 RAJASTHAN TECHNICAL UNIVERSITY, KOTA

**Syllabus**

**IV Year-VIII Semester: B.Tech. Computer Science and Engineering (AI)**

|  |  |  |
| --- | --- | --- |
| **8CAI4-22: Robot Programming Lab** | | |
| **Credit: 1** | | **Max. Marks: 100(IA:60, ETE:40)** |
| **0L+0T+2P** | | **End Term Exam: 2 Hours** |
| **SN** | **List of Experiments** | | |
| **1** | An introduction to robot programming. | | |
| **2** | **Object Detection and Tracking Robot:** Create a robot with a camera that can detect and track objects in its field of view. Implement object detection algorithms and use them for tracking and interaction. | | |
| **3** | **Autonomous Maze Solving Robot:** Construct a robot that can autonomously navigate through a maze from the start to the finish. Implement maze-solving algorithms like A\* or Dijkstra's algorithm. | | |
| **4** | **Reinforcement Learning for Robotic Arm Control:** Train a robotic arm to perform tasks using reinforcement learning. Implement algorithms like Deep Q-Networks (DQN) or Proximal Policy Optimization (PPO) to optimize arm movements. | | |
| **5** | **Human-Robot Interaction using Natural Language Processing (NLP):** Design a robot that can understand and respond to voice commands. Use NLP techniques to process and interpret human language to control the robot's actions. | | |
| **6** | **Robot-Assisted Healthcare and Patient Interaction:** Design a robot that can assist patients and healthcare professionals. Use AI to understand patient needs, provide information, and interact in a helpful and empathetic manner. | | |
| **7** | **Gesture Recognition and Control of Robotic Arm:** Build a robotic arm that responds to hand gestures. Train a machine learning model to recognize gestures, and use them to control the movements of the robotic arm. | | |
| **8** | **Obstacle Avoidance Robot with Ultrasonic Sensors:** Develop a robot capable of navigating an environment while avoiding obstacles using ultrasonic sensors. Implement basic obstacle avoidance algorithms and refine the robot's movements. | | |

**Experiment – 1**

**Aim:** An introduction to robot programming.

**Introduction:**

This course delves into the exciting world of robotics, learning how to program robots to perform various tasks efficiently and effectively. This laboratory manual serves as a guide for understanding the fundamentals of robot programming, exploring key concepts, and gaining hands-on experience with programming robots.

**Objective:**

The objective of this laboratory session is to provide a comprehensive understanding of robot programming principles & techniques. By the end of this session, students should be able to:

* Understand the basics of robot programming languages.
* Familiarize with the common components of a robot programming environment.
* Develop essential skills in programming robots to perform specific tasks.
* Gain practical experience through hands-on exercises and projects.

**Prerequisites:**

Before proceeding with this laboratory session, it is assumed that students have a basic understanding of:

* Programming fundamentals, including variables, control structures, and functions.
* Mathematics, particularly algebra and geometry.
* Familiarity with basic robotics concepts such as sensors, actuators, and kinematics.

**Equipment and Software:**

For this laboratory session, following equipment and software are needed:

* **Robot platform:** This could be a physical robot or a simulator depending on the availability and requirements of the laboratory setup.
* **Programming environment:** A software environment where you can write, compile, and execute robot programs. Common examples include ROS (Robot Operating System), MATLAB Robotics Toolbox, or proprietary software provided by the robot manufacturer.
* **Computer with internet access:** To download necessary software, access online resources, and communicate with instructors or peers if required.

**Laboratory Activities:**

The laboratory session will consist of the following activities:

* **Introduction to Robot Programming Languages:** Students will explore different programming languages commonly used in robotics, such as Python, C++, and ROS-specific languages. Understanding the syntax and features of these languages is essential for effective robot programming.
* **Setting up the Programming Environment:** Students will learn how to set up the programming environment, including installing necessary software, configuring communication with the robot platform, and setting up simulation environments if applicable.
* **Basic Robot Control:** Students will start with simple exercises to control the robot's motion, including moving forward, backward, turning, and stopping. These exercises will help students understand how to send commands to the robot and control its behavior.
* **Sensor Integration:** Sensors play a crucial role in robotics by providing feedback about the robot's environment. Students will learn how to integrate sensors such as proximity sensors, cameras, and encoders into your robot programs to enable more sophisticated behaviors.
* **Path Planning and Navigation:** Students will delve into algorithms for path planning and navigation, allowing the robot to autonomously navigate its environment while avoiding obstacles. Concepts such as localization, mapping, and obstacle avoidance will be covered.
* **Advanced Topics (Optional):** Depending on the time and resources available, students may explore advanced topics such as robot manipulation, vision-based control, machine learning for robotics, or collaborative robotics.

