Universitat Autònoma de Barcelona

escola d'enginyeria

3595: Eye Tracking for On-Screen Reading

Memoria del proyecto de
Ingeniería en Informática
realizado por
Bibiana do Canto Angonese
y dirigido por
Dimosthenis Karatzas
y
Oriol Ramos Terrades

Bellaterra, 22 de junio de 2011

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, D. KARATZAS i O. Ramos

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat: .....

Bellaterra, 17 de junio de 2011

Índice

1. Introducción

El Eye Tracker, un dispositivo que monitoriza los ojos de un usuario, tiene varios usos. Este proyecto se basa en hacer un estudio sobre uno de ellos: ayudar en la lectura de textos en pantalla. La motivación principal de este proyecto era la posibilidad de facilitar la transcripción de documentos escritos a mano con asistencia humana, utilizando el seguimiento de ojos y el reconocimiento de voz. Sin embargo, la idea fue desviada a una dirección distinta, haciendo que surgieran varias preguntas. ¿Cómo actúan nuestros ojos cuando leemos? ¿Sería posible saber al momento cuándo una palabra es difícil para una persona, sólo con la mirada de esta persona? ¿Se puede distinguir entre pasar la vista por encima de un texto y leerlo? En este proyecto hemos intentado dar una respuesta a estas preguntas mediante dos experimentos.

1.1. Objetivos

El objetivo de este proyecto es analizar las acciones del usuario al leer un documento. En concreto, consiste en analizar el comportamiento de los ojos al leer una palabra difícil, y también poder diferenciar cuándo el usuario está leyendo o simplemente pasando la vista por encima del texto. Para poder llevar a cabo este proyecto, se ha utilizado un Eye Tracker, que detecta y registra el movimiento de los ojos, devolviendo el punto de la pantalla donde está mirando el usuario y el *timestamp* (valor que representa una fecha y/u hora única).

El proyecto está dividido en diferentes objetivos:

1. Crear una interfície gráfica que muestre la imagen de un documento y que permita visualizar metadatos relevantes. Como por ejemplo, encuadrar la posición de cada palabra de un texto cualquiera del que se disponiera de ground truth.

2. Mejorar dicha aplicación para poder controlar el Eye Tracker remotamente y así poder ejecutar ciertos experimentos:

   - Incluir una capa de comunicación con el Eye Tracker.
   - Implementar la interfaz necesaria para controlar el Eye Tracker remotamente y poder grabar datos.

- Mejorar la interfaz para poder visualizar la posición de los ojos al momento.
- Mejorar la interfaz para poder visualizar datos que han sido previamente grabados.

3. Procesar los datos grabados para quitar posible ruído, y también extraer eventos clave (fijaciones y sacadas).

4. Clasificar palabras según su dificultad, usando características conseguidas de los movimientos de los ojos de los usuarios mientras las leían.

5. Clasificar las sacadas en diferentes patrones de lectura (leer y pasar la vista por encima) basados en heurísticas halladas en [1].

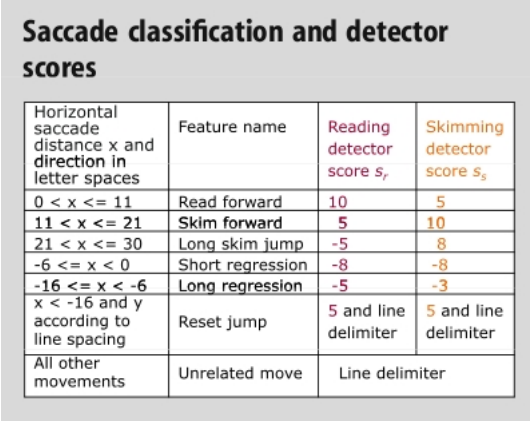
1.2. Estado del arte

Antes que nada, empecemos viendo algunas investigaciones que se han llevado a cabo con el Eye Tracker.

Una de las maneras en las que se usa el Eye Tracker es para ayudar a las personas con discapacidades físicas y mejorar su calidad de vida y también la de su familia. Gracias a ello, usando los ojos pueden controlar el ordenador para escribir, navegar por internet, jugar, y, más importante aún, comunicarse con el mundo exterior [2]. En Boston University se estudió la posibilidad de usar los ojos como si fueran el ratón del ordenador, resultando en que si una persona es capaz de mover los ojos y parpadear, tendría una nueva forma de comunicación [3].

Para alcanzar uno de los objetivos marcados en este proyecto, implementaremos la idea usada en el Text 2.0, en el que utilizan el Eye Tracker para mejorar la experiencia de lectura de los usuarios [4]. Mientras el usuario lee un texto, las imágenes van cambiando dependiendo de la frase que esté leyendo el usuario, y también se pueden escuchar sonidos y efectos especiales. La aplicación también detecta cuándo el usuario está leyendo, o sólo mirando por encima el texto (en este caso las partes más importantes del texto son resaltadas), y sabe cuándo el usuario ha perdido el punto donde estaba leyendo. Sin embargo, Text 2.0 aún está en fase de desarrollo, y de momento sólo es una idea de cómo será leer en el futuro. Uno de los métodos que

se han utilizado en este proyecto ha sido detectar las fijaciones (los puntos en los que se fija el lector) y las sacadas (movimientos rápidos del ojo) del usuario. Analizando estos datos, por ejemplo, se puede estimar si el usuario está leyendo o pasando la mirada por encima del texto. En la siguiente tabla (Figura 1) se especifica la heurística que utilizan para distinguir entre las dos acciones:



Saccade classification and detector scores

Horizontal saccade distance x and direction in letter spaces	Feature name	Reading detector score s_r	Skimming detector score s_s
0 < x <= 11	Read forward	10	5
11 < x <= 21	Skim forward	5	10
21 < x <= 30	Long skim jump	-5	8
-6 <= x < 0	Short regression	-8	-8
-16 <= x < -6	Long regression	-5	-3
x < -16 and y according to line spacing	Reset jump	5 and line delimiter	5 and line delimiter
All other movements	Unrelated move	Line delimiter	

Figura 1: Clasificación de sacadas [1]

Esta tabla depende de la distancia y dirección de las sacadas horizontales, y se utiliza para sumar o restar puntos al hecho de estar leyendo o sólo pasar por encima la mirada del texto. La acción que tenga más puntos, será la escogida [1]. En este proyecto, hemos utilizado estas heurísticas en uno de los experimentos.

1.3. Planificación del proyecto

En el siguiente diagrama de Gantt (Figura 2) podemos ver cómo ha sido repartido el trabajo desde Noviembre, teniendo en cuenta que la mayor parte del trabajo fue efectuada en el segundo semestre.

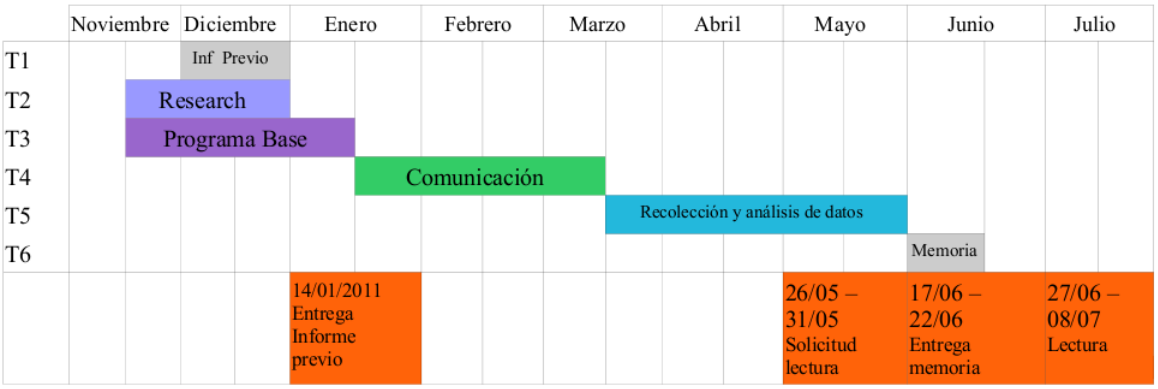


Figura 2: Diagrama de Gantt del proyecto

A continuación podemos ver la descripción de las tareas del proyecto:

- T1 - Informe Previo: Consistió en redactar el Informe Previo del proyecto
- T2 - Research: Se buscó información relevante al proyecto, tanto para escribir la sección del Estado del Arte del informe, como para aprender a desarrollar una interfaz de usuario en Java.
- T3 - Programa base: Se desarrolló una interfaz gráfica que mostrase la imagen de un documento, y que leyera el ground truth correspondiente con las coordenadas de cada palabra y línea de dicho documento.
- T4 - Comunicación: Se amplió la interfaz gráfica para que fuera capaz de enviar datos al Eye Tracker y recibir datos. Al final de esta tarea, fue posible marcar en la pantalla el punto donde el usuario estaba mirando en el momento.
- T5 - Recolección y análisis de datos: Se amplió la interfaz gráfica para que se pudiera grabar datos y efectuar experimentos. Después, se pudo visualizar los datos ya guardados, y analizar estos datos para la clasificación.
- T6 - Memoria: Se escribió la memoria del proyecto.

2. Fundamentos teóricos

En los siguientes apartados serán explicados los fundamentos teóricos necesarios para poder entender el funcionamiento de este proyecto. Primero veremos cuáles son los tipos de movimientos que pueden hacer los ojos, y después veremos qué es un Eye Tracker, cómo funciona, y cuáles son los pasos necesarios para recolectar datos.

2.1. Movimientos del ojo

Cuando estamos despiertos, los ojos se mueven rápidamente con una frecuencia de aproximadamente cuatro veces por segundo, para poder enfocar en las diferentes partes del entorno visual. Ya que los movimientos del ojo están asociados con el sistema cognitivo del cerebro, podemos analizar los movimientos oculares para derivar información de los procesos cognitivos del cerebro [1]. A continuación podemos ver los movimientos oculares más destacados:

2.1.1. Fijaciones

Una fijación es la acción de mantener la mirada en un mismo sitio por durante cierto tiempo. El hecho de leer algo implica que hayan fijaciones en lugares sucesivos a través de la página o pantalla que se esté leyendo. Pero el ojo no está nunca completamente quieto durante una fijación: movimientos fijacionales tienen lugar [6]. Estos movimientos fijacionales se dividen en tres categorías:

- Microsacadas: Estos movimientos involuntarios son pequeños movimientos rápidos, como “tirones”. De hecho, son pequeñas versiones de las sacadas (explicadas más adelante) [7].
- Derivas oculares: Estos movimientos son movimientos que ocurren al azar cuando el sujeto se fija en un objeto. Las derivas oculares son necesarias para evitar que la imagen visual desaparezca [8].
- Microtemblores oculares: Son temblores del ojo que son constantes, fisiológicos, con una frecuencia alta, y amplitud baja. Son debidos a la actividad constante de las unidades oculomotoras del tronco cerebral [9].

