

Design and Computational Modelling of a 3-DoF Robotic Arm for Assistive Object Retrieval

Mr. Kiyan Maulik Vora

Student, School, Aditya Birla World Aacdemy

Abstract

Designed a 3-DoF robotic arm for assistive object retrieval using Onshape CAD and 3D printing. Developed Python algorithms using NumPy to implement rotation matrices and homogeneous transformations. Extended the model to support rotations about arbitrary pivot points. Tested coordinate transformations through mathematical and geometric validation. Explored applications in assistive robotics and identified pathways towards integrating physical arm geometry with computational control.

Robotic arms are highly promising technologies in the domain of assistive technologies that allow people with limited mobility to gain a high degree of independence in executing daily operations, such as reaching out to an object or taking medicine. Thus, in August 2026, there were approximately 16.8 million disabled people in the United Kingdom. Mobility impairments were the most frequently reported types of impairments amongst working-age disabled adults, which comprise 42 percent of the entire group. The problem may be exemplified by stroke; estimates suggest that the rate of persistent upper extremities impairment in people who suffered stroke varies between 40% and 70%.

One of the key problems associated with assistive arms is the need to translate the coordinates of the target into movements of arm's joints. The present study focuses on this challenge from a fundamental perspective and considers a conversion of the coordinates from the global to the local frame through rotation and translation of an object. The project aims to demonstrate this conversion by means of a 3-DoF arm designed in computer-aided design (CAD).

This project was started due to a keen personal interest in robotics and theoretical physics and is aimed at applying the two to create something useful. While personally I do not have anyone with this impairment in my circles of acquaintances, the problem itself is recognized as one having wide application all over the world. Thus, this project is undertaken with the intention of assisting people who face difficulties.

The current document provides information about background research and assumptions, a design of a 3-DoF robotic arm, its mathematical development, software implementation and testing, and finally, reflections on the process.

Project objective and success criteria

The objective of the project is to design a 3-DOF robotic arm in computer-aided design (CAD) and to develop a computational model, employing rotation and homogeneous transformation matrices capable of transforming coordinates from the global (real-world) coordinate frame into the arm's local coordinate

frame. This transformation will constitute a primary computational step in a reach-and-retrieve task in an assistive robot. The project is considered successful when the following achievements are obtained:

- A complete CAD model of a 3-DOF arm is created with specified lengths of links and joint locations proper for a compact (desk- or wheelchair-mountable) reaching device.
- Code is written for calculating the transformation of the coordinates for any translation and a sequence of rotations.
- Outputs of the code are tested to check the accuracy of the transformation.
- An example of the actual transformation of coordinates of a real-life object is provided.
- Objectives mentioned above are achieved according to the list below.

Objectives

- Investigation of the rotation matrices and homogeneous transformations and their applications in robotics including common conventions used in robotics to define and transform coordinate frames.
- Overview of the existing robotic arms for assistive technologies and understanding of their design constraints and use cases.
- Design of a 3-DOF arm in CAD specifying lengths of the links, kinds of joints, and their movement ranges proper for a reaching task at arm's length.
- Derivation of a mathematical relationship between the global and the local coordinate frames in a case of a sequential transformation consisting of a sequence of rotations and a translation.
- Development of the Python code with utilization of the NumPy to calculate rotation matrices for X, Y, and Z axes and to obtain a homogeneous transformation matrix.
- Testing the code for ensuring correctness of the calculation of the transformation.
- Analysis of the results.

Approximately 16.8 million people in the United Kingdom were reported to have a disability in 2023/24, with the mobility impairment being the most frequently reported kind of impairment of working-age disabled adults comprising 42% of the population group (House of Commons Library, 2024). Moreover, stroke and other conditions affecting upper limbs may result in lasting deficits that affect people's ability to perform everyday activities. A compact robotic arm for reaching nearby objects may provide a lot of independence without being too expensive for people affected by such conditions.

