Técnicas de regresión para la estimación de la localización de la mirada

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial,
Reconocimiento de Formas e Imagen Digital

Autor
Jorge Mansanet Sandín

Directores
Alberto Albiol Colomer
Roberto Paredes Palacios

Septiembre de 2012

Índice

Resumen

Desde hace más de 30 años han habido muchos trabajos relacionados con la estimación de la mirada. Esto es debido, en parte, a la cantidad de campos de aplicación posibles y que hacen referencia a temáticas muy diferentes. Sin embargo, aun existen desafíos que resolver, debido a posibles problemas como la oclusión, la diferencia entre los ojos de las personas, las condiciones de luz, diferentes escalas, etc. Aunque la tendencia general es intentar mejorar la precisión de los sistemas, no tenemos que dejar de lado otros aspectos importantes. Por ejemplo, en determinadas aplicaciones podemos estar más interesados en trabajar con un sistema no intrusivo y/o en condiciones no controladas, pero por el contrario, no requerir tanta precisión. La utilización de cámaras de consumo nos puede permitir, además, tener un sistema de localización de la mirada bastante económico y que cumpla nuestras necesidades.

Por ello nuestro trabajo se centra en este tema. Por un lado, hemos realizado un estudio preliminar en unas condiciones controladas para estudiar qué técnica y configuración de parámetros va mejor para el sistema ideado. Por otro lado, hemos creado un aplicación en tiempo real para demostrar el funcionamiento del sistema. Esta aplicación obtiene la posición en la pantalla donde estamos mirando con unos resultados suficientemente buenos para el tipo de aplicaciones donde se podría implantar.

Capítulo 1

Introducción

En los últimos años se han desarrollado muchos trabajos referentes al concepto de interacción humano - máquina (HCI). En concreto, la estimación de la mirada de una persona es un problema que se ha intentado abordar desde hace mucho tiempo [17] [22]. Esto es debido a los muchos campos de aplicación posibles: seguridad, publicidad, ayuda a discapacitados, etc. Pese a esto, el problema sigue suponiendo un desafío para los investigadores debido a problemas inherentes como la oclusión, la variabilidad entre los ojos de distintas personas, las condiciones de iluminación, variación en las escalas, etc. Además, para estimar la mirada necesitamos resolver otros problemas relacionados, como son la detección de la cara de la persona, la situación de los ojos en la cara y la estimación de la pose de la cabeza.

Nuestro trabajo se centra en intentar estimar la posición donde la persona está mirando delante de una pantalla. Este problema se conoce en la literatura como la estimación del *Point-of-regard* (POR). Normalmente, si tenemos la dirección de la mirada, podemos obtener fácilmente la posición en la pantalla, calculando la intersección del rayo, que contiene la dirección de la mirada, con la pantalla de visión. Sin embargo, en nuestro caso realizamos la estimación de la posición mediante un método directo, obteniendo el punto donde estamos mirando en la pantalla.

Existen multitud de técnicas para la estimación de la mirada [14]. Muchas de estas técnicas son bastante intrusivas, utilizando, muchas veces, iluminación infrarroja o complejos sistemas hardware. Podemos ver un ejemplo de este tipo de sistemas en la Figura 1.1.

Además, muchas de las técnicas requieren escenarios controlados. Normalmente, este tipo de técnicas obtienen buenos resultados en la estimación, sacrificando otros aspectos como son la sencillez del sistema o el ambiente no controlado en el que puede estar la persona [24] [29] [33]. Sin embargo, nuestro interés se centra en desarrollar un sistema no intrusivo, capaz de trabajar en un ambiente no controlado de iluminación,

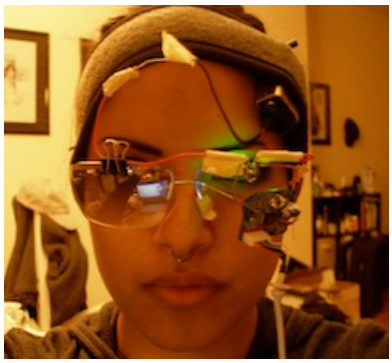


Figura 1.1: Complejo sistema hardware para la estimación de la mirada.

y utilizando cámaras de consumo no especializadas. Algunos trabajos relacionados son [7] [30]. Como es de esperar este sistema obtendrá unos resultados menos precisos que los anteriores comentados. Sin embargo, el tipo de aplicaciones hacia las que se orienta esta técnica, no necesitan demasiada precisión. Por ejemplo, si pensamos en el campo de la ayuda a discapacitados, podríamos aplicarlo para la ayuda en la lectura de un documento. Esto consistiría en detectar cuándo la mirada llega al final de una página, para que pase automáticamente a la siguiente sin necesidad de interacción. Para ello solo tendríamos que distinguir entre un número limitado de zonas en la pantalla. Comentar que el sistema creado es individual para cada persona. Es decir, existe una fase inicial *offline* en la que la aplicación es entrenada con la persona en cuestión. Una vez entrenado, podemos trabajar de forma *online* sin necesidad de volver a entrenar si no cambia el sujeto.

1.1. Estado del arte

Como hemos comentado antes, el tema abordado ha sido tratado extensamente desde hace muchos años. Una parte fundamental en la estimación de la mirada corresponde a la detección y seguimiento de la posición del ojo. Podemos clasificar las técnicas de este campo en dos grupos: basadas en la forma o basadas en la apariencia [14]. El primer grupo intenta detectar el ojo utilizando características locales del mismo y de su contorno (iris, pupila, *glint*, etc). Las características utilizadas pueden ser bordes, *corners* o respuestas de filtros en puntos concretos de la imagen [16] [13] [18]. Los modelos basados en la apariencia (aproximación holística) intentan detectar el ojo basándose en un modelo que se compara, mediante medidas de similitud, con distintas zonas de la imagen [21] [1]. En nuestro caso hemos utilizado un proceso en el que ajustamos un modelo deformable a la figura de la cara [31]. El modelo está formado por una distribución de puntos característicos, entre los que se encuentran los que hacen

referencia al ojo. Para pequeñas variaciones de la posición de la cabeza, este método nos proporciona una estimación muy exacta de la posición de los ojos.

Otra parte fundamental es la propia estimación de la dirección de la mirada. En nuestro caso vamos a estimar el POR debido a que queremos conocer la posición donde estamos mirando delante de una pantalla. La mayoría de los métodos de estimación de la mirada se basan en extraer características locales de los ojos. La principal razón es que hay partes del ojo que son fácilmente detectables, como son la pupila o el *glint* (utilizando luz activa). Dentro de este conjunto de técnicas existen dos aproximaciones según en qué se basan: basadas en interpolación (regresión) o basadas en un modelo 3D. Las técnicas basadas en interpolación [13] [27] [36] asumen que el modelo que nos traduce las características locales de la imagen a coordenadas de la dirección de la mirada se puede estimar mediante técnicas de regresión. Por ejemplo, podemos asumir un polinomio como modelo paramétrico [27], o utilizar una red neuronal [19]. Las técnicas basadas en modelos 3D se basan en un modelo geométrico del ojo que nos da la dirección de la mirada. El punto donde miramos se obtiene intersectando la dirección con el objeto que se mira [28] [35]. Pese a que estas técnicas obtienen muy buenos resultados, muchas de ellas se basan en métodos intrusivos o complicados sistemas hardware.

Nuestro método se encuadra en el conjunto de técnicas basadas en la apariencia. Al igual que en la detección de los ojos, este tipo de técnicas no buscan características locales, sino utilizar directamente la información de la imagen. Básicamente, estos métodos buscan una caracterización representativa de la información que nos da la imagen, para aprender un modelo de regresión que aprenda a interpretar los cambios en la apariencia de las imágenes. Esto se consigue entrenando el sistema con un conjunto de datos de entrenamiento que representa las posibles variaciones que puede haber. Con esta información buscamos una función que nos devuelve directamente las coordenadas en la pantalla. Muchos trabajos obtienen imágenes recortadas de los ojos que sirven para entrenar funciones de regresión con tecnicas como, redes neuronales [3] [32], procesos gausianos [36], o *manifold learning* [34]. Una de las ventajas de estos métodos es que no requieren la calibración de la cámara ni de las posiciones de los objetos, ya que el mapeo se realiza directamente desde el contenido de las imágenes. Otra ventaja importante es que reducimos el posible error cometido al estimar partes específicas del ojo como el iris o el *glint*. Además, no utilizamos técnicas intrusivas, y en nuestro caso, las cámaras utilizadas son de bajo coste (cámaras de consumo).

Las posibles aplicaciones que utilizan la estimación de la mirada son muchas y de muy diferentes campos. En la literatura podemos distinguir dos tipos de aplicaciones [10]: de diagnóstico e interactivas. Las aplicaciones de diagnóstico proponen un método

para obtener el *Point of Regard* de la persona. Esta información es útil para analizar qué tipo de anuncios mira una persona, ayuda a pilotos en cabinas de aviones, estudio de donde fija la atención el ser humano, etc. Los aplicaciones interactivas se centran en utilizar la mirada como interfaz de control de un dispositivo, o para modificar el comportamiento de una aplicación basada en la atención del usuario. Utilizando la mirada para una determinada acción dejamos las dos manos libres para realizar otras tareas, aumentando las posibilidades de interacción. Dentro de este tipo de técnicas destacan las que se centran en ayuda a discapacitados. En algunos casos, la mirada se convierte en la única herramienta para la comunicación e interacción con humanos y dispositivos.