La función de estos tres movimientos involutarios es evitar los efectos de saturación en los receptores visuales en la retina que causarían desvanecimiento de la percepción [1]. Esto lo llevan a cabo estimulando las neuronas de las áreas visuales del cerebro [6]. Para este proyecto, no se han tenido en cuenta estos tres movimientos fijacionales, ya que, al ser demasiado rápidos, son indetectables por el Eye Tracker.

Frecuentemente, las fijaciones duran de 200 a 350 milisegundos, y son el parámetro ocular más utilizado por los psicólogos de la percepción y atención, así como en la ciencia visual y en la neurociencia [10]. En este proyecto, hemos analizado las fijaciones hechas por los usuarios en el primer experimento efectuado.

2.1.2. Sacadas

Las sacadas son movimientos voluntarios, aunque también pueden ser involuntarios en algunas ocasiones, como los realizados en la fase REM de sueño. Nos permiten visualizar las diversas zonas de una escena. Con ellos dirigimos la mirada a varias sitios de nuestros alrededores y se nos hace más fácil la recogida de información. Su objetivo principal es de colocar la imagen visual en la fóvea, es decir, la región de la retina que dispone de mayor agudeza visual (Figura 3).

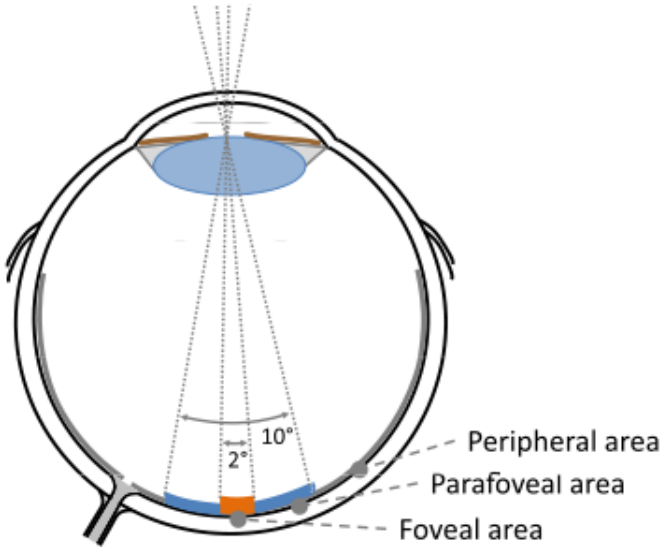


Figura 3: Regiones foveales, parafoveales y perferales de la retina [1]

La característica más distintiva de una sacada es la relación entre el tamaño del movimiento (en grados) y la velocidad punta (grados por segundo). Por ejemplo, si una sacada es de un tamaño de 80º de ángulo visual, la velocidad de la sacada puede alcanzar a ser de 700 grados por segundo [10]. Otra manera de describir una sacada, es decir que es el movimiento ocular entre dos fijaciones [11]. Una de las diferencias entre fijaciones y sacadas, es que en las fijaciones se lleva a cabo un proceso cognitivo en nuestro cerebro, mientras que en las sacadas no se transmite ningún tipo de información [12]. En este proyecto hemos usado la distancia recorrida en una sacada para diferenciar el modo de lectura de los usuarios entre "leer" y "pasar la vista por encima".

2.2. Eye Tracker

Eye tracking es la ciencia de medir el movimiento de los ojos, normalmente en respuesta a estímulos visuales, auditorios o cognitivos. Existen varios métodos de eye tracking [13]:

- Oculografía eléctrica (EOG): Estos sistemas de eye tracking originales siguen los cambios en campos magnéticos cuando los ojos se mueven, ya que hay diferencias entre las polaridades del ojo de atrás hacia delante. Normalmente, electrodos son colocados encima y debajo de los ojos, o en los lados de los ojos (Figura 4).

- Sistemas de bobina: Estos métodos siguen los movimientos del ojo observando una bobina magnética insertada en el ojo, sea cirúrgicamente o como parte de una lente de contacto. La cabeza del usuario debe estar sujeta, u otra bobina debe ser usada para analizar la posición de la cabeza. Este tipo de eye tracking es invasivo y peligroso, por lo que es usado mayoritariamente en estudios de animales.

- Sistemas Purkinje duales: Estos sistemas siguen reflejos de luz en las partes delantera y trasera de la córnea. Calculando geométricamente la orientación de estos reflejos, se puede determinar la posición del ojo. Normalmente este método es muy preciso, pero requiere tener la cabeza inmobilizada tal como se muestra en la Figura 5.

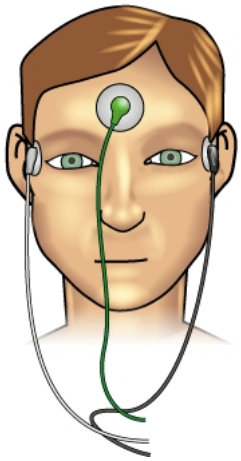


Figura 4: Oculografía eléctrica [15]

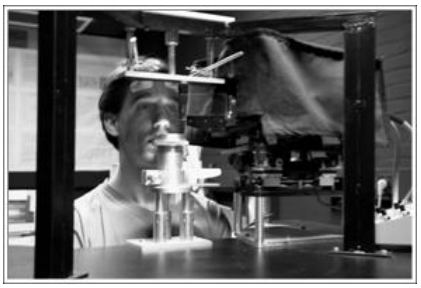


Figura 5: Sistema Purkinje dual [16]

- Sistemas de pupila claros: Si enviamos infrarojos directamente al ojo, se produce un efecto brillante en la córnea. Al seguir los movimientos de este reflejo claro, estos sistemas siguen movimientos del ojo orbitales. Usando un algoritmo calibrado, el sistema puede traducir estos movimientos del ojo a posiciones de la mirada. Este método requiere un método externo de seguimiento de la cabeza, o que la cabeza esté inmobilizada.

- Sistemas de pupila oscuros: Al igual que con el método anterior, estos sistemas iluminan el ojo con una luz infraroja desde una cámara sensitiva a los infrarojos. El ojo y la cara reflejan esta iluminación, pero la pupila absorbe la mayoría de los infrarojos y aparecerá como una elipse oscura con mucho contraste. Se determina el centro de la pupila con un software de análisis de imagen, y ésta es puesta en correspondencia con la posición de mirada usando un algoritmo de eye tracking. Estos sistemas son versátiles y más faciles de configurar que los demás, pero también requieren algún tipo de compensación para los movimientos de cabeza. Para este proyecto, hemos usado un eye tracker que utiliza este sistema.

Para este proyecto se ha usado un Eye Tracker SMI RED. Su nombre viene de Remote Eyetracking Device, y está desarrollado para poder medir los movimientos del ojo sin ningún tipo de contacto con el usuario, y siendo capaz de compensar los movimientos de la cabeza. En particular, este Eye Tracker coge muestras a una frecuencia de 50

a 60 Hz y tiene en cuenta los dos ojos. Para saber más detalles técnicos de este Eye Tracker, se puede consultar la documentación en [17].

2.2.1. Funcionamiento

El módulo RED (Figura 6) utiliza LEDs infrarojos para iluminar los ojos. Los ojos reflejan estos rayos, y el Eye Tracker calcula el ángulo entre los infrarojos enviados y los recibidos. Para que funcione correctamente la persona debe sentarse a unos 60-80 cm. del Eye Tracker. Este Eye Tracker está integrado con una pantalla de 22 pulgadas (ver Figura 7) y un ordenador llamado el *Stimulus PC*. Este ordenador se encarga de mostrar el *estímulo* por la pantalla. Es decir, aquello que la persona mirará a la hora de seguir su mirada y recibir los datos del Eye Tracker. Es posible trabajar con sólo este ordenador, ya que tiene un programa llamado iViewX, que se encarga de hacer la calibración y grabar los datos. Sin embargo, en este proyecto hemos usado un ordenador más, que se encargaba de mostrar el estímulo por la pantalla, envíar comandos al programa iViewX y recibir datos vía una conexión Ethernet.

Figura 6: Eye Tracker RED [13]

Figura 7: Monitor estímulo [13]

2.2.2. Calibración

Para seguir el movimiento de los ojos y querer mostrar los puntos a los que miramos, es necesario saber antes cuál es la correspondencia entre el ángulo que forman los rayos infrarojos al reflejarse en nuestros ojos, y las coordenadas de la pantalla estímulo. Ésta es la función de la calibración. En una calibración, se muestran por la pantalla estímulo varios puntos (uno a uno) a los que el usuario debe mirar. iViewX permite hacer calibraciones de 2, 5, 9 y 13 puntos, y de cuantos más puntos sea la calibración, más precisa será ésta. Las coordenadas de estos puntos ya han sido definidas antes de hacer la calibración, así como la resolución de la pantalla estímulo. Después de la calibración, cuando el usuario mire a la pantalla, el Eye Tracker sabrá a qué coordenadas se está mirando.

2.2.3. Data streaming

Como se ha dicho antes, el Eye Tracker envía datos al ordenador que ha dado los comandos. Estos datos y comandos son enviados mediante sockets UDP. Los datos que recibimos del eye tracker pueden tener los siguientes campos:

- TS: timestamp en milisegundos
- TU: timestamp en microsegundos
- DX, DY: diámetro de la pupila (horizontal y vertical)
- PX, PY: posición de la pupila
- CX, CY: posición del reflejo corneal
- SX, SY: posición de la mirada en la pantalla (coordenadas x e y)
- SC: contador de escena
- ET: información sobre el ojo grabado. Posibles valores son izquierdo, derecho o binocular.

Antes de solicitar que el Eye Tracker nos empiece a enviar datos, le podemos indicar cómo queremos que represente la información: podemos elegir cuáles de los campos anteriores nos interesa, y sólo recibir éstos.

3. Desarrollo

A continuación se explican el desarrollo de las diferentes fases del proyecto, con sus respectivos diagramas de clases UML.

