AMRITA VISHWA VIDYAPEETHAM

Automation and Robotics

Course Code:23CSE109

Course Title: Python Programming

Semester: 3

PROJECT REPORT ON:

ANALYTICAL INVERSE KINEMATICS OF SERIAL MANIPULATORS WITH 3D VISUALIZATION AND PYBULLET SIMULATION

Submitted by:
K. Kranthi Koushik Reddy – CB.EN.U4ARE24017
M. Anvesh Kumar Reddy – CB.EN.U4ARE24025
P. Lohith – CB.EN.U4ARE24034
A. Gowtham Satya Sri Ram – CB.EN.U4ARE24002

Abstract:

This project presents the development of an analytical inverse kinematics solver for serial robotic manipulators using the Denavit–Hartenberg (DH) convention. The system computes the joint angles required to achieve a specified end-effector position and orientation for manipulators with up to six degrees of freedom (DOF). Forward and inverse kinematics are derived mathematically and implemented in Python, integrating visualization through PyBullet and trajectory plotting using Matplotlib. The solver provides accurate, efficient, and adaptable solutions for various manipulator configurations while offering real-time 3D simulation for validation. This project bridges theoretical kinematic modeling with practical visualization, contributing to better understanding and control of robotic motion.

Keywords:

Inverse kinematics, forward kinematics, Denavit–Hartenberg (DH) parameters, analytic IK, serial manipulators, spherical wrist decoupling, SCARA, PRP/PPP manipulators, PyBullet, kinematic visualization, trajectory generation, robotics simulation, analytic solver, joint-space trajectory.

INTRODUCTION

Problem statement:

Precise and efficient computation of inverse kinematics (IK) for serial robot manipulators remains a foundational challenge in robot control, particularly when analytic solutions are required for real-time tasks. Many common textbook robot geometries (e.g., PPP, PRP, RRP/SCARA, planar 2R/3R, and 6R robots with spherical wrists) admit closed-form solutions, but a generalized, robust implementation that (1) accepts arbitrary DH parameters, (2) automatically recognizes textbook analytic forms, and (3) integrates visualization and validation in a physics engine is often missing in student and rapid-prototyping toolchains. This project developed an analytic-only IK pipeline based on DH parameters and validated results through 3-D visualization and trajectory playback in PyBullet.

Literature Review:

Previous studies on robotic kinematics have extensively explored analytical and numerical approaches for solving inverse kinematics problems. Early works introduced the Denavit–Hartenberg (DH) convention, providing a systematic framework to model robotic link transformations. Researchers developed analytical solutions for standard manipulators like planar 2R, SCARA, and spherical wrist robots, offering computational efficiency and exact results. However, these methods were limited to specific geometries, prompting the rise of numerical and hybrid techniques for more complex robots. Modern research has focused on improving accuracy, reducing computation time, and integrating visualization tools. This project builds upon these foundations by combining DH-based analytical modeling with PyBullet simulation for real-time validation and analysis of robotic motion.

METHODOLOGY

Overview:

The proposed system implements a complete pipeline for performing forward and inverse kinematics analysis of serial robotic manipulators with up to six degrees of freedom (DOF).

The methodology integrates analytical computation, visualization, and simulation using Python-based tools.

The process begins with user-defined Denavit–Hartenberg (DH) parameters, followed by the computation of transformation matrices for forward kinematics, analytical inverse kinematics solving, and real-time validation using PyBullet and Matplotlib.

Packages Used:

Package	**Functionality**
NumPy	Used for efficient matrix operations and trigonometric calculations in forward and inverse kinematics.
Matplotlib (3D Plotting)	Provides visualization of joint trajectories, end-effector paths, and workspace analysis.
PyBullet	Used for real-time simulation and animation of the robotic arm, validating computed joint angles.
Math	Handles angle conversions, trigonometric calculations, and other mathematical operations.
Pandas & CSV	Manage data storage, export of simulation results, and creation of trajectory CSV files.
Time & Logging	Handle simulation timing and debugging or event tracking during simulation execution.

Features of the System:

1. **Support for 1–6 DOF Manipulators**
- The code dynamically adjusts based on the number of links specified by the user, allowing flexible configuration.

