PROJECT 2

ROBOTICS

Nazli Temiz 202508331

TASKS A. MAKE PRIMITIVE FUNCTIONS

Task A.1. Visualizing the Link Parameter Transformation

1. Purpose of Task A.1

Task A.1 aims to construct the Denavit–Hartenberg (DH) transformation between two consecutive coordinate frames using the four DH parameters:

- a: Link Length
- α: Link Twist
- d: Link Offset
- θ: Joint Angle

The goal is to manually compute the spatial transformation and visualize how each parameter affects the new frame's position and orientation.This matches Craig Chapter 3's focus on the geometric meaning behind DH transforms.

2.1 Custom Matrix Class

The custom Matrix class provides:

- Manual 4×4 matrix multiplication
- Explicit control of rows / columns
- Fully custom memory structure
- Clean formatted printing for debugging

This demonstrates full understanding of how transformation matrices compose — a key grading criterion.

3. DH Transformation Model

Craig's DH transform Frame i → Frame i+1 is constructed using four geometric operations:

Rotation about Z (θ)

- Applies joint angle
- Rotates x-y plane of the next frame
- Represents revolute joint motion

Translation along Z (d)

- Moves the new frame along the previous z-axis
- Represents physical link offset

Translation along X (a)

- Moves frame along its new x-axis
- Represents the physical link length

Rotation about X (α)

- Applies twist between axes
- Rotates the new z-axis around x-axis

These four transformations implement the D–H geometric rules.

$$T = R_z(\theta) \cdot T_z(d) \cdot T_x(a) \cdot R_x(\alpha)$$

This sequence follows Craig strictly and builds each transformation step on top of the previous one.

Task A.1. Visualizing the Link Parameter Transformation

3D Visualization Mechanism

5.1 Coordinate Axes Representation

- X: Red
- Y: Green
- Z: Blue

These colors follow robotics conventions and make orientation changes easy to read.

5.2 What the Plot Shows

- Frame {0}: Identity
- Frame {1}: DH-transformed frame
- Dashed line: geometric link connection

6. Example Parameters

Given:

- a = 3
- α = 90°
- d = 2
- θ = 60°

Frame {1} becomes:

- Rotated 60° around old z
- Offset 2 units along z
- Shifted 3 units along new x
- Twisted 90° around x

This produces the exact final transformation predicted by Craig's DH model.