Any robotic arm addressing this issue would be aimed at solving one general problem, which is a transformation of the coordinates of a target in the real world into the coordinate system of the arm. Thus, the present project directly addresses this problem and develops a framework which is applicable to future addition of sensors and grasping tools and further solution of inverse kinematics.

There were several possible methodologies that were applicable to different parts of the project:

- Transformation of the coordinate frame method: rotation matrices calculated using the Euler angles are straightforward to comprehend and apply to physical rotations while avoiding some drawbacks such as gimbal lock. Quaternion-based transformation matrices also avoid gimbal lock and are efficient computationally, however, they are harder to derive manually. Finally, DH parameter description method is a well-established convention in robotics and can be used to describe a kinematic chain; however, this method requires additional effort. Since a 3-DoF system is rather simple and the current project concentrates on a thorough understanding of the underlying mathematics, rotation matrices based on the Euler angles were chosen.

- CAD and mechanical design issues concerning actuation: servo motors have position control functionality and can easily connect with a microcontroller; therefore, they were chosen for a small-scale project involving a 3-DoF arm without a need for special driver circuitry and additional logic. In turn, stepper motors allow higher precision and repeatability, however, they require more complex driver circuits and coding to function properly. Finally, DC motors with encoders combine speed and high torque; however, they have the same requirements as stepper motors. Hobby servo motors with built-in drivers were used as actuators for the arm joints because they were appropriate for this particular 3-DoF configuration without any extra hardware.

For the creation of the arm's structural elements, additive manufacturing allows quick creation of custom shapes of links and joints with precise alignment according to the design with a comparatively cheap cost and fast iteration during the development process. On the other hand, printed components will be less durable and precise comparing to machined ones. In turn, ready-made components provide higher durability but no manufacturing delay but are limited by the design by usual dimensions and geometries, so they limit flexibility of the arm's geometry. It was decided to 3D print the components of the arm since this approach is cost-effective and allows production of custom shapes taken from the CAD model.

The arm was designed in Onshape as a 3-DoF manipulator consisting of a rotating base joint, two connecting links, and a two-pronged gripper end effector. Figures 1-3 show the completed CAD model from different angles and configurations, illustrating the arm's range of motion.

Figure 1: Isometric view of the arm in a mid-extended pose, showing the base joint, both links, and the gripper.

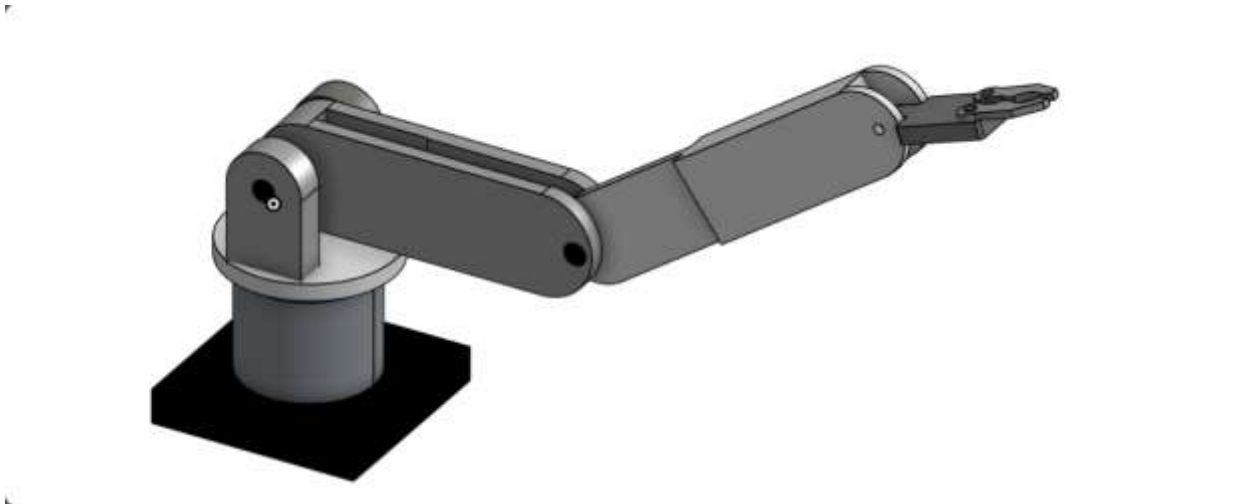
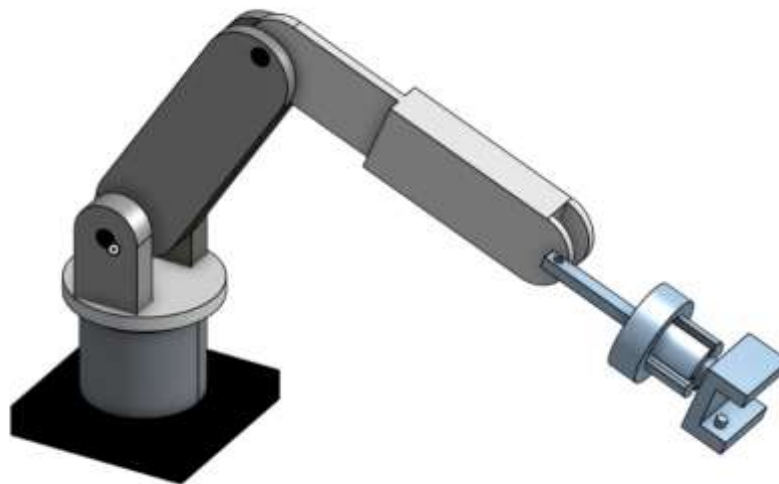


