Computer Control in Robotics
CEG 4158 - Fall 2018 Project Report:

Automated Classification with Robotic Manipulation

Project Members:
Davies Michael
Ejinwunmi Korede
Orimoloye Abayomi – 7573985

1. Introduction

This report contains detailed information about a project for course CEG4158 carried out in the fall of 2018. The objective of this project was to transform the real physical world, bounded by constraints, between an initial state and a final state. More specifically, we implemented a robotic system that classified objects based on their shapes by means of manipulating these objects. The entire process; measurements gathering; perception; modelling; planning; and action, was automated since human intervention was prohibited.

2. Robot Modelling

To model the robot, we employed the Denavit-Hartenberg convention of attaching reference frames to the joints of the robot from the base to the end effector. Figure 1. shows a simple structure of the robot and its joints with axes attached to each joint, base, and end-effector in accordance with the rules of the D-H convention.

2.1.Denavit-Hartenberg Model and Parameters

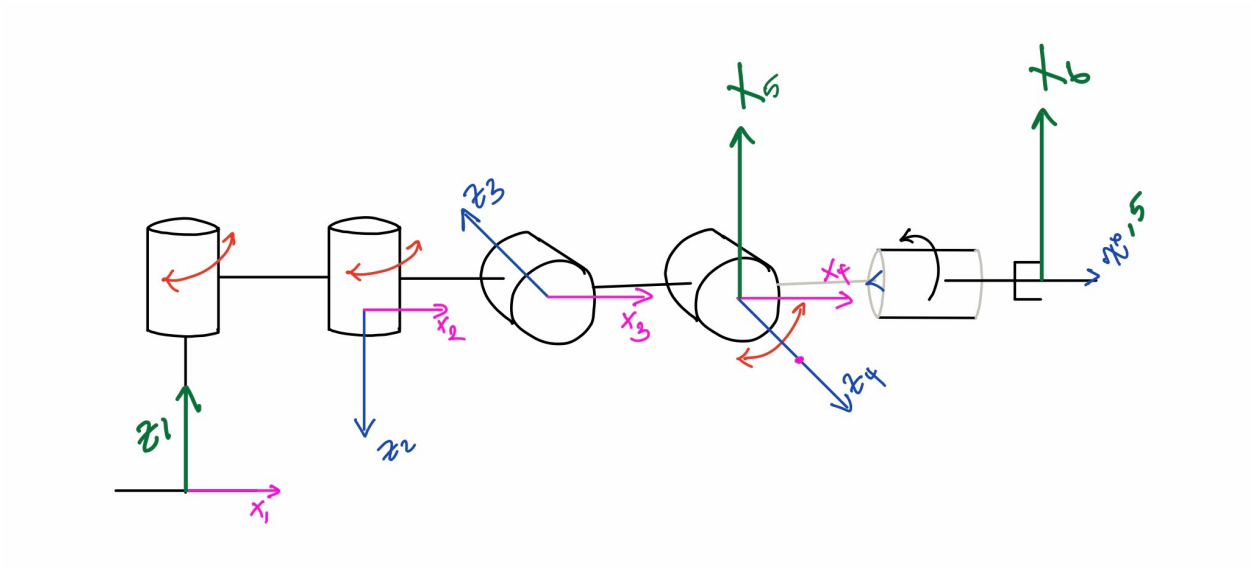

Figure 1. D-H figure with selected axes

[***Comment***: *For servo motors 1 and 5 corresponding to joints 1 and 5 above respectively, the z-axes were chosen to point in directions that are opposite to the actual positive rotations of the motors. This was done in order to select the most convenient reference frames to work with. To enable that the motors rotate in the correct directions during manipulation, we inverted the step values generated by our slope equations that will be discussed later in this report.*]

From figure 1, the D-H parameters required to model the system were derived. There are five joints in the robot, hence five sets of D-H parameters were derived as shown in table 1.

I	l_i (cm)	d_i (cm)	α_i (°)	θ_i (°)
1	l_1	d_1	180	θ_1

2	l_2	0	90	θ_2
3	l_3	0	180	θ_3
4	0	0	90	$90+\theta_4$
5	0	d_5	0	θ_5

Table 1. D-H Table showing D-H parameters

[***Comment***: *l_i and d_i values are left as constants for easy tuning and adjustments in code.*]

2.2. Forward Kinematics

The geometrical relationship that exists between each pair of successive joints in our robot's model above can be described by means of composed A matrices. Since we have five joints in our model, we must then have five A matrices. Equation 1 below gives the general model of the A matrix.

$$A_i = \begin{matrix} Cos\,\theta_i & -Sin\,\theta_i Cos\alpha_i & Sin\,\theta_i Sin\alpha_i & l_i Cos\,\theta_i \\ Sin\,\theta_i & Cos\,\theta_i Cos\alpha_i & -Cos\,\theta_i Sin\alpha_i & l_i Sin\,\theta_i \\ 0 & Sin\alpha_i & Cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{matrix}$$

(**eq.** 2.2.1)

From this general model, the five A matrices needed to manipulate our system were generated;

$$A_1 = \begin{matrix} Cos\,\theta_1 & Sin\,\theta_1 & 0 & l_1 Cos\,\theta_1 \\ Sin\,\theta_i & -Cos\,\theta_1 & 0 & l_1 Sin\,\theta_1 \\ 0 & 0 & -1 & d_1 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_2 = \begin{matrix} Cos\,\theta_2 & 0 & Sin\,\theta_2 & l_2 Cos\,\theta_2 \\ Sin\,\theta_2 & 0 & -Cos\,\theta_2 & l_2 Sin\,\theta_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_3 = \begin{matrix} Cos\,\theta_3 & Sin\,\theta_3 & 0 & l_3 Cos\,\theta_3 \\ Sin\,\theta_3 & -Cos\,\theta_3 & 0 & l_3 Sin\,\theta_3 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_4 = \begin{matrix} Cos\,(90+\theta_4) & 0 & Sin\,(90+\theta_4) & 0 \\ Sin\,(90+\theta_4) & 0 & -Cos\,(90+\theta_4) & 0 \\ 0 & 1 & Cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$A_5 = \begin{matrix} Cos\,\theta_5 & -Sin\,\theta_5 & 0 & 0 \\ Sin\,\theta_5 & Cos\,\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{matrix}$$

(**eq.** 2.2.2)

Then, by multiplying all of matrices $A_{1\text{-}5}$ together, the equivalent matrix describing the *forward kinematics* in position and orientation of the end effector relative to the base of the robot was generated;

$$Q_{\text{end-effector/Base}} = A = A1 * A2 * A3 * A4 * A5$$

$$A = \begin{matrix} a_x & b_x & c_x & P_x \\ a_y & b_y & c_y & P_y \\ a_z & b_z & c_z & P_z \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$a_x = Cos\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 + Sin\,(\theta_1 - \theta_2)Sin\,\theta_5$$
$$a_y = Sin\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 - Cos\,(\theta_1 - \theta_2)Sin\,\theta_5$$
$$a_z = Sin(90 + \theta_4 - \theta_3)Cos\,\theta_5$$

$$b_x = -Cos\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Sin\,\theta_5 + Sin\,(\theta_1 - \theta_2)Cos\,\theta_5$$
$$b_y = -Sin\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Sin\,\theta_5 - Cos\,(\theta_1 - \theta_2)Cos\,\theta_5$$
$$b_z = -Sin(90 + \theta_4 - \theta_3)Sin\,\theta_5$$

$$c_x = Cos\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)$$
$$c_y = Sin\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)$$
$$c_z = -Cos(\theta_3 - 90 - \theta_4)$$

$$P_x = d_5Cos\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) + l_3Cos\,(\theta_1 - \theta_2)Cos(\theta_3) + l_2Cos\,(\theta_1 - \theta_2) + l_1Cos\,(\theta_1)$$
$$P_y = d_5Sin\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) + l_3Sin\,(\theta_1 - \theta_2)Cos(\theta_3) + l_2Sin\,(\theta_1 - \theta_2) + l_1Sin(\theta_1)$$
$$P_z = -d_5Cos(\theta_3 - 90 - \theta_4) - l_3Sin\,(\theta_3) + d_1$$

(**eq.** 2.2.3)

Solving these matrices will give degree values for θ_1-θ_5. However, the motors of the ROBIX respond to inputs in units of steps not degrees. Hence, conversion equations were necessary to convert the theta values from degrees to steps.

The conversion between degrees and steps can be modelled by a linear equation:

$$y = mx + b$$

where y and x are the angular displacement of the motor in degrees and steps respectively. Since all five motors have distinct limitations on their angular displacements, there needed to be a different linear equation for each motor. Given the following assumptions;

- At home configuration (steps = 0), each motor has an angular displacement of 0°

- The maximum and minimum number of steps attainable by each motor is 1400 and -1400, ($x_1, and\ x_2$), respectively. Each corresponding to the maximum and minimum attainable angular displacement, ($y_1, and\ y_2$), respectively (in °) for each motor, as measured with a ruler and protractor.

The linear equation for each motor was derived;

Servo 1:	$(x_1, y_1, x_2, y_2) = (1400, 77.5, -1400, -77.5) \rightarrow x_{servo1} = \frac{-y_{servo1}}{0.0554}$
Servo 2:	$(x_1, y_1, x_2, y_2) = (1400, 60, -1400, -60) \rightarrow x_{servo2} = \frac{y_{servo2}}{0.0429}$
Servo 3:	$(x_1, y_1, x_2, y_2) = (1400, 77.5, -1400, -77.5) \rightarrow x_{servo3} = \frac{y_{servo3}}{0.0554}$
Servo 4:	$(x_1, y_1, x_2, y_2) = (1400, 82.5, -1400, -82.5) \rightarrow x_{servo4} = \frac{y_{servo4}}{0.0589}$
Servo 5:	$(x_1, y_1, x_2, y_2) = (1400, 75, -1400, -75) \rightarrow x_{servo5} = \frac{-y_{servo5}}{0.0536}$

(**eq.** 2.2.4)

As was commented on earlier in figure 1, the step equations for servos 1 and 5 have been inverted to yield rotational motion in the correct direction.

2.3. Implementation and Validation of Forward Kinematics

Utilizing the A matrices and slope equations above, the forward kinematics was implemented in a Matlab script. Code snippet 1 (cs. 2.2.1) shows the code used to generate the forward kinematics model by automatically multiplying each A matrix given by [eq. 2.2.2].

[cs. 2.2.2] also generates the forward kinematics model. However, the set of equations in [eq. 2.2.3] were used. This was done to verify that no error was made during the manual multiplication of the A matrices which were further simplified using known trigonometric identities. This was necessary since these equations were later used to solve the inverse kinematics of the system. (**cs**. 2.2.1) was sufficient in computing the forward kinematics for the project.

To verify the forward kinematics model, sample values were assigned to θ_1 -θ_5. For example, we set (θ_1, θ_2, θ_3, θ_4, θ_5) to (25°, 25°, 25°, 25°, 25°) and got a $Q_{\text{end-effector/Base}}$ matrix. Using our slope equations, we generated the equivalent step values of the motors to be (-451, 583, 451, 424, -466). These values are approximated to the nearest integer since the ROBIX only responds to integer step values.

$$Q_{\text{end-effector/Base}} = \begin{matrix} 0 & 0 & 1 & 34.5009 \\ -0.4226 & -0.9063 & 0 & 4.2262 \\ 0.9063 & -0.4226 & 0 & 9.9643 \\ 0 & 0 & 0 & 1 \end{matrix}$$

The ROBIX was programmed with the step values derived using code snippet [cs. 2.2.3].

```
Move all to 0;
Move 1 to -451
Move 2 to 583
Move 1 to 451
Move 1 to 424
Move 1 to -466
```

(**cs.** 2.2.3)

The position of the end-effector relative to the base of the robot was observed. As expected, relative to the base of the robot, the end-effector was located at (x, y, z) positions approximately equal to (34.5009cm, 4.2262cm, 9.9643cm). The orientation of the end-effector was also as expected.

3. Inverse Kinematics

To solve the inverse kinematics, equations were derived for the joint coordinates of the robot (θ_1, θ_2, θ_3, θ_4, θ_5) that allow it to reach a given position and orientation specified by the end-effector.

