International Journal of Information Technology, Control and Automation (IJITCA)
Vol.15, No.1/2/3/4, October 2025

OBTAINING INVERSE KINEMATICS EQUATIONS FOR A PLANAR BALL-PLATE ROBOT

Ricardo Francisco Martínez-González, José Antonio Hernández-Reyes and Guillermo Sánchez-Vázquez

Tecnológico Nacional de México - IT Veracruz, Veracruz, Veracruz, Mexico

ABSTRACT

This work focuses on the derivation and validation of inverse kinematics equations for a planar ball-plate robot, a critical step for its precise control. The robot's kinematic model was developed considering a simplified equilateral triangular base and mobile platform. We detail the mathematical procedures for determining the z-coordinates of the platform's points and establishing the unit vector normal to the platform, which are fundamental for the inverse kinematics solution. The derived equations allow for the calculation of the joint angles necessary to achieve a desired ball position on the plate.

For modeling and simulation, Matlab and Simulink were utilized. The robot's SolidWorks design was exported to Simulink using the Simscape Multibody Link tool, and a PID controller was integrated to achieve realistic simulated behavior. Simulation results demonstrate that the derived inverse kinematics equations accurately guide the robot, with the simulated ball trajectory closely matching the desired circular path. Furthermore, computer vision techniques, implemented with OpenCV in Python, were employed for real-time detection and tracking of both the platform and the ball. This visual feedback system provides crucial positional data, allowing for the potential closure of the control loop for adaptive visual control. This project successfully combines precise inverse kinematics with visual feedback, laying a robust foundation for advanced control systems in planar ball-plate robots.

KEYWORDS

Inverse Kinematics, Ball-Plate Robot, Computer Vision

1. INTRODUCTION

The field of robotics demands a deep understanding of the interplay between mechanical design and computational control [1]. This work focuses on deriving the inverse kinematics equations for a specific type of manipulator: a planar ball-plate robot. These equations are fundamental for controlling a robot's position and orientation, allowing the determination of the joint movements needed to reach a desired point in space [2].

For modeling and simulation of this robotic system, Matlab and its simulation environment, Simulink, were chosen. Matlab, with its matrix-based approach, offers an intuitive and efficient platform for expressing and solving complex mathematical problems, which is ideal for robotic kinematics. Its integrated visualization tools facilitate data analysis and the extraction of critical information [3]. Simulink, in turn, complements Matlab by providing a block diagram environment for multi-domain simulation. It allows for graphical modeling of dynamic systems, offering customizable libraries and differential equation solvers that accelerate the design and verification process [4]. Simulink's ability to handle complex systems makes it a standard tool in control engineering [5].

Additionally, the project incorporates computer vision for the detection and tracking of key elements in the robot's environment. For this purpose, OpenCV (Open Source Computer Vision Library) will be used, a robust open-source library widely employed in areas like facial recognition and object identification [6]. Computer vision is crucial for endowing robots with autonomy and the ability to interact intelligently with their surroundings [7]. Integrating OpenCV will enable the robot to perceive its environment, which is crucial for applications requiring dynamic interaction with external elements, such as tracking a moving sphere.

The following sections will detail the mathematical development for obtaining the inverse kinematics equations of the planar ball-plate robot. Subsequently, we'll explain how the robot model was exported to Simulink and configured for simulation. Finally, we'll present and analyze the results obtained from both the robot simulation and real-time image processing, demonstrating the feasibility and accuracy of the proposed approach.

2. INVERSE KINEMATICS

To obtain the equations, we must consider Figure 1, which shows a simplified model of the robot to facilitate its analysis.

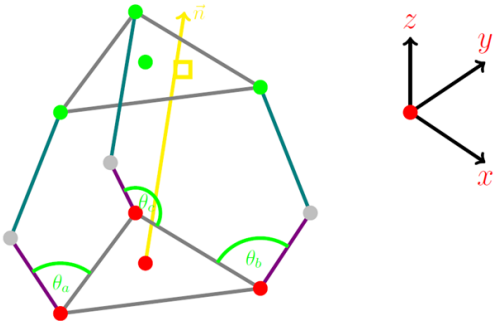


Figure 1. Simplified robot model

As shown in Figure 1, the values to be found for inverse kinematics are θ of each arm, considering the vector n as the sphere's location and that it's normal to the mobile platform. For inverse kinematics calculations, the equilateral triangular base shown in Figure 2 must be considered. Kinematic analysis is a fundamental step in the design of any robotic manipulator [8]. To find the y-coordinates of points b_0 and c_0 in Figure 2, Equation 1 and Equation 2 are used, which correspond to the circumradius and apothem in an equilateral triangle, respectively:

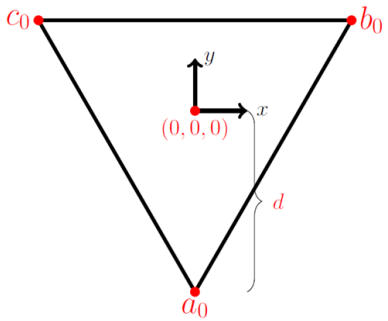


Figure 2. Robot fixed base

$$r = \frac{l \cdot \sqrt{3}}{3}$$ Equation 1

$$a_y = \frac{l \cdot \sqrt{3}}{6}$$ Equation 2

Solving for l from Equation 1, where r=d, and rationalizing, gives

$$l = \sqrt{3} \cdot d$$ Equation 3

Substituting the result of Equation 3 into Equation 2, and solving,

$$a_y = \frac{l}{2} \cdot d$$ Equation 4

Therefore, b_y and c_y will be equal to the result of Equation 4

$$b_y = \frac{l}{2} \cdot d, \quad c_y = \frac{l}{2} \cdot d$$ Equation 5

To find the x-coordinates of points b_0 and c_0, the value of the side calculated in Equation 3 must be halved,

$$b_x = \frac{\sqrt{3} \cdot d}{2}, \quad c_x = -\frac{\sqrt{3} \cdot d}{2}$$ Equation 6