1.2. Objetivos

El trabajo intenta conseguir varios objetivos. Por un lado queremos realizar un estudio comparativo para saber qué tipo de algoritmo funciona mejor con el sistema que hemos creado. Los algoritmos probados implementan la técnica de regresión para obtener la posición a la que miramos en la pantalla (POR), a partir de las imágenes obtenidas con las cámaras de consumo. Otro objetivo es crear una base de datos con la que poder entrenar y testear las técnicas. Además de realizar el estudio comparativo, pretendemos calcular cuál es la combinación óptima de parámetros del modelo que nos proporciona el mejor resultado para la base de datos construida. Otro objetivo es comparar la diferencia en las prestaciones del sistema al usar una *webcam*, que nos proporciona una imagen en color, y un dispositivo tipo *Kinect*, que además nos da la información de la profundidad. Estas pruebas las hemos realizado en un ambiente controlado, ya que hemos fijado la pose de la cabeza para evitar posibles errores debidos al movimiento de la misma. La iluminación y otros parámetros del ambiente de trabajo no son controlados. Como se comentará al final del trabajo, actualmente estamos trabajando en eliminar esta restricción y poder trabajar en un ambiente totalmente no controlado y no intrusivo.

La segunda parte del trabajo se centra en poner en práctica los resultados obtenidos en el estudio previo. Para ello hemos creado una aplicación en tiempo real que estima el POR sobre la pantalla. La aplicación desarrollada consta de una fase de entrenamiento y una de test para cada persona. Una vez realizado el entrenamiento de una persona no es necesario volver a realizarlo de nuevo. El objetivo de la aplicación es comprobar el funcionamiento del sistema desarrollado.

Capítulo 2

Metodología utilizada

2.1. Descripción general del proceso

En esta sección vamos a explicar, de forma genérica, las etapas que hemos desarrollado en el trabajo. Como hemos comentado antes, nuestro objetivo es estimar el *Point of regard* sobre el que estamos mirando en una pantalla. Pese a que el trabajo consta de distintas partes diferenciadas, el método general que hemos implemetado sigue las etapas que podemos ver en la Figura 2.1. Notar que estas etapas son específicas para cada persona.

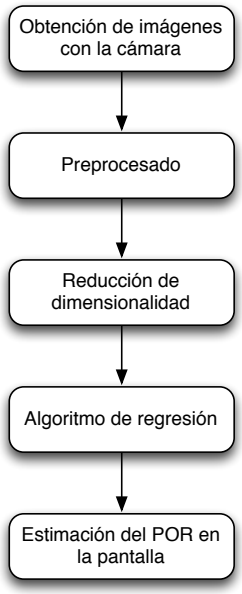


Figura 2.1: Metodología general seguida en el trabajo.

El primer paso consiste en captar las imágenes con el tipo de cámara que estemos trabajando. Debido a que la información importante para estimar el POR se centra en

los ojos, debemos realizar un preprocesado adecuado para resaltar esta información y eliminar el resto. Esta información podría usarse directamente para estimar la regresión. Sin embargo, existen algunos inconvenientes al usar esta representación directa: la alta dimensionalidad y correlación que existe en el vector de características. Por ello aplicamos una reducción de dimensionalidad que solucione estos problemas. Una vez tenemos una representación adecuada de los datos podemos utilizarlos para estimar la regresión con los algoritmos propuestos. El resultado de estos algoritmos nos dará una estimación del POR en la pantalla según donde estén mirando nuestros ojos.

2.2. Entorno de adquisición

La estimación de la mirada viene afectada en gran medida por los errores que se pueden producir debido a cambios de pose de la cabeza de la persona. Para obtener unos resultados fiables hemos evitado los movimientos de la cabeza contruyendo una estructura parecida a un oftalmómetro, y fijando la cabeza a ella. El sujeto sobre el que se van a hacer las pruebas coloca su cabeza en el aparato evitando prácticamente cualquier movimiento. Las condiciones de iluminación no están controladas. En la Figura 2.2 podemos ver varias imágenes que describen el sistema. En ellas se aprecia la estructura para fijar la cabeza, así como la disposición tanto de la persona como de la cámara y la pantalla. Los ojos se sitúan a $52cm.$ de la pantalla, aproximadamente.

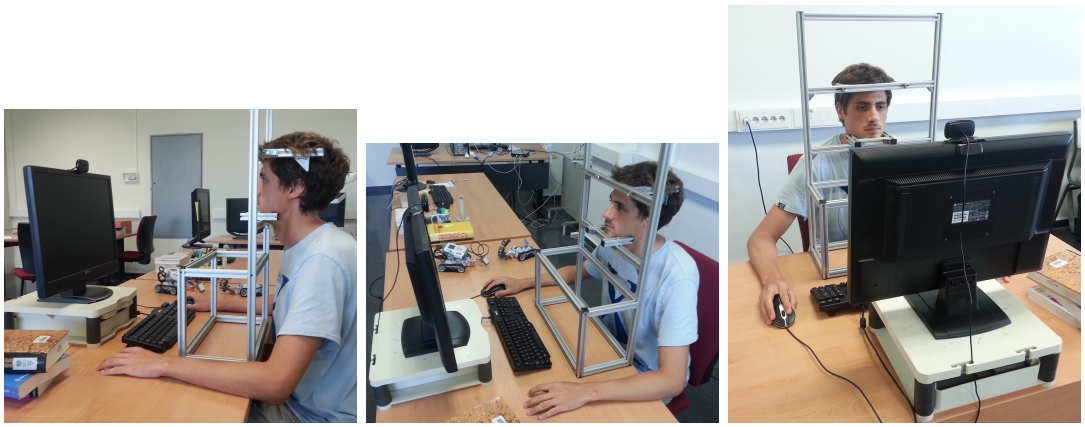


Figura 2.2: Sistema creado para realizar el estudio manteniendo la pose controlada. Notar que la iluminación así como otros aspectos del sistema no están controlados.

2.3. Detección de ojos y preprocesado

La detección de ojos es fundamental para la estimación del POR. Dada la imagen que proviene de la cámara debemos utilizar un método, lo más exacto posible, para saber la posición de los ojos en la imagen. Para ello hemos utilizado la librería que nos proporciona Jason M. Saragih [31] como sistema de *face tracking* en tiempo real. Esta librería es la implementación de un algoritmo que consiste en ajustar un modelo deformable formado por puntos (*Point Distribution Model*) a un objeto concreto, en este caso la cara. Esta técnica se basa en la estimación de detectores locales (*Constrained Local Models*). Gracias a una serie de estrategias de optimización, explicadas en el artículo y basadas en la simplificación de la distribución de los puntos de interés, podemos obtener un modelo de puntos que se ajusta bastante bien a la cara. En la Figura 2.3 podemos ver los puntos de interés del modelo comentados con anterioridad.

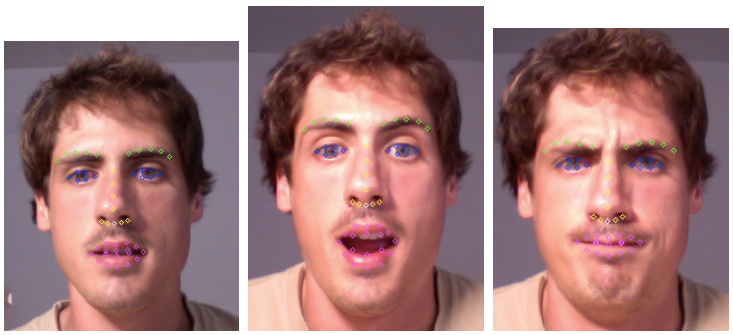


Figura 2.3: Ejemplos de ajuste del modelo a distintas caras de una misma persona.

Debido a que tenemos acceso a la posición de cada uno de los puntos, podemos calcular la posición de cada ojo obteniendo la media de las posiciones de los puntos de interés que lo conforman.

Una vez tenemos la imagen captada con la cámara y la posición de los ojos, realizamos un preprocesado para captar la información representativa en la estimación del POR. Para ello, realizamos un alineamiento de los mismo mediante una transformación afín, obteniendo una imagen normalizada. De esta forma colocamos los ojos en posiciones conocidas, $(50, 50)$ y $(150, 50)$, en una imagen de 200 x 100 pixels. Sobre la imagen normalizada recortamos los ojos y aplicamos una normalización en intensidad dejando los valores con media 0 y varianza unidad. El valor de los pixels de la imagen con los dos ojos formará el vector de características con el que vamos a trabajar. En la Figura 2.4 podemos ver un diagrama del preprocesado de los datos, desde la imagen original captada con la cámara hasta la imagen con los dos ojos normalizados. Dicha imagen con los dos ojos contiene la información necesaria para estimar el POR.

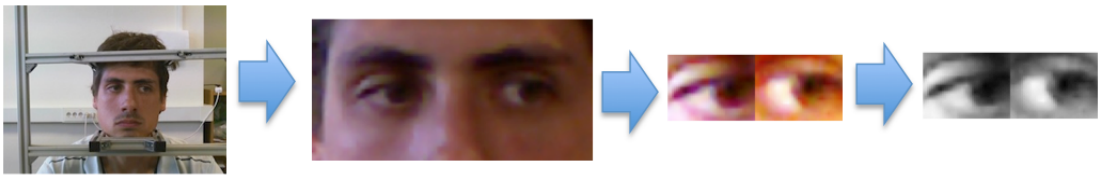


Figura 2.4: Preprocesado de la imagen.

2.4. Cámaras utilizadas

Uno de los objetivos previstos en el trabajo es comparar el resultado obtenido utilizando dos tipos de cámaras de consumo. Por un lado, hemos realizado las pruebas con una *webcam* marca *Logitech* modelo *HD Webcam C525*. En la Tabla 2.1 mostramos las especificaciones técnicas de la cámara y una imagen de la misma [23]:

Tipo de sensor	Color
Resolución	1280 x 720 píxeles
Audio	Micrófono integrado con tecnología Logitech RightSound
Enfoque	Automático
Conexión	USB 2.0
Sensor	8 megapíxeles

Cuadro 2.1: Especificaciones técnicas de la cámara *HD Webcam C525*.

