TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 69, Issue II, June, 2026

PYTHON-BASED APPLICATION FOR GRAPHICAL REPRESENTATION OF THE END-EFFECTOR TRAJECTORY IN A RRR KINEMATIC ASSEMBLY

Angela-Miruna NEACȘU-PAVEL, Daniela TUNSOIU, Ileana DUGĂEȘESCU, Vlad-Cristian ENACHE, Mihaela-Elena ULMEANU, Cristian-Vasile DOICIN

Abstract: *This study presents a Python-based application designed for the graphical representation of the end-effector trajectory in an RRR kinematic assembly. The research focuses on developing a python script that facilitates the analysis and visualization of robotic manipulator motion, aiding in both academic and industrial applications. The study explores one structural analysis variant using three cylindrical joints, one of which is fixed. The position kinematic equations describe the system's motion. The Python implementation leverages key libraries, such as NumPy and Matplotlib employing an efficient computational strategy to model the kinematic behavior and generate graphical outputs. The application allows for user-coded input parameters, with predefined values tested to validate the model. The code is structured and explained, ensuring clarity and reproducibility. The system generates static trajectory graphs, providing a clear representation of the end-effector's path and validating the theoretical kinematic model. The results demonstrate that the proposed Python-based RRR model accurately simulates the motion of a human-like arm segment, confirming the model's effectiveness for kinematic analysis and educational applications.*

Key words: *Python-based, cylindrical joints, end-effector trajectory*

1. INTRODUCTION

In the field of robotic design, the selection of joints and kinematic pairs represents a critical challenge in ensuring appropriate mobility and interaction capabilities. The joints employed in such applications are often inspired by the goal of replicating the natural motions of the human body. Among the most frequently utilized types are spherical joints [1], pivotal joints and rocker joints, selected for their ability to provide degrees of freedom required to emulate human movement [2, 3].

Hybrid rigid-flexible joints, which combine rigid elements with compliant components, are designed to balance stiffness and flexibility. These are particularly valuable in applications involving human-robot interaction, especially in physical rehabilitation contexts, where adaptability and user comfort are essential [4].

Recent advances in modular and adaptive robotics have underscored the central role of joint design and control in achieving flexibility, scalability, and intuitive human-robot interaction. Modular Reconfigurable Robots (MRRs) rely on standardized joint-link modules that allow rapid reconfiguration to accommodate diverse tasks. To support their functional autonomy, a graph-theoretic framework has been proposed for automatic kinematic modeling, enhancing the adaptability and reuse of such robotic systems [8].

Complementing these structural capabilities, the assembler robot Belty demonstrates how proprioceptive actuators and kinematic couplings enable dynamic, energy-efficient, and repeatable modular assembly, even in unstructured environments [5]. In simulation domains, inverse kinematics of redundant manipulators has been effectively solved using metaheuristic algorithms such as Simulated Annealing, with dynamic behavior modeled via the Lagrange–Euler formulation [6]. These developments highlight the increasing synergy

Received: 28.04.26; **Similarities**: 07.05.26: **Reviewed**: 28.04. /28.04.26: **Accepted 1**5.06.26.

between joint design, control algorithms, and kinematic modeling.

In collaborative contexts, trajectory generation must also account for human predictability. Modeling robot motion based on rational action theory has been shown to enhance interaction quality and trust, emphasizing the importance of integrating behavioral intelligibility into joint actuation strategies [7].

Within this context, the present study explores whether a simplified RRR (Rotation–Rotation–Rotation) kinematic model implemented in Python can accurately simulate and visualize planar trajectories analogous to those of a human arm. The central research question guiding this work is: *Can a Python-based implementation of a three-joint RRR model using cylindrical joints sufficiently replicate end-effector paths characteristic of anthropomorphic motion within a two-dimensional workspace?* The underlying hypothesis is that such a simplified structure, with one rotational degree of freedom per joint, is adequate for approximating the spatial behavior of a planar robotic limb.

