## Work Done (Coding)

**Hand-Gesture Regonition**

import time

import cv2

import mediapipe as mp

import pyautogui

cap = cv2.VideoCapture(0)

hand\_detector = mp.solutions.hands.Hands(min\_detection\_confidence=0.7, min\_tracking\_confidence=0.7)

drawing\_utils = mp.solutions.drawing\_utils

screen\_width, screen\_height = pyautogui.size()

index\_y = 0

click\_active = False

click\_threshold = 30

click\_duration = 60

last\_click\_time = 0

middle\_finger\_y = 0

right\_click\_active = False

right\_click\_threshold = 30

right\_click\_duration = 60

right\_last\_click\_time = 0

while True:

\_, frame = cap.read()

frame = cv2.flip(frame, 1)

frame\_height, frame\_width, \_ = frame.shape

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

output = hand\_detector.process(rgb\_frame)

hands = output.multi\_hand\_landmarks

if hands:

for hand in hands:

drawing\_utils.draw\_landmarks(frame, hand)

landmarks = hand.landmark

index\_finger = landmarks[8]

thumb = landmarks[4]

middle\_finger = landmarks[12]

index\_x = int(index\_finger.x \* frame\_width)

index\_y = int(index\_finger.y \* frame\_height)

thumb\_x = int(thumb.x \* frame\_width)

thumb\_y = int(thumb.y \* frame\_height)

middle\_x = int(middle\_finger.x \* frame\_width)

middle\_y = int(middle\_finger.y \* frame\_height)

cv2.circle(img=frame, center=(index\_x, index\_y), radius=10, color=(0, 255, 255))

cv2.circle(img=frame, center=(thumb\_x, thumb\_y), radius=10, color=(0, 255, 255))

cv2.circle(img=frame, center=(middle\_x, middle\_y), radius=10, color=(0, 255, 255))

index\_screen\_x, index\_screen\_y = screen\_width / frame\_width \* index\_x, screen\_height / frame\_height \* index\_y

pyautogui.moveTo(index\_screen\_x, index\_screen\_y)

if abs(index\_y - thumb\_y) < click\_threshold:

current\_time = time.time()

if not click\_active:

pyautogui.mouseDown(button='left')

click\_active = True

last\_click\_time = current\_time

elif current\_time - last\_click\_time > click\_duration:

pyautogui.mouseUp(button='left')

click\_active = False

else:

if click\_active:

pyautogui.mouseUp(button='left')

click\_active = False

if abs(index\_y - middle\_y) < right\_click\_threshold:

current\_time = time.time()

if not right\_click\_active:

pyautogui.mouseDown(button='right')

right\_click\_active = True

right\_last\_click\_time = current\_time

elif current\_time - right\_last\_click\_time > right\_click\_duration:

pyautogui.mouseUp(button='right')

right\_click\_active = False

else:

if right\_click\_active:

pyautogui.mouseUp(button='right')

right\_click\_active = False

else:

if click\_active:

pyautogui.mouseUp(button='left')

click\_active = False

if right\_click\_active:

pyautogui.mouseUp(button='right')

right\_click\_active = False

cv2.imshow('Hand Gesture Mouse', frame)

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

break

cap.release()

cv2.destroyAllWindows()

**White-Board.1 -**

import tkinter as tk

class Whiteboard:

def \_\_init\_\_(self, root, width=800, height=600):

self.root = root

self.width = width

self.height = height

self.canvas = tk.Canvas(self.root, width=self.width, height=self.height, bg="white")

self.canvas.pack()

# Bind mouse wheel event to zoom

self.canvas.bind("<MouseWheel>", self.zoom)

# Bind mouse click events to drawing and erasing

self.canvas.bind("<Button-1>", self.draw\_start)

self.canvas.bind("<B1-Motion>", self.draw\_move)

self.canvas.bind("<Button-3>", self.erase\_start)

self.canvas.bind("<B3-Motion>", self.erase\_move)

self.canvas.bind("<ButtonRelease-1>", self.draw\_end)

self.canvas.bind("<ButtonRelease-3>", self.erase\_end)