As a result, points a_0, b_0, and c_0 correspond to,

$$a_0 = (a_x, a_y, a_z) = (0, -d, 0)$$
$$b_0 = (b_x, b_y, b_z) = (\frac{\sqrt{3}}{2} \cdot d, \frac{1}{2} d, 0)$$
$$c_0 = (c_x, c_y, c_z) = (\frac{\sqrt{3}}{2} \cdot d, \frac{1}{2} d, 0)$$ Equation 7

The mobile platform, like the base, is also an equilateral triangle as shown in Figure 3, where the center point is given by the coordinates of variable h, with h_z being a known user-provided value. Parallel robots often use triangular platform and base configurations due to their structural and kinematic advantages [9].

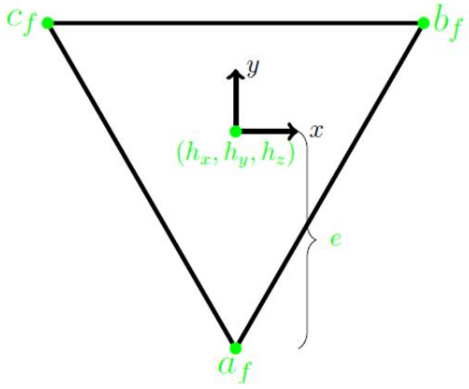


Figure 3. Robot mobile platform

To find the coordinates of points a_f, b_f, and c_f of the platform, we cannot follow the same method as for the base, because the platform's movement must be taken into account, leading to the model shown in Figure 4.

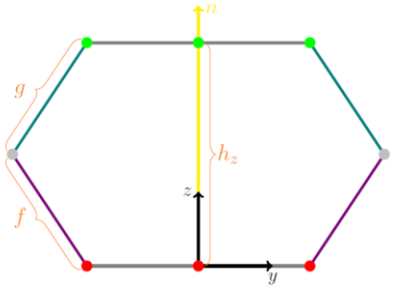


Figure 4. Robot in "Home" state

Considering Figure 4 and the robot's position as the "Home" state, the values of vector n are: <nx, ny, 1>. The next step is to adapt the results of Equation 7 with the platform's values, taking the x and y coordinates as fixed, resulting in the following:

$$a_f = (a_{fx}, a_{fy}, a_{fz}) = (0, -e, a_{fz})$$
$$b_f = (b_{fx}, b_{fy}, b_{fz}) = (\frac{\sqrt{3}}{2}e, \frac{1}{2}e, b_{fz})$$
$$c_f = (c_{fx}, c_{fy}, c_{fz}) = (\frac{\sqrt{3}}{2}e, \frac{1}{2}e, c_{fz})$$

Equation 8

Finally, with the data obtained in Equation 8, the z-coordinates of the points are found using Equation 9, where x is the point a, b, or c to be found.

$$\vec{n} = \vec{P_f}x$$

Equation 9

To find vector (Pfx), the coordinates of point x are subtracted from the coordinates of point P_f, in this case $x=a_f$.

$$\vec{P_f a_f} = (0, -e, a_z) = (0, 0, h_z)$$
$$\vec{P_f a_f} = \langle 0 - 0, -e - 0, a_z - h_z \rangle$$
$$\vec{P_f a_f} = \langle 0, -e, a_z - h_z \rangle$$

Equation 10

Substitute the value from Equation 10 into Equation 9

$$\vec{n} \cdot \vec{P_f a_f} = \langle n_x, n_y, 1 \rangle \cdot \langle 0, -e, a_z - h_z \rangle$$
$$\vec{n} \cdot \vec{P_f a_f} = n_x \cdot 0 - e + 1 \cdot a_z - h_z$$
$$\vec{n} \cdot \vec{P_f a_f} = -en_y + a_z - h_z$$

Equation 11

Equating the result of Equation 11 to 0, and solving for a_z,

$$-en_y + h_z = 0$$
$$a_z = h_z + en_y$$

Equation 12

Equation 12 rules the a_z coordinate for any point in the platform's movements. Repeat the procedure from Equation 10 to Equation 12, changing the values for b and c, obtaining the results shown in Equation 13:

$$a_z = h_z + en_y$$
$$b_z = h_z - \frac{e}{2}(\sqrt{3} \cdot n_x + n_y)$$
$$c_z = h_z + \frac{e}{2}(\sqrt{3} \cdot n_x - n_y)$$

Equation 13

To find the x-coordinates of points b and c, use the blue right triangle shown in Figure 5 as a reference.