Figure 2: The arm raised into a steeper pose, showing the second link's rotation range and the open gripper.



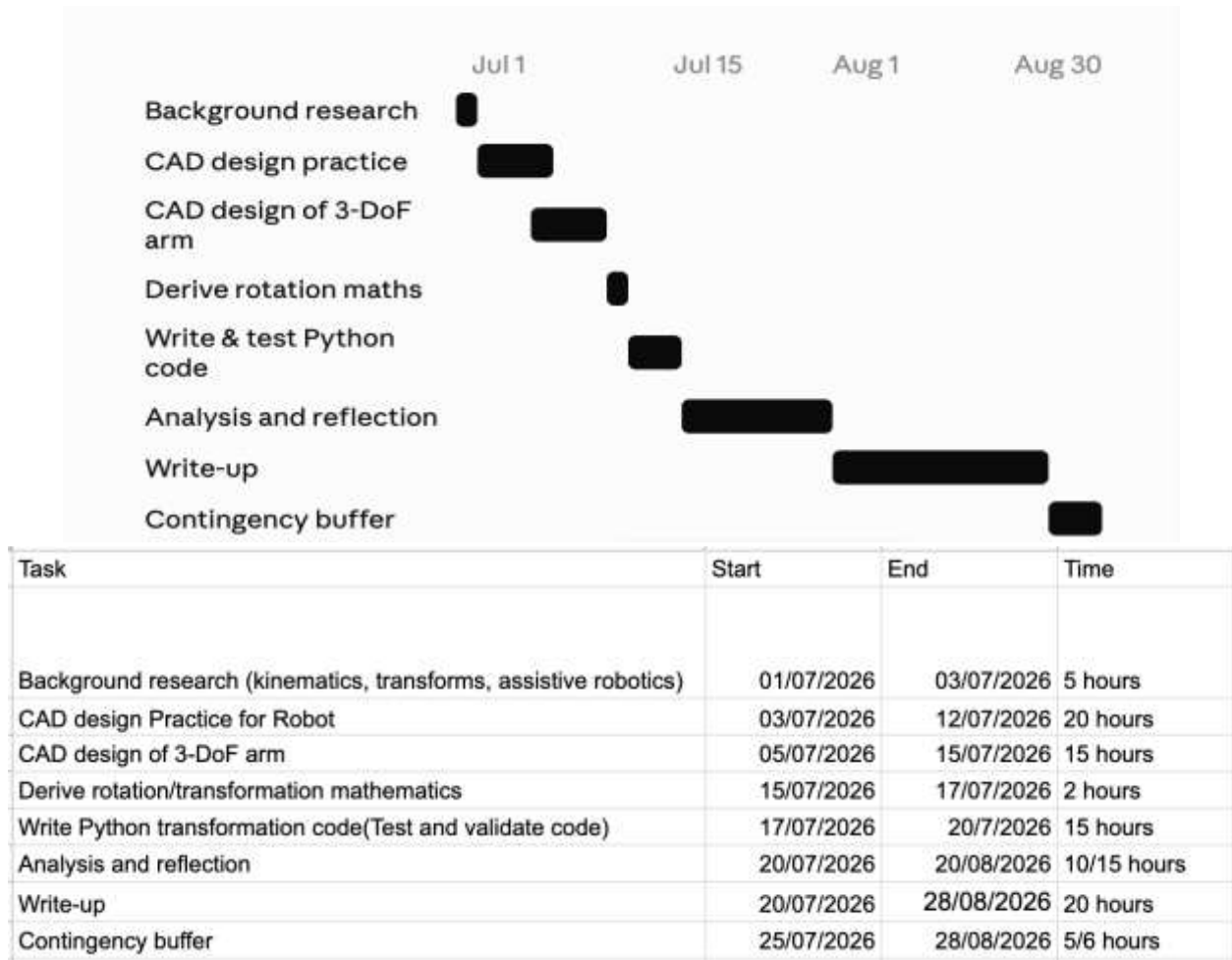
Figure 3: A further extended pose which shows the rotation of the robot, showing the connecting rod and gripper mechanism in more detail.