3.1. Aplicación

La aplicación desarrollada ha sido programada en Java, usando el entorno de programación Eclipse. Para crear la interfaz de usuario, se ha utilizado el paquete java.awt (Abstract Windows Toolkit), que contiene todas las clases para crear interfaces y pintar gráficos e imágenes. Se ha elegido este paquete por su simplicidad y estabilidad. Otros paquetes como el Java Swing tenían una complejidad y objetos innecesarios para este proyecto. La información básica para crear la interfaz gráfica fue encontrada en [18]. La aplicación debía tener las siguientes opciones:

1. Ser una interfaz gráfica de usuario, con una barra de menús en la que se pudieran escoger diferentes acciones.

2. Abrir y mostrar una imagen en formato jpg y poder mostrarla en pantalla completa.

3. Teniendo en cuenta que cada imagen que mostrase un texto tenía asociada un fichero de texto con las coordenadas de las líneas y/o palabras del texto, poder abrir este fichero y encuadrar las líneas y/o palabras del texto de la imagen. Esto fue útil después para identificar qué palabra estaba leyendo el usuario exactamente.

Tal como podemos ver en el siguiente diagrama de clases UML[1] (Figura 8), la primera fase de este proyecto contiene seis clases principales, de las cuales, cabe

[1]Este diagrama y los siguientes sólo muestran las clases y métodos importantes.

destacar:

MenuEvents: Esta clase procesa todos los eventos de menú recibidos.

ImageCanvas: Esta clase muestra una imagen, y es la que realiza todas las acciones que tengan que ver con la imagen: encuadrar las palabras/líneas del texto, aplicar el zoom, enseñar la imagen en modo pantalla completa.

Las clases **MainApp** y **Window** crean la ventana principal de la aplicación, y las clases **Words** y **Lines** son clases auxiliares necesarias para leer los archivos de texto que contienen las coordenadas de las palabras o líneas y guardar la información para poder usarla.

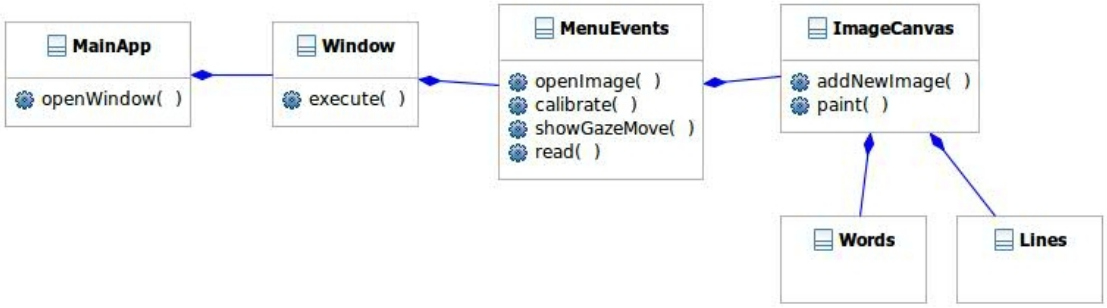


Figura 8: Diagrama de clases UML de la primera fase

Podemos ver la aplicación resultante de esta fase en las Figuras 9 y 10.

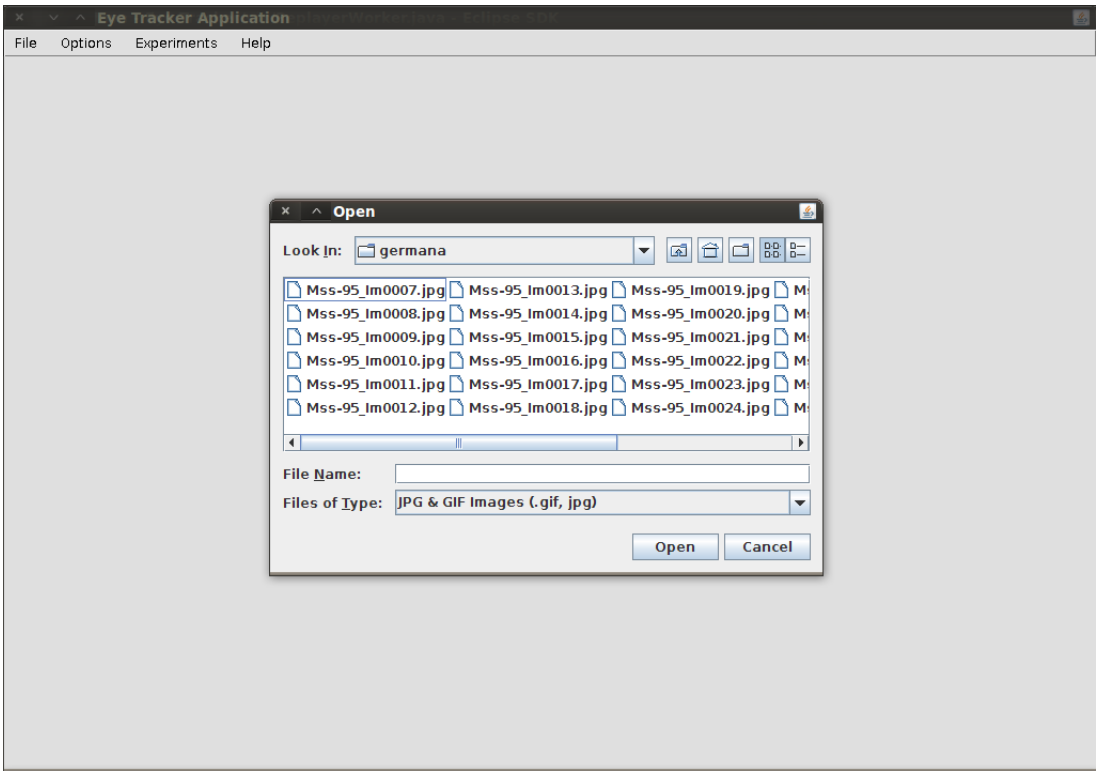


Figura 9: Funcionalidad de elegir una imagen

Para esta parte se han usado imágenes de documentos antíguos, como se puede ver en la Figura 10. Sin embargo, para la parte experimental del proyecto se han utilizado textos escritos por ordenador, ya que los documentos antíguos no contenían todo el *ground truth* de las coodenadas de cada palabra, y la letra de éstos resultaba difícil de entender.

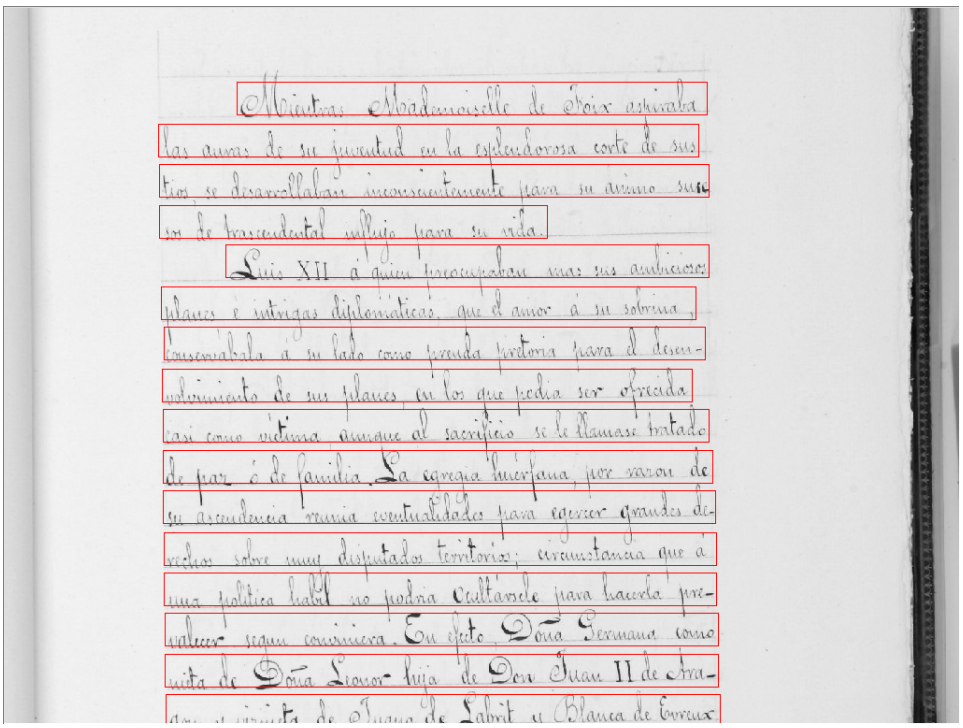


Figura 10: Imagen en pantalla completa con las líneas del texto encuadradas

3.2. Comunicación

La aplicación debía de ser capaz de comunicarse con el iViewX y con el Eye Tracker. Para llevarlo a cabo, se han utilizado Sockets UDP (User Datagram Protocol). La diferencia entre TCP (Transmission Control Protocol) y UDP es que UDP es más rápido, ya que no se preocupa en que llegue toda la información. Debido a esto, UDP se utiliza más para streaming de vídeo y audio, y no para datos importantes como páginas web, información de bases de datos, etc [19]. La posible pérdida de datagramas se ha tenido que tener en cuenta para la comunicación de la aplicación con el Eye Tracker.

En la Figura 11, podemos ver cómo se produce la comunicación de una forma abstracta. Esto no es más que una comunicación Cliente-Servidor, siendo el cliente nuestra aplicación y el servidor el Eye Tracker, que obedece a nuestras peticiones y nos envía información. Básicamente, la aplicación desarrollada envía comandos al

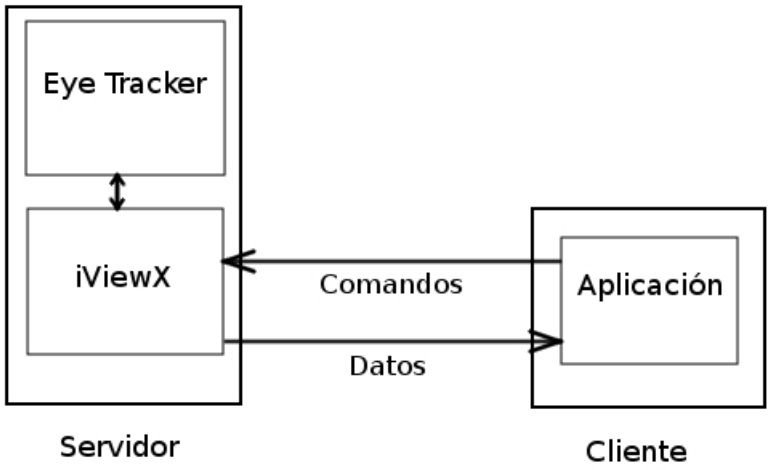


Figura 11: Comunicación Eye Tracker - Aplicación

iViewX. Tales comandos pueden ser opciones como “calibrar”, “especificar el formato de los datos”, “grabar”, etc. Por otra parte, lo que la aplicación recibe son datos, como por ejemplo, las coordenadas de pantalla de la calibración, el punto de calibración que se está evaluando, y, lo que es más importante, los datos de la mirada del usuario.