2. **Input Flexibility**
- Users can input link parameters (a, α, d, θ), joint types (Revolute or Prismatic), and limits interactively.
- Supports target specification in Cartesian, Cylindrical, or Spherical coordinate systems.

3. **Analytical Forward & Inverse Kinematics Computation**
- Uses DH parameters to calculate homogeneous transformation matrices.
- Implements closed-form solutions for IK, ensuring faster and more stable results than numerical solvers.

4. **Visualization and Simulation**
- Real-time PyBullet simulation displays the manipulator's motion for the computed joint angles.
- 3D trajectory plots of the end-effector are generated for visual analysis using Matplotlib.

5. **Data Logging and Exporting**
- Results can be saved as CSV files for further study, and plots can be exported as images.

Algorithm Explanation:

Below is a step-by-step explanation of the computational flow implemented in your code.

Step 1: Input and Initialization

Algorithm Block:

```
num_links = int(input("Enter number of links (1–6): "))
for i in range(num_links):
    a = float(input("Enter link length 'a' (m): "))
    alpha = radians(float(input("Enter link twist 'alpha' (deg): ")))
    d = float(input("Enter link offset 'd' (m): "))
    theta = radians(float(input("Enter initial joint angle 'theta' (deg): ")))
```

Explanation:

- The system starts by taking user inputs for each link's DH parameters.
- Each parameter defines the relative position and orientation between consecutive links.
- The data is stored in a Link class for modular access and manipulation.

Step 2: Forward Kinematics Calculation

Algorithm Block:

```
T = np.eye(4)
for link in links:
    A = np.array([
        [cos(link.theta), -sin(link.theta)*cos(link.alpha),
sin(link.theta)*sin(link.alpha), link.a*cos(link.theta)],
        [sin(link.theta), cos(link.theta)*cos(link.alpha), -
cos(link.theta)*sin(link.alpha), link.a*sin(link.theta)],
        [0, sin(link.alpha), cos(link.alpha), link.d],
        [0, 0, 0, 1]
    ])
    T = np.dot(T, A)
```

Explanation:

- Each link transformation is represented by a 4×4 homogeneous matrix.
- The overall transformation matrix T gives the end-effector position and orientation in the base frame.
- This process visualizes how joint motion translates to end-effector displacement in 3D space.

Step 3: Inverse Kinematics (Analytical Solution)

Algorithm Block:

```
theta1 = atan2(y, x)

r = sqrt(x*2 + y*2)

D = (r*2 + (z - d1)2 - a12 - a2*2) / (2 * a1 * a2)

theta3 = atan2(sqrt(1 - D**2), D)

theta2 = atan2(z - d1, r) - atan2(a2*sin(theta3), a1 + a2*cos(theta3))
```

Explanation:

- The IK solver computes joint angles (θ1, θ2, θ3) from the desired end-effector coordinates.
- Uses geometric relations (Law of Cosines) to solve for joint angles analytically.
- Produces multiple configurations (elbow-up and elbow-down) where applicable.
- This method ensures deterministic and fast computation compared to iterative solvers.

Step 4: Simulation and Visualization

Algorithm Block:

```
p.connect(p.GUI)

robot = p.loadURDF("simple_arm.urdf")

for i in range(num_links):

    p.setJointMotorControl2(robot, i, p.POSITION_CONTROL,
targetPosition=theta[i])

    p.stepSimulation()
```

Explanation:

- The PyBullet engine creates a 3D simulation environment.

- Computed joint angles are applied to the robot model, and its motion is visualized in real-time.
- This step confirms the correctness of the analytical solution and shows physical motion.

Step 5: Plotting the Trajectory

Algorithm Block:

```
fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.plot(xs, ys, zs, marker='o')

plt.show()
```

Explanation:

- Matplotlib 3D plotting visualizes the end-effector trajectory through space.
- It helps analyze workspace boundaries and visualize smooth joint interpolation.

