**Robotics Vision – Group 5**

**Task 1 - Camera Calibration Given Simulated Checkerboard Data**

**Method and Derivation**

We perform the camera calibration to obtain the intrinsic and extrinsic parameters of the camera. With these parameters we will then be able to get the position and the orientation of the World frame with respect to the camera. The intrinsic and extrinsic equations are shown below:

With these equations, we can obtain a camera matrix H:

We know the values of x, y, X, Y but not H, so to do the camera calibration, we we need to take as many pictures as possible of a checkerboard in different positions and orientations where we can get the (Xi, Yi) coordinates with respect to the world frame. After acquiring the pictures, we then need the pixel position coordinates (xi,yi) of the corner points of each image for which we get the equation:

The proportional sign means we cannot calculate h values, however it also means the left side is a scalar multiple of the right side and therefore the cross product is 0:

And, from this, we can form a matrix with all the points of the checkerboard forming the following equation:

We can then find the h vector using Singular Value Decomposition thus we will calculate the matrix H. This is only for one image of the checkerboard, so we need to repeat this process for each image.

Using the know h values we can then calculate , so we get:

We then form the following equation putting in the variables obtained from each image:

Now we find the vector b using again, Singular Value Decomposition.

We also know that:

After obtaining the vectors h and b, we can go and recover the intrinsic parameters using the following equations:

So, now that we have the intrinsic parameters and with them, we form the matrix K:

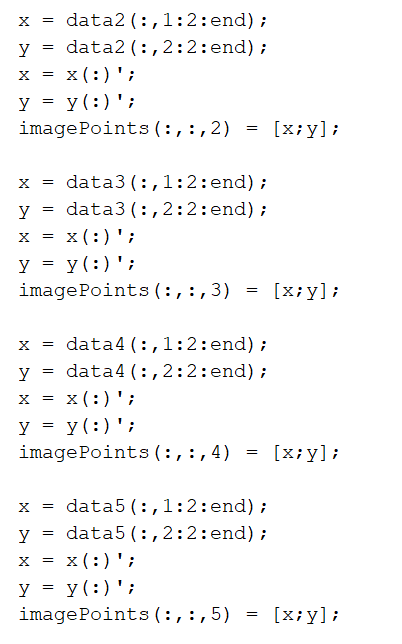
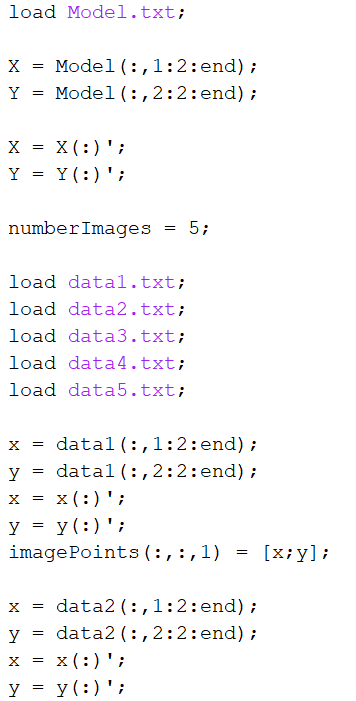
And then go onto find the extrinsic parameters. We know that for each image we have

We find the extrinsic parameters using a scaling factor to be:

**MATLAB code**

1. We first load the corner points of the physical printout checkerboard given in ‘Model.txt’. ‘data1.txt’ to ‘data5.txt’ gives the pixel coordinates of the 5 images. Typically, if the corner points are not given, a corner detection algorithm is used to achieve these points instead.

1



Text, application

Description automatically generated with medium confidence

2

1. We then build the matrix phi for each image to find the h vector using singular value decomposition. The for loop goes from one to five as there are 5 images, and phi is double the lines as each corner point has two rows in the matrix.

In order to calculate the h vector we use the “svd()” function in matlab, after which we form the vector h by taking the last column of V.

Graphical user interface, text, application

Description automatically generated

3

1. After finding h, we can go and find the values. To do that we just follow the equations. After finding these values we form the matrix v for each image, and in the end we find the b vector by again applying the “svd()” function on the v matrix, and taking the last column of V2.

Text, letter

Description automatically generated

1. After finding the b vector we were able to recover the intrinsic parameters (and put them in matrix K), by just following the equations.

4

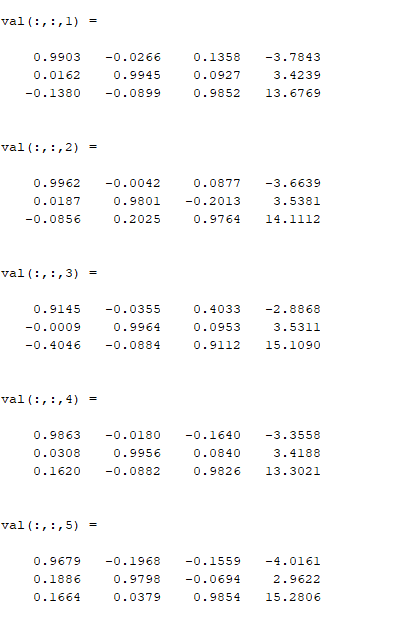
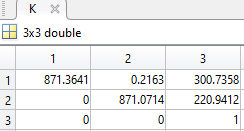
Text

Description automatically generated

1. In the end we recovered the extrinsic parameters for each image in “finalParameters” and also saved the extrinsic parameters of the first images as “Tcw” as this is going to be for the world coordinates calibration.

5

**Results**



Text, letter

Description automatically generated

For the results we can see that the values we obtained (left) are very similar to Zhang’s results (right), thus we can say that our code works properly.

**Task 2 – Robot Calibration given simulated TCP data**

**Method and Derivation**

After camera calibration (Task 1) is done, we will now do the robot calibration. For task 2, instead of placing the end-effector TCP on the corner points of the checker board. We were instead given sample positions of the checkerboard corner points according to the robot coordinate system (X­­ri, Yri, Zri) shown in table 1.0, as well as the positions of each corner point of the checkerboard (Xi, Yi, 0) with respect to the world frame shown in table 1.1.

Table 1.0 Coordinates of simulated probed positions of the TCP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point | 1 | 2 | 3 | 4 | 5 | 6 |
| Xri | -121.6 | -100.4 | -79.16 | -56.39 | -35.55 | 54.32 |
| Yri | 222.4 | 178 | 133.6 | 177.2 | 110.9 | 219.3 |
| Zri | -20 | -20.04 | -20.08 | -19.96 | -20.04 | -19.7 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Point | 1 | 2 | 3 | 4 | 5 | 6 |
| Xi | -22 | 0 | 22 | 44 | 66 | 154 |
| Yi | -22 | 22 | 66 | 22 | 88 | -22 |

Table 1.1 Coordinates of checkerboard corner points

Since we have the same ‘points’ expressed in 2 different frames – world and robot frames. We use the transformation matrix to find the relationship between these frames. In the world frame, there is no 'Z' because we are looking at a 2D plane. In the transformation matrix we need to solve for 2 parameters: ρij and tx,y,z.

From here, we can find the relationship between the 2 frames as:

**MATLAB code**

Text, letter

Description automatically generated

1

1. Firstly, we define the given example coordinates of the world and robot coordinates.

Graphical user interface, text, application, email

Description automatically generated

2

1. We then wrote out the variables that appear in the equations proven above, so that they could be substituted in a matrix.

Then did the first calculation for ρ11, ρ12, tx.

Graphical user interface, text, application, email

Description automatically generated

3

1. The variables were changed to calculate ρ21, ρ22, ty and the same method was applied to ρ31, ρ32, tz.

Graphical user interface, text, application

Description automatically generated

1. Lastly, the values calculated can be summarized in their respective matrixes and saved as a single homogenous transformation matrix.

4

**Results**

We finally get the result for Trw:

Text

Description automatically generated

Which is very close to the expected final answer we were given of:

**Task 3 – Image processing given simulated image**

**Method and Derivation**

For task 3, our goal was to find the shape colour, position and orientation of the objects in a simulated image (figure 1.0).

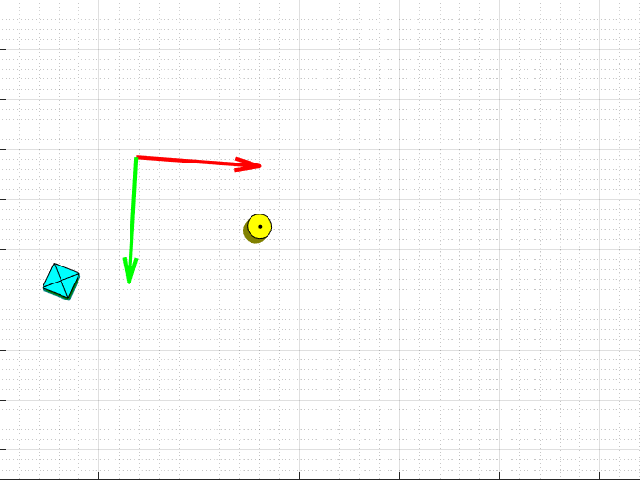


Figure 1.0: CameraView.bmp

In order to find the 2D position of the object in a pixel frame, we had to go through a method of image processing. With image processing, we adapt the image to allow us to analyse the picture better, so that we can perform tasks such as object recognition.

The first step in image processing is to convert the colour image to a grey-scale image. Colour images have 3 layers of red, green and blue. This is quite complex so we will grey-scale the colour image. Typically, the colour we see is an additive property of the 3 colours so taking an average of the 3 colours:

We can get an average value of the 3 colours. Through grey-scaling, the brightness of the pixel is determined by its value, where max value of 255 is for white and 0 is for black. Therefore we can use this to perform thresholding which allows us to highlight important information such as the object we are trying to capture in the image.

Afterwards, we binarize the image by making values above a threshold in the grey-scale image = 1 and values below = 0 which would give a purely black and white image.

As can be seen from the given image, there is a black dot in the centre of the circle and cross in the square which can be interpreted as ‘noise’ which we want to remove. Therefore we use a method of pixel group processing called Median filtering, which is a nonlinear processing method. Averaging methods can be susceptible to outliers which can heavily affect the outcome, therefore using median filters, no outliers will affect it and instead will be almost completely removed. In our case, our median filter takes into account 3 pixels ahead and behind of the current (i,j) pixel coordinate because the ‘noise’ was quite large and couldn’t be filtered with anything less.

Next, we performed blob detection, finding all the properties of the object such as centroid, circularity, perimeter, orientation.

To find the center of the object we used moments where the p-qth moment of an image is defined as:

This essentially is the sum of x location to power p and y location power q, multiplied by intensity of pixel (=1) at that x,y location (distance). For example, at p=q=0 we get:

Which is the sum of all the pixels in the object and therefore is the area.

