# Laboratory Manual

| | |
|---|---|
| **Name of the School:** | **School of Computing Sciences and Engineering** |
| **Name of the Department:** | **Computer Science and Engineering** |
| **Name of the Programme:** | **B.Tech AIML** |
| **Course/ Course Code:** | **Robotics using AI Lab 17YCM712** |
| **Academic Year:** | **2023-24** |

# Guidelines

**Laboratory rules**

1. Attendance is required for all lab classes. Students who do not attend lab will not receive credit.

2. Ensure that you are aware of the test and its procedure before each lab class. **<u>You will NOT be allowed to attend the class if you a renort unprepared!</u>**

3. Personal safety is top priority. Do not use equipment that is not assigned to you.

4. All accidents must be reported to your instructor or laboratory supervisor.

5. The surroundings near the equipment must be cleaned before leaving each lab class.

6. Ensure that readings are checked and marked by your TA for each lab period.

## Laboratory report

1. Each student has to submit the report for each experiment.
2. Your report is to be written only in the space provided in the manual
3. Please write your number and, batch and group number.
4. Your report must be neat, well organized, and make a professional impact. Label all axes and include proper units.
5. Your reports should be submitted within 7 days and before the beginning of the lab class of the next batch.
6. Your reports will be as per rubrics provided at the end of each experiment

Anyone caught plagiarizing work in the laboratory report, from a current or past student's notebook, will receive 0 on the grading scale.

| | |
|---|---|
| Name of Student | ........................................ |
| Academic Year | ........................................ |
| Programme | ........................................ |
| Class/ Roll No. | ........................................ |
| PRN No. | ........................................ |

# CERTIFICTAE

This is to certify that

Mr./Miss……………………………..…………………………………….

Roll no…………………... PRN No…………………………………

of class……… …………..has satisfactorily/unsatisfactorily

completed the Term Work of  Course ……………………………….

in this School during academic year ………………….

| | | |
|---|---|---|
| Course Teacher | Head of Department | Dean Academics |

# Table of Contents

## EXPERIMENT 1

**Aim : Solve any problem using depth first search.**

**Theory: Depth First Traversal (or DFS)** for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

The step by step process to implement the DFS traversal is given as follows -

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

**Code:**

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])}

dfs(graph, '0')
```

**Output:**

```
0
2
1
3
4
```

# EXPERIMENT 2

**Aim** : Solve any problem using best first search

**Theory**: In BFS and DFS, when we are at a node, we can consider any of the adjacent as the next node. So both BFS and DFS blindly explore paths without considering any cost function.

The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

Best first search algorithm:

- Step 1: Place the starting node into the OPEN list.
- Step 2: If the OPEN list is empty, Stop and return failure.
- Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- Step 4: Expand the node n, and generate the successors of node n.
- Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- Step 7: Return to Step 2.

**Code**:

```
from queue import PriorityQueue

v = 14
```

```
graph = [[] for i in range(v)]


# Function For Implementing Best First Search
# Gives output path having lowest cost



def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
        print()

# Function for adding edges to graph



def addedge(x, y, cost):
        graph[x].append((y, cost))
        graph[y].append((x, cost))



# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

**Output:**

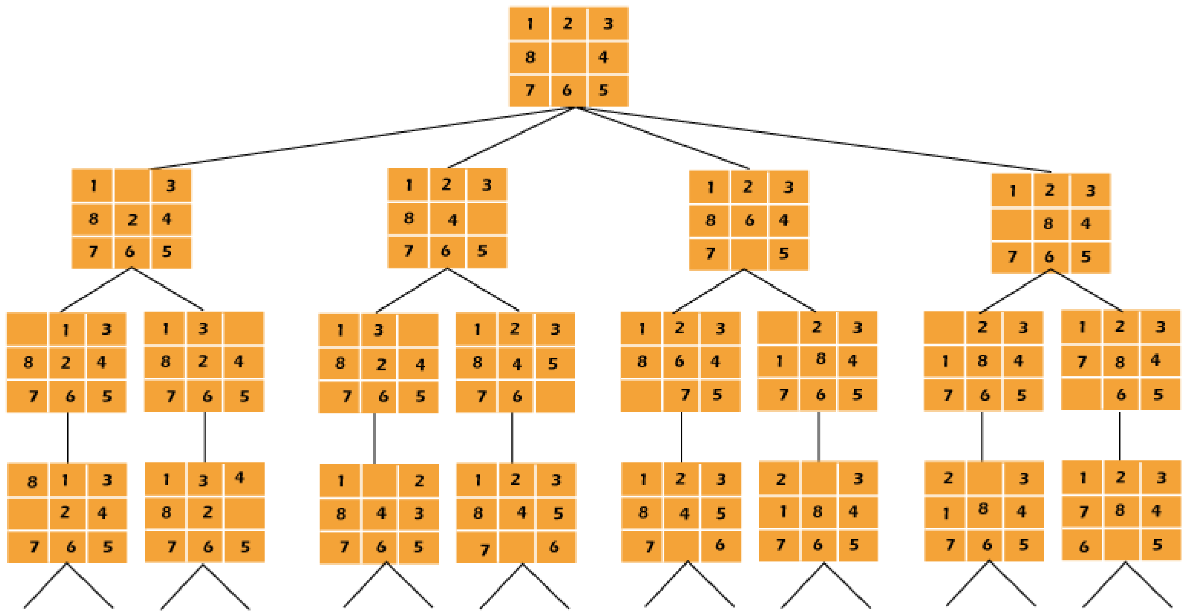
```
0 1 3 2 8 9
```

## EXPERIMENT 3

**Aim : Solve 8 puzzle problem using best first search**

**Theory:** The 8 puzzle problem solution is covered in this article. A **3 by 3 board with 8 tiles** (each tile has a number from 1 to 8) and a **single empty space** is provided. The goal is to **use the vacant space to arrange the numbers on the tiles** such that they match the final arrangement. **Four neighbouring** (left, right, above, and below) **tiles** can be moved into the available area.

For instance,



We can search the state space tree using a **breadth-first approach**. It always locates the **goal state that is closest to the root**. However, the algorithm tries the **same series of movements as DFS** regardless of the initial state.

**Code:**

```
# Python3 program to print the path from root
# node to destination node for N*N-1 puzzle
# algorithm using Branch and Bound
# The solution assumes that instance of
# puzzle is solvable

# Importing copy for deepcopy function
import copy

# Importing the heap functions from python
# library for Priority Queue
from heapq import heappush, heappop

# This variable can be changed to change
# the program from 8 puzzle(n=3) to 15
# puzzle(n=4) to 24 puzzle(n=5)...
n = 3

# bottom, left, top, right
row = [ 1, 0, -1, 0 ]
col = [ 0, -1, 0, 1 ]

# A class for Priority Queue
class priorityQueue:
    # Constructor to initialize a
    # Priority Queue
    def __init__(self):
        self.heap = []

    # Inserts a new key 'k'
    def push(self, k):
        heappush(self.heap, k)

    # Method to remove minimum element
    # from Priority Queue
    def pop(self):
        return heappop(self.heap)

    # Method to know if the Queue is empty
    def empty(self):
        if not self.heap:
            return True
        else:
            return False

# Node structure
```

```
class node:

    def __init__(self, parent, mat, empty_tile_pos,
                          cost, level):

        # Stores the parent node of the
        # current node helps in tracing
        # path when the answer is found
        self.parent = parent

        # Stores the matrix
        self.mat = mat

        # Stores the position at which the
        # empty space tile exists in the matrix
        self.empty_tile_pos = empty_tile_pos

        # Stores the number of misplaced tiles
        self.cost = cost

        # Stores the number of moves so far
        self.level = level
    # This method is defined so that the
    # priority queue is formed based on
    # the cost variable of the objects
    def __lt__(self, nxt):
        return self.cost < nxt.cost

# Function to calculate the number of
# misplaced tiles ie. number of non-blank
# tiles not in their goal position
def calculateCost(mat, final) -> int:

    count = 0
    for i in range(n):
        for j in range(n):
            if ((mat[i][j]) and
                (mat[i][j] != final[i][j])):
                count += 1

    return count

def newNode(mat, empty_tile_pos, new_empty_tile_pos,
            level, parent, final) -> node:
```

```
    # Copy data from parent matrix to current matrix
    new_mat = copy.deepcopy(mat)

    # Move tile by 1 position
    x1 = empty_tile_pos[0]
    y1 = empty_tile_pos[1]
    x2 = new_empty_tile_pos[0]
    y2 = new_empty_tile_pos[1]
    new_mat[x1][y1], new_mat[x2][y2] = new_mat[x2][y2], new_mat[x1][y1]

    # Set number of misplaced tiles
    cost = calculateCost(new_mat, final)

    new_node = node(parent, new_mat, new_empty_tile_pos,
                    cost, level)
    return new_node

# Function to print the N x N matrix
def printMatrix(mat):

    for i in range(n):
        for j in range(n):
            print("%d " % (mat[i][j]), end = " ")
        print()

# Function to check if (x, y) is a valid
# matrix coordinate
def isSafe(x, y):

    return x >= 0 and x < n and y >= 0 and y < n

# Print path from root node to destination node
def printPath(root):

    if root == None:
        return

    printPath(root.parent)
    printMatrix(root.mat)
    print()

# Function to solve N*N - 1 puzzle algorithm
# using Branch and Bound. empty_tile_pos is
# the blank tile position in the initial state.
def solve(initial, empty_tile_pos, final):
```

```
# Create a priority queue to store live
# nodes of search tree
pq = priorityQueue()

# Create the root node
cost = calculateCost(initial, final)
root = node(None, initial,
                        empty_tile_pos, cost, 0)

# Add root to list of live nodes
pq.push(root)

# Finds a live node with least cost,
# add its children to list of live
# nodes and finally deletes it from
# the list.
while not pq.empty():

        # Find a live node with least estimated
        # cost and delete it from the list of
        # live nodes
        minimum = pq.pop()
```

```
# If minimum is the answer node
if minimum.cost == 0:

    # Print the path from root to
    # destination;
    printPath(minimum)
    return

# Generate all possible children
for i in range(4):
    new_tile_pos = [
        minimum.empty_tile_pos[0] + row[i],
        minimum.empty_tile_pos[1] + col[i], ]

    if isSafe(new_tile_pos[0], new_tile_pos[1]):

        # Create a child node
        child = newNode(minimum.mat,
                        minimum.empty_tile_pos,
                        new_tile_pos,
                        minimum.level + 1,
                        minimum, final,)
```

```
                                # Add child to list of live nodes
                                pq.push(child)

# Driver Code

# Initial configuration
# Value 0 is used for empty space
initial = [ [ 1, 2, 3 ],
                        [ 5, 6, 0 ],
                        [ 7, 8, 4 ] ]

# Solvable Final configuration
# Value 0 is used for empty space
final = [ [ 1, 2, 3 ],
                [ 5, 8, 6 ],
                [ 0, 7, 4 ] ]

# Blank tile coordinates in
# initial configuration
empty_tile_pos = [ 1, 2 ]

# Function call to solve the puzzle
```

```
solve(initial, empty_tile_pos, final)
```

