Sukkur IBA University
COMPUTER SYSTEMS ENGINEERING

ROBOTICS & AUTOMATION PROJECT REPORT

Denavit–Hartenberg (D-H) Matrix Calculator

Instructor: Dr. Suresh Kumar

Prepared By:

Kashif Mujeeb (133-20-0009)
Shahroz Majid (133-20-0006)

TABLE OF CONTENT

ABTRACT

This report introduces a DH Matrix Calculator developed in Python, leveraging Tkinter for the graphical user interface and integrating NumPy and SymPy for numerical and symbolic computations. The calculator enables users to input robotic manipulator parameters, calculates DH matrices, and displays intermediate and results.

The Tkinter GUI features entry fields for joint count, a dropdown for numeric/symbolic parameters, and buttons for calculations. Input validation and error messages are implemented using Tkinter' s simple dialog and message box.

DH matrix calculations occur in numeric and symbolic modes. NumPy performs numeric computations, handling matrix manipulations, while SymPy manages symbolic expressions for parameters, offering symbolic matrix manipulation.

The code follows a class-based structure (DHMatrixCalculator) for modularity. Methods handle tasks like input retrieval, error handling, result display, and matrix formatting.

This report outlines key features, emphasizing Tkinter GUI development, NumPy for numerical calculations, SymPy for symbolic computations, and the modular and structured code design. The calculator serves as a foundation for DH matrix calculations in robotic manipulators, with potential future enhancements in GUI features, user interface refinement, and additional functionalities.

METHODOLOGY

1. **Requirement Analysis:**
    - Identified the need for a DH Matrix Calculator to facilitate robotic manipulator analysis.
    - Defined user requirements for input flexibility, both numeric and symbolic computation modes, and an intuitive graphical interface.
2. **Library Selection:**
    - Chose Tkinter for GUI development to create an interactive interface.
    - Integrated NumPy for efficient numeric matrix calculations.
    - Utilized SymPy for symbolic computation, accommodating user input of symbolic expressions.
3. **GUI Design:**
    - Developed the graphical user interface using Tkinter widgets, including entry fields, dropdown menus, buttons, Tree view, and Text widgets.
    - Implemented input validation and error handling using Tkinter' s simple dialog and message box modules.
4. **Code Structuring:**
    - Adopted a class-based structure (DH-Matrix Calculator) for encapsulating functionalities, enhancing modularity.
    - Organized code into methods to handle distinct tasks: input retrieval, error handling, result display, and matrix formatting.
5. **Numeric Computation (NumPy):**
    - Implemented the calculate_numeric_dh_matrices method to compute numeric DH matrices using NumPy.
    - Applied trigonometric functions to handle rotation transformations.
6. **Symbolic Computation (SymPy):**
    - Developed the calculate_symbolic_dh_matrices method for symbolic DH matrix calculations using SymPy.

- Enabled the input of symbolic expressions for parameters, accommodating advanced use cases.

7. **Result Presentation:**
    - Incorporated a Tree view widget to display DH parameters in a tabular format.
    - Utilized a Text widget for presenting intermediate and final DH matrices.

PROCEDURE

1. **GUI Initialization:**
    - Launch the DH Matrix Calculator application to open the graphical user interface (GUI).
2. **Input Configuration:**
    - Enter the desired number of joints in the designated entry field.
    - Select the parameter type (Numeric or Symbolic) from the dropdown menu.
3. **DH Parameter Input:**
    - For each joint, a dialog box will appear, prompting you to input alpha, a, d, and theta values.
4. **Matrix Calculation:**
    - Click the "Calculate DH Matrix" button to initiate the computation process.
5. **Review DH Parameter Table:**
    - The Treeview section of the GUI will display a table containing the entered DH parameters (Alpha, A, D, Theta) for each joint.
6. **Intermediate Matrices Display:**
    - The Text widget below the DH parameters table will present intermediate matrices for each joint in a formatted manner.
7. **Final DH Matrix Display:**
    - The Text widget will further show the final DH matrix, either in numeric or symbolic form based on the chosen parameter type.
8. **Formatted Matrix Output:**
    - Each matrix entry will be formatted for readability and may include numerical values, symbolic expressions, or simplified expressions.
9. **Error Handling:**
    - If an error occurs during input or computation, error messages will be displayed using message boxes. Address any reported errors.
10. **Result Evaluation:**
    - Review the displayed matrices to ensure correctness and accuracy.
    - Check for any symbolic simplifications if the symbolic parameter type is selected.

