PERIYAR MANIAMMAI INSTITUTE OF SCIENCE AND TECHNOLOGY
THANJAVUR

DEPARTMENT OF ______________________________

(SPECIALIZATION WITH ROBOTICS)

ARTIFICIAL INTELLIGENCE AND COMPUTER VISION FOR ROBOTICS

LAB MANUAL

NAME: ____

REGISTER No.:____

INDEX

S.no	EXPERIMENT	Page no.

EXPERIMENT – 01: IMPLEMENTATION OF BREADTH FIRST SEARCH IN PYTHON

AIM: To write an algorithm for Breadth first a search.

PROCEDURE/STEPS:

Step 1: Open Jupyter Notebook
Step 2: Create a new Python file
Step 3: Import all required libraries
Step 4: Write the Code for Breadth first search

Python Code:

Creating data

```
graph={'Sam':['Aaron','Binny'],'Aaron':['Sam','Christine','Danny'],'Binny':['Elvin','Flin’],
'Christine':['Aaron'],'Danny':['Aaron'],'Elvin':['Binny','Gini'],'Flin':['Binny'],'Gini':['Elvin']}
```

Importing library

```
from collections import deque
```

Creating function

```
def bfs(graph,start,goal):
    visited=[]
    queue=deque([start])
    while queue:
        node=queue.popleft()
    if node not in visited:
        visited.append(node)
        print("I have visited:",node)
        neighbours=graph[node]
        if node==goal:
            return("I have reached the goal, this is my traversed path:",visited)
for neighbour in neighbours:
queue.append(neighbour)
```

RESULT: Hence, We have created and implemented Breadth first search algorithm.

EXPERIMENT – 02: IMPLEMENTATION OF A* ALGORITHM IN PYTHON

AIM: To write an algorithm for A* search in python

PROCEDURE/STEPS:

Step 1: Open Jupyter Notebook
Step 2: Create a new Python file
Step 3: Import all required libraries
Step 4: Write the Code for A* algorithm

Python Code

```
import networkx as nx
import matplotlib.pyplot as plt
def dist(a, b):
        (x1, y1) = a
        (x2, y2) = b
        return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5
G = nx.grid_graph(dim=[3, 3])
nx.set_edge_attributes(G, {e: e[1][0] * 2 for e in G.edges()}, "cost")
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels = True, node_color="#f86e00")
edge_labels = nx.get_edge_attributes(G, "cost")
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.show()
path = nx.astar_path(G, (0, 0), (2, 2), heuristic=dist, weight="cost")
length = nx.astar_path_length(G, (0, 0), (2, 2), heuristic=dist, weight="cost")
print('Path :',path)
print('Path Length',length)
```

RESULT: Hence, We have created and implemented A* algorithm.

EXPERIMENT – 03: BUILDING A REGRESSION MODEL IN PYTHON

AIM: To build a simple linear regression model using Scikit learn library.

PROCEDURE/STEPS:

1. Import all the required python libraries
2. Import Dataset
3. View the dataset
4. Remove unnecessary columns
5. Reshape the dataset
6. Divide dataset into training set and testing set
7. Import linear regression class
8. Create an object of the linear regression class
9. Fitting the data
10. Predicting the output

Program :

```
import warnings
warnings.simplefilter("ignore")
import numpy as np
import pandas as pd
dataset = pd.read_csv("Admission_Predict_Ver1.1.csv")
dataset
dataset = dataset.drop(['Serial No.','TOEFL Score','University
Rating','SOP','LOR','CGPA','Research'],axis=1)
dataset
x = dataset.iloc[:,0].values.reshape(-1,1)
y = dataset.iloc[:,-1].values.reshape(-1,1)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train,y_train)
y_pred = lm.predict(x_test)
```

RESULT: Thus a Simple linear regression model has been implemented successfully.

EXPERIMENT – 04: FUNDAMENTALS OF IMAGE PROCESSING

AIM: Perform fundamental operations on an image using OpenCV

PROCEDURE/STEPS:

Operations:

1. Import library
2. Reading an Image
3. Displaying an Image
4. Displaying a video
5. Converting image to gray scale
6. Wait for a key press to close the windows
7. Close all OpenCV windows

Pseudo Code:

# Import necessary libraries

```
import cv2
```

# Load an image from file

```
image = cv2.imread("image_path.jpg")
```

# Display the image

```
cv2.imshow("Original Image", image)
```

# Convert the image to grayscale

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

# Display the grayscale image

```
cv2.imshow("Grayscale Image", gray_image)
```

# Wait for a key press to close the windows

```
cv2.waitKey(0)
```

# Close all OpenCV windows

```
cv2.destroyAllWindows()
```

# Display a video

```
video_path = "video_file_path.mp4"
video_capture = cv2.VideoCapture(video_path)
if not video_capture.isOpened():
  print("Error: Could not open video file.")
  exit()
     while True:
       ret, frame = video_capture.read()
       if not ret:
          print("Error: Failed to read frame.")
          break
        cv2.imshow("Video", frame)
       if cv2.waitKey(1) & 0xFF == ord('q'):
          break
video_capture.release()
cv2.destroyAllWindows()
```

RESULT: Thus Basic operation of CV was executed successfully

EXPERIMENT – 05: IMAGE THRESHOLDING USING OPENCV IN PYTHON

AIM: Perform Image thresholding using OpenCV

PROCEDURE/STEPS:

Operations:

1. Import library
2. Load an Image
3. Convert Image to grayscale image
4. Apply Binary thresholding
5. Apply Binary inverse thresholding
6. Apply Adaptive thresholding (Gaussian)
7. Apply Adaptive thresholding (Mean)
8. Display the images