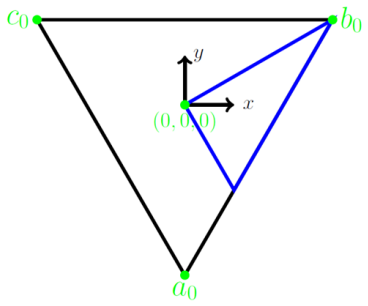


Figure 5. Blue triangle

As shown in Figure 5, the center of the mobile platform should be taken as the origin to obtain the coordinates of b_0 and the end of the vector perpendicular to it. The results are shown in Equation 14:

$$b_f = \langle \frac{\mp\sqrt{3}}{2}, \frac{1}{2}, 0 \rangle$$
$$\perp b_f = \langle \frac{1}{2}, -\frac{\mp\sqrt{3}}{2}, 0 \rangle$$

Equation 14

With the data obtained in Equation 14, express b_{fx} in terms of b_{fy}:

$$\frac{1}{2} b_f - \frac{\mp\sqrt{3}}{2} b_{fy} = 0$$
$$b_{fx} = \mp\sqrt{3} \cdot b_{fy}$$

Equation 15

Therefore,

$$c_{fx} = -\mp\sqrt{3} \cdot c_{fy}$$

Equation 16

Taking into account the values obtained in Equation 13, Equation 15, and Equation 16, the platform points are as shown in Equation 17:

$$a_f = (a_{fx}, a_{fy}, a_{fz}) = (0, a_{fy}, h_z + e n_y)$$
$$b_f = (b_{fx}, b_{fy}, b_{fz}) = (\mp\sqrt{3} \cdot b_{fy}, b_{fy}, h_z - \frac{e}{2}(\mp\sqrt{3} \cdot n_x + n_y))$$
$$c_f = (c_{fx}, c_{fy}, c_{fz}) = (-\mp\sqrt{3} \cdot c_{fy}, c_{fy}, h_z + \frac{e}{2}(\mp\sqrt{3} \cdot n_x - n_y))$$

Equation 17

Figure 6 shows the behavior of the cross product of vectors a and b in space.

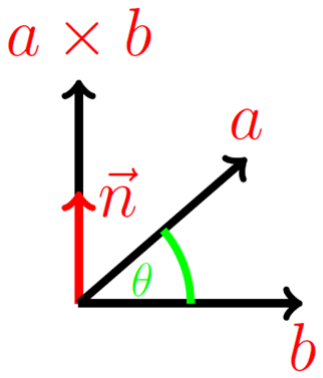


Figure 6: Cross product diagram of a and b

According to Figure 6, Equation 18 is deduced:

$$a \times b = (\|a\| \|b\| sen(\theta))\vec{n}$$

Equation 18

Where vec(n) is the unit vector orthogonal to vectors a and b. To apply Equation 18, vectors a and b will be those shown in blue in Figure 7.

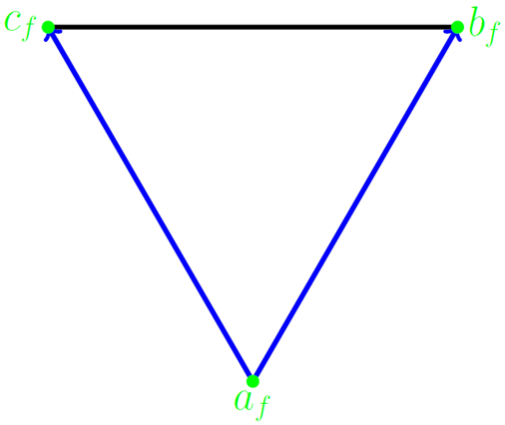


Figure 7. Vectors a and b on the platform

Equation 19 shows the magnitudes of vectors a and b.

$$\| b_f \vec{-} a_f \| = \sqrt{3} e$$
$$\| c_f \vec{-} a_f \| = \sqrt{3} e$$

Equation 19

Therefore, Equation 18 with the obtained values is shown in Equation 20:

$$b_f \vec{-} a_f \times c_f \vec{-} a_f = (\sqrt{3} e \sqrt{3} e \, sen(60°)) \vec{n}$$
$$b_f \vec{-} a_f \times c_f \vec{-} a_f = \frac{3\sqrt{3} e^2}{2} \vec{n}$$
$$b_f \vec{-} a_f \times c_f \vec{-} a_f = \omega \vec{n}$$

Equation 20

where $\omega = \frac{3\sqrt{3} e^2}{2}$

From Equation 20, we proceed to find the values of the vectors by subtracting the coordinates of the points from Equation 17, obtaining the results shown in Equation 21:

$$b_f \vec{-} a_f \langle \sqrt{3} b_{fy}, b_{fy} - a_{fy}, b_{fz} - a_{fz} \rangle$$
$$c_f \vec{-} a_f = \langle -\sqrt{3} c_{fy}, c_{fy} - a_{fy}, c_{fz} - a_{fz} \rangle$$

Equation 21

With the values from Equation 21, we obtain,

$$\omega \vec{n} = \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ \sqrt{3} b_{fy} & b_{fy} - a_{fy} & t \\ -\sqrt{3} c_{fy} & c_{fy} - a_{fy} & v \end{bmatrix}$$

Equation 22

The values for t and v are shown in Equation 23:

$$t = b_{fz} - a_{fz}$$
$$v = c_{fz} - a_{fz}$$

Equation 23

For practical purposes, we will solve it part by part, starting with vec (j) and solving for c*fy*, obtaining the result shown in Equation 24:

$$\omega n_y = -\sqrt{\mp 3}\, t c_{fy} - \sqrt{\mp 3}\, v b_{fy}$$
$$\frac{\omega n_y}{-\sqrt{\mp 3}} = t c_{fy} + v b_{fy}$$
$$c_{fy} = \frac{\frac{\omega n_y}{-\sqrt{\mp 3}} - v b_{fy}}{t}$$

