11/29/2024

Milestone #4

Robotics (MCTR911) W’24 course project

Robotics (MCTR911)

Project Milestone: MS4

Team Number: 8

**Industrial Robotics**

Seif Sabra, Dina Mamdouh, Yusuf Swelam, Anthony Usama, Faris Ebiede

This milestone focused on implementing the forward kinematics of a 5-DOF robotic arm using the Denavit-Hartenberg (DH) convention within the CoppeliaSim simulation environment. The analysis involved defining the DH parameters for each joint, developing a Python script to compute the transformation matrices, and validating the calculated end-effector position against the simulated position in CoppeliaSim. The software design utilized Python and the CoppeliaSim remote API for joint control and data acquisition. Visualization was achieved directly within the CoppeliaSim environment, observing the robot's motion and comparing the calculated and simulated end-effector positions. The results demonstrated accurate forward kinematics calculations, with the calculated end-effector position closely matching the simulated position after the robot reached its target joint configuration.

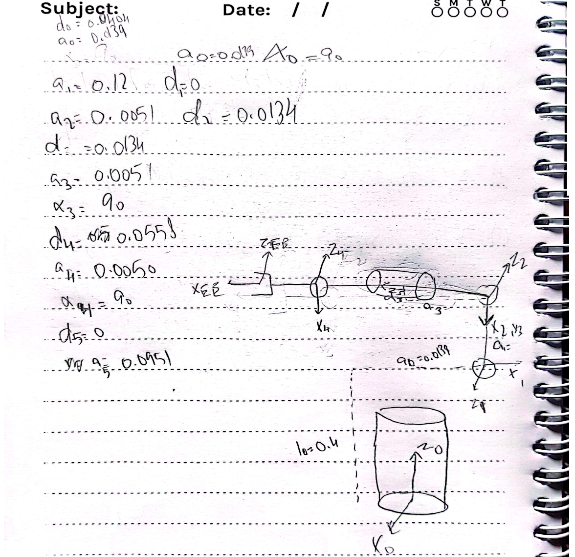
1. INTRODUCTION

The robotic manipulator is designed to seamlessly integrate into existing automated production lines, streamlining the handling of automotive components. This can dramatically increase production efficiency and accuracy, reducing manual labor and potential human error.

Versatile Handling of Automotive Products: The manipulator's ability to handle both circular and angular objects makes it highly adaptable for a wide range of automotive components, such as engine parts, body panels, and interior

Improved Efficiency and Safety: By automating handling tasks, the manipulator contributes to increased production efficiency and a safer working environment. It eliminates the need for workers to perform potentially hazardous or repetitive tasks, reducing the risk of injuries and worker fatigue.

1. TOPIC 02 (EX. ROBOT’S FRAME ASSIGNMENT)



1. TOPIC 03 (EX. DH CONVENTION)

Table 2: DH- Parameters Table

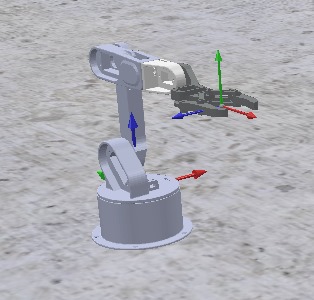
|  |  |  |  |
| --- | --- | --- | --- |
| Theta | d | a | alpha |
| q1 | 0.040 | 0.014 | 90 |
| q2 | 0 | 0.12 | 0 |
| q3 | 0.013 | 0.005 | 90.0 |
| q4 | 0 | 0.095 | 0 |

1. SIMULATION RESULTS

A close-up of a number

Description automatically generated

Initial position:

Final position:

Environment:



1. **Trajectory planning**
2. **Circular Path Trajectory:**

For the circular path trajectory, the motion is governed by the following parametric equations:

𝑥(𝑡)= 𝑥𝑐+ 𝑟cos (0.2 𝜋 𝑡)

𝑦(𝑡)= 𝑦𝑐+ 𝑟 sin (0.2 𝜋 𝑡)

𝑧(𝑡)= 𝑧𝑐

where:

* Xc and Yc are the center coordinates of the circle.
* r is the radius of the circle.
* t is the time parameter, ranging from 0 to 10 seconds.
* Zc remains constant as we assume the z-coordinate does not change.