Furthermore, using moments, we can calculate even more parameters, starting with centroid, the formula is:

Now that we have the centroid, we can calculate all moments with respect to the center of the object using a formula for the central moment:

As we now have the 2 essential properties, centroid and orientation, the robot in theory can pick up the object. However, to make things easier for us to recognize and detect objects, we can perform a few calculations to get corners, or circularity of the object.

Here we will focus on the circularity calculation which has a formula of:

For example, in an ideal case, the circularity for a circle is 1 and π/4 for square.



**MATLAB code**

talk about the filter for circle and how not used because it makes circularity worse

Graphical user interface, text, application

Description automatically generated

1. The camera image ‘CameraView.bmp’ is imported. Then the image is transformed to greyscale using the ‘unsigned 8-bit integer’ where all elements in the vector have a range from 0-255

1

Text

Description automatically generated

1. Firstly, we will process the cylinder. As we know the cylinder is yellow, this means RGB(255,255,0) which is put as a condition statement. The code will iterate every column for each row, turning all yellow pixels white and other pixels black.

2

Text

Description automatically generated with medium confidence

1. A Median filter is then applied to remove noise (the black dot in the centre of the circle). In this case, we had to look 3 pixels ahead and 3 pixels behind in order to remove it.

3

1. Text, letter

   Description automatically generatedNext, we convert the image to binary. As can be seen, there is 2 Ibw\_cylinder, the 1 currently commented out is the filtered image and the other is non-filtered image. (see figure 2.0 for the difference between filtered and non-filtered)

4

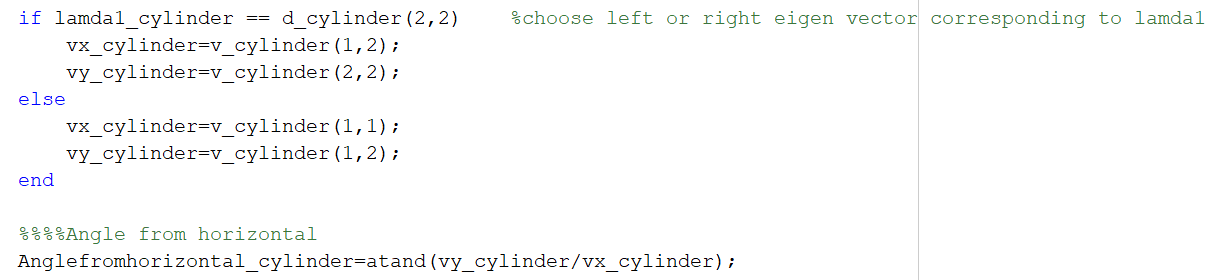
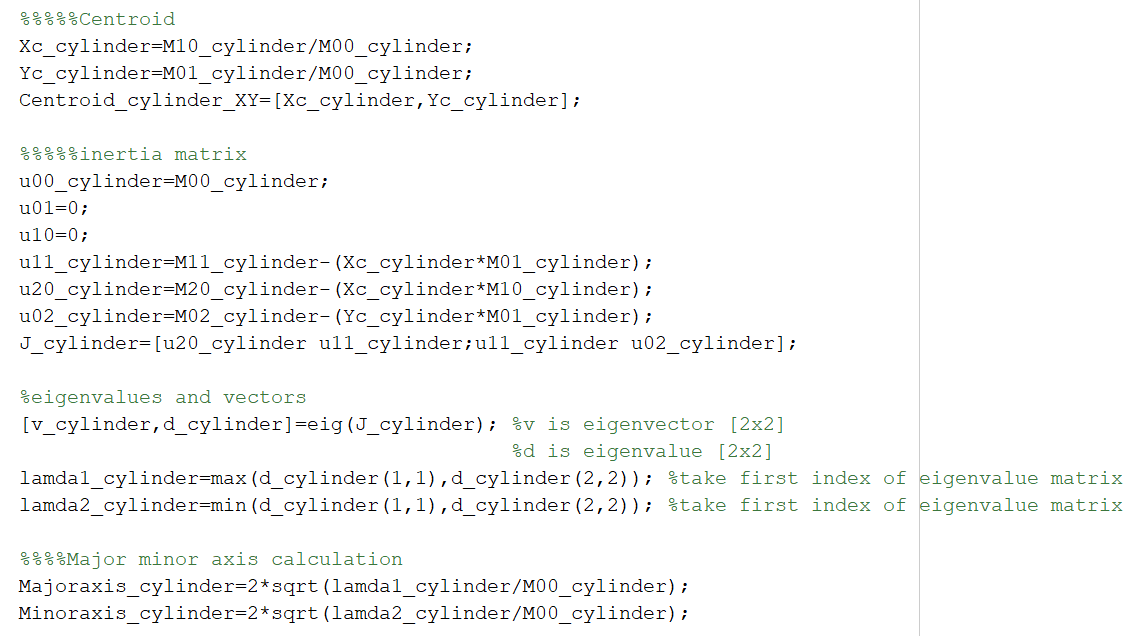
Text

Description automatically generated with medium confidence

1. The moment is calculated next. Which is necessary to allow us to find the centre of the shape and thus which pixel coordinate the object is in in the image.

The p-qth moments of the image is calculated by the sum of the ‘i’ location to the power p or ‘j’ location to the power q, multiplied by the intensity of the pixel (in this case 1) at that i,j location

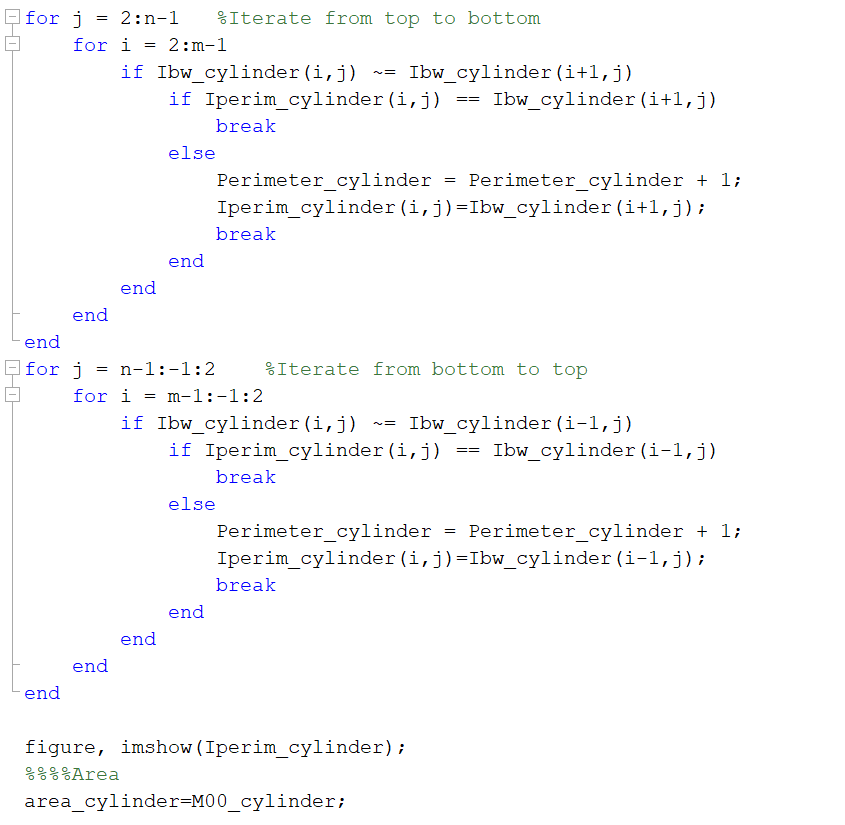
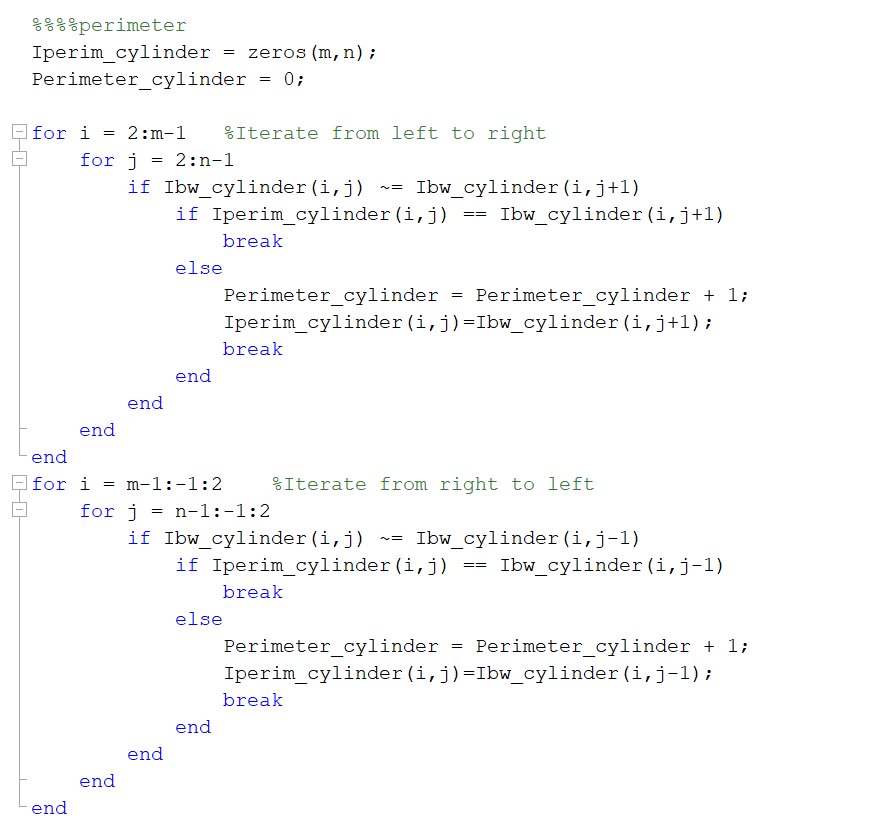
5



1. Here, we calculate a few important properties of the shape such as centroid, major/minor axis of shape, and the orientation of the object.

This is essentially done by inputting values into the formulas derived above.

6



1. Next we wanted to calculate the perimeter. In order to do this, we decided to iterate from left to right, right to left, top to bottom and bottom to top, see figure 2.1 for details. If the pixel after the current was different, then it would add +1 to the counter, then start again on the next row. In order to not count the same pixels, if a pixel found in a separate iteration, then the line is skipped.

I then added each counted pixel to a new blank grid so that the perimeter could be plot and visualised (figure 2.0).

