<!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: 1

});

};

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);

});

}

// Suma de dos vectores

function vectorAdd(vec1, vec2) {

if (vec1.length !== vec2.length) throw new Error('Los vectores deben tener la misma longitud');

return vec1.map((val, index) => val + vec2[index]);

}

// Producto de un escalar por un vector

function scalarMultiply(scalar, vec) {

return vec.map(val => scalar \* val);

}

// Producto punto entre dos vectores

function dotProduct(vec1, vec2) {

if (vec1.length !== vec2.length) throw new Error('Los vectores deben tener la misma longitud');

return vec1.reduce((sum, val, index) => sum + val \* vec2[index], 0);

}

// Magnitud de un vector

function magnitude(vec) {

return Math.sqrt(vec.reduce((sum, val) => sum + val \* val, 0));

}

// Coseno del ángulo entre dos vectores

function cosineBetweenVectors(vec1, vec2) {

const dotProd = dotProduct(vec1, vec2);

const magVec1 = magnitude(vec1);

const magVec2 = magnitude(vec2);

if (magVec1 === 0 || magVec2 === 0) throw new Error('La magnitud de un vector no puede ser cero');

return dotProd / (magVec1 \* magVec2);

}

// 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) {

//calculating vectors

const a\_vectorIndice = [landmarks[7].x-landmarks[6].x,

landmarks[7].y-landmarks[6].y,

landmarks[7].z-landmarks[6].z];

const a\_vectorMedio = [landmarks[11].x-landmarks[10].x,

landmarks[11].y-landmarks[10].y,

landmarks[11].z-landmarks[10].z];

const a\_vectorPulgar = [landmarks[3].x-landmarks[2].x,

landmarks[3].y-landmarks[2].y,

landmarks[3].z-landmarks[2].z];

const a\_vectorCorazon = [landmarks[15].x-landmarks[14].x,

landmarks[15].y-landmarks[14].y,

landmarks[15].z-landmarks[14].z];

const a\_vectorMenique = [landmarks[19].x-landmarks[18].x,

landmarks[19].y-landmarks[18].y,

landmarks[19].z-landmarks[18].z];

//console.log("a\_vectorIndice",a\_vectorIndice);

//console.log("a\_vectorMedio",a\_vectorMedio);

//console.log("a\_vectorPulgar",a\_vectorPulgar);

//console.log("a\_vectorCorazon",a\_vectorCorazon);

//console.log("a\_vectorMenique",a\_vectorMenique);

//calculating Cos between elements

const i\_cosIndicePulgar = cosineBetweenVectors(a\_vectorIndice,a\_vectorPulgar);

const i\_cosIndiceMedio = cosineBetweenVectors(a\_vectorIndice,a\_vectorMedio);

const i\_cosIndiceCorazon = cosineBetweenVectors(a\_vectorIndice,a\_vectorCorazon);

const i\_cosIndiceMenique = cosineBetweenVectors(a\_vectorIndice,a\_vectorMenique);

//console.clear();

//console.log("i\_cosIndicePulgar",i\_cosIndicePulgar);

//console.log("i\_cosIndiceMedio",i\_cosIndiceMedio);

//console.log("i\_cosIndiceCorazon",i\_cosIndiceCorazon);

//console.log("i\_cosIndiceMenique",i\_cosIndiceMenique);

const i\_acum = i\_cosIndicePulgar + i\_cosIndiceMedio + i\_cosIndiceCorazon + i\_cosIndiceMenique;

// Draw landmarks

console.log("i\_acum",i\_acum);

if(i\_acum > -2.5 && i\_acum < -1.5){

//drawLandmarks(canvasCtx, [landmarks[7],landmarks[6]], { color: "#00F000", lineWidth: 2 });

//drawLandmarks(canvasCtx, [landmarks[11],landmarks[10]], { color: "#FF0000", lineWidth: 2 });

//drawLandmarks(canvasCtx, [landmarks[3],landmarks[2]], { color: "#F00000", lineWidth: 2 });

//drawLandmarks(canvasCtx, [landmarks[15],landmarks[14]], { color: "#000F00", lineWidth: 2 });

//drawLandmarks(canvasCtx, [landmarks[19],landmarks[18]], { color: "#0000FF", lineWidth: 2 });

drawLandmarks(canvasCtx, [landmarks[8]], { color: "#FFF000", lineWidth: 2 });

}

}

}

}

// Call this function again for the next frame

requestAnimationFrame(predictWebcam);

}

</script>

</body>

</html>