**Output:**

```
1  2  3
5  6  0
7  8  4

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

1  2  3
5  8  6
0  7  4
```

## EXPERIMENT 4

**Aim : Solve any problem using breadth first search**

**Theory:** *The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.*

The steps involved in the BFS algorithm to explore a graph are given as follows -

**Step 1:** SET STATUS = 1 (ready state) for each node in G

**Step 2:** Enqueue the starting node A and set its STATUS = 2 (waiting state)

**Step 3:** Repeat Steps 4 and 5 until QUEUE is empty

**Step 4:** Dequeue a node N. Process it and set its STATUS = 3 (processed state).

**Step 5:** Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2

(waiting state)

[END OF LOOP]

**Step 6:** EXIT

**Code:**

```
import collections


# BFS algorithm

def bfs(graph, root):
    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        # Dequeue a vertex from queue
        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        # If not visited, mark it as visited, and
        # enqueue it
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)


if __name__ == '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
    print("Following is Breadth First Traversal: ")
    bfs(graph, 0)
```

**Output:**

```
Following is Breadth First Traversal:
0 1 2 3
```

## EXPERIMENT 5

**Aim:** Image Processing using OpenCV
**Theory**: The script utilizes the OpenCV library, a popular open-source computer vision library, for image processing tasks. OpenCV provides a comprehensive set of functions for tasks such as reading and manipulating images, color space conversions, and basic computer vision operations.

In this specific code:

cv2 (OpenCV): Used for reading and manipulating images, converting images to grayscale (cv2.cvtColor()), saving images (cv2.imwrite()), and displaying images (cv2.imshow(), cv2.waitKey(), cv2.destroyAllWindows()).

**Code:**

```
import cv2


def process_image(input_path, output_path):
    # Read the input image
    image = cv2.imread(input_path)


    # Convert the image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


    # Save the original and processed images
    cv2.imwrite(output_path + 'original.jpg', image)
    cv2.imwrite(output_path + 'grayscale.jpg', gray_image)


    # Display the images (optional)
    cv2.imshow('Original Image', image)

    cv2.imshow('Grayscale Image', gray_image)

    cv2.waitKey(0)

    cv2.destroyAllWindows()


if __name__ == "__main__":

    # Specify the path to your input image

    input_image_path = 'path/to/your/image.jpg'


    # Specify the path where you want to save the output images

    output_image_path = 'path/to/save/'


    process_image(input_image_path, output_image_path)
```

Output:

