TRABAJO DE FIN DE GRADO

Universidad
Carlos III de Madrid
www.uc3m.es

SISTEMA DE SEGURIDAD BASADO EN TÉCNICAS DE PROCESAMIENTO DE IMAGEN.

AUTOR: RODRIGO HÉCTOR DE LUIS GARCÍA.

PROFESOR: FERNANDO GARCÍA FERNÁNDEZ.

ÍNDICE DE CAPÍTULOS

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Actualmente las alarmas están consideradas una pieza fundamental en nuestra sociedad. Su evolución ha sido progresiva, empezando como sistemas de seguridad que salvaguardaban de robos, atracos o incendios, y llegando a convertirse, hoy en día, en sistemas capaces de garantizar la seguridad de empresas y hogares.

Todo ello ha favorecido la obligatoriedad de su instalación en determinados establecimientos como bancos, cajas de ahorro y entidades de crédito, armerías y joyerías.

En el ámbito social, la inestabilidad económica latente en Europa ha hecho que dichos sistemas de seguridad cobren un papel fundamental, estando presentes en la mayoría de los hogares.

La OCDE, organización que agrupa a 34 países miembros y cuya misión es promover políticas que mejoren el bienestar social y político de las personas, señala que España acumula hasta 420 denuncias debidas a asaltos a viviendas por cada 100.000 habitantes.

En este marco de creciente inseguridad los sistemas de vigilancia han atraído el interés de la población. Según la ACAES, la instalación de sistemas de seguridad en una vivienda unifamiliar oscila de los 2.000 a 5.000 euros y el mantenimiento anual con conexión a una central de alarmas asciende a 500 euros anuales, siendo los sistemas de seguridad de perímetros los más demandados actualmente.

Ante el elevado precio en el que oscilan los sistemas de seguridad conectados a una central, las cámaras de red han cobrado especial relevancia en nuestra sociedad.

En 1996 tuvo lugar la presentación de la primera cámara de red. Durante sus primeros años, la tecnología diseñada no era equiparable con las cámaras analógicas de nivel profesional.

No obstante, con la integración del protocolo TCP/IP en diversas áreas y el creciente desarrollo de las cámaras digitales, las cámaras de red empezaron a contemplarse como posibles sistemas de seguridad.

Hoy en día, las cámaras de red han alcanzado la tecnología de las cámaras analógicas y cumplen los mismos requisitos y especificaciones llegando a adquirir, en numerosas áreas, una mejora del rendimiento respecto a las cámaras analógicas.

El objetivo de este proyecto es el de realizar un sistema de video-vigilancia de bajo coste utilizando técnicas de procesamiento de imagen para detectar a los posibles intrusos del sistema.

Este sistema estará basado en la captación de imágenes a través de una cámara de red colocada en el entorno de vigilancia. Las imágenes serán enviadas a una interfaz Web y a un ordenador de control colocado fuera del entorno de vigilancia. Este ordenador de control llevará a cabo tareas de procesamiento de imagen con el fin de detectar posibles intrusos en el sistema.

1. ESTADO DEL ARTE

En este capítulo se describe el estado del arte relacionado con los sistemas de vídeo-vigilancia actuales.

En el apartado 2.1 "*Cámaras IP"* se dará una descripción general del funcionamiento de este tipo de dispositivos y tipo técnicas que se utilizan actualmente en esta clase de sistemas en el ámbito de video-vigilancia. En el apartado 2.2 "*Técnicas de procesamiento de imagen*" se intentará dar una idea general de cuáles son los objetivos del procesamiento de imagen, y qué tipo de técnicas y algoritmos se suelen utilizar en este ámbito para la detección de movimiento o para el seguimiento de objetos en el procesado.

1.1 Cámaras IP

Una red de vigilancia IP (Figura 1) es una red local de terminales de vídeo que, de la misma forma que envía datos multimedia fuera de la red local, recibe instrucciones para configurarse, controlarse o incluso interactuar con el entorno.

Figura 1: Cámara IP.

La vigilancia digital se ha inclinado de forma natural hacia el Protocolo de Internet por tratarse de un medio idóneo para dicha actividad. El protocolo IP se caracteriza por su versatilidad, ya que no tiene limitaciones de magnitud, así como por su robustez y ubicuidad, pues permite utilizar cada terminal de vigilancia como un nexo con el resto de la red.

Cada cámara o terminal de la red local tiene su propia dirección IP dinámica y capacidad para gestionar la comunicación de forma autónoma a través de la red. Además, todo lo necesario para la obtención de las imágenes a través de la red se encuentra dentro de la misma unidad.

Este tipo de dispositivos se conectan directamente a la red e incorporan firmware propio para servidor web, servidor FTP, cliente FTP y cliente de correo electrónico generalmente. Asimismo, se incluyen entradas para alarmas y salidas de relé. Las cámaras de red más especializadas también pueden equiparse con muchas otras funciones avanzadas como son el análisis de imágenes, detección de movimiento principalmente, o salidas de vídeo analógico.

La red de vigilancia IP incorpora los elementos físicos característicos de una red Ethernet convencional: enrutadores, *hubs*, conmutadores, terminales y cableado de fibra óptica o par trenzado. También se pueden montar sobre una infraestructura inalámbrica parcial o totalmente. Las redes de vigilancia IP complejas, donde existe un gran tráfico de datos y funciones, suelen ser redes Ethernet totalmente conmutadas, es decir, redes full dúplex donde se evita el riesgo de colisión y se puede transmitir simultáneamente en dos sentidos.

En la figura inferior (Figura 2) se muestra el esquema general del funcionamiento de una red TCP/IP con dos terminales de grabación conectados a la misma. Como se puede apreciar en la imagen, gracias al protocolo de red naturalmente implementado para este tipo de cámaras es posible ver los resultados e interactuar con los dispositivos tanto dentro como fuera de la red local.

Figura 2: Esquema de red de una cámara IP.

La grabación de vídeo por parte de las cámaras IP se basa en la conversión de la imagen en una matriz de bits, mediante un proceso fotoeléctrico. Puede ser de dos tipos: CCD (*charge-coupled device*) ó CMOS (*complementary metal-oxide semiconductor*). Los sensores CCD se emplean desde

hace décadas y siguen siendo más eficaces en entornos de baja luminosidad, ya que son más sensibles. En cambio, los chips CMOS tienen toda su lógica integrada, y su bajo coste y volumen, los hace más apropiados para aplicaciones generales y de pequeño tamaño.

En la siguiente figura (Figura 3) se muestra, a modo ejemplificativo, el esquema de señal de un sensor CMOS.

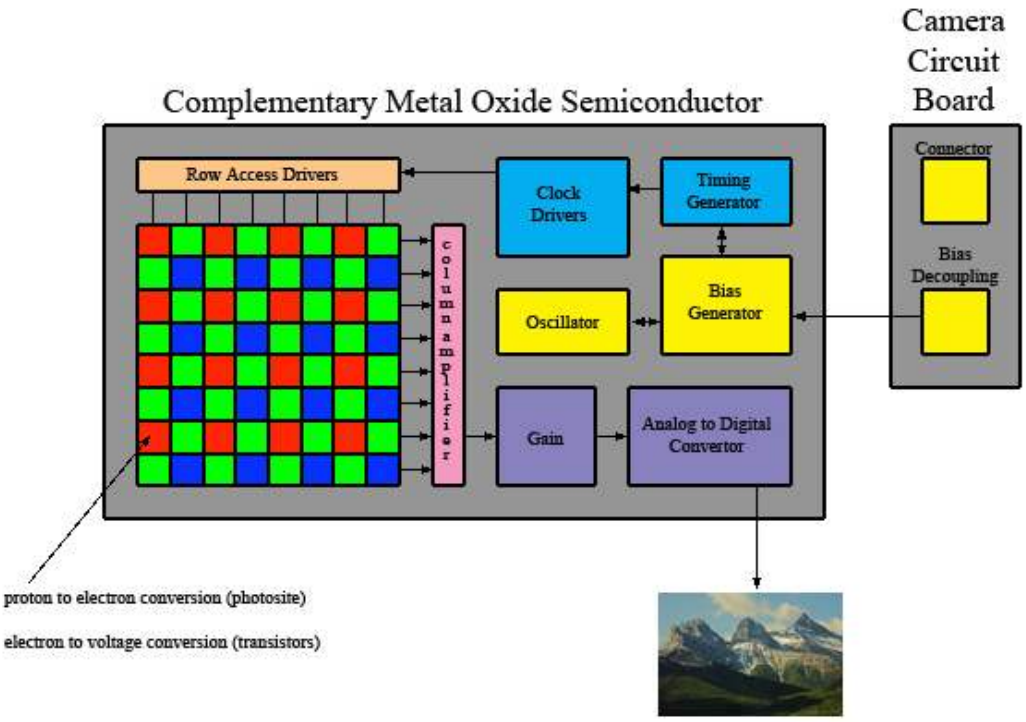


Figura 3: Esquema de señal de un sensor CMOS.

1.1.1 Detección de movimiento por parte de las cámaras IP

Tradicionalmente, las cámaras IP del mercado cuenta con un sensor fotoeléctrico que determina la existencia de movimiento en la secuencia grabada, y de este modo, pueden determinar si es necesario enviar un correo al usuario o guardar la imagen grabada por el sistema.

Un ejemplo de una cámara IP con capacidad para detectar movimiento en el entorno de seguridad se muestra en la imagen inferior (Figura 4). Los elementos indicados en la cámara mostrada suelen estar presentes en la mayor parte de las cámaras IP comerciales.

Figura 4: Elementos de una cámara IP comercial.

Como se aprecia en la Figura 4, los sensores utilizados para la detección de movimiento están basados en la utilización de fotodiodos.

1.1.2 Principales problemas de los fotodiodos

El principal inconveniente de los fotodiodos, en el ámbito de sistemas de seguridad, está relacionado con el hecho de no poder discernir entre peligros reales en el entorno de seguridad y fluctuaciones lumínicas, que se traducirán en variaciones de la corriente generada por los fotodiodos.

Estas fluctuaciones constituyen lo que se denomina ruido. Estas fuentes de ruido serán especialmente perceptibles en condiciones de baja irradiancia, es decir, baja luminosidad en el entorno.

Otro tipo de inconvenientes derivados de este tipo de tecnología es la baja capacidad para discriminar qué constituye un peligro para el sistema, y qué no.

En la Figura 5 se muestra un ejemplo de un entorno real de baja luminosidad, donde se ha captado una imagen que no aporta ningún dato significativo para la seguridad del sistema, ya que los datos recogidos se basan en un cambio de luz brusco a causa del movimiento continuo de las nubes que generan luces y sombras al cruzar por delante del sol.

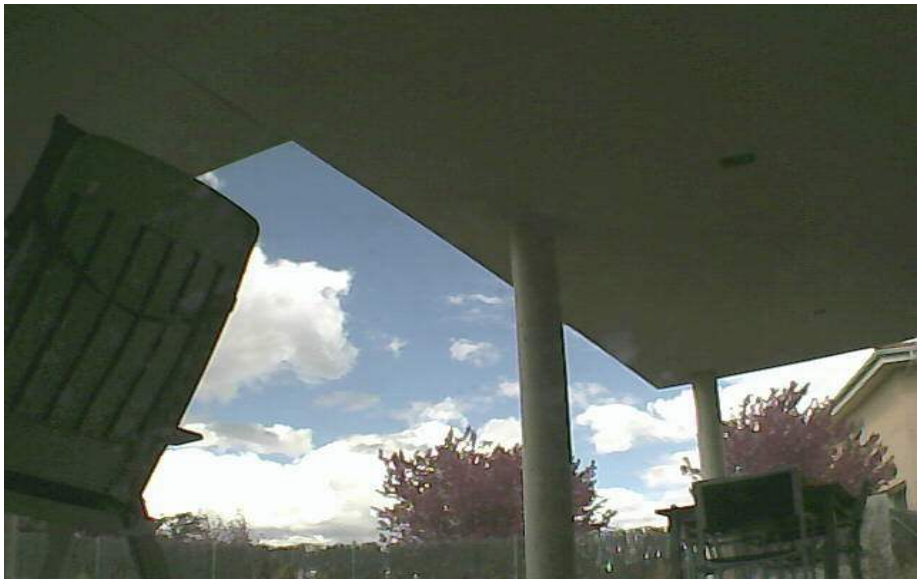


Figura 5: Falso positivo obtenido con una cámara IP FI8918W del fabricante Foscam localizada en un jardín.

En la Figura 6 se muestra un ejemplo de una detección de movimiento que no influye en la seguridad del sistema: un animal ha cruzado por delante de la cámara.

Figura 6: Falso positivo debido a la aparición de un perro en la secuencia obtenido con una cámara IP FI8918W del fabricante Foscam localizada en un jardín.

De este modo, todos los problemas mencionados anteriormente hacen de los fotodiodos sistemas poco fiables para su uso en entornos de seguridad.

1.2 Técnicas de procesamiento de imagen

Con el fin de mejorar los sistemas de vigilancia actuales, en los últimos años el Procesamiento de Imagen Digital ha sido ampliamente utilizado por diversas razones:

-Mejora de la imagen digital con fines interpretativos.

-Toma de decisiones de manera automática de acuerdo al contenido de la imagen digital.

-Mayor capacidad de discriminación entre situaciones de peligro para el sistema, y situaciones que no afectan a la seguridad del sistema.

-Posibilidad de catalagoción de los objetos que entran al sistema.

En este capítulo se van a comentar algunas de las técnicas de procesamiento de imagen que suelen ser empleadas en el entorno de sistemas de seguridad. Asimismo, se comentarán algunas de las técnicas utilizadas para el seguimiento de objetos en un entorno de seguridad.

1.2.1 Seguimiento de objetos

El proceso de seguimiento puede ser en línea, o causal, donde la información disponible para el procesamiento sólo incluye el pasado y el presente, o fuera de línea, o no causal, donde todos los datos procedentes del vídeo están disponibles.

El seguimiento en tiempo real es el nombre dado a los métodos de seguimiento en línea. Su objetivo principal es el procesamiento de la información a medida que llega y la elaboración de un resultado de seguimiento antes de que el siguiente fotograma de vídeo esté disponible. Para los sistemas de vigilancia y otros tipos de aplicaciones, es necesario que el método de seguimiento sea en tiempo real.

Es importante establecer una distinción entre el tiempo real, o en línea, y métodos fuera de línea. Por su naturaleza, los métodos fuera de línea no pueden procesar la información en tiempo real, por lo que no suelen ser útiles para aplicaciones interactivas. No obstante, para las aplicaciones que se pueden realizar después de los hechos, como la indexación de vídeo o la recogida de estadísticas de tráfico, los métodos fuera de línea son una opción atractiva.

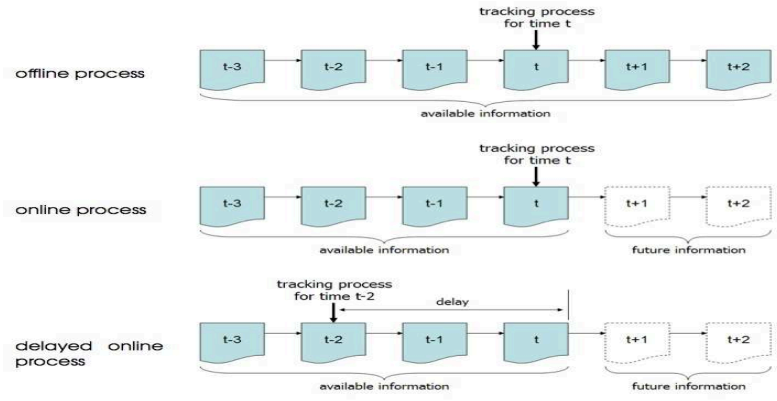


Figura 7: Seguimiento Online y Offline.

Los métodos *offline* o fuera de línea, presentan la ventaja de tener acceso a toda la información de vídeo, incluyendo la información futura, que puede mejorar la fiabilidad del seguimiento.

En la Figura 7 podemos ver como la información disponible varía de un tipo de procesamiento a otro. A pesar del elevado *frame-rate,* o imágenes por segundo, que pueda tener nuestro dispositivo de grabación, siempre existe una penalización en términos de tiempo relacionada con el procesamiento que estemos llevando a cabo de la imagen obtenida.

De este modo, se deberá considerar cuáles son las necesidades del sistema. Se deberá sopesar si es necesario realizar muchas operaciones sobre la imagen obtenida incrementando el *delay,* o tiempo de espera, entre nuestro procesamiento y suceso real, o por el contrario, estimar si es necesario disponer de resultados lo antes posible para poder responder a las demandas del sistema.

1.2.2 Principales dificultades en los sistemas de seguimiento

El cambio de apariencia es un problema difícil e inherente al seguimiento de objetos visuales. En la Figura 8 se muestra un ejemplo de cómo una persona deforma su cuerpo al desplazarse. Los métodos de seguimiento deben tener en cuenta estas posibles variaciones del objeto rastreado.

Figura 8: Objeto de seguimiento deformado al desplazarse.

1.2.3 Modelado de objetos

En un sistema de vigilancia donde se deben seguir intrusos por el entorno, la capacidad de modelar correctamente el objeto que se desea seguir, decidirá de forma directa los resultados obtenidos por el sistema. A la hora de modelar objetos en una secuencia de imágenes se deben escoger correctamente las características del objeto a seguir. Estas características pueden estar relacionadas con:

- La relación del objeto con el fondo de la imagen.
- El color del objeto.
- Forma del objeto.
- La textura del objeto.
- Puntos característicos que definen el objeto.
- Características exclusivas del objeto de seguimiento.

En este subcapítulo se intentará dar una idea general al lector de cómo se pueden modelar las características mencionadas.

1.2.3.1 Modelado de objetos en función del fondo

La substracción de fondo es un método ampliamente utilizado para la detección de objetos en movimiento en secuencias de vídeo obtenidos por cámaras estáticas. El objetivo de esta técnica es el de detectar objetos que se mueven a partir de la diferencia entre la imagen actual y una imagen referencia.

En este tipo de técnica el modelo de fondo, o *background*, debe ser una representación de la escena sin objetos en movimiento y ha de mantenerse actualizada periódicamente con el fin de adaptarse a las condiciones variables de iluminación y otras modificaciones que se puedan dar en el entorno.

La substracción de fondo es una técnica sencilla de implementar y que no requiere gastos computacionales demasiado extensos. En el Capítulo 5.4.2 se analizarán con mayor profundidad este tipo de técnicas.

Figura 9: Resultado de obtenido mediante substracción de fondo.

1.2.3.2 Modelado de objetos en función del color

El problema del modelado de objetos puede ser demasiado simple si tomamos como suposición que los objetos a seguir únicamente constan de un color.

O puede convertirse en un problema complejo si tenemos en cuenta que en un mismo objeto aparecen diferentes colores, texturas, intensidades...

En la Figura 10 se muestra el problema descrito anteriormente. En función de qué parte de la imagen estemos tomando como referencia encontraremos un histograma de colores u otro.

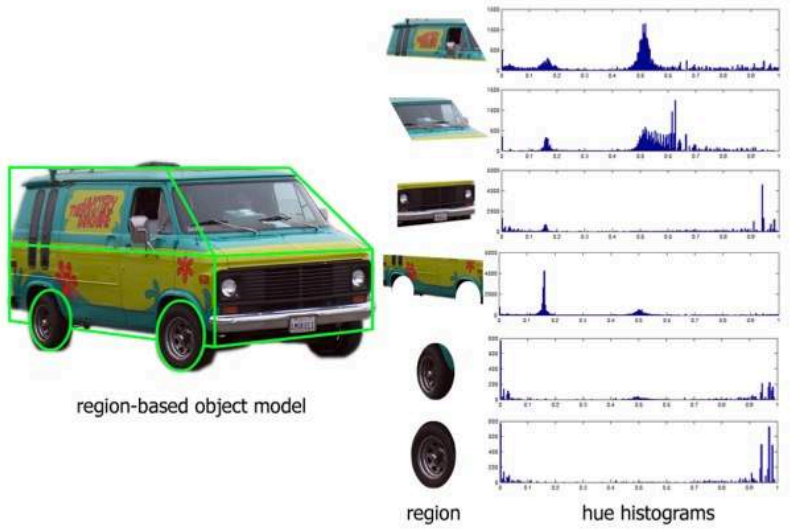


Figura 10: Histogramas de colores de las distintas regiones de un objeto.

La imagen superior (Figura 10) muestra el principal problema del seguimiento basado en colores. En función de la región de la imagen escogida encontraremos grandes diferencias entre los histogramas de colores.

Típicamente, una imagen digital utiliza el modelo de color RGB. No obstante, existen otros modelos de color. El modelo de color HSV puede dar un cierto grado de invariabilidad a cambios de iluminación en la escena.

Figura 11: Imagen en el espacio de color HSV.

1.2.3.3 Modelado de objetos en función de la forma

La forma de un objeto puede ser una poderosa herramienta a la hora de detectar y localizar un objeto en una imagen o un video. La forma de un objeto, en este sentido, se caracteriza por sus contornos y bordes dominantes en la imagen, que pueden ser descritos por la geometría básica, tales como líneas y curvas.

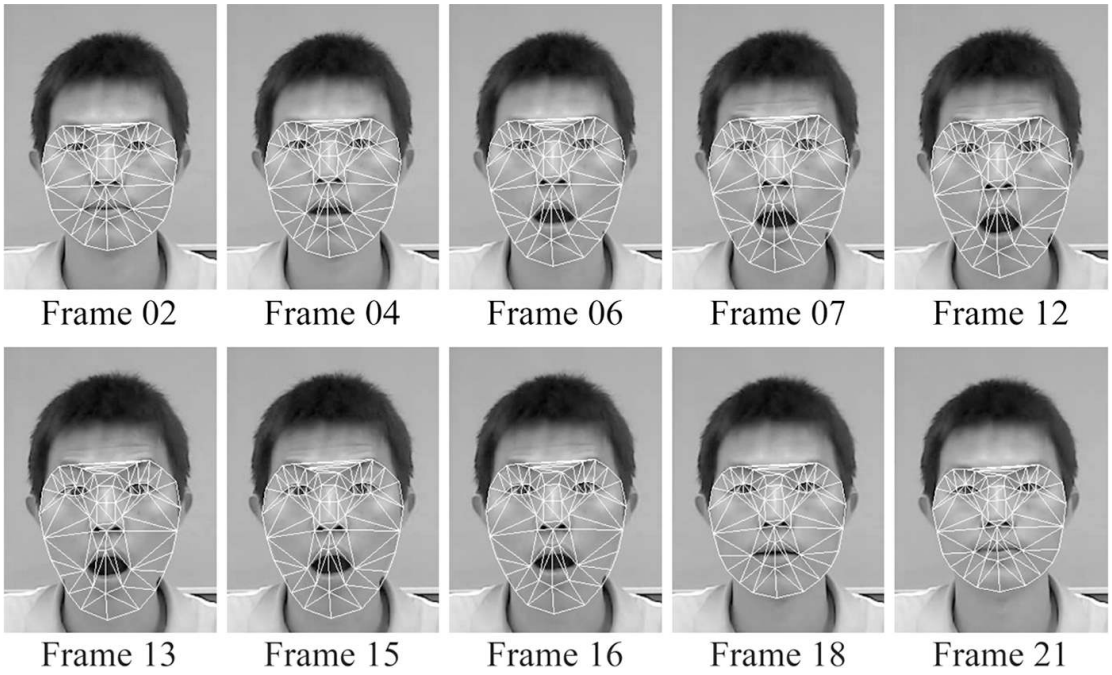


Figura 12: Ejemplo de modelo de apariencia activa utilizado para la detección del rostro de una persona.