A picture containing text, shoji

Description automatically generated

7

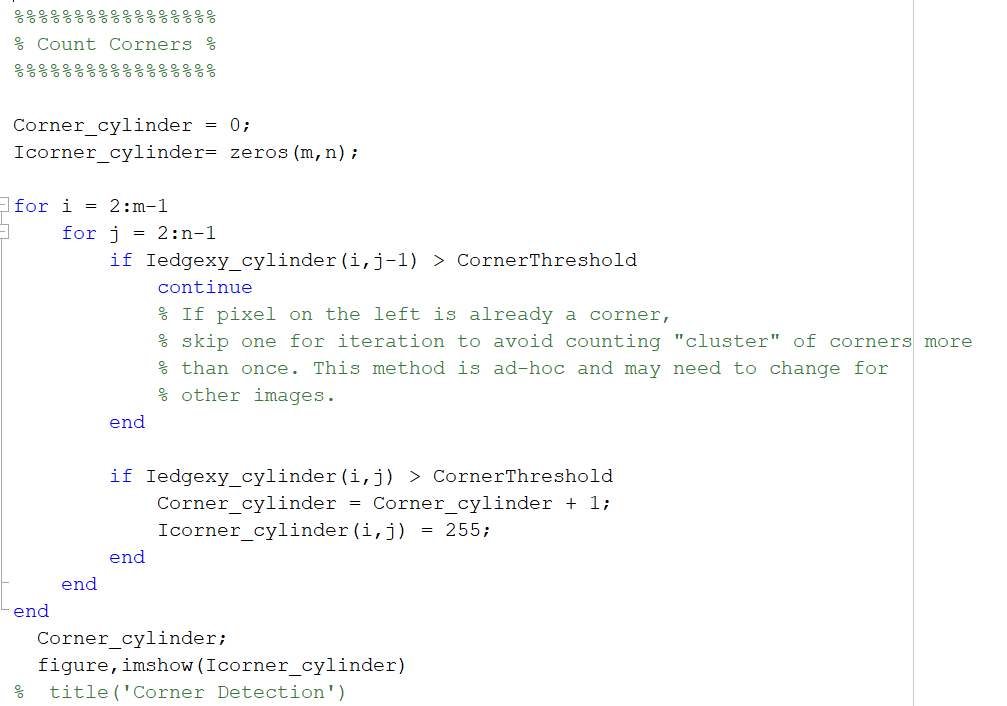
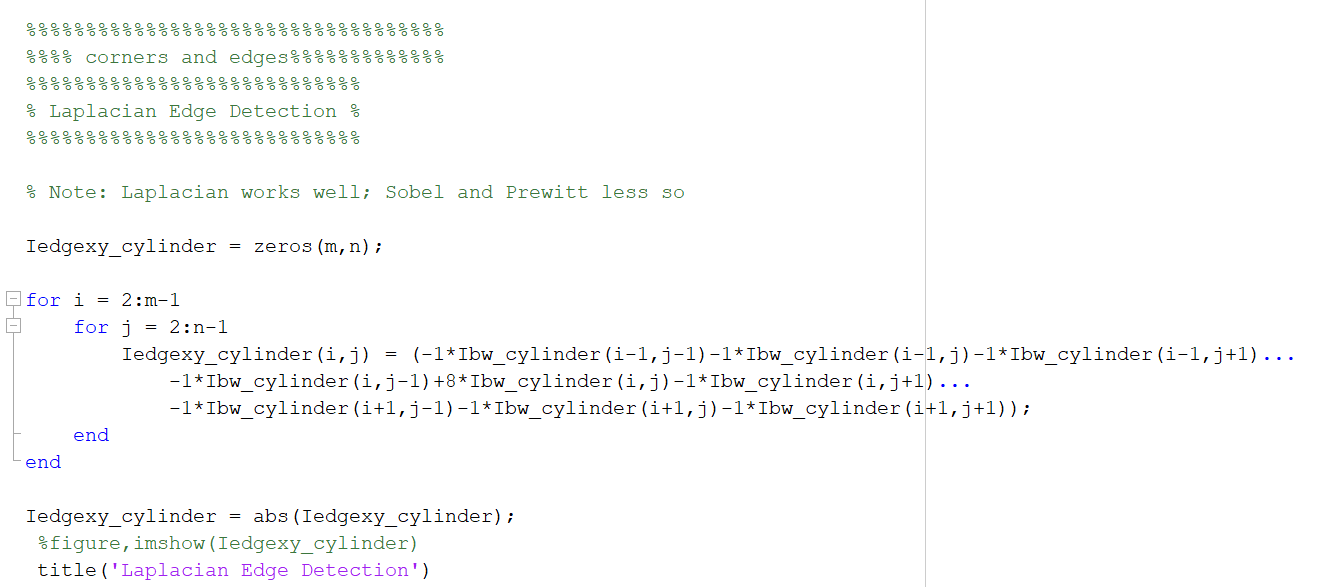
Figure 2.1: Diagram showing direction of iteration

1. Text

   Description automatically generatedLastly we calculate and show the circularity, RGB values, displayed in a table, see table 1.0, 1.1

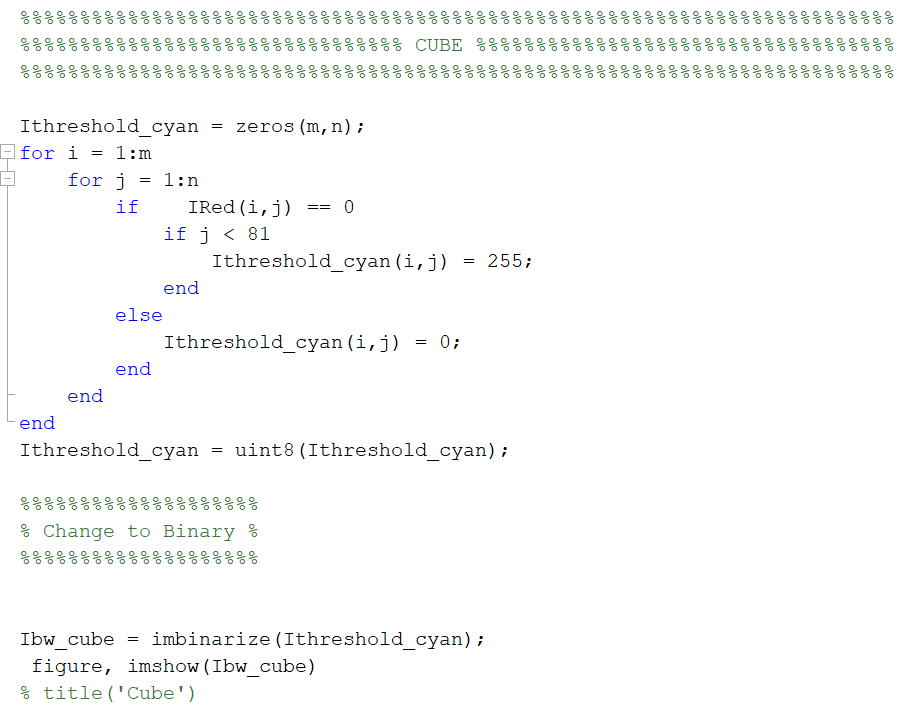
8

1. We also attempted to do corner and edge detection of the shape, using Laplacian edge detection and corner counting.

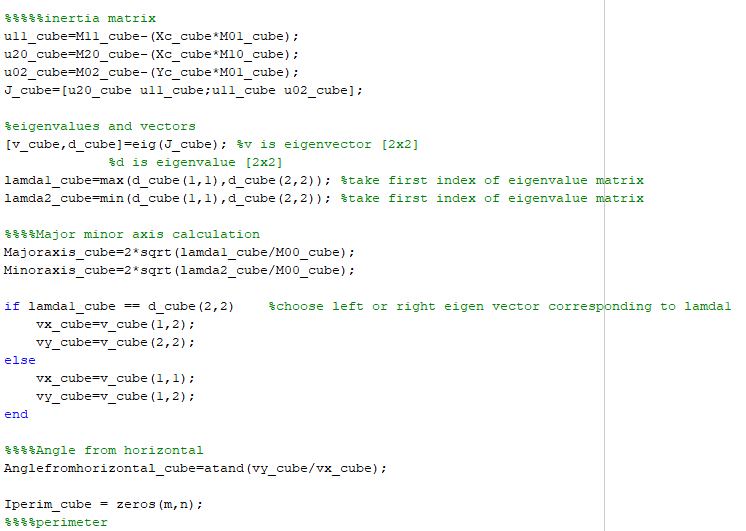


9

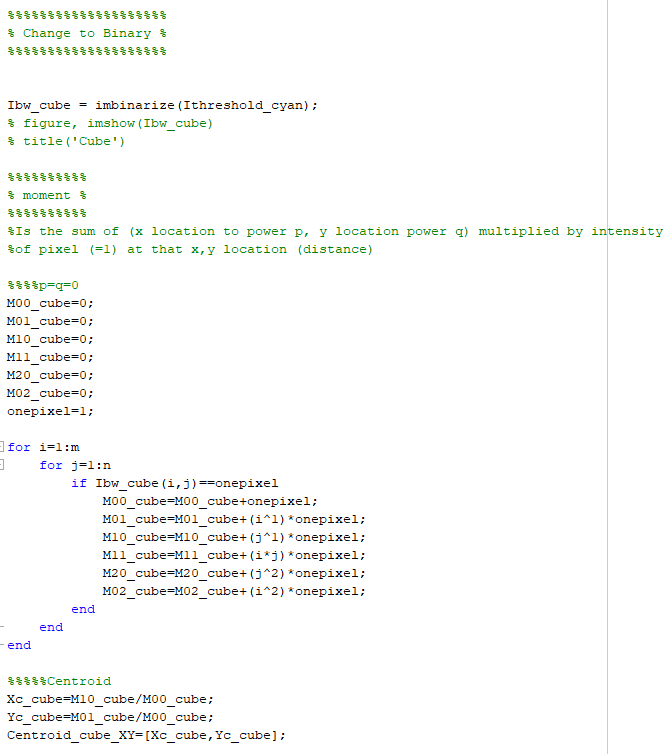
1. The exact same process to the cylinder was performed on the cube, however the only difference being, to detect the threshold, the pixels were limited to detecting the left side only.