## EXPERIMENT 6

**Aim:** Use of open source computer vision programming tool open CV

**Theory:**

the OpenCV library is used to perform real-time edge detection on video frames captured from a webcam. Below is a brief description of the methods used:

1. **cv2.VideoCapture():** Opens a connection to the webcam or, alternatively, a specified video file. In this case, **capture = cv2.VideoCapture(0)** opens a connection to the default webcam (index 0).
2. **cv2.cvtColor():** Converts each frame captured from the webcam from the default BGR (Blue, Green, Red) color format to grayscale. This simplifies edge detection.
3. **cv2.Canny():** Applies the Canny edge detection algorithm to the grayscale frame. The parameters **(50, 150)** represent the lower and upper thresholds for edge detection.
4. **cv2.imshow():** Displays the original and processed frames in separate windows named 'Original' and 'Edges', respectively.
5. **cv2.waitKey() and cv2.destroyAllWindows():** The script enters a loop where it continuously captures frames, converts them to grayscale, applies edge detection, and displays the frames. The loop breaks when the 'q' key is pressed. After exiting the loop, the capture object is released, and all open windows are closed.

**Code:**

```
import cv2


def edge_detection(capture):
    while True:
        # Capture frame-by-frame
        ret, frame = capture.read()
        # If the frame was not read, exit the loop
        if not ret:
            break

        # Convert the frame to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Apply Canny edge detection
        edges = cv2.Canny(gray, 50, 150)

        # Display the original and processed frames
        cv2.imshow('Original', frame)
        cv2.imshow('Edges', edges)

        # Break the loop when 'q' key is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the capture object and close windows
    capture.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Open a connection to the webcam (you can also specify a video file)
    capture = cv2.VideoCapture(0)
```

```
if not capture.isOpened():
    print("Error: Couldn't open the webcam.")
else:
    # Call the edge_detection function with the capture object
    edge_detection(capture)
```

## EXPERIMENT 7

**Aim:** Image processing on input images

**Theory:**

The OpenCV library is used to perform various image processing operations on a given input image. Here's a summary of the script and the methods used:

1. **cv2 (OpenCV):** An open-source computer vision library that provides a wide range of image and video processing functions.
2. **cv2.imread():** Loads the input image specified by **input_path** using OpenCV's image reading function.
3. **cv2.imshow() and cv2.waitKey():** Displays the original, resized, grayscale, and blurred images in separate windows and waits for a key press to proceed to the next step.
4. **cv2.resize():** Resizes the original image to dimensions (400, 300) using OpenCV's image resizing function.
5. **cv2.cvtColor():** Converts the original image to grayscale using the BGR to grayscale color space conversion.
6. **cv2.GaussianBlur():** Applies a Gaussian blur to the grayscale image to reduce noise and smooth the image. The kernel size is (5, 5), and the standard deviation is set to 0.
7. **cv2.imwrite():** Saves the processed (blurred) image to the specified output path.
8. **cv2.imread():** Reads the saved processed image from the output path.
9. **cv2.destroyAllWindows():** Closes all open windows.

**Code:**

```
import cv2
import numpy as np

def image_processing(input_path, output_path):
    # Load the image
    original_image = cv2.imread(input_path)

    # Check if the image is loaded successfully
    if original_image is None:
        print("Error: Unable to load the image.")
        return
```

```
# Display the original image
cv2.imshow('Original Image', original_image)
cv2.waitKey(0)

# Resize the image
resized_image = cv2.resize(original_image, (400, 300))

# Display the resized image
cv2.imshow('Resized Image', resized_image)
cv2.waitKey(0)

# Convert the image to grayscale
grayscale_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
cv2.imshow('Grayscale Image', grayscale_image)
cv2.waitKey(0)

# Apply Gaussian blur to the grayscale image
blurred_image = cv2.GaussianBlur(grayscale_image, (5, 5), 0)

# Display the blurred image
cv2.imshow('Blurred Image', blurred_image)
cv2.waitKey(0)

# Save the processed image
cv2.imwrite(output_path, blurred_image)

# Display the processed image
processed_image = cv2.imread(output_path)
cv2.imshow('Processed Image', processed_image)
cv2.waitKey(0)

# Close all windows
cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    # Specify the path to your input image
    input_image_path = r"C:\Users\ASUS\OneDrive\Desktop\butterfly.jpg"

    # Specify the path where you want to save the output image
    output_image_path =
'"C:\\Users\\ASUS\\OneDrive\\Desktop\\processed_image.jpg'

    image_processing(input_image_path, output_image_path)
```