**Conclusion:**

Robot programming is a fascinating field that combines elements of computer science, engineering, and mathematics. Through this laboratory session, students will gain valuable hands-on experience and develop the skills necessary to program robots for a wide range of applications.

**Experiment – 2**

**Aim:** Object Detection and Tracking Robot:

In this example, we'll use the MobileNet-SSD (Single Shot MultiBox Detector) model, which is a popular object detection model that's capable of detecting multiple objects in an image. First, you'll need to install the necessary libraries. You can do this with pip:

pip install opencv-python

pip install imutils

**Code:**

Here's a simple example of object detection using OpenCV and MobileNet-SSD:

import cv2

import imutils

**# Load pre-trained model**

net = cv2.dnn.readNetFromCaffe('MobileNetSSD\_deploy.prototxt.txt', 'MobileNetSSD\_deploy.caffemodel')

**# Initialize video stream**

vs = cv2.VideoCapture(0)

while True:

**# Read the next frame from the video stream**

\_, frame = vs.read()

**# Resize the frame to have a maximum width of 400 pixels**

frame = imutils.resize(frame, width=400)

**# Convert the frame to a blob**

blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300), 127.5)

**# Pass the blob through the network and obtain the detections**

net.setInput(blob)

detections = net.forward()

**# Loop over the detections**

for i in np.arange(0, detections.shape[2]):

**# Extract the confidence (i.e., probability) associated with the prediction**

confidence = detections[0, 0, i, 2]

**# Filter out weak detections by ensuring the `confidence` is greater than the minimum confidence**

if confidence > 0.2:

**# Extract the index of the class label from the `detections`**

idx = int(detections[0, 0, i, 1])

**# Draw the prediction on the frame**

label = "{}: {:.2f}%".format(CLASSES[idx], confidence \* 100)

cv2.putText(frame, label, (startX, startY-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

**# Show the output frame**

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

**# If the `q` key was pressed, break from the loop**

if key == ord("q"):

break

**# Clean up**

cv2.destroyAllWindows()

vs.stop()

**Experiment – 3**

**Aim:** Autonomous Maze Solving Robot:

The A algorithm is a popular path finding algorithm used in many applications, including games and robotics. It works by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

**Code:**

import heapq

def heuristic(a, b):

return abs(b[0] - a[0]) + abs(b[1] - a[1])

def astar(array, start, goal):

neighbors = [(0,1),(0,-1),(1,0),(-1,0)]

close\_set = set()

came\_from = {}

gscore = {start:0}

fscore = {start:heuristic(start, goal)}

oheap = []

heapq.heappush(oheap, (fscore[start], start))

while oheap:

current = heapq.heappop(oheap)[1]

if current == goal:

data = []

while current in came\_from:

data.append(current)

current = came\_from[current]

return data

close\_set.add(current)

for i, j in neighbors:

neighbor = current[0] + i, current[1] + j

tentative\_g\_score = gscore[current] + heuristic(current, neighbor)

if 0 <= neighbor[0] < array.shape[0]:

if 0 <= neighbor[1] < array.shape[1]:

if array[neighbor[0]][neighbor[1]] == 1:

continue

else:

# array bound y walls

continue

else:

# array bound x walls

continue

if neighbor in close\_set and tentative\_g\_score >= gscore.get(neighbor, 0):

continue

if tentative\_g\_score < gscore.get(neighbor, 0) or neighbor not in [i[1]for i in oheap]:

came\_from[neighbor] = current

gscore[neighbor] = tentative\_g\_score

fscore[neighbor] = tentative\_g\_score + heuristic(neighbor, goal)

heapq.heappush(oheap, (fscore[neighbor], neighbor))

return False

# Define the maze - 0 is open, 1 is blocked

maze = [[0, 0, 0, 0, 1, 0],

[1, 1, 0, 0, 1, 0],

[0, 0, 0, 1, 0, 0],

[0, 1, 1, 0, 0, 1],

[0, 0, 0, 0, 1, 0]]

start = (0, 0) # Starting position

end = (4, 5) # Ending position

path = astar(maze, start, end)

print(path)