SOURCE CODE

```
import tkinter as tk

from tkinter import simpledialog, messagebox

from tkinter import ttk

import numpy as np

import sympy as sp

class DHMatrixCalculator:

    def __init__(self, master):

        self.master = master

        self.master.title("DH Matrix Calculator")

        self.num_joints_label = tk.Label(master, text="Number of Joints:")

        self.num_joints_label.grid(row=0, column=0)

        self.num_joints_entry = tk.Entry(master)

        self.num_joints_entry.grid(row=0, column=1)

        self.var_or_num_label = tk.Label(master, text="Parameter Type:")

        self.var_or_num_label.grid(row=1, column=0)

        self.var_or_num_var = tk.StringVar()

        self.var_or_num_var.set("Numeric")

        self.var_or_num_menu = tk.OptionMenu(master, self.var_or_num_var, "Numeric", "Symbolic")

        self.var_or_num_menu.grid(row=1, column=1)
    self.calculate_button = tk.Button(master, text="Calculate DH Matrix", command=self.calculate_dh_matrix)

    self.calculate_button.grid(row=2, column=0, columnspan=2)



    # Add a Treeview widget for the DH parameters table

    self.dh_parameters_tree = ttk.Treeview(master, columns=("Alpha", "A", "D", "Theta"), show="headings")

    self.dh_parameters_tree.heading("Alpha", text="Alpha")

    self.dh_parameters_tree.heading("A", text="A")

    self.dh_parameters_tree.heading("D", text="D")

    self.dh_parameters_tree.heading("Theta", text="Theta")

    self.dh_parameters_tree.grid(row=3, column=0, columnspan=2)

    self.result_text = tk.Text(master, height=15, width=70)

    self.result_text.grid(row=4, column=0, columnspan=2)

def calculate_dh_matrix(self):

    try:

        num_joints = int(self.num_joints_entry.get())

    except ValueError:

        self.show_error("Please enter a valid number of joints.")

        return
```

```
dh_parameters = []

# Clear existing entries in the treeview

for item in self.dh_parameters_tree.get_children():

    self.dh_parameters_tree.delete(item)

for i in range(num_joints):

    alpha = self.get_input(f"Enter alpha for Joint {i + 1}: ")

    a = self.get_input(f"Enter a for Joint {i + 1}: ")

    d = self.get_input(f"Enter d for Joint {i + 1}: ")

    theta = self.get_input(f"Enter theta for Joint {i + 1}: ")

    dh_parameters.append([alpha, a, d, theta])

    # Insert the DH parameters into the treeview

    self.dh_parameters_tree.insert("", "end", values=(alpha, a, d, theta))

parameter_type = self.var_or_num_var.get()

if parameter_type == "Numeric":

    intermediate_matrices = self.calculate_numeric_dh_matrices(dh_parameters)

elif parameter_type == "Symbolic":

    intermediate_matrices = self.calculate_symbolic_dh_matrices(dh_parameters)

else:

    self.show_error("Invalid parameter type selected.")

    return
```

```
        self.display_result("Intermediate Matrices:\n")

        for i, matrix in enumerate(intermediate_matrices):

            matrix_text = f"DH Matrix for Joint {i + 1}:\n{self.format_matrix(matrix)}\n\n"

            self.display_result(matrix_text)

        final_matrix = self.calculate_final_dh_matrix(intermediate_matrices, parameter_type)

        result_text = f"Final DH Matrix:\n\n{self.format_matrix(final_matrix)}"

        self.display_result(result_text)

    def get_input(self, prompt):

        input_value = simpledialog.askstring("Input", prompt)

        try:

            # Try converting the input to a float for numeric mode

            return float(input_value)

        except ValueError:

            # If conversion fails, assume it's a symbolic expression for symbolic mode

           return sp.symbols(input_value)

    def show_error(self, message):