To test this, we derive the analytical forward kinematics of the RRR mechanism, implement the model in Python, and evaluate the resulting trajectories against expected motion patterns. Compared to proprietary tools such as MATLAB (robust robotics toolboxes but limited by licensing constraints) the proposed Python-based solution offers an accessible, open-source alternative. Its clear structure and reliance on well-supported libraries like NumPy and Matplotlib make it particularly suitable for educational use and early-stage design validation.

This work thus positions the developed tool as a lightweight, transparent platform for visualizing and understanding kinematic behavior, contributing both a functional simulator and a pedagogical resource for robotics instruction and research.

2. STRUCTURAL AND KINEMATIC ANALYSIS OF THE END-EFFECTOR

2.1 Structural Analysis

Kinematic joints can also be classified based on the geometry of the contact surface between elements, which directly influences the type of motion allowed. This classification presents two fundamental types: cylindrical and spherical. Each joint enabling specific motion profiles for different mechanical and robotic applications.

Table 1

Classification of joints based on the geometry of the contact surface

Type of Kinematic Joint	Motion	Contact Surface
Cylindrical	Rotation between two elements	Cylindrical surface
Spherical	Rotation on three directions	Spherical surface

Cylindrical joints allow rotational motion between two elements, where the contact surface is an area/portion of a cylindrical surface. These are among the most widely used in rotational motion transmission mechanisms. Typically, cylindrical joints are found in shaft couplings or gear systems and are employed in bearings or components that rotate or oscillate within a fixed plane.

Spherical joints enable rotational movement in three-dimensional space, where the relative motion occurs over a spherical surface. These joints are highly advantageous in applications requiring multi-directional mobility without permitting translational displacement. Their versatility makes them suitable for robotic joints, especially those replicating complex anatomical movements.

Based on the classification of joints by motion and contact geometry [2], this structural analysis focuses on a system with cylindrical joints (Fig. 1).

Figure 1 presents the kinematic diagram of a humanoid robot composed of multiple mechanical elements interconnected through joints. The mechanical system includes 14 kinematic links, connected by cylindrical joints. The points P1, P2, P3, P4 and P5 represent the end-effectors of the system-terminal points responsible for executing the trajectories. The joints O1, O2, O3 and O4 are fixed joints, meaning they remain stationary relative to the system's reference frame. These fixed joints provide structural stability and serve as anchors,

constraining and guiding the motion of the remaining elements. Joints A, B, C, D, E, F, G, H, K and J are cylindrical, allowing rotational motion around a single axis.

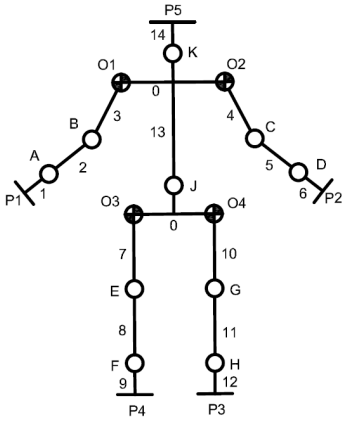

Fig. 1. Kinematic diagram of a bipedal robotic system

The elements forming the robot's arms are connected through fixed joints O1 and O2, which allow rotation about a fixed point. These joints enable the arms to move independently while remaining mechanically linked through the interconnected kinematic components.

The use of only cylindrical joints simplifies the kinematic analysis by introducing a single degree of freedom per joint and eliminating translational effects. Such configurations are common in robotic manipulators and linkage systems, offering high precision in angular positioning and control.

2.2 Kinematic Analysis

Kinematic analysis is a fundamental step in the study and design of robotic and mechanical systems, as it allows the determination of position, velocity and acceleration of various components without considering the forces involved. This type of analysis focuses on the geometric and motion-related aspects of the mechanism, offering insight into how motion is transmitted and coordinated across joints and kinematic elements.

In the structure of humanoid robotic systems kinematic analysis is important for defining the motion capabilities of this, planning trajectories and ensuring accurate control of end-effectors. By analyzing the degrees of freedom, joint types and relative motions, it can optimize the system for specific tasks such as manipulation, locomotion or interaction with the environment.