En la Figura 12 se muestra un ejemplo de Modelos de Apariencia Activa. Los Modelos de Apariencia Activa permiten la parametrización de la textura y la forma combinada. Estos están acoplados a un algoritmo de búsqueda eficiente que le puede decir exactamente dónde y cómo un modelo se encuentra en un marco de imagen. Este modelo fue introducido por *Edwards, Cootes y Taylor*. Su eficiencia en ámbitos de reconocimientos gestuales o para encontrar las similitudes entre diferentes personas en el ámbito de la Medicina, ha sido ampliamente probada.

Por ejemplo, la forma de una cara es generalmente ovalada y deformable en la forma de un óvalo como se aprecia en la Figura 12. Los modelos de Apariencia Activa, utilizan técnicas de minimización de energía para bloquear los bordes de la imagen, que coincide con su conocimiento previo del modelo de los objetos.

1.2.3.4 Modelado de objetos en función de la textura

Al igual que el color, la textura de un objeto puede ser una característica de identificación. Los modelos de textura tratan de representar los patrones regulares de una imagen como, por ejemplo, el patrón en un par de pantalones. Por lo general, podemos clasificar los modelos de textura en dos grupos:

-modelos estadísticos: recogen estadísticas directamente en la imagen. Los modelos más simples de la textura incluyen histogramas 1D de niveles de gris, matrices de coocurrencia, y las diferencias de niveles de gris (Figura 13).

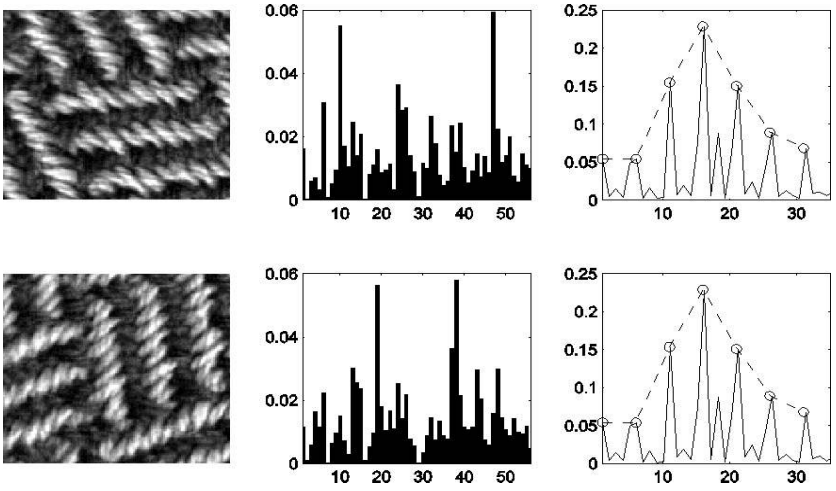


Figura 13: Texturas de un objeto basadas en un histograma de la imagen escala de grises.

-modelos espectrales: recogen estadísticas relacionadas con las características calculadas en el dominio de frecuencia de las respuestas de los filtros aplicados a la imagen. Una ventaja de los modelos espectrales es que los filtros son selectivos: pueden mejorar ciertas características, mientras que suprimen otras. Los filtros de Gabor bidimensionales han demostrado ser muy populares para el modelado de la textura, debido a su eficacia en la detección de la frecuencia y la orientación dominante en los patrones de textura.

1.2.3.5 Modelado de objetos en función de puntos característicos

Un objeto, en una imagen dada, puede ser definido por una serie de puntos característicos que lo constituyen.

Existen varios algoritmos de clasificación de objetos en función de sus puntos característicos. En este tipo de técnicas se clasifican los objetos en relación a su vecino más cercano. Las dos técnicas más utilizadas actualmente son:

-Scale Invariant Feature Transform (SIFT): Este algoritmo fue publicado por *David Lowe* en el año 1999, (1). El método de *Lowe* para la generación de la imagen característica, transforma una imagen en una gran colección de vectores característicos, cada uno de los cuales es invariante a la traslación de la imagen, escala y rotación de la misma. Además, el método es parcialmente invariante a los cambios de iluminación (Figura 14).

Figura 14: Extractor de puntos característicos de la imagen SIFT.

-Detección de esquinas: El detector de esquinas de *Harris* (2) es sin duda el algoritmo de mayor renombre en este apartado en la detección de puntos característicos (Figura 15).

Tomando como premisa que los puntos menos afines a cambiar ante traslaciones, rotaciones y escalado son las esquinas, Harris desarrolló un algoritmo orientado a la detección de esquinas.

El algoritmo detector de esquinas de Harris se basa en un principio central: en una esquina, la intensidad de la imagen debe cambiar en gran medida en múltiples direcciones.

El algoritmo de Harris utiliza matrices de autocorrelación con valores estrechamente relacionados con la intensidad de la imagen.

Figura 15: Ejemplo del detector de esquinas desarrollado por Harris.

1.2.3.6 Modelado de objetos con *Haar-like-Features*

En el año 2001, Viola y Jones (3) propusieron el primer marco de la detección de objetos en tiempo real. Este marco, al ser capaces de operar en tiempo real en el hardware de 2001, se dedicó en gran parte a la detección de rostro humano. En la actualidad, la capacidad para detectar rostros viene integrada con casi todas las cámaras digitales y teléfonos celulares en el mercado.

El algoritmo propuesto por Viola y Jones se centra en la creación de un clasificador basado en la combinación de varios clasificadores que intentar detectar características muy simples en una posible cara.

De este modo el clasificador desarrollado por Viola y Jones, recorre las posibles localizaciones de la imagen rechazando las regiones donde no es probable que exista un rostro cuando uno de los clasificadores más simples ha dado como resultado un negativo. Se puede definir el clasificador como una serie de etapas para la detección de personas (Figura 16).

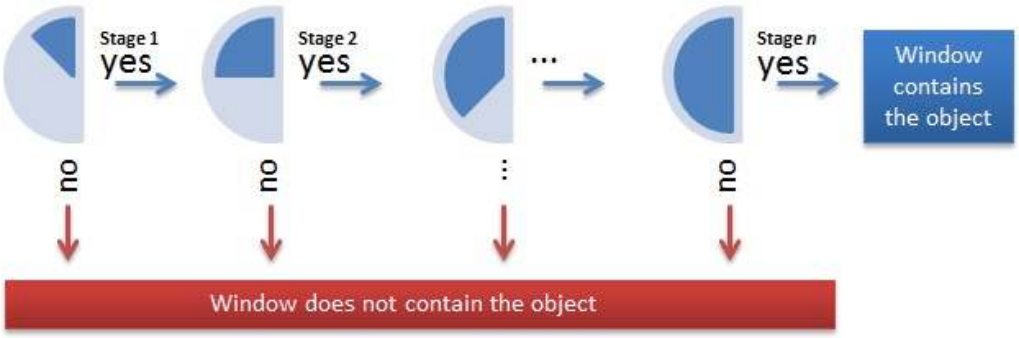


Figura 16: Etapas de un clasificador de Haar.

La idea de este algoritmo es la de recorrer todos las posibles localizaciones de una característica dentro de una imagen hasta que es capaz de encontrar un positivo en la detección de una característica. Una vez encontrada una característica, el algoritmo busca otro tipo de característica del objeto que se pretende localizar. Una vez existen varios positivos dentro de una ventana de búsqueda se puede determinar que el objeto existe dentro de una región de la imagen.

2. DESCRIPCIÓN GENERAL DEL SISTEMA

En este se tratará de dar una descripción general de los diferentes módulos que forman parte del sistema.

El objetivo del proyecto, es el de crear un sistema de seguridad capaz de detectar personas y seguirlas por el entorno de seguridad, así como interactuar con el usuario del sistema.

Con el fin de grabar imágenes del entorno donde ha sido emplazado el sistema desarrollado, se elabora una cámara IP utilizando una cámara, dos servo-motores y un ordenador de bajo coste (Figura 17).

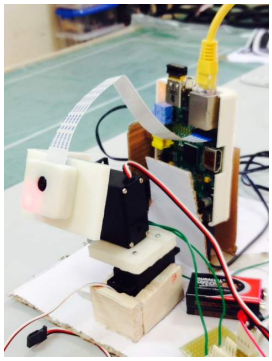


Figura 17: Vista de la cámara de red desarrollada con capacidad de rotación en dos ejes.

El sistema es capaz de mostrar imágenes en tiempo real (dependiendo de la velocidad de conexión a Internet) del entorno donde ha sido emplazado el sistema de seguridad, a través de una interfaz Web.

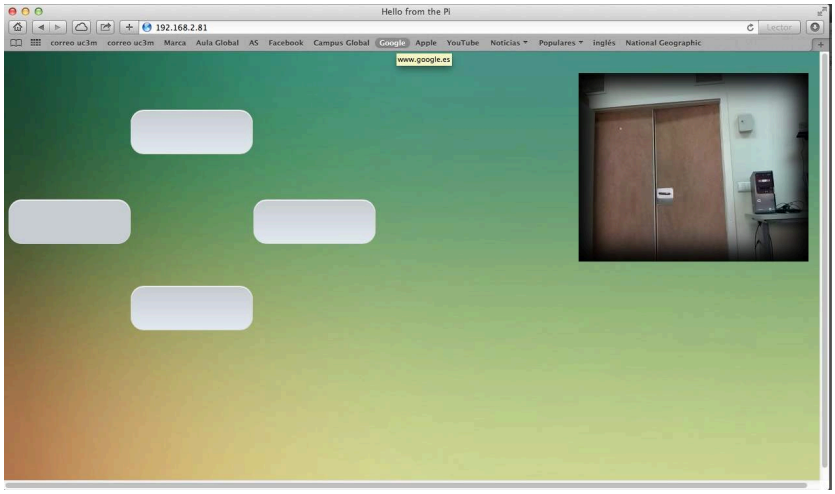


Figura 18: Vista de la interfaz Web del sistema.

Esta interfaz Web es la encargada tanto de mostrar las imágenes grabadas, como de permitir al usuario interactuar con la rotación de la cámara del sistema a través de una serie de botones situados en la interfaz Web (Figura 18).

Las imágenes grabadas por el sistema son enviadas simultáneamente a una interfaz Web y un ordenador conectado a la red local. Este ordenador se encarga de procesar las imágenes y determinar si existe o no una persona en el entorno de seguridad (Figura 19). En caso de una detección positiva, el ordenador almacenará la imagen donde se ha detectado una persona con el fin de dar la mayor información posible sobre la seguridad de su sistema al usuario.

Figura 19: Detección de una persona en la imagen procesada.

A partir de este momento el ordenador se comunica con la cámara Web para seguir a la persona que ha entrado en el entorno a través del movimiento de dos servo-motores. El seguimiento de personas, en la secuencia de imágenes obtenidas mediante la cámara de red (Figura 20), se lleva a cabo a través de técnicas de procesamiento de imagen.

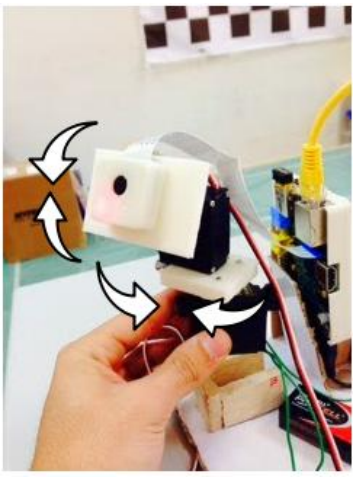


Figura 20: Direcciones de rotación de la cámara del sistema.

El sistema es capaz de seguir a una única persona al mismo tiempo, sin embargo, si entran varias personas en el entorno de seguridad se almacena la imagen donde se ha detectado al menos una persona.

3. HARDWARE DEL SISTEMA

Este capítulo recoge los principales elementos físicos utilizados para el desarrollo del sistema. Cabe destacar que el Capítulo 3.1 recoge todos los elementos físicos (*Hardware*) que han sido utilizados con el fin de desarrollar una cámara de red de bajo coste.

3.1 Cámara IP

Para la implementación de este proyecto se ha decido desarrollar íntegramente una cámara IP con capacidad de rotación en dos 2 ejes. A continuación se describen los principales elementos de la cámara IP desarrollada.

3.1.1 Raspberry PI

La cámara IP, anteriormente mencionada, utiliza como unidad central una *Raspberry Pi.* Este elemento es un ordenador de placa reducida o placa única (*SBC*) de bajo coste, desarrollado en Reino Unido por la Fundación *Raspberry Pi* (Figura 21).

Las principales características de este ordenador de pequeñas dimensiones son:

- Cuenta con un *System-On-a-Chip (SoC)* Broadcom, el cual incluye un microprocesador ARM1176JZF-S 700 MHz, con posibilidad de elevar esta frecuencia hasta 1 GHz.

- Una memoria *RAM* de 512 MiB.

- El sistema utiliza una tarjeta SD como disco duro del sistema.

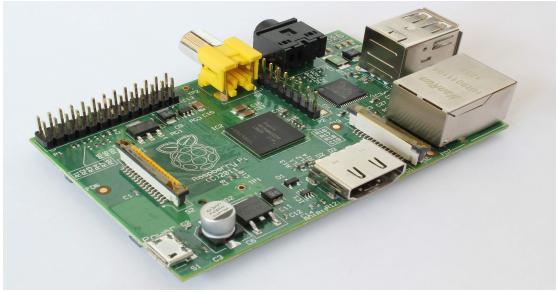


Figura 21: Placa de bajo coste Raspberry PI.

3.1.2 Raspberry PI Camera

La *Raspberry Pi Camera* es una cámara especialmente desarrollada por la Fundación *Raspberry* PI, para trabajar con la placa anteriormente descrita. A pesar de sus pequeñas dimensiones, la *Raspberry Pi Camera* cuenta con unas características muy interesantes:

- 5MP Omnivision 5647 Camera Module (Figura 22)
- Resolución (máxima): 2592 x 1944.
- Formatos de vídeo soportados por la cámara: 1080p 30fps, 720p.
- *Frame-Rate* máximo: 60fps .

Figura 22: Raspberry PI camera module.

3.1.3 Servo-Motores

Con el fin de permitir la rotación de la cámara descrita anteriormente, se han decidido utilizar dos servo-motores *Futaba s3003*.

Un servo-motor es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable en dicha posición.

Un servomotor es un motor eléctrico que puede ser controlado tanto en velocidad como en posición. Existen varios tipos (analógicos, digitales, basados en motores sin escobillas, etc) y en función de las necesidades del sistema se deberá emplear uno u otro. Además, cuando se necesite un coste de adquisición inferior, se escogerá principalmente el servo-motor analógico.

3.1.3.1 Principio de funcionamiento del servo-motor analógico.

Un servo-motor analógico cuenta con un circuito de control y un potenciómetro conectado al eje central del servo-motor, así como tres señales de entrada:

- *Vcc*: Alimentación.
- GND: Tierra (Ground).
- Señal de control.

El potenciómetro del servo-motor permite a la circuitería de control supervisar el ángulo del servo-motor. Si el eje está en el ángulo correcto, entonces el motor está apagado. Si el circuito chequea que el ángulo no es el correcto, el motor girará en la dirección adecuada hasta llegar al ángulo correcto. El rango de giro de un servo-motor generalmente está comprendido entre 0 y 180 grados. En la Figura 23 se muestra el despiezado de un servo-motor analógico:

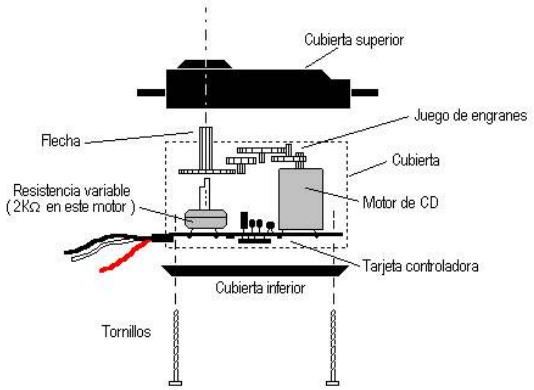


Figura 23: Despiezado de un servo-motor analógico.

Las principales características de los servo-motores generalmente están relacionadas con el par máximo que puede ofrecer el servo-motor y la velocidad de rotación. Según las especificaciones del fabricante, con una alimentación de 6 Voltios, es posible obtener un par de 4.1 kg/cm y una velocidad de rotación de 0.19 sec/60.

Figura 24: Servo-motor analógico Futaba s3003.

3.2 Ordenador de control

Las primeras ideas del proyecto estaban encaminadas a la ejecución de todos los procesos del sistema en la *Raspberry PI*. Sin embargo, a la hora de testear la velocidad de procesamiento de imagen, se constató que el *frame-rate* del proceso era demasiado bajo, inconveniente que se comentará en detalle más adelante.

Por este motivo se decidió externalizar el procesamiento de imagen a un ordenador de control más potente. El ordenador utilizado ha sido un *MacBook Pro 13".* Nótese que para la instalación del sistema, no es necesario utilizar este ordenador en concreto. Pero sí es necesario instalar en el ordenador de control un sistema operativo basado en *Unix.*

4 SOFTWARE DEL SISTEMA

En este capítulo se describen los principales recursos *Software* utilizados para el desarrollo del sistema. Entre ellos se encuentran los sistemas operativos utilizados, los lenguajes de programación y las librerías utilizadas para el desarrollo del proyecto.

4.1 Sistemas Operativos

Como ya se ha mencionado anteriormente para el desarrollo de una cámara IP se decidió emplear una *Raspberry Pi* (un ordenador de pequeñas dimensiones). Los fabricantes recomiendan el uso de una versión de *Debian (Linux)* adaptada. No obstante, existen otras versiones de *Linux* que se pueden incorporar al sistema, así como otros Sistemas Operativos no relacionados con *Linux.* De todos modos, el fabricante solo ofrece garantías con el uso de *Debian.*

4.1.1 *Debian*

Debian es un sistema operativo basado en *Linux*. El principal motivo para la elección de este operativo es que proporciona una gran facilidad y fiabilidad a la hora de programar en C y C++.

Otro motivo importante por el que se decidió utilizar este sistema operativo, está relacionado con el hecho de estar basado en UNIX, y la facilidad de comunicación y transferencia de datos que existe entre los sistemas operativos basados en UNIX.

4.2 Lenguajes de programación

4.2.1 Lenguaje de programación C

C es un lenguaje imperativo, es decir, de procedimiento. Fue implementado por primera vez por Dennis M. Ritchie en los laboratorios Bell como resultado de un proceso de desarrollo comenzado con un lenguaje anterior denominado B, inventado por Kenneth L. Thompson. Las principales características del lenguaje son:

- Proporciona acceso de bajo nivel a la memoria.

- Requiere tiempos de ejecución especialmente bajos.

- Gran flexibilidad a la hora de trabajar con sistemas operativos basados en UNIX.

- Disponibilidad de librerías para la comunicación entre procesos.

- Permite la portabilidad entre una gran variedad de plataformas informáticas.

4.2.2 Lenguaje de programación PHP

PHP que se conoce oficialmente como "*Hypertext Preprocessor*" fue lanzado en el año 1995. PHP fue originalmente diseñado para reemplazar un conjunto de scripts de Perl para mantener su *Personal Home Pages*, también conocidos como PHP.

PHP fue originalmente diseñado para crear páginas web dinámicas e interactivas. Es un lenguaje de script del lado del servidor generalmente utilizado en un contexto de HTML, *HyperText Markup Language*. El lenguaje PHP en un script puede consultar bases de datos, crear imágenes, leer y escribir archivos y comunicarse con los servidores remotos. La salida de código PHP se combina con HTML en secuencia de comandos y el resultado se envía al cliente.

Es posible utilizar PHP en casi todos los sistemas operativos. PHP puede ser usado en todos los principales sistemas operativos, incluyendo Linux, Microsoft Windows, Mac OS X, y RISC OS.

4.2.3 Lenguaje de programación HTML

HTML, *HyperText Markup Language,* es un estándar que sirve de referencia para la elaboración de páginas web. En sus diferentes versiones, define una estructura básica y un código, denominado código HTML, para la definición de contenido de una página web, como texto, imágenes, etc.

El lenguaje HTML basa su filosofía de desarrollo en la referenciación. Para añadir un elemento externo a la página (imagen, vídeo, *script*, etc.), este no se inserta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene sólo texto mientras que recae en el navegador web, que es el

interpretador del código, la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma formc por cualquier navegador web actualizado.

4.2.4 Lenguaje de programación JavaScript

JavaScript es un lenguaje de programación dinámico. Se utiliza con mayor frecuencia como parte de los navegadores web, cuyas implementaciones permiten scripts del lado del cliente para interactuar con el usuario, controlar el navegador, se comunican de forma asíncrona, y alteran el contenido del documento que se muestra.

El lenguaje JavaScript se integra dentro del código HTML de las páginas web y actúa en cuanto un evento (un *click*, por ejemplo) es ejecutado.

El lenguaje que fue inventado por *Brendan Eich* en la empresa *Netscape Communications*, apareció por primera vez en el *Netscape Navigator 2.0*. Tradicionalmente se viene utilizando en el marco de aplicaciones cliente/servidor.

4.2.5 Lenguaje de programación CSS

Las hojas CSS son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML. El *World Wide Web Consortium* (*W3C*) es el encargado de formular la especificación de las hojas de estilo que servirá de estándar para los navegadores.

La idea del desarrollo de CSS es separar la estructura de un documento de su presentación. La información de presentación se proporciona separada en una hoja de estilo con la que se especifica como se ha de mostrar: color, fuente, alineación del texto y tamaño, además puede ser adjuntada tanto como un documento separado o en el mismo documento HTML.

4.3 Librerías de programación utilizadas

4.3.1 Librerías de OpenCV 2.4

OpenCV (*Open Source Computer Vision*) es una librería de funciones de especialmente destinadas al procesamiento de imagen por computador. Fue desarrollada por *Intel Russia Research.* Para obtener más información se aporta el siguiente enlace al lector [E3].

4.3.1.1 Licencia BSD

Las librerías de OpenCV están sujetas a la licencia BSD. La licencia BSD es la licencia de software otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*). Es una licencia de software libre. Esta licencia tiene muy pocas restricciones y permite el uso del código fuente en software no libre. El lector puede encontrar la licencia completa en el siguiente enlace [E4].

4.3.2 ServoBlaster

La librería *ServoBlaster* ha sido desarrollada específicamente para dar acceso a los programadores de la Raspiberry Pi a los periféricos del microprocesador ARM1176JZF-S. Está desarrollada en lenguaje C.

4.3.2.1 Licencia de *ServoBlaster*

El software está distribuido bajo la licencia *GNU General Public Licence.* Esta licencia permite a los usuarios finales leer, modificar y compartir el *Software.* Para más información sobre la licencia se aporta el siguiente enlace [E6].

4.4 Servidor Web Apache

El objetivo del Proyecto *Apache HTTP Server* es desarrollar y mantener la fuente abierta del servidor HTTP (*HiperText Trasfer Protocol*) para los sistemas operativos que contienen el UNIX y Windows. El Proyecto Apache HTTP Server

proporciona un entorno de servidor seguro, eficiente y extensible, que proporciona servicios HTTP en sincronización con los estándares HTTP actuales.

Después de haber sido revisado muchas veces, el servidor Apache se ha convertido en el software de servidor web más popular del mundo y puede ser ampliamente utilizado en casi todas las plataformas informáticas. Apache es un software de código abierto, por lo que hay muchas personas que desarrollan nuevas capacidades del servidor Apache, las características y la modificación del defecto original.

4.4.1 Apache Licence

La Licencia Apache permite al usuario del software la libertad de usarlo para cualquier propósito, distribuirlo, modificarlo y distribuir versiones modificadas de ese software.