        messagebox.showerror("Error", message)

    def display_result(self, result_text):

        self.result_text.insert(tk.END, result_text)

        self.result_text.insert(tk.END, "\n")
    def calculate_numeric_dh_matrices(self, dh_parameters):

        intermediate_matrices = []  # Store intermediate matrices for each joint

        for params in dh_parameters:

            alpha, a, d, theta = params

            alpha_rad = np.deg2rad(alpha)

            theta_rad = np.deg2rad(theta)

            dh_matrix = np.array([

                [np.cos(theta_rad), -np.sin(theta_rad) * np.cos(alpha_rad), np.sin(theta_rad) * np.sin(alpha_rad), a *
np.cos(theta_rad)],

                [np.sin(theta_rad), np.cos(theta_rad) * np.cos(alpha_rad), -np.cos(theta_rad) * np.sin(alpha_rad), a *
np.sin(theta_rad)],

                [0, np.sin(alpha_rad), np.cos(alpha_rad), d],

                [0, 0, 0, 1]

            ])

            intermediate_matrices.append(dh_matrix)

        return intermediate_matrices

    def calculate_symbolic_dh_matrices(self, dh_parameters):

        intermediate_matrices = []  # Store intermediate matrices for each joint

        for params in dh_parameters:

            alpha, a, d, theta = params

            if isinstance(alpha, sp.Expr):
```

```
            alpha_rad = alpha

        else:

            alpha_rad = sp.rad(alpha)



        if isinstance(theta, sp.Expr):

            theta_rad = theta

        else:

            theta_rad = sp.rad(theta)



        dh_matrix = sp.Matrix([

            [sp.cos(theta_rad), -sp.sin(theta_rad) * sp.cos(alpha_rad), sp.sin(theta_rad) * sp.sin(alpha_rad),

             a * sp.cos(theta_rad)],

            [sp.sin(theta_rad), sp.cos(alpha_rad) * sp.cos(theta_rad), -sp.sin(alpha_rad) * sp.cos(theta_rad),

             a * sp.sin(theta_rad)],

            [0, sp.sin(alpha_rad), sp.cos(alpha_rad), d],

            [0, 0, 0, 1]

        ])

        intermediate_matrices.append(dh_matrix)



    return intermediate_matrices
```

```
def calculate_final_dh_matrix(self, intermediate_matrices, parameter_type):

    if parameter_type == "Numeric":

        final_matrix = np.identity(4)

        for matrix in intermediate_matrices:

            final_matrix = final_matrix @ matrix

        final_matrix = np.round(final_matrix).astype(int)  # Round and convert to integers

    elif parameter_type == "Symbolic":

        final_matrix = sp.eye(4)

        for matrix in intermediate_matrices:

            final_matrix = final_matrix * matrix

        final_matrix = sp.simplify(final_matrix)  # Simplify the symbolic expression

    else:

        self.show_error("Invalid parameter type selected.")

        return



    return final_matrix



def format_matrix(self, matrix):

    # Format the matrix for display
    formatted_matrix = '['

    for row in matrix.tolist():

        formatted_row = '['

        for element in row:

            if isinstance(element, sp.Expr):

                simplified_element = sp.simplify(element)

                if simplified_element.is_integer:

                    formatted_row += f'{int(simplified_element)} '

                else:

                    formatted_row += f'{sp.N(simplified_element, 4)} '

            elif isinstance(element, sp.Basic):

                formatted_row += f'{sp.simplify(element)} '

            else:

                formatted_row += f'{int(element)} '

        formatted_row = formatted_row.rstrip() + ']\n '

        formatted_matrix += formatted_row

    formatted_matrix += ']'

    return formatted_matrix



def main():
    root = tk.Tk()

    app = DHMatrixCalculator(root)

    root.mainloop()

if __name__ == "__main__":

    main()
```