Software approach. I could have used an existing robotics library (such as [modern_robotics](#) or [spatialmath](#)) to handle coordinate transforms. I chose instead to write the transformation logic from scratch, since this ensured I fully understood the underlying mathematics rather than treating it as a black box, which was central to the purpose of the project.

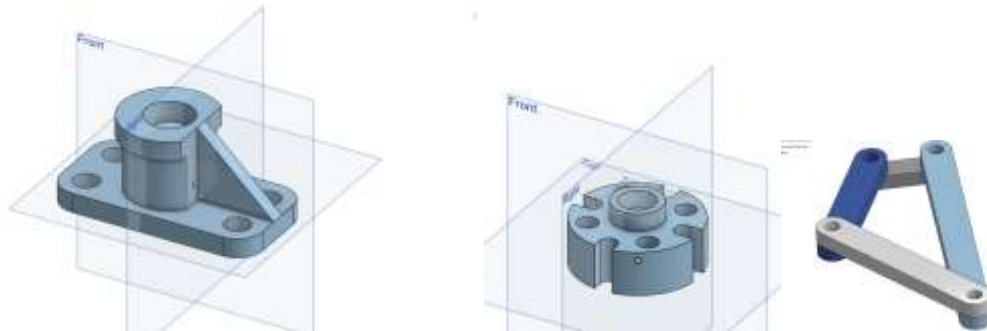
My approach was to work through the project in three connected stages: first, researching and deriving the mathematical foundation of coordinate transforms; second, designing the arm in CAD; and third, implementing and validating the computational model in code. I developed the CAD design and the transformation code largely in parallel, since the CAD model informed realistic link lengths and joint constraints that could later be used as inputs to the transform calculations, while the underlying mathematics did not depend on the CAD geometry being finalised first. This allowed me to make progress on both fronts without one blocking the other, and to bring them together once both were sufficiently developed.