Using the structure developed in [eq. 2.2.3], we can estimate the value of each coordinate;

$$Q_{\text{end-effector/Base}} = \text{A} = \text{A1} * \text{A2} * \text{A3} * \text{A4} * \text{A5}$$

$$A = \begin{matrix} a_x & b_x & c_x & P_x \\ a_y & b_y & c_y & P_y \\ a_z & b_z & c_z & P_z \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$a_x = Cos\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 + Sin\,(\theta_1 - \theta_2)Sin\,\theta_5$$
$$a_y = Sin\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 - Cos\,(\theta_1 - \theta_2)Sin\,\theta_5$$
$$a_z = Sin(90 + \theta_4 - \theta_3)Cos\,\theta_5$$

$$b_x = -Cos\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Sin\,\theta_5 + Sin\,(\theta_1 - \theta_2)Cos\,\theta_5$$
$$b_y = -Sin\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Sin\,\theta_5 - Cos\,(\theta_1 - \theta_2)Cos\,\theta_5$$
$$b_z = -Sin(90 + \theta_4 - \theta_3)Sin\,\theta_5$$

$$c_x = Cos\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)$$
$$c_y = Sin\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)$$
$$c_z = -Cos(\theta_3 - 90 - \theta_4)$$

$$P_x = KCos\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) + 6Cos\,(\theta_1 - \theta_2)Cos(\theta_3) + 10Cos\,(\theta_1 - \theta_2) + 10Cos\,(\theta_1)$$
$$P_y = KSin\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) + 6Sin\,(\theta_1 - \theta_2)Cos(\theta_3) + 10Sin\,(\theta_1 - \theta_2) + 10Sin(\theta_1)$$
$$P_z = -KCos(\theta_3 - 90 - \theta_4) - 6Sin\,(\theta_3) + 12.5$$

(**eq.** 2.2.3)

For θ_3,

We have;

$$P_z = -KCos(\theta_3 - 90 - \theta_4) - 6Sin\,(\theta_3) + 12.5 \quad \text{(i)}$$
$$c_z = -Cos(\theta_3 - 90 - \theta_4) \quad \text{(ii)}$$

Substituting (ii) in (i) we get;

$$P_z = Kc_z - 6Sin\,(\theta_3) + 12.5$$
$$-\frac{P_z - Kc_z - 12.5}{6} = Sin\,(\theta_3)$$
$$\theta_3 = Sin^{-1}\left(\frac{-P_z + Kc_z + 12.5}{6}\right)$$

(**eq**. 3.1)

[***Comment:*** *It's important to note that this equation yields two possible values for* θ_3]

For θ_4,

We have;

$$c_x = Cos\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)$$
$$c_y = Sin\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)$$

Then,

$$c_x^2 = Cos^2(\theta_1 - \theta_2)Sin^2(90 + \theta_4 - \theta_3)$$
$$c_y^2 = Sin^2(\theta_1 - \theta_2)Sin^2(90 + \theta_4 - \theta_3)$$
$$c_x^2 + c_y^2 = Cos^2(\theta_1 - \theta_2)Sin^2(90 + \theta_4 - \theta_3) + Sin^2(\theta_1 - \theta_2)Sin^2(90 + \theta_4 - \theta_3)$$
$$c_x^2 + c_y^2 = Sin^2(90 + \theta_4 - \theta_3)\,(Cos^2(\theta_1 - \theta_2) + Sin^2(\theta_1 - \theta_2))$$

But,

$$Cos^2(\theta_1 - \theta_2) + Sin^2(\theta_1 - \theta_2) = 1$$

Then,

$$c_x^2 + c_y^2 = Sin^2(90 + \theta_4 - \theta_3) = Cos^2(\theta_4 - \theta_3)$$
$$\theta_4 - \theta_3 = Cos^{-1}\left(\sqrt{c_x^2 + c_y^2}\right)$$
$$\theta_4 = Cos^{-1}\left(\sqrt{c_x^2 + c_y^2}\right) + \theta_3$$

(**eq**. 3.2)

For θ_5,

We have;

$$a_z = Sin(90 + \theta_4 - \theta_3)Cos\,\theta_5$$
$$b_z = -Sin(90 + \theta_4 - \theta_3)Sin\,\theta_5$$

Then,

$$\frac{b_z}{a_z} = \frac{-Sin(90 + \theta_4 - \theta_3)Sin\,\theta_5}{Sin(90 + \theta_4 - \theta_3)Cos\,\theta_5} = -tan\theta_5$$
$$tan\theta_5 = \frac{b_z}{-a_z}$$
$$\theta_5 = atan2\left(\frac{b_z}{-a_z}\right)$$

(**eq**. 3.3)

[***Comment:*** *Although atan2() gives us a precise solutions for* θ_5*, complications may arise if* $a_z = 0$*. For this reason, an alternative equation for* θ_5 *was derived*]

For θ_5 (alternative solution),

We have;

$$a_x = Cos\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 + Sin\,(\theta_1 - \theta_2)Sin\,\theta_5$$
$$a_y = Sin\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 - Cos\,(\theta_1 - \theta_2)Sin\,\theta_5$$

Then,

$$\begin{aligned} a_x^2 = {} & Cos^2(\theta_1 - \theta_2)Cos^2(\theta_3 - 90 - \theta_4)Cos^2\theta_5 \\ & + 2Cos(\theta_1 - \theta_2)Sin(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\theta_5 Sin\theta_5 \\ & + Sin^2(\theta_1 - \theta_2)Sin^2\theta_5 \end{aligned}$$

$$\begin{aligned} a_y^2 = {} & Sin^2(\theta_1 - \theta_2)Cos^2(\theta_3 - 90 - \theta_4)Cos^2\theta_5 \\ & - 2Cos(\theta_1 - \theta_2)Sin(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\theta_5 Sin\theta_5 \\ & + Cos^2(\theta_1 - \theta_2)Sin^2\theta_5 \end{aligned}$$

$$\begin{aligned} a_x^2 + a_y^2 = {} & Cos^2(\theta_3 - 90 - \theta_4)Cos^2\theta_5(Cos^2(\theta_1 - \theta_2) + Sin^2(\theta_1 - \theta_2)) \\ & + Sin^2\theta_5(Cos^2(\theta_1 - \theta_2) + Sin^2(\theta_1 - \theta_2)) \end{aligned}$$

But,

$$Cos^2(\theta_1 - \theta_2) + Sin^2(\theta_1 - \theta_2) = 1$$

Then,

$$a_x^2 + a_y^2 = Cos^2(\theta_3 - 90 - \theta_4)Cos^2\theta_5 + Sin^2\theta_5$$

Notice that,

$$c_z^2 = Cos^2(\theta_3 - 90 - \theta_4)$$

And,

$$Sin^2\theta_5 = 1 - Cos^2\theta_5$$

Then,

$$a_x^2 + a_y^2 = c_z^2 Cos^2\theta_5 + 1 - Cos^2\theta_5$$
$$a_x^2 + a_y^2 - 1 = (c_z^2 - 1)Cos^2\theta_5$$
$$Cos^2\theta_5 = \frac{a_x^2 + a_y^2 - 1}{c_z^2 - 1}$$
$$\theta_5 = Cos^{-1}\left(\sqrt{\frac{a_x^2 + a_y^2 - 1}{c_z^2 - 1}}\right)$$

(**eq**. 3.4)

[***Comment:*** *This equation gives more solutions (precisely 4) for θ_5. However, complications could still arise if $c_z = 1$. In the case where $c_z = 1$ and $a_z = 0$, this means that there is no roll and the end-effector is pointing directly upwards. Such a configuration for the end effector was undesirable and may never be reached.*]

For θ_2,

We have;

$$P_x = KCos\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) + 6Cos\,(\theta_1 - \theta_2)Cos(\theta_3) + 10Cos\,(\theta_1 - \theta_2) + 10Cos\,(\theta_1)$$
$$P_y = KSin\,(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) + 6Sin\,(\theta_1 - \theta_2)Cos(\theta_3) + 10Sin\,(\theta_1 - \theta_2) + 10Sin(\theta_1)$$

Then,

$$\begin{aligned} P_x^2 = {} & K^2Cos^2(\theta_1 - \theta_2)Sin^2(90 + \theta_4 - \theta_3) + 12KCos^2(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)Cos(\theta_3) \\ & + 20KCos^2(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3) \\ & + 20KCos(\theta_1 - \theta_2)Sin(90 + \theta_4 - \theta_3)Cos(\theta_1) + 36Cos^2(\theta_1 - \theta_2)Cos^2(\theta_3) \\ & + 120Cos^2(\theta_1 - \theta_2)Cos(\theta_3) + 120Cos(\theta_1 - \theta_2)Cos(\theta_3)Cos(\theta_1) \\ & + 100Cos^2(\theta_1 - \theta_2) + 200Cos(\theta_1 - \theta_2)Cos(\theta_1) + 100Cos^2(\theta_1) \end{aligned}$$

$$P_y^2 = K^2Sin^2(\theta_1-\theta_2)Sin^2(90+\theta_4-\theta_3) + 12KSin^2(\theta_1-\theta_2)Sin(90+\theta_4-\theta_3)Cos(\theta_3) + 20KSin^2(\theta_1-\theta_2)Sin(90+\theta_4-\theta_3) + 20KSin(\theta_1-\theta_2)Sin(90+\theta_4-\theta_3)Sin(\theta_1) + 36Sin^2(\theta_1-\theta_2)Cos^2(\theta_3) + 120Sin^2(\theta_1-\theta_2)Cos(\theta_3) + 120Sin(\theta_1-\theta_2)Cos(\theta_3)Sin(\theta_1) + 100Sin^2(\theta_1-\theta_2) + 200Sin(\theta_1-\theta_2)Sin(\theta_1) + 100Sin^2(\theta_1)$$

$$P_x^2 + P_y^2 = K^2Sin^2(90+\theta_4-\theta_3) + 12KSin(90+\theta_4-\theta_3)Cos(\theta_3) + 20KSin(90+\theta_4-\theta_3) + 20KCos(\theta_1-\theta_2)Sin(90+\theta_4-\theta_3)Cos(\theta_1) + 20KSin(\theta_1-\theta_2)Sin(90+\theta_4-\theta_3)Sin(\theta_1) + 36Cos^2(\theta_3) + 120Cos(\theta_3) + 120Cos(\theta_1-\theta_2)Cos(\theta_3)Cos(\theta_1) + 120Sin(\theta_1-\theta_2)Cos(\theta_3)Sin(\theta_1) + 100 + 200Cos(\theta_1-\theta_2)Cos(\theta_1) + 200Sin(\theta_1-\theta_2)Sin(\theta_1) + 100$$

After simplification with Trig identities, we get;

$$P_x^2 + P_y^2 = K^2Cos^2(\theta_4-\theta_3) + 12KCos(\theta_4-\theta_3)Cos(\theta_3) + 20KCos(\theta_4-\theta_3) + 36Cos^2(\theta_3) + 120Cos(\theta_3) + 200 + Cos(\theta_2)[20KCos(\theta_4-\theta_3) + 120Cos(\theta_3) + 200]$$

Then,

$$P_x^2 + P_y^2 - K^2Cos^2(\theta_4-\theta_3) - 12KCos(\theta_4-\theta_3)Cos(\theta_3) - 20KCos(\theta_4-\theta_3) - 36Cos^2(\theta_3) - 120Cos(\theta_3) - 200 = Cos(\theta_2)[20KCos(\theta_4-\theta_3) + 120Cos(\theta_3) + 200]$$

$$Cos(\theta_2) = \frac{P_x^2 + P_y^2 - K^2Cos^2(\theta_4-\theta_3) - 12KCos(\theta_4-\theta_3)Cos(\theta_3) - 20KCos(\theta_4-\theta_3) - 36Cos^2(\theta_3) - 120Cos(\theta_3) - 200}{[20KCos(\theta_4-\theta_3) + 120Cos(\theta_3) + 200]}$$

$$\theta_2 = \cos^{-1}\left(\frac{P_x^2 + P_y^2 - K^2Cos^2(\theta_4-\theta_3) - 12KCos(\theta_4-\theta_3)Cos(\theta_3) - 20KCos(\theta_4-\theta_3) - 36Cos^2(\theta_3) - 120Cos(\theta_3) - 200}{[20KCos(\theta_4-\theta_3) + 120Cos(\theta_3) + 200]}\right)$$

(**eq**. 3.5)

For θ_1,

We have;

$$b_x = -Cos\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5 + Sin\,(\theta_1-\theta_2)Cos\,\theta_5$$

$$b_y = -Sin\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5 - Cos\,(\theta_1-\theta_2)Cos\,\theta_5$$

Then,

$$\frac{b_y}{b_x} = \frac{-Sin\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5 - Cos\,(\theta_1-\theta_2)Cos\,\theta_5}{-Cos\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5 + Sin\,(\theta_1-\theta_2)Cos\,\theta_5}$$

$$\frac{b_y}{b_x} = \frac{\frac{-Sin\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5}{-Cos\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5} - \frac{Cos\,(\theta_1-\theta_2)Cos\,\theta_5}{-Cos\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5}}{1 + \frac{Sin\,(\theta_1-\theta_2)Cos\,\theta_5}{-Cos\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5}}$$

$$\frac{b_y}{b_x} = \frac{\frac{Sin\,(\theta_1-\theta_2)}{Cos\,(\theta_1-\theta_2)} + \frac{Cos\,\theta_5}{Cos(\theta_3-90-\theta_4)Sin\,\theta_5}}{1 - \frac{Sin\,(\theta_1-\theta_2)Cos\,\theta_5}{Cos\,(\theta_1-\theta_2)Cos(\theta_3-90-\theta_4)Sin\,\theta_5}}$$

$$\frac{b_y}{b_x} = \frac{tan(\theta_1-\theta_2) + \frac{1}{Cos(\theta_3-90-\theta_4)tan\theta_5}}{1 - \frac{tan(\theta_1-\theta_2)}{Cos(\theta_3-90-\theta_4)tan\theta_5}}$$

Using the trig Identity,

$$tan(a+b) = \frac{tan(a) + \tan(b)}{1 - \tan(a)\tan(b)}$$

We see that,

$$a = \theta_1 - \theta_2$$

$$b = \text{atan2}\left(\frac{1}{Cos(\theta_3 - 90 - \theta_4)tan\theta_5}\right)$$

Then,

$$\frac{b_y}{b_x} = tan(\text{a} + b)$$

$$\text{a} + b = atan2\left(\frac{b_y}{b_x}\right)$$

$$\text{a} = atan2\left(\frac{b_y}{b_x}\right) - b$$

$$\theta_1 - \theta_2 = atan2\left(\frac{b_y}{b_x}\right) - \text{atan2}\left(\frac{1}{Cos(\theta_3 - 90 - \theta_4)tan\theta_5}\right)$$

$$\theta_1 = atan2\left(\frac{b_y}{b_x}\right) - \text{atan2}\left(\frac{1}{Cos(\theta_3 - 90 - \theta_4)tan\theta_5}\right) + \theta_2$$

(**eq**. 3.6)

Upon using the equations above to test our real life work space, it was soon discovered that some of the equations presented above would not be sufficient for the scenarios we were give. To be precise, The values for θ_5 would be undesirable. The objects are placed in the work space and the reference frames for the objects, targets and robot were chosen such that there is a complete 180 degree rotation <u>of</u> the z axis when going from the base of the robot to the end effector when picking up or dropping off an object. This meant that a_z, b_z, c_x, *and* c_y would always result to 0. A new equation for θ_5 needed to be developed and as a consequence, a new equation for θ_1 was also developed. This was done by using math techniques learned in class as shown below.