10.1



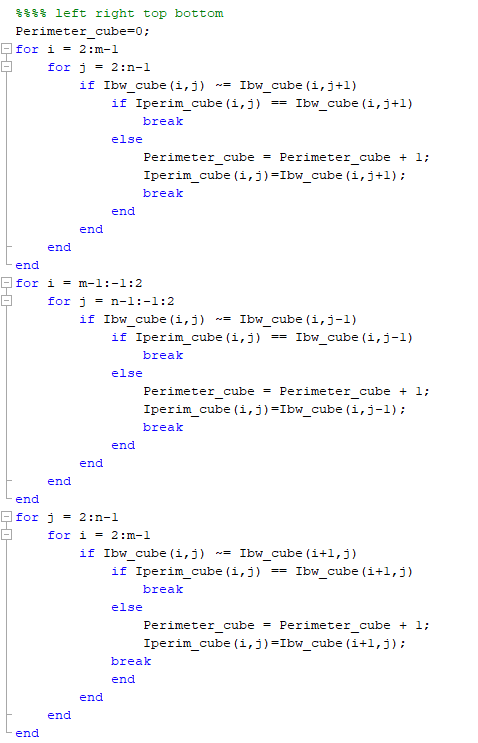
10.3



10.2



10.5



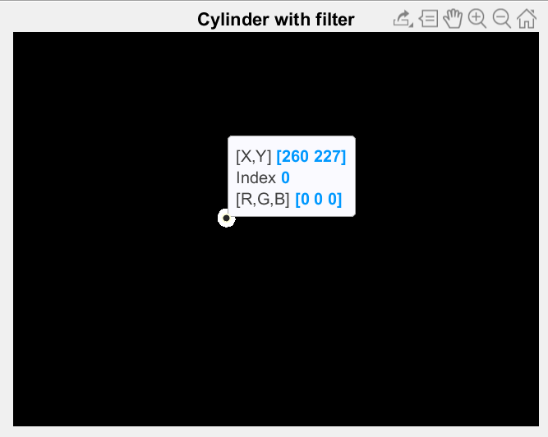
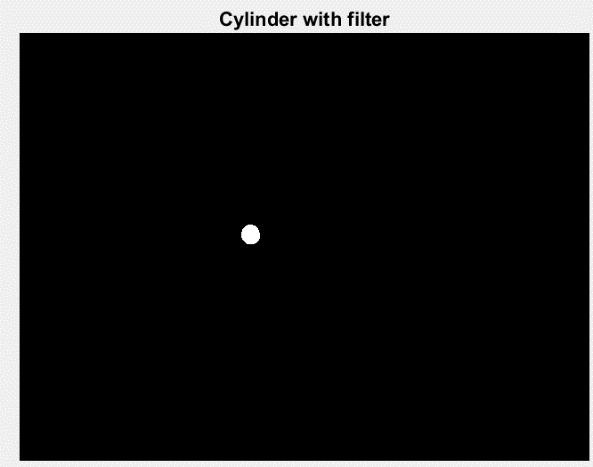
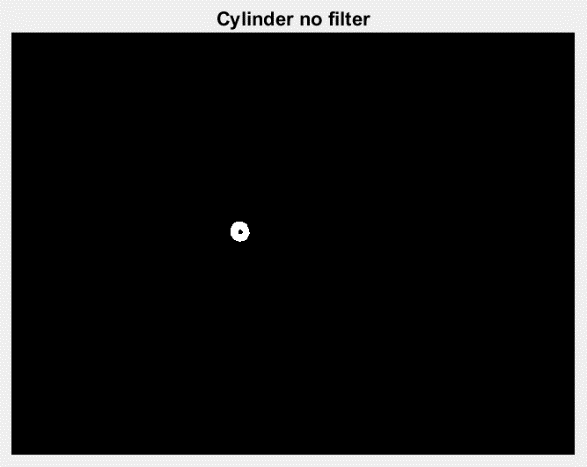
10.4

**Results - Cylinder**

As can be seen from figure 2.0, we can see that without filter, there is a black dot in the centre of the circle, whereas, with filter the dot was filtered out, however the circular area was visibly smaller. In addition, the table of properties for each cylinder is also shown below (table 1.0, 1.1). An interesting observation is that the circularity of the cylinder without filter is ‘better’ than the circularity of the cylinder with the filter as an ideal circularity of a circle is 1. The reason this could be is because the cylinder is too small so that the ratio between area and perimeter isn't significant enough, therefore messing up the calculation.

Other than this, the results seemed to be correct, such as the centroid coordinates that match with the MATLAB ‘data tips’ function (Figure 2.0 right).

Figure 2.0 - output after, thresholding, filtering and binarizing the image. Also used MATLAB data tips function to find center of cylinder coordinates (Right)



Graphical user interface, text

Description automatically generated with medium confidence

Table 1.0 – properties of cylinder with no filter

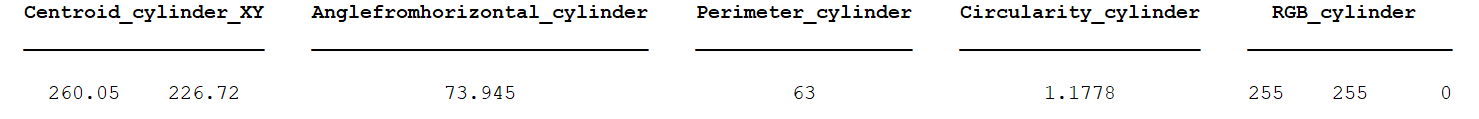
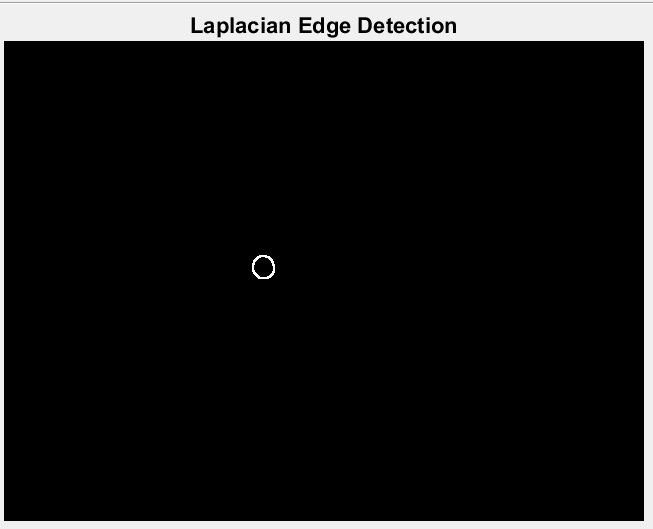
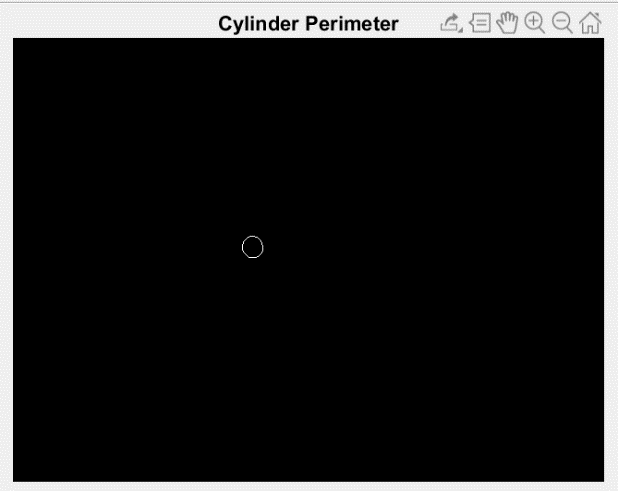
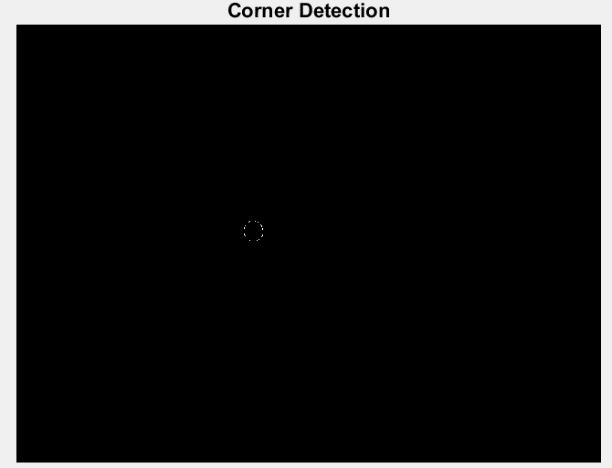


Table 1.1 – properties of cylinder with filter

Below, shows figures 2.1 of the perimeter, edge detection and corner detection. The perimeter method of counting pixels seemed to work well because on comparison, the counted pixels seemed to form an exact perimeter of the cylinder. In corner and edge detection, the results also seemed to work well.

Figure 2.1 – Cylinder perimeter, Laplacian edge detection and corner detection algorithms output



**Results - Cube**

Because the cube did not require filtering, the binarized figure showed the exact shape of the cube which made the process of calculating the perimeter and its properties much easier and accurate.

However, once again the circularity of the cube (table 1.2) seemed to be incorrect as the value we should be expecting is around π/4, but instead we get 0.99391 which is what the circle should get. Again, this is likely due to the cube being too small so that the ratio between area and perimeter isn't significant enough.

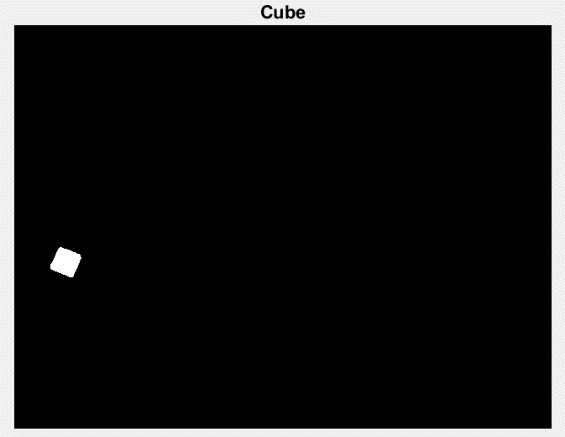
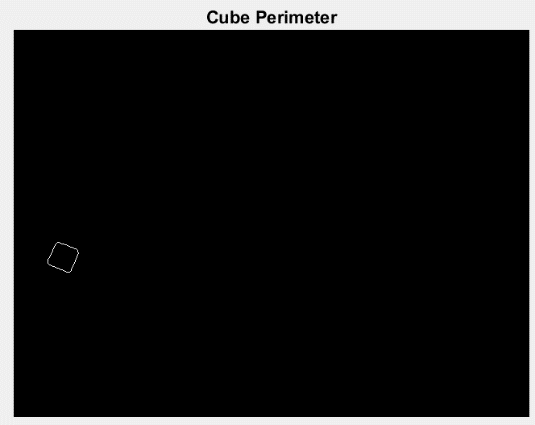
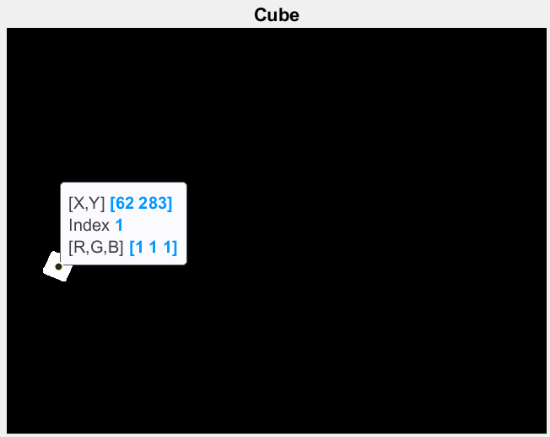