Equation 24

We will then proceed to solve vec(k) by solving for a_{fy}. The result is shown in Equation 25:

$$\omega n_z = -\sqrt{\mp 3}\, b_{fy}(c_{fy} - a_{fy}) - (b_{fy} - a_{fy})(-\sqrt{\mp 3}\, c_{fy})$$
$$\omega n_z = \sqrt{\mp 3}(b_{fy} c_{fy} - a_{fy} b_{fy} + b_{fy} c_{fy} - c_{fy} a_{fy})$$
$$\omega n_z = \sqrt{\mp 3}(2 b_{fy} c_{fy} - a_{fy} b_{fy} - c_{fy} a_{fy})$$
$$\frac{\omega n_z}{\sqrt{\mp 3}} = 2 b_{fy} c_{fy} - a_{fy}(b_{fy} + c_{fy})$$
$$-a_{fy} = \frac{\frac{\omega n_z}{\sqrt{\mp 3}} - 2 b_{fy} c_{fy}}{b_{fy} + c_{fy}}$$
$$a_{fy} = \frac{2 b_{fy} c_{fy} - \frac{\omega n_z}{\sqrt{\mp 3}}}{b_{fy} + c_{fy}}$$

Equation 25

Finally, we will solve for the value of i:

$$\omega n_x = v(b_{fy} - a_{fy}) - t(c_{fy} - a_{fy})$$
$$\omega n_x = v b_{fy} - v a_{fy} - t c_{fy} - t a_{fy}$$
$$\omega n_x = v b_{fy} - t c_{fy} + a_{fy}(t - v)$$

Equation 26

Substitute the value of c_{fy} obtained in Equation 24 and the value of a_{fy} obtained in Equation 25 into Equation 26, then solve for b_{fy}. Equation 27 shows the result.

$$b_{fy} = \frac{e}{2}\left(1 + \frac{n_x^2 + \sqrt{\mp 3}\, n_x n_y}{n_z + 1}\right)$$

Equation 27

The result of Equation 27 is substituted into Equation 24,

$$c_{fy}=\frac{e}{2}\left(1-\frac{n_x^2+\mp\sqrt{3}n_xn_y}{n_z+1}\right)$$

Equation 28

Lastly, the results of Equation 27 and Equation 28 are substituted into Equation 25,

$$a_{fy}=\frac{e}{2}\left(1-\frac{n_x^2+3n_z^2+3n_z}{n_z+1-n_x^2}+\frac{n_x^4-3n_x^2n_y^2}{(n_z+1)(n_z+1-n_x^2)}\right)$$

Equation 29

Therefore, the inverse kinematics equations are as follows:

Arm A Angle

$$a_y=d+\left(\frac{e}{2}\right)\left(1-\frac{n_x^2+3n_z^2+3n_z}{n_z+1-n_x^2}+\frac{n_x^4-3n_x^2n_y^2}{(n_z+1)(n_z+1-n_x^2)}\right)$$

$$a_z=h_z+en_y$$

$$a_m=\sqrt{a_y^2+a_z^2}$$

$$\theta_a=\arccos\left(\frac{a_y}{a_m}\right)+\arccos\left(\frac{a_m^2+f^2-g^2}{2a_mf}\right)$$

Equation 30

Arm B Angle

$$b_x=\frac{\mp\sqrt{3}}{2}\left(e\left(1-\frac{n_x^2+\mp\sqrt{3}n_xn_y}{n_z+1}\right)-d\right)$$

$$b_y=\frac{b_x}{\mp\sqrt{3}}$$

$$b_z=h_z-\frac{e}{2}\left(\mp\sqrt{3}n_x+n_y\right)$$

$$b_m=\sqrt{b_x^2+b_y^2+b_z^2}$$

$$\theta_b=\arccos\left(\frac{\mp\sqrt{3}b_x+b_y}{-2b_m}\right)+\arccos\left(\frac{b_m^2+f^2+g^2}{2b_mf}\right)$$

Equation 31

Arm C Angle

$$c_x = \frac{\mp\sqrt{3}}{2}\left(d - e\left(1 - \frac{n_x^2 + \mp\sqrt{3}n_x n_y}{n_z + 1}\right)\right)$$

$$c_y = \frac{-c_x}{\mp\sqrt{3}}$$

$$c_z = h_z + \frac{e}{2}\left(\mp\sqrt{3}n_x + n_y\right)$$

$$c_m = \sqrt{c_x^2 + c_y^2 + c_z^2}$$

$$\theta_c = \arccos\left(\frac{\mp\sqrt{3}c_x + c_y}{2c_m}\right) + \arccos\left(\frac{c_m^2 + f^2 - g^2}{2c_m f}\right)$$

Equation 32

3. ROBOT DESIGN

With the inverse kinematics data obtained, the SolidWorks model, identical to the physical construction, is created, as seen in Figure 8.

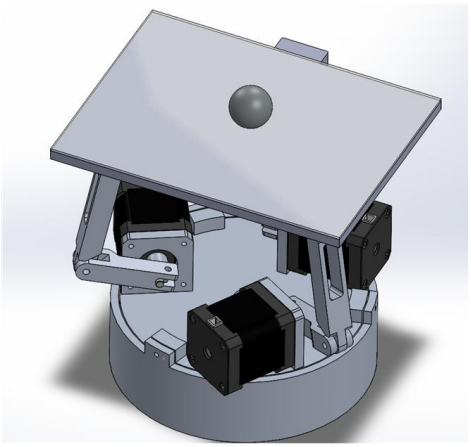


Figure 8. Final robot design

For simulation purposes, a simplified design model is needed without compromising functionality. Based on Figure 8, the simplified SolidWorks model is shown in Figure 9.