La Licencia Apache sólo exige que se mantenga una noticia que informe a los receptores que en la distribución se ha usado código con la Licencia Apache. Para más información sobre la licencia de *Apache* se aporta el siguiente enlace [E5].

4.5 AJAX

AJAX, acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML *AJAX* no es un lenguaje de programación, ni tampoco es una nueva tecnología. La utilización de *AJAX* ha sido posible desde que se introdujo el elemento *XMLHttpRequest*. *AJAX* significa "*Asynchronous JavaScript and XML*". La principal ventaja que aporta la utilización de *AJAX* se basa en que no es necesario hacer continuos refrescos de la página web, sino que permite una comunicación asíncrona entre *JavaScript* y el servidor.

4.6 POSIX

POSIX es un acrónimo de *"Portable Operating System Interface",* que es una familia de estándares especificados por la IEEE para el mantenimiento de la compatibilidad entre los sistemas operativos.

POSIX define la interfaz de programación de aplicaciones (API), junto con intérpretes de línea de comandos y las interfaces de servicios públicos, para la compatibilidad del software con las variantes de *Unix* y otros sistemas operativos. Para el sistema diseñado se han utilizado *POSIX threads y POSIX sockets.*

4.6.1 TCP/IP POSIX Sockets

Los *POSIX sockets* fueron desarrollados en la universidad de *Berkeley*. La primera versión de esta librería fue sacada al mercado en el año 1983 para el sistema operativo *4.2BSD Unix Operative System.* El protocolo *TCP/IP* se basó en esta librería de comunicación para su implementación.

La comunicación mediante *sockets* generalmente se basa en una arquitectura cliente/servidor. Para las comunicaciones de TCP, existe un host a la escucha de solicitudes de conexión entrantes. Cuando llega una petición, el host del servidor lo aceptará, por lo que los datos de puntos pueden ser transferidos entre los hosts.

La API de *sockets* utiliza dos mecanismos para entregar los datos a nivel de aplicación: puertos y *sockets*.

Todas las pilas TCP/IP tienen 65.536 puertos para TCP y UDP. Hay una completa variedad de puertos para UDP (numeradas 0-65535) y otro con el mismo esquema de numeración para TCP. Los dos conjuntos no se superponen.

Un puerto no es una interfaz física, sino que es un concepto que simplifica el concepto de las comunicaciones por Internet. Al recibir un paquete, la pila de protocolos dirige dicho paquete al puerto específico. Si no hay ninguna aplicación escuchando en ese puerto, el paquete se descarta y un error puede ser devuelto al remitente. Sin embargo, las aplicaciones pueden crear *sockets*, que les permiten adherirse a un puerto. Una vez una aplicación esté adherida a un puerto todas los paquetes entrantes con destino a ese puerto serán pasados a la aplicación.

4.6.2 *POSIX Sockets* en el entorno de *Unix*

A la hora de comunicar varios procesos que se están ejecutando en una misma máquina se pueden utilizar *sockets* del entorno de ejecución

Antes de definir este tipo de *sockets,* es necesario definir que es un descriptor de fichero. Un descriptor de fichero es simplemente un dato entero

que contiene una dirección a la memoria del ordenador. Nótese que esta dirección no tiene por qué ser la dirección de la memoria física, sino una "*clave*" de acceso a una dirección de la memoria.

Los *POSIX sockets* del entorno de *Unix* están basados en descriptores de ficheros. Básicamente, cuando dos procesos se comunican mediante *sockets*, se están pasando descriptores de fichero entre sí con el fin de crear un canal de comunicación entre los dos procesos.

4.6.3 POSIX threads

El enfoque principal de los *threads* es el de poder conseguir implementar procesos multitarea. Esta técnica se ha empleado con éxito por lo menos durante el último cuarto de siglo para construir sistemas informáticos altamente sensibles, robustos y que hacen de todo: desde los transbordadores espaciales que vuelan hasta la decodificación de los programas de televisión por satélite.

Mediante la utilización de *threads* (en español "hilos") podemos conseguir tener varios procesos ejecutando al mismo tiempo sin necesidad de vernos obligados a utilizar técnicas de comunicación como podrían ser los *pipes* o la memoria compartida entre procesos. Básicamente el uso de *threads* facilita a los programadores la compartición de recursos y variables entre procesos que se ejecutan de forma paralela.

5 ARQUITECTURA FUNCIONAL

5.1 Rotación del sistema de seguridad

Como ya se ha mencionado en el Capítulo 3.1.3, el sistema desarrollado cuenta con dos servo-motores.

La utilización de estos servo-motores está directamente relacionada con el seguimiento de intrusos en el entorno de seguridad, y la capacidad de dar al usuario control sobre su sistema (poder rotar la cámara desde una interfaz web).

5.1.1 Señal de control de un servo-motor

Los servo-motores son generalmente controlados por pulsos eléctricos *PWM,* o *Pulse-Width Modulation.* Una señal *PWM* es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones, o para controlar la cantidad de energía que se envía a una carga (Figura 25).

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva, en relación con el período. Expresado matemáticamente:

$$D = \frac{Ton}{T} \qquad [1.1]$$

- D: ciclo de trabajo.
- Ton: tiempo en que la función es positiva (ancho del pulso).
- T: periodo de la función.

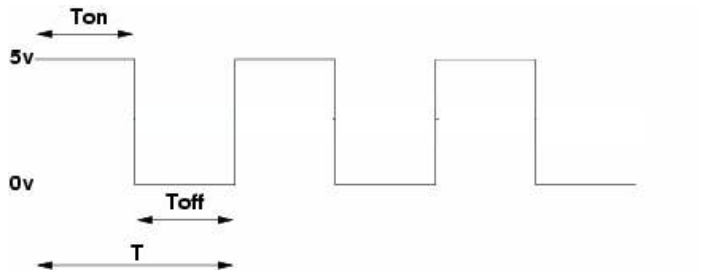


Figura 25: Señal PWM (Pulse Width Modulation) donde se muestran los tiempos Ton (tiempo en alto) y Toff (tiempo en estado bajo).

Otro parámetro importante de las señales *PWM* es el voltaje medio que tenemos en la señal (Figura 26). Dicho voltaje medio se define como:

$$Vmedio = \frac{Ton * Vmax - Toff * Vmin}{T} \qquad [1.2]$$

- *Ton:* tiempo en que la función presenta su valor máximo.
- *Toff*: tiempo en que la función presenta su valor mínimo.
- *T*: periodo de la función.
- *Vmax*: voltaje máximo de la función.
- Vmin: voltaje mínimo de la función.

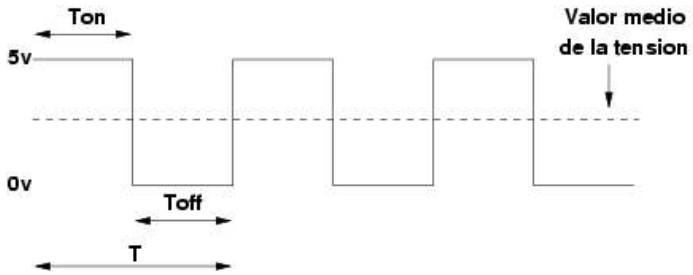


Figura 26: Señal PWM (Pulse Width Modulation) donde se muestra el voltage medio de la señal.

5.1.2 Control de la posición de un servo-motor analógico

El control de servo-motores analógicos se realiza a través de señales *PWM* inyectadas a la señal de control.

En función del ciclo de trabajo que utilicemos, el servo-motor girará en un rango de 0 a 180 grados. Generalmente, el periodo de la señales de control viene especificado en *milisegundos*. La Figura 27 espera clarificar este concepto:

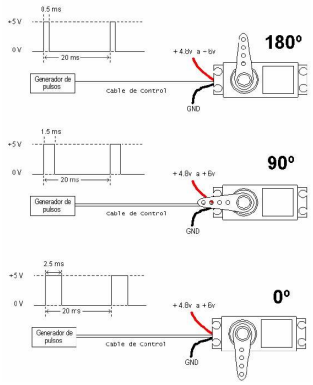


Figura 27: Rotación de un servo-motor analógico en función del ciclo de trabajo de la señal de control.

5.1.3 Giro de un servo-motor en función de la señal PWM

Como se puede apreciar en la Figura 27, modificando la duración del pulso alto (Ton) conseguimos girar un ángulo θ.

El ciclo de trabajo [1.1] que debe tener la señal de control, y el periodo de la misma, varían en función del fabricante, no obstante, generalmente son:

- T = 20 ms
- D [5 , 10] %
- Ton [1.5 , 2] ms

Es sencillo notar, que para el caso del motor anteriormente mencionado, la duración del pulso alto para conseguir un ángulo de posición θ, estará dada por la fórmula:

$$Ton = 1 + \frac{\theta}{180} \ [ms] \qquad [1.3]$$

5.1.4 Circuitería de control de la posición de un servo-motor analógico

El uso de señales *PWM* para el control de servo-motores se debe al hecho de que permiten controlar de manera precisa el voltaje medio [1.2] generado. En la figura inferior (Figura 28) se adjunta un diagrama con el esquema de señal de la circuitería de control de un servo-motor:

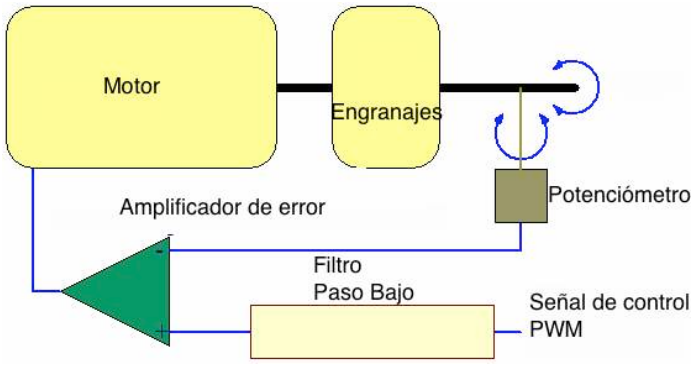


Figura 28: Esquema de la circuitería de control de un servo-motor analógico común.

Observando la Figura 28, la señal de control entra a un filtro paso bajo que transformará la señal *PWM* de entrada en una señal de voltaje. El voltaje resultante de esta operación, es el voltaje medio de la señal *PWM.*

El potenciómetro mostrado en la figura está conectado al eje que será girado por el motor. De este modo, la posición del eje de giro está directamente relacionada con la tensión del potenciómetro.

El amplificador de error es un amplificador operacional con realimentación negativa.

La salida del amplificador de error es un voltaje negativo o positivo que representa la diferencia entre sus entradas. Cuanto mayor sea la diferencia, mayor es la tensión. La salida del amplificador de error se utiliza para accionar el motor.

5.1.5 Hardware de control de los servo-motores

Para la generación de la señal de control de los servo-motores se utiliza la *Raspberry PI.*

La *Raspberry PI* cuenta con un microcontrolador *ARM1176JZF-S*. Siguiendo la línea de tendencia de la arquitectura *ARM,* el microcontrolador de la *Raspberry PI* cuenta con diversos periféricos como temporizadores, *ADCs, DACs, etc.* En la Figura 29 se muestra una esquemático del microcontrolador. Nótese como se han resaltado los temporizadores mencionados.

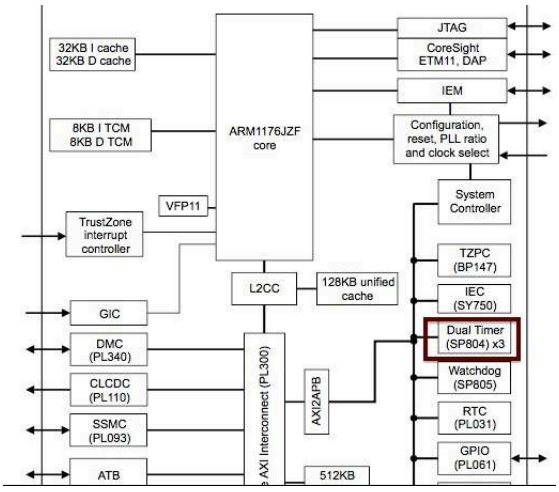


Figura 29: Esquema del microcontrolador ARM1176JZF-S donde se muestran los temporizadores utilizados para la generación de señales PWM resaltados en granate.

5.1.6 Software de control de los servo-motores

El principal problema de la generación de pulsos *PWM,* se basa en la precisión que deben tener este tipo de pulsos. Si el pulso no es estable, el servo-

motor no se quedará fijo en una posición. De este modo, si el ciclo de trabajo oscila:

$$Ton = Ton \pm \Delta t \qquad [1.4]$$

la posición del servo-motor también oscilará:

$$\theta = \frac{Ton \pm \Delta t - 1}{180} \qquad [1.5]$$

Durante el desarrollo del proyecto se observó que las señales *PWM* generadas tenían muy poca precisión en los ciclos de trabajo, lo que se traducía en continuas vibraciones en los servo-motores del sistema. Se llegó a la conclusión de que el paso por la *CPU* retardaba demasiado las actualizaciones de los registros de los temporizadores del microcontrolador.

Para solucionar este problema se decidió generar las señales de control de los servo-motores, a través de accesos al *DMA (Direct Memory Access)* del microcontrolador.

La función del *DMA* es la de controlar el sistema de memoria sin utilizar la *CPU*. Periféricos como el *ADC, DAC* y los *temporizadores* requieren movimientos frecuentes y regulares de la memoria de sus respectivos registros, y la utilización de la *CPU* para acceder a sus respectivos registros de memoria, conlleva retardos en la actualización de los mismos.

Para acceder al *DMA* [E.1]del microcontrolador se ha utilizado la librería *ServoBlaster* (Capítulo 4.3.2) (bajo la licencia *GNU General Public Licence*). Mediante el uso de esta librería se puede acceder al registro de los temporizadores del microcontrolador, con sentencias de código simples donde podemos especificar la frecuencia de actualización de los registros de los dos temporizadores utilizados (50 *Herzios*) y el tiempo que queremos de pulso alto (1000 – 2000 [ms]).

5.1.7 Diseño de la estructura de rotación

Para permitir que la cámara del sistema de seguridad rote, se diseñaron dos piezas en formato CAD, con el fin de poder imprimir estas piezas en una impresora 3D (Figura 30).

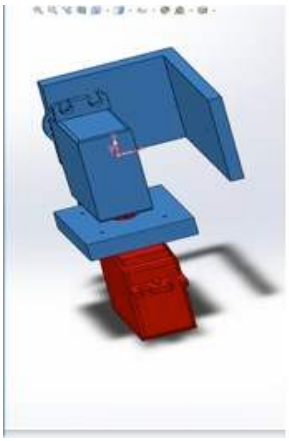


Figura 30: Diseño CAD de la estrcutura de rotación de la cámara de vigilancia.

Añadiendo dos servo-motores a la estructura se consigue la rotación de la cámara del sistema en el eje horizontal y vertical (Figura 31).

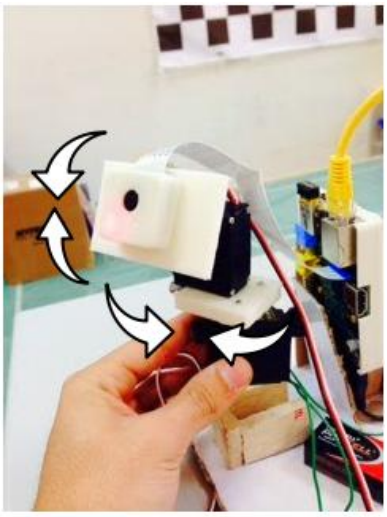


Figura 31: Rotación de la cámara de seguridad del sistema.

5.1.5 Elección de los servo-motores

Los servo-motores analógicos comerciales traen dos parámetros representativos de sus características. El par máximo que puede generar el servo-motor y la velocidad máxima de rotación.

Elegir una medida apropiada para la velocidad máxima de una persona puede ser una tarea compleja. Para este caso de estudio se ha decidido utilizar una velocidad máxima de 4 m/s. Como se demuestra en la Figura 32, la rotación deberá ser mayor en función de la lejanía de la persona a nuestro centro de rotación. Se tomará como situación más desfavorable una distancia al objeto de seguimiento de 2 metros.

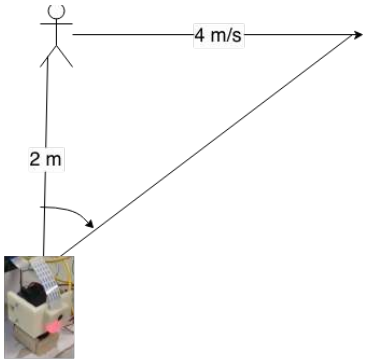


Figura 32: Esquema de rotación de la cámara ante la situación más desfavorable en la que una persona se mueve a 4 m/s a una distancia de 2 m de la cámara.

Dada la estimación de una velocidad máxima de una persona de 4 m/s, y considerando el caso más desfavorable de una distancia al objeto de seguimiento de 2 m, la velocidad de giro máxima debe ser igual a 180°/s.

Lo primero que se debe hacer para evaluar los parámetros de un servo-motor es realizar un perfil de las velocidades requeridas por el sistema:

Tiempo	Intervalo	Velocidad de rotación	Posición
Segundos	Segundos	rpm	Grados
0.250	0.250	30	45
0.750	0.500	30	135
1.000	0.250	0	180

Tabla 1: Perfil de velocidades del sistema

De la Tabla 1 se puede concluir que el servo-motor escogido debe ser capaz de trabajar al menos a 30 *rpm*. Con una aceleración máxima de 120 $\frac{rpm}{s^2}$.

Una vez calculado el perfil de velocidades se puede calcular el Par máximo necesario utilizando:

$$\tau = I \cdot \frac{d\omega}{dt} \qquad [1.6]$$

- τ: Par.
- *I:* Inercia del sólido.
- $\frac{d\omega}{dt}$: Aceleración angular.

Nótese que cada uno de los servo-motores únicamente rota en un eje, por lo tanto, no es necesario construir una matriz de inercias.

Para calcular la Inercia del sistema se ha utilizado el software *SolidWorks:*

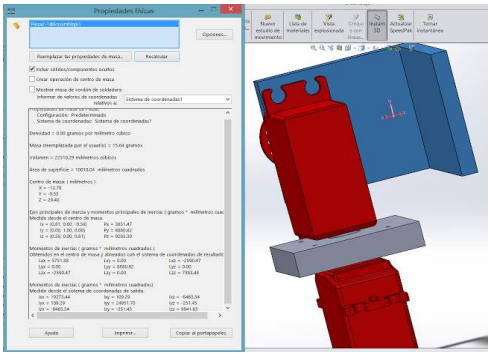


Figura 33: Inercia de la estructura de rotación en el eje vertical.

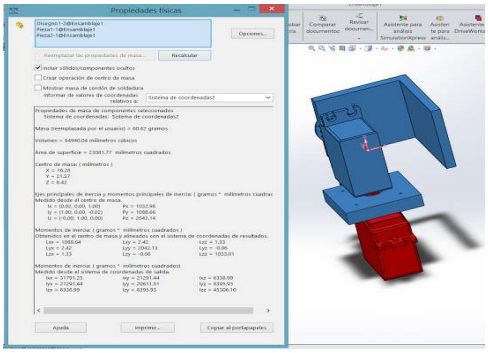


Figura 34: Inercia de la esructura de rotación en el eje horizontal.

Con el fin de tener una medida precisa del peso de cada una de las partes del sistema, se ha utilizado una balanza de precisión (Figura 35):

Figura 35: Pesos precisos de cada una de las piezas utilizadas en la estructura de rotación del sistema.

Una vez obtenidas las inercias de cada una de las piezas que van a ser rotadas, podemos obtener el par necesario para cada una de las dos rotaciones en base a la situación más desfavorable (intervalo de tiempo [0,0.250] segundos).

Utilizando [1.6] para la rotación horizontal:

- $\frac{d\omega}{dt} = 12.5664 \ \frac{rad}{s^2}$
- $I = 2.2661 \cdot 10^{-5} \ kg \cdot m^2$
- $\tau = 2.8476 \ \cdot 10^{-4} \ Nm$

Utilizando [1.6]para la rotación vertical:

- $\frac{d\omega}{dt} = 12.5664 \frac{rad}{s^2}$
- $I = 1.9273 \cdot 10^{-5} kg \cdot m^2$
- $\tau = 2.4407 \cdot 10^{-4} Nm$

El servo-motor analógico *Futaba-S3003* tiene las siguientes especificaciones, que podemos observar en la Tabla 2:

Alimentación (Voltios)	Par-Máximo (Nm)	Velocidad-Máxima(rpm)
4.8	$32 \cdot 10^{-3}$	43.4783
6	$41 \cdot 10^{-3}$	52.6316

Tabla 2: Características Futaba-S3303

Se puede afirmar que el servo *Futaba-S3003* (Figura 36) cumple las especificaciones del sistema tanto en Par-Máximo como en velocidad de rotación con los dos perfiles de alimentación propuestos por el fabricante.

Figura 36: Servo-motor Futaba S3003.

5.2 Comunicación entre procesos

El sistema de vigilancia cuenta con varios procesos que se llevan a cabo de forma paralela. Para que el sistema completo pueda funcionar correctamente, es necesario comunicar y sincronizar los procesos entre sí. En el 3 se ha comentado que el sistema desarrollado cuenta con una interfaz Web y un ordenador donde se llevan a cabo tareas de procesamiento de imagen.

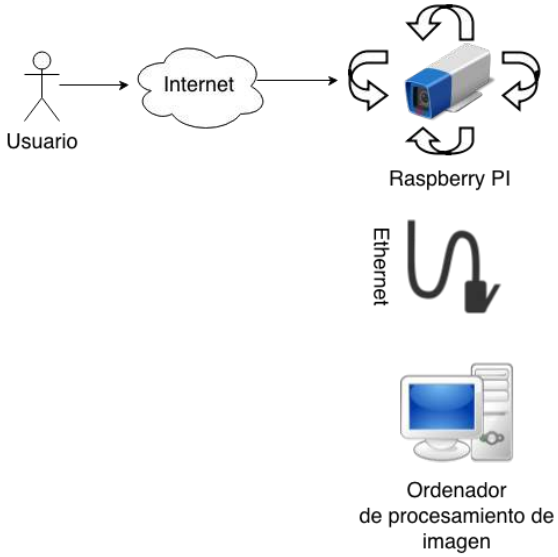


Figura 37: Elementos a comunicar entre sí.

Tal y como se aprecia en la Figura 37 el ordenador donde se llevan a cabo tareas de procesamiento de imagen está conectado a través de *Ethernet* a la *Raspberry PI,* que hace las funciones de una cámara de red. Además existe una interfaz Web para permitir al usuario interactuar con el sistema. Tanto el ordenador de procesamiento de imagen como el usuario, pueden controlar la rotación de la cámara de vigilancia, por lo tanto es necesario establecer un protocolo de comunicación y sincronización para los dos procesos.

Con el fin de permitir el control de los servo-motores tanto al ordenador de procesamiento de imagen como a la interfaz Web, se desarrolla un proceso paralelo para el control de los servo-motores. Este proceso es capaz de recibir mensajes y sincronizarlos.

La idea general se basa en tener dos servidores paralelos que puedan *escuchar* ante peticiones de dos clientes (ordenador e interfaz Web) en un mismos proceso. En función de los mensajes entrantes, este proceso controla la rotación de los servo-motores mencionados en el C5.1 .

El proceso mencionado está escrito en lenguaje C, y se ejecuta en la *Raspberry PI,* que como ya se ha mencionado, hace las veces de cámara IP para el sistema.