<u>For θ_5 (alternative solution),</u>

We have;

$$a_x = Cos\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 + Sin\,(\theta_1 - \theta_2)Sin\,\theta_5$$

$$a_y = Sin\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 - Cos\,(\theta_1 - \theta_2)Sin\,\theta_5$$

Then,

$$a_x Cos\,(\theta_1 - \theta_2) = Cos^2\,(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 + Cos\,(\theta_1 - \theta_2)Sin\,(\theta_1 - \theta_2)Sin\,\theta_5$$

$$a_y Sin\,(\theta_1 - \theta_2) = Sin^2(\theta_1 - \theta_2)Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5 - Cos\,(\theta_1 - \theta_2)Sin\,(\theta_1 - \theta_2)Sin\,\theta_5$$

$$a_x Cos\,(\theta_1 - \theta_2) + a_y Sin\,(\theta_1 - \theta_2) = Cos(\theta_3 - 90 - \theta_4)Cos\,\theta_5$$

Then,

$$\frac{a_x Cos\,(\theta_1 - \theta_2) + a_y Sin\,(\theta_1 - \theta_2)}{Cos(\theta_3 - 90 - \theta_4)} = Cos\,\theta_5$$

Then,

$$\theta_5 = Cos^{-1}\left(\frac{a_x Cos\,(\theta_1 - \theta_2) + a_y Sin\,(\theta_1 - \theta_2)}{Cos(\theta_3 - 90 - \theta_4)}\right)$$

Similarly, θ_1 was developed as;

$$\theta_1 = atan2\left(\frac{P_y}{P_x}\right) - \text{atan2}\left(\frac{sin\theta_2(l_3 cos\theta_3 + l_2)}{cos\theta_2(l_3 cos\theta_3 + l_2) + l_1)}\right)$$

equations derived for the joint coordinates of the robot (θ_1, θ_2, θ_3, θ_4, θ_5) are summarized in [eq. 3.7] below;

$$\theta_1 = atan2\left(\frac{P_y}{P_x}\right) - \text{atan2}\left(\frac{sin\theta_2(l_3 cos\theta_3 + l_2)}{cos\theta_2(l_3 cos\theta_3 + l_2) + l_1)}\right)$$

$$\theta_2 = \cos^{-1}\left(\frac{P_x^2 + P_y^2 - {d_5}^2 Cos^2(\theta_4 - \theta_3) - 2l_3 d_5 Cos(\theta_4 - \theta_3)Cos(\theta_3) - 2l_2 d_5 Cos(\theta_4 - \theta_3) - {l_3}^2 Cos^2(\theta_3) - 2l_3 l_2 Cos(\theta_3) - {l_1}^2 - {l_2}^2}{[2l_1 d_5 Cos(\theta_4 - \theta_3) + 2l_3 l_1 Cos(\theta_3) + 2l_1 l_2]}\right)$$

$$\theta_3 = Sin^{-1}\left(\frac{-P_z + d_5 c_z + d_1}{l_3}\right)$$

$$\theta_4 = Cos^{-1}\left(\sqrt{c_x^2 + c_y^2}\right) + \theta_3$$

$$\theta_5 = Cos^{-1}\left(\frac{a_x Cos\,(\theta_1 - \theta_2) + a_y Sin\,(\theta_1 - \theta_2)}{Cos(\theta_3 - 90 - \theta_4)}\right)$$

(**eq**. 3.7)

[***Comment:*** *When implementing these equations in MATLAB, the order in which the theta values are derived is important as some of the theta values depend on other theta values. Ideally, θ_3 would be derived first, then θ_4,* then *θ_2, then θ_1, and finally θ_5.]*

3.1.Implementation and Validation of Inverse Kinematics

The block diagram in figure 2 gives an overview of the inputs and outputs of the inverse kinematics function, ***getInverseParams***, developed to solve the inverse kinematics of the system.

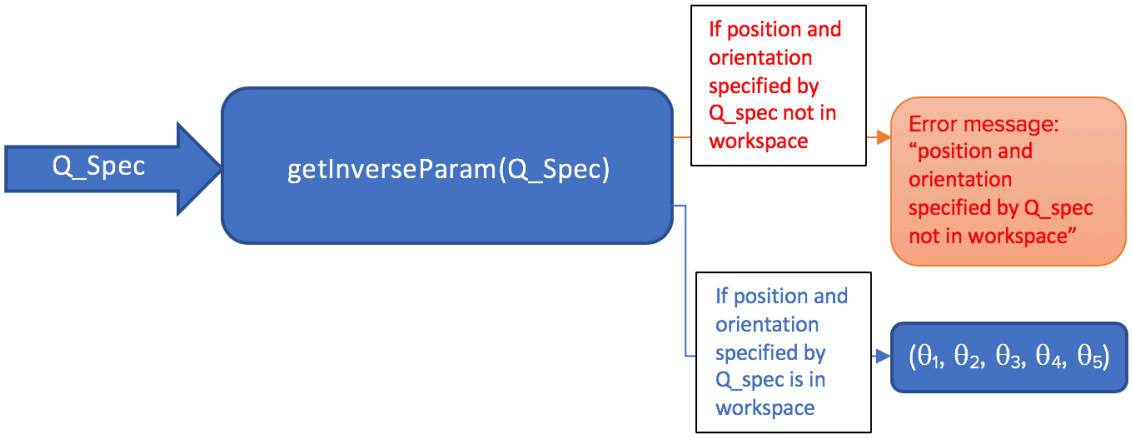

Figure 2. Block diagram showing I/O for inverse kinematics function

The function, given an input matrix Q_Spec specifying the desired position and orientation of the end-effector, attempts to solve the inverse kinematics using the equations in [eq. 3.7]. The code snippet [cs. 3.1.1] in the appendix shows the MATLAB code used to perform this operation.

The equations for the joint coordinates in [eq. 3.7] give multiple solutions to each theta value. But not all solutions yield the desired result. To ensure that only the desired theta values are used and given at the output, two operations we set in place.

Firstly, after each theta value is calculated the resulting vector containing the possible values for that theta (for example, theta3 in the code above could be a vector containing two possible values for theta3) is passed through a function ***"checkThetaRange_and_Complex(thetaValues, whichTheta)"***.

checkThetaRange_and_Complex receives two inputs; a vector "***thetaValues***" containing the calculated solutions for a theta value specified by integer "***whichTheta***". Since each motor has a range of theta values it can physically attain, ***checkThetaRange_and_Complex*** filters out values not within this range and/or complex values, stores the remaining values in a vector ***newThetaValues*** and returns them to be used in subsequent calculations. [cs. 3.1.2] shows the implementation of ***checkThetaRange_and_Complex***.

Secondly, after all theta values have been calculated and filtered through ***checkThetaRange_and_Complex***, all possible combinations of thetas 1 through 5 are used in the forward kinematics equations in [eq. 2.2.3] or [cs. 2.2.2]. For each combination, a new matrix Q_new is generated and compared to Q_Spec. when Q_new is approximately equal to Q_Spec (both round to two decimal places), the theta values responsible for generating this Q_new are saved and returned as the desired solution. In the event that no combination could yield the desired result, an error message Is printed

signifying that the desired matrix Q_Spec has specified a position and orientation of the end-effector that is unreachable and out of the work-space.

In the event that Q_Spec is within the workspace and joint parameters to reach the desired position and orientation are computed at the output, then the computed joint parameters are passed through a function ***`getMotorStepValues`***. ***`getMotorStepValues`*** calculates the corresponding step values for each theta using slope equations in [eq. 2.2.4]. [cs. 3.1.3] shows the implementation of ***`getMotorStepValues`***.

To verify our inverse kinematics model, we tested the function ***`getInverseParams`*** in 3 test scenarios;

- Scenario 1: Input to ***`getInverseParams`*** is Q_spec generated using forward kinematics function with all joint parameters within the rotation limits of the motors (hence within the workspace)

  For example,

$$(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5) = (25,25,25,25,25) \rightarrow [getForwardMatrix] \rightarrow Q_{Spec} = \begin{matrix} 0 & 0 & 1 & 34.5009 \\ -0.4226 & -0.9063 & 0 & 4.2262 \\ 0.9063 & -0.4226 & 0 & 9.9643 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$Q_{Spec} = \begin{matrix} 0 & 0 & 1 & 34.5009 \\ -0.4226 & -0.9063 & 0 & 4.2262 \\ 0.9063 & -0.4226 & 0 & 9.9643 \\ 0 & 0 & 0 & 1 \end{matrix} \rightarrow [getInverseParams] \rightarrow (\theta_1,\theta_2,\theta_3,\theta_4,\theta_5) = (25,25,25,25,25)$$

  As expected, the correct values for the joint parameters were given at the output

- Scenario 2: Input to ***`getInverseParams`*** is Q_spec generated using forward kinematics function with all joint parameters outside the rotation limits of the motors (hence outside the workspace)

  For example,

$$(125,-105,95,205,205) \rightarrow [getForwardMatrix] \rightarrow Q_{Spec} = \begin{matrix} -0.2237 & 0.9495 & 0.2198 & -9.6290 \\ -0.9241 & -0.2783 & 0.2620 & 3.5517 \\ 0.3100 & -0.1445 & 0.9397 & 15.9198 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$Q_{Spec} = \begin{matrix} -0.2237 & 0.9495 & 0.2198 & -9.6290 \\ -0.9241 & -0.2783 & 0.2620 & 3.5517 \\ 0.3100 & -0.1445 & 0.9397 & 15.9198 \\ 0 & 0 & 0 & 1 \end{matrix} \rightarrow [getInverseParams] \rightarrow Error\ Message$$

  As expected, values were not computed for the joint parameters and an error message was given because Q_Spec specified a position and orientation of the end-effector outside of the workspace

- Scenario 3: Input to ***`getInverseParams`*** is Q_spec generated using forward kinematics function with joint parameters within the rotation limits for some motors while other joint parameters were outside the limits of the motors. This yielded the same result as scenario 2 as expected.