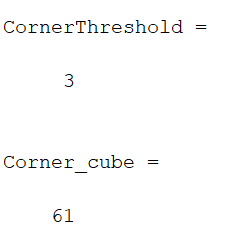
Figure 2.2 - output after, thresholding, filtering and binarizing the image. Also used MATLAB data tips function to find center of cube coordinates (Right)

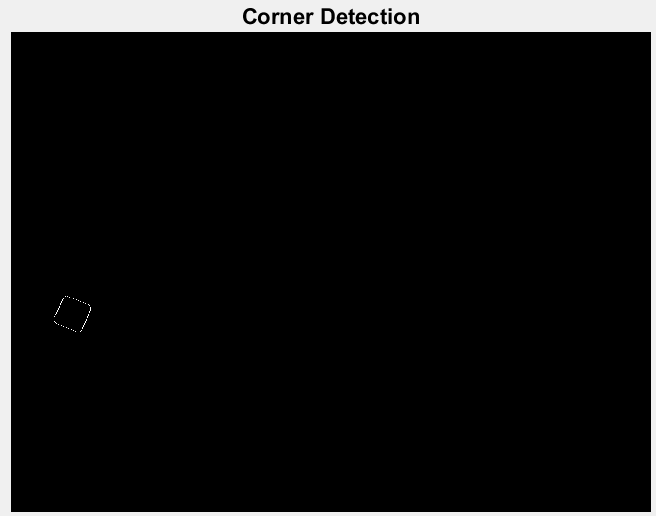
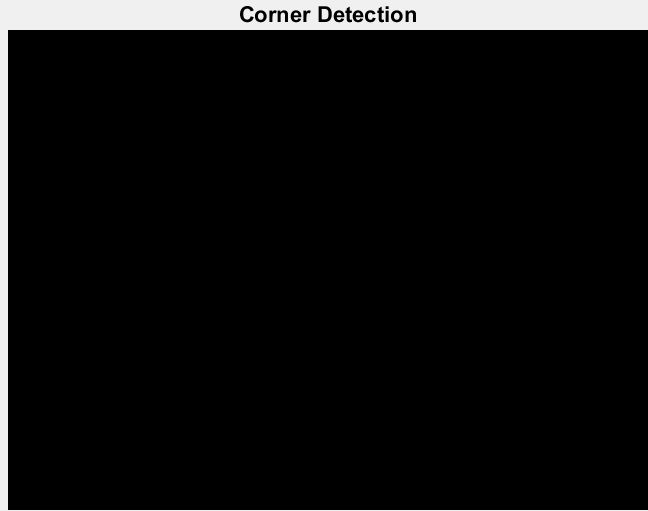


Graphical user interface, text

Description automatically generated with medium confidence

Table 1.2 – properties of cube

****Corner detection was attempted with the cube however, as can be seen in figure 2.3 there is a significant change in the number of corners detected when the corner threshold differs between 3 or 4 pixels. This is likely due to the size of the shape and therefore this result was ignored.



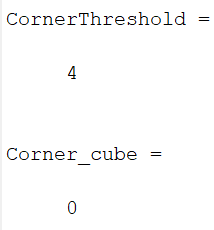


Figure 2.3 – Corner algorithm for cube with corner threshold of 3 (left) and 4 (right)

**Task 4 – Relating pixel position to robot position using simulated data**

**Method and Derivation**

After finding the 2D position of the image in the pixel frame (x,y) from task 3, we can now find the relationship between the pixel coordinate and the world coordinate (X,Y). Then using all the parameters we calculated previously we can find the relationship between camera and world coordinate to get 3D position of the object (Xr, Yr, Zr).

This can be expressed mathematically as shown:

Text, letter

Description automatically generated**MATLAB code**

1. Firstly we load in the matrices calculated previously as well as redefine K and Tcw parameters.

1

1. Text, letter

   Description automatically generatedThen after finding the extrinsic parameter H, we can calculate the object in real 3D space. We do this by finding the world coordinate which is the inverse of H\*Pixel coordinate of cube/cylinder. The last row is then normalised.

2

Finally to calculate the position of the object in the robot frame, the transformation matrix (imported before), is multiplied by the world coordinates to get the coordinates of the object in the robot frame and we just take the first 3 rows to remove the ‘1’.

**Results**

The results we get from this calculation is:

Text

Description automatically generatedText

Description automatically generated

Which makes sense as the Z coordinates for both cube and cylinder are on the same plane (remains around -20), and the X,Y coordinates match proportionally to the 2D image we were given.

**Task 5 -** **Camera Calibration Experiment**

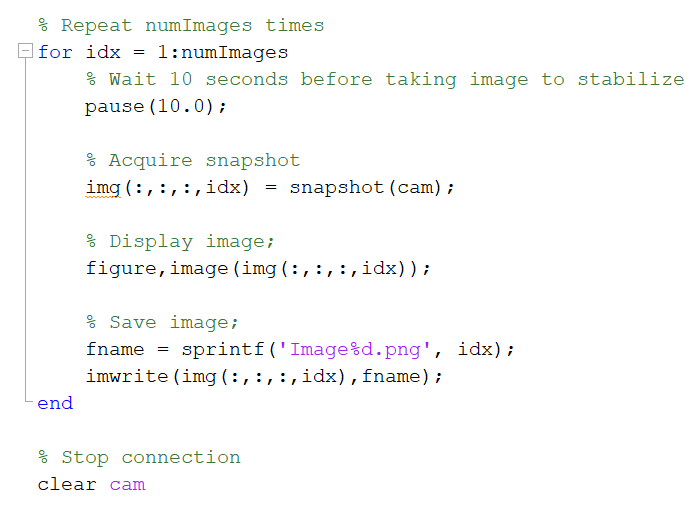
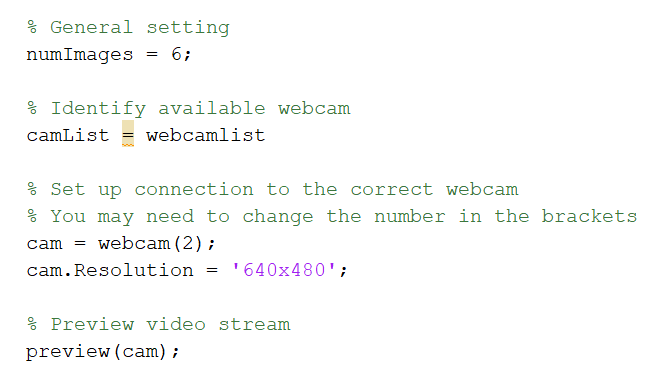
**Method and Derivation**

First, we measured the length of a checkboard square which turned out to be 21mm(figure 3.0), we then went ahead and took 6 images of the checkerboard with different orientations and positions using MATLAB code provided to us, with an additional 7th image of the checkerboard in the intended working position. After having all the data needed, we went ahead and adapted the code from task one to our conditions.

Figure 3.0 – Measurement of the checkerboard



**MATLAB code**



1. We firstly take 6 different images of the checkerboard for the camera calibration.

1

Text, application

Description automatically generated

1. After that we take another image with the checkerboard in a fixed position, as this is the intended working position.

2

Graphical user interface, text, application

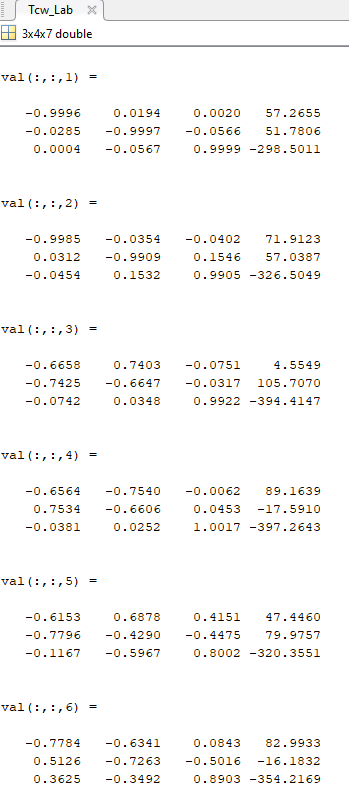
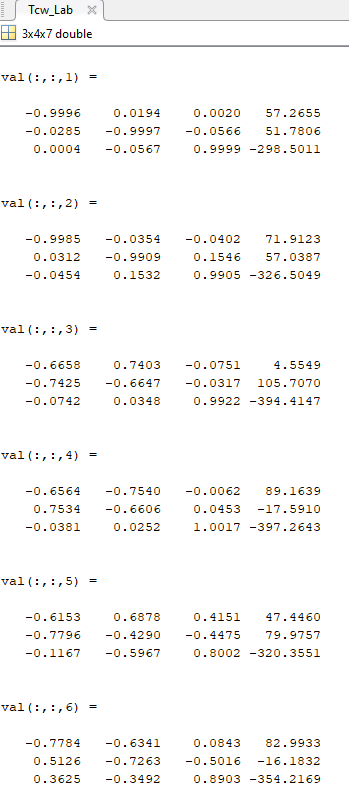
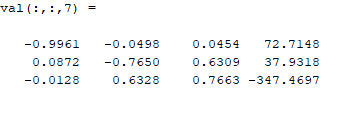
Description automatically generated

3

1. After that we extract the checkerboard points and reuse the code used in task 1. The only changes made are the number of images, and the size of matrix phi now becomes “phi(54\*2, 9)” while H becomes H(3,3,7) (there are 7 images).

**Results**

For our 7 images we then got the following results for K, and Tcw which we save as a .mat file to be used later in task8. There are 7 Tcw\_lab matrices because of the 7 images however we only take the first one which corresponds to the first working image.



**Table

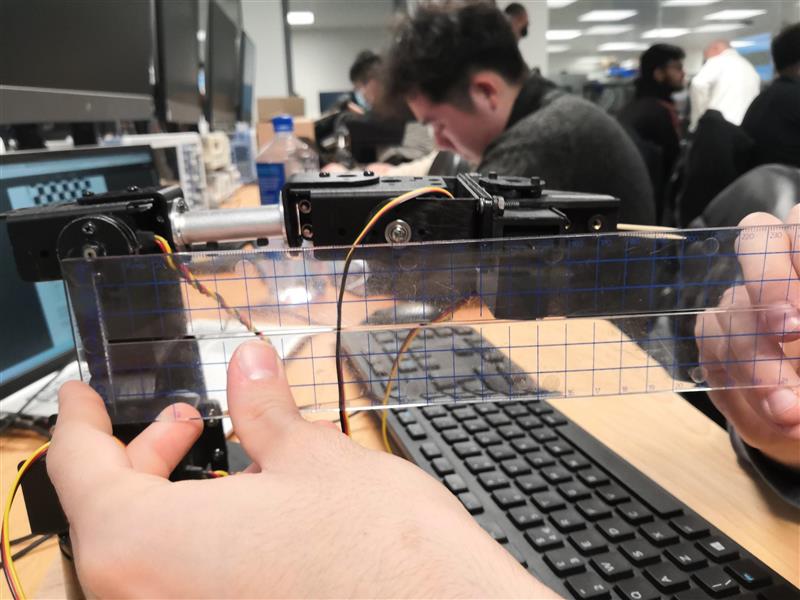
Description automatically generated**

**Task 6 – Robot calibration experiment**

**Method and Derivation**

Task 6 was the robot calibration. For this we used the inverse kinematics code from the last report to control the end-effector using potentiometers to move it to certain locations on the checkerboard and touching the corners with a toothpick grabbed by the end-effector and measure the coordinates of the point P with respect to frame {3} shown in figure 4.0. As the robot moved to each location, the serial monitor on the Arduino was turned on so we could take readings of the angles for each joint of the robot θ1,2,3.