## EXPERIMENT 8

**Aim:** Artificial Intelligence-Based Chatbot for Appliance Control

**Theory:**

1. **NLTK (Natural Language Toolkit):** A powerful library for natural language processing tasks in Python.
2. **PorterStemmer:** An NLTK stemmer used to reduce words to their root form.
3. **word_tokenize:** A function from NLTK that tokenizes a sentence into individual words.
4. **Rules Dictionary (rules):** A set of predefined rules mapping specific user inputs to corresponding chatbot responses.
5. **get_response function:** Tokenizes and stems the user's input, then checks for matches in the predefined rules. If a match is found, the associated response is returned. If no match is found, a default response is provided.
6. **chat function:** Initiates a conversation with the user, repeatedly prompting for input. The chatbot responds based on the rules until the user types 'quit,' ending the conversation.
7. **Main Execution (if __name__ == "__main__":):** Calls the **chat** function, starting the chatbot interaction.

**Code:**

```
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Initialize the NLTK stemmer
ps = PorterStemmer()

# Define a simple set of rules
rules = {
    'hi': 'Hello!',
    'hello': 'Hi there!',
    'how are you': 'I am doing well, thank you!',
    'bye': 'Goodbye! Have a great day.',
    'quit': 'Goodbye! If you want to chat again, just type "hello".'
}
```

```
def get_response(message):
    # Tokenize and stem the input message
    words = word_tokenize(message.lower())
    stemmed_words = [ps.stem(word) for word in words]

    # Check if any rule matches
    for key in rules:
        if ps.stem(key) in stemmed_words:
            return rules[key]

    # Default response if no rule matches
    return "I'm sorry, I don't understand that."

def chat():
    print("Chatbot: Hi! Type 'quit' to exit.")

    while True:
        user_input = input("You: ").lower()

        if user_input == 'quit':
            print("Chatbot: Goodbye!")
            break

        response = get_response(user_input)
        print("Chatbot:", response)

if __name__ == "__main__":
    chat()
```

**Output:**

```
Chatbot: Hi! Type 'quit' to exit.

You: hi

Chatbot: Hello!

You: bye

Chatbot: Goodbye! Have a great day.
```

```
You: quit

Chatbot: Goodbye!
```

**EXPERINMENT 9**

**Aim :** Wireless Gesture-Controlled Robot

**Theory :**

**Hardware Components:**

1. **Gesture Sensor:**
    - Use a sensor that can capture gestures. Popular options include:
        - Accelerometer and Gyroscope Sensors: MPU6050, MPU9250, or similar sensors can be used to capture motion and orientation changes.
        - Camera-Based Systems: Depth cameras like Kinect or simple webcam-based solutions can be used for more complex gesture recognition.
2. **Microcontroller:**
    - Choose a microcontroller to process the sensor data and control the robot. Arduino and Raspberry Pi are common choices.
        - Arduino: Great for simpler projects.
        - Raspberry Pi: Offers more processing power and can handle more complex tasks.
3. **Wireless Module:**
    - Use a wireless module for communication between the gesture sensor/controller and the robot.
        - Bluetooth: Modules like HC-05/HC-06 for short-range control.
        - Wi-Fi: ESP8266 or ESP32 for longer-range control.
        - RF (Radio Frequency) modules: For basic wireless communication.
4. **Motor Driver and Motors:**
    - Choose motors and a motor driver that suit the size and weight of your robot.

    - DC Motors: Common for simple robots.
    - Servo Motors or Stepper Motors: For more precise control.

5. **Robot Chassis:**
    - Select a chassis that accommodates your chosen motors and allows space for other components.
6. **Power Supply:**
    - Ensure you have a suitable power source for your motors and electronics. It could be batteries or a rechargeable power bank.

**Software:**

1. **Gesture Recognition Algorithm:**
    - Implement a gesture recognition algorithm based on the sensor data. This could be simple threshold-based logic or a more sophisticated machine learning model.
2. **Microcontroller Code:**
    - Write code for your microcontroller to read data from the gesture sensor, interpret gestures, and send corresponding commands to the robot.
3. **Wireless Communication Code:**
    - Implement code for wireless communication to transmit gesture commands from the controller to the robot.
4. **Motor Control Code:**
    - Write code to control the motors based on the received gesture commands.

**Integration:**

1. **Connect Hardware:**
    - Connect the gesture sensor, microcontroller, wireless module, and motor driver according to your circuit diagram.
2. **Program Microcontroller:**
    - Upload the gesture recognition and communication code to the microcontroller.
3. **Assemble the Robot:**
    - Attach the motors and wheels to the chassis, ensuring proper balance and weight distribution.
4. **Power Up:**
    - Power up the robot with the required power source.
5. **Testing:**
    - Test the robot in a controlled environment to ensure that it responds correctly to your gestures.