Figura 2.5: Cámara *HD Webcam C525* de *Logitech*.

Por otro lado hemos utilizado el dispositivo *Kinect* de *Microsoft*. Este dispositivo nos da información de profundidad gracias a su sensor de infrarrojos, además de la imagen a color RGB. En la Tabla 2.2 mostramos las especificaciones técnicas del dispositivo y una imagen del mismo [26]:

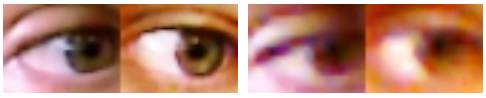
Para comprobar la diferencia cualitativa entre las imágenes captadas con una cámara y la otra, en la Figura 2.7 vemos dos ejemplos de las imágenes de los ojos obtenidas con cada cámara.

Como se puede apreciar, la calidad de la imagen es superior en la cámara *Logitech*. Esto es debido, principalmente, a la mayor calidad del sensor y mejore resolución como

Tipo de sensor	Color y profundidad
Resolución de color VGA	640 x 480 pixels @30FPS
Resolución de profundidad	640 x 480 pixels @30FPS
Audio	*Array* de 4 micrófonos
Motor	Ajuste vertical del sensor
Rango de profundidad	1.2 m hasta 3.5 m
Campo de visión horizontal	57 grados
Campo de visión vertical	43 grados
Rango de inclinación física	27 grados

Cuadro 2.2: Especificaciones técnicas del dispositivo *Kinect*.

Figura 2.6: Dispositivo *Kinect* de *Microsoft*.



(a) Logitech (b) Kinect

Figura 2.7: Ejemplos de ojos obtenidos con los dos dispositivos propuestos.

se puede apreciar en las características.

2.5. Creación de la base de datos

La mayoría de técnicas basadas en la apariencia necesitan un conjunto de datos con los que entrenar los algoritmos que estiman la regresión, así como muestras de test para comprobar los resultados. Los algoritmos que vamos a utilizar en nuestro método requieren, también, estos dos conjuntos de datos. Además, también necesitamos un tercer conjunto para ajustar los parámetros de los algoritmos. A este conjunto lo llamaremos *tune*, y ha de ser diferente al de test. Debido a una serie de motivos hemos decidido crear nuestra propia base de datos. Una de las razones es que queremos comparar el funcionamiento con distintas cámaras de consumo, por lo que necesitamos imágenes tomadas con cada tipo de cámara. La otra razón es que no hemos encontrado

en la literatura una base de datos que se ajuste a las condiciones bajo las que se hacen las pruebas.

En la Tabla 2.3 podemos ver un resumen de las características principales de la base de datos que hemos creado.

	número de personas	imágenes por persona	imágenes por posición	número de posiciones	distribución de posiciones
train	7	360	30	12	rejilla
test	7	150	15	10	aleatoria
tune	7	150	15	10	aleatoria

Cuadro 2.3: Tabla resumen con las características principales de las base de datos creada.

De los siete candidatos seleccionados, había cinco de ellos que utilizaban gafas. Por ello hemos obtenido las imágenes tanto con como sin gafas. Para captar las imágenes solicitamos a la persona que mire fijamente a la pantalla, sobre la cual irán apareciendo una serie de círculos donde fijar la atención. Para la fase de obtención de imágenes de entrenamiento hemos seguido una distribución de rejilla. En concreto hemos cubierto cuatro posiciones horizontales y tres verticales, sobre una pantalla de 20 pulgadas con una resolución de 1680 x 1050 pixels. El círculo que hemos utilizado es de 20 pixels de diámetro. En la Figura 2.8 vemos las doce posiciones utilizadas en la rejilla.

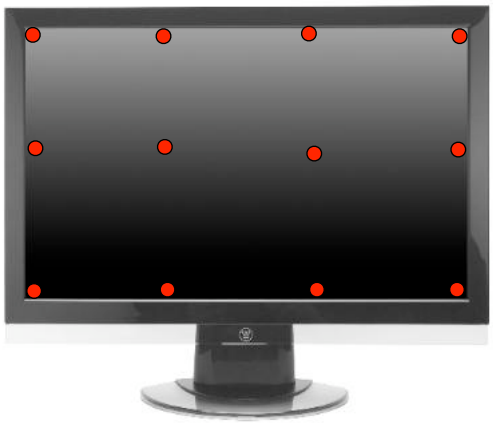


Figura 2.8: Rejilla utilizada para la obtención de imágenes de entrenamiento.

En la Figura 2.9 podemos ver el resultado de las imágenes de los ojos obtenidas para cada una de las doce posiciones de la rejilla para uno de los sujetos de la base de datos.

Figura 2.9: Imágenes de ojos para cada posición de la rejilla.

Junto a las propias imágenes también guardamos la posición del círculo en la pantalla, en unidades de pixels. De esta forma tenemos información suficiente para poder hacer la regresión que nos estime la posición de la pantalla donde estamos mirando a partir de la imagen de nuestros ojos. Debido a que la fase de entrenamiento es específica para cada persona, el uso de esta rejilla nos proporciona un compromiso entre tener suficientes datos relevantes para la regresión y que el proceso de obtención de imágenes de entrenamiento no sea excesivamente pesado. Si utilizamos una rejilla más densa y más imágenes por posición se obtendrán mejores resultados, pero el proceso es mucho más cansado para la persona.

Una vez tenemos las imágenes de entrenamiento procedemos a obtener las de test y las de *tune*. Para ello realizamos un proceso similar. En este caso no utilizamos una estructura de rejilla, sino que los puntos van apareciendo aleatoriamente en la pantalla. De esta forma obtenemos posiciones más reales sobre las que hacer la regresión, y no que coincidan necesariamente con puntos de la rejilla. De igual forma que antes nos guardamos las coordenadas (x, y) del punto en la pantalla.

Notar que todo este proceso se ha realizo para las dos cámaras estudiadas.

2.6. Reducción de la dimensionalidad

Una vez tenemos las imágenes de los ojos podemos proceder a reducir su dimensionalidad. Nuestro vector de características está formado por una imagen que contiene los dos ojos. Este vector es de tamaño 82 x 31 pixels, lo que equivale a un vector de 2.542 dimensiones. Debido a la alta dimensionalidad y a la correlación del vector de características hemos decidido reducir la dimensión de los datos. Existen multitud de técnicas de reducción de dimensionalidad en la literatura [12]. Por su sencillez y facilidad de aplicación hemos decidido utilizar una técnica muy común llamada *Principal Component Analysis* (PCA) [25]. PCA busca una transformación lineal que proyecte los datos en un nuevo sistema de coordenadas ortogonal. El primer eje de dicho sistema

contiene la máxima varianza de los datos, la segunda varianza más grande se representa en el segundo eje, y así sucesivamente. Los ejes del sistema se llaman componentes principales, y podemos obtener tantas como dimensiones tengan los datos originales.

Dentro de la reducción de dimensionalidad hemos dividido dos partes diferenciadas. Por un lado, tenemos que crear una matriz de proyección. Para ello aplicamos PCA sobre las muestras de entrenamiento de cada persona, obteniendo la matriz de proyección a partir de los autovectores. Los vectores de esta base ortogonal forman los "auto-ojos" obtenidos con el PCA. Por otro lado, una vez que tenemos la matriz de proyección podemos proyectar las muestras, tanto de entrenamiento como de test y de *tune*.

Debido a que proyectamos los datos a un subespacio de menor dimensión formado por las componentes de mayor varianza, es de esperar que las muestras queden más separadas en el espacio de representación. Para demostrar esto podemos ver la Figura 2.10, donde representamos las muestras de entrenamiento de un sujeto de la base de datos una vez proyectadas con PCA sobre las dos primeras componentes principales. Cada color y forma geométrica representa el conjunto de imágenes que se ha tomado para cada una de las doce posiciones de la rejilla de entrenamiento.

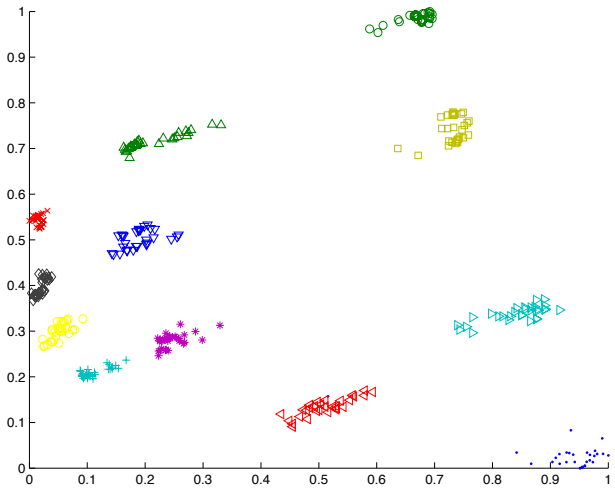


Figura 2.10: Representación de las muestras de entrenamiento una vez proyectadas con PCA sobre sus dos primeras componentes principales.

Como podemos apreciar, las muestras quedan bastante separadas en el espacio, aunque solo hayan sido proyectadas sobre sus dos componentes principales. En nuestro trabajo no hemos fijado el número de dimensiones a proyectar sino que elegimos el porcentaje de energía de la señal original que queremos mantener, una vez proyectada. Después de realizar algunas pruebas hemos decidido mantener el 95 % de la energía,

dando como resultado un nuevo vector de datos proyectado de 18 dimensiones (2.542 era la dimensión original). Con este valor conseguimos reducir bastante la dimensión de la señal, además de eliminar las componentes con menor energía que únicamente aportan ruido.