4. Image Processing

For our image processing, we used the following sample image as our test image

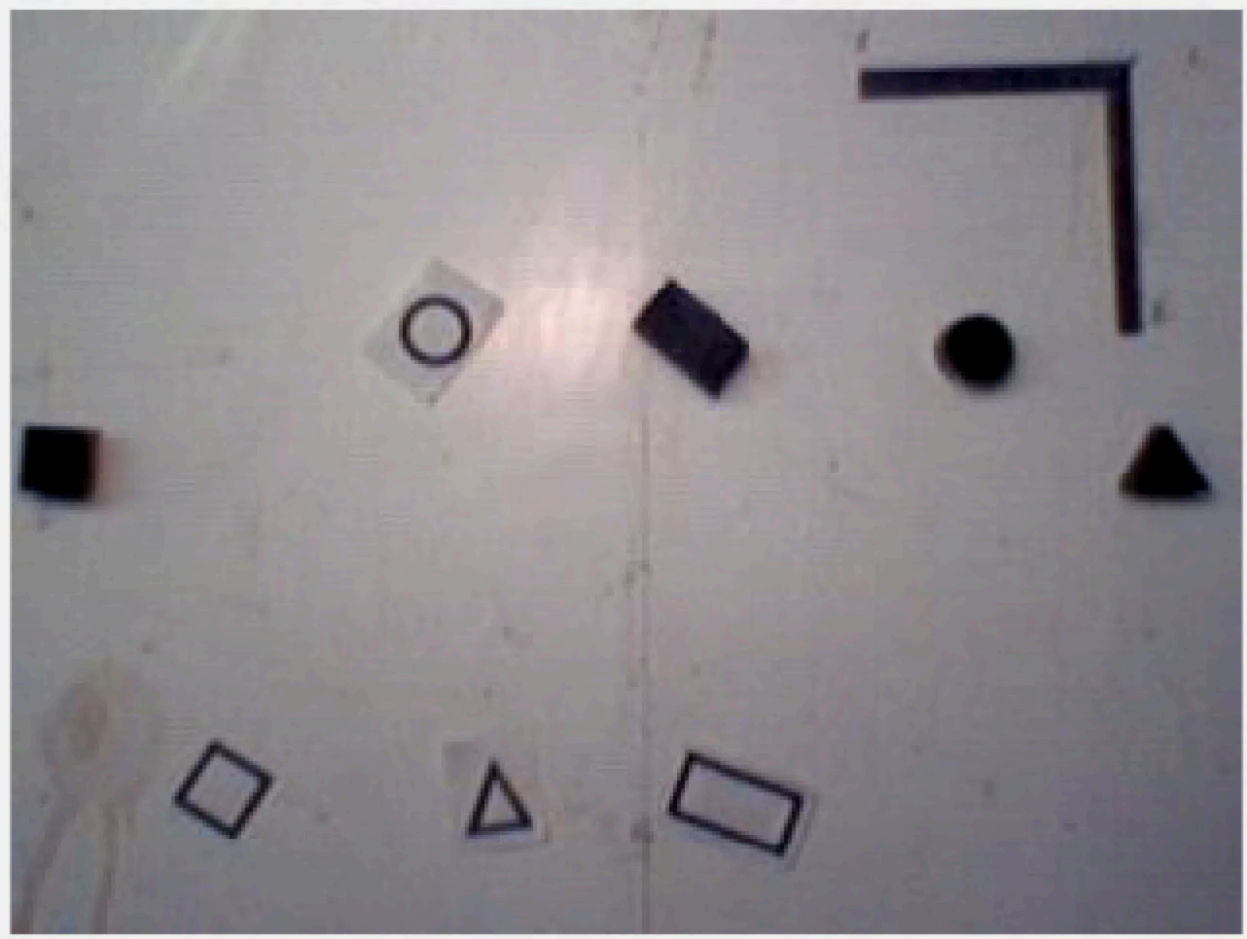

Figure 3. Sample Image

The image was loaded into the workspace and converted into a grayscale image using the *rgb2gray() function*. We then set a threshold value in order to convert our sample image into a binary image. This threshold value was dynamic and occasionally adjusted a few times during the demo by reason of the amount of light intensity on the sample image. In order to be able to take measurements of each object in the image, we labelled each object in the workspace using the *bwlabel* function and inserted the labelled objects into the *regionprops* function in order to get all the measurements for all the labelled objects identified in the image.

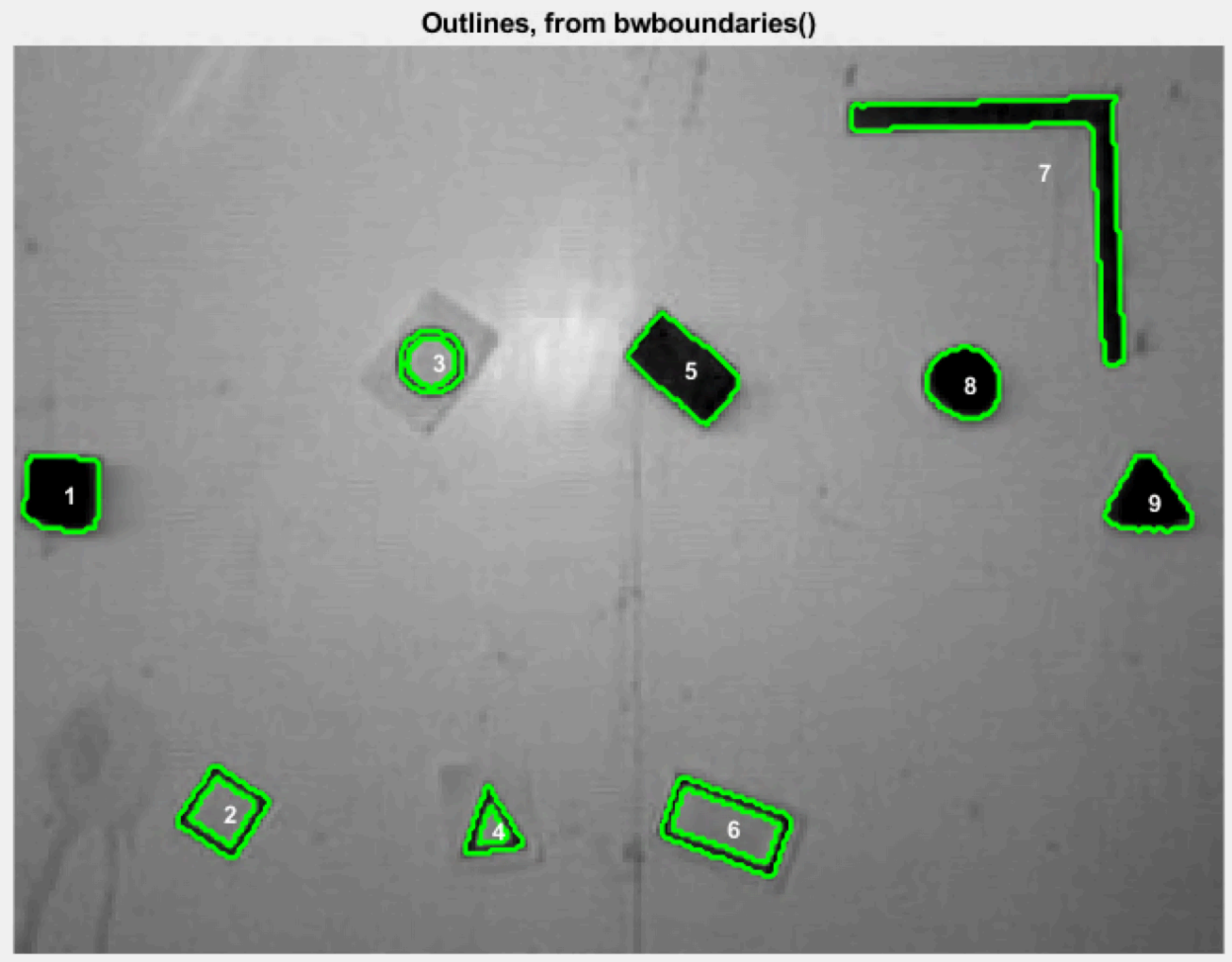

Figure 4. Outlines each object with their specific ids.

In order to convert from pixels to the real world, we measured the size of our reference frame in the lab (given by the 'L' shaped image in figure**) to be 10cm. We then measured the same L shaped reference in the image in pixels to be 72 pixels. This value was gotten by using region props to get the dimensions of the bounding box of the L shaped reference. The height of this bounding box is approximately the length of the reference provided the picture was taken such that the top of the L is parallel to the top edge of the picture.. This lead us to a scaling factor of *10/72* from pixels to cm which we used throughout the rest of the code. We were then able to detect the reference frame in the image by checking the area and storing its position in *x_Reference*, *y_Reference*, *position_Reference* and classifying the reference frame in our *classified* array.

4.1. Segmentation and recognition of objects

In this section we discuss the methods taken in recognizing the shapes of the objects captured in the picture. Firstly we created arrays for objects and targets of each shape (for example, rectangle _Objects and rectangle_Targets would be 2 separate arrays containing rectangle objects and targets respectively). In total there were 8 arrays; 2 for each shape.

We also created arrays called *classified[]*, *unclassified[]* and *objectIds[]*. The classified array contains all the objects and targets that have been *classified* as a

particular shape while the *ObjectIds* contains the identification number of all the objects and targets captured in the image. Exactly how these arrays become useful will be explained later in this section

The first shape recognition was circle. To recognise a circle from the image, we need to determine the circularity of all the shapes in the workspace and if the circularity of a shape was found to be less than or equal to 1 then it was classified as a circle. However, a square may sometimes slip into this category, so we implemented extra limits on the area and perimeter of the objects for better precision. To determine the circularity of a shape, the formula used is;

$$circularity = \frac{\text{Perimeter}^2}{4 * \text{pi} * \text{filledArea}}$$

So all shapes with circularity less than or equal to 1 are put in the circles array with their centroids converted to centimeters. To classify the circle as an object or a target , we used a parameter called EulerNumber. If the EulerNumber is equal to 1 then it is an object else If the EulerNumber is equal to 0 it is a target and they are stored in their respective arrays i.e. circle_Objects and circle_Targets respectively. The id of the shapes classified as circles are put in the classified array.

For the rectangle detection, we used the MinorAxisLength, MajorAxisLength, Area and Perimeter. If the shape has the MajorAxisLength > (MinoirAxisLength+10), Area <680 & >120, and perimeter <200 & >80 then its classified as a rectangle. Again to classify if it's an object or a target , we used a function called EulerNumber. If the EulerNumber is = 1 then it's an object but if it's = 0 it's a target and they are stored in their respective arrays ie rectangle_Objects and rectangle_Targets. The id of the shapes classified as rectangles are put in the classified array.

In order to detect triangle objects and targets, we made use of one of the properties of the *regionprops* function referred to as the bounding box. The bounding box overlays the object in the image with a rectangular box and fills the image. We can then calculate the ratio of the filled region within this box with the unfilled region. However, this doesn't fully guarantee that the region we are observing is a triangle object or target. To be more confident, we will check other parameter such as the *FilledArea* and the *Circularity* of the object. If all these parameters meet the required measurements, then we are certain that it is a triangle object or target. Now, in order to fully distinguish between the target and the object, we will calculate the Euler Number of the shape. If the euler number is equal 1, then it is a triangular object and if 0 then it is a triangular target. After identifying these shapes, we then calculate their x and y coordinates, orientation, store there id value in our *classified* array and identify them as triangle objects/targets in our triangle_Objects/triangle_Targets array.

Lastly, every shape in the objectid array that isn't found in the classified array meaning it hasn't been classified as a circle, rectangle or triangle, is then put in an array called unclassified. From the remaining shapes in the unclassified array anyone which has a EulerNumber = 1 & an Area > 100 is put in the square_Objects array and any shape with the EulerNumber = 0 & an Area > 100 is put in the square_Targets array. This was our compromise in detecting square since they have properties that overlap with a lot of the other shapes.

It should also be noted that along with the x y coordinates of each object, the orientation of the object, as gotten from the regionprops orientation parameter, was stored. So if an image were to have two rectangles, *rectangle_Objects* becomes a nxm (n=2, m=3) matrix with a structure as follows;

rectangle_Objects = [x_position1, y_position1, orientation1;
x_position2, y_position2, orientation2]

If an image were to have four rectangles, *rectangle_Objects* becomes a nxm (n=4, m=3) matrix with a structure as follows;

rectangle_Objects = [x_position1, y_position1, orientation1;
x_position2, y_position2, orientation2;
x_position3, y_position3, orientation3;
x_position4, y_position4, orientation4]

Code snippet (**cs** 4.1) shows our implementation of these tasks. The code was verified to work by ensuring that all the values present in each object and target matrix were correspondent to the right shape, object and target, checking their orientation and centroid values as printed on our command line window.

5. Implementation of Modules

Now, we've explained our forward kinematics model, inverse and image processing modules. The link between these modules and our output from the system which is a generated text file containg the commands to control the Robix robot is a function called *performAction.m* as can be seen in code snippet (cs 5.1) in the appendix. *performAction.m* takes in the parameters gotten form the image processing sequentially. For each object and target pair for a specific shape, the inverse kinematics is solved. If the object or target is within our workspace, then instructions are inputed in the robix block. If for example, either object or target is not within the workspace, an error message is printed out. In the event that an object was picked up but its corresponding tareget was not in the workspace, commands to drop the object back down would be written into the text file generated. Figure 5 below shows how transformations in our workspace was implemented.