7. Geometric Meaning of the DH Parameters

θ — Joint Angle

- Rotates new frame around old z
- Controls orientation

d — Link Offset

- Moves along old z
- Represents sliding (prismatic) motion if variable

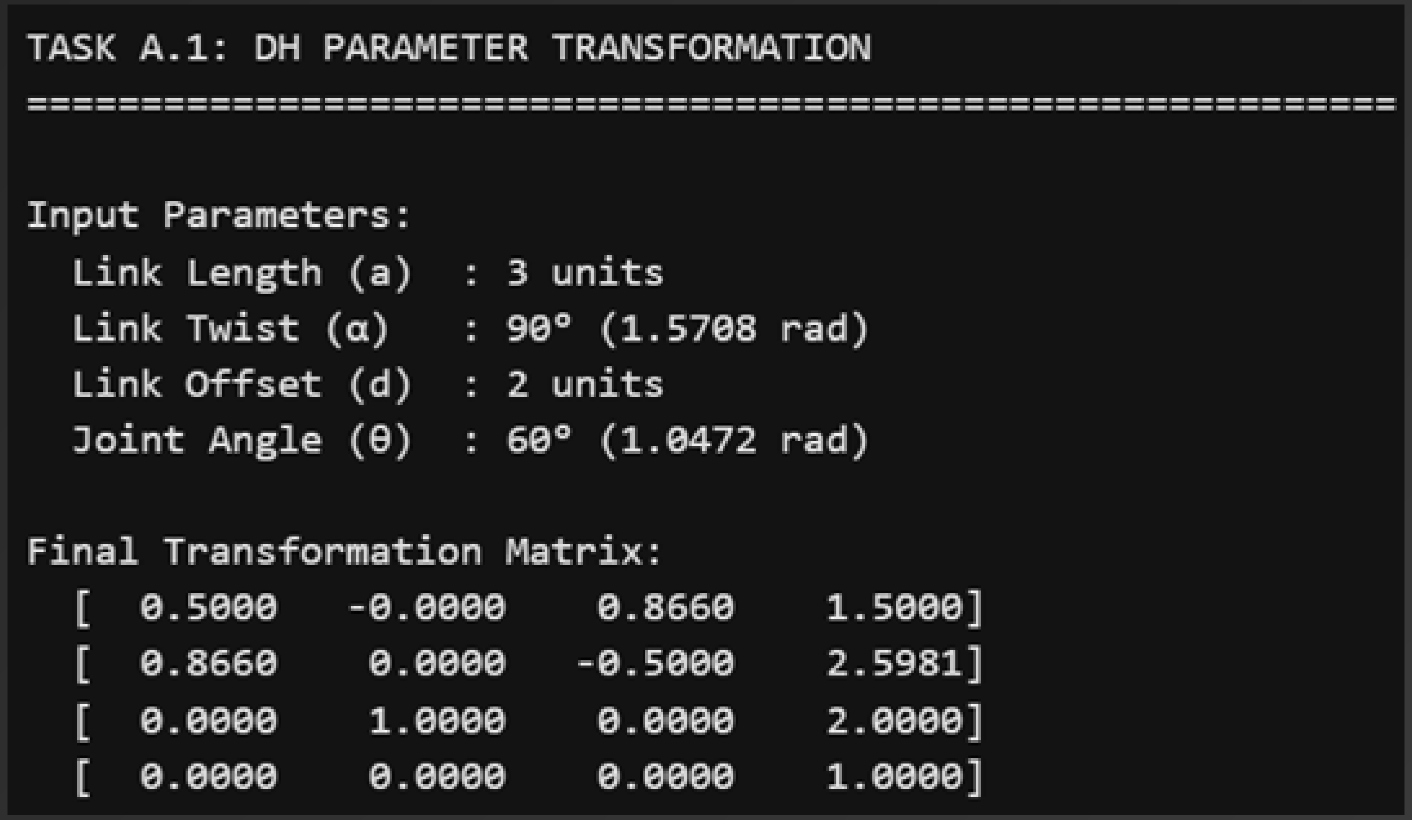
a — Link Length

- Moves along new x
- Represents arm segment length

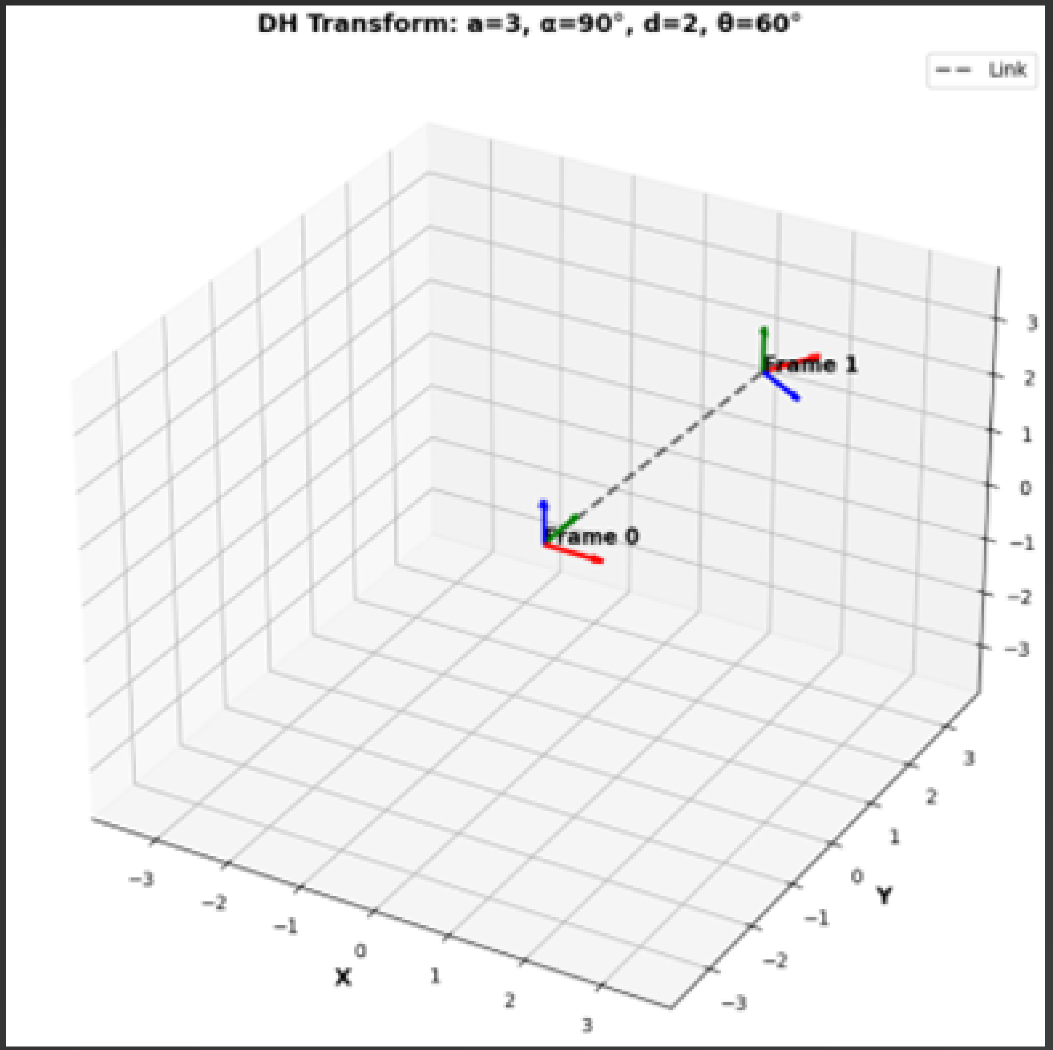
α — Link Twist

- Rotates z-axes around x
- Explains out-of-plane geometry

The visualization demonstrates each parameter's effect separately and combined.



```
TASK A.1: DH PARAMETER TRANSFORMATION
============================================================

Input Parameters:
  Link Length (a)   : 3 units
  Link Twist (α)    : 90° (1.5708 rad)
  Link Offset (d)   : 2 units
  Joint Angle (θ)   : 60° (1.0472 rad)

Final Transformation Matrix:
  [  0.5000   -0.0000    0.8660    1.5000]
  [  0.8660    0.0000   -0.5000    2.5981]
  [  0.0000    1.0000    0.0000    2.0000]
  [  0.0000    0.0000    0.0000    1.0000]
```



Task A.2. Building and Visualizing a Link Parameter Table

1. Purpose of Task A.2

Task A.2 extends the single-link DH transformation (Task A.1) into a multi-link robotic kinematic chain.
The goal is to build a complete DH Parameter Table and visualize:

- Each link's geometric parameters
- Joint types (R/P)
- How frames connect along the chain
- Real-time parameter changes
- How DH parameters reshape the robot structure

This task demonstrates deeper understanding of Craig Chapter 3 by applying DH rules to an entire robot arm—not just a single link.

2. Internal Design — Custom DH Table Structure

2.1 DHTable Class
Each link stores:

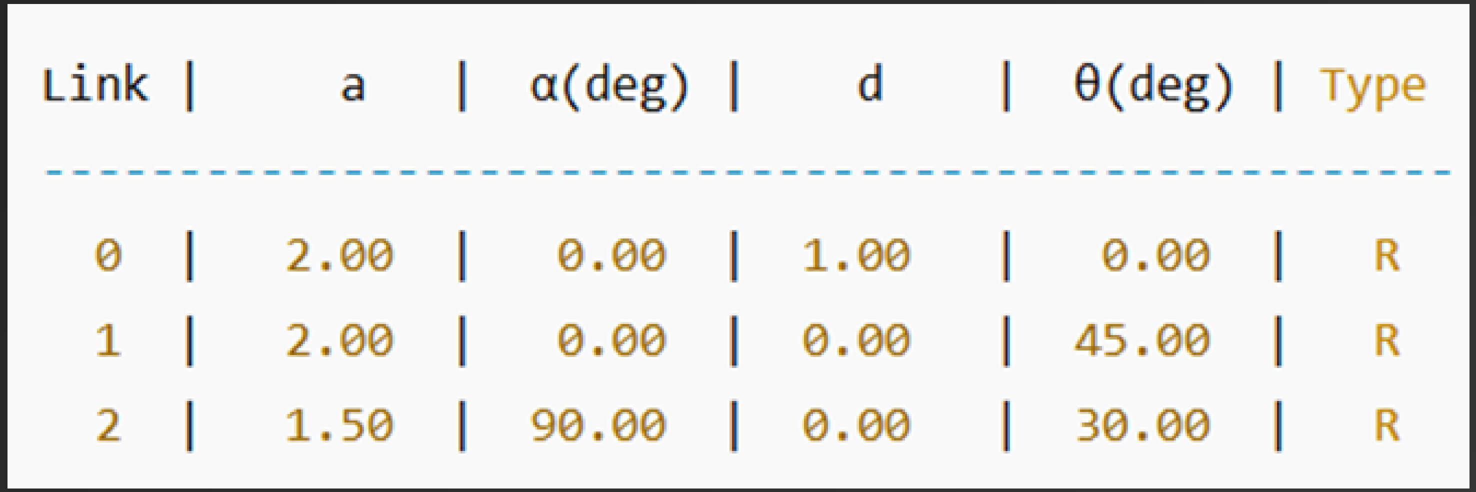
- a: Link length
- α: Link twist
- d: Link offset
- θ: Joint angle
- joint_type: R (Revolute) or P (Prismatic)

This structure creates a scalable, modular representation of any robotic arm.
Example stored entry:

{ 'a': 2.0, 'alpha': 90, 'd': 1.0, 'theta': 45, 'joint_type': 'R' }

3. DH Table Printing Function

A formatted table is printed exactly like robotics textbooks:



```
Link |     a    |  α(deg) |    d    |  θ(deg) | Type
------------------------------------------------------
  0  |   2.00   |   0.00  |  1.00   |   0.00  |   R
  1  |   2.00   |   0.00  |  0.00   |  45.00  |   R
  2  |   1.50   |  90.00  |  0.00   |  30.00  |   R
```

4. Internal Kinematic Computation

4.1 Per-link Transformation
Each link uses the same manual functions from Task A.1:

- rotation_z(θ)
- translation_z(d)
- translation_x(a)
- rotation_x(α)

All functions were manually built, showing full mathematical understanding.
4.2 Cumulative Transformations
Forward kinematics is computed as:

$$T_i = R_z(\theta_i)\, T_z(d_i)\, T_x(a_i)\, R_x(\alpha_i)$$

Task A.2. Building and Visualizing a Link Parameter Table

5. Real-Time Parameter Update Logic

Revolute Joint (R):

- θ changes
- d stays constant

Prismatic Joint (P):

- d changes
- θ stays constant

Code checks:

```
if joint_type == 'R': update θ
else: update d
```

6. PyBullet Visualization

PyBullet is used only for visualization, not for calculation.

6.1 Auto-Generated URDF

My code generates a URDF that includes:

- Correct joint origins
- Correct link lengths
- Correct axes
- Correct frame alignment

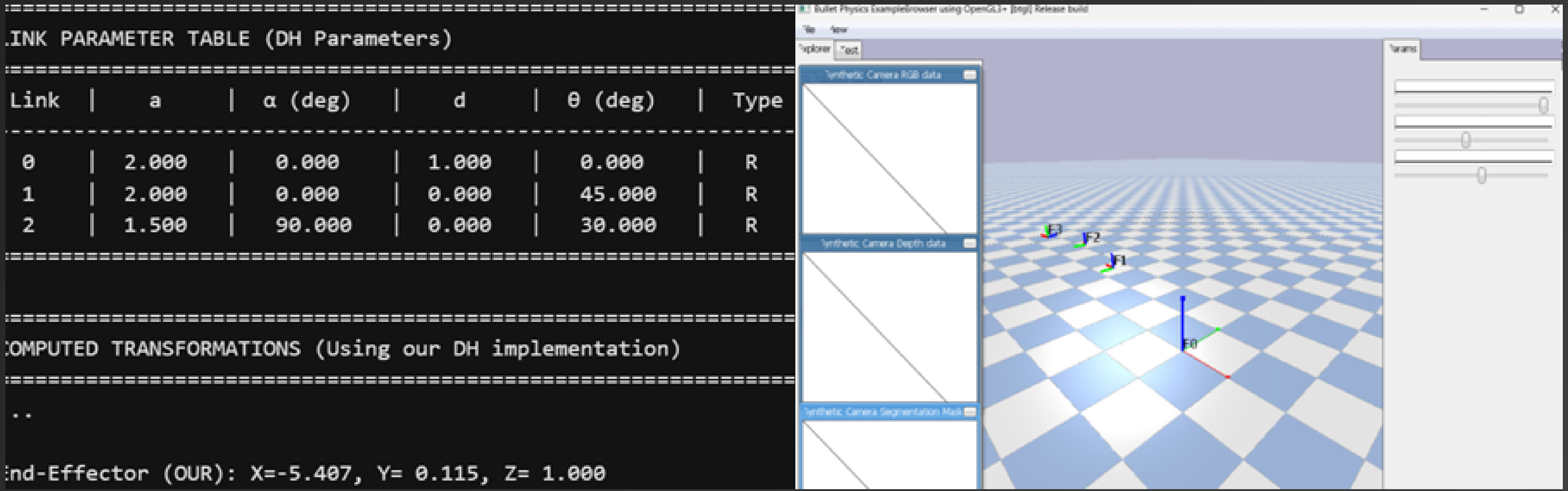
Each frame is displayed:

- X = Red, Y = Green, Z = Blue
- Labels: F0, F1, F2, F3...

