DESARROLLO DE INTERFACES NATURALES GESTUALES CON KINECT

Julia Saenz julisaenz99@gmail.com

Alejo Schön aleeschon@gmail.com

Luciano Nahuel Espinosa nachuespinosa@gmail.com

Laboratorio emmeLab
Facultad de Artes
Universidad Nacional de La Plata
Argentina

Introducción

Los gestos con las manos son parte inherente a la comunicación entre personas, ya sea como sustitución o acompañamiento de la comunicación verbal y su utilización como método de interacción humano-máquina (HCI) se viene explorando hace tiempo [1]. Existe un enorme repertorio de gestos posibles, que pueden categorizarse según diferentes parámetros: las partes del cuerpo que requiere (una o dos manos, solo dedos, mano y brazo, etc); si son estáticos o dinámicos; si se usan para comunicarse con la interfaz o para manipularla; y si son predefinidos anteriormente o si son de forma libre [2]. De la misma forma, se han desarrollado a lo largo de los años un gran número de tecnologías de captura y procesamiento óptico específicas para este tipo de interacción, con diversos niveles de complejidad, detalle de calibración y condiciones óptimas [3]. Para poder tomar una decisión apropiada a la hora de diseñar una interfaz controlada por gestos, es necesario hacer un

HCI

interfaces gestuales

detección del cuerpo

análisis previo de las posibilidades de diferentes tecnologías según la forma en la que detecta los gestos, las configuraciones que permiten, los datos que proporcionan y su rendimiento en diferentes situaciones.

Este trabajo busca analizar tres programas de detección de manos, cada uno utilizando un distinto método de captación: cascadas de Haar, método de detección de objetos que recorre la imagen en sectores cada vez más pequeños y busca en cada uno si aparecen ciertas características visuales relacionadas con el objeto que se haya entrenado para detectar [4]; blobs por diferencia, lo que implica identificar y delimitar las porciones más brillantes u oscuras de la imagen en regiones de interés llamadas blobs [5]; y modelos de red neuronal, una serie de capas de nodos o neuronas interconectadas que procesan una información y devuelven otra. La definición de un modelo se realiza mediante el entrenamiento de la red, proceso en el que se le muestran ejemplos de lo que se quiere que el modelo realice (por ejemplo, imágenes en las que haya o no manos) para que este identifique la serie de relaciones correctas [6].

Con este fin se desarrolló una interfaz gestual para la solicitud de turnos, como las que se pueden encontrar en hospitales, que puede ser controlada mediante cualquiera de los tres programas analizados. De cada uno se describe detalladamente el proceso de implementación teniendo en cuenta la facilidad de configuración en cada caso y se realizaron pruebas del desempeño en diversas condiciones espaciales. A partir de esto, se busca que este trabajo sirva como material pedagógico introductorio de las posibilidades y limitaciones en cuanto las diferentes formas de desarrollar una interfaz gestual, de tal forma que todo el código mencionado en este trabajo está disponible en un repositorio de GitHub[1].

Metodología

El sistema desarrollado para este trabajo cuenta con una interfaz gráfica que puede conectarse a tres diferentes programas de detección mediante Open Sound Control (OSC) [7]. Cada uno de estos programas detecta la palma de la mano de un usuario y envía la posición normalizada a la interfaz gráfica, que usa este dato para guiar el cursor a través de la pantalla de la misma forma que lo haría un mouse.

Se buscó rediseñar una interfaz ya existente y de uso habitual en espacios públicos, ya que estas necesitan poder funcionar correctamente diversos espacios y ser rápidamente comprensible por una gran variedad de usuarios. Se eligió desarrollar un gestor de turnos, comúnmente utilizado en hospitales, centros médicos y bancos, teniendo en consideración que este tipo de interfaces suelen tener una estructura y navegación simples e intuitivas. Por esta misma razón y tomando en cuenta que la experiencia del usuario tiene que ser lo más breve posible, se decidió interactuar únicamente a través del desplazamiento de la mano.

Figura 1

Interfaz táctil de gestión de turnos localizada en un hospital

La interfaz para la gestión de turnos fue desarrollada en Processing [8] y comprende de cuatro botones que son seleccionados cuando el usuario sostiene la mano sobre alguno de ellos por 4 segundos. Al terminar el proceso, muestra un mensaje de confirmación de turno y regresa al estado inicial.

En esta pantalla el usuario puede elegir cualquiera de las cuatro opciones depende que tramite quiera hacer

Figura 2

Interfaz de gestión de turnos realizada para el prototipo. En esta pantalla el usuario recibe feedback de la opción que seleccione y puede obtener su comprobante