Para llevar a cabo esta fase, se han debido de añadir las siguientes funcionalidades a la interfaz anterior:

1. Conectar con el Eye Tracker utilizando un socket UDP.

2. Poder llevar a cabo la calibración, eligiendo entre 2, 5, 9 y 13 puntos de calibración. Al seleccionar uno de ellos, la aplicación muestra un fondo blanco (en pantalla completa) con un círculo en las coordenadas de la pantalla especificadas para el punto de calibración actual.

3. Sólo después de haber calibrado, poder abrir una imagen y mostrar los puntos en la pantalla a los que mira el usuario.

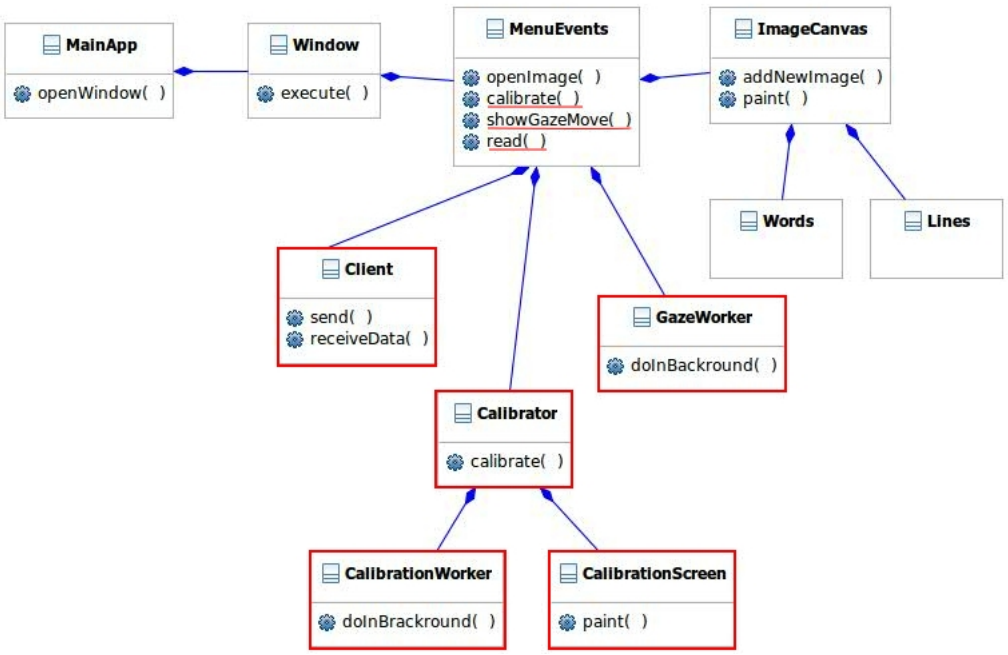


Figura 12: Diagrama de clases UML de la segunda fase

Para esta fase se han añadido a la aplicación las clases recuadradas en rojo (Figura 12).

Veamos estas nuevas clases:

- **Cliente:** Se encarga de enviar comandos al Eye Tracker y de recibir información.
- **Calibrator:** Esta clase se encarga de efectuar la calibración.
- **CalibrationScreen:** Muestra los puntos determinados de calibración por pantalla, y espera que el usuario apriete la tecla "espacio" para cambiar al siguiente punto de calibración.
- **CalibrationWorker:** Esta clase se encarga de que la clase Cliente envíe y reciba los datos necesarios para la calibración. [2]

[2]La razón por la cual necesitamos esta clase (y cualquier otra cuyo nombre acabe en "Worker" que veamos en este proyecto) es explicada en el Problema 2 de la seccion 6.1.

- **GazeWorker:** Esta clase se encarga de que el cliente envíe comandos y reciba los datos del streaming y que éstos lleguen a MenuEvents, que a su vez se los pasará a ImageCanvas para mostrarlos en la pantalla.

Además, se han debido de añadir funciones nuevas a **MenuEvents**: el método *calibrate* para poder hacer una calibración de 2, 5, 9 o 13 puntos (a elegir), el método *showGazeMov* para mostrar la mirada del usuario mientras lee un texto en la pantalla y el método *read* para grabar la mirada del usuario mientras lee, pero no mostrarla. Además, se ha modificado el método *paint* de **ImageCanvas** para poder mostrar la mirada del usuario encima de la imagen elegida en el caso de que se esté ejecutando el método *showGazeMov* de MenuEvents.

3.2.1. Calibración

Básicamente, la calibración funciona de la forma vista en la Tabla 1. Cuando la aplicación le dice al Eye Tracker que quiere hacer una calibración, debe recibir y guardar la resolución de la pantalla de calibración (por si es necesario hacer conversiones después) y las coordenadas de los puntos de calibración (para poder mostrarlos uno a uno). Despues, al recibir el mensaje *ET_CHG i*, mostrar el punto *i* por pantalla (como por ejemplo en la Figura 13), y sólo cambiarlo al recibir el mensaje *ET_CHG i+1* o *ET_FIN* (el cual indica que se ha acabado la calibración).

Tabla 1: Pasos ejecutados durante la calibración

t	**Cliente**	**Servidor**	**Explicación**
1	ET_CAL 5		Indica el número de puntos de calibración (5)
2		ET_CSZ 1024 768	El servidor informa al cliente de la resolución de la pantalla de calibración
3		ET_PNT 1 512 384	A partir de aquí, el servidor informa cuáles serán las coordenadas de los cinco puntos de calibración
4		ET_PNT 2 51 38	
5		ET_PNT 3 973 730	
6		ET_PNT 4 51 730	
7		ET_PNT 5 973 730	
8		ET_CHG 1	El servidor indica que empezará a tomar datos para el punto 1
9	(ET_ACC)		Opcional: el cliente comanda que se pase al siguiente punto de calibración
10		ET_CHG 2	
11	(ET_ACC)		
12		ET_CHG 3	
13	(ET_ACC)		
14		ET_CHG 4	
15	(ET_ACC)		
16		ET_CHG 5	
17	(ET_ACC)		
18		ET_FIN	El servidor indica que ya se han acabado los puntos de calibración

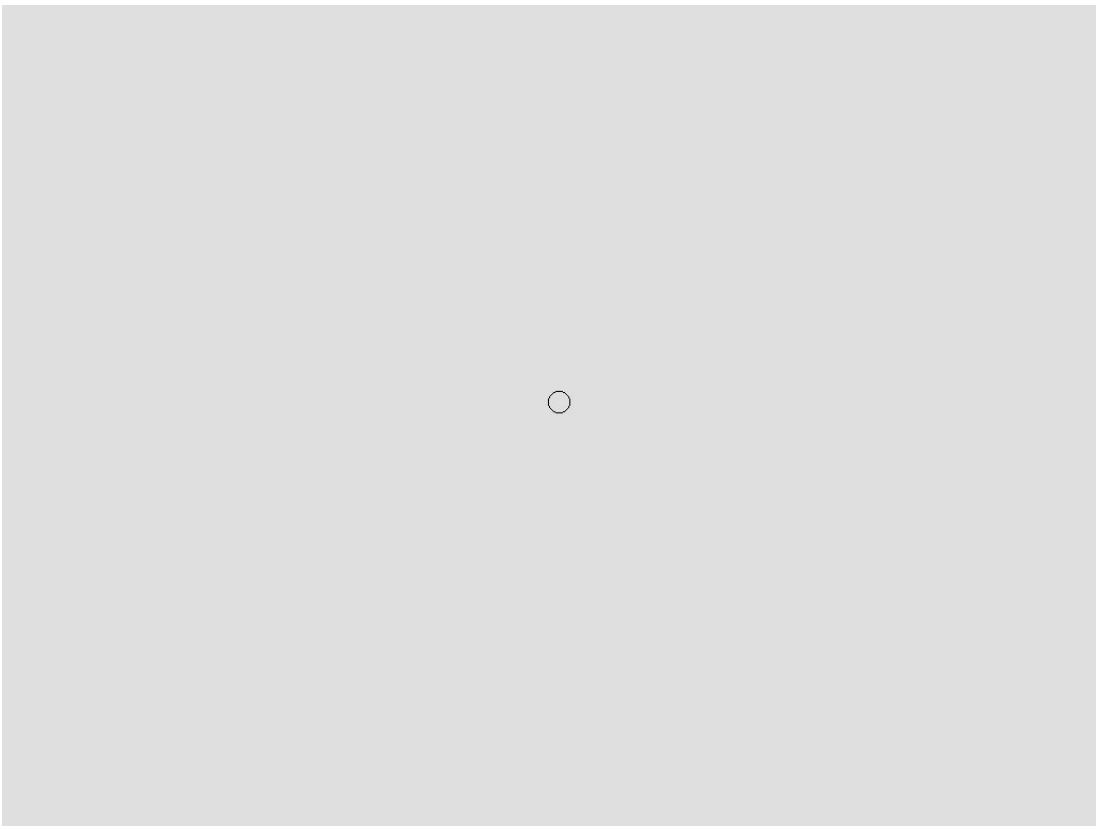


Figura 13: Ejemplo de un punto de calibración

3.2.2. Data streaming

Para la funcionalidad de recibir datos del Eye Tracker y opcionalmente grabarlos se han seguido los pasos de la Tabla 2.