Figure 4.0 – Image captured when measuring length from frame {3} to point P



Then using forward kinematics code from last report, we substituted in the each angle to find the cartesian positions of the tool tip with respect to the robot frame Xri, Yri, Zri.

Finally we can substitute in all the both the world and robot coordinates into the MATLAB code which does the calculation shown in task 2, so that we will get a single homogenous transformation matrix for the position and orientation of the world frame wrt to the robot frame.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Point | Xi | Yi | θ1i | θ2i | θ3i | Xri | Yri | Zri | Picture |
| 1 | 0 | 0 | 126.41 | 62.98 | 1015.01 | -11.99 | 16.25 | -6.11 |  |
| 2 | 42 | 0 | 115.04 | 70.50 | 118.33 | -7.42 | 15.89 | -7.17 |  |
| 3 | 42 | 21 | 113.62 | 62.19 | 103.97 | -8.18 | 18.66 | -6.10 |  |
| 4 | 0 | 63 | 116.34 | 40.41 | 69.24 | -11.50 | 23.23 | -4.34 |  |
| 5 | 63 | 84 | 100.96 | 41.6 | 70.58 | -4.90 | 25.29 | -4.24 | **A picture containing text  Description automatically generated** |
| 6 | 105 | 63 | 92.00 | 54.03 | 89.66 | -0.8 | 22.92 | -5.00 | **A hand holding a black and white checkered checkered checkered checkered checkered checkered  Description automatically generated with medium confidence** |
| 7 | 126 | 21 | 86.12 | 70.03 | 113.17 | 1.28 | 18.82 | -5.96 |  |
| 8 | 168 | 0 | 72.08 | 75.99 | 125.06 | 5.03 | 15.54 | -7.22 |  |
| 9 | 168 | 63 | 75.27 | 53.07 | 89.13 | 5.84 | 22.21 | -5.22 | **A person writing on a piece of paper  Description automatically generated with medium confidence** |
| 10 | 147 | 105 | 81.56 | 34.98 | 59.86 | 3.98 | 26.83 | -3.71 |  |

Table 2.0 – Summary of all coordinates and corresponding photo taken during lab

**Arduino and MATLAB code**

A picture containing calendar

Description automatically generated

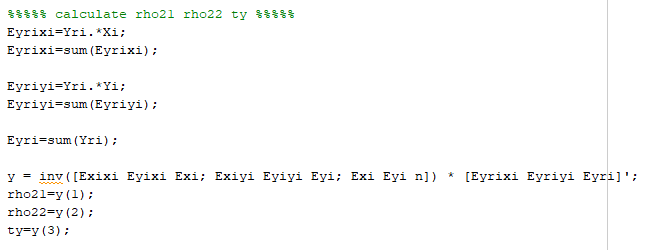
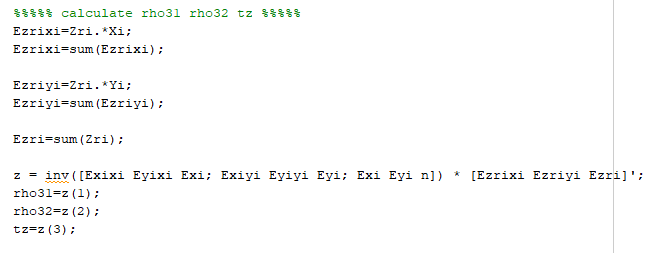
1. Firstly, the world coordinate (corners of checkerboard) and robot coordinates (calculated from forward kinematics) are input into its respective vectors.

1

1. Graphical user interface, text, application, email

   Description automatically generatedThen we wrote out the variables for ρ11, ρ12, tx  and that were to go into the matrix as well as extracting the results from the matrix.

2



1. And like task 2, the variables were changed for ρ21, ρ22, ty and as well as ρ31, ρ32, tz

3

Graphical user interface, text, application

Description automatically generated

4

1. Finally the values calculated are combined to form the a homogenous transformation matrix Trw\_lab.mat.

**Results**

Text

Description automatically generatedAfter completing the full procedure, we get the a homogenous transformation matrix which is demonstrates the robot-to-world relationship figure 4.1 Which we will use in task 8 to relate pixel position to robot position:

Figure 4.1 – Final matrix for robot-to-world relationship

**Task 7 – Image processing experiment**

**Method and Derivation**

After task 6, the robot is now calibrated and we can remove the checkerboard. We use exactly the same code as task 5 to take photo at the working place but change only the image name into object1. After doing this, we get a three-dimensional RGB image. What we now need to do is to use image processing method to determine the shape, position and orientation of the object.

First, we turn the original RGB image into grey scale image. Doing this is because, firstly, the original image is three-dimensional image, and it is complicated to do the further image processing. Grey scale image makes the implementation of filter and threshold become much easier. Secondly, problem might occur as we use the RGB image to determine the edge. This is because the edge detection is based on the detection of changing gradient. However, at the same pixel point, the gradient for different colour might be different as well (because the implement of the detection carries out individually for red, blue and green), that means the edge are different as well. Thus, error might occur.

After doing the grey scale image, we use threshold method to convert the image into black and white image. Instead of using the histogram to determine the threshold, we use the RGB finder to find the target RGB parameter, and we use this to adjust the value of threshold. The threshold gave a range of approximate values that the object could take on depending on the lighting conditions and when the correct threshold range is found, the object finding location is quite robust.

Finally, there may be some noise, which we need to get rid of if we want to get a clear edge. Thus, we use a median filter. Then, after getting rid of noise, we put the image into binary, so that this can be doing the further blob detection.

After getting a clear image and convert it into binary, we calculate the key parameters such as moments which was derived in task 3 which can help us to get the centre of object.

**MATLAB code**

1. Text

   Description automatically generatedThis part is importing the image and convert the original 3D image into 1D grayscale image. The formular is to add each element for RGB together and divided them by 3. which gives the mean of the RGB elements pixel by pixel.

1

Text, letter

Description automatically generated

1. This part we use two ‘for loop’ to scan through the image. After getting the grey scale image, we generate a zero matrix with same size as the 2D grayscale image. After that, according to the threshold we determined from the RGB finder, we use several condition statements to threshold the image. Above the threshold, where if the pixel is in the range, we set it to 255, and below the threshold we set to 0, which then gives us a black and white image.

2

A border is set up around the image of size 50 pixels from sides, top and bottom because otherwise the object would be out of range of the robot arm and camera.

1. Text

   Description automatically generatedFor this one, we scan through the image, and take the median for the adjacent pixel (PGP method). We divide the entire image into group pixels and take median in terms of groups.

3

A picture containing graphical user interface

Description automatically generated

1. Change from the black and white image (0 or 255) into binary image (0 and 1) using imbinarize function

4

Text

Description automatically generated

1. According to the formular of moment, we first calculate M00 to M02, and then multiplied by intensity of pixel (=1) at that x,y location (distance)

5

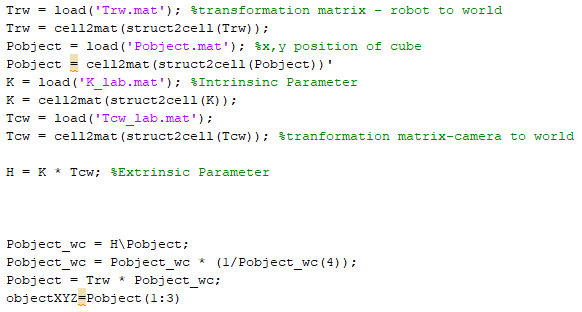
1. Text

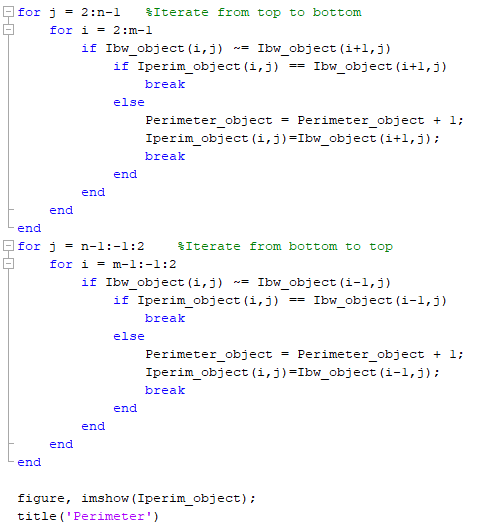
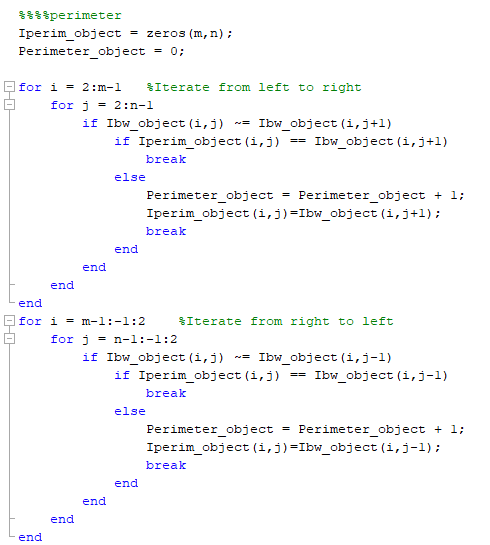
   Description automatically generatedAfter we have the centroid of the target, we can implement the parameter we calculated above to derive major/minor axis of the target, the shape, the and also the orientation of the object according to the formular in the method part of task7

8

6

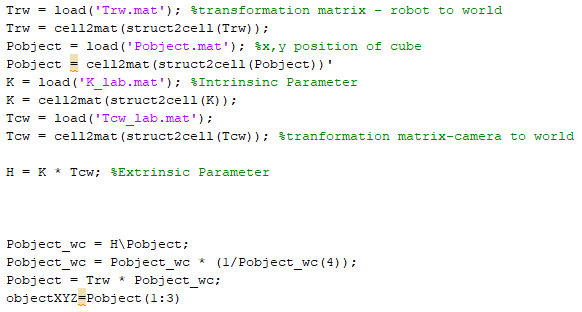
1. Here we calculate the perimeter, which we write the code to scan the image from left to right; right to left; top to bottom; and bottom to top. Also, we introduce a counter to count the row and column that we are going to scan. If the current was different after scan, than we add one to the counter, which means scan the next row. Also, if the pixel found in a separate iteration, then the line is skipped.