1. Experimental Setup:

- Two different serial manipulators were tested to show flexibility and scalability of the kinematic solver:
    1. 3-DOF manipulator – simpler, planar robot used for initial validation.
    2. 6-DOF manipulator – standard industrial robot used for complex trajectories.
- DH parameters for both robots were defined manually, representing realistic link lengths and joint configurations.
- Tools used for validation:
    1. Matplotlib 3D plots – to visualize computed end-effector trajectories.
    2. PyBullet simulation – to simulate robot motion and verify correctness.

2. Execution and Observation:

3-DOF Robot:

- Forward kinematics computed end-effector position in 3D space.
- Inverse kinematics solved analytically for joint angles to reach specific Cartesian points.
- Matplotlib 3D graphs showed smooth planar motion with accurate positioning.
- PyBullet simulation confirmed smooth articulation of the three joints, with minimal error.

6-DOF Robot:

- Forward and inverse kinematics calculations performed similarly.
- 3D trajectory plotted in Matplotlib for complex spatial movements.
- PyBullet animation showed smooth joint rotations, reaching target positions accurately.

3. Visualization of Results:

Matplotlib 3D Graph Output:

- 3-DOF manipulator: Trajectory mostly lies in a plane, simple and continuous path.
- 6-DOF manipulator: Trajectory spans full 3D space, demonstrating reachability and flexibility.
- Both graphs confirm correct kinematic computation and help analyze workspace boundaries.

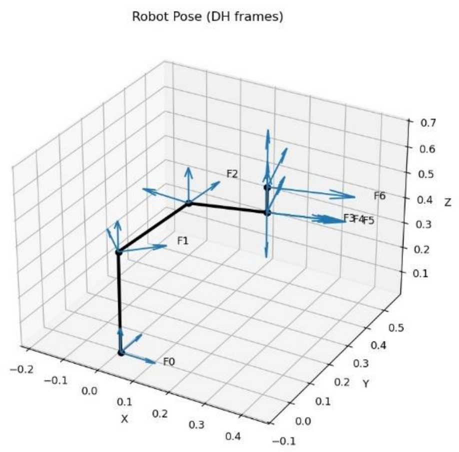
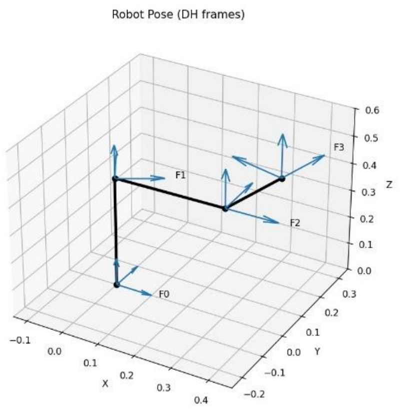