Los dos servidores a la espera de peticiones de procesos clientes hacen uso de *POSIX sockets* (Capítulo 4.6.2 y 4.6.1).

Con el fin de sincronizar los dos servidores y ejecutar en paralelo los dos servidores, se utilizan *POSIX threads,* Capítulo 4.6.4. Otra opción habría consistido en el uso de una región de memoria compartida entre los dos servidores, no obstante, esta técnica suele tener un manejo más tedioso a la hora de sincronizar las lecturas y escrituras de las zonas de memoria compartida.

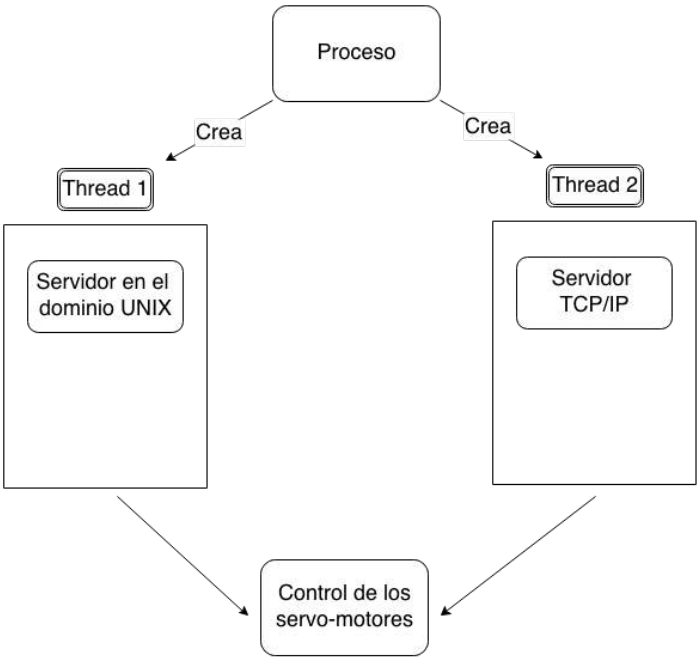


Figura 38: Esquema general de la comunicación entre procesos.

En la Figura 38 se muestra un esquema del proceso, nótese que la interfaz web utiliza *sockets* del dominio *Unix* (descritos en el Capítulo 4.6.2) al ejecutarse en el mismo terminal, mientras que el servidor que se comunica con el ordenador de control utiliza *sockets* del dominio *TCP/IP* (descritos en el Capítulo 4.6.1).

5.2.1 Implementación de la comunicación entre procesos

Para poder escuchar correctamente a las peticiones de los respectivos clientes del sistema se implementan dos servidores en lenguaje C.

Existen varios dominios de comunicación para los sistemas operativos basados en *Unix*, no obstante, para el sistema desarrollado solo se han tenido en cuenta dos de ellos:

- AF_INET (unidos mediante una red TCP/IP).

- AF_UNIX (en el mismo Sistema Operativo).

En la Figura 39 se muestra el esquema de creación de un *socket* servidor con las llamadas a los respectivos métodos.

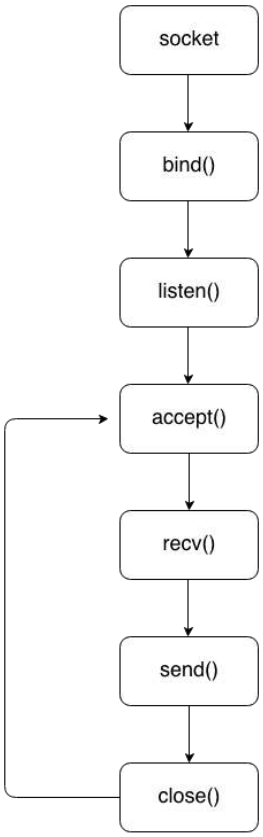


Figura 39: Diagrama de flujo de un proceso servidor.

5.2.1.1 Implementación de un servidor TCP/IP

Un servidor del dominio AF_INET, se comunica con un servidor mediante el envío de *paquetes* en forma de *streams*. Además existe la posibilidad de comunicarse a través de datagramas, no obstante el uso de *streams* provee un método de comunicación libre de errores, por lo tanto no es necesario llevar a cabo una reordenación del *stream* ni implementar métodos de detección de errores. Como ya se ha mencionado en el Capítulo 4.6.1, este tipo de *sockets* son la base del protocolo de comunicación *TCP/IP.*

Los pasos a seguir para la configuración de un servidor *TCP/IP* son:

1. Creación de la estructura *socket*.

2. Con el método *bind ()* se define el puerto en el que el servidor escuchará las peticiones de los clientes.

3. Definición con el método *listen()* del número de conexiones pendientes que podemos permitir. Esto quiere decir que si estamos escuchando a un cliente, y otro cliente hace una petición, este último se quedará a la espera de que el servidor despache la comunicación con el cliente actual.

4. Con el método *accept()* el servidor permanece en estado de espera hasta que un cliente hace una petición. Esta función permite acceder a una de las peticiones pendientes de la función *listen().*

5. Con el método *recv()* recibimos un *stream* de caracteres del cliente.

6. Con el método *send()* respondemos al cliente a través de un *stream* de caracteres.

7. Utilizando el método *close()* cerramos la conexión con el cliente.

5.2.1.2 Implementación de un servidor en el dominio *Unix*

La creación de un servidor del dominio *TCP/IP* es similar a la creación de un servidor del dominio *Unix*, la principal diferencia se basa en el método de comunicación utilizado. Los paquetes enviados no se envían a través del protocolo *TCP/IP,* sino utilizando descriptores de ficheros, en el Capítulo 4.6.2 se explica con más detalle este tipo de comunicación.

5.2.3 Implementación de un proceso multitarea

Como ya se ha avanzado previamente, el proceso de comunicación debe mantener dos procesos (servidores) ejecutándose de forma paralela, para el desarrollo del proceso multitarea se ha hecho uso de *threads.*

Los *threads* en un sistema operativo basado en *Unix* son creados por un proceso que vamos a llamar *"padre".* Los *threads* creados por el proceso "*padre"* realmente comparten todos los recursos utilizados por este proceso "*padre"*, lo que se traduce en que son capaces de acceder a la región de memoria creada por este proceso *"padre"* y a todos los datos almacenados en ella. No obstante, el sistema operativo es capaz de administrarlos para que se ejecuten de manera paralela como si fueran un proceso independiente. En la Figura 40 se muestra un esquema del proceso *"padre"*, y las regiones de memoria:

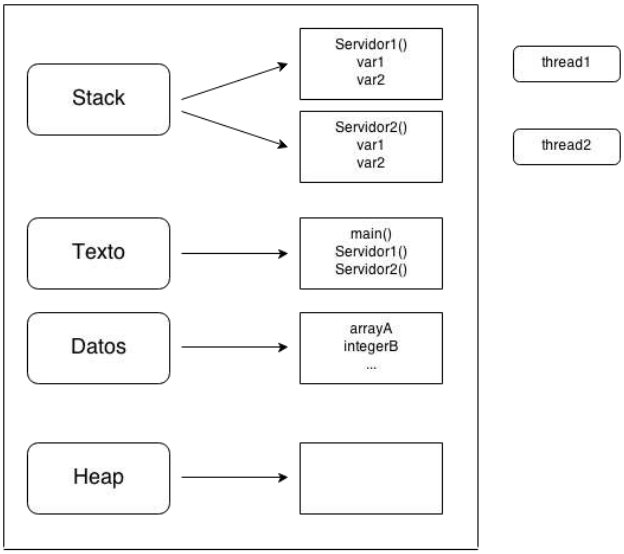


Figura 40: Esquema de memoria del proceso padre.

Como se puede observar en la Figura 40 los *threads* realmente están alojados en el *stack* del proceso "padre". De este modo se cumple lo mencionado anteriormente de que los *threads* creados pueden acceder a los recursos del proceso que los crea, manteniendo una ejecución paralela.

5.2.3.1 Control de la lectura y escritura de variables compartidas

Dado que existen dos procesos que pueden acceder a la región de memoria del proceso "*padre*", se debe controlar que nunca ocurra el caso en el que dos *threads* tratan de leer o escribir en la misma variable (región de memoria). La falta de control sobre el acceso de los *threads* a las mismas regiones de memoria puede dar lugares a errores inesperados en la ejecución del programa, o valores erróneos sobre la región de memoria modificada.

Con el fin de controlar que nunca se dé el caso en que dos *threads* tratan de acceder a la misma región de memoria, se han utilizado *mutexes* (Figura 41).

Los *mutexes* se pueden ver como semáforos. Cuando un *thread* está accediendo a una variable compartida, bloquea la ejecución del segundo *thread,* una vez ha terminado la tarea, desbloquea el semáforo y el segundo *thread* puede continuar su ejecución normal. Los *mutexes* están incluidos en el estándar *POSIX* bajo la librería <pthread.h>.

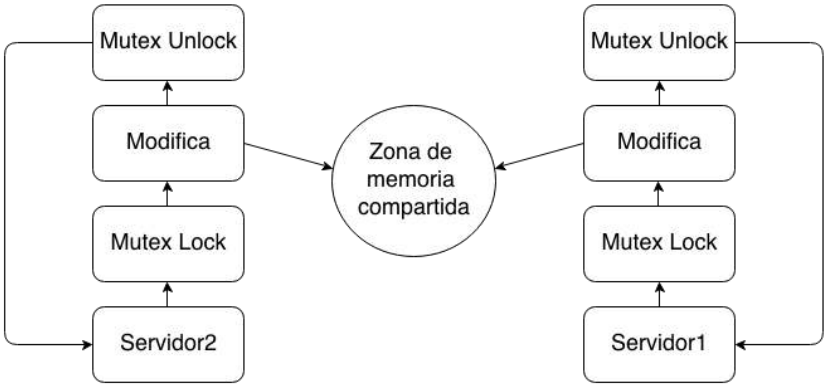


Figura 41: Acceso a las zonas de memoria compartida desde los threads implementados.

5.2.4 Manejo de señales del sistema operativo

Existen varios escenarios en los cuales el sistema operativo podría cerrar inesperadamente el proceso principal. En el caso de que esto ocurra, es necesario haber cerrado correctamente los servidores creados y liberar la memoria utilizada por cada uno de los *threads.*

Como ya se ha mencionado en el Capítulo 5.2.3, los *threads* creados por el proceso principal utilizan un espacio de memoria. Por lo tanto si no se cierran correctamente, podríamos vernos en el caso de que ese espacio de memoria siga siendo utilizado por los *threads*.

En el caso de no cerrar correctamente el servidor *TCP/IP* mencionado en el Capítulo 5.2.1.1, el sistema operativo no permitiría adjuntar de nuevo el servidor al mismo puerto, con lo que no se podría reiniciar la ejecución del programa hasta después de haber reiniciado el ordenador.

En general existen varios sucesos en los cuales el sistema operativo podría tomar el control de un proceso y cerrarlo inesperadamente (fallo en la memoria, señal de cierre del desde el terminal, fallo al guardar un objeto en la memoria, memoria insuficiente para el proceso, etc) (Figura 42). Existen muchas señales que el sistema operativo puede enviar al proceso en ejecución, no es el objetivo de este capítulo listar todas ellas. Con el objetivo de ganar robustez en el sistema ante posibles errores se capturan algunas de las señales más comunes: `SIGINT`, `SIGKILL`, `SIGALRM`, `SIGHUP`, `SIGSEGV`.

La librería *<signal.h>,* incluida en el estándar *POSIX,* provee métodos con los que procesar correctamente las señales enviadas desde el sistema operativo al proceso.

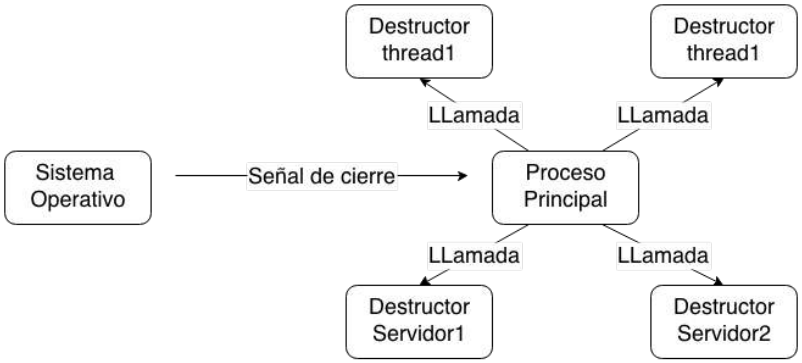


Figura 42: Respuesta a la petición de cierre del proceso desde el sistema operativo por parte del proceso padre.

5.2.5 Diagrama UML

Con el fin de clarificar el proceso seguido para la comunicación entre los procesos del sistema se ha elaborado un diagrama UML(Figura 43) que pretende clarificar las relaciones entre los módulos explicados en este capítulo:

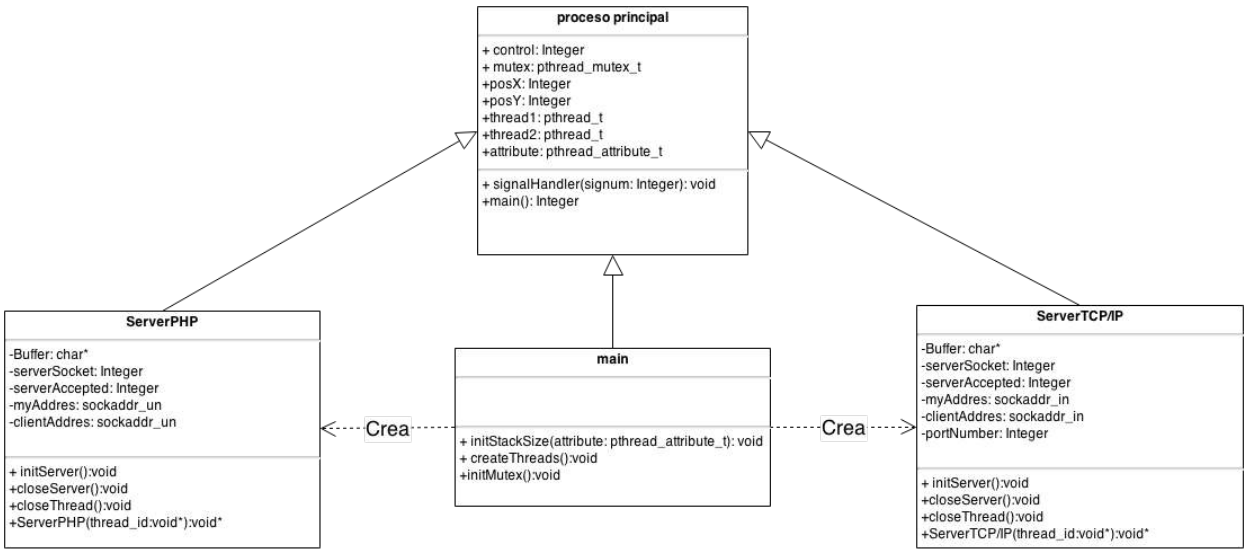


Figura 43: Diagrama UML del proceso.

5.2.6 Tiempos de comunicación

En el Gráfico 1 se muestran los tiempos de comunicación entre el ordenador de control y el servidor *socket IP*. Nótese que la figura inferior cubre los tiempos que tarda un mensaje completo (envío del mensaje desde el cliente, respuesta al mensaje desde el servidor) en ser transmitido. Los tiempos de comunicación entre la interfaz Web y el servidor *socket* del dominio *Unix* no se muestran, dado que dependerán altamente de la conexión a Internet disponible para el usuario.

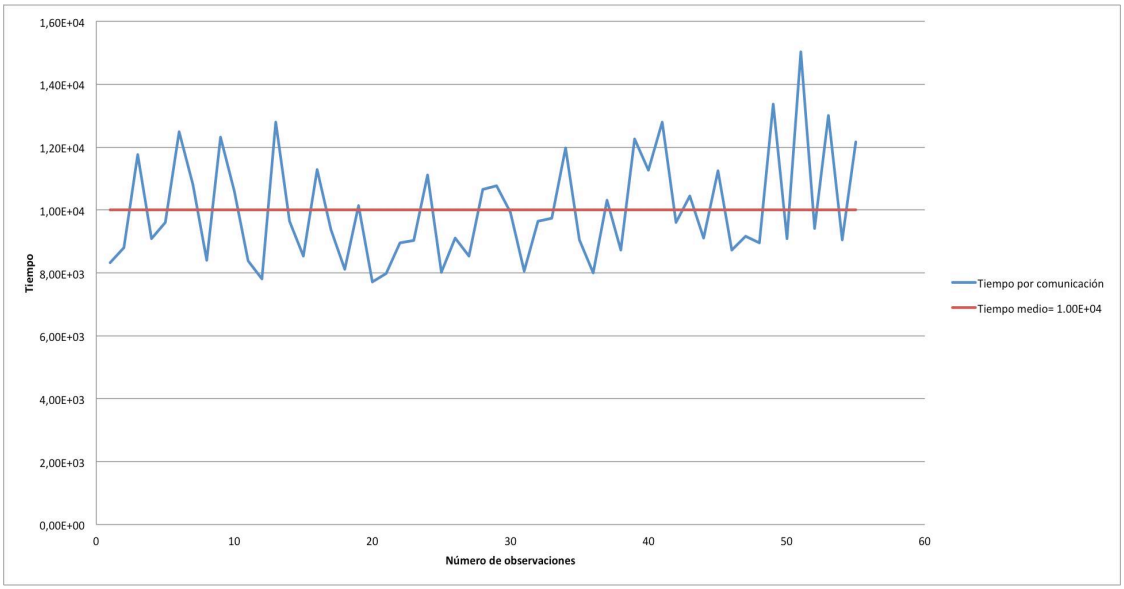


Gráfico 1: Tiempo de comunicación entre el ordenador de procesamiento de imagen y la Raspberry PI.

5.3 Servidor Web

El sistema implementado tiene una interfaz Web para hacer accesible al usuario el control del sistema de seguridad.

Desde esta interfaz Web, el usuario puede ver la imagen del estado actual de su entorno de seguridad e interactuar con los servo-motores instalados para la rotación de la cámara.

Para desarrollar la interfaz mencionada, es preciso implementar un servidor Web en la *Raspberry PI*, que como ya se ha comentado previamente, hace las veces de cámara de red para el sistema.

5.3.1 Definición servidor-cliente HTTP

HTTP significa *HiperText Transfer Protocol* (protocolo de transferencia de hipertexto). HTTP es el protocolo de red que se utiliza para entregar prácticamente todos los archivos y otro tipo de datos, colectivamente llamados recursos, en la World Wide Web, ya sean archivos HTML, archivos de imagen, resultados de la consulta, o cualquier otra cosa.

Un navegador es un cliente HTTP, ya que envía peticiones a un servidor HTTP (servidor Web), que luego envía las respuestas de vuelta al cliente.

5.3.2 Modelo comunicación HTTP entre un cliente y un servidor

Como la mayoría de los protocolos de red, HTTP utiliza el modelo cliente-servidor: un cliente HTTP abre una conexión y envía un mensaje de solicitud a un servidor HTTP; el servidor devuelve un mensaje de respuesta que, por lo general, contiene el recurso que fue solicitado. Después de la entrega de la respuesta, el servidor cierra la conexión (HTTP haciendo un protocolo sin estado, es decir, que no mantiene ninguna información de conexión entre las transacciones). En la Figura 44 se puede observar un esquemático simple de la comunicación entre un servidor y un cliente.

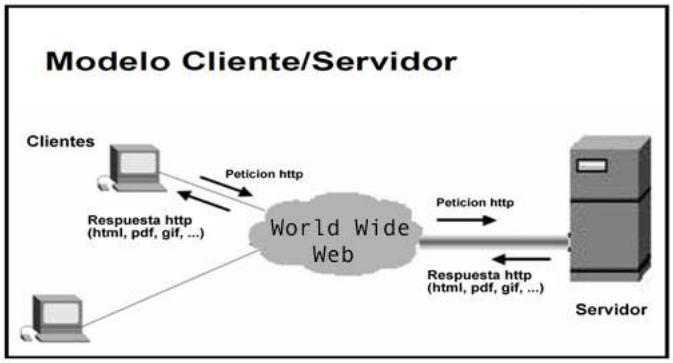


Figura 44: Modelo cliente-servidor HTTP.

5.3.3 Software de servidor HTTP

Para la implementación del servidor Web es necesario poseer *Software* específico del lado del servidor. Este software debe ser capaz de llevar a cabo el intercambio de datos entre el cliente y el servidor. Para esta tarea se decidió utilizar *Apache* (comentado en el Capítulo 4.4). En la Figura 45 se muestra un esquema muy simplificado de la comunicación entre un cliente y el servidor.

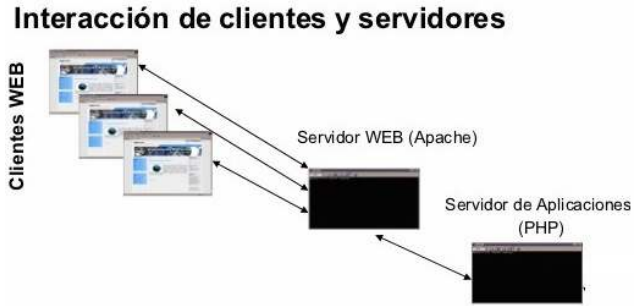


Figura 45: Interacción de clientes y servidor a través de Apache.

Tal y como muestra la figura superior, *Apache* se encarga de habilitar la comunicación entre las aplicaciones desarrolladas del lado del servidor (en este caso el *script* está escrito en lenguaje PHP).

La utilización de *Apache*, además de permitir el intercambio de datos entre el servidor y el cliente, incluye otras ventajas como son la seguridad del servidor. El sistema de archivos almacenado en el ordenador utilizado como servidor no debe ser en ningún caso alterado por el cliente.

5.3.4 Implementación de un servidor web en la Raspberry PI

5.3.4.1 Esquema general

La Figura 46 muestra un diagrama simplificado del funcionamiento del servidor implementado.

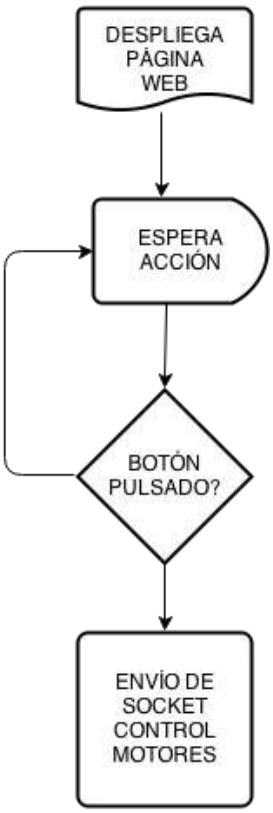


Figura 46: Diagrama de flujo de la interfaz Web con el cliente.

Con el fin de implementar una aplicación Web entre cliente y servidor, se propone una arquitectura de tres capas. Este tipo de arquitectura permite estructurar claramente las aplicaciones web facilitando la adaptabilidad y extensibilidad de las mismas, así como una mayor escalabilidad. Las tres capas estarían conformadas por:

- **Capa de presentación**. Incluye la interfaz gráfica de usuario capaz interactuar con el cliente.

- **Capa de aplicación**. Esta capa implementa la lógica de actuación en la interfaz**.**

- **Capa de datos**. Esta capa implementa el acceso y envío de datos ocultando a las capas superiores los detalles de los repositorios de información.

5.3.4.2 Diagrama de módulos

Un diagrama de módulos muestra la localización de objetos y clases en módulos del diseño físico de un sistema. Un diagrama de módulos representa parte o la totalidad de la arquitectura de módulos del sistema.