En este caso, si lo que deseamos es mostrar al momento la mirada del usuario en la imagen (opción "Show gaze movement" en la Figura 14), cada vez que recibamos el mensaje *ET_SPL tstmp: x, y* tan sólo tenemos que mostrar en la pantalla un círculo en las coordenadas "x" e "y", haciendo las conversiones necesarias si la resolución de la pantalla no es la misma que la resolución de la calibración. En cambio, si decidimos que sólamente queremos grabar los datos (opción "Start reading" en la Figura 14), no hace falta mostrar en la pantalla la mirada del usuario. El programa iVIewX guardará los datos grabados en un fichero binario de extensión .idf. (iView

Data File). Para poder leer este fichero se necesita el programa IDF Event Detector de iViewX, que nos da un fichero .txt con los eventos grabados: fijaciones, sacadas y parpadeos. Se consideran los eventos que duran más de 80 ms (parámetro introducido en el Event Detector)

Tabla 2: Pasos ejecutados durante el data streaming

t	**Cliente**	**Servidor**	**Explicación**
1	ET_FRM %TS: %SX, %SY		El cliente indica el formato en el que quiere recibir los datos
2	(ET_REC)		Opcional: el cliente ordena que se deben grabar los datos
3	ET_STR		El cliente ordena que empiece el data streaming
i		ET_SPL time: x, y	El servidor envía el timestamps (en ms) y las coordenadas x e y de la mirada
n	(ET_STP)		Opcional: El cliente ordena que se dejen de grabar los datos
n+1	ET_EST		El cliente indica que el servidor deje de enviar datos
n+2	(ET_SAV nom_archivo)		Opcional: El cliente ordena que se guarde en el disco duro un fichero con los datos grabados

Figura 14: Menú de la aplicación

3.3. Recolección y análisis de datos

En esta fase, era necesario implementar dos experimentos (explicados con detalle en las secciones 4.1 y 4.2). Para el Experimento 1, la aplicación debía permitir leer un texto y grabar los movientos de los ojos al mismo tiempo. Esta funcionalidad ya estaba implementada antes, y sólo se tuvieron que hacer unos cambios. Para el segundo, se debía poder leer un texto y grabar los movimientos, pero en un tiempo limitado.

En esta fase, se han añadido dos métodos principales a **MenuEvents** (Figura 15), ya que la mayor parte ya estaba hecha en la fase anterior. El método *wordDifficulty* es el encargado de ejecutar el Experimento 1 y simplemente ejecuta el método *read*, pero con una imagen ya predefinida (la imagen del experimento). El método *wordSearching*

hace lo mismo, pero añadiendo un contador de tiempo. Cuando pasan diez segundos, la imagen desaparece de la pantalla.

Figura 15: Cambios hechos a MenuEvents

3.4. Pre-procesamiento y visualización

Finalmente, en esta cuarta fase se ha implementado una clase más: **ReplayerWorker**, que se encarga de leer el fichero de eventos generado por el IDF Event Detector y enviarle los datos a **MenuEvents** (el cual se los envía a **ImageCanvas**) para dibujar las fijaciones (con círculos) y las sacadas (líneas conectando las fijaciones).

El diagrama de clases final de la aplicación implementada se puede ver en la Figura 16, así como la clase y el método nuevos añadidos en esta última fase.

3.4.1. Segmentación de líneas

Dado que nunca podemos tener una grabación sin ruído, se ha implementado una segmentación de líneas, para que todas las fijaciones caigan en su línea correspondiente. Necesitamos esto porque, para saber dentro de qué palabra está una fijación, disponemos de las coordenadas de los recuadros de cada palabra. Si se produce ruído, por pequeño que sea, puede que las coordenadas de la fijación ya no estén dentro de ninguna palabra. En la Figura 17 vemos un ejemplo de esto. La aplicación no ha encuadrado ninguna palabra, porque no ha detectado ninguna fijación en ninguna palabra. En cambio, en la Figura 18, donde se ha usado la segmentación de líneas, vemos que casi todas las palabras están encuadradas.

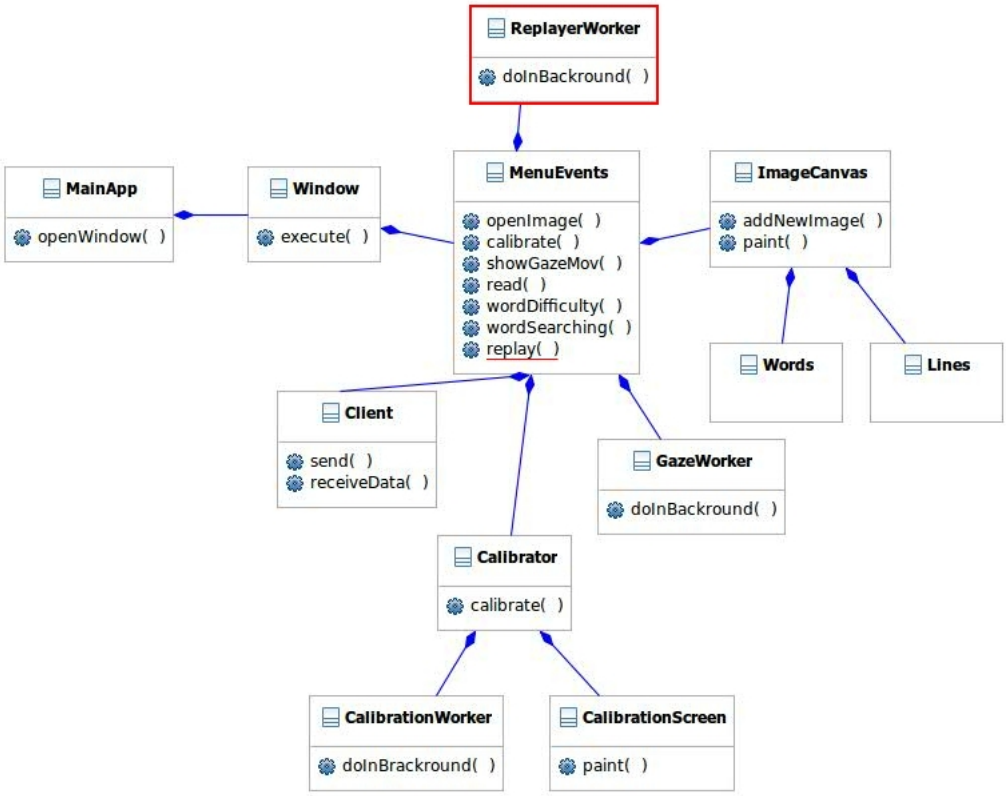


Figura 16: Diagrama de clases UML de la cuarta fase

La implementación de la segmentación de líneas contiene los siguientes pasos:

1. Recibir coordenadas de nueva fijación
2. Si no habíamos recibido otra fijación antes, mirar si ésta está cerca de la primera línea. Si no lo está, ignorar la fijación.
3. En caso contrario, calcular las diferencias de x e y para la fijación actual y la anterior.
4. Si el valor absoluto del cambio de x es mayor que 400 pixels, puede ser que se haya producido un cambio de línea.
5. Mirar el cambio de y para confirmarlo (si el cambio de y es mayor que la distancia entre dos líneas de este texto, se ha cambiado de línea). En el caso de un cambio de línea, se decrementa o incrementa una unidad de la línea actual.

6. Cuando ya sabemos en qué línea recae la fijación, cambiar el valor de y al valor de la mitad de la línea actual.

En la Figura 17 vemos todo el ruído producido cuando este voluntario leía. En cambio, en la Figura 18 vemos cómo ha sido quitado el ruído vertical gracias a la segmentación de líneas.