This study focuses on the forward kinematics of the mechanical system, identifying how input parameters influence the final position of the end-effector and how to compute the necessary joint configurations to reach a desired target point in space.

The kinematic model (Fig. 2) used in this analysis consists of three cylindrical joints and three links - referred to an RRR (Rotation-Rotation-Rotation) kinematic assembly. The configuration is representative of a simplified mechanical system in which each joint permits rotation about a fixed axis, contributing a single degree of freedom.

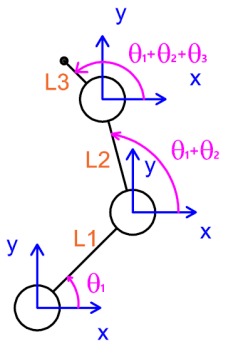

Fig. 2. Kinematic model of RRR assembly

To analyze the motion of the RRR kinematic assembly, some geometric and angular parameters must be defined. These are presented in Table 2.

Table 2

Kinematic Parameters

Parameter	Description
L_1	length of the first kinematic element - corresponds to the upper arm
L_2	length of the second kinematic element - represents the forearm
L_3	length of the third kinematic element - corresponds to the hand segment
θ_1	rotation angle of the first kinematic element relative to the base
θ_2	relative angle between the first and second kinematic element
θ_3	relative angle between the second and third kinematic element

To determine the spatial configuration of the RRR kinematic assembly it is necessary to calculate the Cartesian coordinates of key points

along the mechanical system, corresponding to anatomical reference points of the human arm: the elbow, the wrist and the end-effector.

- Position 1: Elbow Joint, corresponding to the endpoint of the first kinematic element L1. Its position is defined in terms of the first joint angle θ_1.

$$X_1 = L_1 cos(\theta_1) \quad (1)$$
$$Y_1 = L_1 sin(\theta_1) \quad (2)$$

- Position 2: Wrist Joint, representing the end of the second kinematic element L2, extending from the elbow. It is influenced by both θ_1 and θ_2.

$$X_2 = X_1 + L_2 cos(\theta_1 + \theta_2) \quad (3)$$
$$Y_2 = Y_1 + L_2 sin(\theta_1 + \theta_2) \quad (4)$$

- Position 3: End-Effector, located at the end of the third kinematic element L3, whose orientation is determined by the cumulative sum of all three joint angles.

$$X_3 = X_2 + L_3 cos(\theta_1 + \theta_2 + \theta_3) \quad (5)$$
$$Y_3 = Y_2 + L_3 sin(\theta_1 + \theta_2 + \theta_3) \quad (6)$$

These equations form the basis for forward kinematics in RRR assembly which are important for trajectory planning and motion control simulation.

3. PYTHON BASED APPLICATION CODE DEVELOPMENT

A block diagram (Fig. 3) has been constructed to outline the logical flow of a kinematic simulation program written in Python, focusing on simulating the motion of a robotic arm with three links and three joints.

At the core, Python provides built-in data types, control structures and standard libraries that enable efficient scripting and automation [9]. Its dynamic typing and automatic memory management simplify code implementation and reduce the complexity associated with lower-level languages. The code was based on the principles of several libraries like: *NumPy*, *SciPy, Matplotlib, Pandas, SymPy*, *Scikit-learn*, *TensorFlow*, *PyTorch, OpenCV, Tkinter / PyQt / Streamlit* [9] and [10].

The Python application which allowed the graphical representation of the end-effector trajectory in an RRR kinematic assembly is presented below.