For each time step, we compute the inverse kinematics to obtain the joint angles q1, q2 and q3.

1. Linear Path Trajectory:

The linear path trajectory is defined by the linear interpolation between the initial and final positions:

𝑥(𝑡)= 𝑥0+(𝑥𝑓−𝑥0)𝑡10

𝑦(𝑡)= 𝑦0+(𝑦𝑓−𝑦0)𝑡10

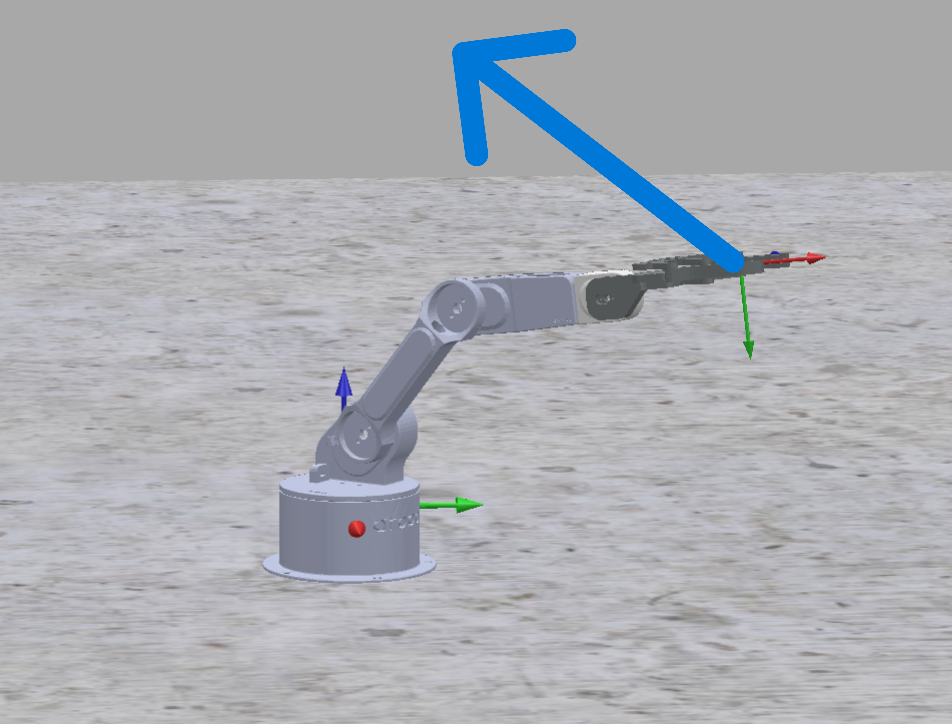
𝑧(𝑡)= 𝑧0+(𝑧𝑓−𝑧0)𝑡10

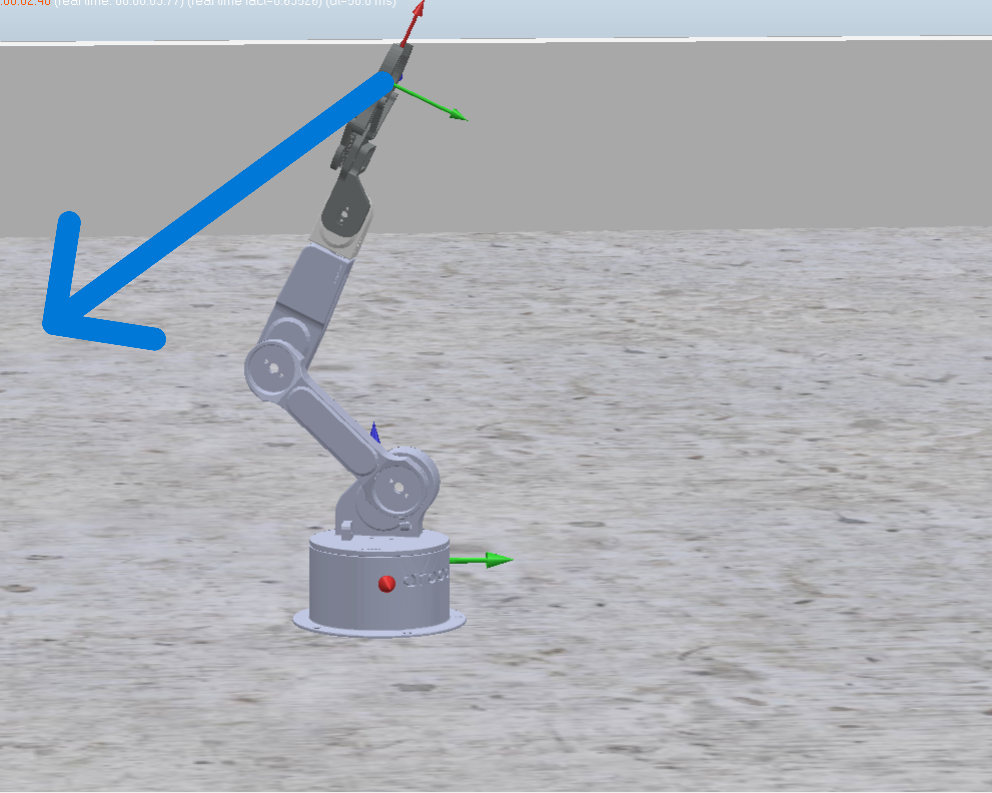
where:

* *x0,y0,z0* are the initial coordinates.
* xf, yf , zf are the final coordinates.
* t is the time parameter for the forward motion.

For the backward motion, the same equations are used but the direction is reversed.

1. **Visualization of the robot motion**





A drawing of a blue arrow

Description automatically generated

A computer generated image of a robotic arm

Description automatically generated

### **Comments on the Validation in CoppeliaSim**

To validate the trajectories in CoppeliaSim, the following steps were taken:

1. **Initialization**: The initial and desired positions were set based on the simulation environment.
2. **Trajectory Generation**: Using the defined equations, both the circular and linear trajectories were generated.
3. **Inverse Kinematics**: For each trajectory point, the inverse kinematics were solved to get the joint angles *q1q1*, *q2q2*, and *q3q3*.
4. **Simulation Actuation**: The joint angles were then set in the simulation environment, and the end-effector's position was printed to verify the correctness.

The results showed that the end-effector followed the desired paths accurately, confirming the correctness of the inverse kinematics and the trajectory generation process

1. CONCLUSINS AND FUTURE RECOMMENDATIONS

This milestone successfully implemented the forward kinematics of the 5-DOF robotic arm in CoppeliaSim. The Python script accurately computed the transformation