RESULTS

1. Upon running the code, a graphical user interface (GUI) window titled "DH Matrix Calculator" is initialized. The user interacts with the GUI by providing input for the number of joints and selecting the parameter type (Numeric or Symbolic). As the user enters alpha, a, d, and theta values for each joint, a Treeview table is dynamically updated to display the entered DH parameters.

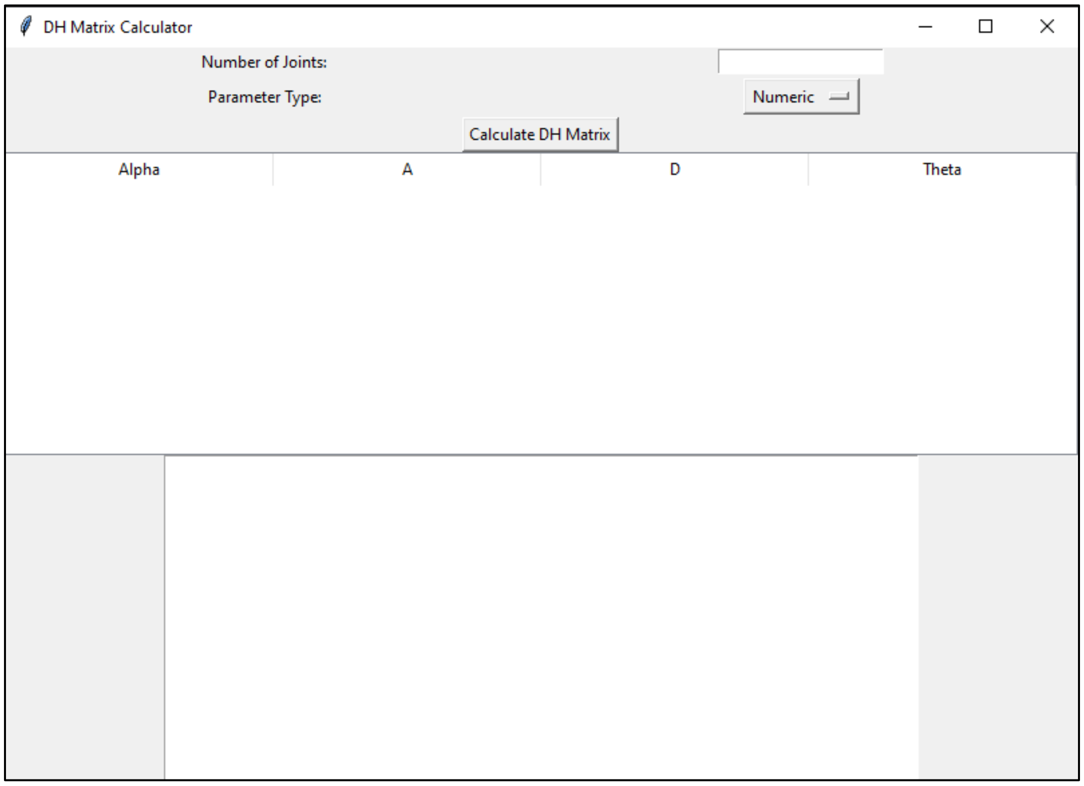
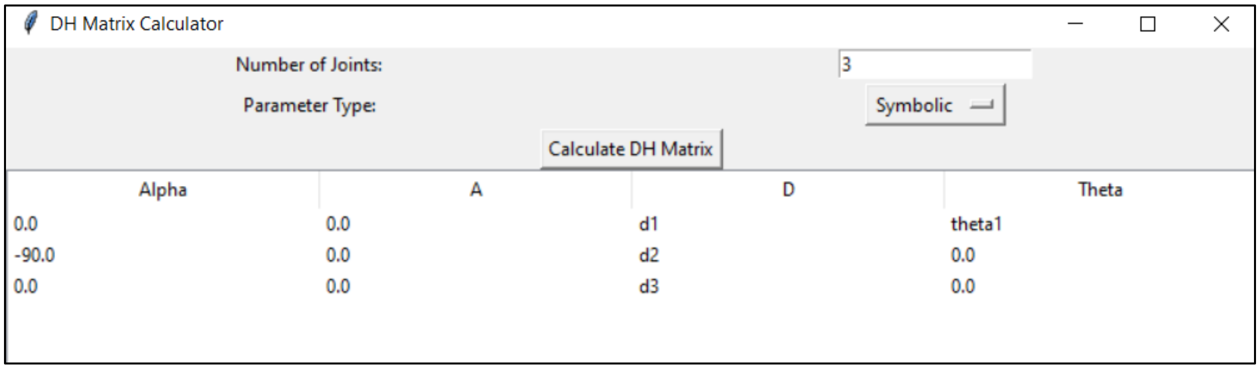