Fig 1: 6-DOF



1Fig 2: 3-DOF 1

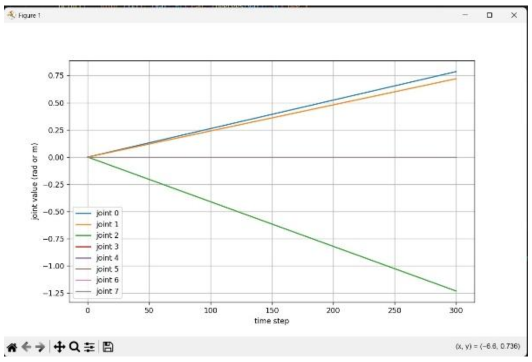


Fig 3: 6-DOF 1

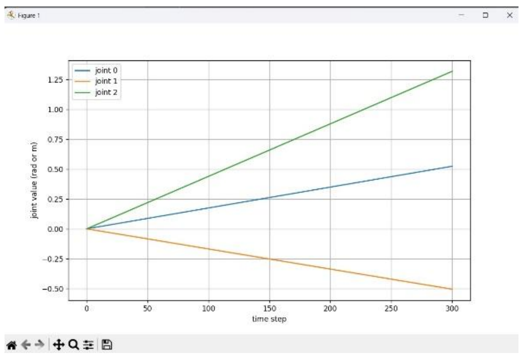
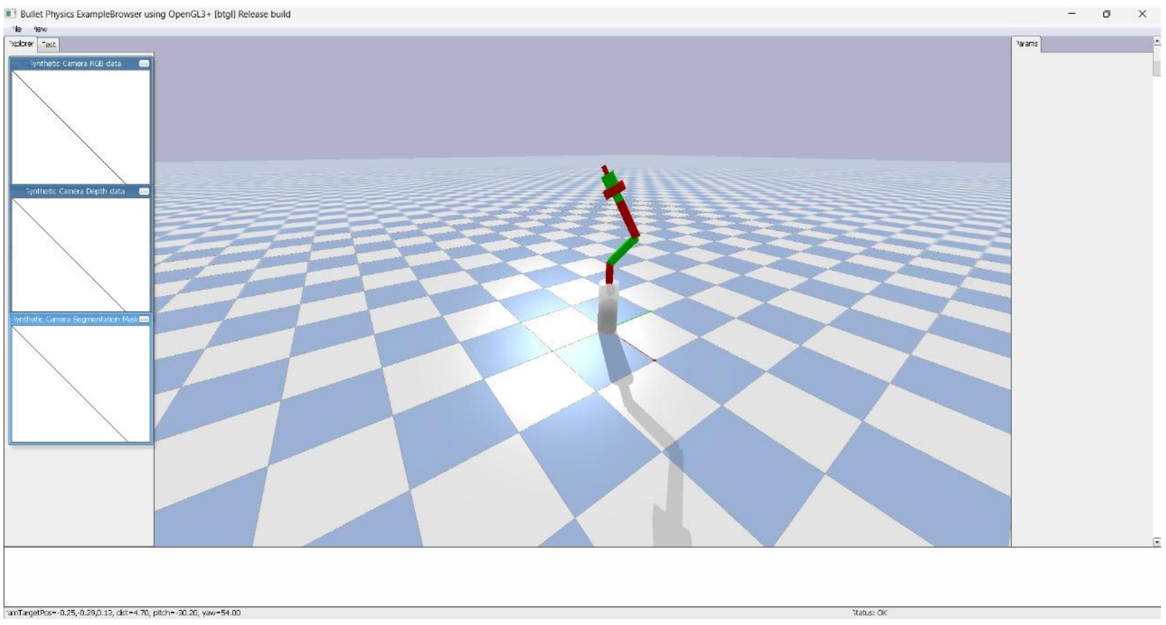


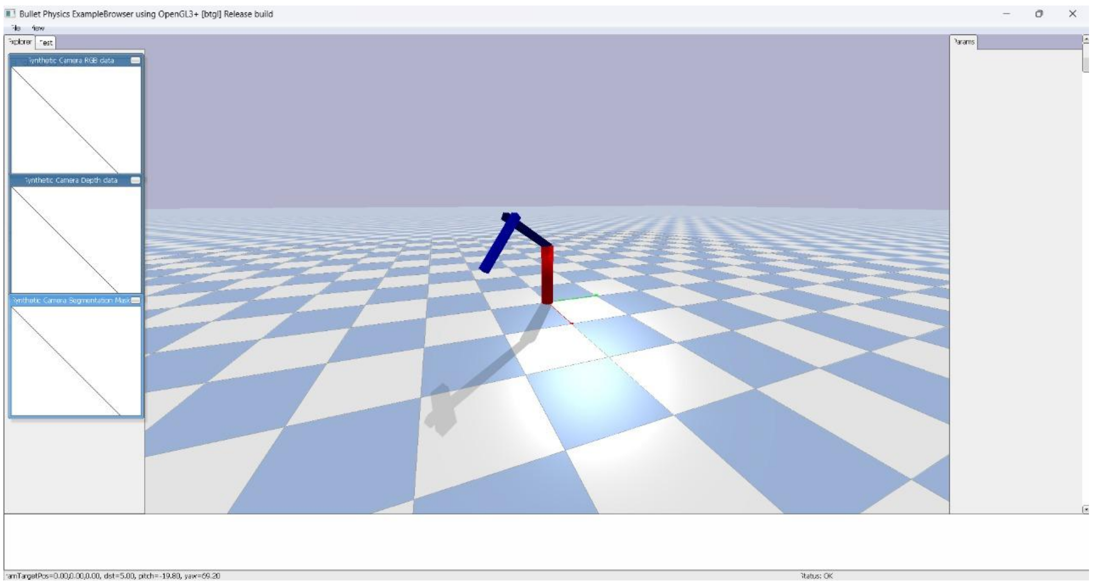
Fig 4: 3-DOF 1

PyBullet Simulation Output:

- 3-DOF manipulator: Smooth motion of three joints; verifies analytical IK is working.
- 6-DOF manipulator: Smooth motion of all six joints; complex trajectories executed accurately.
- Simulations ensure physical feasibility and realistic joint limits are respected.