matrices and the end-effector position based on the provided DH parameters. The close agreement between the calculated and simulated end-effector positions validates the correctness of the implemented DH model and the forward kinematics calculations.

However, the current implementation only addresses forward kinematics. A more complete and functional robotic system requires the implementation of inverse kinematics to enable the robot to reach desired end-effector positions. Additionally, trajectory planning algorithms should be incorporated to generate smooth and efficient motions between different configurations. Finally, integrating a control strategy, potentially using Fuzzy Logic Control (FLC) or other advanced control techniques, would allow for precise and robust control of the robot arm in dynamic environments.

**Future Recommendations:**

* **Implement Inverse Kinematics:** Develop an inverse kinematics solver to determine the joint angles required to achieve desired end-effector positions and orientations.
* **Trajectory Planning:** Integrate trajectory planning algorithms to generate smooth and time-optimal paths for the robot arm.
* **Control System Design:** Implement a control strategy, such as Fuzzy Logic Control (FLC) or PID control, to accurately track desired trajectories and compensate for external disturbances.
* **Collision Detection and Avoidance:** Incorporate collision detection and avoidance algorithms to ensure safe operation of the robot arm in complex environments.
* **Dynamic Simulation:** Extend the simulation to include dynamic effects such as gravity, inertia, and friction for a more realistic representation of the robot's behavior.