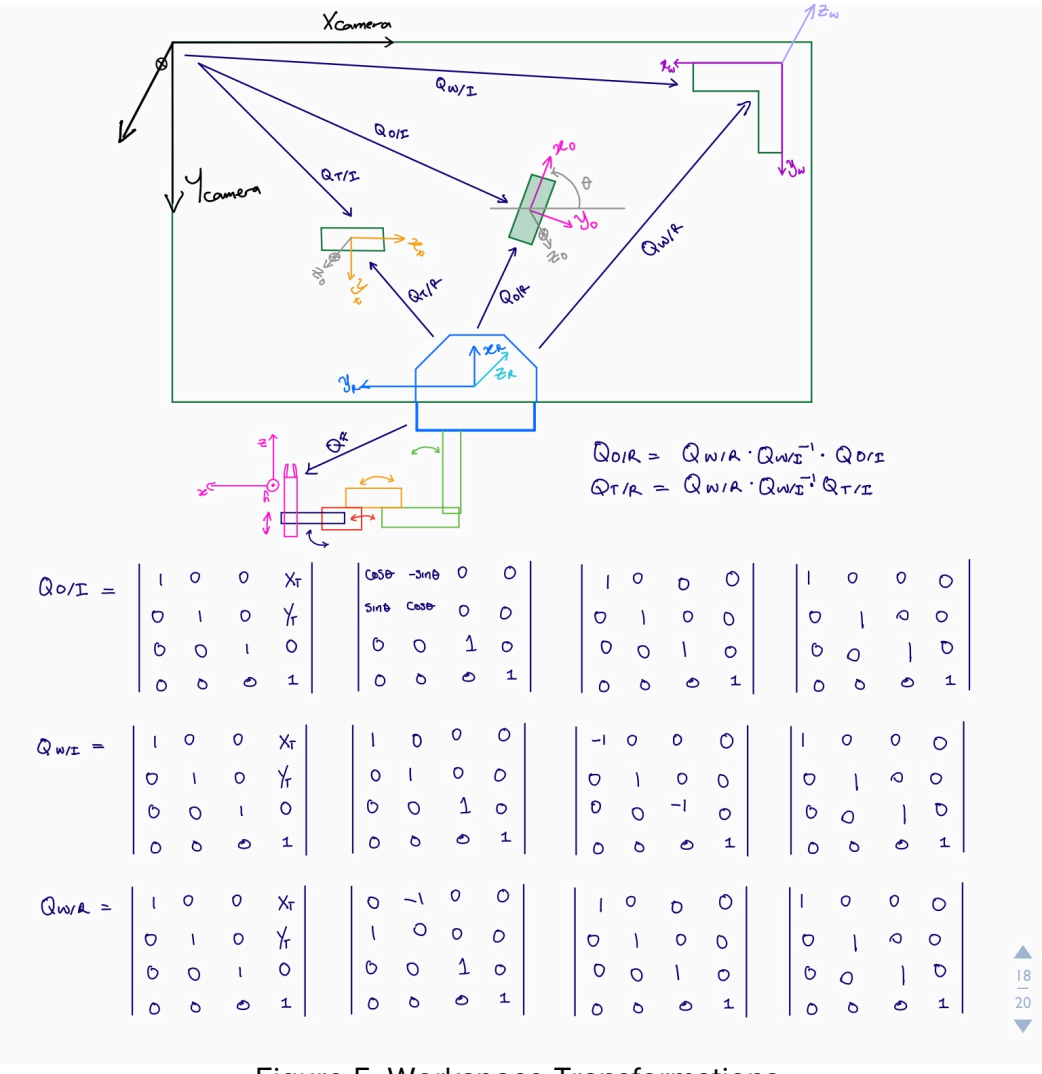

Figure 5. Workspace Transformations.

6. Discussion and Conclusion

Overall, our system behaved in ways that we expected. Although, there were some areas worth improvement. More specifically, the precision of the location of an object or target was sometimes not exactly accurate although acceptable. Also the orientation of the gripper was sometime off by 90 degrees. We believe this is because we did not take into account that the home configuration of the end gripper as gotten from our DH table added a 90 degrees shift to the actual home configuration of the robot. Also errors in precision could be due to improper measurements of the workspace.

A good thing to point out is that by taking advantage of the motor speed control on the ROBIX, we were able to achieve smooth and fluent motions in our workspace.

An area of improvement would be our path planning. Although for all our demonstrated scenarios in the lab, there were no issues with the path we programmed our ROBIX to take from object to target. For a more complex workspace, where other objects could be obstacles, we would need to plan better, the route we take from object to target so that we do not collide with obstacles In the workspace.

Altogether, this was an excellent project with so much to offer in terms of knowledge and experience in robotics. We are excited to see where the next steps take us.

Appendix

```
function forwardMatrix = getForwardMatrix(theta1, theta2, theta3, theta4,
theta5, K)

% Generate A matrices and Q_Forward = Q_effector_base (matrix defining how
% to get the base to the end effector)
A1 = [cosd(theta1) sind(theta1) 0 (10*cosd(theta1));
      sind(theta1) -cosd(theta1) 0 (10*sind(theta1));
      0 0 -1 12.5;
      0 0 0 1];
A2 = [cosd(theta2) 0 sind(theta2) (10*cosd(theta2));
      sind(theta2) 0 -cosd(theta2) (10*sind(theta2));
      0 1 0 0;
      0 0 0 1];
A3 = [cosd(theta3) sind(theta3) 0 (6*cosd(theta3));
      sind(theta3) -cosd(theta3) 0 (6*sind(theta3));
      0 0 -1 0;
      0 0 0 1];
A4 = [cosd(90+theta4) 0 sind(90+theta4) 0;
      sind(90+theta4) 0 -cosd(90+theta4) 0;
      0 1 0 0;
      0 0 0 1];
A5 = [cosd(theta5) -sind(theta5) 0 0;
      sind(theta5) cosd(theta5) 0 0;
      0 0 1 K;
      0 0 0 1];

forwardMatrix = A1*A2*A3*A4*A5;

end
```

(cs. 2.2.1)

```
function f = Q_Forward_eq(theta1, theta2, theta3, theta4, theta5)

% Generate A matrices and Q_Forward = Q_effector_base (matrix defining how
% to get the base to the end effector)
K = 4;

a_x = cosd(theta1-theta2)*cosd(theta3-90-theta4)*cosd(theta5) + sind(theta1-
      theta2)*sind(theta5);
b_x = -cosd(theta1-theta2)*cosd(theta3-90-theta4)*sind(theta5) +
sind(theta1-
      theta2)*cosd(theta5);
c_x = cosd(theta1-theta2)*sind(90+theta4-theta3);
P_x = K*cosd(theta1-theta2)*sind(90+theta4-theta3) + 6*cosd(theta1-
      theta2)*cosd(theta3) + 10*cosd(theta1-theta2) + 10*cosd(theta1);
a_y = sind(theta1-theta2)*cosd(theta3-90-theta4)*cosd(theta5) - cosd(theta1-
      theta2)*sind(theta5);
b_y = -sind(theta1-theta2)*cosd(theta3-90-theta4)*sind(theta5) -
cosd(theta1-
```

```
      theta2)*cosd(theta5);
c_y = sind(theta1-theta2)*sind(90+theta4-theta3);
P_y = K*sind(theta1-theta2)*sind(90+theta4-theta3) + 6*sind(theta1-
      theta2)*cosd(theta3) + 10*sind(theta1-theta2) + 10*sind(theta1);
a_z = sind(90+theta4-theta3)*cosd(theta5);
b_z = -sind(90+theta4-theta3)*sind(theta5);
c_z = -cosd(theta3-90-theta4);
P_z = -K*cosd(theta3-90-theta4) + -6*sind(theta3) + 12.5;

f = [a_x b_x c_x P_x; a_y b_y c_y P_y; a_z b_z c_z P_z; 0 0 0 1];

end
```

(**cs.** 2.2.2)

```
function thetaParams = getInverseParams(Q_Spec)
K = 8.5;    l_1 = 8.5;   l_2 = 8.5;   l_3 = 6;    d_1 = 12.5;
;%K =d_5

a_x = Q_Spec(1,1); b_x = Q_Spec(1,2); c_x = Q_Spec(1,3); p_x = Q_Spec(1,4);

a_y = Q_Spec(2,1); b_y = Q_Spec(2,2); c_y = Q_Spec(2,3); p_y = Q_Spec(2,4);

a_z = Q_Spec(3,1); b_z = Q_Spec(3,2); c_z = Q_Spec(3,3); p_z = Q_Spec(3,4);

theta3 = asind(((K*c_z)+d_1-p_z)/l_3);
theta3 = [theta3 180-theta3];
newtheta3 = unique(checkThetaRange_and_Complex(theta3, 3));  %check theta3
values
placeHolder = newtheta3;
for i=1:length(placeHolder)
    p_znew = K*c_z - l_3*sind(placeHolder(i)) + d_1;
    if round(p_znew,1) ~= round(p_z,1)
        newtheta3 = newtheta3(newtheta3 ~= placeHolder(i));
    end
end

% for theta4
theta4 = [];
for i = 1:length(newtheta3)
    r = [1 -1] * sqrt(c_x^2 + c_y^2);
    temp = [acosd(r) -acosd(r)];
    for j = 1:length(temp)
        theta4 = [theta4 temp(j)+newtheta3(i)];
    end
end
newtheta4 = unique(round(checkThetaRange_and_Complex(theta4,4),2));  %check
theta4 values
placeHolder = newtheta4;
for i=1:length(placeHolder)
    for j=1:length(newtheta3)
        c_znew = -cosd(newtheta3(j)-90-placeHolder(i));
        if round(c_znew,1) ~= round(c_z,1)
            newtheta4 = newtheta4(newtheta4 ~= placeHolder(i));
        end
```

```
    end
end

% for theta2
theta2 = [];
for i = 1:length(newtheta3)
    for j = 1:length(newtheta4)
        alpha = p_x^2 + p_y^2 - (K^2*((cosd(newtheta4(j)-newtheta3(i)))^2))
- ((2*l_3)*K*cosd(newtheta4(j)-newtheta3(i))*cosd(newtheta3(i))) -
((2*l_2)*K*cosd(newtheta4(j)-newtheta3(i))) -
((l_3^2)*((cosd(newtheta3(i)))^2)) - ((2*l_3*l_2)*cosd(newtheta3(i))) -
(l_1^2) - (l_2^2);
        beta = ((2*l_1*K)*cosd(newtheta4(j)-newtheta3(i))) +
((2*l_3*l_1)*cosd(newtheta3(i))) + (2*l_1*l_2);
        result = [acosd(alpha/beta) -acosd(alpha/beta)];
        theta2 = [theta2 result];
    end
end
theta2 = checkThetaRange_and_Complex(theta2,2);  %check theta2 values

% for theta1
theta1 = [];
theta1_2 = [];
for i=1:length(theta2)
    for j=1:length(newtheta3)
        for k=1:length(newtheta4)
            first_val = acosd(c_x/(sind(90+newtheta4(k)-newtheta3(j))));
            second_val = asind(c_y/(sind(90+newtheta4(k)-newtheta3(j))));
            result = [(first_val + theta2(i)) ((-first_val) + theta2(i))
(second_val + theta2(i)) ((180 - second_val) + theta2(i))];
            theta1 = [theta1 result];
        end
        result = atan2d(p_y,p_x) +
atan2d(sind(theta2(i))*((l_3*cosd(newtheta3(j))) +
l_2),(cosd(theta2(i))*((l_3*cosd(newtheta3(j))) + l_2))+l_1);
        theta1_2 = [theta1_2 result];
    end
end
theta1 = checkThetaRange_and_Complex(theta1,1);  %check theta2 values
theta1_2 = checkThetaRange_and_Complex(theta1_2,1);  %check theta2 values

placeHolder1 = [theta1 theta1_2];
placeHolder2 = theta2;
newthetavalues = getNewThetaValues(placeHolder1, placeHolder2,newtheta3,
newtheta4, K, l_1, l_2, l_3, p_x, p_y);
newtheta1 = newthetavalues{1};
newtheta2 = newthetavalues{2};

% for theta5
theta5 = [];
for i = 1:length(newtheta1)
    for j = 1:length(newtheta2)
        result = a_x*cosd(newtheta1(i)-newtheta2(j)) +
a_y*sind(newtheta1(i)-newtheta2(j));
        theta5 = [theta5 acosd(result) -acosd(result)];
    end
end
```

```
newtheta5 = unique(round(checkThetaRange_and_Complex(theta5,5),2));  %check
theta5 values

theta_vals = 0;
for i=1:length(newtheta1)
    for j=1:length(newtheta2)
        for k=1:length(newtheta3)
            for l=1:length(newtheta4)
                for m=1:length(newtheta5)
                    a_xnew = cosd(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*cosd(newtheta5(m)) +
sind(newtheta1(i)-newtheta2(j))*sind(newtheta5(m));
                    b_xnew = -cosd(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*sind(newtheta5(m)) +
sind(newtheta1(i)-newtheta2(j))*cosd(newtheta5(m));
                    c_xnew = cosd(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k));
                    p_xnew = K*cosd(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k)) + l_3*cosd(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)) + l_2*cosd(newtheta1(i)-newtheta2(j)) +
l_1*cosd(newtheta1(i));
                    a_ynew = sind(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*cosd(newtheta5(m)) -
cosd(newtheta1(i)-newtheta2(j))*sind(newtheta5(m));
                    b_ynew = -sind(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*sind(newtheta5(m)) -
cosd(newtheta1(i)-newtheta2(j))*cosd(newtheta5(m));
                    c_ynew = sind(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k));
                    p_ynew = K*sind(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k)) + l_3*sind(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)) + l_2*sind(newtheta1(i)-newtheta2(j)) +
l_1*sind(newtheta1(i));
                    a_znew = sind(90+newtheta4(l)-
newtheta3(k))*cosd(newtheta5(m));
                    b_znew = -sind(90+newtheta4(l)-
newtheta3(k))*sind(newtheta5(m));
                    c_znew = -cosd(newtheta3(k)-90-newtheta4(l));
                    p_znew = -K*cosd(newtheta3(k)-90-newtheta4(l)) + -
l_3*sind(newtheta3(k)) + d_1;
                    Q_new = [a_xnew b_xnew c_xnew p_xnew; a_ynew b_ynew
c_ynew p_ynew; a_znew b_znew c_znew p_znew; 0 0 0 1];

                    criteria1 =
isalmost(Q_new(1:3,1:3),Q_Spec(1:3,1:3),0.1);
                    criteria2 = isalmost(Q_new(:,4),Q_Spec(:,4),3);
                    if all(criteria1(:)) && all(criteria2')
                        theta_vals = [newtheta1(i) newtheta2(j) newtheta3(k)
newtheta4(l) newtheta5(m)];
                        if isequal(round(Q_new,1),round(Q_Spec,1))
                            thetaParams = [newtheta1(i) newtheta2(j)
newtheta3(k) newtheta4(l) newtheta5(m)];
                            return
                        end
                    end
                end
            end
```

```
        end
    end
end
thetaParams = theta_vals;

end
```