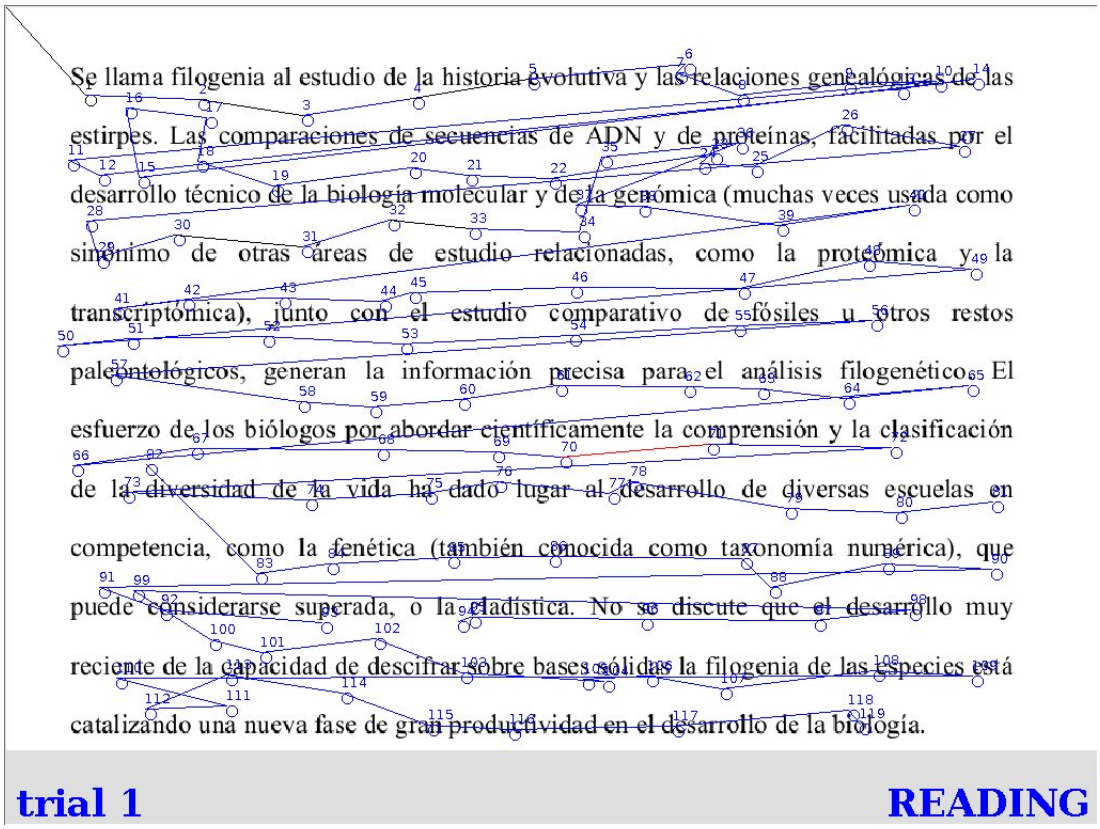


Figura 17: Datos grabados sin hacer la segmentación de líneas

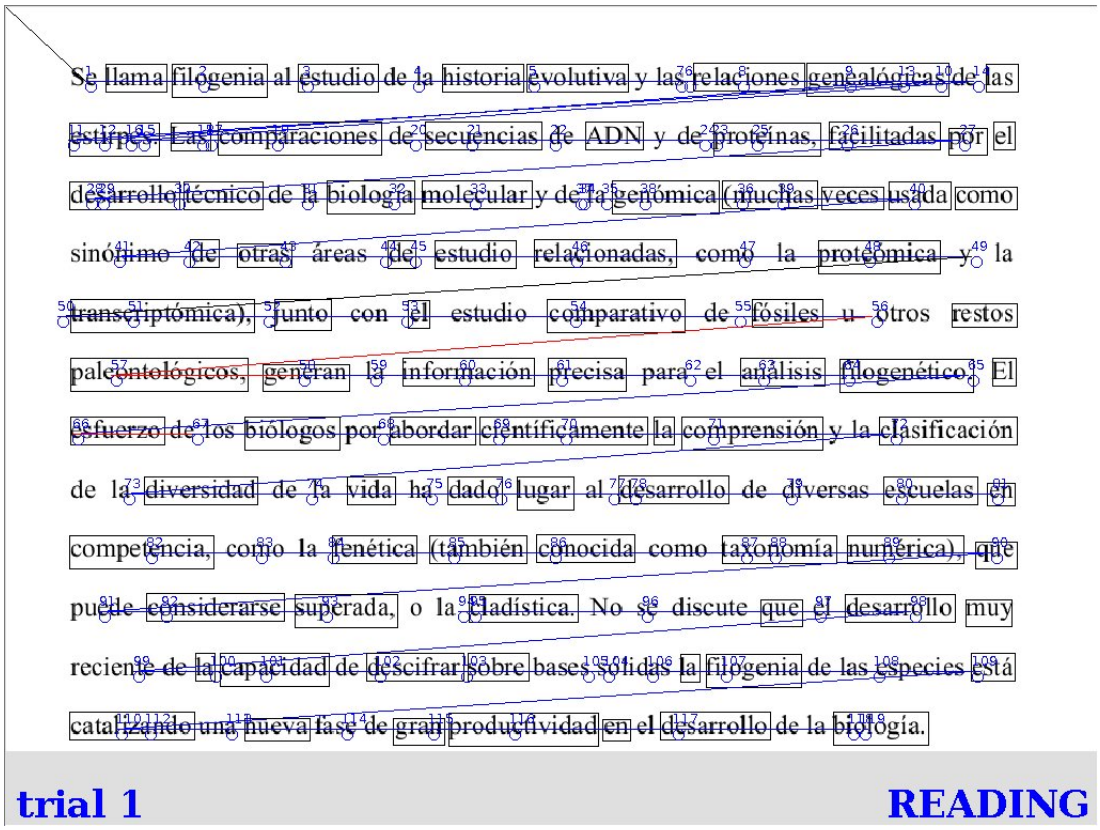


Figura 18: Datos grabados con segmentación de líneas

4. Diseño experimental

Esta parte ha consistido en implementar dos experimentos, ya mencionados en los objetivos del proyecto. El primero consistía en clasificar palabras según su dificultad, utilizando como características el comportamiento de varios voluntarios al leerlas y el segundo consistía en clasificar las sacadas hechas por los voluntarios en dos patrones de lectura diferentes. Estos experimentos se han realizado con seis voluntarios (dos mujeres y cuatro hombres). Cinco de estos voluntarios tenían entre 21 y 22 años, y el voluntario restante tenía 16 años. Veamos estos experimentos con más detalle.

4.1. Experimento 1: Palabras difíciles

El objetivo de este experimento era analizar el comportamiento de la mirada de las personas al leer una palabra desconocida (en un idioma que la persona habla con fluidez). Se quería comprobar si el número de fijaciones y la duración de las fijaciones en la palabra aumentaba al ser una palabra difícil. Este experimento fue inspirado por el montaje experimental usado por Andreas Dengel y Ralf Biebert (DFKI) durante ICDAR 2009. Para llevar a cabo esto, se ha escogido una parte del texto de [20], ya que contiene palabras desconocidas para alguien que no haya estudiado Biología. Después, se ha conseguido la participación de seis voluntarios para llevar a cabo el experimento. Se les ha hecho leer el texto en voz baja (a la vez que se grababan los datos), y despues rellenar un cuestionario. A continuación se incluye una parte de dicho cuestionario:

La palabra	La he visto en el texto		Conozco su significado	
... filogenia	[ ] sí	[ ] no	[ ] sí	[ ] no
... estirpes	[ ] sí	[ ] no	[ ] sí	[ ] no
... genómica	[ ] sí	[ ] no	[ ] sí	[ ] no
... proteómica	[ ] sí	[ ] no	[ ] sí	[ ] no
... transcriptómica	[ ] sí	[ ] no	[ ] sí	[ ] no
... filogenético	[ ] sí	[ ] no	[ ] sí	[ ] no
... fenética	[ ] sí	[ ] no	[ ] sí	[ ] no
... taxonomía	[ ] sí	[ ] no	[ ] sí	[ ] no
... cladística	[ ] sí	[ ] no	[ ] sí	[ ] no

Como podemos ver, los voluntarios han tenido que marcar en qué palabras se habían fijado, y de cuáles no sabían su significado previamente.

4.1.1. Datos para clasificar del Experimento 1

Después de segmentar las líneas, recogemos los siguientes datos para cada palabra escogida:

- Contamos el número de fijaciones que se han producido. Con tal de que el área de una fijación toque una palabra, ya la contamos. Es decir, el centro de la fijación no tiene porqué estar encima de la palabra, si la palabra está dentro del radio horizontal de la fijación ya es suficiente.
- Obtenemos el tiempo total de fijación por palabra, sumando cada tiempo de cada fijación en la palabra.
- Obtenemos el tiempo de la fijación más larga por palabra.

Esto lo hacemos para dos tipos distintos de palabras. Primero, para las palabras que

1. el usuario marcó en el cuestionario que sí las había visto en el texto y también
2. el usuario marcó que no conocía su significado.

Con el primer requisito nos aseguramos que el usuario estaba prestando atención a lo que leía en el texto, y que por lo tanto se ha fijado en las palabras. Y el segundo requisito es lo que define una palabra difícil para el usuario.

Segundo, para un número de palabras fáciles del texto previamente escogidas, que aproximádamente tengan la misma largura que las palabras difíciles. El segundo grupo de palabras será el mismo para cada voluntario, pero el primero no, ya que un voluntario puede no haber encontrado difíciles las mismas palabras que otro voluntario. Al final, el grupo de muestras constaba de 55 palabras, 28 fáciles y 27 difíciles. Estos datos acumulados de todos los voluntarios se escriben en un fichero donde cada línea representa una palabra y tiene el formato:

num_fijaciones tiempo_total_fijaciones tiempo_máximo_fijación ¿difícil?(0 o 1)

4.2. Experimento 2: Encontrar la respuesta

El objetivo de este experimento era implementar la heurística explicada en [1] para diferenciar entre leer y pasar la vista por encima del texto. Para hacer esto, se ha cogido el siguiente texto de Alicia en el País de las Maravillas, de Lewis Carrol:

> Alicia empezaba ya a cansarse de estar sentada con su hermana a la orilla del río, sin tener nada que hacer: había echado un par de ojeadas al libro que su hermana estaba leyendo, pero no tenía dibujos ni diálogos. «¿Y de qué sirve un libro sin dibujos ni diálogos?», se preguntaba Alicia. Así pues, estaba pensando (y pensar le costaba cierto esfuerzo, porque el calor del día la había dejado soñolienta y atontada) si el placer de tejer una guirnalda de margaritas la compensaría del trabajo de levantarse y coger las margaritas, cuando de pronto saltó cerca de ella un Conejo Blanco de ojos rosados.

Se les ha dicho a los mismos voluntarios del experimento anterior que tenían diez segundos para encontrar la respuesta a la pregunta "¿Qué quería tejer Alicia?". Nuestra hipótesis base era que, a causa del tiempo limitado, las personas se saltarían partes del texto para encontrar la respuesta más rápidamente, y que, por lo tanto, habría más clasificación de *skimming* que de lectura.

Como experimento extra, se ha realizado esta misma clasificación de sacadas en el Experimento 1 también, para comprobar si la gente leía más en el Experimento 1 que en el segundo (cosa que debería suceder).

5. Resultados

5.1. Clasificación de sacadas

Los experimentos hechos tienen como salida un archivo de eventos para cada texto leído (producidos con el programa IDF Event Detector como ya se explicó en la sección 3.2.2). Estos eventos están compuestos por fijaciones, sacadas y parpadeos, y para cada evento se especifica cuánto tiempo ha durado el evento, la posición de pantalla donde se estaba mirando, los radios horizontales y verticales (en el caso de las fijaciones) y las posiciones de inicio y fin (en el caso de las sacadas). Para clasificar los datos, la aplicación debe leer estos archivos, y a la vez que se hace esto, se va aplicando la heurística de [1] para diferenciar entre lectura normal y lectura rápida. En la Tabla 3 podemos ver una tabla similar a la mostrada en la seccion 1.2. La

diferencia es que ahora tenemos en cuenta la distancia recorrida por las sacadas en píxeles y no en número de carácteres recorridos.

Tabla 3: Heurística para clasificación de sacadas

Distancia horizontal de la sacada en píxeles	**Puntos lectura**	**Puntos skimming**
$0 < x <= 11 * T$	10	5
$11 * T < x <= 21 * T$	5	10
$21 * T < x <= 30 * T$	-5	8
$-6 * T <= x < 0$	-8	-8
$-16 * T <= x < -6 * T$	-5	-3

Siendo T el tamaño de la letra del texto en píxeles, calculada a partir de la resolución de la pantalla y el tamaño de la letra en milímetros, se aplican estas heurísticas para cada sacada leída del fichero de eventos a dos contadores, *reading* y *skimming*. Después de aplicar las heurísticas, se analizan los contadores de los puntos y el contador con mayor valor indica el tipo de lectura ganador. Al llegar al final de la línea, se ponen los contadores a cero para empezar la siguiente línea.

5.2. Experimento 1

Al contrario que la clasificación del Experimento 2, que se hacía al momento de leer los archivos de eventos, la clasificación del Experimento 1 se realiza después de haber leído los eventos y adquirido los datos para cada palabra. Para clasificar los datos, se ha escrito un script de Matlab que lee el fichero descrito en el apartado anterior, y realiza el aprendizaje y la validación.

