<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hand Detection</title>

<!-- Import MediaPipe and Drawing Utilities -->

<script src="https://cdn.jsdelivr.net/npm/@mediapipe/drawing\_utils/drawing\_utils.js"

crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/@mediapipe/hands/hands.js" crossorigin="anonymous"></script>

<!-- Minimal CSS to center video and canvas -->

<style>

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

video,

canvas {

position: absolute;

transform: rotateY(180deg);

}

/\* Mirror video and canvas \*/

</style>

</head>

<body>

<!-- Video and Canvas Elements for Real-Time Detection -->

<video id="webcam" autoplay playsinline></video>

<canvas id="output\_canvas"></canvas>

<!-- Main JavaScript for Hand Landmark Detection -->

<script type="module">

import { HandLandmarker, FilesetResolver } from "https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@0.10.0";

let handLandmarker; // Hand landmark detection instance

let runningMode = "VIDEO"; // Set running mode to video for real-time detection

let lastVideoTime = -1; // Track video frame timing

// Initialize hand landmark detector

const initializeHandLandmarker = async () => {

const vision = await FilesetResolver.forVisionTasks("https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision@0.10.0/wasm");

handLandmarker = await HandLandmarker.createFromOptions(vision, {

baseOptions: {

modelAssetPath: "https://storage.googleapis.com/mediapipe-models/hand\_landmarker/hand\_landmarker/float16/1/hand\_landmarker.task",

delegate: "GPU"

},

runningMode: runningMode,

numHands: 2

});

};

initializeHandLandmarker(); // Initialize landmarker

const video = document.getElementById("webcam"); // Webcam video element

const canvas = document.getElementById("output\_canvas"); // Canvas for drawing landmarks

const canvasCtx = canvas.getContext("2d");

// Enable webcam and set up real-time detection

if (navigator.mediaDevices?.getUserMedia) {

navigator.mediaDevices.getUserMedia({ video: true }).then((stream) => {

video.srcObject = stream;

video.addEventListener("loadeddata", predictWebcam);

});

}

function magnitude(vector) {

return Math.sqrt(vector.x \* 2 + vector.y \* 2 + vector.z \*\* 2);

}

// Función para calcular el producto punto entre dos vectores

function dotProduct(vectorA, vectorB) {

return vectorA.x \* vectorB.x + vectorA.y \* vectorB.y + vectorA.z \* vectorB.z;

}

// Función para calcular el coseno entre dos vectores

function cosineSimilarity(vectorA, vectorB) {

const dotProd = dotProduct(vectorA, vectorB);

const magA = magnitude(vectorA);

const magB = magnitude(vectorB);

return dotProd / (magA \* magB);

}

function calculateVector(pointA, pointB) {

return {

x: pointA.x - pointB.x,

y: pointA.y - pointB.y,

z: pointA.z - pointB.z

};

}

const fingertipIndices = [12, 16, 20];

// Predict landmarks on each video frame

async function predictWebcam() {

// Ensure canvas matches video dimensions

canvas.width = video.videoWidth;

canvas.height = video.videoHeight;

if (handLandmarker && video.currentTime !== lastVideoTime) {

lastVideoTime = video.currentTime;

// Detect hand landmarks in the current video frame

const results = await handLandmarker.detectForVideo(video, performance.now());

// Clear the canvas before each frame

canvasCtx.clearRect(0, 0, canvas.width, canvas.height);

// If landmarks are detected, iterate through them

if (results.landmarks) {

for (const landmarks of results.landmarks) {

const vectorPairs = [

{ start: 6, end: 8 },

{ start: 10, end: 12 },

{ start: 14, end: 16 },

{ start: 18, end: 20 }

];

// Calcular los vectores y la suma de los cosenos

const vectors = vectorPairs.map(pair => calculateVector(landmarks[pair.start], landmarks[pair.end]));

const cosineSum = vectors.slice(1).reduce((sum, vector, i) => {

return sum + cosineSimilarity(vectors[0], vector);

}, 0);

// Dibujar los puntos y conexiones de la mano

drawConnectors(canvasCtx, landmarks, HAND\_CONNECTIONS, { color: "#00FF00", lineWidth: 5 });

drawLandmarks(canvasCtx, landmarks, { color: "#FF0000", lineWidth: 2 });

// Dibujar cada punto de referencia y sus coordenadas

landmarks.forEach((landmark, index) => {

const { x, y, z } = landmark;

const canvasX = x \* canvas.width;

const canvasY = y \* canvas.height;

// Establecer el color basado en la condición del índice 8

canvasCtx.fillStyle = (index === 8 && cosineSum < -2.0) ? "#FFFF00" : "#0000FF";

// Dibujar un círculo en cada punto de referencia

canvasCtx.beginPath();

canvasCtx.arc(canvasX, canvasY, 5, 0, 2 \* Math.PI);

canvasCtx.fill();

// Dibujar las coordenadas como texto cerca de cada punto de referencia

canvasCtx.fillStyle = "#0000FF"; // Color del texto

canvasCtx.font = "12px Arial";

canvasCtx.fillText(`(${canvasX.toFixed(1)}, ${canvasY.toFixed(1)}, ${z.toFixed(2)})`, canvasX, canvasY);

});

}

}

}

// Call this function again for the next frame

requestAnimationFrame(predictWebcam);

}

</script>

</body>

</html>