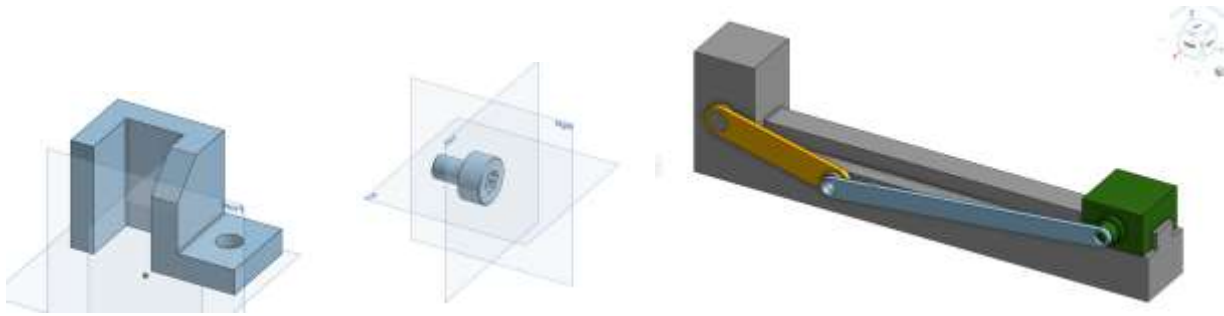
GRANT CHART



For this project, I used Onshape for CAD design, allowing me to model the arm's links, joints, and overall structure before manufacturing. The structural components were 3D printed, and small hobby servo motors were used for actuation at each joint. All coordinate transformation code was written in Python, using the NumPy library for matrix operations.

I also consulted a teacher/mentor throughout the project, who provided guidance on the CAD and mechanical design side, for example practice problems shown below. This input helped ensure the arm's mechanical design was practical to manufacture and assemble, while the mathematical and coding side of the project was worked through independently.





A rotation matrix describes the orientation of one coordinate frame relative to another, and rotations about the X, Y, and Z axes can each be represented as a 3x3 matrix before being combined for successive rotations. A homogeneous transformation matrix extends this by combining rotation and translation into a single 4x4 matrix, allowing both operations to be applied to a coordinate vector via a single matrix multiplication, a standard approach in robotics for representing the position and orientation of one frame relative to another (Sodemann, 2017).

Converting a coordinate from the global frame to the arm's local frame requires inverting this transform: since the inverse of a rotation matrix is equal to its transpose, and translations must be re-expressed in the rotated frame, the global-to-local transform can be computed as the inverse rotation matrix and a correspondingly transformed translation vector.

I learned this material primarily through online video tutorials, and revisited the RoboGrok "Robotics 1" course (a free university-level introductory robotics course covering kinematics) to consolidate my understanding before implementing the mathematics in code.

Reference:

1. Sodemann, A. (2017) Robotics 1: Homogeneous Transformation Matrix from Rotation Matrix. RoboGrok. Available at: https://www.robogrok.com/Robotics_1.php

I tested the code using a range of input cases, including translations, single-axis rotations, and successive multi-axis rotations, checking the outputs against expected geometric behaviour. One documented test case used a translation of $dx=3$, $dy=2$, $dz=1$ about a pivot point at $(0, 1, 0)$, with two successive rotations: 90 degrees about the Y axis followed by 30 degrees about the X axis. This produced the following combined rotation matrix:

```

... Enter dx: 3
Enter dy: 2
Enter dz: 1
Enter pivot x: 0
Enter pivot y: 1
Enter pivot z: 0
Enter the number of successive rotations: 2

Enter the rotations in the order they are to be performed:

Rotation 1 axis (X/Y/Z): y
Rotation 1 angle (degrees): 90
Rotation 2 axis (X/Y/Z): x
Rotation 2 angle (degrees): 30

Combined Rotation Matrix:
[[ 0.      0.      1.      ]
 [ 0.5     0.8660254  0.      ]
 [-0.8660254 0.5     0.      ]]

Homogeneous Transformation Matrix:
[[ 0.      0.      1.      3.      ]
 [ 0.5     0.8660254  0.      2.1339746]
 [-0.8660254 0.5     0.      0.5      ]
 [ 0.      0.      0.      1.      ]]