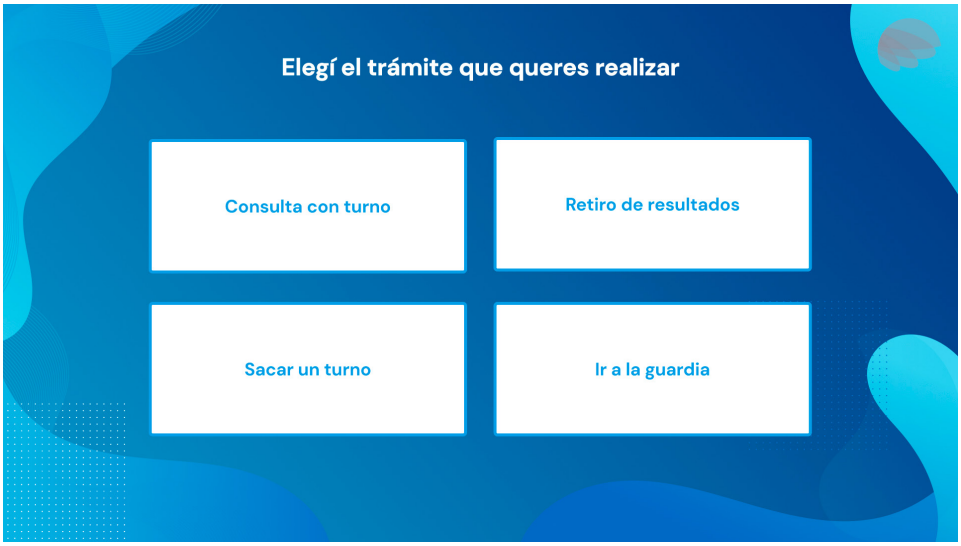


Figura 3

Interfaz de gestión de turnos realizada para el prototipo. En esta pantalla el usuario recibe feedback de la opción que seleccione y puede obtener su comprobante

Diseño de programas de detección

Para la selección de los programas y algoritmos de detección se tuvieron en cuenta dos condiciones: que fuesen de código abierto y sean compatibles con Processing, ya sea una librería propia o mediante algún sistema de comunicación como OSC. Por otro lado, también se buscó que los softwares elegidos tuviesen diferentes métodos de detección: por cascadas de Haar [4], por blobs [5] y por esqueleto con modelos de redes neuronales [6].

Los softwares a evaluar son OpenCV for Processing para la detección por cascadas de Haar [9], TSPS para la detección por blobs [10] y PoseNet para la detección de esqueleto con modelo de redes neuronales [11]. OpenCV for Processing es una librería de Processing que puede descargarse desde el propio entorno de programación mientras que TSPS y PoseNet se pueden conectar a través de sistemas de comunicación.

Cámara

Si bien todos los programas son compatibles con cámara web, en los casos que fuese posible la captura se realizó utilizando una Kinect v1, dispositivo de captura de cuerpo, movimiento y voz desarrollado por PrimeSense en 2010 para la consola de juegos Xbox 360 [12]. Se eligió este dispositivo ya que tiene su propio sistema de iluminación infrarroja, puede funcionar en situaciones de luz baja o variable y no es afectado por el color de los elementos en escena.

En el sistema operativo Windows el driver que se necesita para conectar la Kinect depende de la librería o versión del dispositivo que se utiliza. Para poder instalar o cambiar de driver fácilmente se usó Zadig [13], una aplicación para Windows que permite instalar drivers USB genéricos. Cuando se explique con detalle cada versión de la interfaz se especificará también el nombre del driver que sea necesario.

El único programa que usa una cámara web y no Kinect es PoseNet, ya que fue desarrollada específicamente para la detección de esqueletos con cámara web y aunque es posible conectarlo Kinect, mediante Runway [14] por ejemplo, es una solución contraintuitiva que solo empeora detección y el rendimiento de la interfaz.

Resultados

La interfaz se conecta con los programas de detección mediante OSC, recibiendo la posición de la mano del usuario y adaptándolo a la dimensión de la interfaz. Para realizar la conexión se necesita tener instalada la librería oscP5 en Processing [15], con la cual se instancia un objeto de tipo OscP5, que es el que maneja la comunicación, y un objeto de tipo NetAddress, que guarda la IP y el puerto que se escucha. Como en este caso los dos programas se corren en la misma computadora, la IP es “127.0.0.1”. Por último, se llama al método plug(), que recibe 3 paráme-

```
import oscP5.*;
import netP5.*;
OscP5 oscP5;
NetAddress myRemoteLocation;
void setup(){
  oscP5 = new OscP5(this, 2000);
  myRemoteLocation = new NetAddress("127.0.0.1", 1000);
  oscP5.plug(this, "posicionMano", "/posicionMano");
}
public void posicionMano(float x_, float y_) {
  //Acomoda los valores al tamaño de la pantalla
  x = x_*width;
  y = y_*height;
}
```

tros: una referencia a la instancia de Papplet de Processing en la cual se ejecuta, el nombre del método que va a ejecutar cada vez que llegue un mensaje por osc y el tag del mensaje que espera recibir. Este último tiene que coincidir con el tag del mensaje que esté siendo enviado el programa de detección.

Conectar los programas de esta forma permite probar la interfaz con diferentes programas de detección cambiando solo el puerto del que recibe esos los dos valores y se asegura de que cualquier problema en el control de la interfaz provenga de la detección y no de una diferencia entre interfaces.

Cada programa detector tiene también la librería oscP5 inicializada de la misma manera, solo que los puertos están invertidos: el puerto que escucha el programa principal debe tener el mismo valor que el inicializado en new OscP5() en el programa detector y viceversa.

Detección con OpenCV

Esta versión utiliza las librerías OpenKinect for Processing para leer la imagen de la cámara y OpenCV for Processing para el procesamiento de la imagen; el driver para Kinect necesario es lisbusbK. Con OpenCV se crea una instancia de un objeto de tipo OpenCV que se inicializa con dimensión de la imagen a detectar.

```
opencv =  new OpenCV(this, 640, 480);
```

Una vez inicializado se le carga una Cascada de Haar; OpenCV cuenta con una serie de cascadas integradas a las que se puede llamar directamente por nombre, pero como ninguna de ellas sirve para detectar manos, debe agregarse una nueva pasando la ubicación o ruta absoluta del archivo, lo cual genera el problema de tener que modificar la ruta cada vez que se cambia de máquina o el archivo se mueve de lugar.

