Jhonatan Gonzalez Salguero - 20241695009

<!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>

<!-- Importar MediaPipe y utilidades de dibujo -->

<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>

<!-- CSS mínimo para centrar video y canvas -->

<style>

body { display: flex; justify-content: center; align-items: center; height: 100vh; margin: 0; }

video, canvas { position: absolute; transform: rotateY(180deg); } /\* Espejar video y canvas \*/

</style>

</head>

<body>

<!-- Elementos de video y canvas para detección en tiempo real -->

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

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

<!-- Código principal en JavaScript para detección de puntos clave -->

<script type="module">

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

let handLandmarker; // Instancia de detección de puntos clave de la mano

let runningMode = "VIDEO"; // Establecer el modo de ejecución a video para detección en tiempo real

let lastVideoTime = -1; // Seguir el tiempo del fotograma del video

// Inicializar el detector de puntos clave de la mano

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(); // Inicializar el detector

const video = document.getElementById("webcam"); // Elemento de video de la cámara web

const canvas = document.getElementById("output\_canvas"); // Canvas para dibujar los puntos clave

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

// Activar la cámara web y configurar la detección en tiempo real

if (navigator.mediaDevices?.getUserMedia) {

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

video.srcObject = stream;

video.addEventListener("loadeddata", predictWebcam);

});

}

// Función para calcular el ángulo entre tres puntos

function calculateAngle(p1, p2, p3) {

const vector1 = { x: p1.x - p2.x, y: p1.y - p2.y, z: p1.z - p2.z };

const vector2 = { x: p3.x - p2.x, y: p3.y - p2.y, z: p3.z - p2.z };

const dotProduct = vector1.x \* vector2.x + vector1.y \* vector2.y + vector1.z \* vector2.z;

const magnitude1 = Math.sqrt(vector1.x \* vector1.x + vector1.y \* vector1.y + vector1.z \* vector1.z);

const magnitude2 = Math.sqrt(vector2.x \* vector2.x + vector2.y \* vector2.y + vector2.z \* vector2.z);

const cosineAngle = dotProduct / (magnitude1 \* magnitude2);

const angle = Math.acos(cosineAngle) \* (180 / Math.PI); // Convertir de radianes a grados

return angle;

}

// Predecir los puntos clave en cada fotograma del video

async function predictWebcam() {

// Asegurarse de que el canvas coincida con las dimensiones del video

canvas.width = video.videoWidth;

canvas.height = video.videoHeight;

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

lastVideoTime = video.currentTime;

// Detectar los puntos clave de la mano en el fotograma actual del video

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

// Limpiar el canvas antes de cada fotograma

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

// Si se detectan puntos clave, iterar sobre ellos

if (results.landmarks) {

for (const landmarks of results.landmarks) {

// Dibujar los puntos clave y sus conexiones

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

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

// Calcular los ángulos para el dedo índice (5-6-7)

const angleIndex = calculateAngle(landmarks[5], landmarks[6], landmarks[7]);

const angleMiddle = calculateAngle(landmarks[9], landmarks[10], landmarks[11]); // Dedo medio (9-10-11)

const angleRing = calculateAngle(landmarks[13], landmarks[14], landmarks[15]); // Dedo anular (13-14-15)

const anglePinky = calculateAngle(landmarks[17], landmarks[18], landmarks[19]); // Meñique (17-18-19)

// Verificar si el dedo índice está recto (ángulo cerca de 180)

const isIndexStraight = angleIndex > 160 && angleIndex < 200;

// Verificar si los otros dedos están doblados (ángulos menores a 90 grados)

const areFingersBent = angleMiddle < 90 && angleRing < 90 && anglePinky < 90;

// Mostrar el punto amarillo solo cuando el dedo índice esté recto y los otros dedos estén doblados

if (isIndexStraight && areFingersBent) {

const indexFinger = landmarks[8]; // Punta del dedo índice

const canvasX = indexFinger.x \* canvas.width;

const canvasY = indexFinger.y \* canvas.height;

// Dibujar el punto amarillo en el dedo índice

canvasCtx.fillStyle = "#FFFF00";

canvasCtx.beginPath();

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

canvasCtx.fill();

} else {

// Si no se cumplen las condiciones, mostrar el punto rojo (o no mostrar punto)

const indexFinger = landmarks[8];

const canvasX = indexFinger.x \* canvas.width;

const canvasY = indexFinger.y \* canvas.height;

canvasCtx.fillStyle = "#FF0000"; // Color rojo para el caso por defecto

canvasCtx.beginPath();

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

canvasCtx.fill();

}

// Opcional: Imprimir los ángulos en la consola para depuración

console.log(`Ángulo del dedo índice (5-6-7): ${angleIndex.toFixed(2)}°`);

console.log(`Ángulo del dedo medio (9-10-11): ${angleMiddle.toFixed(2)}°`);

console.log(`Ángulo del dedo anular (13-14-15): ${angleRing.toFixed(2)}°`);

console.log(`Ángulo del meñique (17-18-19): ${anglePinky.toFixed(2)}°`);

}

}

}

// Solicitar el siguiente fotograma

requestAnimationFrame(predictWebcam);

}

</script>

</body>

</html>