2.7. Técnicas de regresión

Una vez reducida la dimensionalidad de los datos, la representación de los mismos (ojos) está formada por un vector de características de 18 valores. Las etapas que hemos comentado hasta ahora hacen referencia al preprocesado de los datos, no mencionando por el momento el concepto de regresión. La representación normalizada de los datos nos servirá como entrada para los algoritmos que vamos a estudiar.

El problema de la clasificación consiste en asignar una etiqueta (clase) discreta a una muestra de entrada desconocida. En el caso de la regresión la clase predicha no es un valor discreto, sino un valor continuo. Por lo tanto, el objetivo es ajustar los parámetros del modelo que predizca el valor de unas variables dependientes de salida (POR) según la variación de unas variables independientes de entrada (representación normalizada de los ojos). Tanto las variables de entrada como de salida pueden ser multi-dimensionales. En nuestro caso, ya hemos comentado que el vector de características de entrada tiene 18 dimensiones, mientras que el valor predicho corresponde a una posición en la pantalla (2 dimensiones).

Existen multitud de técnicas de regresión, tanto lineales como no lineales. Las técnicas de mínimos cuadrados [4] utilizan un regresor lineal que minimiza el error sobre las muestras de entrenamiento. El problema de las técnicas lineales es que la mayoría de problemas que se intentan solucionar tienen un comportamiento no lineal. Otra técnica que tiene bastante éxito es RANSAC [11] [15], la cual consiste en un método iterativo y no determinista que intenta utilizar las muestras representativas para la regresión (*inliers*) y no tener en cuenta los datos que interfieren en el resultado (*ourliers*).

En este trabajo hemos elegido tres técnicas que permiten hacer la regresión en el problema dado: *K-Nearest Neighbor Regression* (KNNR), *Support Vector Regression* (ϵ-SVR) [9] y *Random Forest Regression* (RFR) [5]. A continuación explicamos más en detalle las tres técnicas y cómo las hemos utilizado en nuestro problema de estimación del POR.

2.7.1. K-Nearest Neighbor Regression

K-Nearest Neighbor Regression (KNNR) es un método supervisado para clasificar muestras en base a la cercanía a los ejemplos de entrenamiento en el espacio de representación. De la misma forma que podemos predecir la clase discreta de la nueva muestra desconocida, también podemos estimar un valor continuo de salida estimando la función de regresión. El parámetro **k** del modelo establece el número de muestras vecinas que hay que tener en cuenta en la votación. Si $k = 1$ el valor estimado coincidirá con el valor que tenga la muestra más cercana. En nuestro caso vamos a utilizar un valor mayor que 1, por lo que tenemos que decidir la forma en la que combinamos la aportación de cada vecino para obtener el valor final. La forma más común, y la que hemos utilizado, es realizar la media aritmética del valor de los **k** vecinos más cercanos. Este valor medio es la regresión estimada. También hemos ponderado la contribución de cada vecino según su distancia a la muestra de interés. Concretamente, el peso utilizado es la inversa de su distancia al cuadrado. De esta forma, los vecinos cercanos contribuyen con más importancia en el cálculo de la media, y menos los lejanos. Según esto, el valor medio de regresión se puede calcular como

$$y = \frac{1}{N} \sum_{k=1}^{N} \frac{1}{{d_k}^2} x_k$$

siendo N el número de vecinos comparados, x_k el valor asociado al vecino k, d_k la distancia entre la muestra desconocida y el vecino k. Para utilizar este algoritmo debemos calcular la distancia entre dos vectores. Hay distintos tipos de distancia que se pueden utilizar. Una de ellas es la **L0**, que se calcula según la fórmula

$$d(\mathbf{a}, \mathbf{b}) = \max_{1 \leq i \leq d} |(a_i - b_i)|$$

siendo **a** y **b** los vectores sobre los que se mide la distancia. Otra distancia típica es la **L1**, también conocida como *Manhattan*.

$$d(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{d} |(a_i - b_i)|$$

Finalmente la más común es la **L2**, que corresponde con la euclídia.

$$d(\mathbf{a}, \mathbf{b}) = (\sum_{i=1}^{d} (a_i - b_i)^2)^{1/2}$$

2.7.2. Support Vector Regression

Support Vector Regression (ϵ-SVR) [9] es un algoritmo supervisado de aprendizaje de modelos tanto para clasificación como para regresión. El algoritmo básico es un clasificador lineal binario al que se le suministran muestras de entrenamiento etiquetadas con la clase que corresponda. El modelo a estimar intenta buscar un hiperplano que separe de forma óptima las muestras de cada clase. Esto es maximizando el hueco (margen) que hay entre las muestras más cercanas al plano. En la Figura 2.11 podemos ver este hecho.

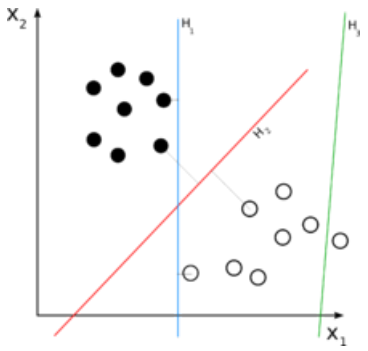


Figura 2.11: Separación de dos clases de muestras con distintos hiperplanos separadores. El hiperplano verde no separa las muestras. El hiperplano azul sí que las separa, pero no de forma óptima. El hiperplano rojo separa las muestras de forma óptima ya que está situado a la máxima distancia de las dos muestras más cercanas al plano (margen).

Además de trabajar con problemas lineales podemos abordar no linealidades a través del **truco del kernel**. Gracias a este truco podemos transformar el espacio original en otro espacio en el que sea más fácil separar las muestras, sin tener que transformar las mismas explícitamente. Los kernels más comunes son

- Polinómico: $k(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c)^d$
- Gausiano (*Radial Basis Funtion*): $k(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma ||\mathbf{x}_i - \mathbf{x}_j||^2)$
- Tangente hiperbólica: $k(\mathbf{x}_i, \mathbf{x}_j) = tanh(k\mathbf{x}_i \cdot \mathbf{x}_j + c)$

Posteriormente se han propuesto algunas modificaciones del algoritmo original para tratar problemas multi-clase o de regresión. Existen varios algoritmos para realizar regresión con SVM. Entre ellos hemos elegido *Support Vector Regression* (ϵ-SVR), propuesto por [9]. A pesar de que se usan los mismo principios que en SVM, existen algunas diferencias. Debido a que la salida del algoritmo es un valor continuo la función de error a minimizar es substituida por la ϵ-*insensitive error function*, la cual da error 0 si la diferencia entre el valor predicho y el real es menor que un parámetro llamado ϵ, donde $\epsilon > 0$

$$E_\epsilon(z) = \begin{cases} 0 & \text{si } z < \epsilon \\ |z| - \epsilon & \text{en otro caso} \end{cases}$$

Según esta condición, el problema típico de minimización que hay que resolver en SVM queda

$$\underset{\mathbf{w},w_0,\boldsymbol{\xi},\boldsymbol{\xi}^*}{min} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

sujeto a una serie de condiciones. Podemos transformar este problema de optimización en un problema dual y resolverlo mediante la técnica de los multiplicadores de Lagrange. Una vez resuelto podemos utilizar la función de estimación aplicándola a cada muestra de entrada.

2.7.3. Random Forest Regression

Random Forest Regression (RFR) [5] es un algoritmo que utiliza un conjunto de modelos formados por árboles de decisión para predecir un valor de salida. En los últimos años esta técnica ha tenido mucho éxito debido a una serie de características [8].

1. Podemos realizar tanto clasificación como regresión, así como *manifold learning*, *semi-supervised learning* y *active learning*.

2. Puede trabajar con problemas supervisados o no supervisados.

3. Podemos obtener gran precisión en los resultados gracias al fenómeno de la generalización, en el cual muchos árboles ligeramente diferentes aportan información para obtener una medida más fiable ante muestras no vistas.

4. Tanto el entrenamiento como el test pueden pueden implementarse en paralelo fácilmente.

Como hemos dicho antes, los *Random Forest* se basan en un conjunto de árboles de decisión. Un árbol de decisión es una colección de nodos y aristas organizados de forma jerárquica. Los nodos pueden ser internos (*splits*) o nodos terminales (hojas). Una muestra entra por arriba del árbol y es sometida a una serie de tests binarios en cada nodo hasta que llega a una hoja, en la que se encuentra la respuesta. Por lo tanto esta estructura podemos pensarla como una técnica para dividir un problema complejo en un conjunto de problemas más simples. Podemos ver estos conceptos de forma gráfica en la Figura 2.12.