(**cs.** 3.1.1)

```
function newThetaValues = checkThetaRange_and_Complex(thetaValues,
whichTheta)
newThetaValues = thetaValues;
% return;
    for i = 1:length(thetaValues)
        if (whichTheta == 1) && (isreal(thetaValues(i)) == 0)
            newThetaValues(i) = abs(newThetaValues(i));
        end

        if (whichTheta == 2) && (isreal(thetaValues(i)) == 0)
            newThetaValues(i) = abs(newThetaValues(i));
        end

        if (whichTheta == 3) && (thetaValues(i) > 77.5 || thetaValues(i) < -
77.5 || isreal(thetaValues(i)) == 0) && (thetaValues(i)-360 > 77.5 ||
thetaValues(i)-360 < -77.5)
            newThetaValues(i) = sign(newThetaValues(i))*77.5;
        end

        if (whichTheta == 4) && (thetaValues(i) > 82.5 || thetaValues(i) < -
82.5 || isreal(thetaValues(i)) == 0) && (thetaValues(i)-360 > 82.5 ||
thetaValues(i)-360 < -82.5)
            newThetaValues(i) = sign(newThetaValues(i))*82.5;
        end

        if (whichTheta == 5) && (isreal(thetaValues(i)) == 0)
            newThetaValues(i) = sign(newThetaValues(i))*75;
        end
    end
    newThetaValues = newThetaValues(newThetaValues == real(newThetaValues));
end
```

(**cs.** 3.1.2)

```
% x1 is the amount of steps required to rotate servo 1 by y1 degrees
function motorValues = getMotorStepValues(y)

% First configuration [Preferred]. Setting x = 0 to represent y = 0 degrees.
% [Taking the home configuration as the refernce].
x1 = -y(1)/(0.0554);
x2 = y(2)/(0.0429);
x3 = y(3)/(0.0554);
x4 = y(4)/(0.0589);
x5 = -y(5)/(0.0536);

motorValues = [x1, x2, x3, x4, x5];

end
```

```
Workspace_image = imread('all outside the workspace.jpg');

% Change to grayscale
Workspace_image_gray = rgb2gray(Workspace_image);


%set threshold and convert image to binary image using threshold
thresholdValue = 80;
Workspace_image_binary = Workspace_image_gray < thresholdValue;
% imshow(Workspace_image_binary)

% Label each blob so we can make measurements of it
Workspace_image_binary_labelled = bwlabel(Workspace_image_binary, 8);
coloredLabels = label2rgb (Workspace_image_binary_labelled, 'hsv', 'k',
'shuffle');
% imshow(Workspace_image_binary_labelled, []);hold on;imshow(coloredLabels);

% get all measurement parameters for each blob(shape) in the binary image
blob_Measurements = regionprops(Workspace_image_binary_labelled,
Workspace_image_gray, 'all');
number_of_blobs = size(blob_Measurements, 1);
imshow(Workspace_image_gray);
title('Outlines, from bwboundaries()');
axis image; % Make sure image is not artificially stretched because of
screen's aspect ratio.
hold on;
boundaries = bwboundaries(Workspace_image_binary);
numberOfBoundaries = size(boundaries, 1);
for k = 1 : numberOfBoundaries
thisBoundary = boundaries{k};
plot(thisBoundary(:,2), thisBoundary(:,1), 'g', 'LineWidth', 2);
end
hold off;


blobECD = zeros(1, number_of_blobs);
% Print header line in the command window.
fprintf(1,'Blob
#      Orientation  Area   Perimeter    Centroid       Diameter   EulerN   M
ajorAxisLength    MinorAxisLength    FilledArea     BoundingBox3     Boundin
gBox4\n');
% Loop over all blobs printing their measurements to the command window.
for k = 1 : number_of_blobs           % Loop through all blobs.
% Find the mean of each blob.  (R2008a has a better way where you can pass
the original image
% directly into regionprops.  The way below works for all versions including
earlier versions.)
thisBlobsPixels = blob_Measurements(k).PixelIdxList;  % Get list of pixels
in current blob.
meanGL = mean(Workspace_image_gray(thisBlobsPixels)); % Find mean intensity
(in original image!)
meanGL2008a = blob_Measurements(k).MeanIntensity; % Mean again, but only for
```

```
version >= R2008a

blobArea = blob_Measurements(k).Area; % Get area.
blobPerimeter = blob_Measurements(k).Perimeter; % Get perimeter.
blobCentroid = blob_Measurements(k).Centroid; % Get centroid one at a time
blobECD(k) = sqrt(4 * blobArea / pi); % Compute ECD - Equivalent Circular
Diameter.
    EulerNumber = blob_Measurements(k).EulerNumber;
    MajorAxisLength = blob_Measurements(k).MajorAxisLength;
    MinorAxisLength = blob_Measurements(k).MinorAxisLength;
    FilledArea = blob_Measurements(k).FilledArea;
    BoundingBox3 = blob_Measurements(k).BoundingBox(3);
    BoundingBox4 = blob_Measurements(k).BoundingBox(4);
    Orientation = blob_Measurements(k).Orientation;

fprintf(1,'#%2d %17.1f %11.1f %8.1f %8.1f %8.1f
%8.1f    %d      %8.1f         %8.1f        %8.1f          %8.1f        %8.1
f\n', k, Orientation, blobArea, blobPerimeter, blobCentroid, blobECD(k),
EulerNumber, MajorAxisLength, MinorAxisLength, FilledArea, BoundingBox3,
BoundingBox4);
% Put the "blob number" labels on the "boundaries" grayscale image.
text(blobCentroid(1), blobCentroid(2), num2str(k), 'FontWeight', 'Bold',
'Color', 'w');
end

pixel_to_cm = 10/72;
% centers = blob_Measurements.Centroid;
figure(2);imshow(Workspace_image_gray); hold on;
%
% a = length(centers);


% Initialize object cells
rectangle_Objects = [];
rectangle_Targets = [];

square_Objects = [];
square_Targets = [];

circle_Objects = [];
circle_Targets = [];

triangle_Objects = [];
triangle_Targets = [];

classified = [];
objectIds = [];

filledImage = imfill(Workspace_image_binary_labelled, 'holes');
circleMeasurements = regionprops(filledImage, 'perimeter',
'area','centroid');
circles = [];
for i = 1 : length(circleMeasurements)
    tempcentroid = circleMeasurements(i).Centroid;

    circularity =
circleMeasurements(i).Perimeter^2/(4*pi*circleMeasurements(i).Area);
```

```
    if(round(circularity,1) <=1)
        circles = [circles; tempcentroid];
    end
end
numberOfCircles = size(circles);
numberOfCircles = numberOfCircles(1);

% Detect and classify Objects
for i = 1 : length(blob_Measurements)
    objectIds = [objectIds i];

    % Classify regionprops parameters for better readability
    Area = blob_Measurements(i).Area;
    BoundingBox3 = blob_Measurements(i).BoundingBox(3);
    BoundingBox4 = blob_Measurements(i).BoundingBox(4);
    Centroid = blob_Measurements(i).Centroid;
    Diameter = blob_Measurements(i).EquivDiameter;
    EulerNumber = blob_Measurements(i).EulerNumber;
    FilledArea = blob_Measurements(i).FilledArea;
    Perimeter = blob_Measurements(i).Perimeter;
    MajorAxisLength = blob_Measurements(i).MajorAxisLength;
    MinorAxisLength = blob_Measurements(i).MinorAxisLength;
    circularity =
circleMeasurements(i).Perimeter^2/(4*pi*circleMeasurements(i).Area);

    %Extrema Params
    TopLeft         = blob_Measurements(i).Extrema(1,:);
    TopRight        = blob_Measurements(i).Extrema(2,:);
    RightTop        = blob_Measurements(i).Extrema(3,:);
    RightBottom     = blob_Measurements(i).Extrema(4,:);
    BottomRight     = blob_Measurements(i).Extrema(5,:);
    BottomLeft      = blob_Measurements(i).Extrema(6,:);
    LeftBottom      = blob_Measurements(i).Extrema(7,:);
    LeftTop         = blob_Measurements(i).Extrema(8,:);

     tri =
round(blob_Measurements(i).BoundingBox(3)*blob_Measurements(i).BoundingBox(4
)*(1/2),0); %Base * Hieght /2

    %Detect Reference Frame
    if(Area > 680)
        plot(TopRight(1),TopRight(2),'ro');
        text(Centroid(1),Centroid(2),num2str(i),'Color','y');
        x_Reference = TopRight(1)*pixel_to_cm;
        y_Reference = TopRight(2)*pixel_to_cm;
        position_Reference = [x_Reference y_Reference];
        classified = [classified i];
    end

    %Detect Rectangle
    if((MajorAxisLength > (MinorAxisLength+10)) && (Area < 680 && Area >
120) && (Perimeter < 200 && Perimeter > 80))
        plot(Centroid(1),Centroid(2),'ro');
        text(Centroid(1),Centroid(2),num2str(i),'Color','y');

        %if Object then store position and orientation
        if EulerNumber == 1
```

```
            x = Centroid(1)*pixel_to_cm;
            y = Centroid(2)*pixel_to_cm;
            orientation = blob_Measurements(i).Orientation;
            rectangle_Objects = [rectangle_Objects; [x y orientation]];
            classified = [classified i];
            continue
        end
        if EulerNumber == 0
            x = Centroid(1)*pixel_to_cm;
            y = Centroid(2)*pixel_to_cm;
            orientation = blob_Measurements(i).Orientation;
            rectangle_Targets = [rectangle_Targets; [x y orientation]];
            classified = [classified i];
            continue
        end
    end

    % DETECT CIRCLE
    for j = 1:numberOfCircles
        circle = circles(j,:);
        criteria = isalmost(circle,Centroid,3);
        if all(criteria) && Perimeter > 30 && FilledArea < 310 && FilledArea
> 240
            if EulerNumber == 1
                x = Centroid(1)*pixel_to_cm;
                y = Centroid(2)*pixel_to_cm;
                orientation = blob_Measurements(i).Orientation;
                circle_Objects = [circle_Objects; [x y orientation]];
                classified = [classified i];
                break
            end
            if EulerNumber == 0
                x = Centroid(1)*pixel_to_cm;
                y = Centroid(2)*pixel_to_cm;
                orientation = blob_Measurements(i).Orientation;
                circle_Targets = [circle_Targets; [x y orientation]];
                classified = [classified i];
                break
            end
        end
    end

%         % DETECT TRIANGLE
    ratio = (BoundingBox3*BoundingBox4)/FilledArea;
    if (ratio > 1.5) && FilledArea < 300 && FilledArea > 120 && circularity
> 1.06
        if EulerNumber == 1
                x = Centroid(1)*pixel_to_cm;
                y = Centroid(2)*pixel_to_cm;
                orientation = blob_Measurements(i).Orientation;
                triangle_Objects = [triangle_Objects; [x y orientation]];
                classified = [classified i];
                continue
            end
            if EulerNumber == 0
                x = Centroid(1)*pixel_to_cm;
                y = Centroid(2)*pixel_to_cm;
```

```
                orientation = blob_Measurements(i).Orientation;
                triangle_Targets = [triangle_Targets; [x y orientation]];
                classified = [classified i];
                continue
            end
    end
end
unclassified = setdiff(objectIds, classified);

for i=1:length(unclassified)
    blobId = unclassified(i);
    EulerNumber = blob_Measurements(blobId).EulerNumber;
    Area = blob_Measurements(blobId).Area;
    Centroid = blob_Measurements(blobId).Centroid;
    orientation = blob_Measurements(blobId).Orientation;

    %if Object then store position and orientation
        if EulerNumber == 1 && Area > 100
            x = Centroid(1)*pixel_to_cm;
            y = Centroid(2)*pixel_to_cm;
            square_Objects = [square_Objects; [x y orientation]];
            continue
        end
        if EulerNumber == 0 && Area > 100
            x = Centroid(1)*pixel_to_cm;
            y = Centroid(2)*pixel_to_cm;
            square_Targets = [square_Targets; [x y orientation]];
            continue
        end
end

% Move Rectangles If any
numberOfObjects = size(rectangle_Objects);
numberOfObjects = numberOfObjects(1);
numberOfTargets = size(rectangle_Targets);
numberOfTargets = numberOfTargets(1);
if numberOfObjects ~= 0
    for i=1:numberOfObjects
        if  numberOfTargets >= i
            objectPicked = PerformAction(rectangle_Objects(i,:),
position_Reference, 'object', 'rectangle');
            if objectPicked == 1
                PerformAction(rectangle_Targets(i,:), position_Reference,
'target', 'rectangle');
            end
        end
    end
end

% Move Circles If any
numberOfObjects = size(circle_Objects);
numberOfObjects = numberOfObjects(1);
numberOfTargets = size(circle_Targets);
numberOfTargets = numberOfTargets(1);
if numberOfObjects ~= 0
    for i=1:numberOfObjects
        if  numberOfTargets >= i
```

```
            objectPicked = PerformAction(circle_Objects(i,:),
position_Reference, 'object', 'circle');
            if objectPicked == 1
                PerformAction(circle_Targets(i,:), position_Reference,
'target', 'circle');
            end
        end
    end
end

% Move Triangles If any
numberOfObjects = size(triangle_Objects);
numberOfObjects = numberOfObjects(1);
numberOfTargets = size(triangle_Targets);
numberOfTargets = numberOfTargets(1);
if numberOfObjects ~= 0
    for i=1:numberOfObjects
        if  numberOfTargets >= i
            objectPicked = PerformAction(triangle_Objects(i,:),
position_Reference, 'object', 'triangle');
            if objectPicked == 1
                PerformAction(triangle_Targets(i,:), position_Reference,
'target', 'triangle');
            end
        end
    end
end

% Move Squares If any
numberOfObjects = size(square_Objects);
numberOfObjects = numberOfObjects(1);
numberOfTargets = size(square_Targets);
numberOfTargets = numberOfTargets(1);
if numberOfObjects ~= 0
    for i=1:numberOfObjects
        if  numberOfTargets >= i
            objectPicked = PerformAction(square_Objects(i,:),
position_Reference, 'object', 'square');
            if objectPicked == 1
                PerformAction(square_Targets(i,:), position_Reference,
'target', 'square');
            end
        end
    end
end

fileID = fopen('Robix Instructions.txt','a');
fprintf(fileID,'\n#End of action sequence\n');
fprintf(fileID,'move all to 0;\n');
fclose(fileID);
```