Luego de agregar la cascada, se carga al objeto OpenCV la imagen sobre la que va a realizar la detección, en este caso el video que recibe de la Kinect, usando el método kinect.getVideoImage() de la librería OpenKinect.

```
opencv.loadCascade('C:/Usuario/Carpeta/Subcarpeta/archivo.xml', true);
opencv.loadImage(kinect.getVideoImage());
```

Teniendo ya la cascada y la imagen, se llama a la función detect(), la cual devuelve un arreglo de Rectangles. El método puede llamarse sin pasar ningún valor por parámetro o pasando cuatro valores que permiten controlar con mayor detalle cómo se está realizando la detección. Los valores son, en orden:

ScaleFactor: refiere a cuánto más chico es el sector en el que detecta la cascada cada pasada; el número está por defecto en 1.1, por lo que en cada pasada el sector será 10% más chico que el anterior. Un número más cercano a 1 implica una detección más lenta pero más meticulosa y un número más alejado corresponde con una detección más rápida pero menos precisa.

MinNeighbours: corresponde con la cantidad mínima de detecciones positivas que tiene que tener una detección para guardarse finalmente en el arreglo, por lo general un número entre 1 y 5 asegura una detección más precisa y evita falsos positivos.

flags: solía usarse para especificar distintos modos de detección pero que ahora no se utiliza, por lo que el valor se deja en 0.

minSize y maxSize: estos dos últimos parámetros indican el menor y mayor tamaño en píxeles que puede tener el objeto detectado, pudiendo así eliminar positivos que aparezcan más cerca o más lejos de lo que se espera.

```
Rectangle [] manos = opencv.detect(1.05, 50, 0, 30, 200);
```

El arreglo de rectángulos contiene todas las instancias de manos que la cascada haya detectado según los parámetros indicados. Como solo necesitamos la posición de una mano guardamos la posición del centro de la mano que corresponda al rectángulo más grande y suavizamos los valores a través de una interpolación lineal de la posición anterior y la actual para reducir el ruido de la captura. En este caso se guarda la mano más grande ya que es menos probable que haya una mano más cercana que la del usuario que esté interactuando. Finalmente la posición es normalizada y enviada al programa de la interfaz mediante OSC.

Detección con TSPS

Esta versión requiere de la aplicación TSPS y el driver para Kinect libusb0. Si bien TSPS usa su propio protocolo de comunicación (TUIO), y existe una librería para leer dicho protocolo en Processing, en nuestro desarrollo no se usa dicha librería ya que entra en conflicto con la librería de oscP5 que se necesita para la comunicación con la interfaz.

Se debe instalar TSPS de la página oficial y al abrirlo empieza a detectar automáticamente la cámara web con la configuración por defecto, la cual

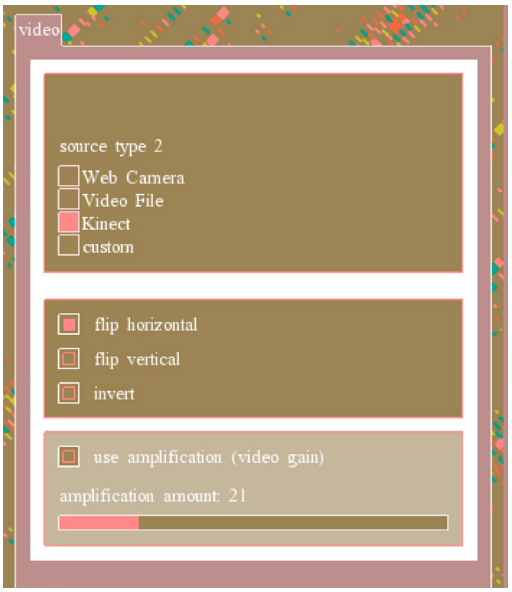


Figura 4

Configuración de la pestaña source

puede ser modificada de varias formas: se puede ir modificando cada slider e ir viendo su efecto en la detección en tiempo real o se puede cargar una configuración definida anteriormente mediante el botón load. En este caso iremos configurando y explicando cada parámetro, pero una vez hecha la primera configuración, puede guardarse como la predeterminada con el botón save, o guardarse como un archivo para poder acceder luego con el botón save as. Es recomendable para un sensado óptimo recalibrar el programa la primera vez que se utiliza en un espacio nuevo.

Los parámetros de TSPS están divididos en 4 categorías principales: *source*, *sensing*, *communication* y *data*. La pestaña source permite elegir la entrada de video ya sea cámara web, Kinect, videos ya grabados o customizado; cada una de estas entradas puede espejarse horizontal o verticalmente y es posible agregarle grano. Para este caso solo seleccionamos Kinect y espejamos horizontalmente la imagen.

La categoría de sensing es donde se va a realizar la calibración de la detección y la imagen. Está dividida en tres categorías: *background*, *differencing* y *tracking*. La pestaña de background permite sacar una captura del fondo, que sirve para evitar que el programa capture cosas permanentes en la escena. Se puede capturar una vez, o programar para que se vaya actualizando progresivamente según un tiempo determinado. En esta configuración solo sacamos una captura al principio de la ejecución.