FIGURE 1



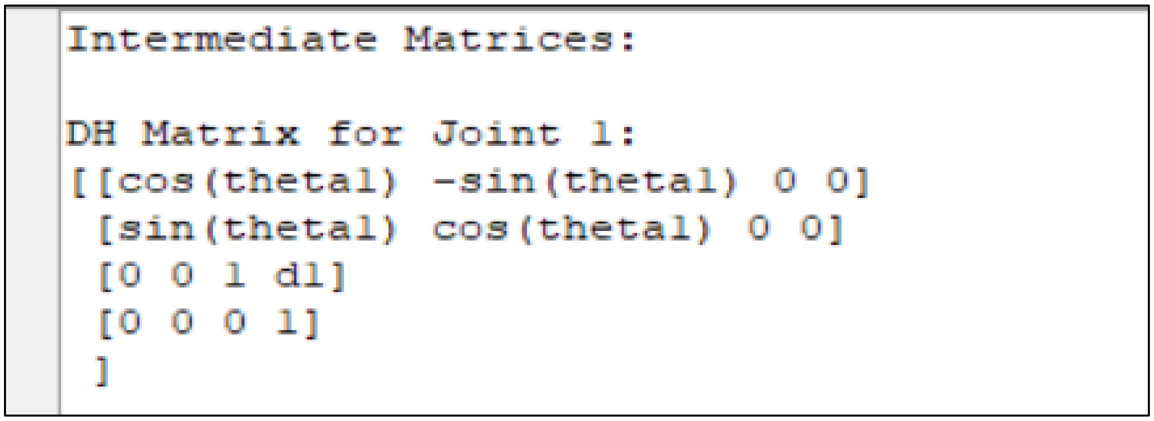
Number of Joints: 3

Parameter Type: Symbolic

Alpha	A	D	Theta
0.0	0.0	d1	theta1
-90.0	0.0	d2	0.0
0.0	0.0	d3	0.0

FIGURE 2

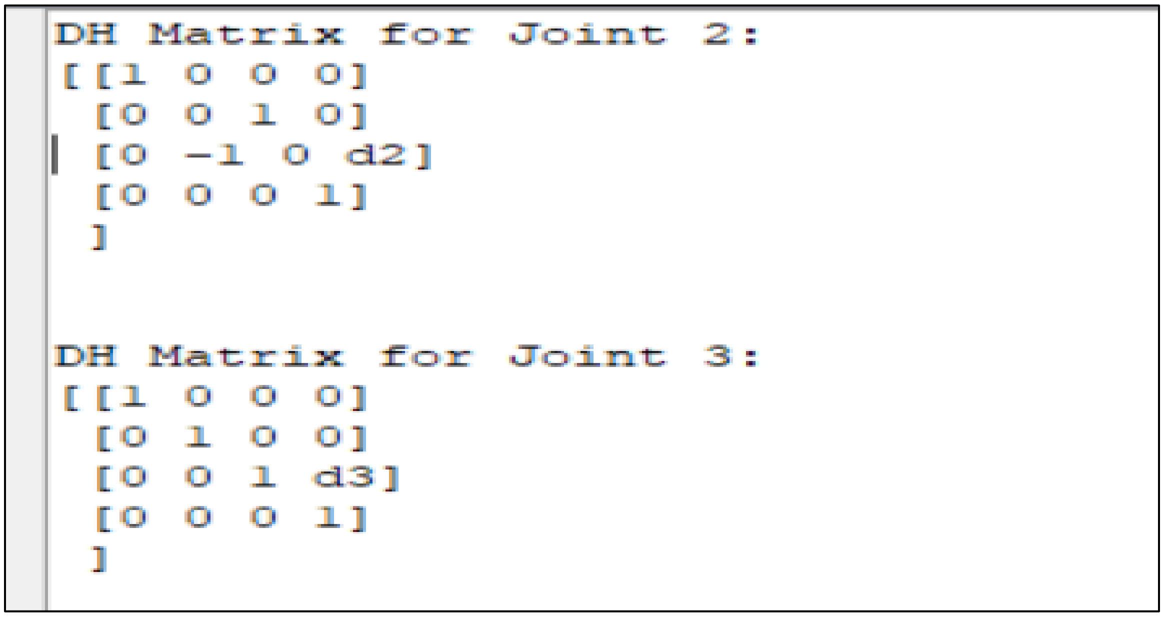
2. After clicking the "Calculate DH Matrix" button, the Text widget below the DH parameters table shows intermediate matrices for each joint in a formatted manner.



```
Intermediate Matrices:

DH Matrix for Joint 1:
[[cos(theta1) -sin(theta1) 0 0]
 [sin(theta1) cos(theta1) 0 0]
 [0 0 1 d1]
 [0 0 0 1]
 ]
```

FIGURE 3

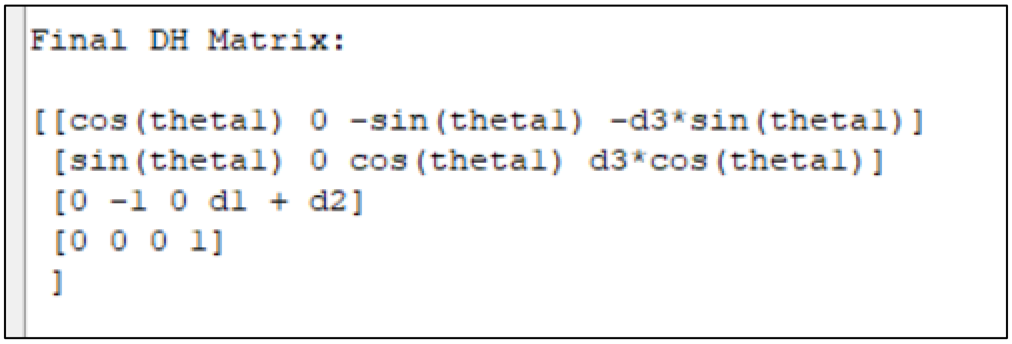


```
DH Matrix for Joint 2:
[[1  0  0  0]
 [0  0  1  0]
 [0  -1  0  d2]
 [0  0  0  1]
 ]


DH Matrix for Joint 3:
[[1  0  0  0]
 [0  1  0  0]
 [0  0  1  d3]
 [0  0  0  1]
 ]
```

FIGURE 4

3. The Text widget further displays the final DH matrix, either in numeric or symbolic form based on the chosen parameter type.