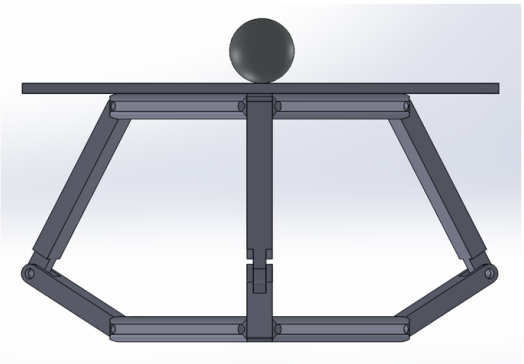


Figure 9. Simplified robot model

3.1. Design to XML

To work with the simulation in Matlab, the simplified model needs to be exported as an XML file using the Simscape Multibody Link tool, which can be installed from its official website [10]. Integrating CAD tools with simulation environments is crucial for validating complex designs and optimizing processes [11]. Once installed, navigate to "Tools" → "Simscape Multibody link" → "Export" → "Simscape Multibody".

In Matlab, once the folder where the XML file was exported is set as the current folder, execute the smimport() command in the Command Window, changing the file name to the one chosen in the previous step.

3.2. Block Diagram Configuration

Executing the command from Section 3.1 will open a Simulink document where blocks represent the model's parts, as shown in Figure 10.

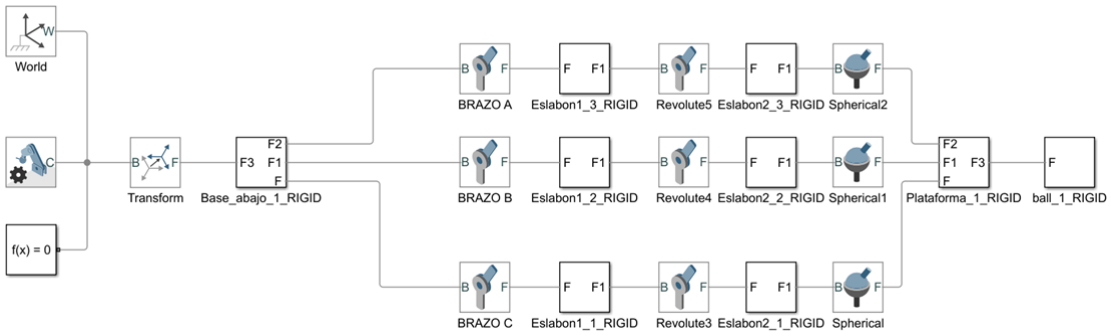


Figure 10: Block diagram created by Simscape

Although the tool facilitates block diagram creation in the Simulink file, some subsequent changes must be made, starting with the Mechanism Configuration block, by applying gravity along the Z-axis.

To simulate the sphere's movement on the platform, there's a library called ContactForce that can be installed by following the instructions on its official website [4]. Once the library is installed, a new section called Simscape Multibody Contact Forces Library appears in the browser. Within the 3D section, you'll find the necessary block. In Figure 11, you can see where the Sphere to Plane Force block is placed to emulate the contact of the sphere with the platform during the simulation.

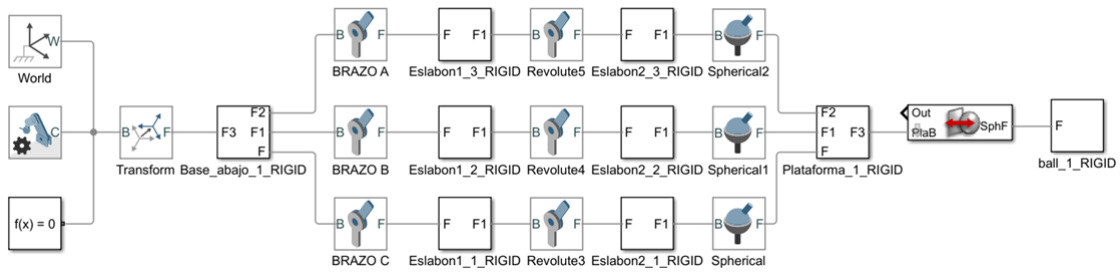


Figure 11. Diagram with the Sphere to Plane Force block added

For the Sphere to Plane Force block to know the platform and ball measurements, its default configuration must be changed. For the Sphere to Plane Force block to function correctly in the simulation, a Transform block must be added. Its configuration values are shown in Code 1 and must be added to the DataFile generated in Section 3.1.

Code 1. Transform Block Data

```
smiData.RigidTransform(20).translation = [ -6.9404293441236247 -
106.4984593676447 117.98887768247265];
smiData.RigidTransform(20).angle = 0;
smiData.RigidTransform(20).axis = [0 0 0];
```

Finally, a 6DOF block will be added, connected to the Transform block and the Sphere to Plane Force block. The complete diagram is shown in Figure 12.