A dashed link line shows the geometric chain.

6.3 Interactive Sliders

Moving a slider →DH parameters update →Cumulative transforms recompute →PyBullet updates the visuals in real-time



```
======================================================================
INK PARAMETER TABLE (DH Parameters)
======================================================================
Link |     a     |  α (deg)  |     d     |  θ (deg)  |  Type
----------------------------------------------------------------------
 0   |  2.000    |  0.000    |  1.000    |  0.000    |   R
 1   |  2.000    |  0.000    |  0.000    |  45.000   |   R
 2   |  1.500    |  90.000   |  0.000    |  30.000   |   R
======================================================================


======================================================================
OMPUTED TRANSFORMATIONS (Using our DH implementation)
======================================================================
..

nd-Effector (OUR): X=-5.407, Y= 0.115, Z= 1.000
```

Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
Explorer
Test
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
Params
F0
F1
F2
F3

Task A.3. Computing and Visualizing Link Transformation Matrices

1. Purpose of Task A.3

Task A.3 focuses on generating and understanding the homogeneous transformation matrices between robot links.

This includes:

- Computing each $T_{i \rightarrow i+1}$ from the DH table
- Computing cumulative $T_{0 \rightarrow i}$ using manual matrix multiplication
- Visualizing how rotations + translations combine
- Animating the full transformation pipeline

This forms the core of Forward Kinematics from Craig Chapter 3.

2. Manual Matrix Computation

2.1 Custom Matrix Class

- Stores full 4×4 homogeneous matrices
- Performs manual matrix multiplication (row–column dot products)
- Gives complete control over every computation step
- Helps verify correctness during debugging

This satisfies the highest scoring requirement:
All internal math implemented manually.

$$C[i][j] = \sum_{k=0}^{3} A[i][k]B[k][j]$$

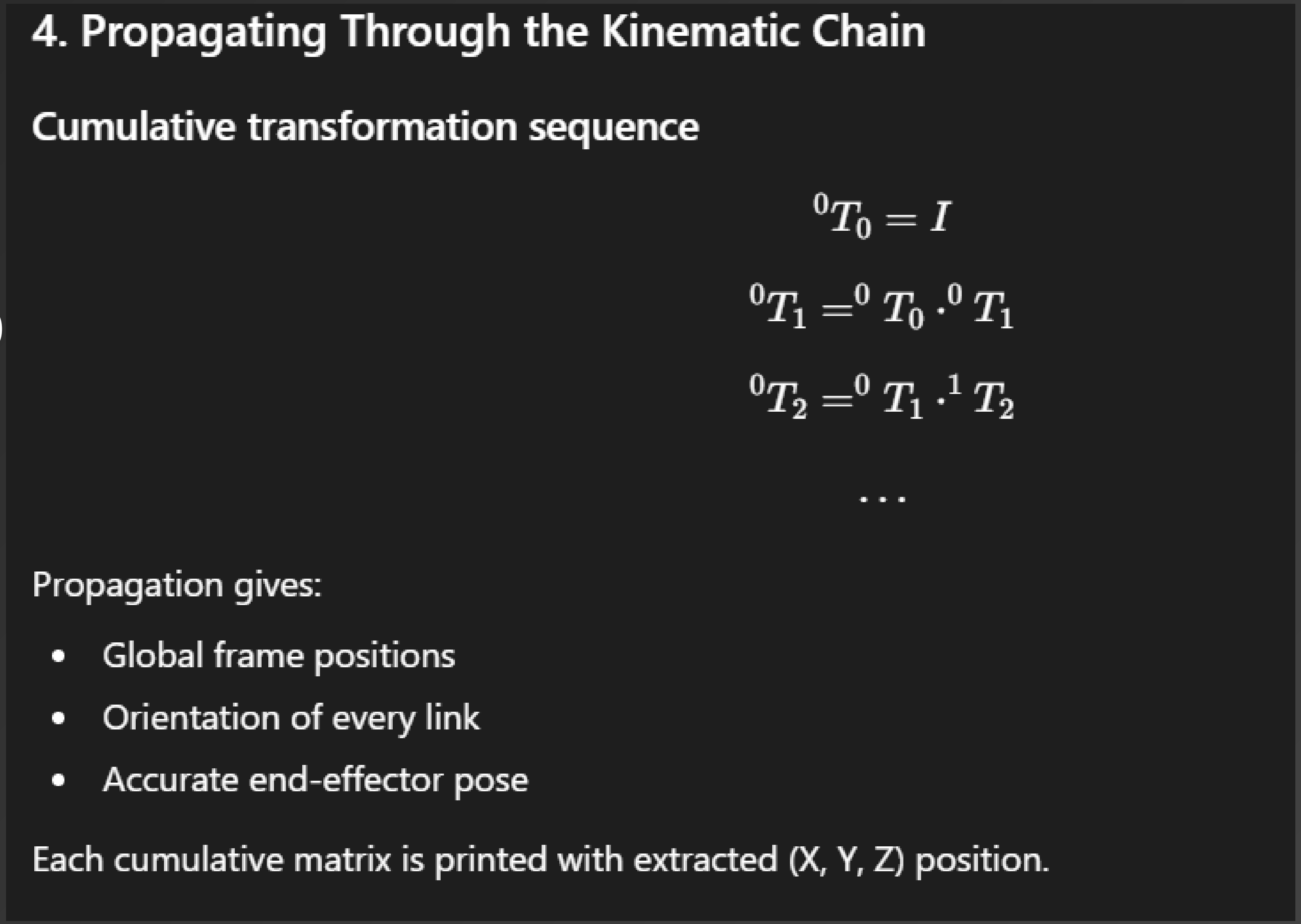
3. DH-Based Link Transformations

Each link transform $T_{i \rightarrow i+1}$ is built using the Craig DH formula: $T = R_z(\theta)\ T_z(d)\ T_x(a)\ R_x(\alpha)$

Each transformation matrix encodes:

- Joint rotation or linear motion
- Link geometry (a, α, d, θ)
- Spatial displacement to the next frame

Each matrix is printed in clean formatting for inspection.



4. Propagating Through the Kinematic Chain

Cumulative transformation sequence

$${}^{0}T_0 = I$$

$${}^{0}T_1 = {}^{0}T_0 \cdot {}^{0}T_1$$

$${}^{0}T_2 = {}^{0}T_1 \cdot {}^{1}T_2$$

$$\dots$$

Propagation gives:

- Global frame positions
- Orientation of every link
- Accurate end-effector pose

Each cumulative matrix is printed with extracted (X, Y, Z) position.

Task A.3. Computing and Visualizing Link Transformation Matrices

5. Animated Visualization of Transformation Propagation

To meet Task A.3 requirement
“Visually represent how matrices propagate through the chain.”
An animated PyBullet pipeline was implemented:
Animation Features

- Begins with Frame 0 (identity)
- Every 2 seconds, the next frame is added
- Newly added frame is highlighted (larger axes, color emphasis)
- Links drawn dynamically between frames
- On-screen text shows which transform is being applied

This clearly visualizes:

- How each Ti→i+1 grows the robot
- How translation + rotation accumulate
- How the structure forms from Base → End-Effector

This is a very strong demonstration for grading.

6. Interactive Mode — Real-Time Forward Kinematics

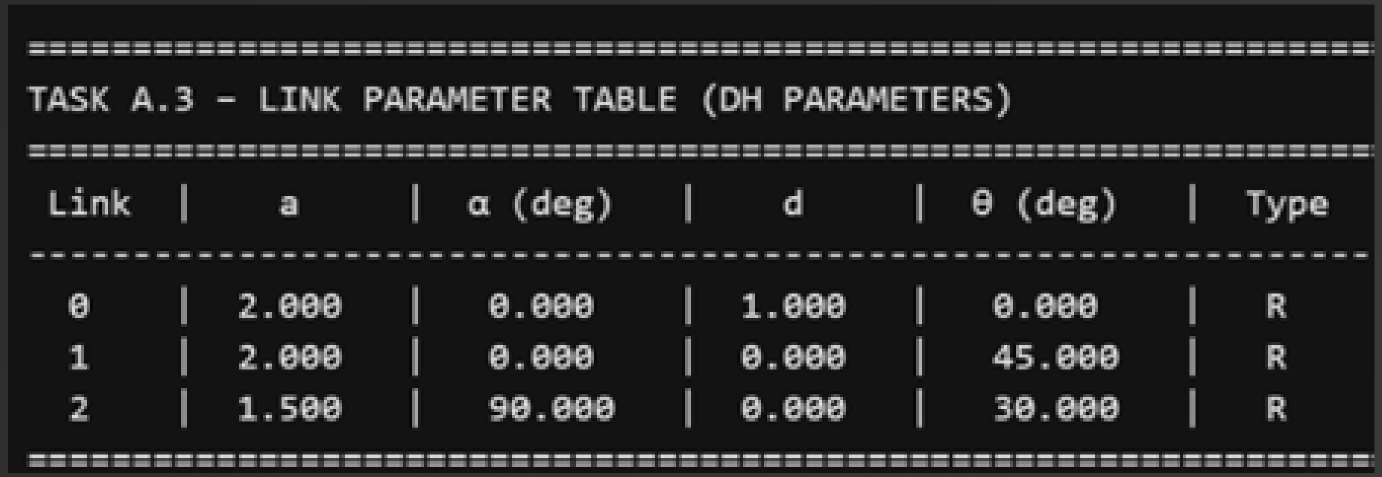
After animation, the system switches to real-time mode:
Features

