UNIVERSIDAD DE GRANADA

Nuevos Paradigmas de Interacción

grado en ingeniería informática

Práctica Gestos

Memoria Técnica

Autores

Vladislav Nikolov Vasilev

José María Sánchez Guerrero

Fernando Vallecillos Ruiz

Rama

Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

1. Introducción

En este proyecto vamos a explicar nuestra *Natural User Interface (NUI)* que hará que nuestra visita a la Alhambra sea más dinámica y productiva. El proyecto constaría de varios dispositivos, como pueden ser principalmente unas gafas de realidad aumentada, un dispositivo *Leap Motion* y un micrófono integrados en las gafas, y un *smartphone* que utilizaremos tanto para manejar el sistema gracias a los múltiples sensores que incorpora como para controlar la aplicación.

En esta segunda versión del proyecto vamos a encargarnos de la interfaz por gestos, es decir, dejaremos a un lado los sensores y el controlador por voz. Vamos a encargarnos de la realidad aumentada y los gestos para manejar el sistema.

2. Interfaz por gestos

Como no disponemos de unas gafas con un Leap Motion incorporado, vamos a realizar nuestra simulación en un programa de `Python` aparte. En nuestra versión, la opción de utilizarlo verticalmente no está disponible, por lo que su funcionamiento se mostrará con el dispositivo sobre la mesa. Se han implementedo los siguientes gestos: *click*, *swipe*, zoom y agarre y movimiento. Un poco más adelante vamos a describir en qué consiste cada gesto y diremos muy brevemente cómo se ha implementado cada uno de ellos.

3. Información y estructuras de datos utilizadas

En esta sección se describirán las especificaciones de nuestra clase principal. Así como diferentes estructuras utilizadas en nuestro código para hacer posible una pequeña interfaz gráfica. Esta permitirá al usuario ver de una forma visual los resultados de nuestro proyecto.

3.1. Clase Leap

Nuestra clase `LeapMotionListener`, se trata de nuestra única clase con los métodos necesarios para poder controlar la información recibida a través del dispositivo.

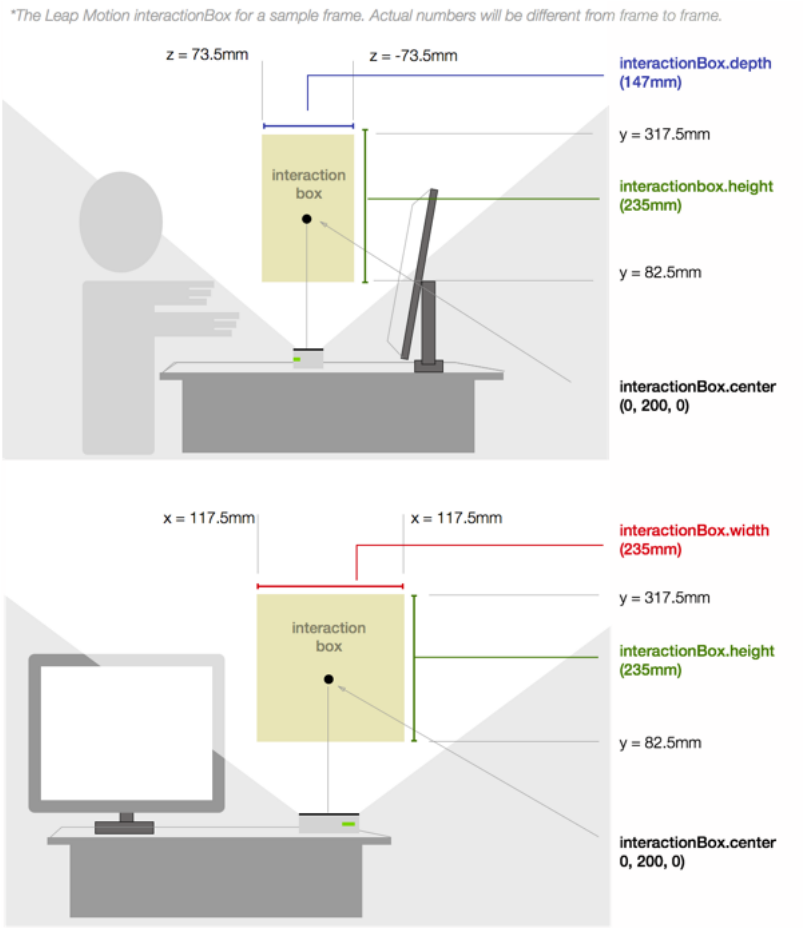
Esta clase consta de cinco métodos los cuales se explican por sí mismos. Sin embargo, se podrían destacar dos métodos:

- `on_connect`: este método es llamado cuando el dispositivo ha sido conectado. La

relevancia de este método se encuentra en que permite el reconocimiento de los diferentes gestos implementados en la clase. A pesar de que solo utilizaremos uno de estos gestos en nuestro proyecto, hemos encontrado que el habilitarlos todos produce una reducción de la malinterpretación de estos.

- `on_frame`: Este método será llamado por cada frame que recibamos desde el dispositivo. Se podría considerar como nuestro método principal. Ya que el número de frames por segundo fluctúa dependiendo de la capacidad de la máquina, decidimos, por el bien de la eficiencia, intentar reducir en número de llamadas a funciones al mínimo.

Además hemos hecho uso del objeto `InteractionBox`. Este objeto representa una región en forma caja en el campo de vista del `LeapMotion`. Esta región es usada para mapear la posición de las manos en el área. Si bien parece una opción fácil de usar, existe el problema del tamaño de esta región. Se puede apreciar en la siguiente imágen:



Como se puede apreciar, la región es apenas $15 \times 20 \times 20$ aunque la visión del dis-

positivo sea bastante mayor. Existe una solución simple para "aumentar" la región de interacción. Esta solución hace uso de la capacidad de modificar las coordenadas antes de normalizarlas con respecto a la caja. Si dividimos las coordenadas previas a la normalización entre dos, resulta en un efecto de "despliegue" de la caja original, al doble de su tamaño. Hemos realizado esta modificación al Eje-x y Eje-z. A pesar de que contradecimos el objetivo del "mapeo perfecto" de la clase `Interaction Box`, esta solución ha sido propuesta por desarrolladores profesionales de *LeapMotion*.

3.2. Interfaz Gráfica

A diferencia de nuevas versiones de *LeapMotion*, los kit de desarrollo en `Python` no tienen una interfaz gráfica. Esto nos dejaba dos opciones posibles:

- Mostrar resultados por la terminal a través del método `on_frame`.
- Crear nuestra propia interfaz gráfica para mostrar los resultados.

Nos decidimos por la segunda opción y gracias a la biblioteca `tkinter`, desarrollamos una forma de visualización de nuestro proyecto. De una forma simplificada, `tkinter` es una biblioteca que permite creación de ventanas donde mostrar texto y/o imágenes. Haciendo uso de variables globales y funciones recursivas en intervalos, hemos conseguido una forma simple y eficiente de representar cambios a tiempo real. Como ha sido mencionado antes, la necesidad por un método lo más óptimo posible que interfiriera lo mínimo con la velocidad de procesamiento de frames del dispositivo era prioridad.

Para este objetivo hacemos uso de:

- Variables globales: la declaración de variables globales, en contraste con variables locales o copia de variables en métodos, permite incrementar la agilidad del programa al acceder a ellas.
- Métodos de actualización de la ventana independiente de la frecuencia del dispositivo: los límites de las máquinas se hacían muy presentes en este punto. La frecuencia del dispositivo es muy alta (y variable) y la capacidad de actualización de la ventana en comparación, era demasiado lenta. Si tuviésemos los recursos suficientes, sería posible un actualización completamente a tiempo real solo dependiendo de los frames, sin embargo nuestra versión crea una ilusión bastante cerca al tiempo real (cada 50 ms). Esto se traduce a la actualización de valores en el método `on_frame`, y el posterior uso de estos en nuestro método recursivo.

4. Descripción de los gestos

En esta sección se van a describir todos los gestos que se han implementado. Se va a hacer una breve descripción de en qué consiste y se va a mostrar muy por encima el código asociado en los casos en los que se haya considerado necesario (cuando lo que se quiere mostrar es relevante ya que se trata de algo no trivial).