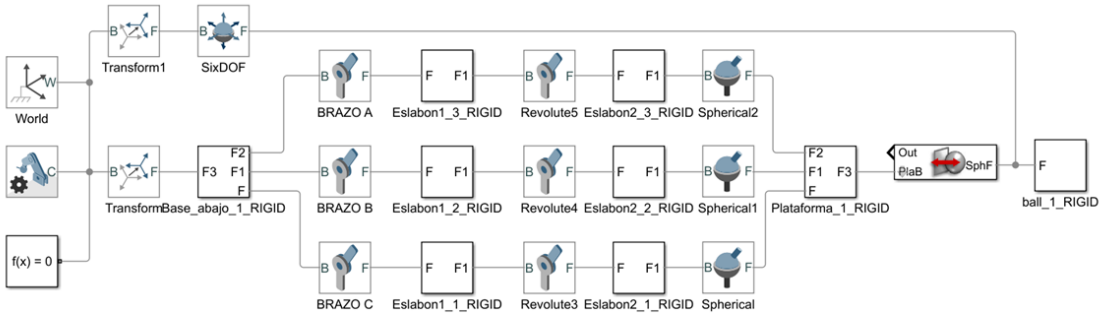


Figure 12. Diagram with Transform and 6DOF blocks

4. PLANAR BALL-PLATE ROBOT SIMULATION

With the values obtained in Section 3.1, the diagram in Figure 16 from the previous section will be modified. In Figure 13, a subsystem was added containing the blocks that simulate the robot. The trajectory block has a clock as input and contains Code 2.

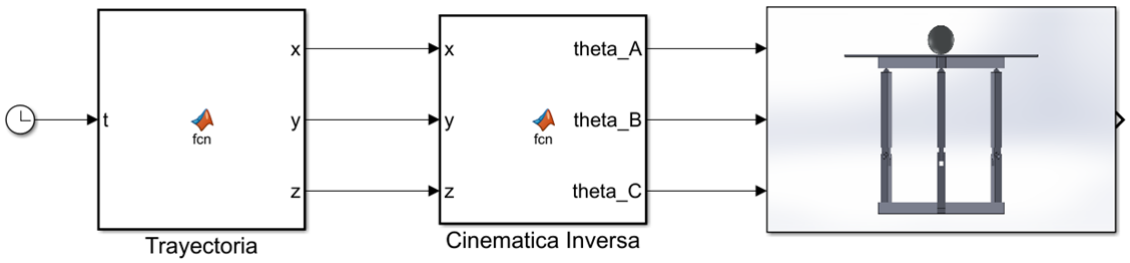


Figure 13. Connection of Simulink subsystems with the simplified planar ball-plate robot model
The code entered in the Trajectory block presents the circular behavior desired for this project, which can be seen in Code 2.

Code 2. Circular Trajectory

```
function [x, y, z] = fcn(t)
nx = 0.10 * cos (t);
ny = 0.10 * sin (t);
nm = sqrt (nx ^2 + ny ^2 + 1);
x = nx / nm;
y = ny / nm;
z = 1/ nm;
end
```

Now the inverse kinematics block contains Code 3, which is essentially the result of the procedure described in Section 2 of this document for inverse kinematics.

Code 3: Inverse Kinematics

```
function [ theta_A , theta_B , theta_C ] = fcn(x, y, z)
theta_A = Brazo_A (x, y, z);
theta_B = Brazo_B (x, y, z);
theta_C = Brazo_C (x, y, z);
end
```

As seen in Code 3, separate functions are called. These functions contain Equation 30, Equation 31, and Equation 32.

5. COMPUTER VISION

Computer vision, also known by other names such as artificial vision or image interpretation, is a discipline that seeks the automatic deduction of the structure and properties of a three-dimensional scenario or world, possibly changing, from one or more captured images. Regarding the images, these can be black and white or color, captured by a single camera or several associated, or even, in a greater degree of generalization of the image concept, come from non-visual sensors such as acoustic, thermal, tactile, among others [7].

5.1. Platform Detection

For platform detection, the OpenCV library will be used through its color detection function, all programmed in Python. Something to keep in mind is that usually the integrated camera of laptops has the number 0, and an external webcam can vary this number; in this particular case, it's number 2 and is configured using the code: cap = cv2.VideoCapture(2). Color segmentation is a fundamental technique in image processing for object identification [12].

The definition of color detection values is done considering that HSV colors are used. For this case, the blue color spectrum will be used, and to define it through code, it's necessary to declare them using NumPy arrays, as shown in Code 4.

Code 4. Color Spectrum Definition

```
azulBajo = np.array([100, 10, 100], np.uint8)
```

```
azulAlto = np.array([130, 255, 255], np.uint8)
```

By default, OpenCV processes the camera image in a BGR color scheme. Therefore, the first step is to convert this scheme to an HSV type, using that frame to start the color search, and subsequently find the contour of objects within that color range. This procedure is shown in Code 5.

Code 5. Mask Creation and Contour Search

```
frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
maskAzul = cv2.inRange(frameHSV, azulBajo, azulAlto)
contornos, _ = cv2.findContours(maskAzul, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

What is done in Code 6 is to calculate the area of the detected zone, and if it's greater than 3000, proceed to find the center of that area and draw a point to identify it. The cv2.putText function places the coordinates of that point, taking the entire camera's field of view as a reference. The cv2.ConvexHull() function smooths the contour to make it appear more uniform, and the last line draws that contour on the original frame.

Code 6. Drawing Contour in Detected Area