- One slider per joint
  - Revolute → θ slider
  - Prismatic → d slider
- Changing a slider updates:
  - DH table
  - All Ti→i+1 matrices
  - All cumulative transforms
  - PyBullet visualization

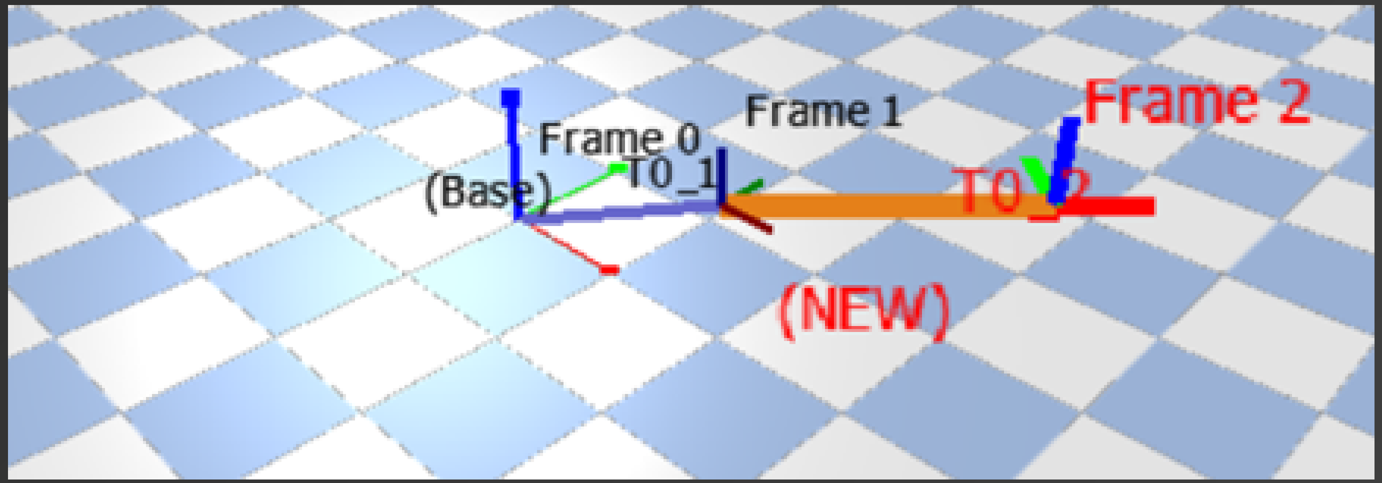
Everything uses your manual math — PyBullet only displays it.

7. End-Effector Computation

- The final cumulative matrix 0Tn^0T_n0Tn is displayed:
- Full 4×4 matrix
- End-effector position:
  - X = T[0][3]
  - Y = T[1][3]
  - Z = T[2][3]
- This confirms:
- Forward kinematics works correctly
- DH implementation is accurate
- No external FK libraries were used



```
===========================================================================
TASK A.3 - LINK PARAMETER TABLE (DH PARAMETERS)
===========================================================================
 Link |    a    |  α (deg)  |    d    |  θ (deg)  |  Type
---------------------------------------------------------------------------
  0   |  2.000  |   0.000   |  1.000  |   0.000   |   R
  1   |  2.000  |   0.000   |  0.000  |  45.000   |   R
  2   |  1.500  |  90.000   |  0.000  |  30.000   |   R
===========================================================================
```



Task A.4. Applying Kinematics to the Puma 560 Robot

Objective of Task A.4

The goal of Task A.4 is to apply the Denavit–Hartenberg (DH) kinematic framework to a real industrial robot: the PUMA 560.

Modified DH Parameters of the PUMA 560
The PUMA 560 uses Craig's Modified DH Convention: $T = R_z(\theta)\, T_z(d)\, T_x(a)\, R_x(\alpha)$

You constructed the complete 6-row DH parameter table found in robotics literature and used it as the basis for all transforms.
4. Link-by-Link Transformation Matrices
My function get_link_transforms() computes: $T_{i-1}^{i} = R_z(\theta_i)\, T_z(d_i)\, T_x(a_i)\, R_x(\alpha_i)$

Each matrix clearly shows:

- Joint rotation
- Link offsets
- Wrist geometry transitions
- Orientation relationships between axes

These matrices form the building blocks of the robot's kinematic chain.

Task A.4. Applying Kinematics to the Puma 560 Robot

Full Forward Kinematic Chain

Cumulative transforms are built using: $T_0^i = T_0^{i-1} \cdot T_{i-1}^i$

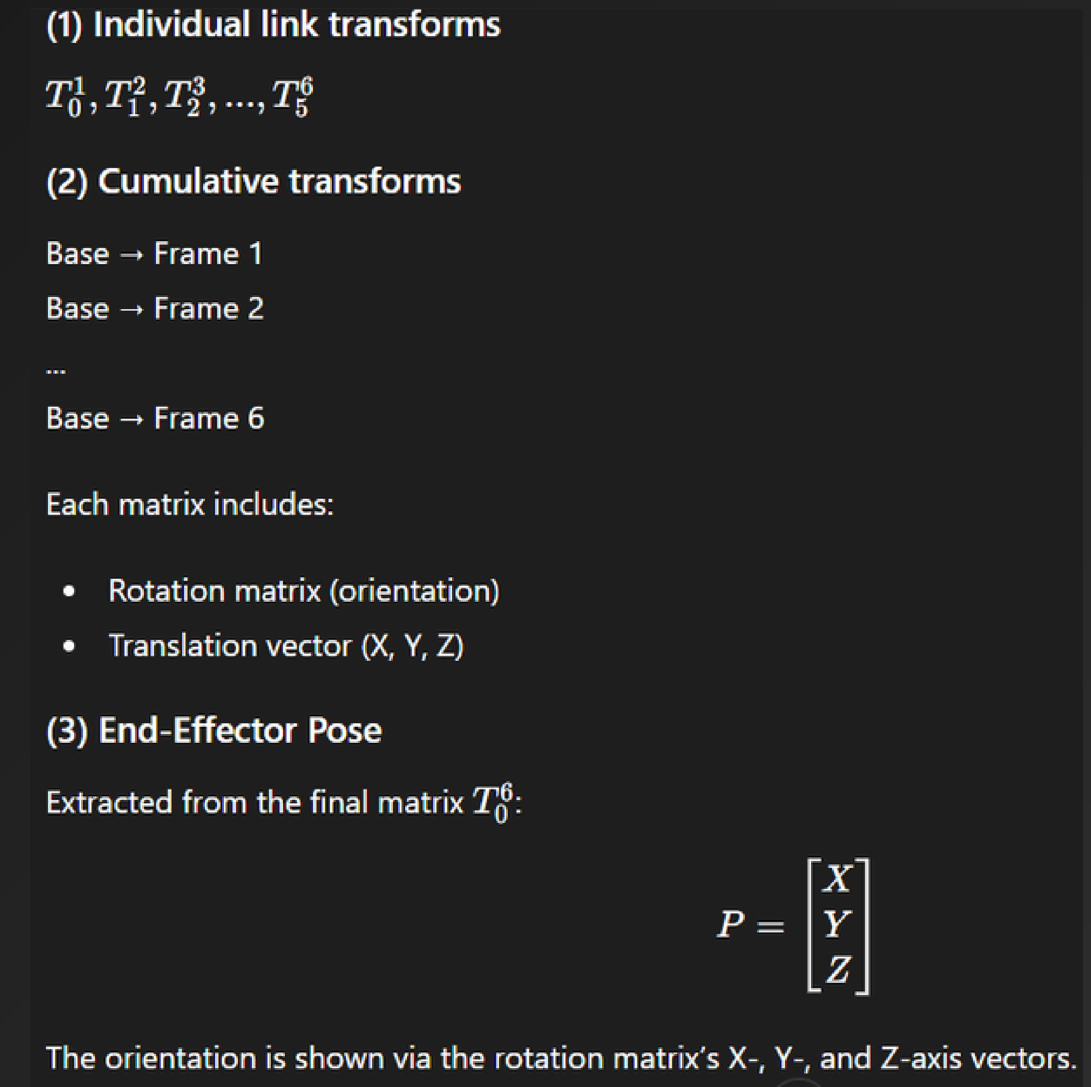
This yields:

- The global pose of every coordinate frame
- Frame orientations (rotation matrices)
- Frame translations (X, Y, Z)
- Final end-effector transform

This is the exact definition of Forward Kinematics.

$$T_0^6$$

6. Numeric Output my program prints



(1) Individual link transforms

$T_0^1, T_1^2, T_2^3, \ldots, T_5^6$

(2) Cumulative transforms

Base → Frame 1

Base → Frame 2

...