5.2.1. Clasificación

Se han utilizado los siguientes métodos de clasificación:

- **K-Nearest-Neighbors (KNN):** Este método consiste en calcular la distancia (en nuestro caso Euclidiana) entre una muestra de test y todas las de aprendizaje, y, de entre los K vecinos más cercanos, se calcula la clase de la muestra test.
- **Árboles de Decisión (AD)**: Este método es un método de clasificación binario que crea un árbol binario a partir de las muestras de aprendizaje, en el que cada nodo es un atributo, cada rama es un posible valor del atributo del nodo anterior, y cada hoja son nodos que clasifican el ejemplo como positivo o negativo.
- **Support Vector Machine (SVM):** Este método es un método de clasificación binario en el que, dadas las muestras de aprendizaje, crea una hipersuperfície que separa las dos clases con el mayor margen posible. A la hora de clasificar una muestra, sólo hay que mirar en qué lado de la hipersuperfície recae la muestra. En este proyecto, se ha utilizado el kernel Gausiano.

5.2.2. Validación

Se han utilizado los siguientes métodos de validación:

- **Hold-out:** Este método consiste en dividir las muestras aleatoriamente en dos grupos. Primero se realizará el aprendizaje con uno de los grupos, y con el otro se realizará el test. Después, se intercambirán los papeles de los grupos.
- **K-fold:** Este método consiste en dividir la muestra en K grupos balanceados aleatorios. Después, se utilizarán K-1 de estos grupos para el aprendizaje, y el grupo restante para el test. Esto se hace K veces, con tal de que todos los grupos sean un grupo de test una vez.

5.2.3. Resultados obtenidos

A continuación podemos ver los resultados obtenidos para el Experimento 1. Los resultados están mostrados en forma de matriz de confusión:

		Valor predicho	
		fácil	**difícil**
Valor real	**fácil**	verdadero negativo	falso negativo
	difícil	falso positivo	verdadero positivo

En cada posición de la matriz 2x2 se pondrán el número de ocurrencias para cada caso:

- Verdadero negativo: Es el caso en el que una muestra es clasificada como palabra *fácil*, y *acierta*.
- Falso negativo: Es el caso en el que una muestra es clasificada como una palabra *fácil*, y *no acierta*.
- Falso positivo: Es el caso en el que una muestra es clasificada como una palabra *difícil* y *no acierta*.
- Verdadero positivo: Es el caso en el que una muestra es clasificada como una palabra *difícil* y *acierta*.

Los valores de las matrices son las acumulaciones de todos los grupos de test clasificados para el método de validación específico. También vienen dados los porcentajes de aciertos (PA):

KNN:

vecinos = 10:[1]

		K-fold		
Hold-out	K=3	K=5	K=10	
78%	76%	75%	75%	PA

25	3	24	4	22	6	23	5
9	18	9	18	8	19	9	18

AD:

		K-fold		
Hold-out	K=3	K=5	K=10	
70%	69%	56%	61%	PA

21	7	21	7	14	14	14	14
9	18	10	17	10	17	7	20

[1]Escojemos que el número de vecinos sea 10 porque al elegir un número muy bajo o muy alto de vecinos, los resultados son bajos. La causa del primero es que, como veremos en la Figura 19, algunas muestras de palabras difíciles están cercanas a las palabras fáciles, y cuantos menos vecinos escojamos, menos probabilidades tenemos de clasificar la muestra en la clase correcta. La causa del segundo es el bajo número de muestras de las que disponemos. Escoger un número grande de vecinos podría llevar a decir que todas las muestras de aprendizaje están cercanas a la muestra de validación.

SVM:

		K-fold		
Hold-out	K=3	K=5	K=10	
69%	67%	69%	70%	PA

19	9	21	7	20	8	22	6
8	19	11	16	9	18	10	17

Podemos observar que los resultados están por encima del 50 % de aciertos. El método de clasificación más eficiente parece ser que es el KNN, cuyos aciertos no bajan del 75 %. Es de esperar que, según el tamaño del grupo de aprendizaje, el número de buenas clasificaciones varíe proporcionalmente. Sin embargo, podemos observar que éste no es el caso, ya que en la mayoría de los casos tener el grupo de aprendizaje del mismo tamaño del grupo de test (Hold-out) nos da mejores resultados.

En la Figura 19 podemos observar cómo están repartidas todas las muestras en el espacio. Se puede observar que a partir de unos valores de los ejes "x", "y" y "z" dejan de haber palabras fáciles. Sin embargo, hay algunas palabras difíciles que están mezcladas con las palabras fáciles, por lo que esto ya dificulta el aprendizaje y clasificación de los métodos de AD y SVM. Sin embargo, para el caso del método KNN, no separamos el espacio en regiones, sino que calculamos la distancia entre puntos. Así pues, si una muestra de test está en la parte donde se mezclan las palabras fáciles y las difíciles, se tendrán en cuenta sus vecinos más cercanos (que podrán ser de las dos clases), y no la región del espacio en la que esté.

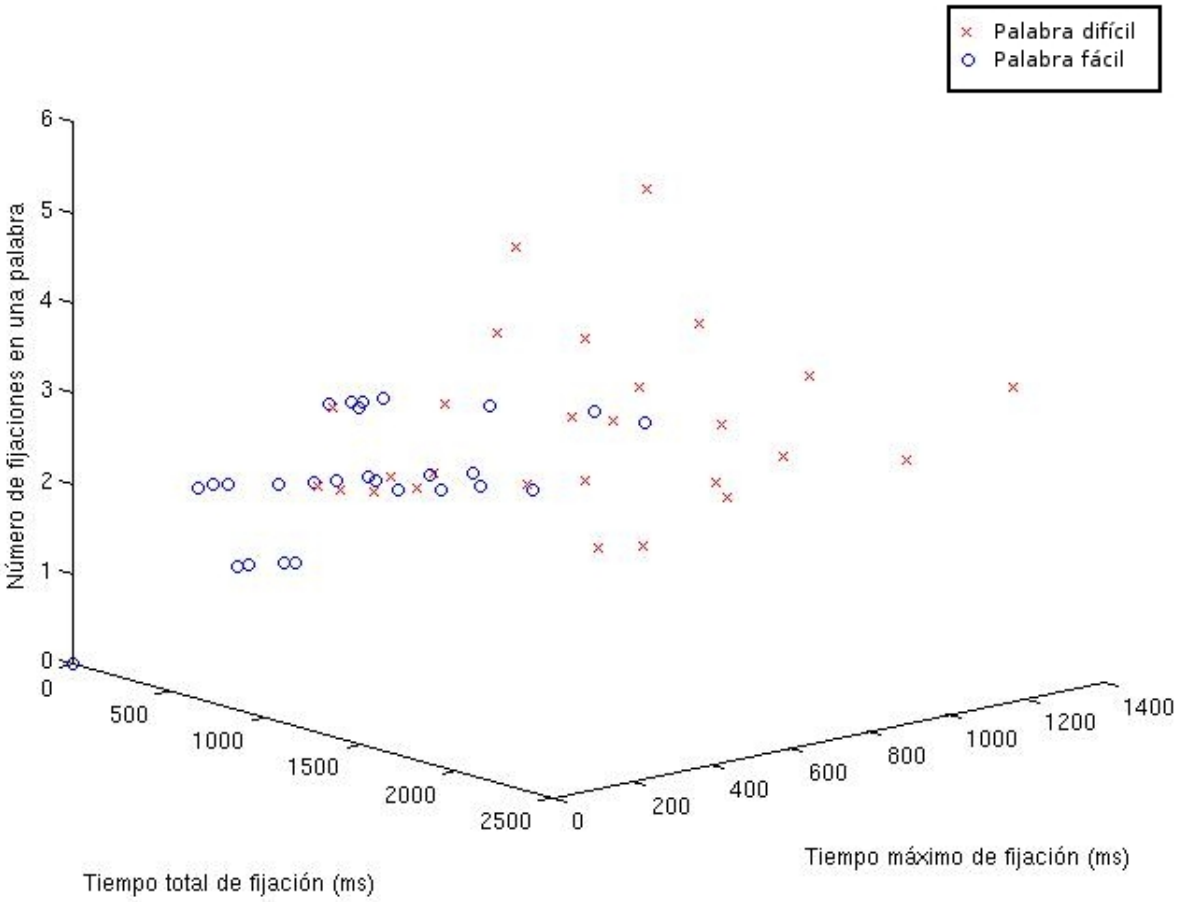


Figura 19: Espacio 3D de las muestras

5.3. Segundo experimento

En la Tabla 4 podemos ver la comparación de los porcentajes de skimming (pasar la vista por encima) de los dos experimentos hechos, para cada una de las seis personas voluntarias.

Tabla 4: Porcentajes de skimming para los dos experimentos hechos por los seis voluntarios

	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$	**Media**
Exp. 1		64	44	36	59	44	49.4
Exp. 2	58	100	74	78	43	40	65.5

Era de esperar que el porcentaje de skimming para el Experimento 1 fuera más bajo que el del Experimento 2, ya que en el primero los voluntarios se han dedicado a leer, y en el segundo tenían que encontrar la respuesta a la pregunta lo más rápido posible. Sin embargo, excepto en un caso, los valores no han sido exactamente del 0 % o del 100 %, ya que las personas tienen diferentes modos de lectura. Podemos ver que la lectura de P_2 es más del tipo de pasar la vista por encima que de leer lentamente. Y por el otro lado, P_6 se ha dedicado a leer más lentamente los dos textos, aunque en el segundo texto tuviera prisa para leerlo. Todas las personas son distintas a la hora de asimilar lo que están leyendo y leen a una velocidad distinta.

6. Discusión

En esta sección veremos los problemas encontrados a lo largo del proyecto, y las mejoras que se podrían implementar, ya sea para obtener mejores resultados en las clasificaciones, o para expandir el proyecto.