Como se puede ver en la Figura 47, se muestra que el módulo principal de la aplicación es *index.php,* el cual se encarga de mostrar el contenido de la página web y llamar al resto de archivos para configurar la apariencia de la página web y la generación de código *JavaScript* (Capítulo 4.2.4) *y AJAX* (Capítulo 4.5).

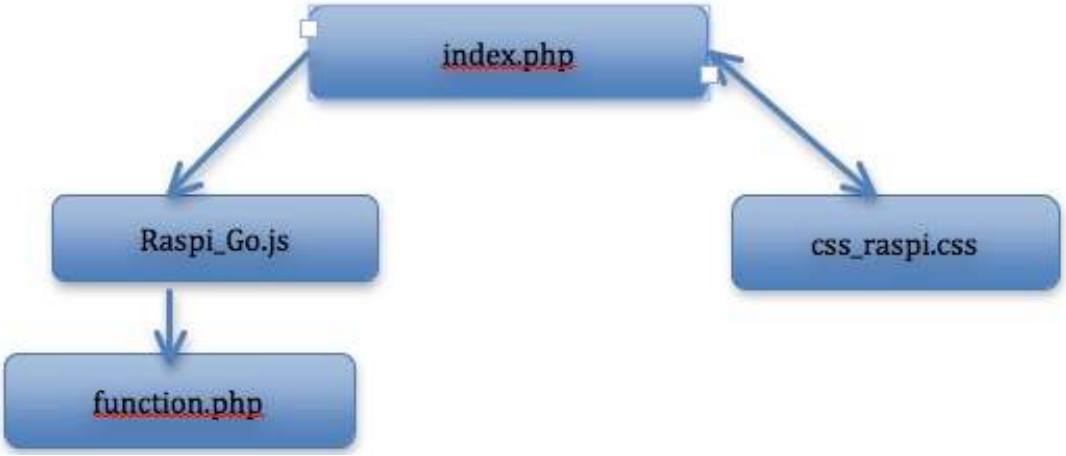


Figura 47: Diagrama de módulos del sistema.

5.3.4.3 Diagrama del sistema de archivos del servidor

En la Figura 48 se adjunta el diagrama de los archivos que forman parte del funcionamiento del servidor web:

Figura 48: Sistema de archivos del servidor Web.

5.3.4.4 Desarrollo de la capa de presentación

La interfaz de usuario es la forma en que los usuarios pueden comunicarse con una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo.

Para el diseño de la interfaz usuario se ha utilizado el lenguaje de *script* HTML (*HiperText Markup Language*) (Capítulo 4.2.3). HTML está basado en etiquetas que organizan los elementos de la página Web que se desea mostrar al usuario. En la Figura 49 se adjunta una imagen de la interfaz usuario implementada.

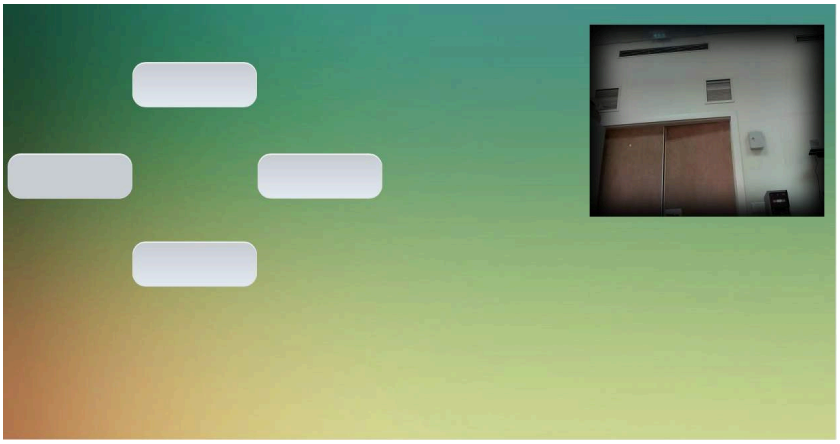


Figura 49: Vista del interfaz Web implementada.

El código *HTML* se organiza en *tags* u objetos que organizan la información desplegada por la página web:

```
<div> <button onclick="jsGo('1')" type="button" class="classname" id="LEFT"></button></div>
<div><button onclick="jsGo('2')" type="button"    class="classname" id="UP"></button>  </div>
```

Figura 50: Ejemplo de HTML para detectar el pulsado de los dos de los botones de la página Web implementada

A la hora de organizar estos objetos se utiliza código en *CSS* (Capítulo 4.2.5). Las hojas *CSS* definen tanto la organización como la presentación de los objetos definidos en *HTML.* Este lenguaje fue inventado debido a que en un principio *HTML* no fue concebido para estructurar el formato las páginas web.

En la Figura 51 se muestra el formato de uno de los botones de la página web:

```
#RIGHT

    {
        max-width:30%;
        max-height:30%;
        min-height:30%;
        min-width: 30%;
        margin-top: 35%;
        margin-left: 60%;
        z-index: 50;
        position: absolute;
        float:UP;
    }
```

Figura 51: Formato CSS de uno de los botones de la interfaz Web implementada.

Las hojas *CSS* también pueden incluir motores gráficos para interactuar con el usuario y conseguir efectos visuales a la hora de interactuar con el usuario. En la Figura 52 se muestra un ejemplo de estos motores aplicado al pulsado de los botones de la página web. Gracias a este tipo de motores gráficos se consigue un mayor realismo cuando el usuario interactúa con la interfaz.

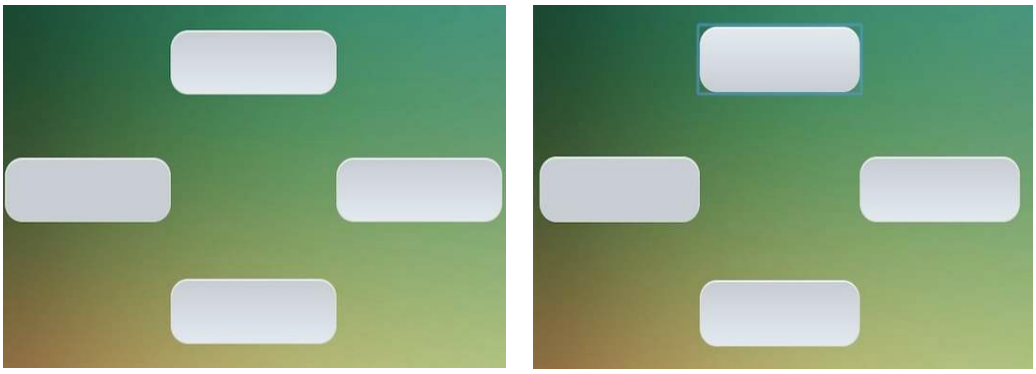


Figura 52: Respuesta gráfica de la interfaz usuario cuando un usuario pulsa uno de los botones mostrados.

5.3.4.5 Desarrollo de la capa de aplicación

La capa de aplicación ha sido desarrollada utilizando el lenguaje *JavaScript.* Como ya se ha definido antes, la capa de aplicación de la página web se encarga de ejecutar la lógica de las operaciones llevadas a cabo por el cliente en la página web.

En la interfaz Web implementada, existen 4 botones con los que el cliente puede interactuar con la rotación de la cámara de vigilancia, rotando la cámara hacia arriba, abajo, derecha e izquierda.

El objetivo de esta capa de aplicación es detectar cuando el usuario ha pulsado uno de los botones.

Para detectar cuando uno de los botones ha sido pulsado, *HTML* dispone de un tipo de *tag* especial llamado *button.* Este tipo de *tag* permite la llamada a una función (de *JavaScript* en este caso) y procesar correctamente la acción llevada a cabo por el usuario. En la Figura 53 se muestra gráficamente este proceso.

Para el procesamiento de datos se asigna un valor 1, 2, 3 o 4, a cada uno de los botones.

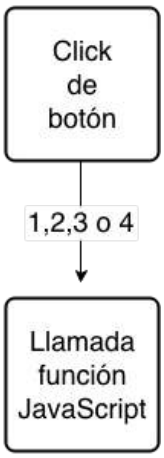


Figura 53: Llamada a JavaScript desde el código HTML cuando el usuario pulsa uno de los botones de la interfaz gráfica.

5.3.4.6 Desarrollo de la capa de datos

Esta es sin duda la capa más compleja a la hora de desarrollar una aplicación web. Como ya hemos definido anteriormente, el código de la página web no se ejecuta el ordenador servidor, sino que se ejecuta en un ordenador cliente (usuario).

5.3.4.6.1 Comunicación del cliente con el servidor

Para poder comunicar el cliente con el servidor (donde se encuentran físicamente los motores del sistema) se utiliza *AJAX.*

AJAX no es un lenguaje de programación, ni tampoco es una nueva tecnología. La utilización de *AJAX* ha sido posible desde que se introdujo el elemento *XMLHttpRequest*. *AJAX* significa "*Asynchronous JavaScript and XML*". La principal ventaja que aporta la utilización de *AJAX* se basa en que no es necesario hacer continuos refrescos de la página web, sino que permite una comunicación asíncrona entre *JavaScript* y el servidor.

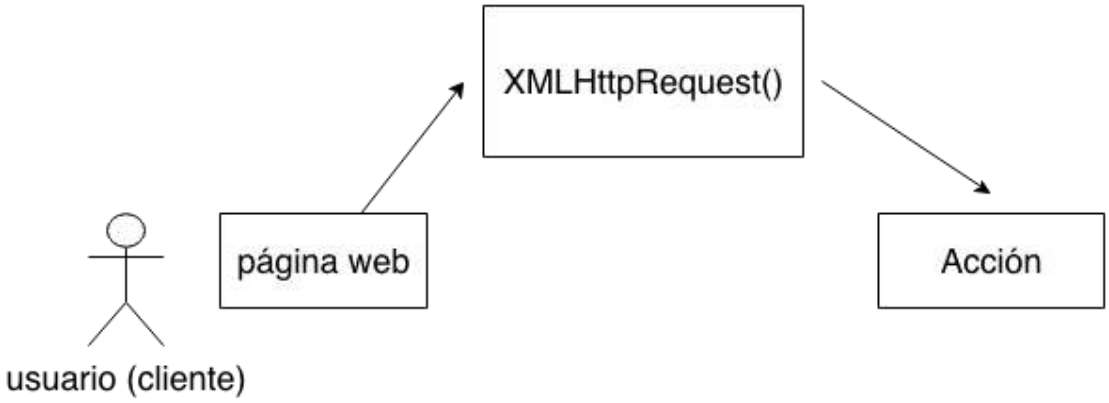


Figura 54: Esquema de comunicación entre el cliente y el servidor utilizando AJAX.

Como ya se ha descrito anteriormente en este capítulo, *HTTP* es un protocolo de transferencia de datos entre un servidor y un cliente. Los intercambios de información entre el cliente y el servidor se llevan a cabo mediante peticiones de cliente y respuestas del servidor.

Para llevar a cabo esta comunicación *HTTP* consta de varios métodos aunque los más comunes son:

- GET: peticiones de datos de un recurso específico.

- POST: envía datos para ser procesados.

El procesamiento de los datos enviados por el cliente, se lleva a cabo en el *Script function.php*, y se puede ver su relación en el sistema de archivos en la Figura 54.

Para el envío de los datos anteriormente procesados en la capa de aplicación, *JavaScript* hace una petición al servidor desde el cliente a través de AJAX con el método *GET* al *Script function.php.*

Para facilitar la comprensión del proceso se adjunta un diagrama de flujo (Figura 55):

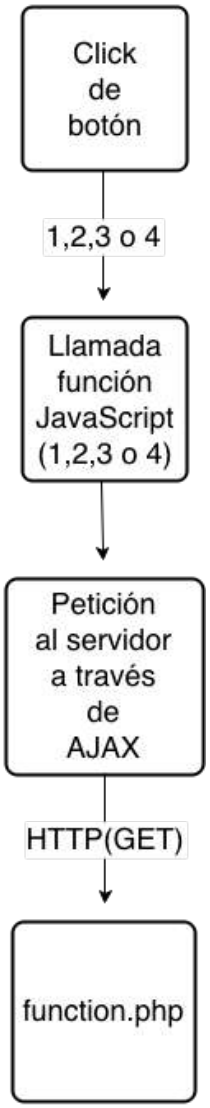


Figura 55: Comunicación con el servidor instalado en la Raspberry PI desde un cliente.

5.3.4.6.2 Comunicación del servidor con el controlador de los motores

Para la comunicación entre el servidor descrito y el proceso controlador de servos se utilizan *sockets.*

Como ya se ha comentado, el servidor de la página web está alojado en el mismo ordenador que se encarga de la administración de las comunicaciones entre procesos y el control de la rotación de los servo-motores (estas partes del proyecto ya han sido comentadas en los capítulos previos).

Teniendo en cuenta que se ha instalado un servidor escrito en lenguaje C (Capítulo 5.2), se necesita un modo de comunicación entre nuestro proceso de interfaz con el usuario (escrito en lenguaje de programación *PHP),* y el proceso

encargado del control de los servo-motores (escrito en lenguaje de programación C).

Para establecer un canal de comunicación entre estos dos procesos se han utilizado *sockets* del dominio *Unix* (Capítulo 4.6.2).

De este modo, la interfaz web usuario queda definida como el cliente del servidor de dominio *Unix* definido en el capítulo anterior.

5.3.4.6.3 Implementación de un socket cliente en PHP

Previamente se ha definido que desde el cliente del servidor se envían cuatro posibles 1, 2, 3 o 4. Estos valores deben ser comunicados al controlador de los servos. Para enviarlos se utilizan *streams* de caracteres.

Para la implementar un *socket* cliente en *PHP* se han seguido los siguientes pasos:

1. Se define una variable *socket* con el descriptor de archivos *Unix* por donde la información va a ser enviada:

```
$socket = stream_socket_client('unix:///var/run/mysocket.sock',
                               $errorno, $errorstr, $timeout);
```

Figura 56: Creación de un socket cliente en el dominio Unix utilizando PHP.

2. Se utiliza el método *fwrite* parda mandar la cadena de caracteres a través del *socket* definido:

```
fwrite($socket, $snd_msg)
```

Figura 57-a: Envió de un mensaje a través de un socket del dominio Unix.

3. Se recoge la respuesta del servidor para asegurarnos de que el mensaje ha llegado:

```
$rcv_msg = fread($socket, 1024)
```

Figura 57-b: Recogida de la respuesta del servidor socket con el que nos estamos comunicando.

5.3.5 *Streaming* de imágenes al servidor Web

Para conseguir que las imágenes que están siendo grabadas por el sistema de seguridad puedan ser visibles tanto al usuario como al ordenador de procesamiento de imagen, se utiliza *VLC.*

VLC es un programa *open source* que permite crear un *stream* de imágenes, y mostrarlas en una dirección IP. No es el objetivo de este proyecto el análisis del *streaming* de imágenes, dada su elevada complejidad.

En el proyecto realizado *VLC* se encarga de obtener imágenes desde la *RaspiCam*, convertirlas al formato .mjpeg (un formato estándar de vídeo) y hacer un crear un *stream* de imágenes en una dirección IP concreta. Desde la página Web explicada, capturamos el *streaming* de imágenes y los mostramos al usuario.

Para el hacer el *streaming* de imágenes se escoge un *frame-rate* de 15 imágenes por segundo, que es el parámetro máximo soportado por el programa.

5.3.6 Diagrama UML

En la Figura 58 se muestra un diagrama UML del servidor Web desarrollado:

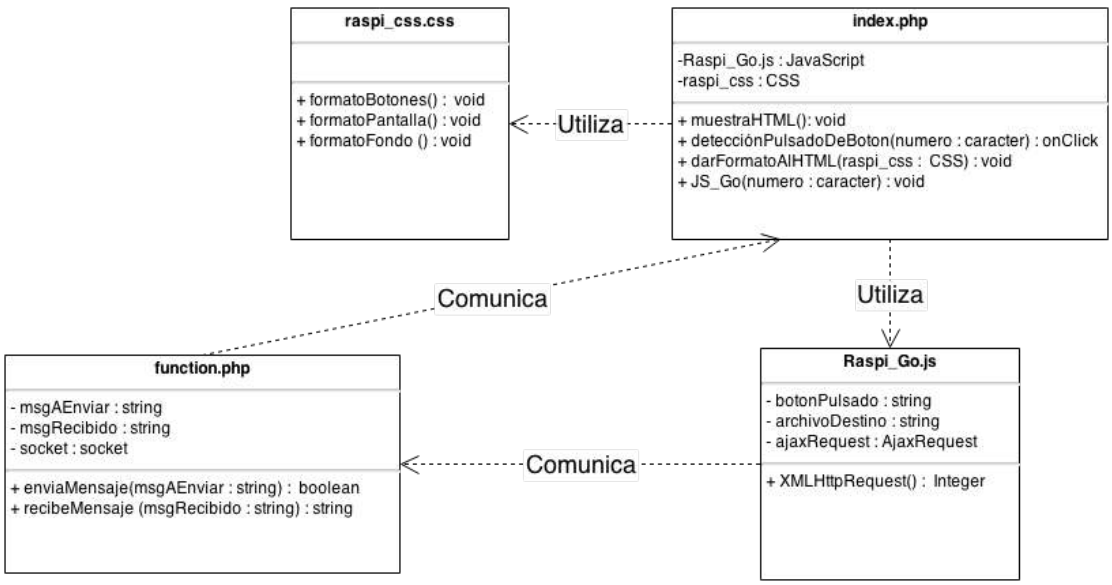


Figura 58: UML del servidor Web.

5.4 Procesamiento de imagen

Para llevar a cabo el procesamiento de imagen del sistema de seguridad, se utiliza un ordenador conectado mediante *Ethernet* a la *Raspberry PI.* El proceso es el siguiente: la *Raspberry PI* envía un *stream* de imágenes a una dirección IP, el ordenador de procesamiento de imagen recoge estas imágenes grabadas, las procesa y se comunica con la *Raspberry PI* para controlar la rotación de los servo-motores.

En los sucesivos capítulos se abordará el algoritmo propuesto para la detección y seguimiento de intrusos en el entorno de seguridad. Las partes más importantes del algoritmo propuesto tienen que ver con:

- Substracción de fondo
- Detección de personas
- Seguimiento

5.4.1 Adquisición de imágenes

Dado que el procesamiento de imagen para un sistema de seguridad debe ser en línea, primeramente es necesario obtener las imágenes grabadas en tiempo real por el sistema de seguridad.

Con el fin de poder capturar un *stream* de imágenes desde la *Raspberry PI* se instala la librería *Open CV* con un módulo extra: *ffmpeg.* Con el módulo mencionado es posible acceder a un *stream* de imágenes enviado a una dirección IP desde el servidor del sistema y almacenarlo en un *buffer* de imágenes.

El *stream* de imágenes se emite desde la *Raspberry* utilizando el programa *VLC.* Existe un *frame-rate* de aproximadamente 15 *frames/segundo*, con un *delay* de 100 *ms.*

5.4.2 Substracción de fondo

La substracción de fondo es uno de los métodos más utilizados de segmentación de imágenes.

En el algoritmo propuesto la substracción de fondo cobra una especial importancia, debido a que en función de la segmentación obtenida en este módulo, el resto de módulos del algoritmo funcionarán mejor o peor.

Además, la substracción de fondo se encarga de detectar movimiento en la escena, por lo tanto, tiene una relación directa con el consumo de recursos computacionales del resto del algoritmo. En principio, la substracción de fondo es una operación más sencilla, en términos de recursos computacionales utilizados, que el resto de módulos del algoritmo.

Dado que la substracción de fondo se encarga de determinar si ha existido movimiento en el sistema, la incorrecta configuración de este módulo del algoritmo traerá como consecuencia un mayor gasto computacional, así como un mayor número de falsos positivos.

La operación de substracción del fondo de una imagen consiste como norma general en: "determinar qué regiones de la imagen han cambiado desde la imagen (t - 1) a la imagen (t)"

Existen numerosas técnicas y algoritmos dedicados a la substracción de fondo (4). Para el desarrollo de este proyecto se consideraron:

- Diferencia entre imágenes
- Diferencia entre imágenes aplicando un filtro gaussiano
- Mixturas de Gaussianas.

5.4.2.1 Diferencia entre imágenes

Esta técnica trata de distinguir la diferencia entre dos imágenes sucesivas calculando cuantos píxeles han cambiado desde la imagen (t - 1) hasta la imagen (t).

Una imagen se compone de píxeles. Los píxeles no son más que pequeñas secciones rectangulares de una imagen. Si vemos una imagen como un matriz con una serie de elementos (píxeles) incrustados en ella, podremos definir que cada píxel tiene asociado un valor. Si la imagen está en escala de grises, cada píxel tendrá un valor entre 0 y 255 (si codificamos cada píxel de la imagen en 8 bits). En cambio si la imagen está en color, cada píxel estará conformado por 3 canales RGB (*Red, Green, Blue*) cada uno de ellos con un valor entre 0 y 255.

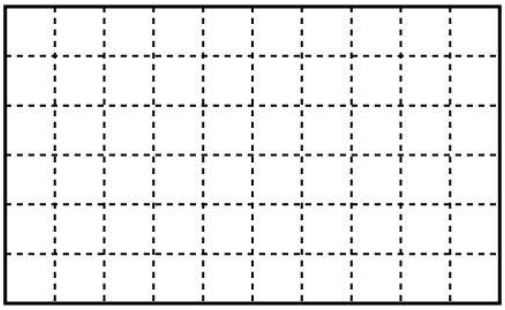


Figura 59: Matriz de píxeles de una imagen.

Una vez caracterizado el valor de los píxeles de una imagen (Figura 59), es posible encontrar la diferencia entre dos imágenes sucesivas:

$$D(t\ +\ 1) = |\ V(x, y, t + 1) -\ V(x, y, t)| \quad [2.1.1]$$

- *x:* se refiere al número de columnas de la imagen.
- *y*: se refiere al número de filas de la imagen.
- *D:* máscara obtenida (*foreground*).
- *V (t+1)*: imagen actual de una secuencia de imágenes.

El principal problema de esta operación está relacionado con el hecho de que cualquier perturbación en la imagen tomada (cambios lumínicos, ruido, etc) afectará terriblemente a la máscara de *foreground* obtenida. En la Figura 60 se muestra qué podría suponer un cambio lumínico excesivo:

Figura 60: Cuando un usuario abre la puerta del entorno de vigilancia se genera un cambio lumínico brusco.

5.4.2.2 Modelo Gaussiano

En el capítulo anterior se explicó cómo se puede obtener la diferencia entre dos imágenes. Una mejora significativa a este método es la aplicación de filtro *Gaussiano* a la imagen.

Aplicando un filtro *Gaussiano* a una imagen, es posible crear un modelo del fondo de la imagen (*background*) a través de la acumulación de otras imágenes de fondo. De este modo, la substracción de fondo se hace más robusta al ruido ambiental.

Si consideramos que el valor medio de un píxel se puede obtener como:

$$\mu_t\ =\ I_t\rho + (1 -\ \rho)\mu_{t-1} \quad [2.2.1]$$

- I: intensidad de un píxel.
- μ_t : media de las intensidades.
- ρ: tamaño de la ventana de píxeles donde queremos aproximar una *Gaussiana.*

Nótese que en el caso de tener una imagen en *RGB* tendríamos que aplicar el filtro a los tres canales de la imagen. Una vez obtenida la media de la intensidad del píxel se obtiene la varianza:

$$\sigma_t{}^2 = d^2\rho + (1 - \rho)\sigma_{t-1}{}^2 \quad [2.2.2]$$

- $\sigma_t{}^2$: varianza de las muestras.
- ρ : tamaño de la ventana de píxeles donde queremos aproximar una *Gaussiana.*

En la ecuación [2.2.2] el valor d se refiere a la distancia *Euclidea* entre el valor del píxel y la media de la gaussiana:

$$d = |I_t - \mu_t| \quad [2.2.3]$$

La decisión de primer plano consiste en clasificar un nuevo valor del píxel como fondo (*background*) o como primer plano (*foreground*) en función de la probabilidad que tenga el píxel de pertenecer a uno de estos dos estados.