Figura 5

Configuración background

La sección de differencing permite modificar el threshold (umbral de luz) de la escena, lo cual sirve para pintar cada pixel de blanco o negro dependiendo de su luminocidad; mientras más alto el umbral, más luz necesita un objeto para ser detectado. Con la Kinect, como la imagen que devuelve varía los niveles de grises según la cercanía, dependiendo el valor del threshold se modifica la distancia máxima posible de detectar. La pesta-

Figura 6

Configuración differencing

ña también permite usar highpass, un filtro que determina si un píxel es blanco o no en base a el píxel correspondiente en la imagen original así como los píxeles vecinos a este, resaltando en blanco solo los píxeles que superen un umbral. Finalmente se puede agregar un suavizado de forma. A continuación se muestra la configuración de este programa pensado para detectar manos a distancia de 1,5 metros como máximo.

La última sección de sensing es la configuración del tracking; se puede configurar por un lado el tamaño mínimo y máximo de un blob según el porcentaje que ocupa en pantalla y utilizar cascadas de haar para complementar la detección. Como solo estamos usando la información de profundidad de la Kinect, la detección por cascadas no nos es útil y solo configuramos el rango de tamaño de blobs deseado.

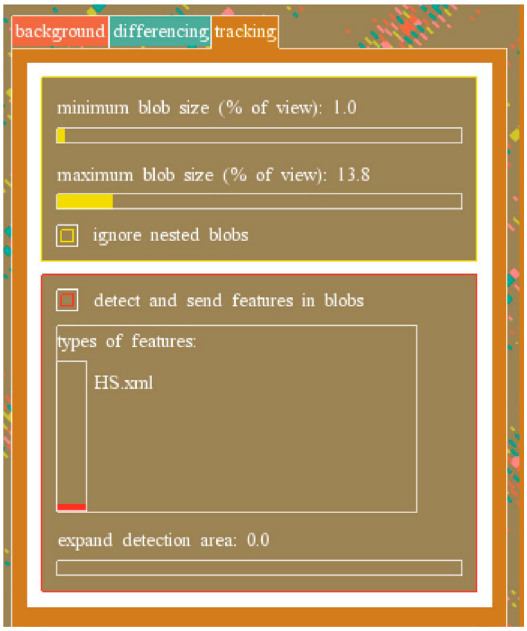


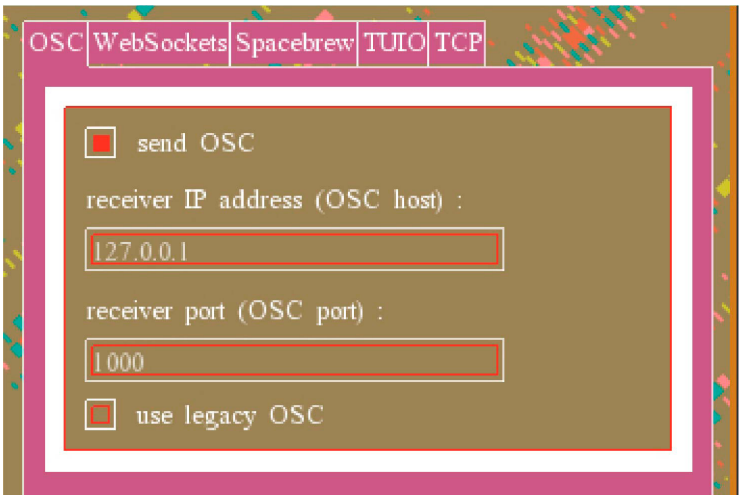
Figura 7

Configuración tracking

La pestaña de comunicación es la que permite enviar los datos a otros programas, puede comunicarse mediante OSC, WebSockets, SpaceBrew, TUIO o TCP. En este caso nos enviamos los datos por OSC asegurándonos de tener la dirección de IP y el puerto de nuestro programa de Processing.

Figura 8

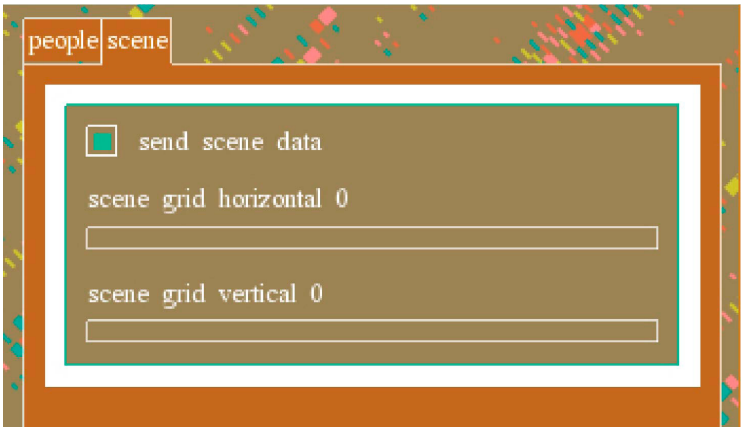
Configuración de la pestaña communication



Por último, la pestaña de data permite configurar la información que se va a enviar. Dentro de data, la pestaña people permite seleccionar si se manda información de los contornos o del flujo óptico y la pestaña scene permite mandar información de la escena y, si se quisiese, dividirla en una grilla. Para este caso necesitamos únicamente saber la posición de la mano, por lo que enviamos solo los datos de la escena.