Antes de continuar, hay que destacar que algunos de los gestos solo están disponibles cuando ambas manos están encima del *Leap Motion*. Por ejemplo, el *click*, el *swipe* y el agarre y movimiento solo pueden ser realizados cuando se detecta una única mano. El zoom solo se puede realizar si se han detectado dos manos y se hace el gesto correspondiente con los dedos. Para comprobar si se usa una mano o dos, podemos hacer los siguiente:

```
if(len(frame.hands) == 1):
    ...

f(len(frame.hands) >= 2 ):
    ...
```

En este caso, comprobamos si se han detectado dos o más manos, ya que puede pasar que otra persona pase la mano por encima del dispositivo. Sin embargo, nos quedamos siempre con las dos primeras manos que hayamos detectado.

4.1. Descripción del *click*

Este gesto sirve para mostrar información extra sobre el objeto. El gesto consiste en cerrar y abrir el puño en la zona sobre la que queremos obtener información. Esta zona se detectaría automáticamente mediante un algoritmo implementado en las gafas, pero como no disponemos de dicha tecnología, utilizaremos una imagen fija con una zona resaltada.

Esto mostrará en la interfaz de nuestras gafas de realidad aumentada una imagen de la zona resaltada, junto con información de esta. La información podrá ser modificada mediante otros gestos descritos posteriormente.

Para saber si estamos haciendo o no un *click*, simplemente tenemos que comprobar la siguiente condición:

```
if (closed_hand and hands[0].grab_strength == 1.0):
    ...
```

```
    if !(cursor_x != -100 and cursor_y != -100 and label.place_info() != {}):
        # If photo is not loaded, we check the position of the cursor and load the photo
        if (cursor_x >= 1100.0 * .75) and (cursor_y <= 1000.0 * .2):
            resetting = True
```

Después se va a comprobar si la mano está posicionada en la zona adecuada, en cuyo caso se mostrará la imagen que contiene información. La zona se debería detectar con técnicas más avanzadas, las cuales no disponemos, asi que nosotros detectaremos la esquina superior derecha de la imagen donde se encuentra el palacio de Carlos V.

4.2. Descripción del *swipe*

Al igual que mostramos la información con el gesto anterior, necesitaremos otro gesto para ocultarla. Nosotros utilizamos el *swipe*, el cual detectará los movimientos en el eje X. De esta formá, podrá ser usado tanto por personas zurdas como diestras sin ningún tipo de problema. Modificaremos el parámetro de su velocidad para que no se pueda confundir con otros gestos que hacemos naturalmente.

La implementación es la siguiente:

```
if (len(frame.gestures()) != 0):
    # Check if the hand is completely open and the gesture is swiping
    if open_hand and (frame.gestures()[0].type == Leap.Gesture.TYPE_SWIPE):
        swipe = SwipeGesture(frame.gestures()[0])
        # Check that the main direction of the swipe is the x-axis (left or right)
        if(abs (swipe.direction[0]) > 0.7):
            swiping = True
```

Simplemente estamos comprobando si se ha detectado algún gesto y si ese gesto se ha realizado con la mano abierta y es un *swipe* en cualquier sentido en el eje X.

Como se puede ver, estamos utilizando un gesto que ya viene en *Leap*, aunque lo estamos adaptando a nuestras necesidades, ya que hemos limitado su funcionamiento al eje X.

4.3. Descripción del zoom

Este gesto sirve para disminuir o aumentar el tamaño de la información mostrada. Para utilizar este gesto tendremos que tener las dos manos sobre la zona de detección

del *Leap Motion* y, posteriormente, extender los dedos pulgar e índice con una apertura mínima entre ellos de 40º. El zoom que se le aplicará a la imagen será directamente proporcional a la distancia en el eje X entre la mano derecha y la mano izquierda. La implementación de dicho gesto es la siguiente:

```
# Let's check if both hands have only the thumb and index extended
two_up_left = True
two_up_right = True

for i in range(0, 2):
    two_up_left = two_up_left and left_hand.fingers[i].is_extended
    two_up_right = two_up_right and right_hand.fingers[i].is_extended

for i in range(2, 5):
    two_up_left = two_up_left and not left_hand.fingers[i].is_extended
    two_up_right = two_up_right and not right_hand.fingers[i].is_extended

# Set the minimum angle between thumb and index to be considered zooming
threshold_angle = 40

# If they have the left index and thumb extended, we calculate the angle
if two_up_left:
    vec1 = left_hand.fingers[0].direction    # Thumb
    vec2 = left_hand.fingers[1].direction    # Index
    angle1 = acos(max(-1.0, min(1.0, vec1.dot(vec2)))) * Leap.RAD_TO_DEG

    # Check if the angle is open enough
    if (angle1 > threshold_angle):
            left_hand_zoom = True
    else:
            left_hand_zoom = False
else:
    left_hand_zoom = False

# If they have the right index and thumb extended, we calculate the angle
if two_up_right:
    vec1 = right_hand.fingers[0].direction    # Thumb
    vec2 = right_hand.fingers[1].direction    # Index
    angle2 = acos(max(-1.0, min(1.0, vec1.dot(vec2)))) * Leap.RAD_TO_DEG

    # Check if the angle is open enough
    if (angle2 > threshold_angle):
            right_hand_zoom = True
    else:
        right_hand_zoom = False
else:
    right_hand_zoom = False
```