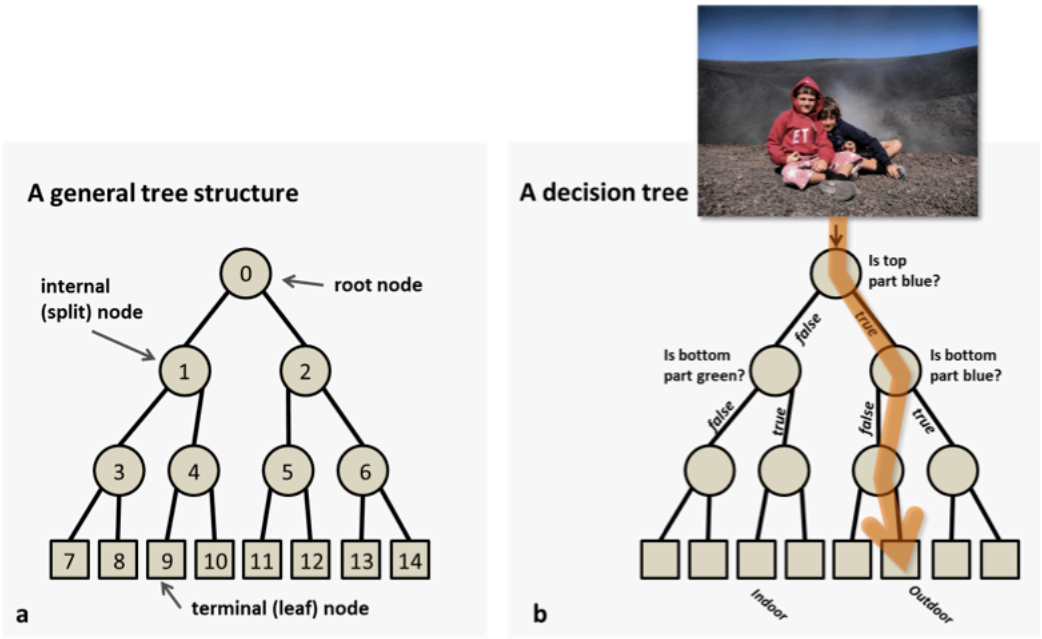


Figura 2.12: **Árbol de decisión. (a)** Estrutura de un árbol de decisión, donde podemos ver los nodos y las aristas **(b)** Ejemplo de clasificación de una imágen, la cual es sometida a un test binario en cada nodo hasta llegar a la hoja solución. [8]

En la fase de entrenamiento el algoritmo intenta optimizar los parámetros de las funciones de *split* a partir de las muestras de entrenamiento

$$\boldsymbol{\theta}_j^* = \underset{\boldsymbol{\theta}_j \in \tau_j}{argmax} I_j$$

Para ello utilizamos la siguiente función de ganancia de información

$$I_j = H(S_j) - \sum_{i \in 1,2} \frac{|S_j^i|}{|S_j|} H(S_j^i)$$

donde S representa el conjunto de muestras que hay en el nodo a dividir, y S^i son los dos conjuntos que se crean de la escisión. La función $H(S)$ mide la entropía del conjunto, y depende del tipo de problema que abordamos. En el caso de la regresión utilizamos funciones de distribución de probabilidad continuas, llegando a la expresión

$$I_j = \sum_{\boldsymbol{v} \in S_j} log(|\boldsymbol{\Lambda_y}(\boldsymbol{v})|) - \sum_{i \in 1,2} \left(\sum_{\boldsymbol{v} \in S_j^i} log(|\boldsymbol{\Lambda_y}(\boldsymbol{v})|) \right)$$

donde $\boldsymbol{\Lambda}_y$ la matriz de covarianza condicional.

Por último comentar el tipo de características que podemos utilizar a la hora de buscar el mejor *split* del nodo. En nuestro caso vamos a utilizar simples clasificadores lineales binarios. Para cada nodo j

$$h(\mathbf{v}, \boldsymbol{\theta}_j) \in 0, 1$$

donde $\mathbf{v}$ es un vector que representa la muestra de entrada y $\boldsymbol{\theta}_j$ son los parámetros a optimizar en el nodo j. Concretamente vamos a comparar si un cierto valor es mayor o menor que un umbral. Estos valores deben ser optimizados en cada nodo realizando una búsqueda exhaustiva de las combinaciones. Sin embargo, es muy importante encontrar un compromiso entre esta optimización y mantener la aleatoriedad de algunos parámetros del algoritmo. Por un lado, puede ocurrir que no hayamos buscado suficientes combinaciones y los *splits* sean demasiado pobres. Por otro lado, debemos mantener cierta aleatoriedad en los parámetros para no obtener árboles correlados y perder el efecto de a generalización.

Finalmente comentar cómo obtenemos un resultado final a partir del resultado que nos proporciona cada árbol. Cada hoja de cada árbol en la que cae la muestra no vista se puede ver como una predicción (distribución de probabilidad). Simplemente haciendo una media aritmética de cada distribución obtenemos el resultado que nos proporciona el *forest* completo.

2.8. Filtro de Kalman

Debido a que uno de los objetivos del trabajo es implementar una aplicación que estime el POR en tiempo real, hemos decidido filtrar el resultado debido al ruido asociado de la medida en cada instante. El filtro de Kalman [20] es un algoritmo que utiliza las medidas observadas en instantes anteriores, y produce estimaciones de salida que tienden a ser más precisas que las simples medidas. La característica principal de estas medidas es el ruido que llevan asociado. En la Figura 2.13 podemos ver un diagrama con las principales etapas que se llevan a cabo.

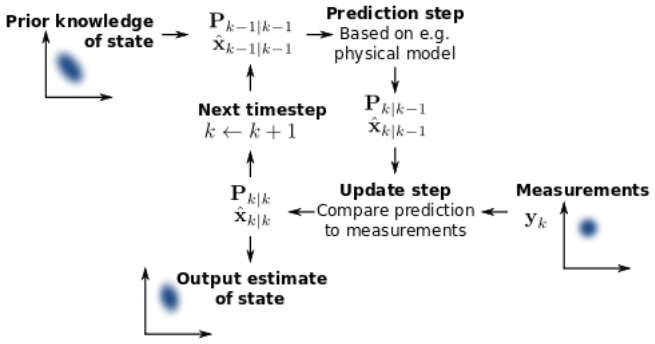


Figura 2.13: Funcionamiento del filtro de Kalman.

Como podemos ver, el algoritmo consta de dos etapas. En la fase de predicción, el filtro predice una valor estimado de las variables asociadas al estado actual, así como de

la incertidumbre. $\hat{a}_{k|k-1}$ representa la estimación del estado del sistema en el instante k, dada información del estado anterior. $P_{k|k-1}$ es la correspondiente incertidumbre. Una vez tenemos la medida ruidosa del instante k, podemos actualizar la estimación inicial produciendo una valor mucho más preciso. El algoritmo funciona en tiempo real utilizando en cada instante la información del estado anterior.

La versión básica del algoritmo supone que el funcionamiento del sistema se basa en un sistema dinámico lineal, y que la incertidumbre de los errores sigue una distribución gausiana conocida. Otras extensiones, como el *Extended Kalman Filter* o el *Unscented Kalman Filter*, pueden trabajar con sistemas no lineales.

En nuestro caso concreto, vamos a utilizar el filtro de Kalman en la implementación de una aplicación en tiempo real que estima la posición de nuestra mirada sobre una pantalla. Los valores obtenidos son medidas ruidosas debido a variaciones que puede haber en la captación de imágenes. Por otro lado, podemos modelar el movimiento de la mirada en la pantalla como un sistema dinámico lineal, por lo que podemos predecir la siguiente posición sabiendo la velocidad y la dirección de avance. Teniendo una estimación de la incertidumbre de la medida y de la estimación tenemos todo lo necesario para utilizar el filtro de Kalman. Como resultado obtenemos una salida menos ruidosa y más precisa, como veremos en el capítulo de resultados.

Capítulo 3

Experimentos y resultados

En esta sección vamos a explicar los experimentos realizados para comprobar las técnicas comentadas anteriormente para la estimación del POR.

3.1. Entrenamiento y ajuste de parámetros

Como hemos comentado anteriormente la última etapa del preprocesado de los datos consiste en reducir la dimensionalidad. Para ello aplicamos PCA al conjunto de datos de entrada obteniendo una matriz de proyección a partir de las muestras de entrenamiento. Sobre esta matriz proyectaremos todas las muestras para reducir su dimensionalidad.

Una vez tenemos completada la fase de preoproceso, podemos probar los algoritmos explicados. Debido a que dichos algoritmos intentan construir modelos basados en parámetros, necesitamos una fase de ajuste de los mismos (*tunning*). Para ello utilizamos la parte de la base de datos que hace referencia a las imágenes de *tune*. Notar que para la fase de *tunning* hemos utilizado, únicamente, las imágenes captadas con la cámara *webcam Logitech*.

Tanto para realizar la optimización de parámetros como para obtener los resultados sobre las muestras no vistas hemos utilizado una medida de error llamada *Mean Absolute Deviation*(MAD), la cual se calcula

$$MAD = \frac{1}{N}\sum_{n=1}^{N} |\boldsymbol{f}_{x_n} - \boldsymbol{f}_\theta(\boldsymbol{x}_n)|$$

siendo N el número de muestras, $\boldsymbol{f}_{x_n}$ el valor real de la posición del círculo, y $\boldsymbol{f}_\theta(\boldsymbol{x}_n)$ el valor estimado por la función de regresión. El error MAD mide el módulo del error que hay entre el valor real y la estimación. Esta medida se calcula para cada sujeto de la

base de datos y luego se obtiene la media para calcular qué combinación de parámetros da el error mínimo.

A continuación explicamos los pasos seguidos para cada algoritmo.

3.1.1. K-Nearest Neighbor Regression

En este caso hay dos parámetros que tenemos que optimizar. Por un lado el valor **k** que representa el número de vecinos con el que comparar. El otro parámetro que tenemos que optimizar es el tipo de distancia que utilizamos. Para la realización de las pruebas hemos modificado el código suministrado por Alfons Juan en algunas asignaturas del Máster IARFID [2].

En la Figura 3.1 mostramos una gráfica con el resultado de la optimización del parámetro **k**. En ella vemos al evolución del error **MAD** según el número de vecinos probados, para distintas distancias.