Figura 9

Configuración de la pestaña data



Teniendo ya la configuración de TSPS optimizada para detectar la mano, se envía el mensaje por osc a Processing para ser interpretado. El formato del mensaje de osc tiene dos configuraciones posibles, dependiendo del tipo de evento que se está enviando: una escena o un evento de persona. Para este caso usamos los datos en las posiciones 0, 2, 3, 4 y 7. Respectivamente, el id único del blob detectado, la edad o tiempo que

lleva el blob en escena, la posición en x e y normalizada del centro de masa del blob (centroide) y la profundidad normalizada del blob.

Cada Blob se guarda en un arreglo de la misma longitud que la cantidad de personas en escena que se actualiza cada vez que hay un evento de tipo /personUpdated.

```
if (theOscMessage.checkAddrPattern("/TSPS/personUpdated/")) {
        for (int i = 0; i < nBlobs; i++) {
        blobs[i] = new Blob();
        blobs[i].id = theOscMessage.get(0).intValue();
        blobs[i].age = theOscMessage.get(2).intValue();
        blobs[i].x = theOscMessage.get(3).floatValue();
        blobs[i].y = theOscMessage.get(4).floatValue();
        blobs[i].depth = theOscMessage.get(7).floatValue();
        }
  }
```

En un caso ideal, la única persona que detectaría es la de la mano, pero en caso contrario el programa necesita una forma de decidir cuál de las personas seguir. Para esto selecciona la persona con más antigüedad y menos profundidad, suponiendo que la mano es la detección más estable y más cercana a la cámara.

```
for (int i = 0; i < nBlobs; i++) {
        if (blobs[i].age >= max.age && blobs[i].depth > max.depth) {
        max = blobs[i];
        }
}
```

Como el valor que pasa TSPS del centroide ya está normalizado, para enviarlo al programa de la interfaz solo se debe realizar el mismo suavizado que en los otros programas y enviamos la posición a la interfaz.

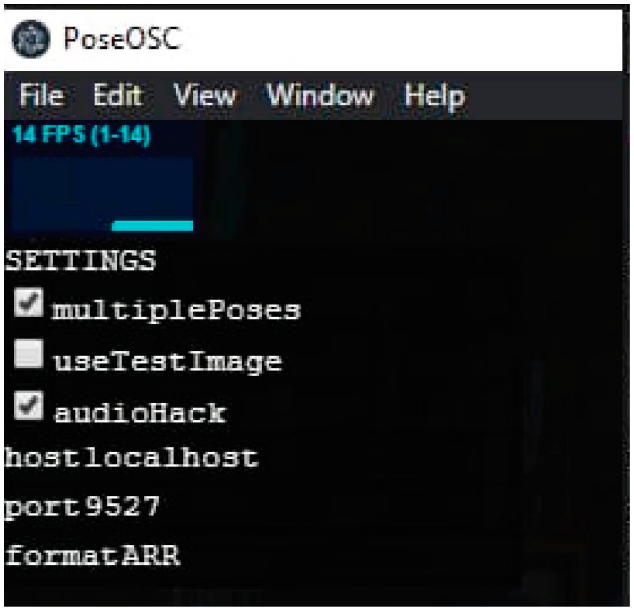
Detección con PoseNet

PoseNet puede conectarse a Processing mediante distintos programas, algunos necesitan una conexión a Internet, lo que no es deseable. Para este trabajo se usó la aplicación PoseOSC, que permite pasar la información detectada directamente a Processing por OSC y tiene la ventaja de ser una configuración simple y no necesita conexión a Internet.

Para usar PoseOSC se debe descargar el archivo de la página de Github [16] bajo la categoría Releases, descomprimirlo y abrir el archivo de nombre “pose-osc”. Al abrirlo empieza a detectar y enviar la información a un puerto automáticamente, mostrando a la izquierda de la pantalla la configuración actual del modelo. En este caso, al decir modelo nos referimos a una red neuronal entrenada para detectar esqueletos.

Figura 10

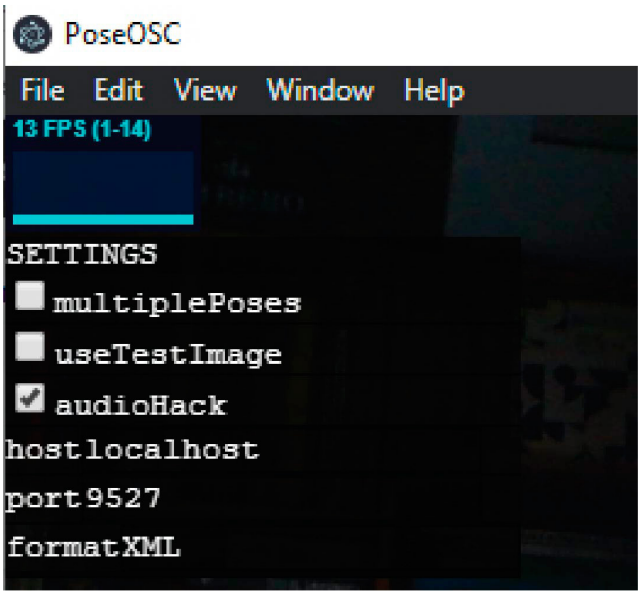
Interfaz de PoseOSC con sus ajustes por defecto