Básicamente, lo que se hace es comprobar si los dedos índice y pulgar de las dos manos están extendidos y el resto cerrados. Después, se comprueba si en cada caso el ángulo

que forman es mayor que 40º, en caso de que la condición anterior se haya satisfecho.

Además de hacer zoom, también está implementada una función en la que, si llegamos a un umbral de zoom en algunas zonas determinadas (por ejemplo, a la fuente de la funete de los leones, o una habitación del Palacio de Carlos V), mostraremos información más detallada sobre ésta.Este tipo de funcionalidades han sido implementadas en la función de actualización de ventana de nuestra interfaz gráfica.

4.4. Descripción del agarre y movimiento

Para mover la información sobre toda la superficie de las gafas de realidad aumentada, utilizaremos el siguiente gesto. Cerraremos el puño (tendrá que ser en las mismas coordenadas sobre las que está la información, para así evitar moverla sin querer con algún gesto natural) y posteriormente la moveremos al lugar que queramos. Para dejar de moverla, simplemente volvemos a abrir la mano.

Si estamos arrastrando la información fuera de nuestro campo de visión, hemos puesto unos límites tanto vertical como horizontalmente, para que el programa no nos permita hacerlo.

Arrastrada la información, también guardaremos las propiedades que conservaba previamente, como por ejemplo, si le habíamos hecho zoom, o la habíamos arrastrado en otro momento. Lo hemos implementado de la siguiente forma:

```
    # If the hand is close
    if (closed_hand and hands[0].grab_strength == 1.0):

        # Check if we were positioning already or we are starting
        if(not positioning):
            # Check if the cursor is positioned over the photo
            inside_range_x = image_pos_X - new_width/2 +25 <= cursor_x <= image_pos_X + new_width/2 -25
            inside_range_y = image_pos_Y - new_height/2 +25 <= cursor_y <= image_pos_Y + new_height/2 -25

        # Check is not an abnormal case and that the photo is loaded
        if(cursor_x != -100 and cursor_y != -100
            and label.place_info() != {}):

            # Check if we can position or were already
            if(positioning or (inside_range_x and inside_range_y)):

                # If we are starting to position
                if hand_origin_X == -100 and hand_origin_Y == -100:
                    hand_origin_X = cursor_x
                    hand_origin_Y = cursor_y
                    positioning = True

```

```
23                  # If we were positioning already
24                  else:
25                      image_pos_X = image_pos_X + ((cursor_x -
    hand_origin_X) if cursor_x - hand_origin_X < 15 else 15)
26                      image_pos_Y = image_pos_Y + ((cursor_y -
    hand_origin_Y) if cursor_y - hand_origin_Y < 15 else 15)
27                      hand_origin_X = cursor_x
28                      hand_origin_Y = cursor_y
```

Para comprobar que estamos haciendo un agarre con la mano, tenemos que comprobar la misma condición que con el *click*, aunque lo que se va a ejecutar va a ser diferente, ya que se va a actualizar la posición de la imagen en función del movimiento realizado, respetando siempre los bordes que se hayan impuesto.

Referencias

[1] Manual SDK en Python
https://developer-archive.leapmotion.com/documentation/python/index.html

[2] *M*anual SDK en C# (consejos sobre el mapeo del sistema de coordenadas)
https://developer-archive.leapmotion.com/documentation/csharp/index.html

[3] *F*oro de desarrollo en LeapMotion
https://forums.leapmotion.com/

[4] *V*ideotutoriales Leap Motion Python
https://www.youtube.com/playlist?list=PLgTGpidiW0iTELuljcIdTkA5SjHa5tudP