```
for c in contornos:
    area = cv2.contourArea(c)
    if area > 3000:
        M = cv2.moments(c)
        if (M["m00"] == 0): M["m00"] = 1
        x = int(M["m10"] / M["m00"])
        y = int(M["m01"] / M["m00"])
        cv2.circle(frame, (x, y), 7, (0, 255, 0), -1)
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(frame, "{},{}".format(x, y), (x + 10, y), font,
0.75, (0, 255, 0), 1, cv2.LINE_AA)
        nuevoContorno = cv2.convexHull(c)
        cv2.drawContours(frame, [nuevoContorno], 0, (255, 0, 0), 3)
```

5.2. Ball Detection

For detecting the ball on the platform, Hough Circle detection [13] will be used. The function in Code 7 first converts the image obtained by the camera to grayscale, then applies a blur effect to reduce background noise. Finally, the HoughCircles function is responsible for finding these circles. Important parameters include 100, referring to the minimum distance between two possible circles; param1 is the sensitivity for circle detection; param2 indicates after how many detected points a circle can be considered; and minRadius and maxRadius define the minimum and maximum radii for circles to be detected. Subsequently, the center of the detected circles is identified and drawn.

Code 7: Creating Distorted Frame and Center Location

```
grayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
blurFrame = cv2.GaussianBlur(grayFrame, (17, 17), 0)
```