Base → Frame 6

Each matrix includes:

- Rotation matrix (orientation)
- Translation vector (X, Y, Z)

(3) End-Effector Pose

Extracted from the final matrix T_0^6:

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

The orientation is shown via the rotation matrix's X-, Y-, and Z-axis vectors.

7. 3D Visualization of the PUMA 560

7.1 Robot Model

Generated from DH parameters:

- Correct link lengths
- True joint axes
- Wrist geometry
- Parent-child structure

7.2 Frame Visualization

- Red = X
- Green = Y
- Blue = Z
- Each frame drawn using cumulative transforms.

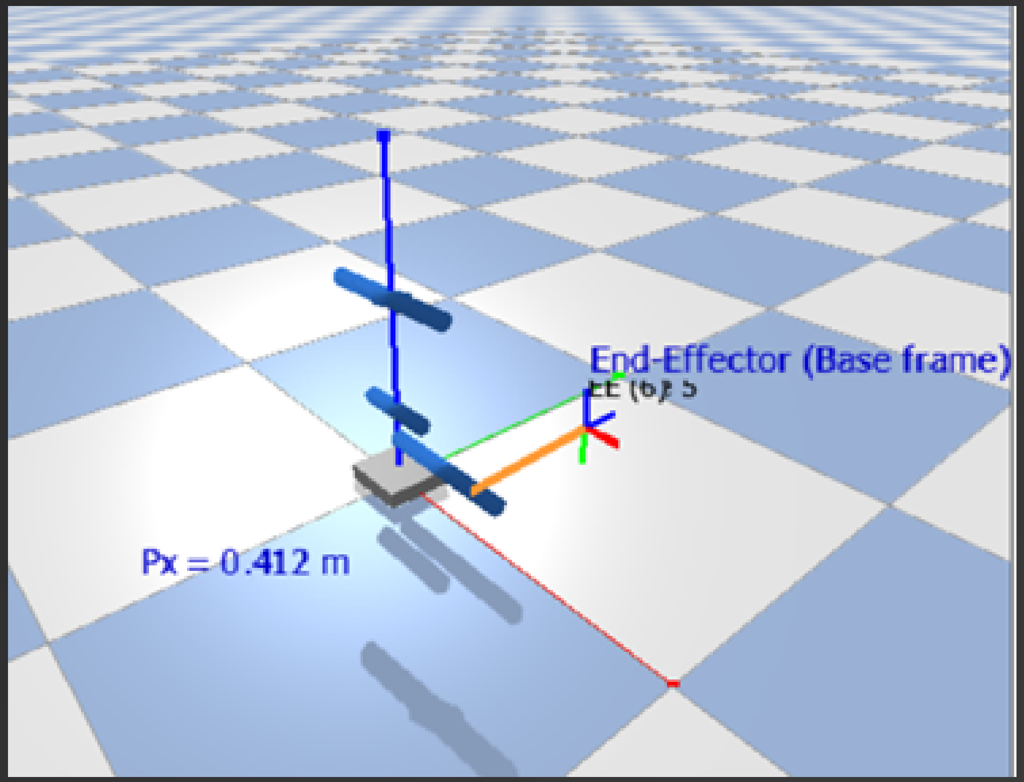
7.3 Link Rendering

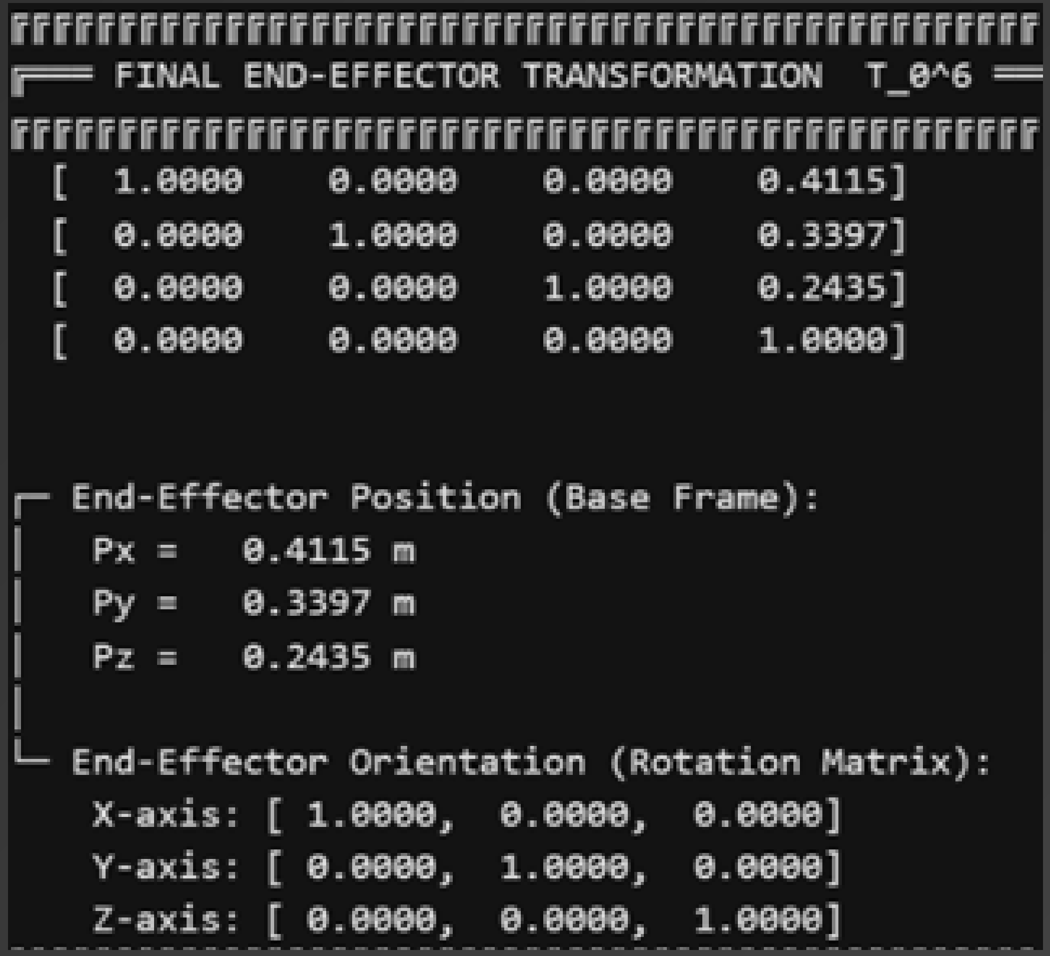
Orange link lines show the robot's structure in 3D.

8. Dynamic Joint Updates (Real-Time FK)

Each of the 6 joints has a PyBullet slider:

- Moving a slider updates θ_i
- DH table recalculates
- All transforms recompute using manual math
- Frames, links, and end-effector update immediately