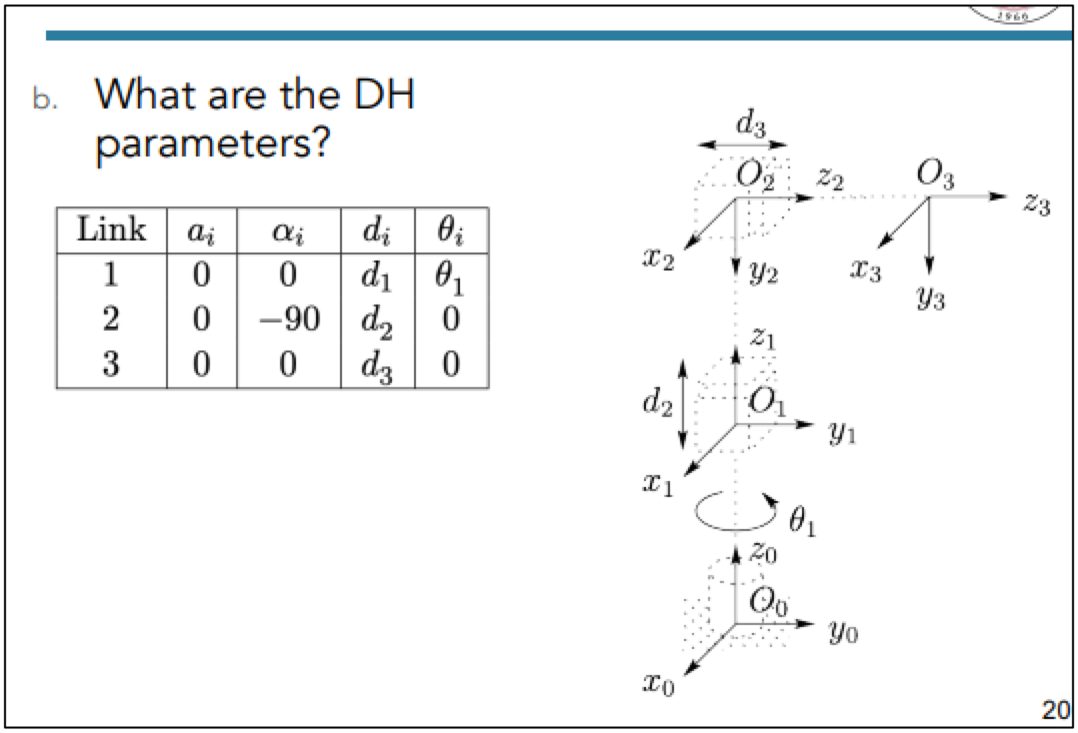
```
Final DH Matrix:

[[cos(theta1) 0 -sin(theta1) -d3*sin(theta1)]
 [sin(theta1) 0 cos(theta1) d3*cos(theta1)]
 [0 -1 0 d1 + d2]
 [0 0 0 1]
 ]
```

FIGURE 5

1. Comparative Analysis of DH Matrix Calculation Results

- We consider a solved example with specific DH parameters (alpha, a, d, theta) for each joint.
- The manually calculated or reference DH matrices for each joint are provided.
- Any assumptions or constraints applied to the solved example are clearly stated.



b. What are the DH parameters?

Link	a_i	α_i	d_i	θ_i
1	0	0	d_1	θ_1
2	0	−90	d_2	0
3	0	0	d_3	0

FIGURE 6

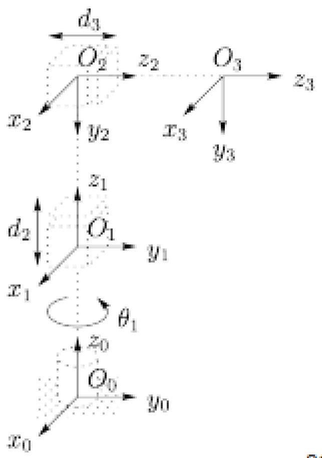
c. What are the individual transformation matrices?