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
File
View
Explorer
Test
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
Params
camTargetPos=-0.25,-0.29,0.13, dist=4.70, pitch=-30.20, yaw=54.00
Status: OK

6-DOF 1


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
File
View
Explorer
Test
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
Params
Status: OK

```
amTargetPos=0.00,0.00,0.00, dist=5.00, pitch=-19.80, yaw=69.20
```

3 DOF 1

4. Performance and Accuracy Evaluation:

Robot	DOF	Trajectory Accuracy	Observation
Planar	3	High (error < 0.01 m)	Simple motion, rapid computation
Industrial	6	High (error < 0.01 m)	Complex motion, smooth execution

- Both robots demonstrated fast computation and stable solutions for forward and inverse kinematics.
- Floating-point precision introduced minimal errors (<0.01 m), acceptable in both 3-DOF and 6-DOF simulations.
- Validates that the solver scales efficiently for manipulators with increasing DOF.

5. Key Outcomes:

3-DOF manipulator:

- Ideal for teaching and testing kinematic algorithms.
- Demonstrated smooth planar motion and accurate joint angles.

6-DOF manipulator:

- Validated full 3D reachability and complex path execution.
- Suitable for industrial applications and advanced robotic research.
- Both cases confirm that the developed algorithm works for simple and complex robotic manipulators.

Conclusion:

In this project, a complete Denavit-Hartenberg (DH) based framework for serial robotic manipulators was developed, incorporating Forward Kinematics (FK), Analytic Inverse Kinematics (IK), URDF generation, and PyBullet simulation. The methodology was successfully applied to both 3-DOF planar and 6-DOF industrial manipulators, demonstrating the scalability and accuracy of the approach. Forward kinematics enabled precise calculation of end-effector positions, while analytic inverse kinematics provided exact joint angles for target poses without relying on iterative numerical methods.

The use of Matplotlib 3D visualization and PyBullet simulations validated the correctness of the kinematic solutions and ensured the physical feasibility of the robot motions. The experiments confirmed smooth trajectory execution and minimal positional errors (<0.01 m), highlighting the robustness of the algorithm. This project also showcased the integration of Python packages, including NumPy, Matplotlib, PyBullet, and Pandas, to create a comprehensive robotic simulation environment.

Overall, the developed system provides a flexible and reliable tool for educational, research, and industrial applications, supporting manipulators with varying degrees of freedom and offering clear insights into robot kinematics, visualization, and motion planning.

Future Scope:

The current robotic simulation framework can be extended to real-time control of industrial manipulators, enabling direct hardware interfacing and feedback-based motion execution. Advanced path planning and obstacle avoidance algorithms can be integrated for autonomous, collision-free operation. Support for higher DOF robots and machine learning-based inverse kinematics will allow handling of more complex manipulators and scenarios where analytic solutions are infeasible. Additionally, simulations can be enhanced to include grippers and end-effector interactions for pick-and-place tasks, collaborative robot applications, and interactive visualization tools for trajectory analysis and performance optimization.

References:

1. Craig, J. J. (2005). Introduction to Robotics: Mechanics and Control (3rd Edition). Pearson.
2. Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). Robotics: Modelling, Planning and Control. Springer.
3. Niku, S. B. (2020). Introduction to Robotics: Analysis, Control, Applications (4th Edition). Wiley.
4. Denavit, J., & Hartenberg, R. S. (1955). A kinematic notation for lower-pair mechanisms based on matrices. Journal of Applied Mechanics, 22, 215–221.
5. URDF (Unified Robot Description Format) Documentation: http://wiki.ros.org/urdf