```

And the coding is

```

import numpy as np

# Rotation about X-axis
def rotation_x(theta):
    theta = np.radians(theta)
    return np.array([
        [1, 0, 0],
        [0, np.cos(theta), -np.sin(theta)],
        [0, np.sin(theta), np.cos(theta)]
    ])

# Rotation about Y-axis
def rotation_y(theta):
    theta = np.radians(theta)
    return np.array([
        [np.cos(theta), 0, np.sin(theta)],
        [0, 1, 0],
        [-np.sin(theta), 0, np.cos(theta)]
    ])

# Rotation about Z-axis
def rotation_z(theta):
    theta = np.radians(theta)
    return np.array([
        [np.cos(theta), -np.sin(theta), 0],
        [np.sin(theta), np.cos(theta), 0],
        [0, 0, 1]
    ])

```

```
# ----- Translation -----  
  
dx = float(input("Enter dx: "))  
dy = float(input("Enter dy: "))  
dz = float(input("Enter dz: "))  
  
d = np.array([dx, dy, dz])  
  
# ----- Pivot Point -----  
  
rx = float(input("Enter pivot x: "))  
ry = float(input("Enter pivot y: "))  
rz = float(input("Enter pivot z: "))  
  
r = np.array([rx, ry, rz])  
  
# ----- Successive Rotations -----  
  
n = int(input("Enter the number of successive rotations: "))  
  
R = np.eye(3)  
  
print("\nEnter the rotations in the order they are to be performed:\n")  
  
for i in range(n):  
    axis = input(f"Rotation {i+1} axis (X/Y/Z): ").strip().upper()  
    angle = float(input(f"Rotation {i+1} angle (degrees): "))  
  
    if axis == "X":  
        R = rotation_x(angle) @ R  
    elif axis == "Y":  
        R = rotation_y(angle) @ R  
    elif axis == "Z":  
        R = rotation_z(angle) @ R  
    else:  
        print("Invalid axis! Please enter X, Y, or Z.")  
        exit()
```

```
# ----- Homogeneous Transformation -----  
  
I = np.eye(3)  
  
# Translation considering pivot point  
t = d + (I - R) @ r  
  
T = np.eye(4)  
T[:3, :3] = R  
T[:3, 3] = t  
  
# Remove floating-point errors  
R[np.abs(R) < 1e-10] = 0  
T[np.abs(T) < 1e-10] = 0  
  
# ----- Output -----  
  
print("\nCombined Rotation Matrix:")  
print(R)  
  
print("\nHomogeneous Transformation Matrix:")  
print(T)
```

The correctness of calculated values was confirmed using a geometric check. Since, in any rotation, the only thing that gets changed is orientation relative to a pivot point, it means that any point situated at that pivot point remains the same. Thus, if a transformation matrix is multiplied by a vector representing a pivot point, the result should be the same as with multiplying just the translation component on it. The test verified that the point (3, 3, 1) which results from transforming the pivot point (0, 1, 0) using the transformation matrix is the same as translating it by the translation vector.

That test proves the aim of the project, stated in the introduction section. Realistically, robot's joints rotate about their own positions, not about the origin; hence, adding support for pivot points makes the model much more realistic for further implementation as a reach-and-retrieve robotic arm. Indeed, in such a system, the rotation at each joint has to be calculated with respect to the position of the joint on the arm rather than the global one. Despite being a computational project that has nothing in common with the actual arm yet, the basic mathematical principle for creating an assisting arm becomes quite clear. Namely, it is transforming target coordinates in order to make them usable by each joint. The logical continuation of the project would include connecting that transformation code with actual arm's joint positions and link lengths and ultimately coupling it to some sensor system which will automatically get real-life coordinates of targets.

One important decision made during the development was an addition of the ability to rotate the coordinate system about an arbitrary pivot point rather than about the origin only. Initially, rotations in the code could only be performed about the origin point, which is not really an accurate simulation of a physical robotic arm behavior. In a multi-link robotic arm, each joint is supposed to rotate about its own location rather than one global one. With that modification implemented in code, the transformation can be calculated in relation to any joint position, which is necessary for simulating multiple-joints behavior, not just one rotating point.

That extension added additional complexity to code, since in order to perform a rotation about an arbitrary point, translation must compensate not only rotation but also the offset between the origin and the pivot point, not just the offset caused by rotation. Without that modification, the model would have been restricted by describing just one rotation about a certain global point, instead of being capable to describe a 3-DoF arm where each joint has its own position on the arm.