La interfaz muestra de arriba hacia abajo: los fps a los que está corriendo el modelo, el tipo de modelo que está usando, si se está usando la imágen de prueba, si se está usando el audioHack, el host (por defecto el local), el puerto al que envía los datos y el formato en el que se envían. Para este prototipo la configuración a utilizar es la siguiente: cómo solo se necesita detectar una persona a la vez, desactivamos multiplePoses; useTestImage se mantiene desactivado ya que es justamente una imagen para probar el funcionamiento del algoritmo; audioHack reproduce un sonido a bajo volumen para asegurarse que el modelo siga corriendo cuando la ventana no está activa, por lo que se mantiene activado; el host y port se mantienen iguales aunque dependiendo de las condiciones de implementación podría ser necesario cambiar el puerto o la dirección ip; y por último el format se cambia de ARR a XML (en mayúsculas). Debería quedar de esta forma:

Figura 11

Interfaz de PoseOSC con los ajustes elegidos para este prototipo



Sin embargo, estos no son los únicos parámetros que pueden configurarse; abriendo la carpeta de PoseOSC/resources/app hay un archivo llamado “settings.json”, el cual puede abrirse con el bloc de notas o cualquier editor de texto. Este archivo tiene el resto de los parámetros del modelo y si lo modificamos las opciones por defecto del programa también se modifican. De esta forma podemos asegurarnos de solo tener que configurar el programa solo una vez. Los primeros seis parámetros son los ya mencionados, que pueden ser modificados desde el archivo para marcarlos como la configuración predeterminada del programa. El resto de los parámetros se dividen en dos categorías: configuración de poseNet y configuración de cámara.

Los primeros sirven principalmente para balancear entre velocidad y precisión en la detección. Por defecto se prioriza siempre la velocidad de la detección, pero también pueden modificarse de la siguiente forma:

architecture: refiere a la arquitectura del modelo para usar, por defecto “MobileNetV1” ya que es más rápida, pero también puede cambiarse a “ResNet50” para un algoritmo más lento pero más preciso.

outputStride: refiere a la calidad de los datos de salida, por defecto 16 pero puede también setearse 8 para máxima calidad (solo si se está usando el modelo “MobileNetV1”) o 32 para mínima calidad (solo si el valor de multiplier es 1.0)

inputResolution: el tamaño de la imagen que se alimenta al modelo, por defecto 257. Mientras más grande el número, mayor calidad al costo de velocidad

multiplier: este valor se usa sólo con “MobileNetV1”. Por defecto 0.75, puede ser 0.5 para más velocidad o 1.0 para mayor calidad.

El resto de los parámetros refieren al id de la cámara (deviceId) y la dimensión (width y height). Por defecto están en null y se actualizan con la cámara que detecte al iniciar la aplicación.

Ya teniendo configurado el modelo y enviando los datos al puerto especificado, en el archivo de Processing se usan las librerías OSCp5 y OpenKinect for Processing de manera muy similar a la versión con OpenCV, con la diferencia de que además del puerto al que va a mandar la posición de la mano, se debe configurar el puerto del que va a recibir los datos de PoseOSC. Esto se hace declarando un objeto de tipo OscProperties y configurando el tamaño máximo en bytes que puede recibir por OSC (por defecto 1536), el puerto al que escucha y al que envía datos.

```
OscProperties myProperties = new OscProperties();
  myProperties.setDatagramSize(10000); // tamaño datos
  myProperties.setListeningPort(9527); // puerto del que recibe
  myProperties.setRemoteAddress("127.0.0.1", 1000); // puerto al que manda
  myRemoteLocation = new NetAddress("127.0.0.1", 5000);
  oscP5 = new OscP5(this, myProperties);
  oscP5.plug(this,"parseData","/poses/xml");
```

De la misma forma que en el programa de la interfaz se usa el método plug() para recibir los datos de OSC, el método se usa en este caso para recibir el archivo y enviarlo directamente al método parseData(), que se encarga de traducir el archivo en variables que se puedan utilizar en el programa. El archivo entra al método como un String del cual extraemos la cantidad de poses detectadas (en este caso siempre 1 o 0), las dimensiones del video y, para cada esqueleto detectado, un arreglo llamado pose que guardamos en una clase del mismo nombre. Cada objeto Pose guarda el score, que representa cuánta confianza tiene el programa de la pose detectada entre 0 y 1 siendo 1 la máxima, y un hashMap de KeyPoints. El hashMap crea esencialmente una lista en la que cada elemento se compone de una clave (forma de referenciar el dato) y un valor; en este caso la clave es un String que refiere a la parte del esqueleto correspondiente al valor, por ejemplo “rightWrist”, y el valor es un objeto de tipo Keypoint, el cual guarda un vector con las posiciones en x e y de la parte del cuerpo detectada y el score de esa parte del cuerpo. Cada objeto Pose, entonces, guarda la posición de cada parte del cuerpo detectada junto a la confianza de detección en cada caso.

```
public class Pose{
  HashMap<String,Keypoint> keypoints;
  float score;
  public Pose(){
        this.keypoints = new HashMap<String,Keypoint>();
        this.score = 0;
  }
}

public class Keypoint{
  PVector position;
  float score;
  public Keypoint(){
        this.position = new PVector(0,0);
        this.score = 0;
  }
}
```