Pseudocode:

```
# Import necessary libraries
import cv2

# Load an image from file
image = cv2.imread("image_path.jpg")

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

# Apply Binary thresholding method
ret, binary_thresholded_image = cv2.threshold(gray_image, threshold_value,
max_value, cv2.THRESH_BINARY)

# Apply Binary inverse threshold method
ret, binary_inverse_thresholded_image = cv2.threshold(gray_image,
threshold_value, max_value, cv2.THRESH_BINARY_INV)


# Apply Adaptive thresholding (Gaussian)
adaptive_thresholded_image_gaussian = cv2.adaptiveThreshold(gray_image,
max_value, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, block_size, C)

# Apply Adaptive thresholding (Mean)
adaptive_thresholded_image_mean = cv2.adaptiveThreshold(gray_image,
max_value, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, block_size, C)

# Display the original and thresholded images for each method
cv2.imshow("Original Image", image)
```

```
cv2.imshow("Binary Thresholded Image", binary_thresholded_image)
cv2.imshow("Binary Inverse Thresholded Image",
binary_inverse_thresholded_image)
cv2.imshow("Adaptive Thresholded Image (Gaussian)",
adaptive_thresholded_image_gaussian)
cv2.imshow("Adaptive Thresholded Image (Mean)",
adaptive_thresholded_image_mean)

# Wait for a key press to close the windows
cv2.waitKey(0)

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

RESULT: Thus image thresholding process using CV was executed successfully.

EXPERIMENT – 06: MORPHOLOGICAL OPERATIONS IN OPENCV

AIM: Perform Morphological Operations on an Image using OpenCV.

PROCEDURE/STEPS:

Operations:

1. Import library
2. Load an Image
3. Convert Image to grayscale image
4. Define a kernel for the operations
5. Erosion
6. Dilation
7. Opening
8. Closing
9. Top Hat
10. Bottom Hat
11. Display the outcome

Pseudocode:

# Import necessary libraries

```
import cv2
```

# Load an image from file

```
image = cv2.imread("image_path.jpg")
```

# Convert the image to grayscale

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

# Define a kernel for the operations

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_width,
kernel_height))
```

# Erosion

```
eroded_image = cv2.erode(gray_image, kernel, iterations=iterations)
```

# Dilation

```
dilated_image = cv2.dilate(gray_image, kernel, iterations=iterations)
```

# Opening (Erosion followed by dilation)

```
opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel,
iterations=iterations)
```

```
# Closing (Dilation followed by erosion)

closed_image = cv2.morphologyEx(gray_image, cv2.MORPH_CLOSE, kernel,
iterations=iterations)

# Morphological gradient (Difference between dilation and erosion)

gradient_image = cv2.morphologyEx(gray_image, cv2.MORPH_GRADIENT, kernel,
iterations=iterations)

# Top hat (Original image - Opening)

tophat_image = cv2.morphologyEx(gray_image, cv2.MORPH_TOPHAT, kernel,
iterations=iterations)

# Black hat (Closing - Original image)

blackhat_image = cv2.morphologyEx(gray_image, cv2.MORPH_BLACKHAT, kernel,
iterations=iterations)

# Display the original and processed images for each operation

cv2.imshow("Original Image", gray_image)

cv2.imshow("Eroded Image", eroded_image)

cv2.imshow("Dilated Image", dilated_image)

cv2.imshow("Opened Image", opened_image)

cv2.imshow("Closed Image", closed_image)

cv2.imshow("Gradient Image", gradient_image)

cv2.imshow("Top Hat Image", tophat_image)

cv2.imshow("Black Hat Image", blackhat_image)

# Wait for a key press to close the windows

cv2.waitKey(0)

# Close all OpenCV windows

cv2.destroyAllWindows()
```

RESULT: Thus morphological operations has be performed on an image Successfully.

EXPERIMENT – 07: EDGE DETECTION USING HOG

AIM:

To perform Feature extraction using Histogram of oriented gradients

PROCEDURE/STEPS:

# Import necessary libraries

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
% matplotlib inline
from skimage.feature import hog
from skimage import data,exposure
```

# Load an image from file

```
image = cv2.imread("image_path.jpg")
```

# Applying HOG

```
fc,img_hog = hog(img,orientations = 8, pixels_per_cell = (16,6),
cells_per_block = (1,1), visualize = True,multichannel = True)
```

# Rescaling an Image

```
rescale_inten = exposure.rescale_intensity(img_hog, in_range = (0,10))
```

# Displaying an Original Image and HOG features

```
figure,(a1,a2) = plt.subplots(1,2,figsize = (12,6), sharex = True, sharey =
True)
a1.axis('off')
a1.imshow(img)
a1.set_title('Original Image')

a2.axis('off')
a2.imshow(rescale_inten)
a2.set_title('Histogram of Oriented Gradients Image')
plt.show()
```

RESULT: Edge Detection using HOG was performed in CV successfully

Experiment 8: Face Detection using Haar Cascade method

Aim:

Build a Face Detection Model using Haar Cascade Method

Pseudocode:

```
# Import necessary libraries
import OpenCV
# Load the pre-trained Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
# Load an image from file
image = cv2.imread("image_path.jpg")
#Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Perform face detection using the Haar cascade classifier
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))
# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
# Display the image with detected faces
cv2.imshow("Detected Faces", image)
# Wait for a key press to close the window
cv2.waitKey(0)
# Close OpenCV window
cv2.destroyAllWindows()
```

Result: Thus face detection using Haar cascade method was executed successfully.