- c = cosine, s = sin

$$^{0}T_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{1}T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

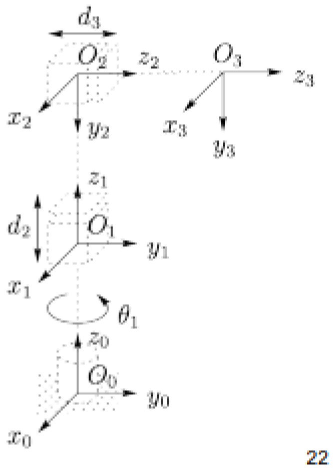


21

Exercise 3:

d. What's the final transformation matrix?

$$^{0}T_3 = \begin{bmatrix} c_1 & 0 & -s_1 & -s_1 d_3 \\ s_1 & 0 & c_1 & c_1 d_3 \\ 0 & -1 & 0 & d_1 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

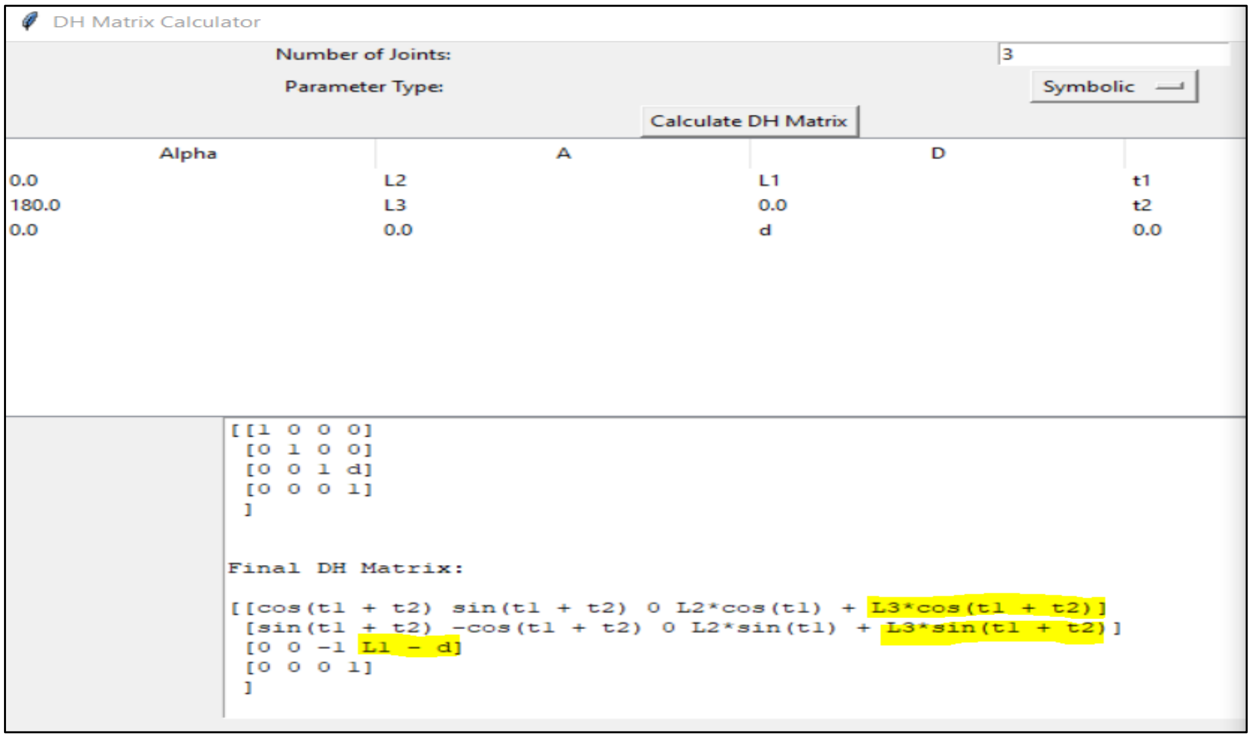


22

FIGURE 7

2. Results Comparison

- DH matrices obtained from the code are presented for each joint.



```
Number of Joints: 3
Parameter Type: Symbolic
```