Ya teniendo los datos correctamente organizados, la detección funciona recibiendo estos datos y preguntando primero si hay una persona en la escena (si nPoses > 0), si la hay intenta guardar el esqueleto correspondiente (en este caso siempre el primer elemento del arreglo Poses), busca la posición de la mano que necesitamos (mano derecha) según su clave y si tiene suficiente confianza en la detección actualiza su posición. El dato suavizado y normalizado se envía por OSC al programa de la interfaz.

```
void detectar(Pose[] poses, int nPoses) {
        if (nPoses > 0) {
        HashMap<String, Keypoint> keypoints;
        try {
        keypoints = poses[0].keypoints; // guarda puntos
        }
        catch(Exception e) {
        return; // evita error si no hay esqueleto guardado
        }
        if (!keypoints.containsKey(parte)) {
        return; // evita error si la parte buscada no se guardó
        }
        PVector p1 = keypoints.get(parte).position;
        float score = keypoints.get(parte).score;
        if (score >= confianza) {
        float ax = p1.x - x;
        float ay = p1.y - y;
        x += ax * easing;
        y += ay * easing;
        }
  }
```

Desempeño en diversos espacios

Los métodos de captación y reconocimiento de gestos suelen necesitar unas condiciones espaciales (lumínicas, fondo, distancia con el usuario) controladas para funcionar de forma óptima, lo que hace difícil su implementación en espacios públicos con condiciones variables [3]. En esta sección analizaremos la calidad de captación de cada método de sensado según si se puede realizar una detección continua (DC) de la mano, si la detección es entrecortada (DE), es decir que perdía la mano intermitentemente, o si directamente no detecta la mano (ND).

Para evaluar la calidad de detección se tomaron en cuenta las tres condiciones espaciales que suelen causar problemas a la hora de detectar gestos: distancia, luz y fondo. La distancia puede ser corta (30 cm), media (1 m) o alta (1,5 m); la luz puede ser poca (oscuridad), media (luz solar difusa) o mucha (luz solar directa) y el fondo puede ser simple (pared lisa, por ejemplo) o complejo (otras personas, cuadros, muebles, etc).

OpenCV

OpenCV se destaca en distancias reducidas o medias y en condiciones de media o poca luz. Al poder configurarse una distancia máxima detección y un tamaño máximo y mínimo posible de la mano, la complejidad del fondo no afecta la calidad de la detección. A partir de un metro de distancia tiene mucha dificultad para detectar la mano y en las situaciones de mucha luz la imagen de la Kinect se quema y no puede detectar la mano independientemente de fondo o distancia.

Figura 12

Cuadro comparativo OpenCV

OpenCV				
		30 cm	1 m	1,5 m
Fondo Simple	Mucha Luz	ND	ND	ND
	Media Luz	DC	DC	DE
	Poca Luz	DC	DC	DE
Fondo Complejo	Mucha Luz	ND	ND	ND
	Media Luz	DC	DC	DE
	Poca Luz	DC	DC	DE

TSPS

Como TSPS funciona mediante detección de blobs, para una detección fiable no puede realmente configurarse una vez y probar cada caso, ya que la configuración apta para detectar una mano a 1,5 metros de distancia, a 1 metro captura todo el cuerpo como un único blob. Por lo tanto, además de la prueba con configuración fija, se realizó una con la configuración del threshold ajustado para la distancia.

El mayor problema de TSPS es que debe ser recalibrado cada vez que la cámara se mueve de lugar. Dicho esto, una vez calibrado para que detecte solo la mano, no tiene problemas según el fondo ni problemas de poca luz. Se desempeña peor en distancias cortas porque es el límite de visión de Kinect y en situaciones de mucha luz se quema la imagen, impidiendo cualquier tipo de detección

Figura 13 a

Cuadro comparativo TSPS

TSPS Fijo				
		30 cm	1 m	1,5 m
Fondo Simple	Mucha Luz	ND	ND	ND
	Media Luz	ND	ND	DC
	Poca Luz	ND	ND	DC
Fondo Complejo	Mucha Luz	ND	ND	ND
	Media Luz	ND	ND	DC
	Poca Luz	ND	ND	DC

TSPS Ajustado				
		30 cm	1 m	1,5 m
Fondo Simple	Mucha Luz	ND	ND	ND
	Media Luz	DE	DC	DC
	Poca Luz	DE	DC	DC
Fondo Complejo	Mucha Luz	ND	ND	ND
	Media Luz	DE	DC	DC
	Poca Luz	DE	DC	DC

Figura 13 b

Cuadro comparativo TSPS

PoseNet

PoseNet se ve afectada en gran medida por los tres factores espaciales. La situación ideal es 1 metro aproximadamente de distancia, media luz y un fondo simple. Cuando el fondo es complejo, especialmente si en este hay elementos en tonalidades similares a las de la piel, el programa confunde las posiciones de las partes del esqueleto, desestabilizando también la posición de la mano. Por otra parte, cuando la luz es directa corre el riesgo de quemarse la imagen y perder la detección y cuando la luz es escasa, como sucede con las cámaras web, pierde detalle la imagen. El último parámetro, la distancia, también afecta la calidad de la detección, siendo más probable que pierda la mano mientras más lejos se encuentra el usuario.