## EXPERINMENT 10

**Aim:** Two Programming exercise for Robots

**Theory :** To solve this problem first we provide some input data. This input data contain **Arm length (l1 and l2), angle (θ1 and θ2) and the position of absolute coordinate(X0, Y0).** After giving this data we will calculate the value of the other coordinates by using above equation. To calculate the values of the coordinate for different angles we will use for loop. By using the for loop we get position of the links for different angles. By using plot command we will get graphical form of data. This graphical data is save by threre figure name and then we combine those figure to get animation of the robotics arm.

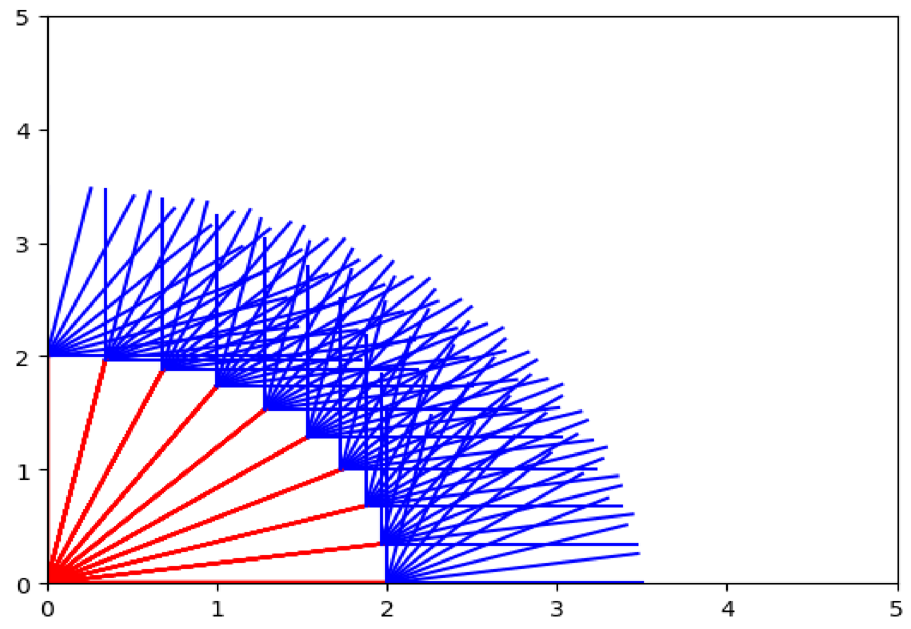
**Code :**

```
import math

import matplotlib.pyplot as plt


# Input Arm length

l1 = 2

l2 = 1.5


# Theta start and end values

theta_start = 0

theta_end = math.pi/2


# Number of theta values

n_theta = 10


# Define Angle variable

theta_1= []
```

```
theta_2= []

# Input Position of (x0,y0)
x0 = 0
y0 = 0

# Values of angle find out
for i in range(0,n_theta):
    theta_value = theta_start+i*(theta_end-theta_start)/(n_theta-1)
    theta_1.append(theta_value)
    theta_2.append(theta_value)

# A constant for getting movie frame.
k=1

# Calculate (x1,y1)
for i in theta_1:
    # Calculating x2, y2 for corresponding theta2 by taking theta 1 value one by one
    for j in theta_2:

        # Calculate coordinates (x1, y1)
        x1= l1* math.cos(i)
        y1= l1* math.sin(i)

        # Calculate (x2,y2)
        x2= x1 + l2*math.cos(j)
        y2= y1 + l2*math.sin(j)

        # Define plot file name for generate animation
        filename = str(k) + '.png'
        k=k+1
        print(filename)

        # Plot of robotics arm
        plt.figure(1)
        plt.plot([x0,x1], [y0,y1],'r')
        plt.plot([x1,x2], [y1,y2],'b')

        # Plot axis limit
        plt.xlim([0,5])
        plt.ylim([0,5])

        # Save plot figure
        plt.savefig(filename)
```

**OUTPUT :**

**100 images generated**


5
4
3
2
1
0
0
1
2
3
4
5

## EXPERINMENT 11

**Aim :** Two Case Studies of Application in Industry

**Theory :**

**Automotive Industry:** Robotic Assembly Lines

**Background:**

The automotive industry has been a pioneer in adopting robotics for manufacturing processes. One notable example is the use of robotic assembly lines in automobile manufacturing plants.