**Experiment – 4**

**Aim:** Reinforcement Learning for Robotic Arm Control:

Training a robotic arm to perform tasks using reinforcement learning involves complex simulations and hardware setups. Here's a simplified Python code snippet using the OpenAI Gym environment and the Proximal Policy Optimization (PPO) algorithm to train a robotic arm in a simulated

environment:

import gym

import numpy as np

import tensorflow as tf

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow\_probability import distributions

# Define the environment

env = gym.make('Pendulum-v0')

# Define the neural network model

model = Sequential([

Dense(64, activation='relu', input\_shape=(env.observation\_space.shape[0],)),

Dense(64, activation='relu'),

Dense(env.action\_space.shape[0])

])

# Define the optimizer

optimizer = Adam(learning\_rate=0.001)

# Define the PPO agent

def ppo\_agent(model, optimizer):

action\_space = env.action\_space.shape[0]

def agent(observation):

observation = np.expand\_dims(observation, axis=0)

action\_logits = model(observation)

action\_distribution = distributions.Categorical(logits=action\_logits)

action = action\_distribution.sample()

return action.numpy()[0]

return agent

# Train the agent using PPO

agent = ppo\_agent(model, optimizer)

for episode in range(1000):

observation = env.reset()

done = False

while not done:

action = agent(observation)

next\_observation, reward, done, \_ = env.step(action)

observation = next\_observation

# Test the trained agent

observation = env.reset()

done = False

while not done:

action = agent(observation)

observation, reward, done, \_ = env.step(action)

env.render()

env.close()

**Experiment 5**

**Aim:** Human-Robot Interaction using Natural Language Processing (NLP):

Designing a robot that can understand and respond to voice commands involves integrating  speech recognition and natural language processing (NLP) techniques. Here's a simplified  Python code snippet using the SpeechRecognition library and the Natural Language Toolkit to  process voice commands and control the robot's actions:

**Code:**

import speech\_recognition as sr

import nltk

from nltk.tokenize import word\_tokenize

# Initialize the speech recognizer

recognizer = sr.Recognizer()

# Function to process voice commands

def process\_voice\_command():

with sr.Microphone() as source:

print("Listening...")

audio = recognizer.listen(source)

try:

command = recognizer.recognize\_google(audio)

print("You said:", command)

return command

except sr.UnknownValueError:

print("Sorry, I could not understand the command.")

return ""

except sr.RequestError:

print("Sorry, my speech service is down.")

return ""

# Function to interpret and respond to voice commands

def interpret\_and\_respond(command):

tokens = word\_tokenize(command)

if 'move' in tokens:

direction = tokens[tokens.index('move') + 1]

if direction == 'forward':

print("Moving the robot forward.")

elif direction == 'backward':

print("Moving the robot backward.")

else:

print("Invalid direction command.")

else:

print("Command not recognized.")

# Main loop to listen for voice commands

while True:

command = process\_voice\_command()

if command:

interpret\_and\_respond(command)

**Experiment 6**

**Aim:** Robot-Assisted Healthcare and Patient Interaction:

Designing a robot to assist patients and healthcare professionals involves integrating AI technologiesfor understanding patient needs, providing information, and interacting empathetically. This Python code snippet using the ChatterBot library to create a chatbot that interact with patients and

healthcare professionals:

In this code:

* We use the ChatterBot library to create a chatbot instance named 'HealthcareBot'.
* We train the chatbot on the English language corpus to provide responses to a wide range of

queries.

* The chat\_with\_bot() function allows users to interact with the chatbot by entering text inputs.
* The chatbot responds to user inputs based on the training data and the conversational patterns.