5.4.2.2.1 Estimación del primer plano

Para clasificar si un nuevo valor (x_t) es primer plano (FG) o es fondo (BG) se calcula la distancia de *Mahalanobis* entre el nuevo valor y la función de probabilidad de fondo del píxel, finalmente se aplica un umbral probabilístico (*Threshold*) para decidir si el píxel pertenece al *background* o al *foreground*:

$$\left|\frac{\mu - x_t}{\sigma}\right| > Threshold \rightarrow foreground \quad [2.2.4]$$

$$\left|\frac{\mu - x_t}{\sigma}\right| < Threshold \rightarrow background \quad [2.2.5]$$

5.4.2.2.2 Resultados del modelo Gaussiano

Con el fin de demostrar el modelado de fondo en función de una *Gaussiana*, se han muestreado las intensidades del píxel central de una secuencia de imágenes, donde la Figura 61 representa el fondo de la imagen:

Figura 61: Se ha utilizado el píxel central de una secuencia de imágenes donde aparece la puerta del despacho LSI para obtener las intensidades de un píxel (Gráfico 2).

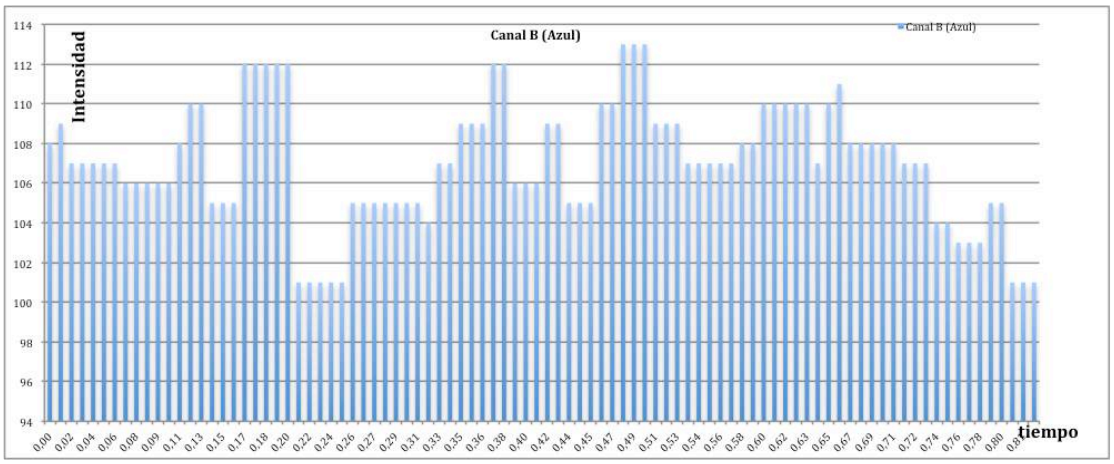


Gráfico 2: Intensidades seguidas por el píxel central de la Figura 60 a los largo de 819 milisegundos.

A continuación se muestra un histograma con las intensidades alcanzadas por el píxel a lo largo de 819 milisegundos de muestreo (Gráfico 2). Junto al histograma se muestra la normal *Gaussiana* del conjunto de muestras:

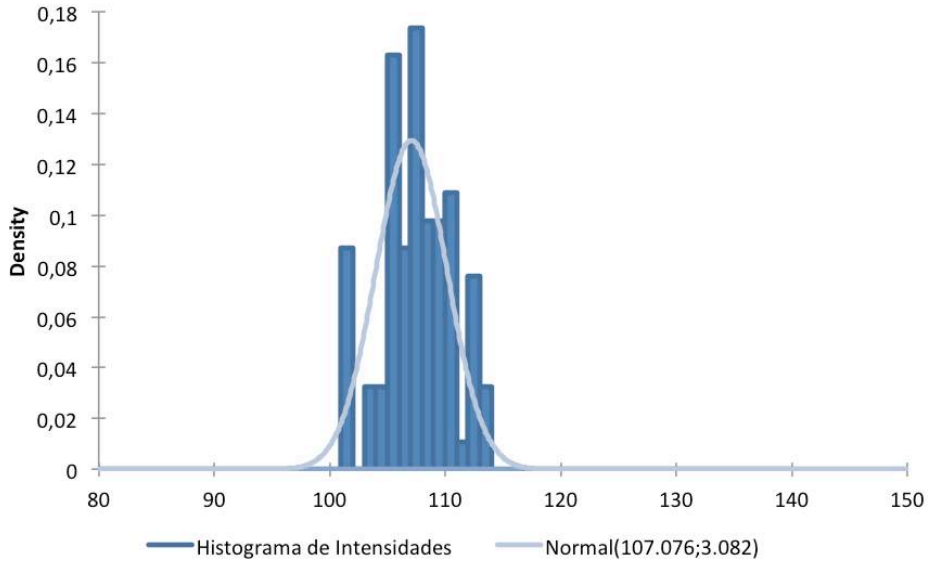


Gráfico 3: Histograma 90 muestras con aproximación Gaussiana al modelo.

Es sencillo apreciar como con un gran número de muestras (se han utilizado 90 muestras) el comportamiento de la intensidad tiende a ajustarse a una normal (Gráfico 3). No obstante, utilizando un elevado número de muestras, la adaptabilidad del modelo a cambios bruscos en la iluminación u otros tipos de ruido (por ejemplo: si tenemos una nube que está cruzando por delante de la ventana), decrece.

Para mejorar la adaptabilidad del modelo es necesario tomar un número menor de muestras. En el Gráfico 4 se muestra un histograma con 40 muestras:

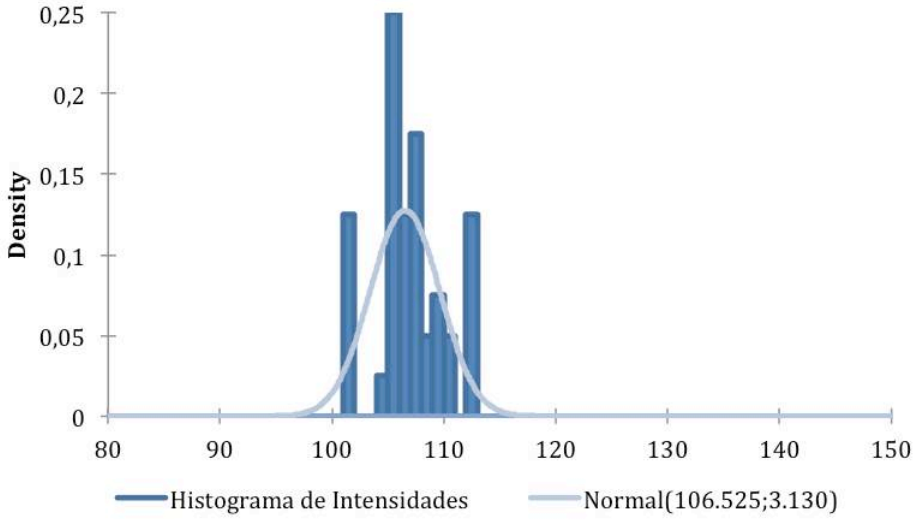


Gráfico 4: Histograma 40 muestras con aproximación Gaussiana al modelo

En el Gráfico 4 el modelo de la normal *Gaussiana* se adapta mucho peor a las intensidades seguidas por el píxel estudiado. El Gráfico 3 demuestra que el modelado del fondo en función de una única *Gaussiana*, necesita un elevado número de muestras, lo que generalmente reducirá la adaptabilidad del modelo de fondo.

5.4.2.3 Mixturas Gaussianas

El modelado de un pixel a través de una *Gaussiana* comete errores cuando el número de muestras no es suficientemente elevado. Para solucionar el problema se explica el modelo planteado por *Stauffer* y *Grimson* (5) (6), en el que se usan varias distribuciones *Gaussianas* por píxel.

La idea general del algoritmo, consiste en modelar cada uno de los píxeles de la imagen de fondo a través de una suma ponderada de K distribuciones *Gaussianas*.

El algoritmo desarrollado por *Stauffer* y *Grimson*, establece que la probabilidad de observar el valor actual de un píxel viene dada por:

$$P(x_t) = \sum_{i=1}^{k} w_{i,t} \cdot \eta(x_t, \mu_{i,t}, \sum_{i,t}) \quad [2.3.1]$$

$$\sum_{i=1}^{k} w_{i,t} = 1 \quad [2.3.2]$$

Donde el parámetro K se refiere al número de distribuciones del modelo, $w_{i,t}$ es un peso asociado a la Gaussiana (i) en el tiempo t con μ_i como media y varianza $\sum_{i,t}$. En la ecuación [2.3.1] η es una función de densidad de probabilidad *Gaussiana*:

$$\eta(x_t, \mu, \sum) = \frac{1}{(2\pi)^{n/2} |\sum|^{1/2}} \cdot e^{\frac{-1}{2}(x_t - \mu)\sum^{-1}(x_t - \mu)} \quad [2.3.3]$$

Por razones de computacionales, *Stauffer* y *Grimson* supusieron que los componentes de color *RGB* son independientes y tienen las mismas varianzas. Por lo tanto, la matriz de covarianza es de la forma:

$$\sum_{i,t} = \sigma_{i,t}^2 \cdot I \quad [2.3.4]$$

Donde I representa la matriz identidad.

Para que el modelo sea efectivo, el modelo debe utilizar como mínimo dos *Gaussianas*. Esto se debe a que en el caso de utilizar una única *Gaussiana* el peso de esta sería igual a 1, pudiendo modelar el fondo a través de una única distribución.

5.4.2.3.1 Estimación del estado

Cuando un nuevo píxel entra en el algoritmo debemos decidir si encaja con la función de probabilidad actual y a cuál de las *Gaussianas* presentes se adapta mejor. La decisión se toma en base a:

$$\eta(x_t, \mu, \Sigma_{i,t}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x_t-\mu)\Sigma^{-1}(x_t-\mu)} \quad [2.3.5]$$

Una vez encontrada la probabilidad con la cada una de las *k Gaussianas* del modelo, se decide cuál es la mayor probabilidad de pertenencia a uno de los *K* modelos. Si el píxel sobrepasa un *Threshold* (probabilidad de pertenencia de ese píxel a la distribución) se clasificará como primer plano, si el píxel se encuentra por debajo del *Threshold* se incluirá la medida en la *Gaussiana k*.

Los valores que forman parte del modelo se usan para actualizar los parámetros de su *Gaussiana* correspondiente. Para aquellos valores que hayan sido clasificados como fuera del modelo se les asignará una nueva Gaussiana que substituirá la *Gaussiana* menos probable ($\omega_{t,k}$ menor) en ese píxel. Esta nueva *Gaussiana* tendrá como parámetros:

- $\mu_{t,k} = x_t$: la nueva *Gaussiana* estará centrada en el nuevo valor que no formaba parte del modelo.

- $\sigma^2_{t,k} = \sigma^2_{init}$: la varianza de la *Gaussiana* vendrá determinada por la varianza inicial del modelo.

- $w_{t,k} = w_{init}$: El peso inicial de la Gaussiana será un valor bajo pero mayor que 0. Hecha esta asignación, será necesario corregir el resto de pesos para que la suma de todos los pesos siga siendo 1.

5.4.2.3.2 Actualización de parámetros

Para actualizar los parámetros $(\mu_t, \Sigma_t, \omega_t)$ de las K *Gaussianas* que componen el modelo, *Stauffer* y *Grimson* solucionaron este problema utilizando el algoritmo *EM (Expectation Maximization*). El algoritmo EM funciona iterando dos pasos: (E-paso) encontrar el valor esperado con respecto a los datos ocultos de la función de verosimilitud de los datos completos (observado y oculto) utilizando los datos observados y las estimaciones actuales de los parámetros; y (M-paso) el cálculo de las estimaciones de máxima verosimilitud de los parámetros con los datos observados y las estimaciones actuales de los datos ocultos.

5.4.2.3.2 Decisión de primer plano

Para determinar qué constituye el primer plano de nuestra imagen es necesario discriminar qué píxeles pertenecen al modelo de fondo de nuestra imagen, y qué píxeles no. Consideraremos elementos del fondo, todos aquellos elementos que tengan $w_{t,k}$ alto y tengan una $\sigma_{t,k}$ baja. Se establece un criterio para discernir qué pertenece al fondo de la imagen:

$$\frac{\omega_{t,k}}{\sigma_{t,k}} \quad [2.3.6]$$

Estableciendo el criterio de confianza mostrado en la ecuación superior, básicamente estamos midiendo la amplitud del pico de nuestra distribución, ya que se divide $\omega_{t,k}$ la altura de la *Gaussiana* entre la anchura de su base $\sigma_{t,k}$.

Para establecer el fondo se determina un umbral de la probabilidad total de las Gaussianas de fondo (T). Los primeros B elementos (en el orden del apuntamiento) que acumulen la probabilidad T (*Threshold*) serán considerados fondo:

$$B = min_b(\Sigma_{k=1}^{b} \omega_k > T) \quad [2.3.7]$$

5.4.2.3.2 Resultados del modelo de Mixturas de Gaussianas

El modelo de *Mixturas Gaussianas* exige una correcta elección del parámetro *K* (número de *Mixturas*). Según *Stauffer y Grimson* (5), este parámetro debe estar entre 2 y 5, ya que menos de 2 M*ixturas* equivale a un modelo basado en una normal *Gaussiana* y a partir de 7 *Mixturas* el modelo no presenta cambios significativos. En el Gráfico 5 se muestra el histograma sobre el que se va a ajustar un modelo de *Mixturas* con diferentes valores para *K.*

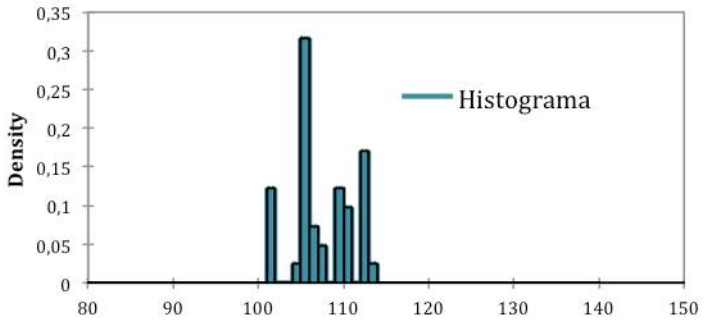


Gráfico 5: Histograma 40 muestras de las intensidades de un píxel.

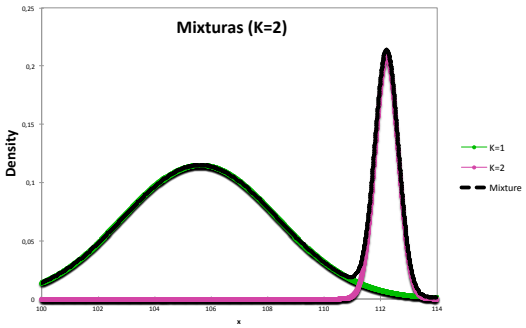


Gráfico 6: Modelo de Mixturas Gaussianas K=2

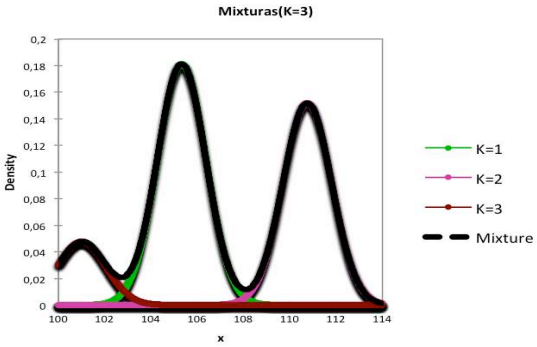


Gráfico 7: Modelo de Mixturas Gaussianas K=3

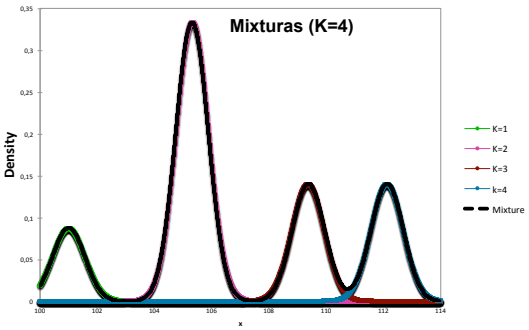


Gráfico 8: Modelo de Mixturas Gaussianas K=4

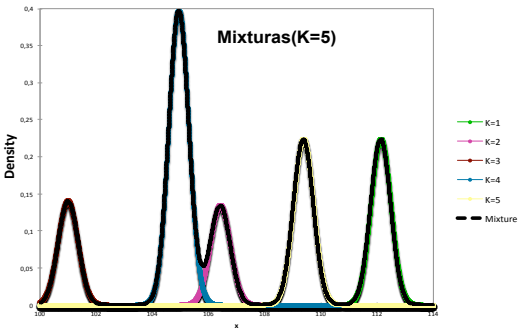


Gráfico 9: Modelo de Mixturas Gaussianas K=5

Observando los Gráficos 6, 7, 8 y 9, se puede determinar que un parámetro K óptimo para la substracción de fondo basado en *Mixturas Gaussianas* estaría entre 4 y 5 *Gaussianas* por *Mixtura*.

5.4.2.4 Elección de la técnica para substraer el fondo

El problema de modelar el fondo a través de una única distribución *Gaussiana* radica en que es necesario tomar muchas muestras para definir correctamente el fondo de la imagen, y al aumentar el número de muestras disminuimos la adaptabilidad de nuestro algoritmo.

Las *Mixturas de Gaussianos* son un método más robusto para modelar el fondo de la imagen. No solo requieren menos muestras para la correcta definición de nuestro fondo, sino que en general, se ajustan mejor a las intensidades seguidas por los píxeles de la imagen.

En la figura inferior (Gráfico 10) se muestran los dos modelos utilizando el histograma mostrado en el Gráfico 5:

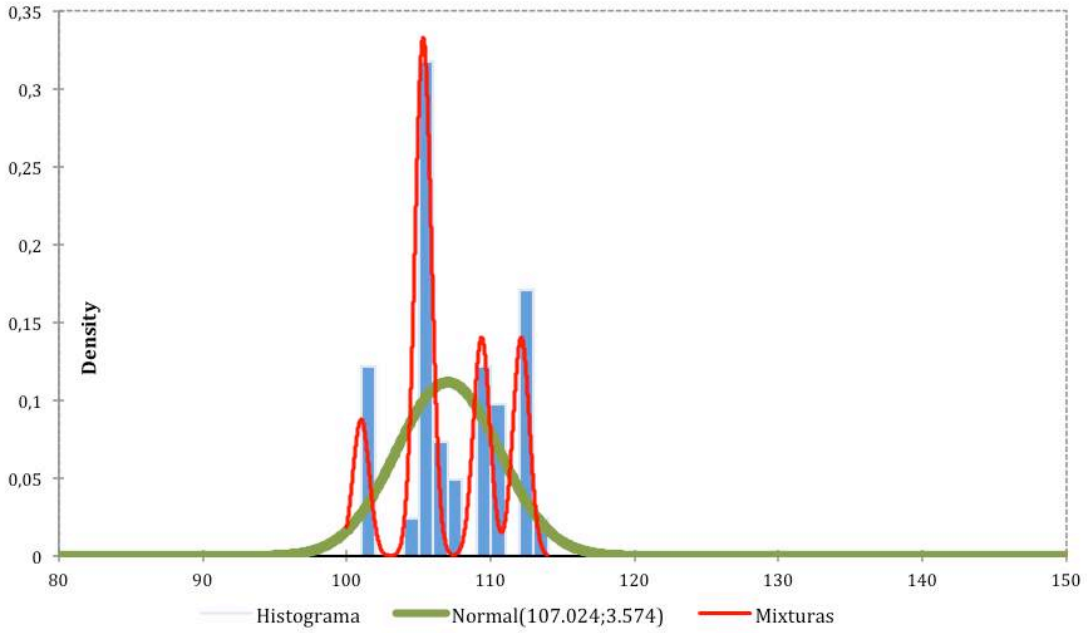
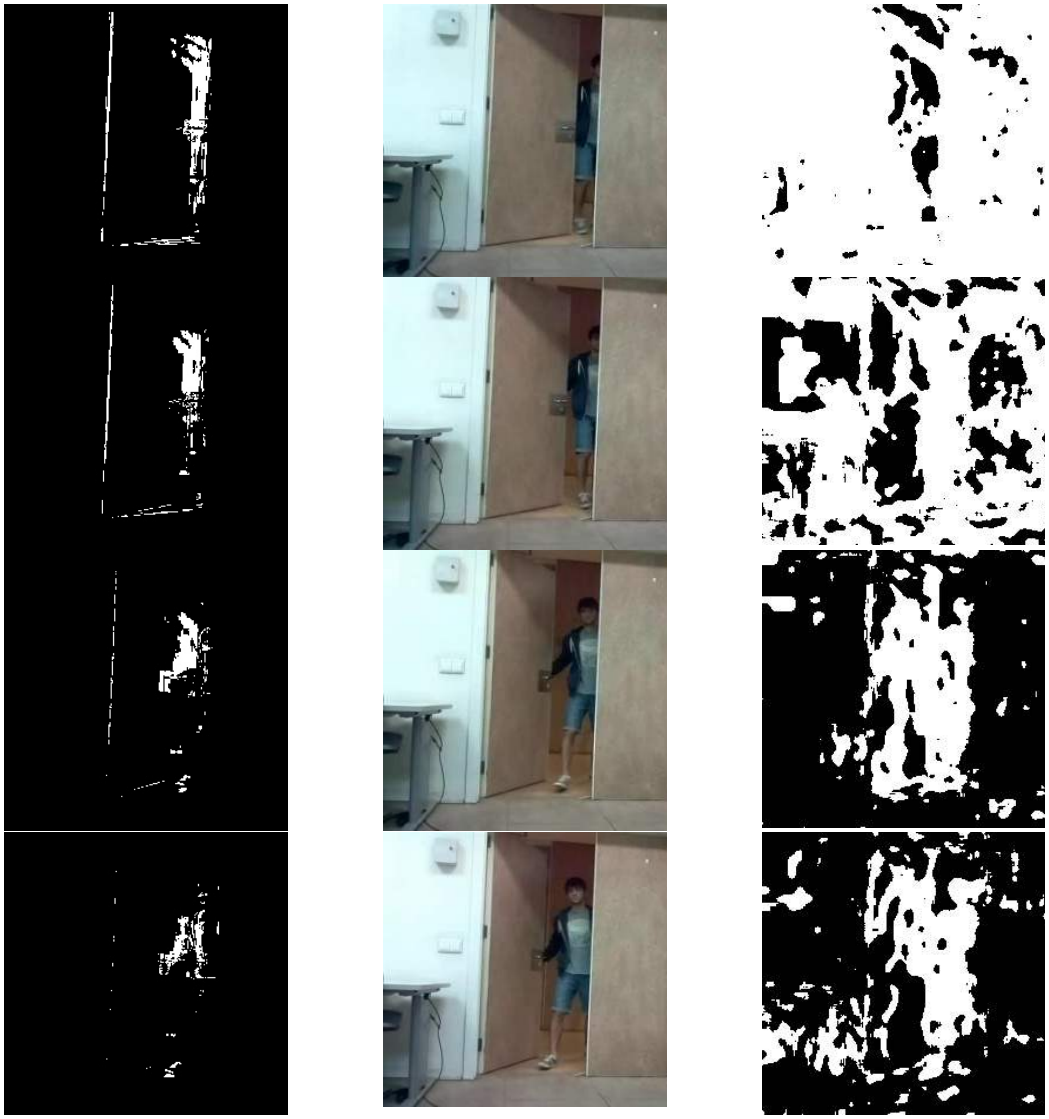


Gráfico 10: Modelo de Mixturas Gaussianas enfrentado con el modelo de una única Gaussiana.