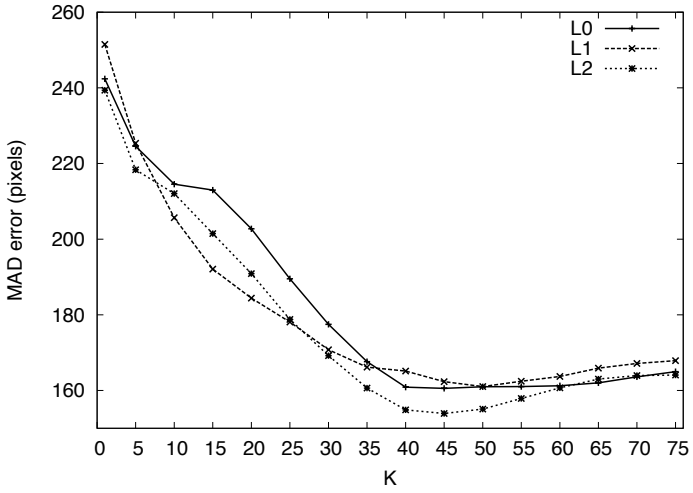


Figura 3.1: Evolución del error MAD con el número de vecinos **k**. Resultados para las distancias L0, L1 y L2.

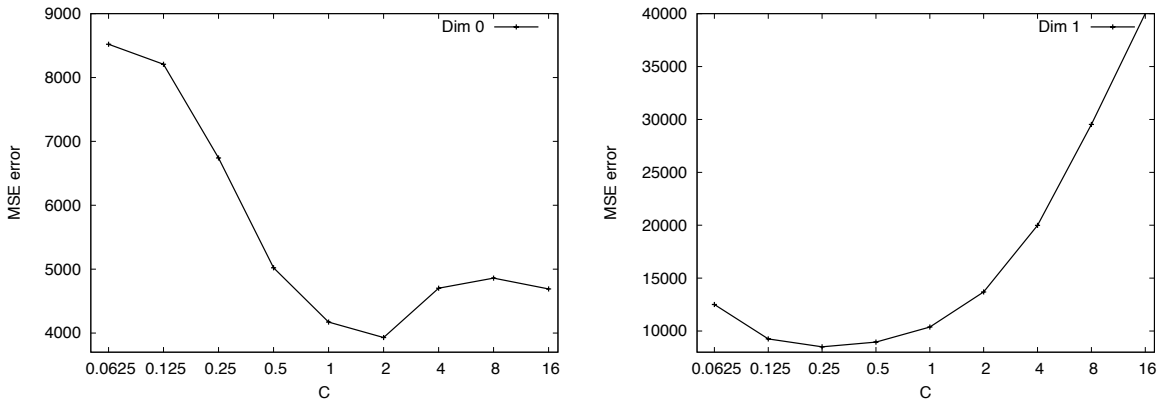
Como podemos apreciar el error MAD disminuye si aumentamos el valor de **k**, para las tres distancia seleccionadas. Al ponderar cada vecino por su distancia, es de esperar que este valor siga bajando conforme aumentemos el valor de **k** (comparar con más vecinos). Sin embargo, en torno a $k = 45$ el error deja de disminuir y se estabiliza, llegando a aumentar ligeramente. Por ello hemos decidido utilizar ese valor de **k** como número de vecinos con el que comparar. Por otro lado, no está claro qué tipo de distancia conviene usar ya que depende mucho del número de vecinos. Según el valor de **k** fijado, hemos decidido utilizar la distancia **L2**, ya que obtiene el mínimo error MAD para 45 vecinos.

3.1.2. Support Vector Regression

De la misma forma que en el método anterior, en este caso también tenemos que realizar una búsqueda exhaustiva para obtener la mejor configuración de parámetros del modelo. Como implementación de la técnica ϵ-SVR hemos utilizado la librería *LIBSVM* [6]. Esta librería requiere que los datos estén en un formato concreto, por lo que el primer paso es transformar nuestro vector de características, tanto para las imágenes de entrenamiento como para las de *tune*.

Otro detalle importante es que la librería *LIBSVM* únicamente estima la regresión sobre un valor unidimensional de salida. Como en nuestro caso la salida es bidimensional (posición x, y en la pantalla) tenemos que estimar una función de regresión por cada dimensión que tengamos. Por lo tanto, el proceso de optimización de parámetros se hace independiente para cada dimensión.

La librería *LIBSVM* nos permite modificar varios parámetros. Podemos elegir distintos tipos de kernel (lineal, polinómico, RBF o sigmoid), así como distintos parámetros para cada uno de ellos. Debido a la cantidad de combinaciones entre los parámetros, únicamente mostramos en la Figura 3.2 la evolución del MAD según los valores del parámetro C (coste), para cada una de las dimensiones. Este parámetro controla la relación entre errores debidos a muestras mal clasificadas y la rigidez del margen.



(a) Primera dimensión

(b) Segunda dimensión

Figura 3.2: Evolución del error MSE dependiendo del parámetro C, para cada una de las dimensiones.

Como podemos ver en las gráficas, en ambos casos existe un valor óptimo para el parámetro C que da el mínimos error. Para la primera dimensión este valor es $C = 2$, mientras que para la segunda dimensión el valor es $C = 0.25$. En referencia al resto de parámetros optimizados, hemos utilizado un kernel polinómico con $C = 1$ y el valor de E fijado a 0.5. Para la primera dimensión utilizamos $G = 0.001$, $r = 25$ y $d = 3$. Para

la segunda dimensión utilizamos $G = 0.01$, $R = 0$, y $D = 4$.

Por otro lado, según *LIBSVM* es conveniente escalar los valores de entrada entre $[-1, 1]$ o $[0, 1]$. Pese a esto, en nuestro problema concreto hemos obtenido mejores resultados utilizando directamente los valores del PCA sin escalar.

3.1.3. Random Forest Regression

En este caso hemos utilizado nuestra propia implementación del algoritmo. Como hemos comentado anteriormente en esta técnica tenemos que construir árboles de decisión buscando en cada nodo del árbol el mejor *split* que nos separe los datos. Según esto nuestro modelo depende de una serie de parámetros que detallamos a continuación.

1. **num_trees**: número de árboles que forman el *forest*.
2. **max_depth**: máxima profundidad que puede alcanzar el árbol.
3. **min_samples**: mínimo número de muestras que debe haber en un nodo para que pueda seguir partiéndose (no se convierta en hoja).
4. **num_th**: número de umbrales aleatorios que se comparan con el resultado del clasificador débil para obtener el resultado binario.
5. **iter**: número de clasificadores débiles binarios que probamos en la optimización de cada nodo.

Debido a la gran cantidad de combinaciones simplemente vamos a enunciar cuál es el mejor resultado obtenido. Sin embargo, la elección del número de árboles es fundamental en esta técnica, debido a que marca, en gran medida, la precisión y el tiempo de cálculo del algoritmo. Por ello en la Figura 3.3 mostramos la evolución del error MAD con el número de árboles utilizados. Notar que cada experimento lo hemos hecho cuatro veces, para obtener una medida más fiable debido al factor aleatorio del algoritmo. Además representamos los resultados para los dos tipos de clasificadores débiles binarios que hemos utilizado. Por un lado, el clasificador **feature** simplemente escoge aleatoriamente un elemento del vector de características y lo compara con un umbral. El clasificador **feature_dif** hace lo mismo, pero el resultado a comparar lo obtenemos de la resta entre dos elementos aleatorios del vector. Notar que los umbrales se escogen aleatoriamente entre el valor mínimo y máximo del resultado de la característica (valor del elemento o resta entre dos elementos del vector) para todas las muestras.

Como podemos ver en la gráfica el error disminuye cuando utilizamos más árboles. Esto concuerda con la teoría, ya que obtenemos distribuciones de probabilidad más

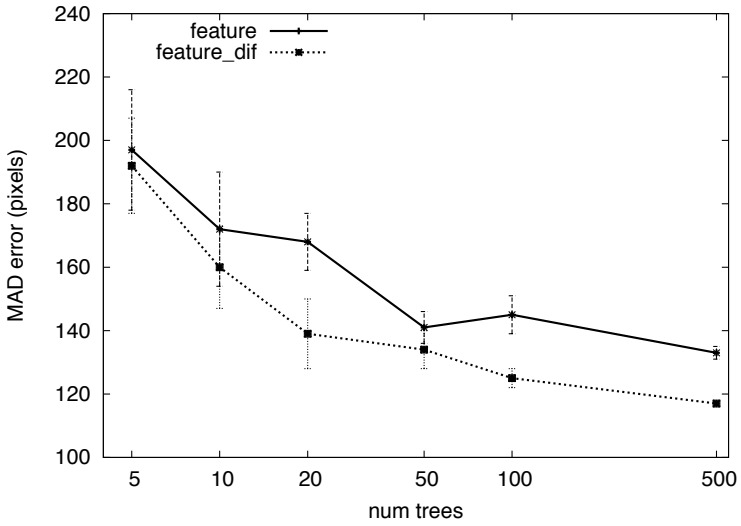


Figura 3.3: Comparación del error MAD dependiendo del número de árboles. Resultados para el clasificador débil **feature** y **feature_dif**.

suaves y cometemos menor error ante muestras no vistas. Por contra, al utilizar más árboles aumentamos el tiempo de cómputo tanto de la fase de entrenamiento como en la de test. También podemos comprobar cómo la desviación típica de la medida disminuye al aumentar el número de árboles. El compromiso que hemos adoptado para realizar las pruebas es utilizar 100 árboles. Por otro lado, podemos ver que la utilización del clasificador débil basado en la resta de dos elementos del vector de características funciona ligeramente mejor que al utilizar simplemente el valor de un elemento del vector. Por lo tanto hemos elegido el primero de ellos. Además de los ya comentados, los parámetros óptimos obtenidos son $min\_samples = 6$, $max\_depth = 11$, $iter = 5$, $num\_th = 5$.