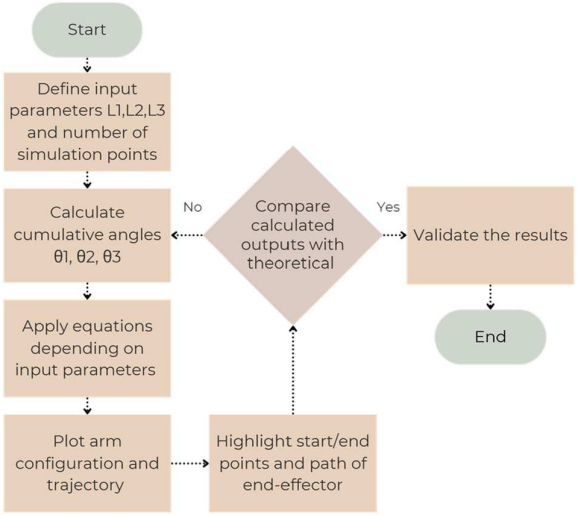

Fig. 3. Block diagram for Python-based application

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 L1 = 150
4 L2 = 130
5 L3 = 60
6 num_points = 100
7 theta1 = np.radians(45) *
np.sin(np.linspace(0, np.pi, num_points))
8 theta2 = np.radians(60) *
np.sin(np.linspace(0, 2*np.pi,
num_points))
9 theta3 = np.radians(30) *
np.sin(np.linspace(0, 3*np.pi,
num_points))
10 x_joints = []
11 y_joints = []
12 x_end = []
13 y_end = []
14 for i in range(num_points):
15 x1 = L1 * np.cos(theta1[i])
16 y1 = L1 * np.sin(theta1[i])
17 x2 = x1 + L2 * np.cos(theta1[i] +
theta2[i])
18 y2 = y1 + L2 * np.sin(theta1[i] +
theta2[i])
19 x3 = x2 + L3 * np.cos(theta1[i] +
theta2[i] + theta3[i])
```