# Set drawing and erasing colors and thickness

self.draw\_color = "black"

self.erase\_color = "white"

self.thickness = 1

# Create color palette

self.color\_palette = tk.Frame(self.root)

self.color\_palette.pack(side=tk.TOP, fill=tk.X)

self.colors = ["red", "orange", "yellow", "green", "cyan", "blue", "violet", "black"]

self.color\_buttons = []

for i, color in enumerate(self.colors):

button = tk.Button(self.color\_palette, width=13, bg=color, command=lambda c=color: self.set\_draw\_color(c))

button.grid(row=0, column=i)

self.color\_buttons.append(button)

# Create pen and eraser buttons

self.pen\_button = tk.Button(self.root, text="Pen", font=("Helvetica", 12), bg="lightblue", activebackground="blue", bd=0, relief="solid", highlightthickness=0.5, command=self.set\_pen\_mode)

self.pen\_button.pack(side=tk.LEFT, padx=5)

self.erase\_button = tk.Button(self.root, text="Eraser", font=("Helvetica", 12), bg="pink", activebackground="purple", bd=0, relief="solid", highlightthickness=0, command=self.set\_erase\_mode)

self.erase\_button.pack(side=tk.RIGHT, padx=5)

# Create thickness bar

self.thickness\_var = tk.IntVar()

self.thickness\_slider = tk.Scale(self.root, from\_=1, to=50, orient=tk.HORIZONTAL, command=self.set\_thickness, variable=self.thickness\_var)

self.thickness\_slider.pack(side=tk.LEFT, padx=5)

# Create clear all button

self.clear\_button = tk.Button(self.root, text="Clear All", font=("Helvetica", 13), bg="lightsalmon", activebackground="darkorange", bd=0, relief="solid", highlightthickness=0, command=self.clear\_canvas)

self.clear\_button.pack(side=tk.BOTTOM, padx=5)

# Create zoom in and zoom out buttons

self.zoom\_in\_button = tk.Button(self.root, text="Zoom In", font=("Helvetica", 12), bg="lightgreen", activebackground="green", bd=0, relief="solid", highlightthickness=0, command=self.zoom\_in)

self.zoom\_in\_button.pack(side=tk.LEFT, padx=5)

self.zoom\_out\_button = tk.Button(self.root, text="Zoom Out", font=("Helvetica", 12), bg="lightcoral", activebackground="red", bd=0, relief="solid", highlightthickness=0, command=self.zoom\_out)

self.zoom\_out\_button.pack(side=tk.LEFT, padx=5)

def set\_pen\_mode(self):

self.draw\_color = "black"

def set\_erase\_mode(self):

self.erase\_color = "white"

def set\_draw\_color(self, color):

self.draw\_color = color

def set\_thickness(self, thickness):

self.thickness = int(thickness)

def draw\_start(self, event):

self.drawing = True

self.start\_x = event.x

self.start\_y = event.y

def draw\_move(self, event):

if self.drawing:

self.end\_x = event.x

self.end\_y = event.y

self.canvas.create\_line(self.start\_x, self.start\_y, self.end\_x, self.end\_y, width=self.thickness, fill=self.draw\_color, capstyle=tk.ROUND)

self.start\_x = self.end\_x

self.start\_y = self.end\_y

def draw\_end(self, event):

self.drawing = False

def erase\_start(self, event):

self.erase = True

self.start\_x = event.x

self.start\_y = event.y

def erase\_move(self, event):

if self.erase:

self.end\_x = event.x

self.end\_y = event.y

self.canvas.create\_rectangle(self.start\_x, self.start\_y, self.end\_x, self.end\_y, width=0, fill=self.erase\_color, outline="")

self.canvas.create\_rectangle(self.start\_x-24, self.start\_y-24, self.end\_x+24, self.end\_y+24, width=0, fill=self.erase\_color, outline="")

self.start\_x = self.end\_x

self.start\_y = self.end\_y

def erase\_end(self, event):

self.erase = False

def clear\_canvas(self):

self.canvas.delete("all")

def zoom(self, event):

if event.delta > 0:

self.canvas.scale("all", event.x, event.y, 1.1, 1.1)

else:

self.canvas.scale("all", event.x, event.y, 1/1.1, 1/1.1)

def zoom(self, event):

if event.delta > 0:

self.canvas.scale("all", event.x, event.y, 1.1, 1.1)

else:

self.canvas.scale("all", event.x, event.y, 1/1.1, 1/1.1)

def zoom\_in(self):

self.canvas.scale("all", self.canvas.winfo\_width()/2, self.canvas.winfo\_height()/2, 1.1, 1.1)

def zoom\_out(self):

self.canvas.scale("all", self.canvas.winfo\_width()/2, self.canvas.winfo\_height()/2, 1/1.1, 1/1.1)