(cs. 4.1)

```
function ActionFlag = PerformAction(O_T_Location,
referenceLocation_wrt_Image, O_T_Type, whichObject)
    referenceLocation_wrt_Robot = [24,-24];
```

```
    if strcmpi(O_T_Type,'object')
        ActionFlag = PickObject(O_T_Location, referenceLocation_wrt_Image,
referenceLocation_wrt_Robot, whichObject);
    end
    if strcmpi(O_T_Type,'target')
        DropObject(O_T_Location, referenceLocation_wrt_Image,
referenceLocation_wrt_Robot, whichObject);
        ActionFlag = 0;
    end

end

function ActionFlag = PickObject(objectLocation,
referenceLocation_wrt_Image, referenceLocation_wrt_Robot, whichObject)
Q_OI_TX     = objectLocation(1);
Q_OI_TY     = objectLocation(2);
roll_OI     = -objectLocation(3);

Q_WI_TX     = referenceLocation_wrt_Image(1);
Q_WI_TY     = referenceLocation_wrt_Image(2);

Q_WR_TX     = referenceLocation_wrt_Robot(1);
Q_WR_TY     = referenceLocation_wrt_Robot(2);

% GET Q_OI MATRIX object wrt Image
Q_OI_T = [1 0 0 Q_OI_TX; 0 1 0 Q_OI_TY; 0 0 1 0; 0 0 0 1];
Q_OI_R = [cosd(roll_OI) -sind(roll_OI) 0 0; sind(roll_OI) cosd(roll_OI) 0 0;
0 0 1 0; 0 0 0 1];
Q_OI_P = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_OI_Y = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_OI   = Q_OI_T*Q_OI_R*Q_OI_P*Q_OI_Y;

% GET Q_WI MATRIX workspace wrt Image
Q_WI_T = [1 0 0 Q_WI_TX; 0 1 0 Q_WI_TY; 0 0 1 0; 0 0 0 1];
Q_WI_R = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WI_P = [-1 0 0 0; 0 1 0 0; 0 0 -1 0; 0 0 0 1];
Q_WI_Y = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WI   = Q_WI_T*Q_WI_R*Q_WI_P*Q_WI_Y;


% GET Q_WR MATRIX workspace wrt Robot base
Q_WR_T = [1 0 0 Q_WR_TX; 0 1 0 Q_WR_TY; 0 0 1 0; 0 0 0 1];
Q_WR_R = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1];
Q_WR_P = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WR_Y = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WR   = Q_WR_T*Q_WR_R*Q_WR_P*Q_WR_Y;

Q_OR = Q_WR * 1/Q_WI * Q_OI;

getStepValues = InverseKinematics(Q_OR);

if length(getStepValues) == 3
    stepValues = getStepValues{3};
    fileID = fopen('Robix Instructions.txt','a');
    fprintf(fileID,'\n#Instructions to pick up %s\n', whichObject);
    fprintf(fileID,'Invert all off;\n');
    fprintf(fileID,'move 6 to minpos;\n');
```

```
    fprintf(fileID,'move 1 to %d;\n', stepValues(1));
    fprintf(fileID,'move 2 to %d;\n', stepValues(2));
    fprintf(fileID,'move 3 to 0;\n');
    fprintf(fileID,'move 5 to %d;\n', stepValues(5));
    fprintf(fileID,'move 4 to %d;\n', stepValues(4));
    fprintf(fileID,'maxspd 3,4 6;\n');
    fprintf(fileID,'move 3 to %d;\n', stepValues(3));
    fprintf(fileID,'move 6 to 0;\n');
    fprintf(fileID,'move 3 to 0;\n');
    fclose(fileID);
    ActionFlag = 1;
else
    fileID = fopen('Robix Instructions.txt','a');
    fprintf(fileID,'\n#object %s out of range\n', whichObject);
    fclose(fileID);
    ActionFlag = 0;
end



end

function DropObject(objectLocation, referenceLocation_wrt_Image,
referenceLocation_wrt_Robot, whichObject)
Q_TI_TX     = objectLocation(1);
Q_TI_TY     = objectLocation(2);
roll_OI     = -objectLocation(3);

Q_WI_TX     = referenceLocation_wrt_Image(1);
Q_WI_TY     = referenceLocation_wrt_Image(2);

Q_WR_TX     = referenceLocation_wrt_Robot(1);
Q_WR_TY     = referenceLocation_wrt_Robot(2);

% GET Q_OI MATRIX object wrt Image
Q_TI_T = [1 0 0 Q_TI_TX; 0 1 0 Q_TI_TY; 0 0 1 0; 0 0 0 1];
Q_TI_R = [cosd(roll_OI) -sind(roll_OI) 0 0; sind(roll_OI) cosd(roll_OI) 0 0;
0 0 1 0; 0 0 0 1];
Q_TI_P = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_TI_Y = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_TI   = Q_TI_T*Q_TI_R*Q_TI_P*Q_TI_Y;

% GET Q_WI MATRIX workspace wrt Image
Q_WI_T = [1 0 0 Q_WI_TX; 0 1 0 Q_WI_TY; 0 0 1 0; 0 0 0 1];
Q_WI_R = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WI_P = [-1 0 0 0; 0 1 0 0; 0 0 -1 0; 0 0 0 1];
Q_WI_Y = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WI   = Q_WI_T*Q_WI_R*Q_WI_P*Q_WI_Y;


% GET Q_WR MATRIX workspace wrt Robot base
Q_WR_T = [1 0 0 Q_WR_TX; 0 1 0 Q_WR_TY; 0 0 1 0; 0 0 0 1];
Q_WR_R = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1];
Q_WR_P = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WR_Y = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
Q_WR   = Q_WR_T*Q_WR_R*Q_WR_P*Q_WR_Y;
```

```
Q_TR = Q_WR * 1/Q_WI * Q_TI;

getStepValues = InverseKinematics(Q_TR);

if length(getStepValues) == 3
        stepValues = getStepValues{3};
        fileID = fopen('Robix Instructions.txt','a');
        fprintf(fileID,'\n#Instructions to drop %s\n', whichObject);
        fprintf(fileID,'move 1 to %d;\n', stepValues(1));
        fprintf(fileID,'move 2 to %d;\n', stepValues(2));
        fprintf(fileID,'move 4 to %d;\n', stepValues(4));
        fprintf(fileID,'move 5 to %d;\n', stepValues(5));
        fprintf(fileID,'move 3 to %d;\n', stepValues(3));
        fprintf(fileID,'move 6 to minpos;\n');
        fprintf(fileID,'move 3 to 0;\n');
        fclose(fileID);
else
    fileID = fopen('Robix Instructions.txt','a');
    fprintf(fileID,'\n#Target out of range\n');
    fprintf(fileID,'move 4 to -818;\n');
    fprintf(fileID,'move 3 to 755;\n');
    fprintf(fileID,'move 6 to minpos;\n');
    fprintf(fileID,'move 3 to 0;\n');
    fclose(fileID);
end
end
```