```
circles = cv2.HoughCircles(blurFrame, cv2.HOUGH_GRADIENT, 1.2, 100,
                           param1=100,      param2=30,      minRadius=75,
maxRadius=400)
if circles is not None:
    circles = np.uint16(np.around(circles))
    chosen = None
    for i in circles[0, :]:
        if chosen is None: chosen = i
        if prevCircle is not None:
            if dist(chosen[0], chosen[1], prevCircle[0], prevCircle[1])
<= \
               dist(i[0], i[1], prevCircle[0], prevCircle[1]):
                chosen = i
        cv2.circle(frame, (chosen[0], chosen[1]), 1, (0, 100, 100), 3)
        cv2.putText(frame,     "{},{}".format(chosen[0],     chosen[1]),
(chosen[0] + 10, chosen[1]), font, 0.75, (0, 255, 0), 1, cv2.LINE_AA)
        cv2.circle(frame, (chosen[0], chosen[1]), chosen[2], (255, 0,
255), 3)
        prevCircle = chosen
```

6. RESULTS

Below are the results obtained from the simulation and image processing for the platform and ball. We describe how Matlab and Simulink were used to model the system's behavior and verify the inverse kinematics equations, as well as the use of computer vision algorithms to detect key elements. We also analyze the graphs comparing the ideal and actual trajectories.

In Figure 14, the 3D simulation is shown, where the sphere's entire behavior can be visually observed, with the advantage of being able to obtain result data by leveraging the full power of Matlab and Simulink tools.

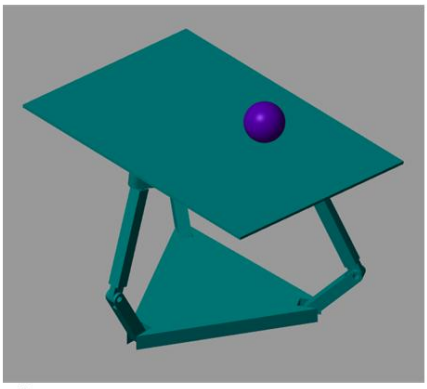


Figure 14. Simulink simulation view

To test the robot's behavior using the equations obtained in Section 2 and due to the limitation of adding a camera in Simulink, a PID block was added as feedback to achieve more realistic robot behavior during the simulation. PID controllers are widely used in control engineering for their effectiveness and simplicity [14]. Using a circular trajectory, Figure 15a shows the graph of the ideal trajectory programmed in the simulation. In Figure 15b, the graph of the actual trajectory

obtained after the simulation finished and starting with the ball at the center of the platform can be observed. As seen in both graphs of Figure 15, the trajectories are practically identical, demonstrating that the inverse kinematics equations are fulfilling their function of adequately moving the arms depending on the ball's location.

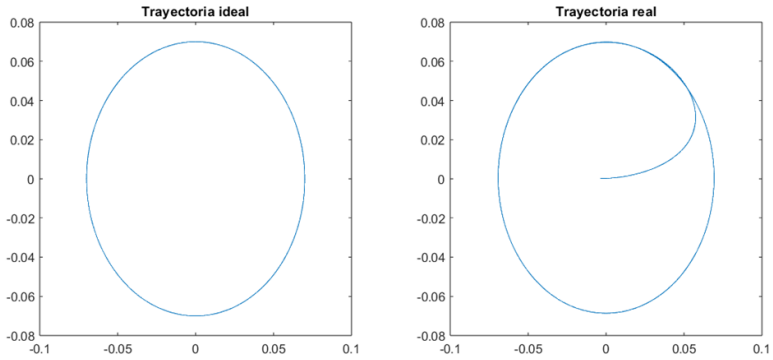
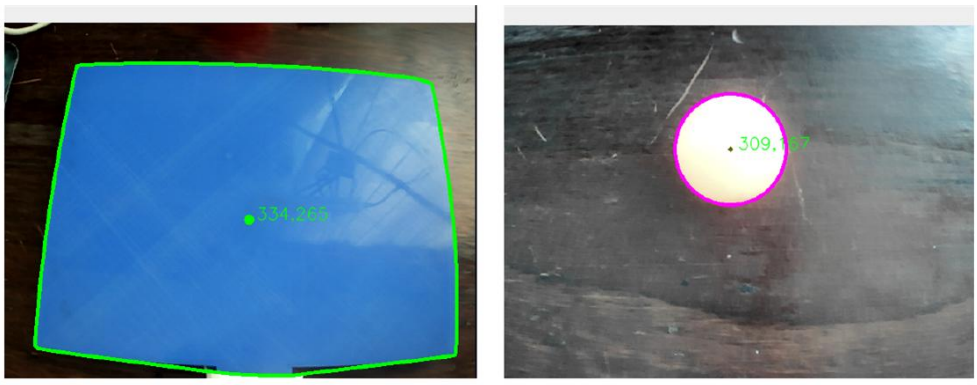


Figure 15. Comparison of ideal (a) and real (b) trajectory

As shown in Section 5, part of the code used to detect the platform and ball separately was explained. In Figure 16a, a window showing the platform detection is visible, displaying the coordinates of the platform's center relative to the camera's entire field of view. Meanwhile, Figure 16b shows the ball detection along with its respective center coordinates.



(a) Plataforma (b) Bola

Figure 16. Platform (a) and sphere (b) detection

The combined result of both codes can be seen in Figure 17. The coordinates of the platform's center point were hidden to improve visibility; however, a green reference point is still shown. The displayed coordinates are those of the ball's center (black dot) with respect to the platform's center point (green dot), and these coordinates will serve as input for the inverse kinematics equations in Section 2.

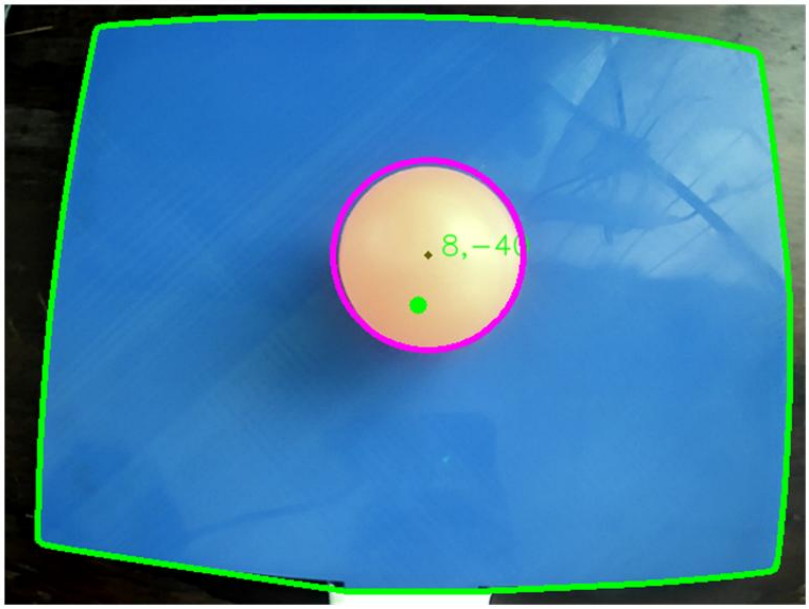


Figure 17. Final computer vision result applied to sphere and platform

7. CONCLUSIONS

This work successfully addressed the derivation of inverse kinematics equations for a planar ball-plate robot, a fundamental step for its precise control. We demonstrated the feasibility of integrating Matlab and Simulink for the dynamic modeling and simulation of the robot, allowing us to validate its behavior before any physical implementation. The 3D simulation in Simulink, complemented by a PID controller, confirmed the accuracy of our equations by achieving nearly identical simulated and desired sphere trajectories.

Additionally, the implementation of computer vision using OpenCV proved effective for real-time platform and sphere detection and tracking. The ability to obtain the coordinates of these elements is crucial for closing the control loop, as these coordinates can be used as input for the inverse kinematics equations, enabling adaptive visual control of the robot.

In summary, this project lays the groundwork for developing a robust control system for the planar ball-plate robot, combining the precision of inverse kinematics with visual feedback. The methodologies employed and the results obtained validate the proposed approach, opening the door for future research in active control and practical applications.

REFERENCES

[1] Siciliano, B., & Khatib, O. (Eds.). (2008). Springer Handbook of Robotics. Springer.
[2] Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). Robot Modeling and Control. John Wiley & Sons.
[3] The MathWorks, Inc. (2024) "MATLAB. The Language of Technical Computing." Retrieved from https://www.mathworks.com
[4] The MathWorks, Inc. (2024) "Simulink. Simulation and Model-Based Design." Retrieved from https://www.mathworks.com/products/simulink.html
[5] Dorf, R. C., & Bishop, R. H. (2011). Modern Control Systems (12th ed.). Pearson.
[6] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media.

[7] Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer Science & Business Media.
[8] Craig, J. J. (2005). Introduction to Robotics: Mechanics and Control (3rd ed.). Pearson Prentice Hall.
[9] Merlet, J. P. (2006). Parallel Robots (2nd ed.). Springer.
[10] The MathWorks, Inc. (2024). Simscape Multibody Link (R2024a). Retrieved from https://www.mathworks.com/products/simscape-multibody.html
[11] Al-Habaibeh, A., & Khusnood, A. (2017). "Integrated CAD/CAE for Smart Product Design and Manufacturing." In Smart Product Design and Manufacturing (pp. 3-21). Springer.
[12] Gonzalez, R. C., & Woods, R. E. (2010). Digital Image Processing (3rd ed.). Pearson Prentice Hall.
[13] Hough, P. V. C. (1962). "Method and means for recognizing complex patterns." U.S. Patent No. 3,069,654.
[14] Ogata, K. (2010). Modern Control Engineering (5th ed.). Prentice Hall.