```
y3 = y2 + L3 * np.sin(theta1[i] + theta2[i] + theta3[i])
x_joints.append([0, x1, x2, x3])
y_joints.append([0, y1, y2, y3])
x_end.append(x3)
y_end.append(y3)
plt.figure(figsize=(19.2,10.8),dpi=100)
colors = ['c', 'm', 'y']
for i in range(0, num_points, 10):
plt.plot(x_joints[i][:2], y_joints[i][:2], colors[0] + 'o-', alpha=0.3, label='L1' if i == 0 else "")
plt.plot(x_joints[i][1:3], y_joints[i][1:3], colors[1] + 'o-', alpha=0.3, label='L2' if i == 0 else "")
plt.plot(x_joints[i][2:], y_joints[i][2:], colors[2] + 'o-', alpha=0.3, label='L3' if i == 0 else "")
plt.plot(x_end, y_end, 'b-',label='End-effector trajectory')
plt.scatter(x_end[0], y_end[0], color='r', label='Start')
plt.scatter(x_end[-1], y_end[-1], color='g', label='End')
plt.xlabel("X")
plt.ylabel("Y")
plt.axis("equal")
plt.grid()
plt.show()
```

The code simulates the kinematics of a robotic arm (RRR assembly) using sinusoidal joint trajectories. The arm consists of three rigid kinematic elements, representing the upper arm (L1), forearm (L2) and wrist segment (L3), connected by cylindrical joints. At each step, the code computes the forward kinematics to determine joint and end-effector positions based on cumulative joint angles. The trajectory of the end-effector is plotted along with the intermediate arm configurations, providing a representation of the arm's motion in a 2D workspace.

4. RESULTS AND DISCUSSIONS

In this section, the input values are assigned to the parameters of the RRR kinematic model, which is designed to replicate the motion of a human arm. Using the established forward kinematic equations, the resulting positions and joint angles were computed and analyzed.

Table 3 summarizes the main values assigned to each link and joint, including link lengths and initial joint angles. These values are used for the forward kinematic calculations. Using these values, the kinematic behavior of the RRR model was analyzed by applying equations (1) to (6). These formulas describe the forward kinematics of the system, enabling the computation of end-effector positions and joint configurations for input conditions. The results obtained reflect the model's ability to replicate human arm-like motion. The results obtained are presented in Table 4.

Table 3

Input Parameters

Parameter	Values
L_1	150 mm
L_2	130 mm
L_3	60 mm
θ_1	45° = π/4
θ_2	60° = π/3
θ_3	30° = π/6

Table 4

Positions for each joint

Positions [mm]		
Elbow	Wrist	End-effector
x_1	x_2	x_3
106,07	72,41	29,99
y_1	y_2	y_3
106,07	231,64	274,06

To implement and validate the mathematical model, specific code segments were developed to implement equations (1) to (6) related to the forward of the RRR structure. This code calculates the end-effector positions in ten points, based on varying joint angles. By comparing the computed outputs with the expected theoretical behavior, the accuracy and consistency of the kinematic equations were verified. Two Python libraries were imported [9] and [10]: *NumPy*, for efficient numerical operations including matrix manipulation and

mathematical functions and *Matplotlib.pyplot*, for creating static, interactive and animated visualizations.

Initially, the three segments representing the kinematic elements (L1, L2, L3) were defined with specific length values. A series of joint angles for each revolute joint was then generated using *np.sin()* to simulate smooth, natural arm-like movements. Variables were initialized to store the coordinates of the joints and the end-effector. A for loop was used to iterate through a user-defined number of trajectory points. Within this loop, trigonometric equations - based on the kinematic model - were applied to calculate the positions of each joint. The resulting x and y coordinates were stored in *x_joints*, *y_joints* for the joints, and *x_end*, *y_end* for the end-effector's trajectory.

To represent each segment, different colors and a transparency level of 0.3 were assigned. A separate loop was used to plot the joint positions at intervals of 10 points, for a better understanding. Segment values were extracted using slicing: [:2] for the upper arm (shoulder to elbow), [1:3] for the forearm (elbow to wrist), and [2:] for the segment from wrist to end-effector, corresponding to L1, L2 and L3.

Graphical representation of the end-effector's path was generated to simulate the arm movements. The plot (Fig. 4) shows the intermediate positions of each segment (colored lines for L1, L2 and L3) at various time steps, as well as the complete trajectory traced by the end-effector (in blue). Start and end positions of the trajectory are marked in red and green respectively, providing clear reference points for the motion cycle.