(cs. 5.1)

```
function g = InverseKinematics(Q_Spec)

    g{1} = Q_Spec;
    g{2} = getInverseParams(Q_Spec);

    if g{2} == 0
        warning('Out Of Bounds. dimensions not attainable!');
    else
        g{2} = checkThetaVal(g{2});
        g{3} = round(getMotorStepValues(g{2}), 0);
    end

end

function thetaParams = getInverseParams(Q_Spec)
K = 8.5;    l_1 = 8.5;   l_2 = 8.5;   l_3 = 6;    d_1 = 12.5;

a_x = Q_Spec(1,1); b_x = Q_Spec(1,2); c_x = Q_Spec(1,3); p_x = Q_Spec(1,4);

a_y = Q_Spec(2,1); b_y = Q_Spec(2,2); c_y = Q_Spec(2,3); p_y = Q_Spec(2,4);

a_z = Q_Spec(3,1); b_z = Q_Spec(3,2); c_z = Q_Spec(3,3); p_z = Q_Spec(3,4);

theta3 = asind(((K*c_z)+d_1-p_z)/l_3);
theta3 = [theta3 180-theta3];
newtheta3 = unique(checkThetaRange_and_Complex(theta3, 3));  %check theta3
values
```

```
placeHolder = newtheta3;
for i=1:length(placeHolder)
    p_znew = K*c_z - l_3*sind(placeHolder(i)) + d_1;
    if round(p_znew,1) ~= round(p_z,1)
        newtheta3 = newtheta3(newtheta3 ~= placeHolder(i));
    end
end

% for theta4
theta4 = [];
for i = 1:length(newtheta3)
    r = [1 -1] * sqrt(c_x^2 + c_y^2);
    temp = [acosd(r) -acosd(r)];
    for j = 1:length(temp)
        theta4 = [theta4 temp(j)+newtheta3(i)];
    end
end
newtheta4 = unique(round(checkThetaRange_and_Complex(theta4,4),2));  %check
theta4 values
placeHolder = newtheta4;
for i=1:length(placeHolder)
    for j=1:length(newtheta3)
        c_znew = -cosd(newtheta3(j)-90-placeHolder(i));
        if round(c_znew,1) ~= round(c_z,1)
            newtheta4 = newtheta4(newtheta4 ~= placeHolder(i));
        end
    end
end

% for theta2
theta2 = [];
for i = 1:length(newtheta3)
    for j = 1:length(newtheta4)
        alpha = p_x^2 + p_y^2 - (K^2*((cosd(newtheta4(j)-newtheta3(i)))^2))
- ((2*l_3)*K*cosd(newtheta4(j)-newtheta3(i))*cosd(newtheta3(i))) -
((2*l_2)*K*cosd(newtheta4(j)-newtheta3(i))) -
((l_3^2)*((cosd(newtheta3(i)))^2)) - ((2*l_3*l_2)*cosd(newtheta3(i))) -
(l_1^2) - (l_2^2);
        beta = ((2*l_1*K)*cosd(newtheta4(j)-newtheta3(i))) +
((2*l_3*l_1)*cosd(newtheta3(i))) + (2*l_1*l_2);
        result = [acosd(alpha/beta) -acosd(alpha/beta)];
        theta2 = [theta2 result];
    end
end
theta2 = checkThetaRange_and_Complex(theta2,2);  %check theta2 values

% for theta1
theta1 = [];
theta1_2 = [];
for i=1:length(theta2)
    for j=1:length(newtheta3)
        for k=1:length(newtheta4)
            first_val = acosd(c_x/(sind(90+newtheta4(k)-newtheta3(j))));
            second_val = asind(c_y/(sind(90+newtheta4(k)-newtheta3(j))));
            result = [(first_val + theta2(i)) ((-first_val) + theta2(i))
(second_val + theta2(i)) ((180 - second_val) + theta2(i))];
            theta1 = [theta1 result];
```

```
        end
        result = atan2d(p_y,p_x) +
atan2d(sind(theta2(i))*((l_3*cosd(newtheta3(j))) +
l_2),(cosd(theta2(i))*((l_3*cosd(newtheta3(j))) + l_2))+l_1);
        theta1_2 = [theta1_2 result];
    end
end
theta1 = checkThetaRange_and_Complex(theta1,1);  %check theta2 values
theta1_2 = checkThetaRange_and_Complex(theta1_2,1);  %check theta2 values

placeHolder1 = [theta1 theta1_2];
placeHolder2 = theta2;
newthetavalues = getNewThetaValues(placeHolder1, placeHolder2,newtheta3,
newtheta4, K, l_1, l_2, l_3, p_x, p_y);
newtheta1 = newthetavalues{1};
newtheta2 = newthetavalues{2};

% for theta5
theta5 = [];
for i = 1:length(newtheta1)
    for j = 1:length(newtheta2)
        result = a_x*cosd(newtheta1(i)-newtheta2(j)) +
a_y*sind(newtheta1(i)-newtheta2(j));
        theta5 = [theta5 acosd(result) -acosd(result)];
    end
end
newtheta5 = unique(round(checkThetaRange_and_Complex(theta5,5),2));  %check
theta5 values

theta_vals = 0;
for i=1:length(newtheta1)
    for j=1:length(newtheta2)
        for k=1:length(newtheta3)
            for l=1:length(newtheta4)
                for m=1:length(newtheta5)
                    a_xnew = cosd(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*cosd(newtheta5(m)) +
sind(newtheta1(i)-newtheta2(j))*sind(newtheta5(m));
                    b_xnew = -cosd(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*sind(newtheta5(m)) +
sind(newtheta1(i)-newtheta2(j))*cosd(newtheta5(m));
                    c_xnew = cosd(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k));
                    p_xnew = K*cosd(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k)) + l_3*cosd(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)) + l_2*cosd(newtheta1(i)-newtheta2(j)) +
l_1*cosd(newtheta1(i));
                    a_ynew = sind(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*cosd(newtheta5(m)) -
cosd(newtheta1(i)-newtheta2(j))*sind(newtheta5(m));
                    b_ynew = -sind(newtheta1(i)-
newtheta2(j))*cosd(newtheta3(k)-90-newtheta4(l))*sind(newtheta5(m)) -
cosd(newtheta1(i)-newtheta2(j))*cosd(newtheta5(m));
                    c_ynew = sind(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k));
                    p_ynew = K*sind(newtheta1(i)-
newtheta2(j))*sind(90+newtheta4(l)-newtheta3(k)) + l_3*sind(newtheta1(i)-
```

```
newtheta2(j))*cosd(newtheta3(k)) + l_2*sind(newtheta1(i)-newtheta2(j)) +
l_1*sind(newtheta1(i));
                    a_znew = sind(90+newtheta4(l)-
newtheta3(k))*cosd(newtheta5(m));
                    b_znew = -sind(90+newtheta4(l)-
newtheta3(k))*sind(newtheta5(m));
                    c_znew = -cosd(newtheta3(k)-90-newtheta4(l));
                    p_znew = -K*cosd(newtheta3(k)-90-newtheta4(l)) + -
l_3*sind(newtheta3(k)) + d_1;
                    Q_new = [a_xnew b_xnew c_xnew p_xnew; a_ynew b_ynew
c_ynew p_ynew; a_znew b_znew c_znew p_znew; 0 0 0 1];

                    criteria1 =
isalmost(Q_new(1:3,1:3),Q_Spec(1:3,1:3),0.1);
                    criteria2 = isalmost(Q_new(:,4),Q_Spec(:,4),3);
                    if all(criteria1(:)) && all(criteria2')
                        theta_vals = [newtheta1(i) newtheta2(j) newtheta3(k)
newtheta4(l) newtheta5(m)];
                        if isequal(round(Q_new,1),round(Q_Spec,1))
                            thetaParams = [newtheta1(i) newtheta2(j)
newtheta3(k) newtheta4(l) newtheta5(m)];
                            return
                        end
                    end
                end
            end
        end
    end
end
thetaParams = theta_vals;

end


function newThetaValues = checkThetaVal(thetaValues)

    for i = 1:length(thetaValues)
        if (i == 1 || i == 3) && (thetaValues(i) > 77.5 || thetaValues(i) <
-77.5)
            if (thetaValues(i)-360 > 77.5 || thetaValues(i)-360 < -77.5)
                thetaValues(i) = sign(thetaValues(i))*77.5;
            else
                thetaValues(i) = thetaValues(i)-360;
            end
        end

        if (i == 2) && (thetaValues(i) > 60 || thetaValues(i) < -60)
            if (thetaValues(i)-360 > 60 || thetaValues(i)-360 < -60)
                thetaValues(i) = sign(thetaValues(i))*60;
            else
                thetaValues(i) = thetaValues(i)-360;
            end
        end

        if (i == 4) && (thetaValues(i) > 82.5 || thetaValues(i) < -82.5)
            if (thetaValues(i)-360 > 82.5 || thetaValues(i)-360 < -82.5)
                thetaValues(i) = sign(thetaValues(i))*82.5;
```

```
            else
                thetaValues(i) = thetaValues(i)-360;
            end
        end

        if (i == 5) && (thetaValues(i) > 75 || thetaValues(i) < -75)
            if (thetaValues(i)-360 > 75 || thetaValues(i)-360 < -75)
                thetaValues(i) = sign(thetaValues(i))*75;
            else
                thetaValues(i) = thetaValues(i)-360;
            end
        end
    end
    newThetaValues = thetaValues;
end

function newThetaValues = getNewThetaValues(placeHolder1, placeHolder2,
newtheta3, newtheta4, K, l_1, l_2, l_3, p_x, p_y)
newtheta1 = [];
newtheta2 = [];
newThetaValues = {[] []};

    for i=1:length(placeHolder1)
        for j=1:length(placeHolder2)
            for k=1:length(newtheta3)
                for l=1:length(newtheta4)
                    p_xnew = K*cosd(placeHolder1(i)-
placeHolder2(j))*sind(90+newtheta4(l)-newtheta3(k)) +
l_3*cosd(placeHolder1(i)-placeHolder2(j))*cosd(newtheta3(k)) +
l_2*cosd(placeHolder1(i)-placeHolder2(j)) + l_1*cosd(placeHolder1(i));
                    p_ynew = K*sind(placeHolder1(i)-
placeHolder2(j))*sind(90+newtheta4(l)-newtheta3(k)) +
l_3*sind(placeHolder1(i)-placeHolder2(j))*cosd(newtheta3(k)) +
l_2*sind(placeHolder1(i)-placeHolder2(j)) + l_1*sind(placeHolder1(i));
                    if (round(p_xnew,0) == round(p_x,0)) && (round(p_ynew,0)
== round(p_y,0))
                        newtheta1 = [newtheta1 placeHolder1(i)];
                        newtheta2 = [newtheta2 placeHolder2(j)];
                        newThetaValues = {newtheta1 newtheta2};
                        return
                    end
                end
            end
        end
    end
end

function forwardMatrix = getForwardMatrix(theta1, theta2, theta3, theta4,
theta5, K, l_1, l_2, l_3, d_1)
% Generate A matrices and Q_Forward = Q_effector_base (matrix defining how
% to get the base to the end effector)
A1 = [cosd(theta1) sind(theta1) 0 (10*cosd(theta1)); sind(theta1) -
cosd(theta1) 0 (l_1*sind(theta1)); 0 0 -1 d_1; 0 0 0 1];
A2 = [cosd(theta2) 0 sind(theta2) (10*cosd(theta2)); sind(theta2) 0 -
cosd(theta2) (l_2*sind(theta2)); 0 1 0 0; 0 0 0 1];
A3 = [cosd(theta3) sind(theta3) 0 (6*cosd(theta3)); sind(theta3) -
cosd(theta3) 0 (l_3*sind(theta3)); 0 0 -1 0; 0 0 0 1];
```

```
A4 = [cosd(90+theta4) 0 sind(90+theta4) 0; sind(90+theta4) 0 -
cosd(90+theta4) 0; 0 1 0 0; 0 0 0 1];
A5 = [cosd(theta5) -sind(theta5) 0 0; sind(theta5) cosd(theta5) 0 0; 0 0 1
K; 0 0 0 1];

forwardMatrix = A1*A2*A3*A4*A5;

end

function QTRPY = getQTRPY(Trans_x,Trans_y,Trans_z,roll,pitch,yaw)

    Q_T = [1 0 0 Trans_x; 0 1 0 Trans_y; 0 0 1 Trans_z; 0 0 0 1];
    Q_R = [cosd(roll) -sind(roll) 0 0; sind(roll) cosd(roll) 0 0; 0 0 1 0; 0
0 0 1];
    Q_P = [cosd(pitch) 0 sind(pitch) 0; 0 1 0 0; -sind(pitch) 0 cosd(pitch)
0; 0 0 0 1];
    Q_Y = [1 0 0 0; 0 cosd(yaw) -sind(yaw) 0; 0 sind(yaw) cosd(yaw) 0; 0 0 0
1];
    QTRPY = Q_T*Q_R*Q_P*Q_Y;

end

% Functions to manipulate servos
% x1 is the amount of steps required to rotate servo 1 by y1 degrees
function motorValues = getMotorStepValues(y)

% First configuration [Preferred]. Setting x = 0 to represent y = 0 degrees.
% [Taking the home configuration as the refernce].
x1 = -y(1)/(0.0554);
x2 = y(2)/(0.0429);
x3 = y(3)/(0.0554);
x4 = y(4)/(0.0589);
x5 = -y(5)/(0.0536);

motorValues = [x1, x2, x3, x4, x5];

end

function newThetaValues = checkThetaRange_and_Complex(thetaValues,
whichTheta)
newThetaValues = thetaValues;
% return;
    for i = 1:length(thetaValues)
        if (whichTheta == 1) && (isreal(thetaValues(i)) == 0)
            newThetaValues(i) = abs(newThetaValues(i));
        end

        if (whichTheta == 2) && (isreal(thetaValues(i)) == 0)
            newThetaValues(i) = abs(newThetaValues(i));
        end

        if (whichTheta == 3) && (thetaValues(i) > 77.5 || thetaValues(i) < -
77.5 || isreal(thetaValues(i)) == 0) && (thetaValues(i)-360 > 77.5 ||
thetaValues(i)-360 < -77.5)
            newThetaValues(i) = sign(newThetaValues(i))*77.5;
        end
```

```
        if (whichTheta == 4) && (thetaValues(i) > 82.5 || thetaValues(i) < -
82.5 || isreal(thetaValues(i)) == 0) && (thetaValues(i)-360 > 82.5 ||
thetaValues(i)-360 < -82.5)
            newThetaValues(i) = sign(newThetaValues(i))*82.5;
        end

        if (whichTheta == 5) && (isreal(thetaValues(i)) == 0)
            newThetaValues(i) = sign(newThetaValues(i))*75;
        end
    end
    newThetaValues = newThetaValues(newThetaValues == real(newThetaValues));
end
```

(InverseKinematics.m)

```
function test = isalmost(a,b,tol)
%
% usage: test = isalmost(a,b,tol)
%
% tests if matrix a is approximately equal to b within a specified
% tolerance interval (b-tol <= a <= b+tol)
%
% note:  if b is given as a scalar, all values in a are compared against
%        the scalar value b
%
% calls: none
%
% inputs:
%
% a(nr,nc) = matrix of data values to test
% b(nr,nc) = matrix of data values for comparison (or a single scalar value)
%      tol = tolerance used in computation
%
% outputs:
%
% test(nr,nc) = matrix of test results:
%
%        test(i,j) = 0 -> a(i,j) is not equal to b(i,j) (or is NaN)
%        test(i,j) = 1 -> a(i,j) is approximately equal to b(i,j)
%
%   author : James Crawford
%   created 01/08/2007
%
%   history: v1.0 (01/08/2007)
%
% get length of input matrix a
[nr,nc] = size(a);
% check input for consistency
if ~all(size(a) == size(b))
   if all(size(b) == [1 1])
      % convert scalar value b to a matrix of size(a)
      b = b*ones(size(a));
   else
      disp('error: input arguments are inconsistent (isalmost.m)')
      disp('(b) must be a matrix of same size as (a) or a single value')
```

```
    end
end
one = ones(size(b));
% perform test
test = (a <= b+tol*one)&(a >= b-tol*one);
%
```

(**isalmost.m**)