 255-265.
5. https://petercorke.com/toolboxes/robotics-toolbox/
6. Lecture Notes
7. K. P. Hawkins. Analytic Inverse Kinematics for the Universal Robots UR5/UR10 Arms, 2013.
8. https://www.universal-robots.com/case-stories/allied-moulded-products-inc/
9. https://www3.panasonic.biz/ac/e/fasys/appli_search/data/images/apl0331.png

Appendix

Moments:

Moments are measurable quantities that describe a function and record its important characteristics. They are also widely used in statistics to describe the probability density function. In simple words, moments of a photo are a set of statistical parameters that determine the density of pixels as well as their intensity. From a mathematical point of view, the image moment M_ij, which is of order (i,j), for a gray image, is calculated as follows:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

$$M_{00} = \sum_x \sum_y I(x,y)$$

For a binary image whose color intensity is 0 or 1, the zero moment represents the number of all non-zero areas, which is equivalent to the area of the detected object because the value of I(x,y) in A binary matrix is either zero or one, which will give us the number of rows with a value of one. For a gray image, the zero moment is equal to the sum of the color intensity of the pixels.

Python functions used in image processing:

```
def get_contour_angle(cnt):
    rect = cv2.minAreaRect(cnt)
    angle = rect[-1]
    width, height = rect[1][0], rect[1][1]
    ratio_size = float(width) / float(height)
    if 1.25 > ratio_size > 0.75:
        if angle < -45:
```

```
            angle = 90 + angle
    else:
        if width < height:
            angle = angle + 180
        else:
            angle = angle + 90

        if angle > 90:
            angle = angle - 180

    return math.radians(angle)

def find_contours(frame, lower_limit, upper_limit):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    _, lightThresh = cv2.threshold(gray, 230, 255, cv2.THRESH_BINARY_INV)
    frame = cv2.bitwise_and(frame, frame, mask=lightThresh)
    blur = cv2.GaussianBlur(frame,(3,3),0)
    #Resize and coverting BGR to HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
    #Finding the range in image
    out = cv2.inRange(hsv, lower_limit, upper_limit)
    dilated = cv2.dilate(out, cv2.getStructuringElement(cv2.MORPH_RECT,ksi
ze=(3,3)))
    contours, _ = cv2.findContours(dilated, cv2.RETR_TREE, cv2.CHAIN_APPRO
X_SIMPLE)
    return contours
def get_pos(frame, lower_limit, upper_limit):
    x_res, y_res = 256, 128
    frame = cv2.resize(frame, (x_res,y_res))
    contours = find_contours(frame, lower_limit, upper_limit)
    result = frame.copy()
    cv2.drawContours(result, contours, -1, (255,255,255), 3)
    cx, cy = 0,0
    min_x, min_y = -10000,-10000
```

```
    rotation = -10000;
    if len(contours) > 0:
        for _,cnt in enumerate(contours):
            M = cv2.moments(cnt)
            if M['m00'] != 0:
                area = cv2.contourArea(cnt)
                if area>500:
                    cx = int(M['m10']/M['m00'])
                    cy = int(M['m01']/M['m00'])
                    if (cy > min_y):
                        min_x = cx
                        min_y = cy
                    cv2.circle(result, (cx, cy), 5, (255, 255, 255), -1)
                    cv2.circle(result, (cx,cy), 10, (0,255,255), 3)
                    rotation = get_contour_angle(cnt)
                    cv2.imwrite('out.png', result)
    else: return (-10000, -10000, -10000)  #default values
    return (min_x,min_y,rotation)
```

MATLAB code related to the main simulation loop and main functions:

```
  while (vrep.simxGetConnectionId(id) == 1)

      [~,state,~,Cuboid,~]=vrep.simxReadProximitySensor(id,ConveyorSensor,
      vrep.simx_opmode_streaming);

      if (state == 1) %if there is cube in front of Conveyor Sensor go and
pick it

          %pick
```

```
            [~,~,color]=GotoNearestCube(id,vrep,Robot,Joints,Camera,Convey
            orSensor);
            CloseVaccum(id,vrep,Cuboid,EE)
            %place
            GotoBasket(id,vrep,Robot,Joints,color)
            %release the cuboid
            OpenVaccum(id,vrep,Cuboid)
            %go to starting point
            RotateJoints(id,vrep,Joints,JointsStartingPos);
        end
    end

    function[p,rotation,color]=GotoNearestCube(id,vrep,Robot,Joints,Camera,C
onveyorSensor)

        [x,y,rotation,color]=GetPositionFromPy(id,vrep,Camera,ConveyorSensor
        );
        z = 0.405;
        p = [x,y,z];
        fprintf('coordinates: [%i,%i,%i] m\n',p(1),p(2),p(3));
        fprintf('rotation: %i degrees\n',rotation*180/pi);
        fprintf('color: %s\n',color);
        T = transl(p);
        theta_x = 0;
        theta_y = pi;
        theta_z = pi/2;
        T(1:3,1:3) = RotationMatrix(theta_z,theta_y,theta_x,'ZYX',true);
        %initial guess
        q0 = [1.2807    0.6263    1.5280   -0.5836    1.5708    0.2901];
        TargetPos = Robot.ikunc(T,q0); %1*6 vector
        TargetPos(6) = TargetPos(6) + rotation;
        RotateJoints(id, vrep, Joints, TargetPos);
    End
```

```
function GotoBasket(id,vrep,Robot,Joints,color)
    if (color == "r")
        x = 0.9;
        y = 0;
        q0 = [0.17,-0.78,-0.81,-3.14,1.59,1.39];
    elseif (color == "b")
        x = 1e-3;
        y = 0.9;
        q0 = [1.75,-0.78,-0.81,-3.13,1.59,-0.19];
    elseif (color == "g")
        x = -0.9;
        y = 0;
        q0 = [0 0 0 0 0 0];
    end
    z = 0.65;
    p = [x,y,z];
    T = transl(p);
    %q0 is an initial guess
    TargetPos = Robot.ikunc(T,q0);
    RotateJoints(id, vrep, Joints, TargetPos);
end
```

Proof of 345 path planning equations

As it was said, we have a polynomial of degree 5 as follows, which angle, angular velocity and angular acceleration at any moment can be calculated according to it.

$$s(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f$$

$$0 \le s \le 1, \qquad 0 \le \tau = \frac{t}{T} \le 1$$

$$\begin{cases} \theta(t) = \theta_I + (\theta_F - \theta_I)s(\tau) \\ \dot{\theta}(t) = (\theta_F - \theta_I)\dfrac{1}{T}s'(\tau) \\ \ddot{\theta}(t) = (\theta_F - \theta_I)\dfrac{1}{T^2}s''(\tau) \end{cases}$$

$$\dot{\theta}(T) = \ddot{\theta}(T) = \dot{\theta}(0) = \ddot{\theta}(0) = 0$$

$$\begin{cases} s(0) = s'(0) = s''(0) = s'(1) = s''(1) = 0 \\ s(1) = 1 \end{cases}$$

$$s'(\tau) = 5a\tau^4 + 4b\tau^3 + 3c\tau^2 + 2d\tau + e$$

$$s''(\tau) = 20a\tau^3 + 12b\tau^2 + 6c\tau + 2d$$

$$\Longrightarrow \begin{cases} \boxed{f = e = d = 0} \\ a + b + c = 1 \\ 5a + 4b + 3c = 0 \\ 20a + 12b + 6c = 0 \end{cases} \Longrightarrow \boxed{a = 6, b = -15, c = 10}$$

$$\Longrightarrow \boxed{s(\tau) = 6\tau^5 - 15\tau^4 + 10\tau^3}$$