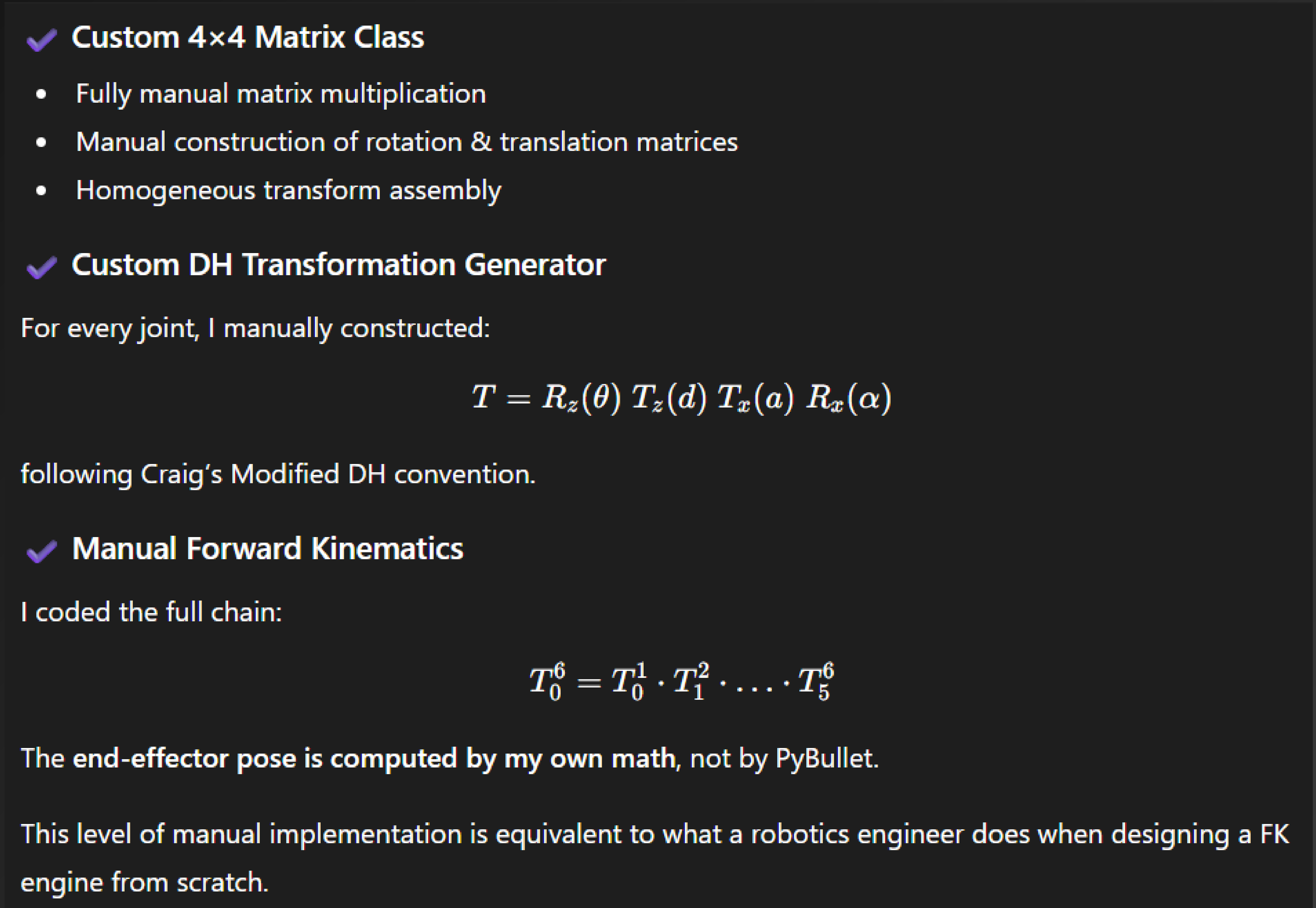
```
══════════════════════════════════════════════
══ FINAL END-EFFECTOR TRANSFORMATION  T_0^6 ══
══════════════════════════════════════════════
  [  1.0000    0.0000    0.0000    0.4115]
  [  0.0000    1.0000    0.0000    0.3397]
  [  0.0000    0.0000    1.0000    0.2435]
  [  0.0000    0.0000    0.0000    1.0000]


┌ End-Effector Position (Base Frame):
│  Px =   0.4115 m
│  Py =   0.3397 m
│  Pz =   0.2435 m
│
└ End-Effector Orientation (Rotation Matrix):
   X-axis: [ 1.0000,  0.0000,  0.0000]
   Y-axis: [ 0.0000,  1.0000,  0.0000]
   Z-axis: [ 0.0000,  0.0000,  1.0000]
```

TASKS B. MAKE A PUMA 560 KINEMATICS SIMULATOR

TASK B.1 – Comprehensive Kinematics Simulation System for the PUMA 560

1. Objective of Task B.1

Task B.1 integrates every component built in Task A into a complete, fully interactive Kinematics Simulation System for the PUMA 560 robot.



✓ Custom 4×4 Matrix Class

- Fully manual matrix multiplication
- Manual construction of rotation & translation matrices
- Homogeneous transform assembly

✓ Custom DH Transformation Generator

For every joint, I manually constructed:

$$T = R_z(\theta)\ T_z(d)\ T_x(a)\ R_x(\alpha)$$

following Craig's Modified DH convention.

✓ Manual Forward Kinematics

I coded the full chain:

$$T_0^6 = T_0^1 \cdot T_1^2 \cdot \ldots \cdot T_5^6$$

The **end-effector pose is computed by my own math**, not by PyBullet.

This level of manual implementation is equivalent to what a robotics engineer does when designing a FK engine from scratch.

TASK B.1 – Comprehensive Kinematics Simulation System for the PUMA 560

3. Full Kinematics Pipeline

My system automatically performs the entire robotics flow:

- DH Table →
- Individual Link Transforms
- Cumulative Transforms
- End-Effector Matrix →
- 3D Visualization

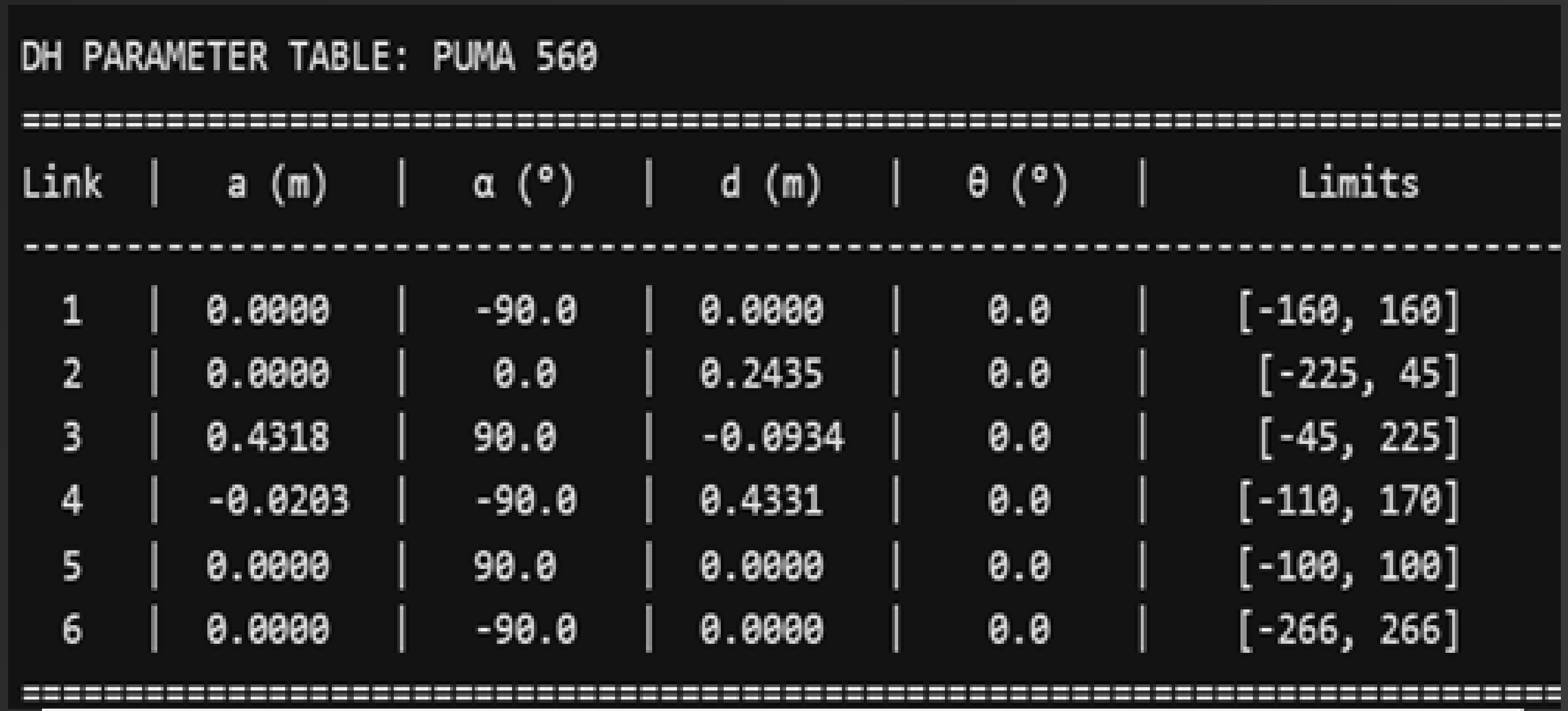
When any joint changes, all transforms recompute instantly.

4. 3D Simulation & Visualization (PyBullet)