6.1. Problemas encontrados

A lo largo de este proyecto, se han encontrado cuatro problemas:

1. Problemas de conexión: A la hora de probar la implementación de la conexión al Eye Tracker y al intentar enviar un datagrama al Eye Tracker, éste no llegaba. Simplemente, el iViewX no daba señales de haber recibido nada. Después de comprobar que los parámetros de conexión fueran los correctos (IPs y puertos de la fuente y el destino) y de comprobar con el programa WireShark (capturador de paquetes) que el datagrama llegaba al ordenador destino, miramos dos programas ejemplo (uno en C++ y otro en C#) en los que se ejecutaba la co-

nexión con el Eye Tracker. Todo era exactamente igual al código que habíamos implementado (excepto las diferencias de los lenguajes, claro). Excepto un detalle difícil de ver. Al final del mensaje que se quería enviar, concatenaban un salto de línea ("\n") cosa que nosotros no hacíamos. Al hacer este paso, la comunicación funcionó a la perfección.

2. Threads: Como hemos visto en la sección 3.2, tenemos varias clases Worker. La razón es la siguiente: el paquete AWT de Java contiene un thread llamado Event Dispatching Thread (EDT), que se ocupa de procesar los eventos de la cola de eventos de la interfaz gráfica de usuario. Esto quiere decir que mientras la aplicación esté haciendo otra cosa (como por ejemplo enviar y recibir datos), AWT no hará nada gráfico hasta que se acabe la operación que se está realizando. Esto era un problema, ya que necesitábamos mostrar elementos por pantalla (ya sea puntos de calibración, o la mirada del usuario) a la vez que se enviaban y recibían datos, o se leían los datos de un fichero. La solución fue utilizar las clases Worker, que contienen el método doInBackround, que actúa como un thread más, y así no interfiere con el trabajo del EDT.

3. Problemas de calibración: Antes, la conexión que se realizaba era inalámbrica. Y varias veces, al hacer la calibración, el programa se atascaba en un punto de calibración concreto, porque el Eye Tracker ya había enviado el mensaje *"ET_CHG i"*, pero el datagrama se había perdido y nuestro programa se quedaba esperando por algo que nunca llegaría debido al uso de sockets UDP. La solución fue conectar los dos ordenadores con un cable Ethernet crossover, y de este modo se pierden muchos menos datagramas.

4. Gafas: Algunas de las personas que se ofrecieron voluntarias llevaban gafas. En función del cristal de éstas, los rayos infrarojos no se reflejan nunca en los ojos, sino en los cristales de las gafas, y, por lo tanto, el Eye Tracker no puede detectar los ojos. Quizás cambiando el ángulo del RED, se puede solucionar este problema. Pero como todos los voluntarios con las gafas problemáticas podían ver lo suficientemente bien sin ellas, simplemente se quitaron las gafas.

6.2. Mejoras

Unas mejoras posibles de este proyecto podrían ser:

1. Escoger más características que definan a las palabras difíciles, como puede ser el número de veces que una persona relee una palabra. Esto se podría hacer teniendo en cuenta cuándo las fijaciones salen de la palabra, y después vuelven a entrar.

2. Conseguir más muestras de palabras difíciles (27 muestras no son muchas). Para solucionar esto, necesitaríamos o un texto más largo, con más palabras complicadas (aunque esto no asegura que vayan a ser difíciles para los voluntarios), o hacer los experimentos con más gente. De esta forma, al tener más datos, podríamos asegurar si realmente el número de fijaciones, el tiempo total de fijaciones y el tiempo máximo de una fijación por palabra difícil son más grandes que los de una palabra fácil excepto por unos casos muy concretos.

3. Hay que tener en cuenta que hemos realizado el aprendizaje y test con los datos de todos los voluntarios mezclados. Ya que cada persona tiene una manera distinta de leer, una mejora sería realizar el aprendizaje y test para cada voluntario por separado y ver si de verdad se producen mejoras en la clasificación.

4. Una vez que se tuviera un clasificador que diera resultados con un porcentaje pequeño de error, se podrían identificar las palabras difíciles al mismo momento en que se estuviera leyendo un texto.

5. Se podría mejorar la manera de clasificar sacadas del experimento 2 usando la lógica difusa. Los valores umbrales de números de carácteres son demasiado estrictos para algo que puede resultar diferente para cada persona. Si pudiéramos evitar esto, quizás se clasificarían mejor las sacadas.

7. Conclusiones

Como hemos mencionado en la introducción, la motivación principal de este proyecto era poder asistir la transcripción de documentos escritos a mano. Utilizando el Eye Tracker para saber qué palabra se está leyendo, y el reconocimiento de voz para identificar la palabra leída en voz alta, la transcripción mediante reconocimiento de carácteres produciría menos errores. No obstante, a medida que avanzaba el proyecto surgieron dificultades que obligaron a reformular algunos de los objetivos dando lugar a los experimentos presentados. El primer cambio que se produjo fue en los tipos de textos utilizados: empezamos a usar textos contemporáneos en vez de las imágenes de documentos antiguos debido a que la caligrafía de los documentos era difícil de leer. Después, debido a las complicaciones que supondría implementar un reconocedor de voz, decidimos desviar los objetivos del proyecto.

Gracias a estos experimentos se ha comprobado que usando el Eye Tracker puede ser posible "interpretar"la mirada de los lectores para asistir a su lectura, como por ejemplo, haciendo que aparezcan las definiciones de las palabras dificiles para el lector o resaltando las partes importantes del texto cuando el lector está pasando la vista por encima [4]. Nuestra experiencia al leer un libro puede ser mejorada gracias a la tecnología de seguimiento de ojos y de una correcta clasificación de datos.

En conclusión, podemos decir que se han cumplido todos los objetivos finales del proyecto, que son los objetivos descritos en este documento. Objetivos como implementar la aplicación principal para visualizar imágenes, implementar la comunicación con el Eye Tracker, y mejorar la aplicación para poder realizar dos experimentos y poder extraer datos de los experimentos grabados. Aunque se puedan implementar las mejoras mencionadas en la sección 6.2, los experimentos base han sido realizados con éxito y con resultados aceptables dados los conjuntos de datos de los que se disponían.

Referencias

[1] Ralf Biedert, Georg Buscher, Andreas Dengel.
The eyeBook – Using Eye Tracking to Enhance the Reading Experience, Informatik Spektrum, Alemania, 2009

[2] Robert J. K. Jacob, Keith S. Karn
Eye Tracking in Human–Computer Interaction and Usability Research: Ready to Deliver the Promises ,
The mind's eye: cognitive and applied aspects of eye movement research, Boston, 2003

[3] Eric Missimer, Margrit Betke.
Blink and Wink Detection for Mouse Pointer Control, Boston University, EEUU, 2010

[4] Text 2.0, `http://text20.net`
Ralf Biedert, Fecha accedido: 28 de Noviembre de 2010.

[5] Georg Buscher, Andreas Dengel, Ludgen van Elst.
Eye Movements as Implicit Relevance Feedback , Florence, Italy, 2008

[6] Fixation (visual), `http://en.wikipedia.org/wiki/Fixation_(visual)`,
Última modificación: 28 de Abril de 2011. Fecha accedido: 7 de Junio de 2011.

[7] Microsaccade, `http://en.wikipedia.org/wiki/Microsaccade`,
Última modificación: 25 de Marzo de 2010. Fecha accedido: 7 de Junio de 2011.

[8] Eye tracking glossary, `http://eyetracking.oneupweb.com/resources/glossary/ocular-drift.htm`
Fecha accedido: 7 de Junio de 2011.

[9] Ocular tremor, `http://en.wikipedia.org/wiki/Ocular_microtremor`,
Última modificación: 21 de Marzo de 2011. Fecha accedido: 7 de Junio de 2011.

[10] Movimientos oculares,
http://www.uned.es/eyemovements-lab/links/tipos_movimientos.htm,
Antonio Crespo PhD & Raúl Cabestrero PhD, Faculty of Psychology-UNED,
Fecha accedido: 7 de Junio de 2011.

[11] Saccade, http://medical-dictionary.thefreedictionary.com/Saccades,
Medical dictionary. Fecha accedido: 7 de Junio de 2011.

[12] A.T. Duchowski, Eye Tracking Methodology, 2nd ed.: Springer, 2007.

[13] IViewX System Manual, versión 2.4. SensoMotoric Instruments, Agosto 2009.

[14] Electrooculography, http://en.wikipedia.org/wiki/Electrooculography,
Última modificación: 31 de Mayo de 2011, Fecha accedido: 7 de Junio de 2011.

[15] http://electrooculography.wordpress.com,
Fecha accedido: 7 de Junio de 2011.

[16] http://ppw.kuleuven.be/labexppsy/lepSite/resources/index.php?
content=purkinje,
Fecha accedido: 7 de Junio de 2011.

[17] RED / RED250 / RED500,
http://www.smivision.com/en/gaze-and-eye-tracking-systems/
products/red-red250-red-500.html,
SensoMotoric Instruments, Fecha accedido: 7 de Junio de 2011.

[18] Jesús Pando Barrón, Manual de Java, Parte I, versión 1.0.

[19] TCP vs. UDP, http://www.skullbox.net/tcpudp.php,
Erik Rodriguez, Fecha accedido: 11 de Junio de 2011.

[20] Biología, http://es.wikipedia.org/wiki/Biolog\%C3\%ADa#Filogenia.

Resumen

A medida que avanza la tecnología, cada vez son más comunes los libros digitales. Por eso, existen varias formas de mejorar la experiencia de lectura del usuario, como mostrar la definición de una palabra que resulte difícil, o resaltar lo importante del texto cuando se pasa la vista por encima. En este proyecto, se ha investigado la base de esto con la ayuda de un Eye Tracker. Se ha implementado una clasificación en palabras fáciles y difíciles dependiendo de cómo una persona lee, y una forma de saber si se está leyendo el texto o pasando la vista por encima.

Resum

Amb l'avanç tecnològic, cada cop son més comuns els llibres digitals. Per això, existeixen diferents maneres de millorar l'experiència de lectura de l'usuari, com mostrar la definició d'una paraula que resulti complicada, o remarcar allò important del text quan es passa la mirada per sobre. En aquest projecte, s'ha investigat la base d'això, amb l'ajuda d'un Eye Tracker. S'ha implementat una classificació en paraules fàcils i difícils depenent de com una persona llegeix, i una forma de saber si s'està llegint el text o només passant la mirada per sobre.

Abstract

With the progress of technology, digital books are becoming more common every day. Because of this, there are several ways to improve the reading experience of a person, like showing the definition of a difficult word, or making the important parts of a text stand out when the person is just skimming. In this project, the base of this has been investigated with the aid of an Eye Tracker. A classification of easy and difficult words depending on the way a person reads has been implemented, as well as a way to know if a person is reading or skimming.