**Code:**

from chatterbot import ChatBot

from chatterbot.trainers import ChatterBotCorpusTrainer

# Create a ChatBot instance

chatbot = ChatBot('HealthcareBot')

# Create a new trainer for the chatbot

trainer = ChatterBotCorpusTrainer(chatbot)

# Train the chatbot on the English language corpus

trainer.train("chatterbot.corpus.english")

# Function to interact with the chatbot

def chat\_with\_bot():

print("HealthcareBot: Hello, how can I assist you today?")

while True:

user\_input = input("You: ")

if user\_input.lower() == 'exit':

print("HealthcareBot: Goodbye!")

break

response = chatbot.get\_response(user\_input)

print("HealthcareBot:", response)

# Start interacting with the chatbot

chat\_with\_bot()

**Experiment – 7**

**Aim:** Gesture Recognition and Control of Robotic Arm:

Building a robotic arm that responds to hand gestures involves training a machine learning model to recognize gestures and mapping them to specific robotic arm movements. Here's a simplified Python code snippet using the MediaPipe library for hand gesture recognition and controlling a robotic arm:

import mediapipe as mp

import cv2

**# Initialize MediaPipe Hands**

mp\_hands = mp.solutions.hands

hands = mp\_hands.Hands()

**# Initialize the robotic arm (simulated)**

class RoboticArm:

def move\_up(self):

print("Moving the robotic arm up")

def move\_down(self):

print("Moving the robotic arm down")

**# Create an instance of the robotic arm**

robotic\_arm = RoboticArm()

**# Function to control the robotic arm based on hand gestures**

def control\_robotic\_arm(hand\_landmarks):

**# Extract hand landmarks for gesture recognition**

**# Implement your gesture recognition logic here**

**# Example: If the hand is open, move the arm up; if the hand is closed, move the arm down**

if hand\_landmarks:

if hand\_landmarks[8].y > hand\_landmarks[5].y:

robotic\_arm.move\_up()

else:

robotic\_arm.move\_down()

**# Open a video stream**

cap = cv2.VideoCapture(0)

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

**# Process the frame to detect hand landmarks**

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

results = hands.process(frame\_rgb)

if results.multi\_hand\_landmarks:

for hand\_landmarks in results.multi\_hand\_landmarks:

control\_robotic\_arm(hand\_landmarks.landmark)

cv2.imshow('Robotic Arm Control', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

**# Release the video stream and close all windows**

cap.release()

cv2.destroyAllWindows()

In this code:

 MediaPipe library is used for hand gesture recognition to detect hand landmarks.

 A robotic arm is simulated with simple movements (up and down).

The control\_robotic\_arm() function interprets hand gestures to control the robotic arm

movements.

 The robotic arm moves up if the hand is open and down if the hand is closed.

 The code captures video frames from the camera, processes them for hand gesture recognition,  and controls the robotic arm based on the detected gestures.

**Experiment 8**

**Aim:** Obstacle Avoidance Robot with Ultrasonic Sensors:

Developing a robot capable of navigating an environment while avoiding obstacles using ultrasonic sensors involves integrating sensor data with obstacle avoidance algorithms. Here's a simplified Python code snippet using simulated sensor data and basic obstacle avoidance logic:

import random

class Robot:

def \_\_init\_\_(self):

self.position = [0, 0] # Initial position of the robot

self.obstacle\_threshold = 5 # Distance threshold to detect obstacles

self.max\_speed = 1 # Maximum speed of the robot

def move(self, direction, distance):

if direction == 'forward':

self.position[0] += distance

elif direction == 'backward':

self.position[0] -= distance

elif direction == 'right':

self.position[1] += distance

elif direction == 'left':

self.position[1] -= distance

def avoid\_obstacle(self):

obstacle\_detected = random.choice([True, False]) **# Simulated obstacle detection**

if obstacle\_detected:

print("Obstacle detected! Avoiding obstacle...")

self.move('backward', 1) **# Move backward to avoid the obstacle**

else:

print("No obstacle detected. Continuing forward...")

self.move('forward', self.max\_speed) **# Move forward**

**# Create an instance of the robot**

robot = Robot()

**# Simulate robot navigation with obstacle avoidance**

for \_ in range(10): **# Simulate 10 steps of movement**

robot.avoid\_obstacle()

print("Current Position:", robot.position)

In this code:

 We define a Robot class with methods to move the robot and avoid obstacles.

 The move() method updates the robot's position based on the specified direction and distance.

 The avoid\_obstacle() method simulates obstacle detection using random values and

Moves the robot backward if an obstacle is detected.

 We create an instance of the robot and simulate its navigation for 10 steps, printing the  current position at each step.