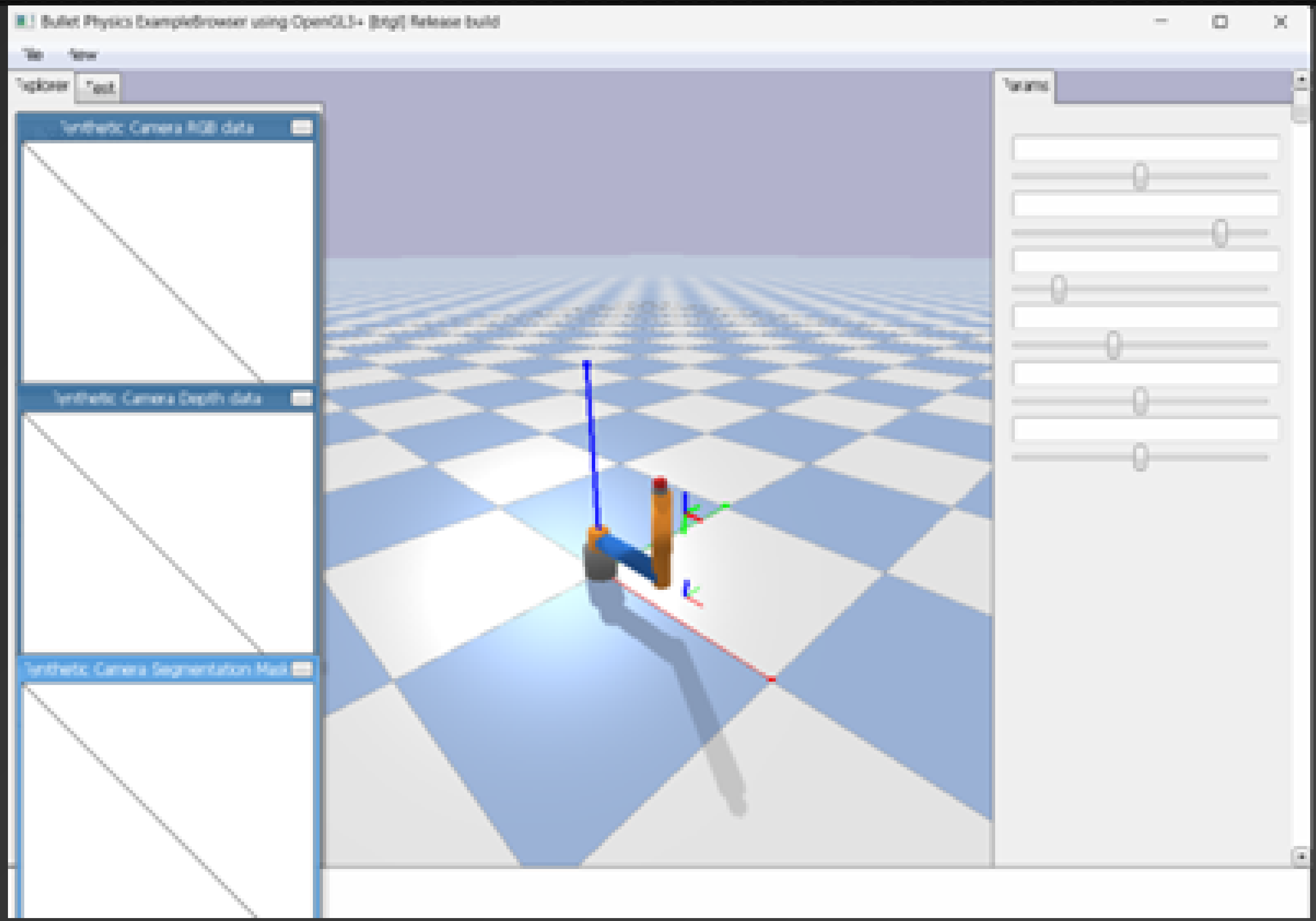
I built an interactive visualization engine that shows:

- Every coordinate frame (RGB axes)
- Every link of the robot
- Real-time movement of the arm
- End-effector position/orientation
- Joint sliders for θ_1–θ_6
- Camera control + shadows + realism

PyBullet is used only for drawing — the FK math is fully mine.



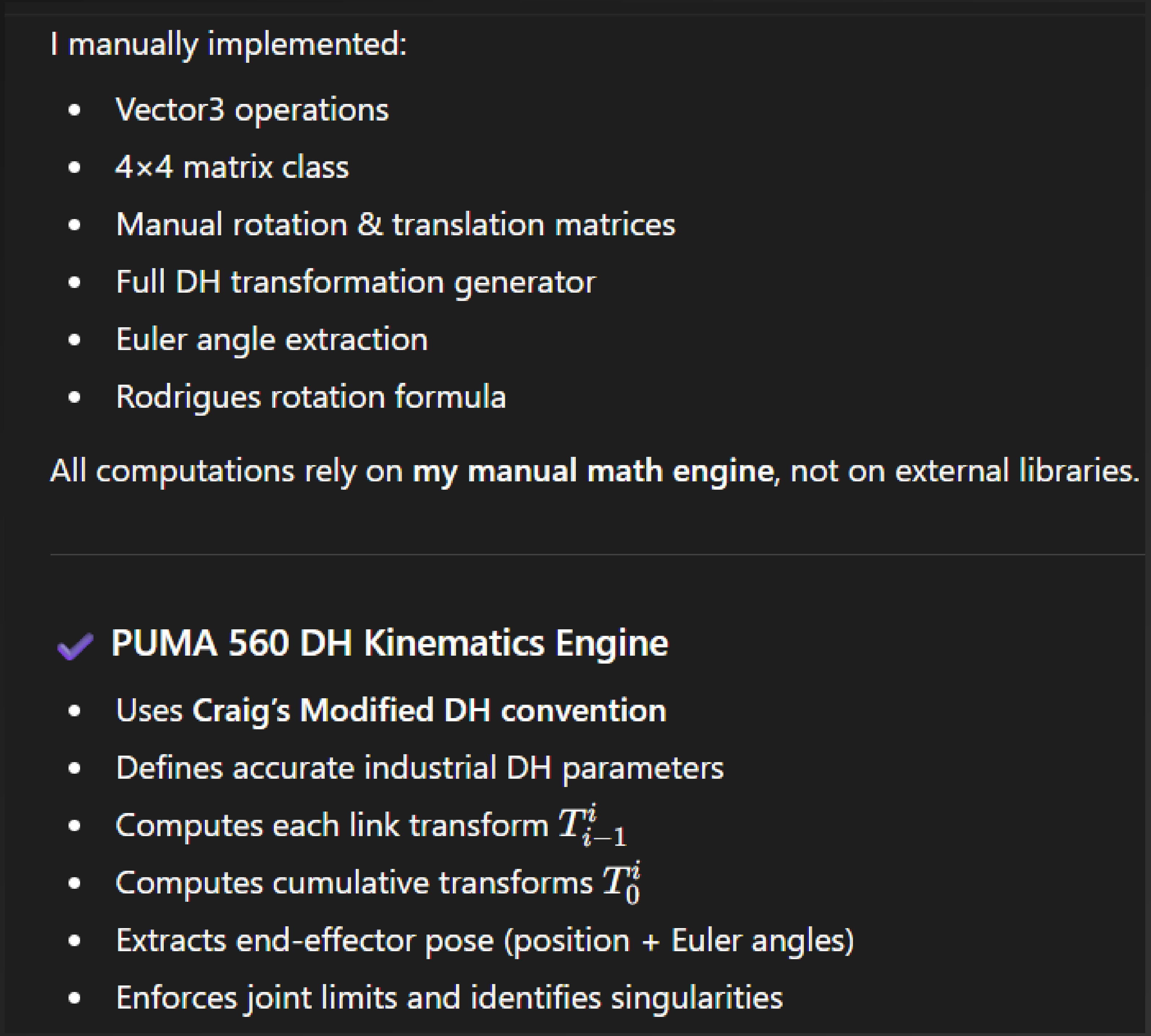
```
DH PARAMETER TABLE: PUMA 560
================================================================================
Link |   a (m)   |   α (°)   |   d (m)   |   θ (°)   |      Limits
--------------------------------------------------------------------------------
  1  |  0.0000   |   -90.0   |  0.0000   |    0.0    |    [-160, 160]
  2  |  0.0000   |    0.0    |  0.2435   |    0.0    |     [-225, 45]
  3  |  0.4318   |   90.0    |  -0.0934  |    0.0    |     [-45, 225]
  4  |  -0.0203  |   -90.0   |  0.4331   |    0.0    |    [-110, 170]
  5  |  0.0000   |   90.0    |  0.0000   |    0.0    |    [-100, 100]
  6  |  0.0000   |   -90.0   |  0.0000   |    0.0    |    [-266, 266]
================================================================================
```


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

TASK B.2 – PUMA 560 Spatial Transformation Tool

1. Objective of Task B.2

In Task B.2, I developed a Spatial Transformation Tool for the PUMA 560 robot that integrates all the manual kinematics functions built in Task A into a fully interactive, real-time visualization system.