The shape of the end-effector trajectory, as generated by the Python simulation, reflects the compounded influence of sinusoidal variations applied to each joint angle. These variations mimic cyclic motion patterns commonly observed in repetitive or oscillatory robotic tasks, such as pick-and-place operations or human-like arm gestures. The use of sinusoidal functions with different frequencies for each joint introduces a dynamic, non-linear interaction among the rotational motions. Specifically, the first joint varies with a single sine wave over one cycle, the second over two cycles, and the third over three, resulting in a superimposed motion pattern that produces a complex, looping trajectory. This layered approach simulates the natural asynchrony in biological limb movements, where joint motions are not always synchronized but contribute collectively to the fluidity of motion.

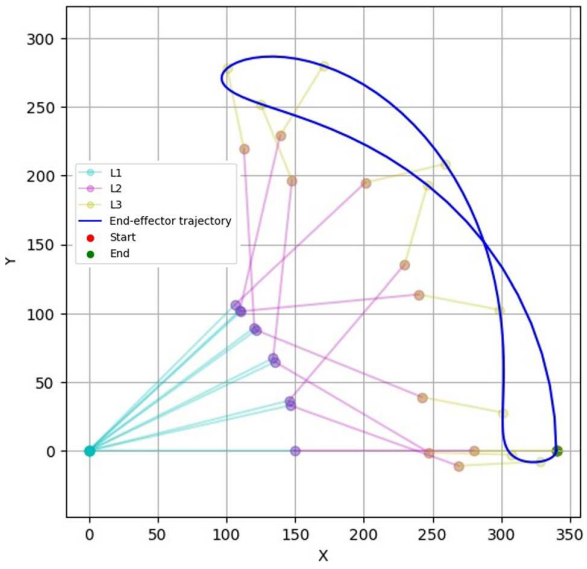

Fig. 4. Graphical representation of the end-effector trajectory obtained with Python

These implementations confirm the model's ability to mimic human-like motion within the defined range of motion for each joint. Also, the trajectory serves to validate the equations of kinematic model and helps in analyzing the behavior of the RRR kinematic assembly.

Analyzing the obtained trajectory, we can conclude that changes in frequency and amplitude of the sinusoidal inputs directly alter the curvature, reach, and periodicity of the end-effector's path. Increasing, for example, the amplitude of a single joint while keeping the others constant elongates the trajectory in a specific direction aligned with that joint's influence, whereas frequency changes introduce more oscillations or inflections. This behavior highlights the sensitivity of robotic manipulators to joint-level inputs and underscores the importance of coordinated control strategies in real-world applications. These observations should be taken into consideration when designing motion profiles for tasks requiring precision, repeatability, or biomimetic characteristics, and the current model offers a flexible platform for exploring these dynamics.

The validation of the proposed kinematic model was carried out by comparing simulation outputs with analytical results derived directly from the forward kinematic equations. Given the known input parameters (link lengths and joint angles), the corresponding end-effector positions were computed manually using trigonometric formulations. These results were then cross-checked with the output from the Python implementation, confirming a high degree of numerical agreement. In particular, the coordinates of the elbow, wrist, and end-effector (Table 4) match those expected from the closed-form expressions (Equations 1–6), thereby validating both the theoretical model and its computational translation. For instance, the end-effector position (x3, y3) calculated through the Python script matches the theoretical result obtained using trigonometric expressions based on joint angles and link lengths. This agreement demonstrates that the mathematical model used in the application faithfully captures the kinematic behavior of the RRR assembly.

It is also important to acknowledge that the sinusoidal motion patterns employed in the simulation were chosen to emulate common cyclic movements observed in robotic arms and have been widely used in existing literature for testing manipulator kinematics. While the current validation focuses on static numerical comparison, future work may involve benchmarking the simulation against real-time robotic testbeds or integrating the model into a more comprehensive physics-based environment to assess dynamic consistency.

Building upon this foundational model, several extensions can be envisioned to improve both the realism and functionality of the application. One natural evolution is the integration of real-time interactivity, where users can manipulate joint angles dynamically and observe immediate changes in the end-effector trajectory. This would require a graphical user interface or integration with platforms like Streamlit or Tkinter. Another important extension is the incorporation of spherical joints, allowing for three degrees of freedom at a single joint, thereby enabling more complex spatial movements and closer anatomical fidelity. Expanding the application into three-dimensional space and integrating with simulation platforms such as ROS or Webots could open paths to hardware deployment and autonomous control strategies.

5. CONCLUSION