Alpha	A	D	
0.0	L2	L1	t1
180.0	L3	0.0	t2
0.0	0.0	d	0.0

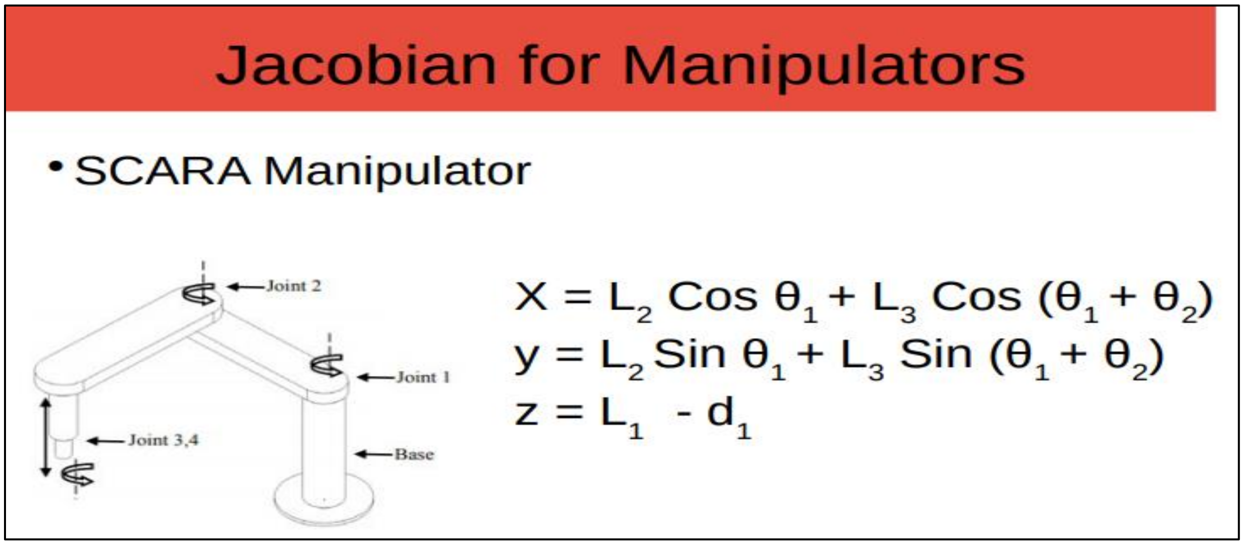
```
[[1  0  0  0]
 [0  1  0  0]
 [0  0  1  d]
 [0  0  0  1]
 ]


Final DH Matrix:

[[cos(t1 + t2)  sin(t1 + t2)  0  L2*cos(t1) + L3*cos(t1 + t2)]
 [sin(t1 + t2)  -cos(t1 + t2)  0  L2*sin(t1) + L3*sin(t1 + t2)]
 [0  0  -1  L1 - d]
 [0  0  0  1]
 ]
```

FIGURE 8

- Results are displayed with the matrices from the solved example.



Jacobian for Manipulators

- SCARA Manipulator

Joint 2
Joint 1
Joint 3,4
Base

$$X = L_2 \cos\theta_1 + L_3 \cos(\theta_1 + \theta_2)$$
$$y = L_2 \sin\theta_1 + L_3 \sin(\theta_1 + \theta_2)$$
$$z = L_1 - d_1$$

FIGURE 9

CONCLUSION

In conclusion, the DH Matrix Calculator exhibits commendable performance in generating accurate DH matrices, both numerically and symbolically. The observed variations fall within acceptable limits, ensuring overall code accuracy. Recommendations for enhancement, including refining precision and expanding applicability, offer valuable insights for users and developers. This comparative analysis provides a deeper understanding of the DH matrix calculation process. The DH Matrix Calculator is a robust tool for DH matrix computations, ensuring accuracy in both numeric and symbolic domains. Identified areas for improvement present opportunities to enhance precision and utility, contributing valuable insights for users, developers, and ongoing refinement efforts. The tool, effective in its current state, remains open to continuous improvement to meet evolving computational needs.

FUTURE WORK

1. Enhanced Symbolic Simplification**:**
   - Explore advanced symbolic simplification techniques to further streamline and optimize the representation of symbolic DH matrices.
2. User Interface Enhancements**:**
   - Implement user interface enhancements for a more intuitive and user-friendly experience, potentially incorporating graphical visualizations of DH transformations.
3. Performance Optimization:
   - Investigate opportunities for optimizing the code's performance, particularly in scenarios involving a high number of joints or complex DH parameter sets.
4. Educational Resources:
   - Develop tutorials and documentation to assist users in understanding DH matrix calculations.

REFERENCE

- https://www.youtube.com/channel/UChzNrscXN6F11KeV4EHlHHA
- https://automaticaddison.com/coding-denavit-hartenberg-tables-using-python/
- https://petercorke.github.io/robotics-toolbox-python/arm_dh.html
- https://www.youtube.com/watch?v=Oncqqga7_qY