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = Whiteboard(root)

root.mainloop()

**White-Board.2 -**

import tkinter as tk

from tkinter import colorchooser, filedialog, messagebox

from PIL import Image, ImageDraw, ImageTk

import os

class InteractiveWhiteboard:

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

self.root = root

self.root.title("Interactive Whiteboard")

self.root.geometry("1000x700")

self.pen\_color = "black"

self.eraser\_on = False

self.pen\_size = 5

self.image = Image.new("RGB", (1000, 700), "white")

self.draw = ImageDraw.Draw(self.image)

self.history = [self.image.copy()]

self.history\_index = 0

self.scale\_factor = 1.0

self.create\_widgets()

self.bind\_events()

# Variables to handle drawing

self.drawing = False

self.last\_x = None

self.last\_y = None

def create\_widgets(self):

self.canvas = tk.Canvas(self.root, bg="white", cursor="cross")

self.canvas.pack(fill=tk.BOTH, expand=True)

self.bottom\_frame = tk.Frame(self.root, bg="lightgrey", height=50)

self.bottom\_frame.pack(fill=tk.X, side=tk.BOTTOM)

self.pen\_button = tk.Button(self.bottom\_frame, text="Pen", command=self.use\_pen, font=("Helvetica", 12), bg="lightblue", activebackground="blue", bd=0, relief="solid", highlightthickness=0.5)

self.pen\_button.grid(row=0, column=0, padx=5)

self.color\_button = tk.Button(self.bottom\_frame, text="Color", command=self.choose\_color, font=("Helvetica", 12), bg="white", activebackground="lightgrey", bd=0, relief="solid", highlightthickness=0.5)

self.color\_button.grid(row=0, column=1, padx=5)

self.eraser\_button = tk.Button(self.bottom\_frame, text="Eraser", command=self.use\_eraser, font=("Helvetica", 12), bg="pink", activebackground="purple", bd=0, relief="solid", highlightthickness=0.5)

self.eraser\_button.grid(row=0, column=2, padx=5)

self.clear\_button = tk.Button(self.bottom\_frame, text="Clear All", command=self.clear\_canvas, font=("Helvetica", 12), bg="lightsalmon", activebackground="darkorange", bd=0, relief="solid", highlightthickness=0.5)

self.clear\_button.grid(row=0, column=3, padx=5)

self.pen\_size\_scale = tk.Scale(self.bottom\_frame, from\_=1, to=50, orient=tk.HORIZONTAL, command=self.change\_pen\_size)

self.pen\_size\_scale.set(self.pen\_size)

self.pen\_size\_scale.grid(row=0, column=4, padx=5)

self.zoom\_in\_button = tk.Button(self.bottom\_frame, text="Zoom In", command=self.zoom\_in, font=("Helvetica", 12), bg="lightgreen", activebackground="green", bd=0, relief="solid", highlightthickness=0.5)

self.zoom\_in\_button.grid(row=0, column=5, padx=5)

self.zoom\_out\_button = tk.Button(self.bottom\_frame, text="Zoom Out", command=self.zoom\_out, font=("Helvetica", 12), bg="lightcoral", activebackground="red", bd=0, relief="solid", highlightthickness=0.5)

self.zoom\_out\_button.grid(row=0, column=6, padx=5)

self.save\_button = tk.Button(self.bottom\_frame, text="Save", command=self.save\_canvas, font=("Helvetica", 12), bg="lightyellow", activebackground="yellow", bd=0, relief="solid", highlightthickness=0.5)

self.save\_button.grid(row=0, column=7, padx=5)

self.load\_button = tk.Button(self.bottom\_frame, text="Load", command=self.load\_canvas, font=("Helvetica", 12), bg="lightyellow", activebackground="yellow", bd=0, relief="solid", highlightthickness=0.5)

self.load\_button.grid(row=0, column=8, padx=5)

self.undo\_button = tk.Button(self.bottom\_frame, text="Undo", command=self.undo, font=("Helvetica", 12), bg="lightblue", activebackground="blue", bd=0, relief="solid", highlightthickness=0.5)

self.undo\_button.grid(row=0, column=9, padx=5)

self.redo\_button = tk.Button(self.bottom\_frame, text="Redo", command=self.redo, font=("Helvetica", 12), bg="lightblue", activebackground="blue", bd=0, relief="solid", highlightthickness=0.5)

self.redo\_button.grid(row=0, column=10, padx=5)