The Python-based application developed in this study proved effective for simulating and visualizing the kinematic behavior of a simplified RRR robotic arm. Its advantages include ease of use, accessibility, and clear educational value, especially in environments lacking access to proprietary software or advanced simulation platforms. However, several limitations must be acknowledged explicitly, so that further research can be conducted accurately. First, the current model is limited to two-dimensional analysis using cylindrical joints only, which restricts its applicability to more complex or spatially dynamic systems. Second, the simulation does not offer real-time interactivity or physical feedback, which are essential for integration into control systems or real-world robotic platforms. Third, the sinusoidal motion profiles, while useful for demonstration purposes, do not reflect task-specific optimization or constraints. Addressing these limitations in future work, by expanding to three-dimensional kinematics, integrating spherical joints, and developing an interactive interface, will enhance both the realism and versatility of the developed tool.

6. REFERENCES

[1] Guo, X., Liu, Y., Wang, Q., Liao, X., *Spherical joint actuator with backstepping sliding mode control for robotic manipulator rigid-flexible coupling system,* IETE Journal of Research, 3985-4001, 2023.

[2] Olsen, A.M., *A mobility-based classification of closed kinematic chains in biomechanics and implications for motor control*, Journal of Experimental Biology, 222(21), jeb195735, 2019

[3] Begon, M., Andersen, M.S., Dumas, R., Multibody kinematics optimization for the estimation of upper and lower limb human

joint kinematics: A systematized methodological review, Journal of Biomechanical Engineering, 140(3), 030801, 2018

[4] Garant, X., Gosselin, C., *Whole-body intuitive physical human-robot interaction with flexible robots using non-collocated proprioceptive sensing*, IEEE Robotics and Automation Letters, 2112-2118, 2024.

[5] Romanishin, J., Rus, D., *Dynamic and repeatable modular assembly: How kinematic couplings and proprioceptive actuators simplify robotic assembly*, Springer Proceedings in Advanced Robotics, 30, 308–318, 2024.

[6] Moezzi, A., Gharib, M., Gunal, M., *Design and implementation of a graphic simulator for inverse kinematics of redundant manipulators*, MDPI Electronics, 9(2), 320-330, 2020.

[7] Karbouj, B., Alshamaa, O., Al Rashwany, K., Krüger, J., *Enhancing Human-Robot Collaborative Predictability through Rational Action Modeling of Robot Trajectories,* CIRP Conference on Manufacturing Systems, 130, 516-523, 2024

[8] Zhang, T., Du, Q., Yang, G., Wang, C., Chen, C.Y., Zhang C., Chen, S., Fang, Z., *Assembly Configuration Representation and Kinematic Modeling for Modular Reconfigurable Robots Based on Graph Theory*, MDPI, 14(3), 433, 2022

[9] *Python*, https://www.python.org/downloads/

[10] *Python Libraries*, https://docs.python.org/3/library/index.html

Aplicație bazată pe Python pentru reprezentarea grafică a traiectoriei end-effectorului într-un ansamblu cinematic RRR

Acest studiu prezintă o aplicație bazată pe Python concepută pentru reprezentarea grafică a traiectoriei end-effectorului într-un ansamblu cinematic RRR. Cercetarea se concentrează pe dezvoltarea unui cod python care facilitează analiza și vizualizarea mișcării unui manipulator robotic, ajutând atât aplicațiile academice, cât și industriale. Studiul explorează o variantă de analiză structurală folosind trei cuple cilindrice, dintre care una este fixă. Ecuațiile cinematice pentru poziție descriu mișcarea sistemului. Implementarea Python folosește biblioteci, cum ar fi NumPy și Matplotlib, utilizând o strategie de calcul eficientă pentru a modela comportamentul cinematic și a genera rezultate grafice. Aplicația permite stabilirea parametrilor de intrare de către utilizator, cu valori predefinite testate pentru a valida modelul. Codul Python este structurat și explicat, asigurând claritate și reproductibilitate. Sistemul generează grafice statice ale traiectoriei, oferind o reprezentare clară a traseului end-effectorului și validând modelul cinematic teoretic.

Angela-Miruna NEACȘU-PAVEL, PhD. Student, Assistant, National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Department of Manufacturing, angela.neacsu@upb.ro, Splaiul Independenței 313, București 060042.

Daniela TUNSOIU, PhD. Student, National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Department of Manufacturing, daniela.citu@upb.ro, Splaiul Independenței 313, București 060042.

Ileana DUGĂEȘESCU, PhD. Eng., Assoc. prof., National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Department of Manufacturing, ileana.dugaesescu@upb.ro, Splaiul Independenței 313, București 060042.

Vlad-Cristian ENACHE, PhD. Student, Assistant, National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Department of Manufacturing, vlad.enache1103@upb.ro, Splaiul Independenței 313, București 060042.

Mihaela-Elena ULMEANU, PhD. Eng., Assoc. prof., National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Department of Manufacturing, mihaela.ulmeanu@upb.ro, Splaiul Independenței 313, București 060042; Academy of Romanian Scientists, Ilfov 3, 050044 Bucharest, Romania.

Cristian-Vasile DOICIN, PhD. Eng., Professor, National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Department of Manufacturing, cristian.doicin@upb.ro, Splaiul Independenței 313, București 060042.