**Case Study 1: Tesla's Gigafactory**

Tesla's Gigafactory, known for producing electric vehicles and batteries, is a prime example of advanced robotic automation. The assembly line at the Gigafactory utilizes a large number of robots for tasks such as welding, painting, and assembling various components of the vehicles. These robots are equipped with advanced sensors and vision systems, allowing them to precisely carry out tasks that are repetitive, time-consuming, and require high precision.

**Benefits:**

Increased efficiency: Robots can work continuously without fatigue and at a faster pace than human workers.

Precision and quality: Robots ensure high precision in tasks like welding and assembly, leading to improved product quality.

Safety: Dangerous and physically demanding tasks can be assigned to robots, reducing the risk of injuries to human workers.

**Challenges:**

High initial costs: Setting up a robotic assembly line involves significant upfront investment.

Maintenance and programming: Regular maintenance and skilled programming are essential for optimal performance.

Healthcare Industry: Surgical Robots

**Background:**

Robotics in healthcare has seen significant advancements, particularly in the field of surgery. Surgical robots are designed to assist surgeons in performing minimally invasive procedures with enhanced precision.

**Case Study 2: da Vinci Surgical System**

The da Vinci Surgical System is a well-known example of a robotic system used in surgery. It consists of robotic arms controlled by a console, where the surgeon sits and operates the system. The robotic arms hold specialized surgical instruments and a camera, allowing the surgeon to perform intricate procedures through small incisions with 3D visualization and precise control.

**Benefits:**

Minimally invasive surgery: Surgical robots enable procedures with smaller incisions, reducing patient recovery time.

Enhanced precision: The robotic arms can make precise movements beyond the capabilities of human hands.

Improved outcomes: With 3D visualization and better control, surgeons can achieve better outcomes in complex procedures.

**Challenges:**

Cost: The initial investment and maintenance costs for surgical robots can be high.

Training: Surgeons need specialized training to operate robotic systems effectively.

## EXPERINMENT 12

**Aim :** Exercise on Robotic Simulation Software

**Theory :**

**Exercise:** Controlling a Simulated Robot with Python

**Objective:** Create a simple Python script to control the movement of a simulated robot in Gazebo.

**Requirements:**

1. ROS 1 installed on your Windows machine.
2. Gazebo simulator installed.
3. Python installed.
4.

**Steps:**

**1. Install ROS on Windows:**

Follow the instructions on the ROS on Windows installation page to install ROS on your Windows machine.

**2. Install Gazebo:**

Follow the instructions on the Gazebo installation page to install Gazebo on Windows.

**3. Create a ROS workspace:**

Open a Command Prompt and run the following commands:

**4.Create a ROS workspace:**

```
mkdir C:\ros_workspace\src
cd C:\ros_workspace
catkin_make
```

**5. Download a simple robot model and controller:**

```
cd C:\ros_workspace\src
git clone https://github.com/ros-simulation/gazebo_ros_demos.git
```

**6. Create a Python script for robot control:**

```
cd C:\ros_workspace\src\gazebo_ros_demos\gazebo_ros_tutorials\scripts
echo. > simple_robot_control.py
```

**7.(Code ) Edit simple_robot_control.py using a text editor**

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

def move_robot():
    rospy.init_node('simple_robot_control', anonymous=True)
    velocity_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()

    # Set linear velocity
    vel_msg.linear.x = 0.2
    vel_msg.linear.y = 0.0
    vel_msg.linear.z = 0.0

    # Set angular velocity
    vel_msg.angular.x = 0.0
    vel_msg.angular.y = 0.0
    vel_msg.angular.z = 0.2

    while not rospy.is_shutdown():
        velocity_publisher.publish(vel_msg)

if __name__ == '__main__':
    try:
        move_robot()
    except rospy.ROSInterruptException:
        pass
```

**8. Run the simulation:**