Es sencillo notar que el modelo de *Mixturas Gaussianas* se adapta mucho mejor a las intensidades seguidas por el píxel.

Para el diseño del algoritmo de substracción de *background* se decidió utilizar el modelo de *Mixturas Gaussianas* (*K=5, Threshold=0.1*) en base a la mejor adaptabilidad al fondo de la imagen. En la Figura 62 se muestran algunos resultados obtenidos con cada una de las dos técnicas:



Mixturas de Gaussianos K=5 Modelos acumulados=5 Threshold=0.1

Imagen utilizada

Modelo de una *Gaussiana Modelos acumulados=5 Threshold=0.1*

Figura 62: Resultados obtenidos aplicando un modelo de Mixturas Gaussianas, enfrentados con los resultados obtenidos mediante la aplicación de un modelo basado en una única Gaussiana.

La apertura de la puerta genera un cambio lumínico importante en el entorno. El modelo basado en una única *Gaussiana* no es capaz de reconocer correctamente qué forma parte del fondo y qué no ante esta situación de variación brusca de la iluminación. Si observamos el modelo basado en *Mixturas*

de Gaussianos, la respuesta de este siempre es mejor, permitiendo apoyar etapas siguientes del seguimiento de personas en la segmentación realizada por este algoritmo.

Finalmente se obtienen los siguientes parámetros para el modelo de Mixturas de *Gaussianos* (Tabla 3)*:*

K	*5*
Modelos de fondo acumulados	*5*
Threshold	*0.1*

Tabla 3: Parámetros utilizados para el modelo de substracción de fondo basado en Mixturas Gaussianas.

5.4.3 Detección de personas

La detección de personas en el sistema realizado se lleva a cabo a través de *HOG* (*Histograms Oriented Gradients*). *HOG* fue desarrollado por *Navneet Dalal y Bill Triggs* (7)*.* El método se basa en la evaluación de las orientaciones de los gradientes de una imagen en un sistema de cuadrículas.

La idea principal del algoritmo reside en que generalmente, se puede modelar la apariencia y forma de un objeto en base a la distribución de los gradientes de intensidad locales, o direcciones de borde.

Las características de utilizadas por *HOG* son similares al descriptor SIFT (1), la información de la orientación de los gradientes se utiliza para incorporar características de los objetos que se desean detectar en la imagen
(7), (8).

Las características del HOG son descripciones de los histogramas de orientación del gradiente de una imagen y se pueden calcular de la siguiente manera:

En la primera etapa del algoritmo se busca obtener los gradientes de una imagen. De este modo cada píxel queda caracterizado por un gradiente en dos direcciones.

En la segunda etapa, la ventana de detección de imagen se divide en pequeñas regiones espaciales, llamadas celdas. Para cada celda, se calcula un histograma local de 1 - D de la orientación del gradiente en esa celda. La acumulación de las orientaciones del gradiente en cada una de las celdas es lo que se como histograma de orientaciones. La acumulación de los histogramas de orientación será utilizada como vectores de características para detectar el objeto. Esta acumulación de los gradientes de una imagen se lleva a cabo a través del cómputo del histograma dentro de bloques de agrupación de las celdas.

La tercera etapa trata de normalizar los resultados de la etapa anterior con el fin de obtener una menor variación ante cambios lumínicos, sombras, etc. Esto se puede hacer mediante la acumulación de una medida de histograma "energía" local de los " bloques". El resultado se utiliza para normalizar cada célula en un bloque. La celda aparece así varias veces en el vector de salida final con diferentes normalizaciones. A continuación, los descriptores de bloque normalizados se definen como Histogramas de Gradiente Orientados (*HOG*).

La última etapa recoge los descriptores HOG de todos los bloques de la ventana de detección en un vector de características para poder clasificar estos correctamente.

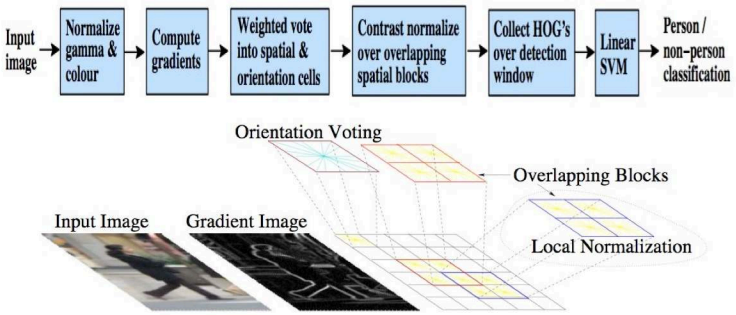


Figura 63: Diagrama de flujo ofrecido por los autores del algoritmo.

En la Figura 63 se muestra el diagrama de flujo mostrado por (8).

En las Figuras 64 y 65 se muestran algunos de los resultados obtenidos por el algoritmo *HOG* desarrollado por el Doctor Fernando García Fernández.

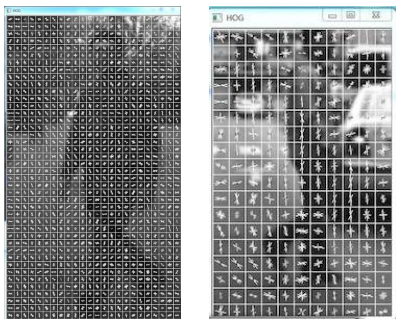


Figura 64: Orientaciones del gradiente de una imagen donde aparece una persona obtenidos por el Doctor Fernando García Fernández.

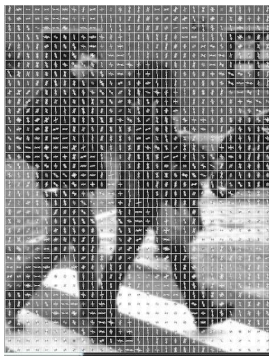


Figura 65: Orientaciones del gradiente de una imagen donde aparecen varias personas obtenidos por el Doctor Fernando García Fernández.

Para la detección de personas el método implementado por la librería *Open CV* permite la utilización de una base de datos por defecto. Generalmente el "entrenamiento" del algoritmo, a través de imágenes con positivos y falsos positivos de una detección, requiere la correcta catalogación de entre 1000 y 1500 imágenes donde puedan aparecer, o no, personas. Dada la excesiva complejidad y tiempo que requiere este entrenamiento, se decidió utilizar el entrenamiento ya implementado en la librería.

El método de la librería, permite la introducción de una serie de parámetros por el usuario, estos son:

- Tamaño de las celdas donde se van a obtener las orientaciones del gradiente de la imagen.

- Escalado del agrupamiento de las celdas.

- Agrupación cuando ocurren varias detecciones en la misma región.

- *Threshold* requerido para la detección, que nos indicará la certidumbre en la detección obtenida.

Generalmente, cuanto menor sea el escalado de los bloques donde se van a agrupar los histogramas de la orientación del gradiente, el algoritmo mejora su detección disminuyendo los falsos positivos. Dado que no suponía un aumento excesivo del tiempo de cómputo, para la implementación de la detección de personas se ha decidido utilizar un escalado de celdas mínimo (1.05).

El parámetro de la agrupación de rectángulos es utilizado para decidir cuantas detecciones se pueden agrupar en un mismo rectángulo de detección. Cuando varias regiones de detección (rectángulos) se superponen entre sí, este parámetro indica al algoritmo cuántas de ellas se pueden llegar a colocar en la misma detección. Nótese que si el algoritmo detecta varias detecciones en una zona de la imagen, lo más común es que sea la misma persona.

El parámetro más importante del algoritmo quizás sea el *Threshold*. Durante las pruebas realizadas con el algoritmo de la librería *Open CV*, se comprobó que este parámetro depende excesivamente de la luz ambiental presente, en ese momento, en el entorno de vigilancia. Es importante remarcar que es posible conseguir resultados fiables con este algoritmo, no obstante es necesario "entrenar" el algoritmo con una base de datos con imágenes propias, con positivos y falsos positivos. El inconveniente de estos entrenamientos radica en que es necesario utilizar entre 1000 y 1500 imágenes. Debido al largo tiempo que hubiese requerido esta tarea, así como la disponibilidad de sujetos de pruebas (personas en el entorno) se decide utilizar el entrenamiento por defecto de la librería. En la Figura 66 se muestran algunos resultados obtenidos sobre la variación de este parámetro en una misma secuencia de imágenes:

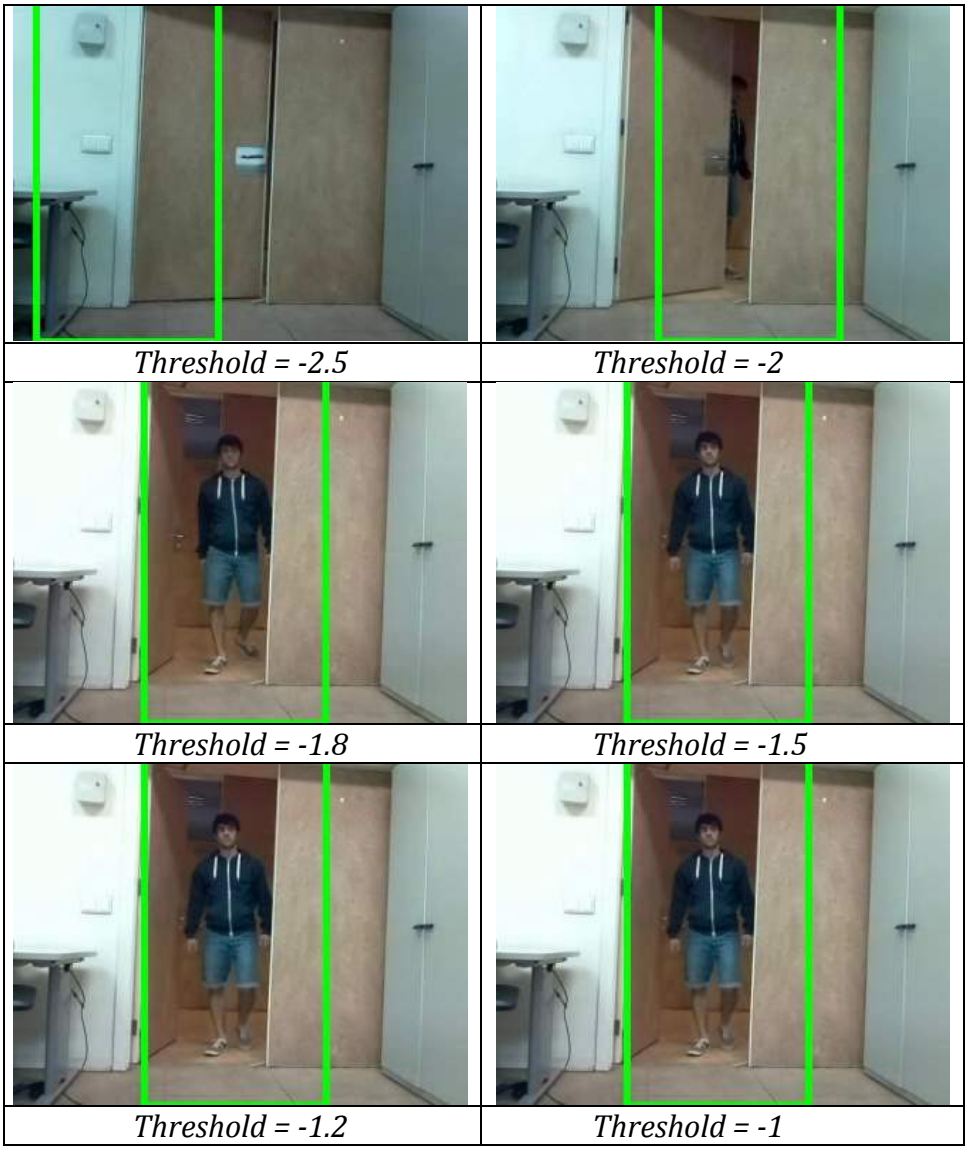


Figura 66: Resultados obtenidos en el sistema implementado en función del Threshold (seguridad sobre la detección) que escogido para el método.

Nótese que con un *Threshold* inferior a -1, no se obtienen detecciones de personas. Analizando las secuencias de vídeo utilizadas para la elaboración del proyecto, se obtienen los datos especificados en la Tabla 4:

Threshold	*[-1.2 , -1.6]*
Agrupación de rectángulos	*3*
Escalado de bloques	*1.05*
Tamaño de celdas (píxeles)	*(8,6)*

Tabla 4: Parámetros utilizados para el algoritmo HOG

5.4.4 Seguimiento de personas

Una vez se ha detectado una persona en la imagen, es necesario poder seguirla por el entorno. De este modo, el ordenador de procesamiento de imagen es capaz de enviar las posiciones a donde deben moverse los servo-motores del sistema, a la *Raspberry PI.*

Para poder seguir a la persona a través del entorno de vigilancia, es necesario modelar correctamente el objeto a seguir, siendo en este caso una persona. Para la realización de este proyecto se han decidido evaluar dos técnicas:

- *Template Matching*
- *Mean Shift*

5.4.4.1 Template Matching

Template Matching es una de las técnicas más utilizadas en sistemas de reconocimiento basados en visión por computador. El objetivo de ésta técnica es el de encontrar regiones similares en una imagen, dada una pequeña plantilla de la última.

La idea general del algoritmo, es la de buscar regiones similares a la plantilla dada dentro de la imagen de búsqueda, y ponderar las detecciones hechas, de tal modo que exista un parámetro de confianza a la hora de evaluar la búsqueda realizada.

Como ya se ha explicado en el Capítulo 5.4.2.1.1, una imagen se compone de una matriz de píxeles, donde cada uno de estos píxeles tiene una intensidad (si trabajamos con imágenes en escala de grises), o tres intensidades (si trabajamos con imágenes en tres canales (*RGB*)). *Template Matching* trabaja con

imágenes en escala de grises, por lo tanto cada píxel está definido por una intensidad entre 0 y 255.

De forma intuitiva se puede llegar a apreciar el hecho de que para encontrar una imagen dentro de otra, los píxeles de la plantilla utilizada deben ser iguales o similares, a los de la imagen de búsqueda dentro de una región determinada de la última. Este es el principio sobre el que trabajan la mayoría de algoritmos destinados al seguimiento de objetos en una secuencia de imágenes.

Para determinar el grado de similitud entre las dos imágenes se utiliza un parámetro representativo, este se conoce como coeficiente de correlación de *Pearson*. En este caso, el coeficiente de correlación trata de dar una medida de lo similares que son dos imágenes en función de la intensidad de sus píxeles.

Si consideramos una imagen de referencia *r* de tamaño *p x q* píxeles, y una plantilla más pequeña *t* de tamaño *m x n* píxeles, el coeficiente de correlación entre la plantilla *t* y la imagen de referencia *r* se define como (nótese que *i* y *j* se refieren a la posición de uno de los píxeles de *r*):

$$\rho_{t,i,j} = \frac{covarianza(t,r)}{\sigma_t \sigma_r} \quad [2.4.1]$$

$$\rho_{t,i,j} = \frac{\Sigma_{x=1}^{m}\Sigma_{y=1}^{n}(t(x,y) - \mu_t)(r_{i,j}(x,y) - \mu_{i,j})}{\sqrt{\Sigma_{x=1}^{m}\Sigma_{y=1}^{n}(t(x,y) - \mu_t)^2}\sqrt{\Sigma_{x=1}^{m}\Sigma_{y=1}^{n}(r_{i,j}(x,y) - \mu_{i,j})}} \quad [2.4.2]$$

- $\rho_{t,i,j}$: matriz de correlación entre *t* y *r*.
- μ_t: media de las intensidades de la plantilla.
- $\mu_{i,j}$: media de la imagen referencia evaluada entre *i,j.*

Una vez estimada la correlación entre la plantilla dada y la imagen de referencia es posible definir la región donde la correlación entre las dos imágenes es mayor.

En el Gráfico 11 se muestra la correlación entre una plantilla (obtenida en la etapa de detección de personas) y una imagen de referencia (Figura 67):

Figura 67: Plantilla e imagen de referencia.

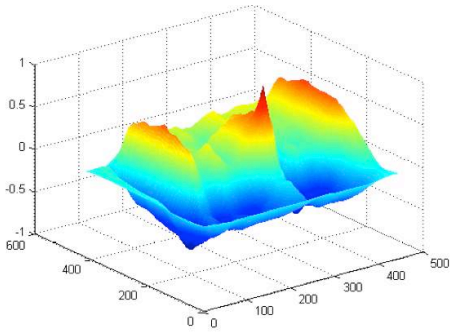


Gráfico 11: Correlación de Pearson [2.4.2] entre la plantilla y la imagen de referencia mostradas en la figura de la izquierda.

Como se puede apreciar en la Figura 67 y el Gráfico 11, la correlación es fuerte en los puntos donde coinciden las dos imágenes.

5.4.4.1.1 Inconvenientes de *Template Matching*

El principal problema de *Template Matching* reside en la acumulación de correlaciones referidas al fondo de la imagen, y no al objeto que se está siguiendo.

En el caso estudiado de seguimiento de personas, los objetos de seguimiento no presentan formas invariantes en el tiempo, sino que presentan rotaciones y deformaciones del cuerpo a la hora de desplazarse por el entorno. En la Figura 68 se presenta el problema de la deformación del objeto de seguimiento cuando este se desplaza:

Figura 68: Demostración visual de como las personas deforman su cuerpo a la hora de desplazarse.

El movimiento de un ser humano, inevitablemente le lleva a deformar su cuerpo con el objetivo de poder desplazarse. Es sencillo apreciar, que si el objeto de seguimiento comienza a deformarse, la correlación entre la plantilla seguida y la imagen de referencia, presentará máximos no en el objeto seguido, sino en el

fondo de la imagen. Por ello, una solución podría estar relacionada con el hecho de fusionar la substracción del fondo de la imagen, con la plantilla seguida.

De cualquier modo, en el sistema de vigilancia desarrollado, esta no era una opción viable, dado que una vez se detecta una persona en el entorno, la cámara empezará a rotar con el fin de seguir el objeto, variando continuamente el fondo de la imagen.

En las Figuras 69, 70, 71 y 72, se muestra el problema descrito con *Template Matching*. Nótese como a medida que el objeto se deforma, la correlación entre las dos imágenes estudiadas presenta sus máximos en las zonas donde el fondo está presente:

Figura 69: Plantilla e imagen de referencia.

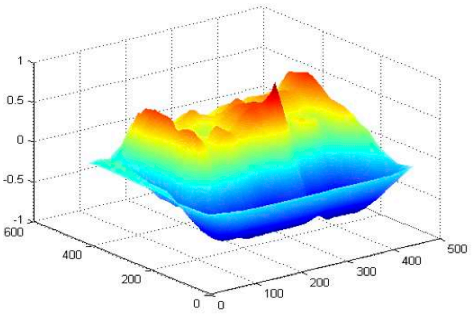


Gráfico 12: Correlación de Pearson [2.4.2].

Figura 70: Plantilla e imagen de referencia.

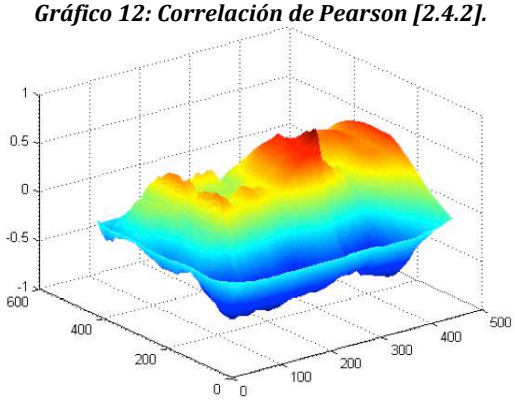


Gráfico 13: Correlación de Pearson [2.4.2].

Figura 71: Plantilla e imagen de referencia.

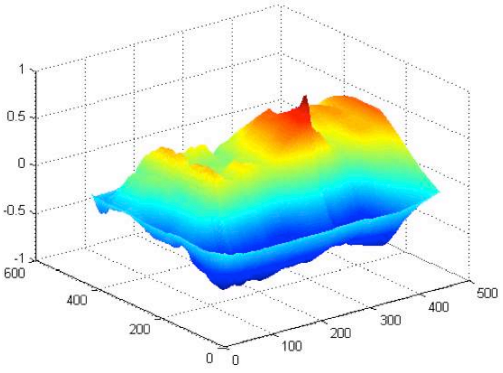


Gráfico 14: Correlación de Pearson [2.4.2].

Figura 72: Plantilla e imagen de referencia.

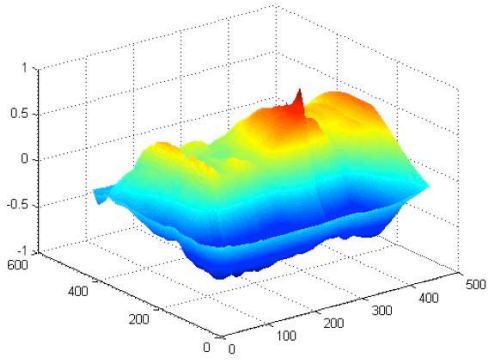


Gráfico 15: Correlación de Pearson [2.4.2].

Observando los Gráficos 12, 13, 14 y 15, se llega a la conclusión de que a medida que el objeto se deforma, la correlación entre las dos imágenes presenta sus máximos en las zonas donde el fondo está presente en la plantilla.

Por lo tanto, se puede afirmar, que *Template Matching* no es un algoritmo óptimo para el seguimiento de personas. Una opción más conveniente podría ser fusionar los resultados de la substracción de fondo y *Template Matching,* consiguiendo así disminuir el peso del fondo de la imagen en la correlación entre la plantilla y la imagen de referencia, no obstante, para el sistema implementado donde el objeto es seguido de forma dinámica a través de la rotación de la cámara, esta no es una opción viable.

5.4.4.2 *Mean Shift*

Mean Shift es un algoritmo iterativo no paramétrico que se puede utilizar para muchos propósitos como encontrar modos, *clustering*, etc. Fue introducido por *Fukunaga y Hostetler* (9), y se ha extendido a otros campos como la Visión por Computador (10).

Mean Shift considera el espacio de características de una imagen como una función de densidad de probabilidad empírica. Si la entrada al algoritmo es un conjunto de puntos, *Mean Shift* los considera como una muestra de la función de densidad de probabilidad. Si existen regiones densas (*clusters*) en el espacio de características, estas regiones densas (*clusters*) se corresponderán con el modo (o máximos locales) de la función de densidad de probabilidad.

Para cada punto dado al algoritmo, *Mean Shift* define una ventana alrededor del punto y calcula la media de los puntos localizados en la ventana. Una vez calculada la media de los puntos localizados en la ventana, se desplaza el centro de la ventana hacia la media de la ventana, y se repite el algoritmo hasta que la media de la ventana y el centro de la ventana convergen. La idea básica es mover el centro de la ventana de búsqueda a regiones más densas de puntos.