7

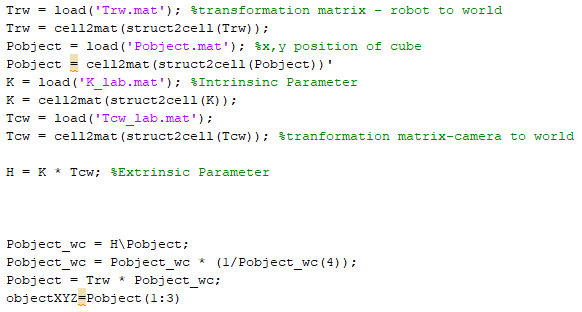


Text, letter

Description automatically generated

8

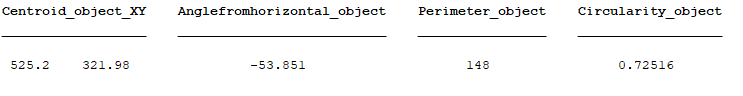
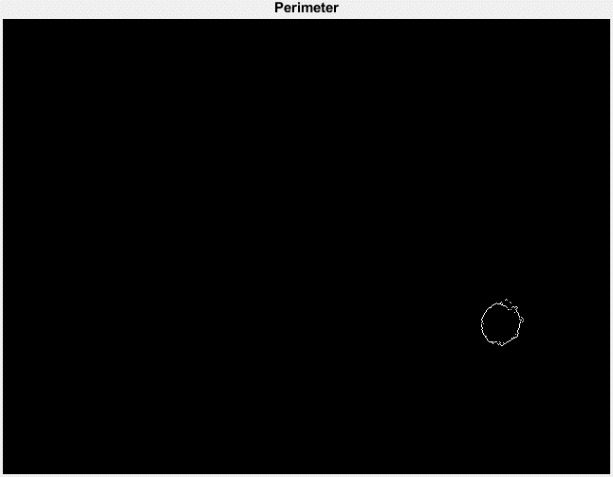
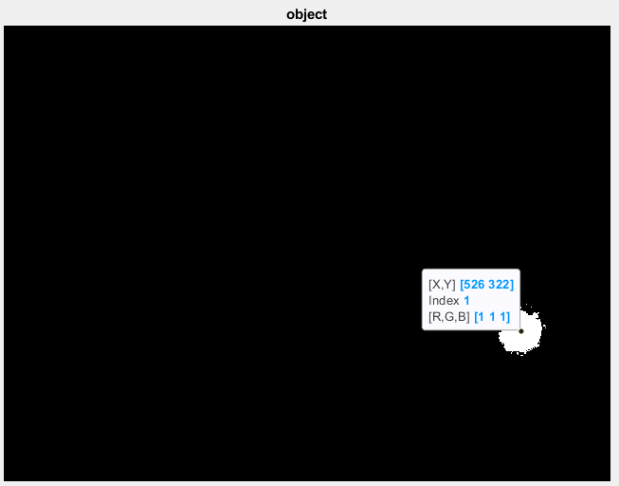
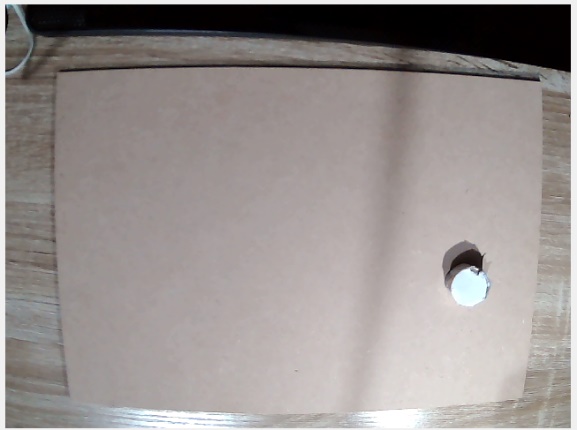
8. Finally, after getting all the values for the parameter, we calculate the circularity and shows together with the RGB value in a table.

8

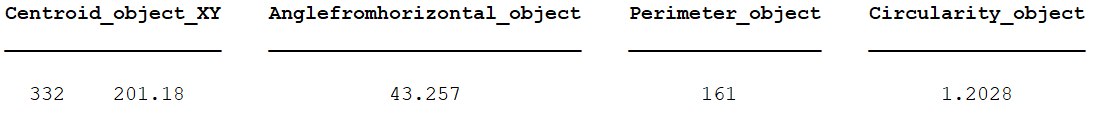
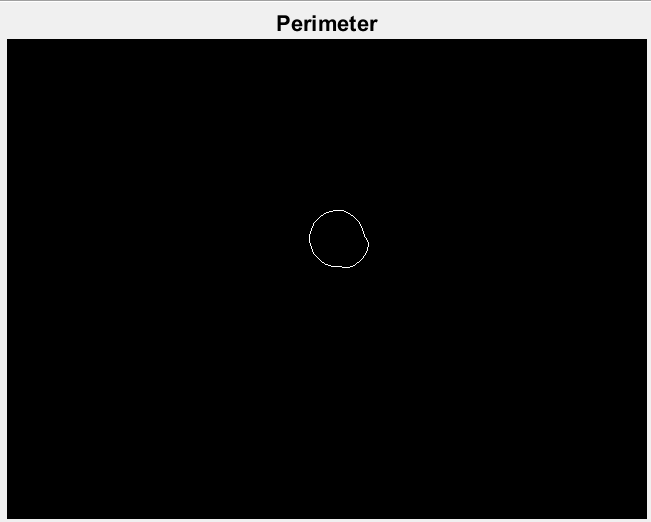
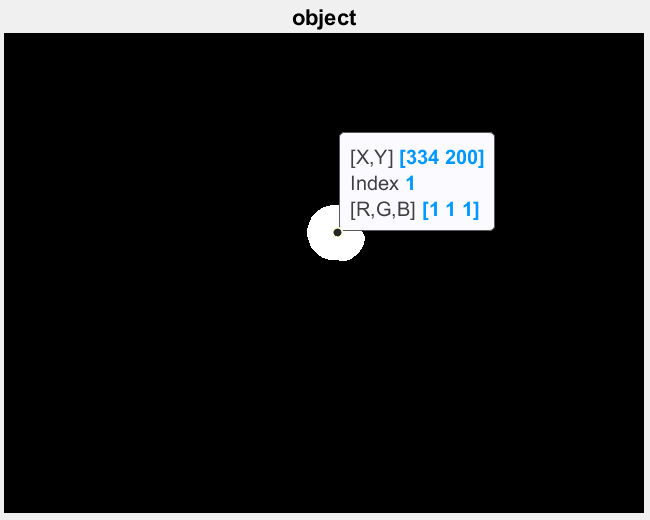
**Results**

After the image is taken, we run the image processing algorithm and the image can be detected successfully as shown below. We can also change the object and its position, and they can be detected accurately as well. As we can see from the result ‘First’, the Centroid\_object\_XY calculated by the MATLAB is 525.2, 321.98. As we click the centre of the processed image, it shows to be 526,322, which matches. After we rotated the target by 90, the angles from horizontal rotate 90 degrees as well. Similar result can be seen in the ‘Second’, where the centroid position and object are both detected. The centroid values are then saved in a .mat file to be used in task 8.

First



Second



**Task 8 – Relating pixel position to robot position using experimental data**

**Method and Derivation**

Until now, we have the transformation matrix of robot to world, the x,y position of the object, the intrinsic and extrinsic parameters, and the transformation matrix of camera to world. The pixel frame and the camera frame are connected by the intrinsic parameter K and the camera to world position and the robot to world position can be connected by the extrinsic parameter, which is H, and from this we can finally get the 2D pixel and the 3D robot frame.

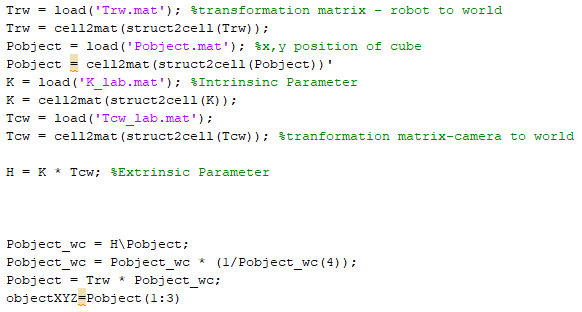
**MATLAB code**

Text, letter

Description automatically generated

1. We load all the .mat files: Trw, K, Tcw, and Pobject in to the workspace, and use cell2mat function to convert the data in the processable format.

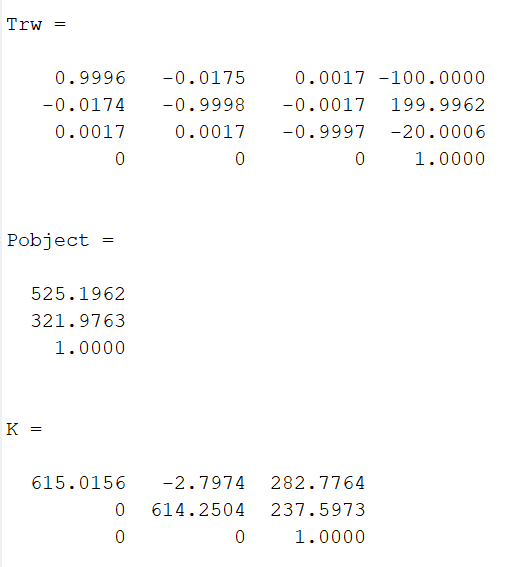
Then, we implement the calculations derived in task 4 to find the XYZ position of object.

1

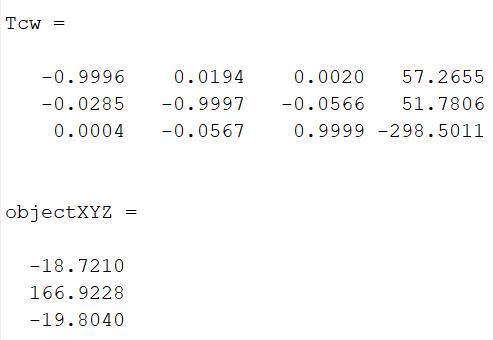
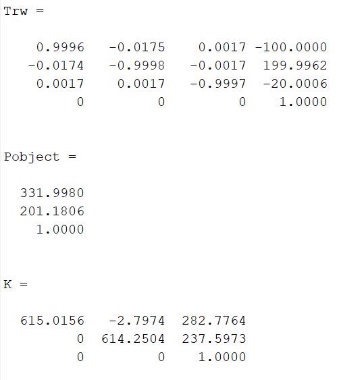
**Results**

The result of the code that we ran separately for the first and second image in task 7: ‘First’ and ‘Second’, gives the objects position. After we changed the position of the target twice, the XYZ position of the robot change as well, and doing a mental check, they make sense as the Z axis remains constant on the plane, but X,Y changes, acting similarly to task 4..