Work with this project considerably increased my understanding of linear algebra applied in robotics, namely how to deal with rotation matrices, translation vectors, and pivot points in order to simulate motion in 3D space. I improved my skills in Python and NumPy programming, mostly working with inputs and validating numerical results.

The main improvement of the project, which I see now, is to integrate actual link lengths and joint positions of the arm designed in CAD directly into the code rather than leaving them separate. Currently, the transformation code works with arbitrary coordinates supplied by user without taking into account the design of the arm in Onshape. Connecting the two will allow calculating realistic joint-to-joint transformations of the actual arm, thereby bringing this project closer to becoming an actual kinematics solver.

The major challenge for me, when working with this project, was my lack of CAD skills. It took me many practice exercises in order to be able to implement this concept in a real-world example which took lots of time and effort. After I got used to Onshape though, the only other big challenge that I've faced was the management of complex measurements and calculations needed for making my robotic arm both complicated and functioning in the real world.

Three-DOF manipulator has three degrees of freedom and thus can be described by three rotation angles. Hence, rotations of its joints can be expressed in terms of 3x3 rotation matrices and multiplied in a certain order since rotation matrices are non-commutative. The current code multiplies those matrices in the exact order in which rotations are performed in order to construct the overall rotation matrix.

One major extension included in this work is the ability to rotate not only about the origin but about an arbitrary pivot point, which is very necessary for a realistic simulation of a robotic joint. From a mathematical perspective, the translation in this case can be expressed as $t = d + (I - R)r$ where d – the translation, R is the overall rotation matrix, I is the identity matrix and r is the position of the pivot point. This way, a point which is situated exactly at the pivot point remains unchanged when rotating since only the translation d is being performed on that point, which is correct for a real robotic joint rotating about its own position.

By combining R and t into one 4x4 homogeneous transformation matrix, it is possible to perform rotation and translation of coordinates using one matrix multiplication, which is commonly used in robotics kinematics. When applied to my robotic arm, every joint, namely base rotation, shoulder and elbow joints use that transformation to calculate required joint rotations which place the gripper in the target point. This way, a link between the coordinates description like "the cup is 30 cm to the left and 15 cm forward" and the real life arm movement described in Figure 1-3 is provided.

Since all the coordinate transformation work was just computational, there were no data ethics issues from this part of project because no personal or third party data was collected or manipulated in the course of computations. For the assembly part, a risk assessment was made prior to starting any work with 3D printer and assembling the pieces because of the risks of burning by the hot end of the 3D printer, injury when handling small pieces and assembling the whole structure, and electrical dangers due to the wiring of the servo motors and checking of wires' polarity before activating the servos.

This project combines two separate fields of knowledge: CAD/mechanical design and computational linear algebra. Instead of using already existing robotics libraries in order to handle coordinate transforms, the rotation and translation were implemented separately using general matrix algebra principles adapted to the particular task of robotic frames' conversion. One interesting aspect of that is a way how the rotation about an arbitrary point feature has been implemented. By seeing the problem in terms of a simple input change, a need to change the underlying mathematical model arose. I derived a modified translation equation $t = d + (I - R)r$ in order to simulate a rotation about local origin and not the world origin. This meant the need to apply an analytical approach and use principles of matrix algebra and physical properties of a robotic joint simultaneously.

When assembling the arm, the dimensional tolerance issue came up with one of the printed parts: it was either too tight or too loose relative to the mating part. After some investigation, the source of that problem was found in tolerances set in the original CAD design which did not take into account precision limits of the printing process. Depending on print settings and material behavior, the manufactured parts can differ from their nominal dimensions.

For that purpose, I returned to Onshape in order to revise the problematic part's design and adjust it to be dimensionally correct in order to achieve proper joint clearance before reprinting. That means rethinking of the joint's fit in terms of both its mechanical properties (it needs to rotate freely) and 3D printing specifics.