3.2. Test

Una vez finalizada la fase de *tunning* hemos obtenido la mejor combinación de parámetros para cada modelo. El siguiente paso es analizar los resultados que proporciona cada algoritmo sobre muestras nunca vistas. Para ello utilizamos el conjunto de test.

Como hemos comentado en la introducción, uno de los objetivos del trabajo es comparar el funcionamiento del sistema con distintos tipos de cámaras de consumo. Por ello hemos realizado las pruebas con las dos cámaras comentadas en la sección anterior. En la Figura 3.4 mostramos el error MAD medio del conjunto de datos de test para cada algoritmo, según el tipo de cámara utilizada.

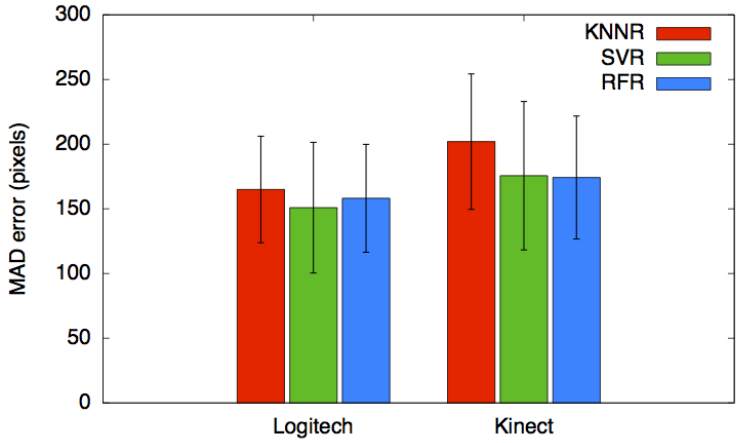


Figura 3.4: Comparación del error MAD en la fase de test con distintas técnicas y las dos cámaras utilizadas.

Como podemos ver en la gráfica, el error cometido con la técnica KNNR es ligeramente mayor en comparación con las otras dos técnicas. Esto puede ser debido al tipo de rejilla utilizada. Al ser poco densa no se obtienen suficientes muestras que representen correctamente el espacio donde hacemos la regresión. Las técnicas SVR y RFR obtienen unos resultados muy parecidos para ambas cámaras, en torno a 150 pixels de error. Este valor nos da una idea de la diferencia medida en pixels del error entre el valor real y el estimado. De forma cualitativa, y dada la resolución de la pantalla utilizada para las pruebas, podemos pensar que seríamos capaces de distinguir 10 zonas horizontales y 7 verticales en la pantalla. Este valor es orientativo debido a las diferencias que puede haber según el sujeto de la base de datos, como puede verse en la desviación típica de las medidas. Comparando los resultados según la cámara utilizada podemos ver que la cámara *Logitech* obtiene mejor resultado que la *Kinect*, aunque la diferencia demasiado grande. Esta diferencia se debe, principalmente, a la menor resolución y peor sensor de la*Kinect*. Notar que a los errores mostrados habría que restar el diámetro del punto donde se mira en la pantalla, el cual tiene 20 pixels de diámetro.

3.3. Ejemplo de aplicación

Para mostrar el funcionamiento del sistema hemos decidido crear una aplicación en tiempo real que estima la posición de la pantalla donde estamos mirando en condiciones similares a las que hemos hecho el estudio. El objetivo de la aplicación es crear un sistema completo de estimación del POR individual para cada persona. Como hemos comentado en la introducción distinguimos una fase de entrenamiento *offline* y una

fase de test *online*. A continuación comentamos cada una de ellas.

3.3.1. Fase de entrenamiento

Esta fase consiste en entrenar nuestro modelo utilizando muestras de entrenamiento obtenidas de la persona que va a utilizar la aplicación. En la Figura 3.5 podemos ver el proceso de entrenamiento.

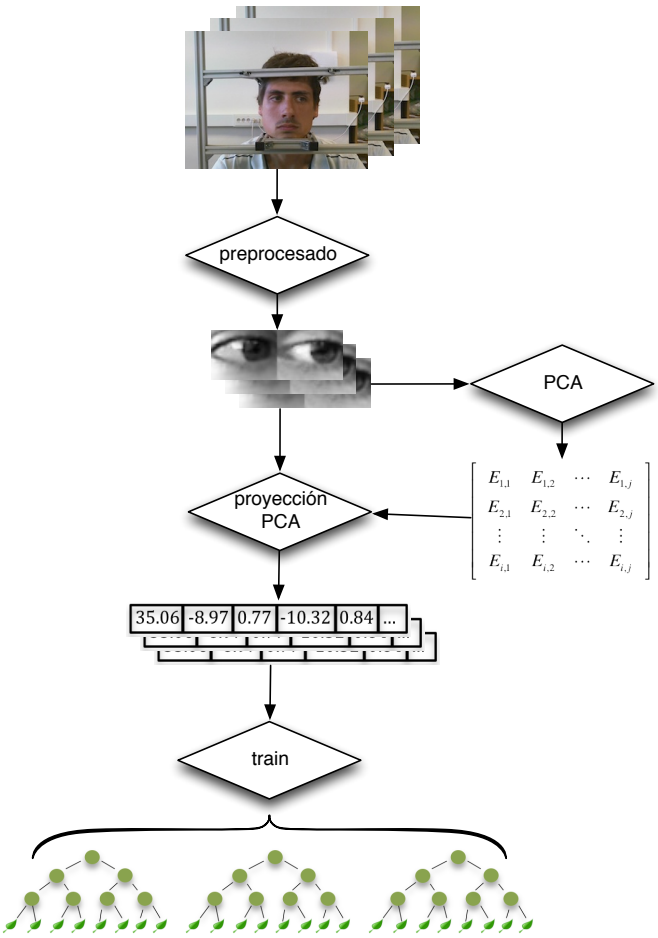


Figura 3.5: Fase de entrenamiento de la aplicación.

Las etapas principales son las que hemos comentado en el capítulo anterior. Primero obtenemos las imágenes con la cámara situando a la persona delante de la pantalla y haciendo que mire a los puntos de la rejilla. Una vez tenemos las imágenes de entrenamiento, aplicamos un preprocesado para obtener las imágenes de los ojos. Seguidamente creamos una matriz de proyección PCA con las muestras de entrenamiento, la cual nos sirve para reducir la dimensionalidad de los datos obteniendo un vector de características. Con estos datos entrenamos el algoritmo seleccionado, que en este caso es *Random Forest Regression* con la mejor configuración de parámetros obtenida en la fase de *tunning* con la base de datos. Hemos elegido este algoritmo porque obtiene los mejores

resultados (junto con SVR), es más sencillo de integrar dentro de la aplicación y va suficientemente rápido.

3.3.2. Fase de test

Una vez tenemos los árboles pasamos a la fase *online* donde se estima el POR de la persona en tiempo real. En la Figura 3.6 vemos las etapas del proceso para cada imagen que se toma de la cámara.

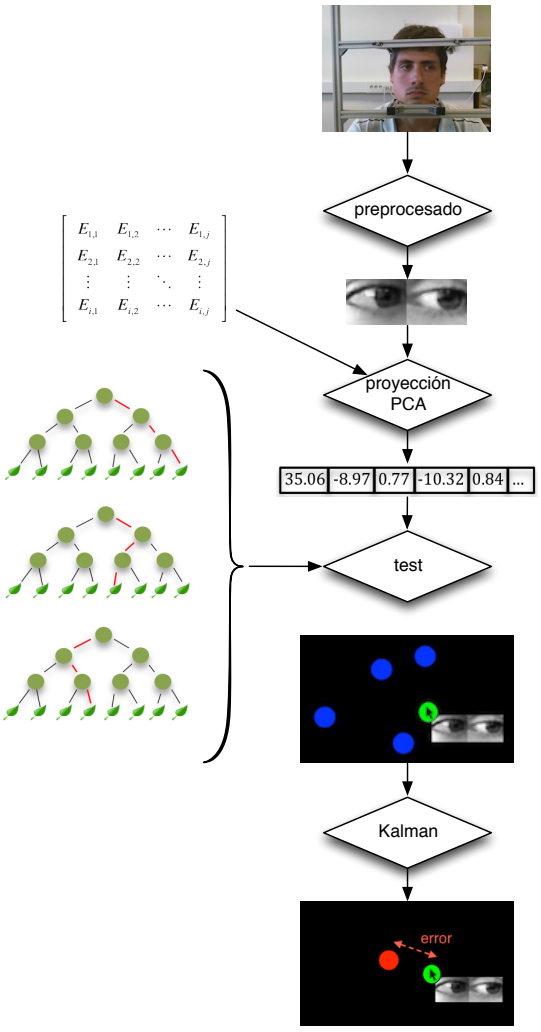


Figura 3.6: Fase de test de la aplicación.

La primera parte del proceso es idéntica a la fase de entrenamiento hasta que obtenemos el vector de características. La única diferencia es que, en este caso, utilizamos la matriz de proyección creada en la fase de entrenamiento para proyectar la muestra actual de test al espacio reducido. Una vez tenemos el vector de características reducido, utilizamos el *random forest* para obtener un valor de regresión. El siguiente paso sería representar en la pantalla el valor estimado. El problema que tenemos es

que nuestra medida es muy ruidosa debido a pequeñas variaciones que pueden aparecer en las imágenes de los ojos. Para eliminar estas variaciones utilizamos un filtro de Kalman. Este filtro suaviza el valor de salida obteniendo una medida mucho menos ruidosa que depende de las medidas en los instantes anteriores. Finalmente dibujamos con un círculo verde la posición real donde estamos mirando, y con un círculo rojo la posición estimada por el algoritmo. La distancia entre los dos puntos representa el error que estamos cometiendo. Algunos resultados del funcionamiento real de la aplicación podemos verlos en la Figura 3.7.