# Create color palette

self.colors = ["red", "orange", "yellow", "green", "cyan", "blue", "violet", "black"]

self.color\_buttons = []

for i, color in enumerate(self.colors):

button = tk.Button(self.bottom\_frame, width=3, bg=color, command=lambda c=color: self.set\_draw\_color(c))

button.grid(row=0, column=11 + i)

self.color\_buttons.append(button)

def bind\_events(self):

self.canvas.bind("<B1-Motion>", self.paint)

self.canvas.bind("<ButtonPress-1>", self.start\_drawing)

self.canvas.bind("<ButtonRelease-1>", self.reset)

self.canvas.bind("<MouseWheel>", self.zoom)

def use\_pen(self):

self.eraser\_on = False

def choose\_color(self):

color = colorchooser.askcolor()

if color[1]:

self.pen\_color = color[1]

self.eraser\_on = False

def use\_eraser(self):

self.eraser\_on = True

def set\_draw\_color(self, color):

self.pen\_color = color

self.eraser\_on = False

def change\_pen\_size(self, event):

self.pen\_size = int(event)

def clear\_canvas(self):

self.canvas.delete("all")

self.image = Image.new("RGB", (1000, 700), "white")

self.draw = ImageDraw.Draw(self.image)

self.history = [self.image.copy()]

self.history\_index = 0

self.update\_canvas()

def zoom(self, event):

if event.delta > 0:

self.zoom\_in()

else:

self.zoom\_out()

def zoom\_in(self):

self.scale\_factor \*= 1.1

self.canvas.scale("all", 0, 0, 1.1, 1.1)

self.canvas.configure(scrollregion=self.canvas.bbox("all"))

def zoom\_out(self):

self.scale\_factor /= 1.1

self.canvas.scale("all", 0, 0, 1 / 1.1, 1 / 1.1)

self.canvas.configure(scrollregion=self.canvas.bbox("all"))

def save\_canvas(self):

file\_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "\*.png")])

if file\_path:

self.image.save(file\_path)

def load\_canvas(self):

file\_path = filedialog.askopenfilename(filetypes=[("PNG files", "\*.png")])

if file\_path:

self.image = Image.open(file\_path)

self.draw = ImageDraw.Draw(self.image)

self.history = [self.image.copy()]

self.history\_index = 0

self.update\_canvas()

def update\_canvas(self):

self.canvas\_image = ImageTk.PhotoImage(self.image)

self.canvas.create\_image(0, 0, anchor=tk.NW, image=self.canvas\_image)

self.canvas.image = self.canvas\_image

def undo(self):

if self.history\_index > 0:

self.history\_index -= 1

self.image = self.history[self.history\_index].copy()

self.draw = ImageDraw.Draw(self.image)

self.update\_canvas()

def redo(self):

if self.history\_index < len(self.history) - 1:

self.history\_index += 1

self.image = self.history[self.history\_index].copy()

self.draw = ImageDraw.Draw(self.image)

self.update\_canvas()

def save\_action(self):

if self.history\_index < len(self.history) - 1:

self.history = self.history[:self.history\_index + 1]

self.history.append(self.image.copy())

self.history\_index += 1

def start\_drawing(self, event):

self.drawing = True

self.last\_x = event.x / self.scale\_factor

self.last\_y = event.y / self.scale\_factor

def paint(self, event):

if self.drawing:

x1, y1 = self.last\_x, self.last\_y

x2, y2 = event.x / self.scale\_factor, event.y / self.scale\_factor

paint\_color = "white" if self.eraser\_on else self.pen\_color

self.canvas.create\_line(x1 \* self.scale\_factor, y1 \* self.scale\_factor,

x2 \* self.scale\_factor, y2 \* self.scale\_factor,

fill=paint\_color, width=self.pen\_size, capstyle=tk.ROUND, smooth=tk.TRUE)

self.draw.line([x1, y1, x2, y2], fill=paint\_color, width=self.pen\_size)

self.last\_x, self.last\_y = x2, y2

def reset(self, event):

self.drawing = False

self.last\_x, self.last\_y = None, None

self.save\_action()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = InteractiveWhiteboard(root)

root.mainloop()