```
cd C:\ros_workspace.\install\setup.bat
```

```
roslaunch gazebo_ros_tutorials empty_world.launch
```

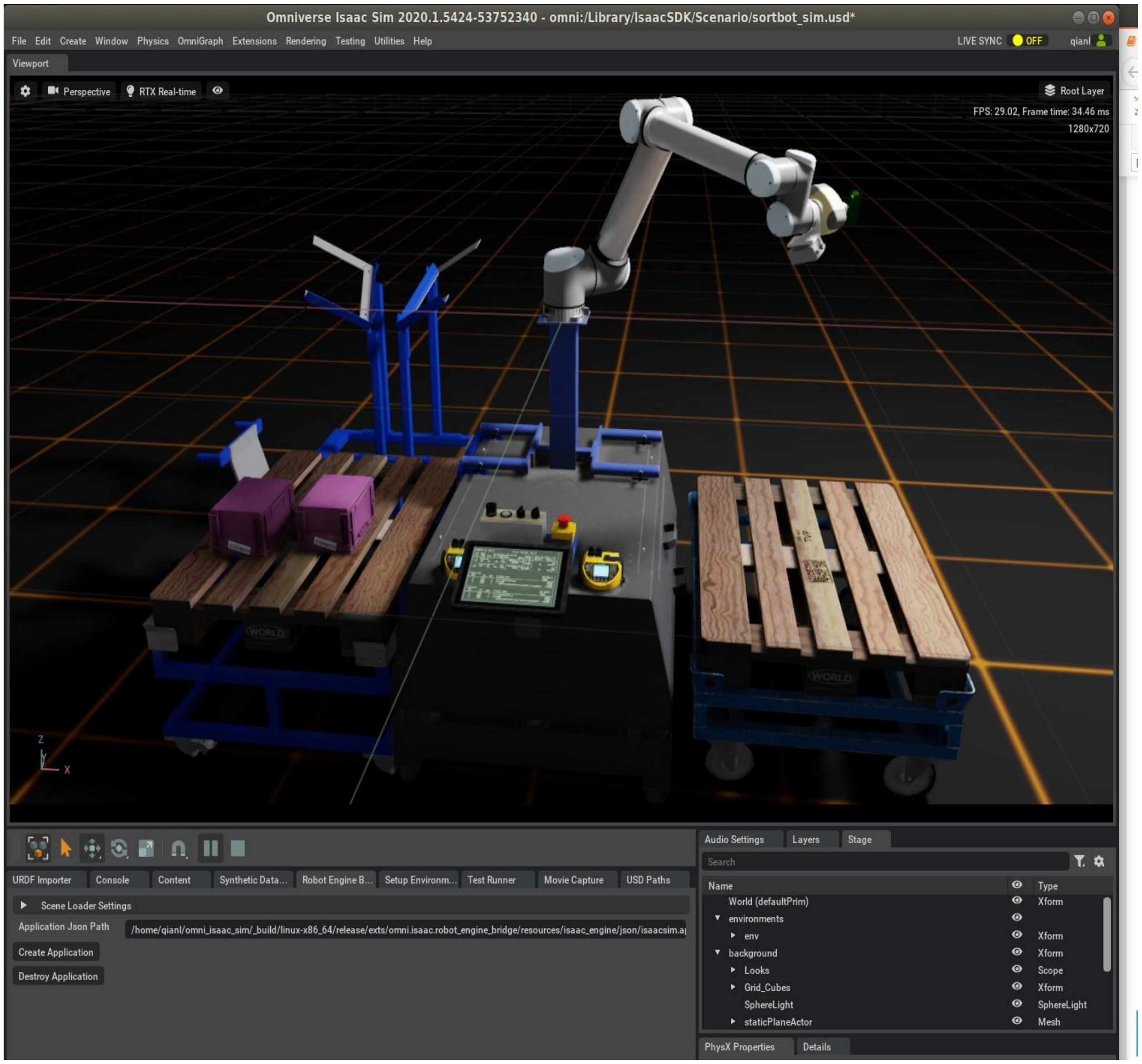
## 9. Open a new terminal:

```
cd C:\ros_workspace\src\gazebo_ros_demos\gazebo_ros_tutorials\scripts
python simple_robot_control.py
```

This script makes the robot move forward linearly and rotate. You can modify the script to perform different actions.

OUTPUT :

**Note :** Output Requires Physical Robotic Instruments and Arms to do Simulation and execute it


Omniverse Isaac Sim 2020.1.5424-53752340 - omni:/Library/IsaacSDK/Scenario/sortbot_sim.usd*
File Edit Create Window Physics OmniGraph Extensions Rendering Testing Utilities Help
LIVE SYNC OFF
qianl
Viewport
Perspective
RTX Real-time
Root Layer
FPS: 29.02, Frame time: 34.46 ms
1280x720
URDF Importer
Console
Content
Synthetic Data...
Robot Engine B...
Setup Environm...
Test Runner
Movie Capture
USD Paths
Scene Loader Settings
Application Json Path
/home/qianl/omni_isaac_sim/_build/linux-x86_64/release/exts/omni.isaac.robot_engine_bridge/resources/isaac_engine/json/isaacsim.a
Create Application
Destroy Application
Audio Settings
Layers
Stage
Search
Name
Type
World (defaultPrim)
Xform
environments
env
Xform
background
Xform
Looks
Scope
Grid_Cubes
Xform
SphereLight
SphereLight
staticPlaneActor
Mesh
PhysX Properties
Details