Figura 3.7: Ejemplos del funcionamiento real del sistema

Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

Este trabajo se ha orientado en estudiar posibles técnicas para realizar la estimación del POR. Básicamente hemos propuesto un método para obtener la posición en la pantalla donde estamos mirando. Nuestra técnica se engloba dentro de las llamadas "basadas en la apariencia", donde utilizamos directamente la información de la imagen para estimar la posición. El método presentado tiene una serie de ventajas: no necesidad de calibración de la cámara ni de los elementos que aparecen en la escena, técnica poco intrusiva (una vez liberada la restricción de la cabeza), utilización de cámaras de bajo coste y no necesidad de controlar la iluminación y otros parámetros de la escena, salvo la pose de la cabeza.

Los objetivos propuestos al inicio se han casi al completo. Por un lado hemos construido una base de datos que nos ha permitido realizar un estudio comparativo sobre distintas técnicas de regresión que podemos utilizar. Como hemos comentado en el trabajo, este estudio lo hemos realizado en un ambiente controlado en el que el sujeto tiene fija la cabeza para evitar posibles errores debidos al cambio de pose. Como se comentará más adelante se pretende continuar el trabajo liberando al sistema de esta restricción para poder trabajar en un ambiente totalmente no controlado. Dentro de las técnicas comparadas, *Support Vector Regression* (ϵ-SVR) [9] y *Random Forest Regression* (RFR) [5] obtienen los mejores resultados, siendo estos muy parecidos. K-Nearest Neighbor Regression (KNNR) obtiene peores resultados, debido, seguramente, a la dificultad de estimar la regresión utilizando una rejilla de entrenamiento poco densa obteniendo pocas muestras de entrenamiento representativas. Una vez hemos decidido qué técnica se adapta mejor a nuestro método, hemos creado un sistema de estimación del POR en tiempo real. Pese a no obtener mejores resultados que muchas técnicas más sofisticadas propuestas en el estado del arte, los resultados son suficientemente buenos

para el campo de aplicaciones comentadas en la introducción. Este tipo de aplicaciones no requieren tanta precisión, pero sí necesitan una serie de requisitos que cumple nuestra técnica. Por otro lado también hemos comprobado que el uso de una cámara RGB-D como la *Kinect*, no empeora en exceso los resultados. Las diferencias entre una y otra se deben a la menor resolución y peor sensor de la *Kinect*. Pese a ello, pensamos que podemos utilizar este tipo de cámaras de consumo RGB-D, obteniendo, a la vez, información de profundidad que puede ser muy útil para corregir errores debidos al movimiento de la cabeza.

4.2. Trabajo futuro

El presente trabajo es una aproximación a un método más completo de estimación del POR. Por un lado, el siguiente paso que tenemos que hacer es eliminar la restricción de que la persona tenga la cabeza sujeta. Para ello debemos estimar la pose de la cabeza para corregir el error cometido en la mirada debido a este hecho. Aprovechando el resultado obtenido, pensamos que la utilización de una cámara de profundidad (*Kinect*) es de gran ayuda para completar este proceso. Teniendo la información de la profundidad es más sencillo y exacto estimar el cambio de pose. De esta forma, conseguiríamos que la aplicación pudiera trabajar en un entorno totalmente no controlado. Por otro lado, mejorar el preprocesado de los datos que suministramos a los algoritmos, mejoraría los resultados. Por ejemplo podríamos aplicar un enventanado gausiano a la zona central de los ojos, lo cual nos ayudaría a eliminar información no representativa. Debido a la relación espacial entre las muestras de entrenamiento, sería interesante estudiar otras técnicas de reducción de dimensionalidad que preserven su topología en el espacio original. De esta forma podríamos adaptarnos mejor al sistema creado y obtener mejores resultados.

Bibliografía

[1] C. Ahlstrom, T. Victor, C. Wege, and E. Steinmetz. Processing of eye/head-tracking data in large-scale naturalistic driving data sets. *Intelligent Transportation Systems, IEEE Transactions on*, 13(2):553 –564, june 2012. 8

[2] Alfons Juan ajuan@iti.upv.es. knnc, 2000. 28

[3] Shumeet Baluja and Dean Pomerleau. Non-intrusive gaze tracking using artificial neural networks. Technical report, 1994. 9

[4] Ake. Björck. *Numerical Methods for Least Squares Problems*. Siam, 1996. 19

[5] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. 19, 22, 37

[6] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. 29

[7] P.M. Corcoran, F. Nanu, S. Petrescu, and P. Bigioi. Real-time eye gaze tracking for gaming design and consumer electronics systems. *Consumer Electronics, IEEE Transactions on*, 58(2):347 –355, may 2012. 8

[8] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Technical Report MSR-TR-2011-114, Microsoft Research, October 2011. 22, 23

[9] Harris Drucker, Chris J.C. Burges, Linda Kaufman, Chris J. C, Burges* Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines, 1996. 19, 21, 37

[10] A. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Methods in molecular biology. Springer, 2007. 9

[11] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. 19

[12] Imola Fodor. A survey of dimension reduction techniques. Technical report, 2002. 17

[13] Dan Witzner Hansen and Arthur E. C. Pece. Eye tracking in the wild. *Comput. Vis. Image Underst.*, 98(1):155–181, April 2005. 8, 9

[14] D.W. Hansen and Qiang Ji. In the eye of the beholder: A survey of models for eyes and gaze. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):478 –500, march 2010. 7, 8

[15] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. 19

[16] R. Herpers, M. Michaelis, K. h. Lichtenauer, and G. Sommer. Edge and keypoint detection in facial regions. In *In Killington, VT*, pages 212–217. IEEE Computer Society Press, 1996. 8

[17] T.E. Hutchinson, Jr. White, K.P., W.N. Martin, K.C. Reichert, and L.A. Frey. Human-computer interaction using eye-gaze input. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6):1527 –1534, nov/dec 1989. 7

[18] James P. Ivins and John Porrill. A deformable model of the human iris for measuring small three-dimensional eye movements. *Machine Vision and Applications*, 11:42–51, 1998. 10.1007/s001380050089. 8

[19] Qiang Ji and Xiaojie Yang. Real-time eye, gaze, and face pose tracking for monitoring driver vigilance. *Real-Time Imaging*, 8(5):357–377, October 2002. 9

[20] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. 24

[21] Kin-Man Lam and Hong Yan. An analytic-to-holistic approach for face recognition based on a single frontal view. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(7):673 –686, jul 1998. 8

[22] J. Levine and IBM Research Division. T.J. Watson Research Center. *An Eye-controlled Computer*. Research reports // IBM. IBM Research Division, T.J. Watson Research Center, 1981. 7

[23] Logitech. Logitech hd webcam c525. 14

[24] Hu-Chuan Lu, Guo-Liang Fang, Chao Wang, and Yen-Wei Chen. A novel method for gaze tracking by local pattern model and support vector regressor. *Signal Process.*, 90(4):1290–1299, April 2010. 7

[25] Timmerman M.E. Principal component analysis (2nd ed.). i. t. jolliffe. *Journal of the American Statistical Association*, 98:1082–1083, January 2003. 17

[26] Microsoft. Microsoft kinect. 14

[27] Carlos H. Morimoto and Marcio R. M. Mimica. Eye gaze tracking techniques for interactive applications. *Comput. Vis. Image Underst.*, 98(1):4–24, April 2005. 9

[28] Takehiko Ohno and Naoki Mukawa. A free-head, simple calibration, gaze tracking system that enables gaze-based interaction. In *Proceedings of the 2004 symposium on Eye tracking research & applications*, ETRA '04, pages 115–122, New York, NY, USA, 2004. ACM. 9

[29] F. Pirri, M. Pizzoli, and A. Rudi. A general method for the point of regard estimation in 3d space. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 921–928, Washington, DC, USA, 2011. IEEE Computer Society. 7

[30] M.J. Reale, S. Canavan, Lijun Yin, Kaoning Hu, and T. Hung. A multi-gesture interaction system using a 3-d iris disk model for gaze estimation and an active appearance model for 3-d hand pointing. *Multimedia, IEEE Transactions on*, 13(3):474 –486, june 2011. 8

[31] J.M. Saragih, S. Lucey, and J.F. Cohn. Face alignment through subspace constrained mean-shifts. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1034 –1041, 29 2009-oct. 2 2009. 8, 13

[32] Rainer Stiefelhagen, Jie Yang, and Alex Waibel. Tracking eyes and monitoring eye gaze. In *In Workshop on Perceptual User Interfaces, Ban*, 1997. 9

[33] Kentaro Takemura, Yuji Kohashi, Tsuyoshi Suenaga, Jun Takamatsu, and Tsukasa Ogasawara. Estimating 3d point-of-regard and visualizing gaze trajectories under natural head movements. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, ETRA '10, pages 157–160, New York, NY, USA, 2010. ACM. 7

[34] Kar-Han Tan, D.J. Kriegman, and N. Ahuja. Appearance-based eye gaze estimation. In *Applications of Computer Vision, 2002. (WACV 2002). Proceedings. Sixth IEEE Workshop on*, pages 191 – 195, 2002. 9

[35] Arantxa Villanueva, Rafael Cabeza, and Sonia Porta. Eye tracking: Pupil orientation geometrical modeling. *Image and Vision Computing*, 24(7):663 – 679, 2006. 9

[36] O. Williams, A. Blake, and R. Cipolla. Sparse and semi-supervised visual mapping with the s3gp. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 230 – 237, june 2006. 9