First



Second



**Trajectory planning**

**Task 9 –Trajectory Planning for Robot Motion**

**Method and Derivation**

The trajectory planning is taking into consideration the time profile of the motion that the robot must execute. To control the velocity at the beginning and at the end of the motion to be zero we can use a cubic polynomial to manipulate the motion of the robot and avoid “rough and jerky” moves. When we want our robot to move from one point to another the cubic polynomial comes in very handy as we can control the time it takes for the robot to completely execute the movement while also controlling the number of intermediate movements the robot is going to take.

For our trajectory plan, we wanted the robot to move to the object over a period of 5 seconds, move back to the origin, place object at a new position over a period of 5 seconds and return back to the origin, which meant that multiple target points were included in the code for the robot to do this.

To ensure that the velocities at the start and end points are zero, we can use a cubic polynomial with the following equations:

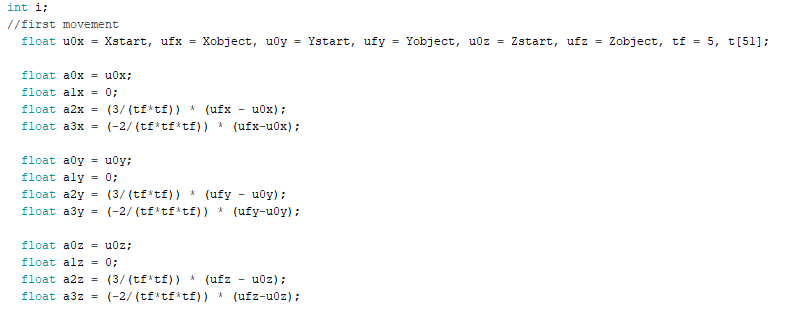
Where u0 is the initial position and uf is the final position. Thus with the initial and final position and the duration of the motion we can form a cubic polynomial for a smooth movement of the robot.

**Arduino code**

1. **Table

   Description automatically generated**We will find the trajectory using cubic polynomial for each axis individually. For this Xstart, Ystart, Zstart are the starting position coordinates, XPosition , YPosition, ZPosition are the coordinates that are going to happen through the motion, Xobject, Yobject, Zobject are the coordinates at which we have the object, Xnew, Ynew, Znew are the coordinates of the place to where we are going to move the object.

1



2

1. Here we declare the initial and final positions of each axis and we calculate the variables a0, a1, a2 and a3 for each axis as well.

Text

Description automatically generated

1. We then declare the position of the place at which we are going to place the object after picking it up and then calculate the variables a0, a1, a2 and a3 for this trajectory.

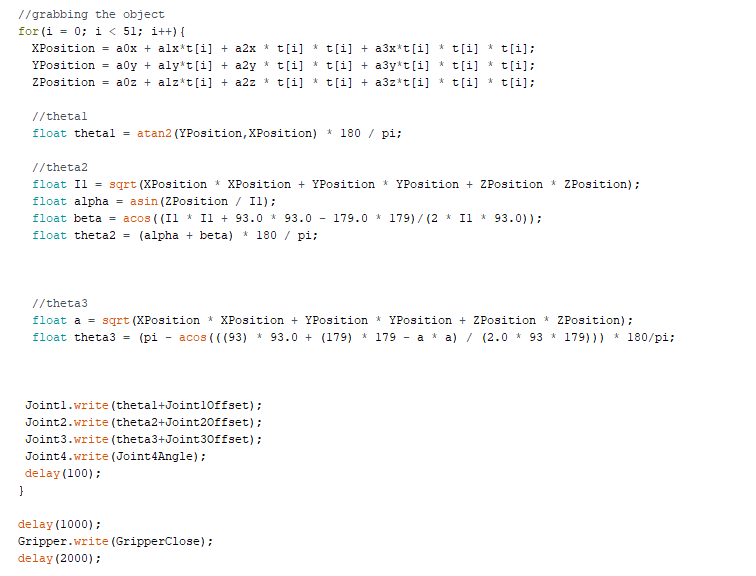
3

Text

Description automatically generated

1. After that we form a time vector that has 50 increments of 0.1 and it’s going to help us in calculating the positions and making the motion last for 5 second.

4



5

1. We then use for loop to make the robot move. To do so we calculate XPosition, YPostion, ZPosition using the equation for each t[i] of the array that we previously made, after which we calculate the angles theta1, theta2 and theta3 using inverse kinematics. After having the angles we use “Joint.write()” for each joint to move and very importantly, at the end of each loop we have a delay of 100 ms, as we want the whole motion to last for 5 seconds and there are 50 loops, thus 50 \* 100 ms = 5 s. After the for loop we get to the desired position, we wait one second, we close the gripper to pick up the object and then we wait another 2 seconds.

Text

Description automatically generated

1. To go back to the initial position, we apply the same logic but this time the loop goes from 50 to 0 as we are going backwards to where we started. We then wait for two second until the next action.

6

Text

Description automatically generated

1. We then want to place the object to a new place so we use again the same logic and the same loop, but now the variables for the final position change to the coordinates of the new end position. Again for each XPosition, YPosition, ZPosition we find the angles and then move the joints, each loop having a delay of 100ms. After reaching the wanted destination the gripper opens to drop the object.

7

Graphical user interface, text, application

Description automatically generated

7

1. In the end we bring the robot to its initial position with a loop that goes from 50 to 0, calculating the positions and angles each loop.

8

**Results**

We chose the starting points to be:

**Xstart = 0; Ystart = 171.5; Zstart = 97.5;**

The objects calculated position:

**Xobject = -89.94; Yobject = 221.94; Zobject = -20.02;**

The position for where the robot was to place the object:

**Xnew = 89.94; Ynew = Yobject; Znew = Zobject;**

Firstly, we input these values into MATLAB, to simulate the cubic trajectory, as can be seen in figure 5.0, we can also see the change of position and speed on each axis.

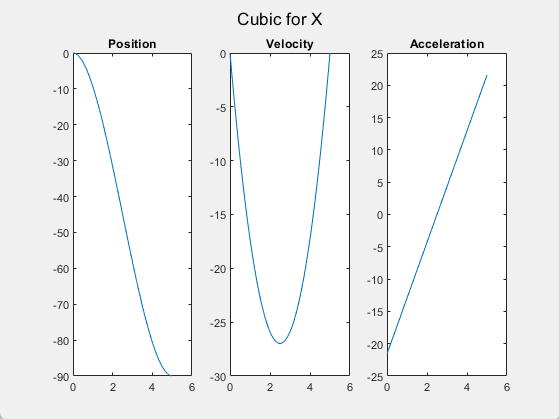
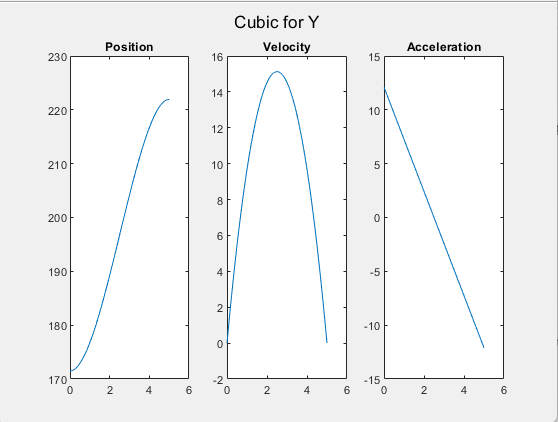


Figure 5.0 - MATLAB simulation of cubic trajectory for X, Y, Z axis

Then after uploading the Arduino code to the robot, we also serial print the values of the axis positions, X, Y, Z to observe the change for 50 increments from the initial position to the final position (figure 5.1), and then in another 50 moves back to the initial position (figure 5.2) and these value match the simulated values done in MATLAB.

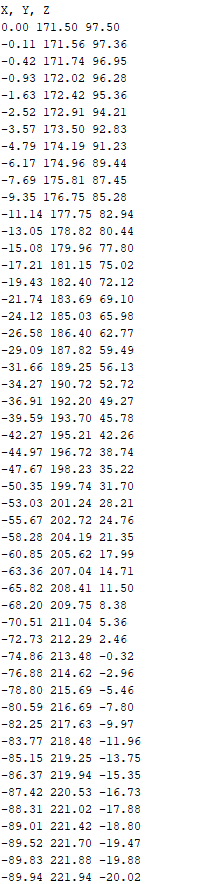
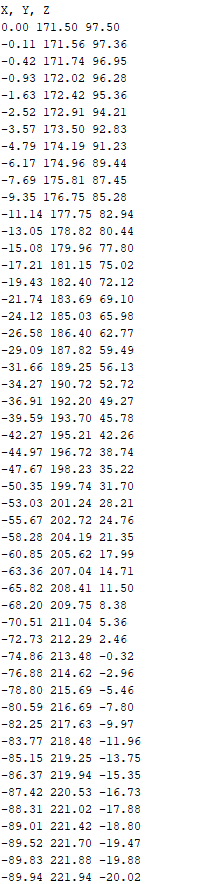


Figure 5.1 - Cubic trajectory positions moving back to the start

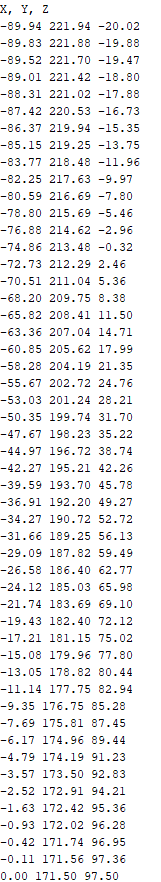
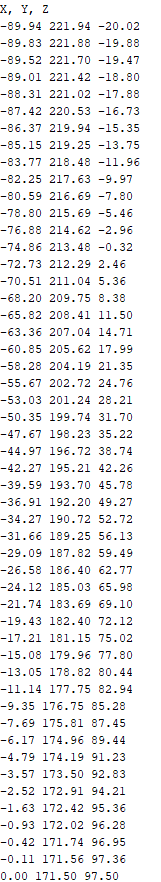


Figure 5.2 Cubic trajectory positions moving back to the start

For these sets of values the robot performed very well, as the robot moved smoothly throughout the whole motion, and the object was picked and dropped to the desired place successfully.