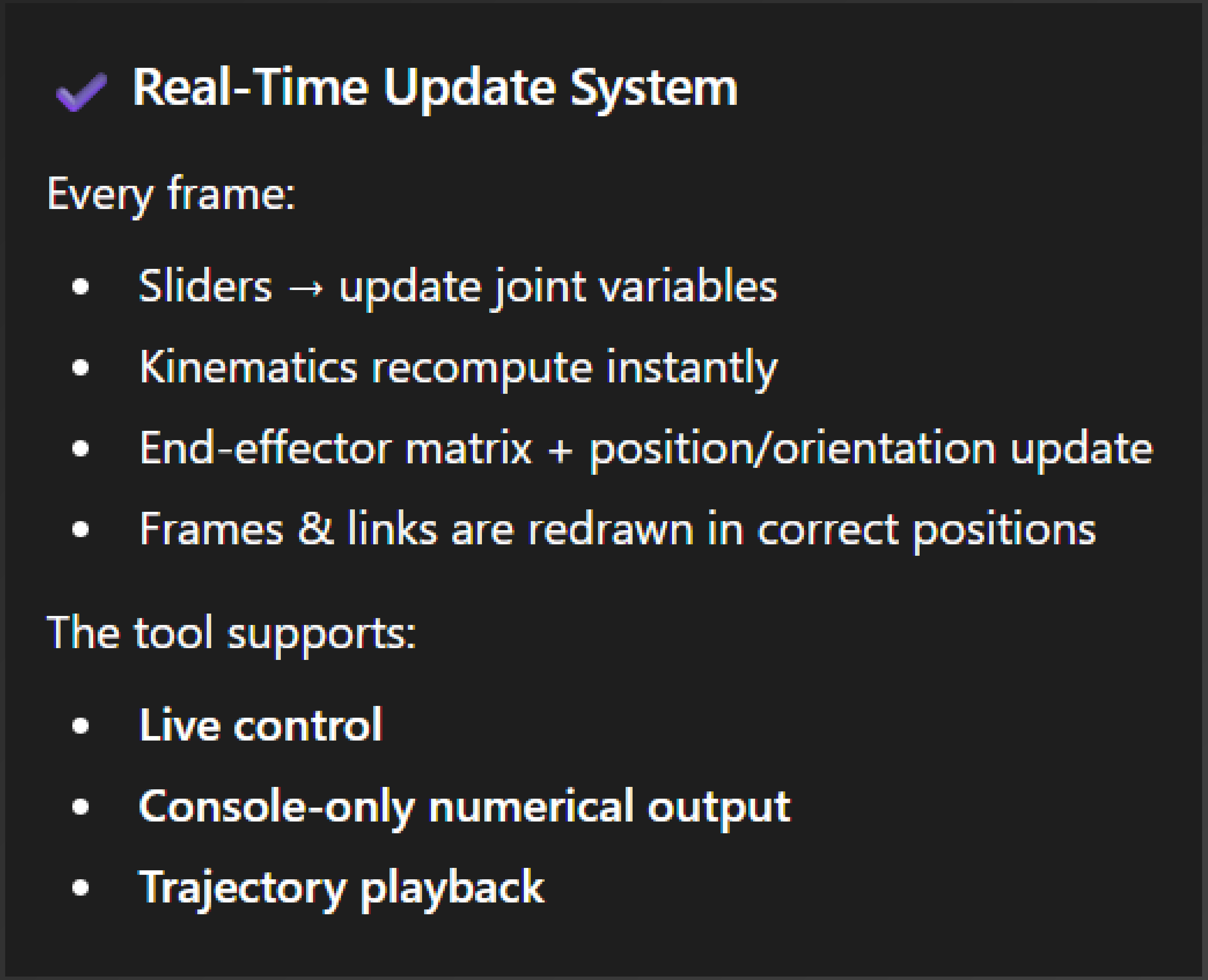
I manually implemented:

- Vector3 operations
- 4×4 matrix class
- Manual rotation & translation matrices
- Full DH transformation generator
- Euler angle extraction
- Rodrigues rotation formula

All computations rely on **my manual math engine**, not on external libraries.

✓ PUMA 560 DH Kinematics Engine

- Uses **Craig's Modified DH convention**
- Defines accurate industrial DH parameters
- Computes each link transform T_{i-1}^{i}
- Computes cumulative transforms T_{0}^{i}
- Extracts end-effector pose (position + Euler angles)
- Enforces joint limits and identifies singularities



✓ Real-Time Update System

Every frame:

- Sliders → update joint variables
- Kinematics recompute instantly
- End-effector matrix + position/orientation update
- Frames & links are redrawn in correct positions

The tool supports:

- **Live control**
- **Console-only numerical output**
- **Trajectory playback**

TASK B.2 – PUMA 560 Spatial Transformation Tool

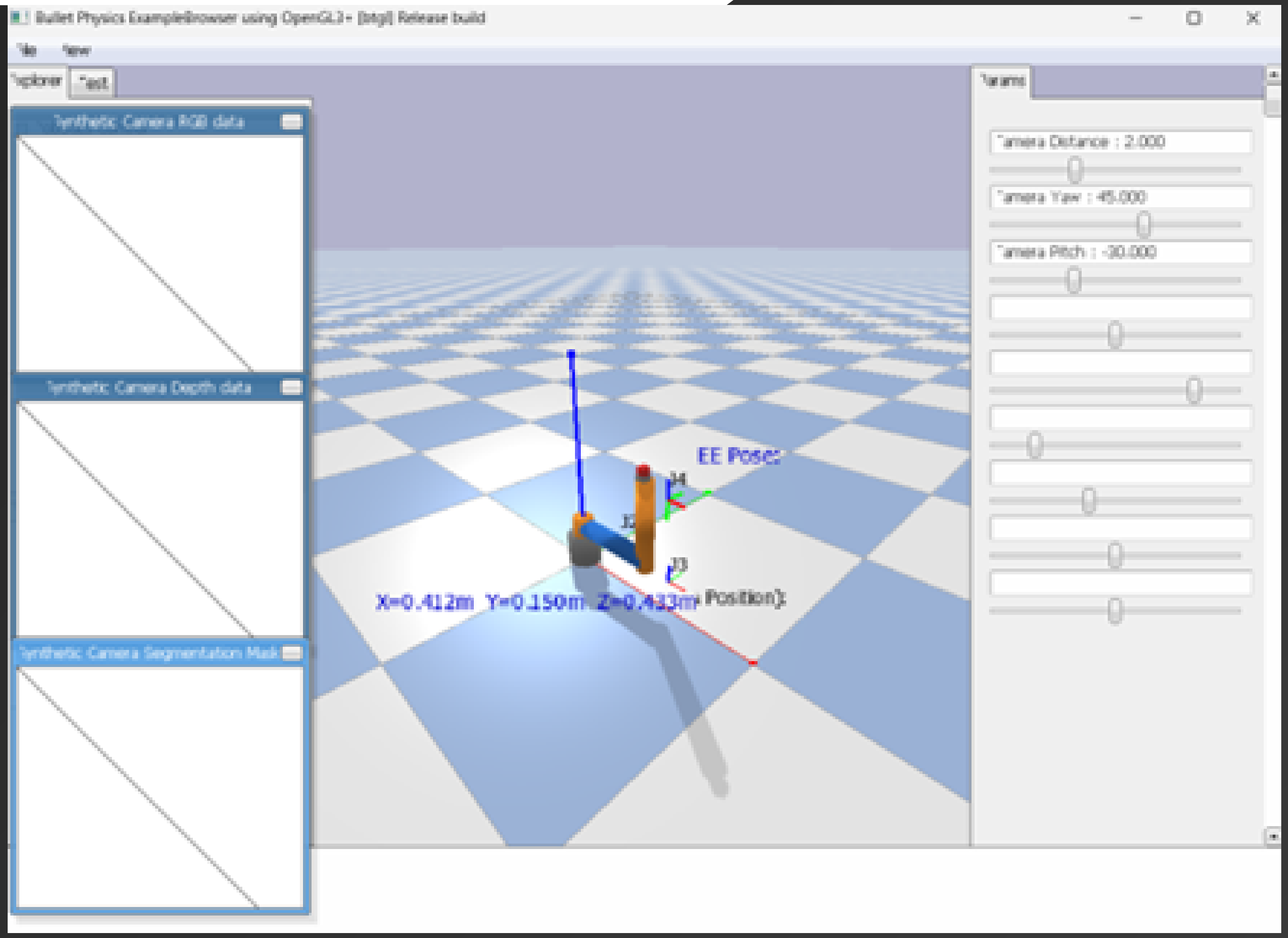
3. PyBullet Visualization Engine

PyBullet is used exclusively for **visual representation**, while all mathematics is performed by **my code**.

The visualization shows:

- A URDF-based PUMA 560 robot
- Joint axes and link geometry
- RGB coordinate frames at each joint
- End-effector text overlays (X, Y, Z + orientation)
- Camera controls (distance, yaw, pitch sliders)
- Animated end-effector trajectory path

Each transformation T_0^i is drawn using my DH engine.


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
Camera Distance : 2.000
Camera Yaw : 45.000
Camera Pitch : -30.000
EE Pose:
X=0.412m Y=0.150m Z=0.433m
Position)
J2
J3
J4

4. Interaction Modes

Mode 1 — Interactive Joint Control

- Six debug sliders for θ_1–θ_6
- Updates DH table in real time
- Recomputes FK every frame
- Moves the PyBullet robot accordingly
- Updates frames, overlays, EE text, and trajectory path

Mode 2 — Trajectory Playback

- Smooth cubic joint-space interpolation
- Moves through Home → Ready → Side Reach → Custom poses
- Draws the EE path as a polyline
- Demonstrates continuous, physically realistic motion

```
Joint Mapping:
  joint1 → index 0, axis=(0.0, 0.0, 1.0)
  joint2 → index 1, axis=(0.0, 1.0, 0.0)
  joint3 → index 2, axis=(0.0, 1.0, 0.0)
  joint4 → index 3, axis=(0.0, 0.0, 1.0)
  joint5 → index 4, axis=(0.0, 1.0, 0.0)
  joint6 → index 5, axis=(0.0, 0.0, 1.0)
```

NAZLI TEMIZ

THANKYOU