Otra condición relevante en la calidad de la detección es la posición de la mano del usuario con respecto al resto del cuerpo. Para asegurarse una

PoseNet				
		30 cm	1 m	1,5 m
Fondo Simple	Mucha Luz	DC	DE	DE
	Media Luz	DC	DC	DC
	Poca Luz	DE	DE	DE
Fondo Complejo	Mucha Luz	DE	DE	DE
	Media Luz	DC	DC	DE
	Poca Luz	DE	DE	DE

Figura 14

Cuadro comparativo PoseNet

detección óptima, la mano no debe superponerse o cruzar a otra parte del cuerpo (como el torso), ya que cuando esto sucede el programa tiene problemas para identificar cada parte del esqueleto.

Conclusiones

Como denotan los resultados, cada método de sensado provee sus propias ventajas y desventajas. En cuanto a facilidad de implementación, OpenCV es la más sencilla, necesitando solo especificar la cámara y la cascada y siendo la calibración detallada totalmente opcional. El poder utilizar cascadas que no estén en la librería permite ampliar la variabilidad de elementos posibles a detectar, aunque se corre el riesgo de usar cascadas poco precisas. Por otra parte, tanto PoseNet como TSPS requieren cierto grado de calibración para funcionar correctamente, siendo TSPS el más detallado y más propenso a fallar con calibraciones poco precisas. También los datos de sensado de estos dos programas, al ser externos a Processing, necesitan ser adaptados a variables de Processing para poder ser utilizados por la interfaz.

Por otro lado, ninguno de los programas es deseable para situaciones en las que tanto la luminosidad como el fondo y la distancia del usuario sean variables, sino que es preferible intentar controlar estas variables lo más posible. De los tres programas analizados, el único que no tuvo problema con situaciones de mucha luz es el que utiliza cámara web, ya que en el caso de la Kinect la luz directa quema la imagen, eliminando todo detalle.

Revisando el desempeño de cada programa, PoseNet es la más susceptible a diferencias de fondo, ya que no pueden hacerse diferencias de profundidad y la menos afectada por diferencias de distancia; OpenCV tiene mayores problemas a mayores distancias, ya que los detalles para reconocer manos se hacen más difusos pero por otra parte es estable independientemente del fondo y la luz; y TSPS, si es calibrada para cada distancia, tiene una detección continua independientemente de luz y fondo y tiene mayor dificultad en la detección de distancias cortas.

Para finalizar, ninguna de estas tecnologías, si lo que buscamos es minimizar errores en la interacción, es realmente óptima para espacios en los que estas tres condiciones sean variables. No hay soluciones que permitan conectar la interfaz y asegurar una detección confiable sin importar las condiciones vistas a lo largo del trabajo, siempre es necesario un grado de recalibración del programa y mientras más se ajuste a las condiciones particulares de un caso, mayor es la calidad de la detección.

Referencias

1. Cortés-Rico L., Piedrahita-Solórzano, G. (2019). *Interacciones basadas en gestos: revisión crítica*. TecnoLógicas (Vol. 22, pp. 119-132). https://doi.org/10.22430/22565337.1512

2. Vuletic, T. (2019). *Systematic literature review of hand gestures used in human computer interaction interfaces*. International Journal of Human-Computer Studies. https://doi.org/10.1016/j.ijhcs.2019.03.011

3. Yasen, M., Jusoh, S. (2019). *A systematic review on hand gesture recognition techniques, challenges and applications*. PeerJ Computer Science 5:e218 https://doi.org/10.7717/peerj-cs.218

4. Viola, P., & Jones, M. (2001). *Rapid object detection using a boosted cascade of simple features*. En Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I-I). IEEE. https://doi.org/10.1109/CVPR.2001.990517

5. Wang L., Ju H. (2008). *A Robust Blob Detection and Delineation Method*. 2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing. (pp. 827-830). https://doi.org/10.1109/ETTandGRS.2008.294

6. Khan, S., Rahmani, H., Shah, S., Bennamoun, M. (2018). *A Guide to Convolutional Neural Networks for Computer Vision*. En Synthesis Lectures on Computer Vision. https://doi.org/10.2200/S00822ED-1V01Y201712COV015

7. Documentación de OSC (http://opensoundcontrol.org/index.html)

8. Processing Foundation. (2020). Processing (Versión 3.5.4) [Aplicación] Descargado de: https://processing.org/download/

9. atduskgreg. (2017). OpenCV for Processing (Versión 0.5.4) [Librería de Processing]. Descargado de: https://github.com/atduskgreg/opencv-processing/releases

10. LAB at Rockwell, IDEO Labs. (2017). TSPS (Versión 1.3.6) [Aplicación]. Descargado de: https://www.tsps.cc/

11. Oved, D. (2018) *Real-time human pose estimation in the browser with tensorflow. js*. TensorFlow Medium. Disponible en: https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5

12. Jana, A. (2012). *Kinect for windows SDK programming guide*. Packt Publishing Ltd.

13. Batard, P. (2020). Zadig (Versión 2.5) [Aplicación]. Descargado de: https://zadig.akeo.ie/

14. Runway AI. (2020). Runway (Versión 0.17.7) [Aplicación]. Descargado en: https://runwayml.com/

15. Schlegel, A. (2011). oscP5 (Versión 0.9.8) [Librería de Processing] Descargado en: http://www.sojamo.de/libraries/oscP5/

16. LingDong. (2020). PoseOSC (Versión 0.0.3) [Aplicación]. Descargado de: https://github.com/LingDong-/PoseOSC/releases/tag/0.0.3

Notas

1. Proyecto disponible en: https://github.com/emmelab/interfacesGestuales