Para estimar la función de densidad de la imagen *Mean Shift* estima un *Kernel* sobre las características de la imagen. Un *Kernel* en una espacio de d-dimensiones R^d que debe satisfacer [2.5.1] y [2.5.2]:

$$\int_{R^d} \phi(x) = 1 \quad [2.5.1]$$

$$\phi(x) \geq 0 \quad [2.5.2]$$

Existen varios tipos de *Kernel* que podemos aplicar al espacio de características de una imagen, no obstante, el más común es el *Gaussiano*:

$$\phi(x) = e^{\frac{-x^2}{2\sigma^2}} \quad [2.5.3]$$

La estimación de un *Kernel* es un modelo no paramétrico utilizado para estimar la función de densidad de una variable aleatoria. Esto por lo general, se conoce como *Parzen Windows*. Dados n puntos, $x_i = 1 \dots n$, con una función de densidad dada por $K(x)$ un radio para las ventanas de *Parzen h*:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{\bar{x} - x_i}{h}) \quad [2.5.4]$$

Con una distribución de la probabilidad dada por:

$$P(x) = \frac{1}{n} \sum_{i=1}^{n} K(x - x_i) \quad [2.5.5]$$

Una vez estimado el *Kernel* del espacio de características de la imagen, se cumple que en el centro de masas de la función de densidad: $\nabla \hat{f}(x) = 0$. El gradiente de la función de densidad viene dado por:

$$\nabla \hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K'(\frac{\bar{x} - x_i}{h}) \quad [2.5.6]$$

$$\sum_{i=1}^{n} K'(\frac{\bar{x} - x_i}{h})\,\vec{x} = \sum_{i=1}^{n} K'(\frac{\bar{x} - x_i}{h})\,\overrightarrow{x_i} \quad [2.5.7]$$

$$\vec{x} = \frac{\sum_{i=1}^{n} K'(\frac{\bar{x} - x_i}{h})\overrightarrow{x_i}}{\sum_{i=1}^{n} K'(\frac{\bar{x} - x_i}{h})} \quad [2.5.8]$$

Como se explicó anteriormente, *Mean Shift* trata los puntos del espacio de características como una función de densidad de probabilidad. Las regiones densas en el espacio de características corresponden a máximos o modos locales. De este modo, para cada punto de los datos observados, se obtiene el gradiente de hacia los máximos locales de la función de densidad. Una vez obtenido el gradiente, se mueve la ventana en función de:

$$m(x) = \frac{\sum_{i=1}^{n} -K'(\frac{\bar{x} - x_i}{h})x_i}{\sum_{i=1}^{n} -K'(\frac{\bar{x} - x_i}{h})} - x \quad [2.5.8]$$

M(x) denota la cantidad de movimiento conocida como *Mean Shift.* Este movimiento muestra la diferencia entre la media de la función de densidad calculada en la ventana de búsqueda y el centro de la ventana. Dado que se ha calculado el gradiente de la función de densidad en [2.5.6], este vector siempre apuntará en la dirección de máximo incremento en la función de densidad. Después de una serie de iteraciones, el centro de la ventana siempre acaba convergiendo de tal forma que $\boldsymbol{m(x) \simeq 0}$.

5.4.4.2.1 *CAMShift*

CAMShift (*Continously Adaptive Mean Shift*) (10) es una modificación de *Mean Shift* con la que se puede obtener la variación del tamaño del objeto seguido. Ya se ha comentado como *Mean Shift* define una ventana de búsqueda y a través de iteraciones (movimientos de la ventana de búsqueda) busca encontrar convergencia entre la media de los puntos de la ventana, y el centro de la ventana. Los gastos computacionales del algoritmo pueden ser reducidos a través del cálculo de los puntos centrales (x_c, y_c) del *backProjection* (Capítulo 5.4.4.2.2) utilizando:

$$x_c = \frac{\sum_x \sum_y xP(x,y)}{\sum_x \sum_y P(x,y)} \quad ; \quad y_c = \frac{\sum_x \sum_y yP(x,y)}{\sum_x \sum_y P(x,y)} \qquad [2.5.9] \qquad [2.5.10]$$

$$P(x,y) = h(I(x,y)) \quad [2.5.11]$$

Donde $h(I(x,y))$ denota la distribución de probabilidad del *backProjection* utilizado, con un histograma de colores *h*, y una ventana donde se busca la distribución de probabilidad dada por *h.*

Las ecuaciones superiores [2.5.9] y [2.5.10] pueden ser simplificadas por:

$$x_c = \frac{M_{10}}{M_{00}} \qquad\qquad y_c = \frac{M_{01}}{M_{00}} \quad [2.5.12] \qquad [2.5.13]$$

Donde M_{00} representa el momento cero de la imagen, y tanto M_{10} como M_{01} representan los momentos de primer orden de la imagen. Los momentos de una imagen son una media ponderada de las intensidades de una imagen, muy útiles para encontrar los centroides de una imagen. Una vez calculados los momentos de la imagen se actualiza el tamaño de la ventana en base a [2.5.14] hasta que el algoritmo encuentra convergencia.

$$s = 2 \cdot \sqrt{\frac{M_{00}}{256}} \qquad [2.5.14]$$

5.4.4.2.2 Elección del espacio de características

La gran ventaja que aportan tanto *MeanShift* como *CAMShift*, es la elección del espacio de características. Para el proyecto realizado se ha considerado el color de la camiseta de la persona seguida, como la característica más representativa de una persona.

El motivo de utilizar la camiseta como el color más representativo de una persona, también está relacionado con el hecho de que el torso es generalmente la parte del cuerpo que menos se deforma cuando una persona se está moviendo. Obsérvese como en una secuencia de vídeo donde la persona se está moviendo a una gran velocidad, el torso parece ser la parte del cuerpo más invariante (Figura 73).

Figura 73: Demostración visual de como la parte del cuerpo que menos se deforma durante el desplazamiento de una persona es el torso.

Para detectar la región de la camiseta de una persona, se utiliza una plantilla centrada en la región donde el algoritmo *HOG* (Capítulo 5.4.3) ha encontrado una persona. La idea de utilizar este tipo de plantillas se obtiene de (11) el autor del trabajo mencionado utiliza plantillas más complejas, no obstante, en el Capítulo 5.4.4.2.3 se comentará una aproximación diferente al uso de plantillas complejas, basada en los resultados obtenidos desde la substracción de fondo.

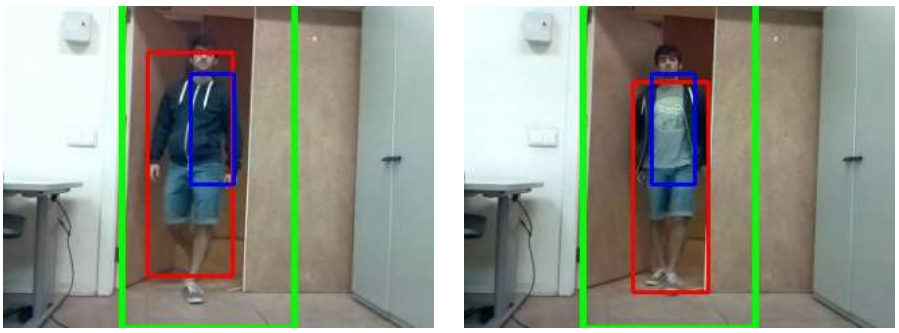


Figura 74: Detección de una persona en el entorno de seguridad. El rectángulo verde viene dado por el algoritmo HOG. El rectángulo rojo viene dado por los píxeles que han sido marcados como foreground. El rectángulo azul es la plantilla donde se espera encontrar el color de la camiseta del intruso.

Una vez detectada la región de la camiseta podemos calcular un histograma de colores de la región encontrada. La gran ventaja que aporta el centrar nuestro seguimiento en el color de la camiseta se basa en que, incluso cuando las camisetas presentan dibujos, el color de las camisetas suele ser bastante uniforme (Figura 74).

Para estimar el histograma de colores de interés se convierte la imagen al espacio de color HSV (*Hue, Saturation, Value*). El espacio de color HSV es menos dependiente a los cambios de iluminación y ruido que el modelo RGB. En el modelo HSV el matiz se representa por una región circular; una región triangular separada, puede ser usada para representar la saturación y el valor del color (Figura 75).

Figura 75: En la figura de la izquierda se muestra una imagen en el espacio de color HSV. En la figura de la derecha se muestra una imagen en el espacio de color RGB.

La imagen es convertida al espacio de color HSV con el fin de poder obtener una imagen en escala de grises que represente el matiz y la saturación de cada uno de los píxeles de la imagen (Figuras 76 y 77).

Figura 76: Máscara de obtenida a partir de la substracción de fondo.

Figura 77: Imagen que representa el matiz de los colores de la Figura 76 en escala de grises.

Una vez obtenida la imagen en escala de grises (*Hue*) con el matiz y la saturación de los píxeles de interés se puede formar un histograma con las ocurrencias de esos colores en la imagen (Gráfico 16).

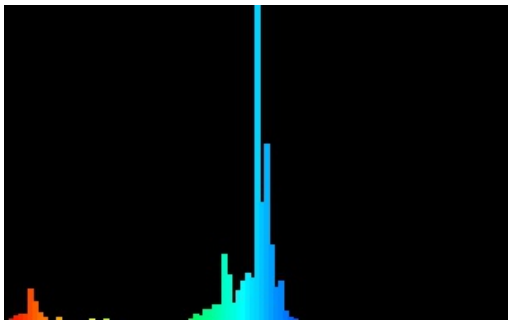


Gráfico 16: Histograma de colores.

Finalmente, una vez hallado el histograma de colores, se obtiene el *backprojection* (retroproyección) de la imagen completa. El *backprojection* de una imagen viene a representar la relación entre un píxel de la imagen y el histograma obtenido para representar el objeto de seguimiento. Por ejemplo, si el color de un píxel es amarillo, y el peso del color amarillo en el histograma es del 20%, se multiplicará 255*0.2 (en el caso de que representemos cada píxel

con 8 bits). El *backprojection* de la imagen, será el espacio de características utilizado por *CAMShift* (Figura 78 y 79).

Figura 78: En la figura de la derecha se muestra el backprojection de la imagen de la izquierda.

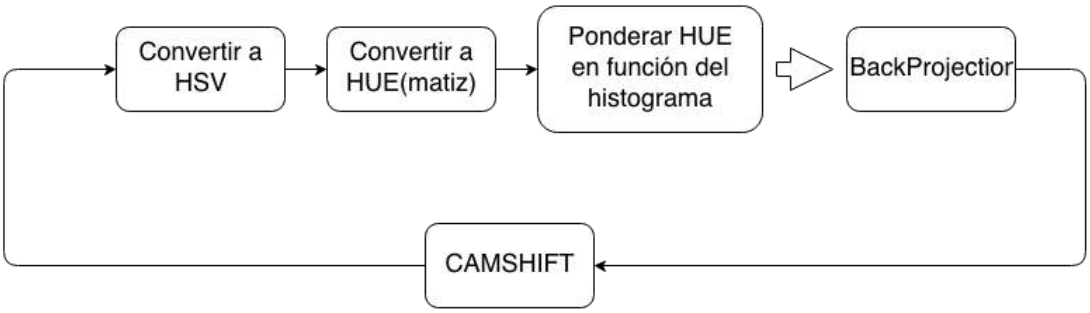


Figura 79: Diagrama de flujo simplificado del proceso de seguimiento.

5.4.4.2.3 Incorporación de la substracción de fondo a la detección del espacio de características

A la hora de escoger los colores característicos del intruso que ha entrado en el entorno de seguridad, es necesario discriminar qué colores forman parte del fondo, y cuáles forman parte de la camiseta del intruso. En la Figura 80 se muestra un histograma de colores (Gráfico 17), en el que los colores del fondo también presentan relevancia en el modelo generado del color de la camiseta.

Figura 80: Detección de una persona en el entorno antes de lo previsto.

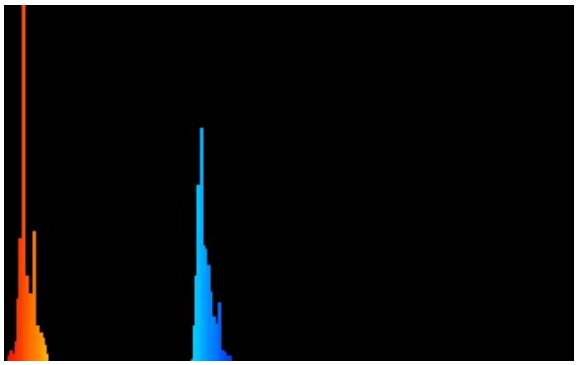


Gráfico 17: Histograma de colores donde el fondo de la imagen ha cobrado excesiva relevancia, obtenido a partir de la Figura 79.

En este caso, el algoritmo de detección de personas (Capítulo 5.4.3) ha detectado una persona antes de entrar completamente en la habitación, el resultado es un histograma de colores en el que también aparecen los colores (anaranjados) de la puerta.

Se propone un fusión entre la etapa de substracción de fondo del sistema (Capítulo 5.4.1), y una plantilla centrada en la región obtenida desde el algoritmo de detección de personas, con el objetivo de disminuir la relevancia del fondo en el algoritmo de seguimiento. Es sencillo notar que los colores que deseamos seguir deben estar presentes en los píxeles marcados como *foreground.*

Figura 81: Obtención de una máscara a partir de la substracción de fondo.

En la Figura 81 se muestra como los colores que deseamos seguir, se encuentran en los píxeles marcados como primer plano. No obstante, los píxeles de la substracción de fondo presentados en la figura superior pueden llegar a presentar una información insuficiente del color a seguir. Es sencillo apreciar cómo en la figura superior los píxeles marcados como primer plano están dispersos en la imagen. Con el fin de mejorar esta máscara obtenida, se emplea una operación de morfológica conocida como Cierre [2.5.17]. La operación morfológica Cierre incluye una Dilatación[2.5.15] de la imagen y una Erosión [2.5.16**]** posterior de la misma imagen. Nótese que en [2.5.15] ,[2.5.16] y [2.5.17] A denota una imagen binarizada, y B denota el elemento estructural con el que se desea llevar a cabo la operación (un círculo, un rectángulo, …).

$$(A \oplus B) \quad [2.5.15]$$

$$(A \ominus B) \quad [2.5.16]$$

$$A \cdot B = (A \oplus B) \ominus B \quad [2.5.17]$$

En [E.2] se da una explicación más precisa de estas operaciones morfológicas. En las Figuras 82, 83 y 84, se da una demostración la operación morfológica Cierre:

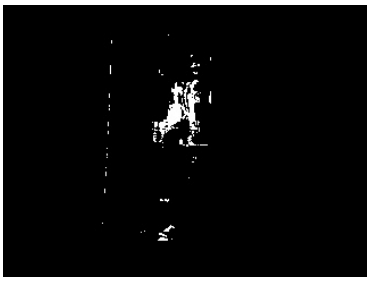


Figura 82: Máscara del foreground.

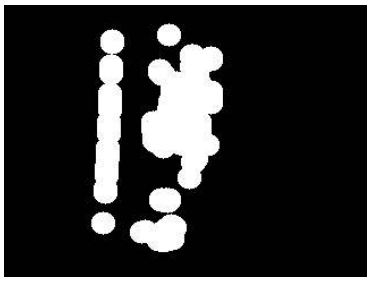


Figura 83: Dilatación de la Figura 81.

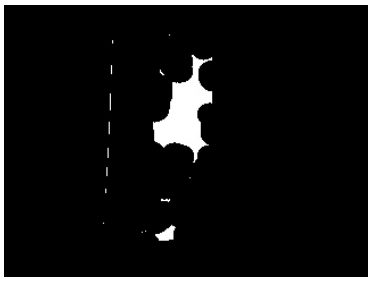


Figura 84: Erosión de la Figura 82.

Una vez mejorada la máscara binaria obtenida de la etapa de substracción de fondo del sistema, se obtiene el histograma de colores entre la plantilla centrada en la detección de personas y la máscara. Nótese como los colores del fondo apenas están presentes en el nuevo histograma de colores. (Gráfico 18 A y B)

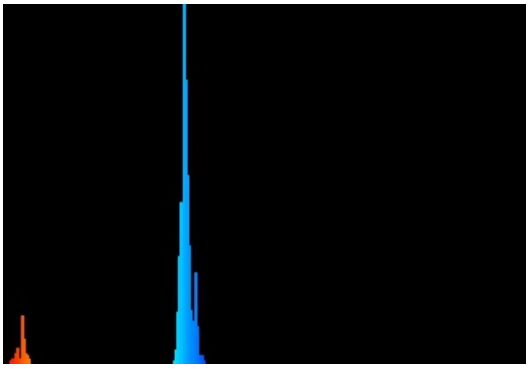


Gráfico 18 A: Histograma de colores obtenido de la Figura 80 utilizando la substracción de fondo.

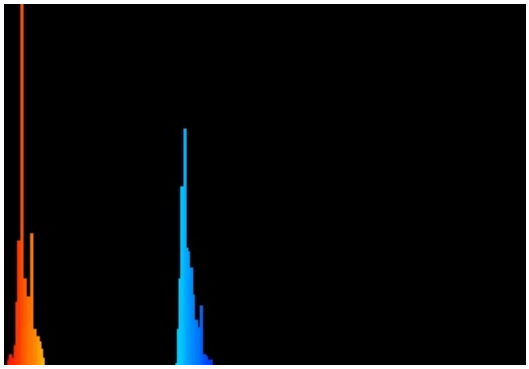


Gráfico 18 B: Histograma de colores obtenido de la Figura 80 sin utilizar la substracción de fondo.

5.4.4.2.4 Elección de la región del espacio de color HSV

Como ya se ha comentado en el Capítulo 5.4.4.2.2 el algoritmo *CAMShift,* utiliza una imagen en escala de grises donde a cada uno de los píxeles del espacio *HSV* se les asigna una intensidad (0-255) en función del espacio HSV. Con el fin de mejorar los resultados obtenidos por *CAMShift* se debe escoger una región de confianza dentro del modelo *HSV.* En la Figura 85 se muestra el modelo de color *HSV:*

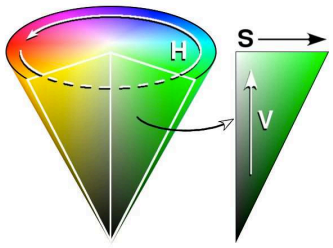


Figura 85: Espacio de color HSV

Para la implementación del algoritmo de seguimiento se decidió utilizar un valor (*Hue*) Matiz entre (0°,360°), es sencillo notar el hecho de que si se desea obtener toda la gama de colores posibles, el valor del matiz debe incluir la circunferencia completa del cono mostrado en la figura superior.

Los valores de (*Value*) Valor están localizados entre (10,255). Se escoge un valor mínimo con de 10 con el fin de eliminar los valores oscuros del modelo.

La elección de una región de confianza para la *Saturation, o* saturación, determinará, en gran medida, los resultados obtenidos por el algoritmo *CAMShift.* Nótese en la Figura 84 como los valores extremos de la saturación están muy diferenciados entre sí, por lo tanto, es necesario evaluar qué saturación mínima es necesaria para definir correctamente nuestro objeto de seguimiento. En la gráfica inferior se muestra el error absoluto cometido por el algoritmo *CAMShift* respecto a la posición del objeto en función de la Saturación mínima escogida:

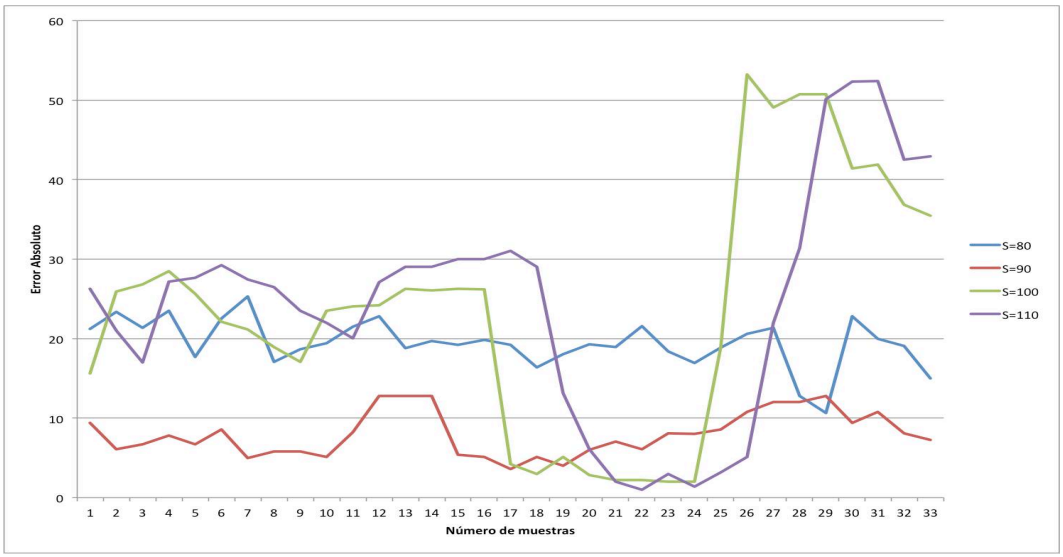


Gráfico 19: Error absoluto cometido por el algoritmo CAMShift en función de la saturación mínima del espacio de color HSV.

Tomando el Gráfico 19 como referencia se decide utilizar los siguientes valores (Tabla 5) para el modelado de los objetos de seguimiento en el espacio de color HSV:

Saturación	[90 , 256]
Matiz	[0° , 360°]
Valor	[10 , 256]

Tabla 5: Región del espacio de color HSV utilizada para el algoritmo.

5.4.4.2.2 Elección del tamaño del histograma de colores utilizado

La elección del tamaño del histograma de colores que representa el objeto a seguir determinará en gran medida los resultados obtenidos por el algoritmo *CAMShift.* Un histograma demasiado grande, contendrá una gran variedad de colores, aumentando la probabilidad de incluir en el histograma los colores del fondo de la imagen. Mientras que un histograma de colores demasiado pequeño reducirá la respuesta ante rotaciones y deformaciones del objeto seguido.

A continuación se muestra la imagen *backprojection* obtenida en función del tamaño del histograma:

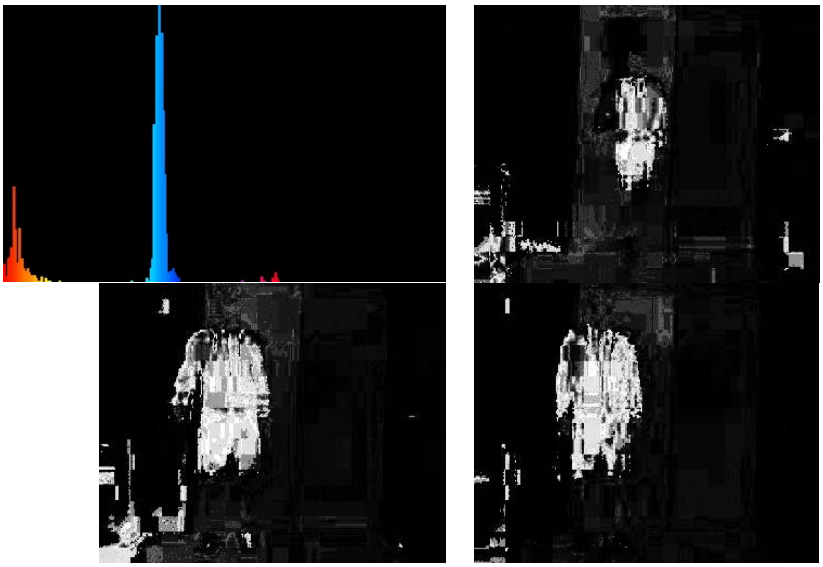


Gráfico 20: Histograma de colores utilizando 200 columnas y ejemplos de imágenes backprojection obtenidas.

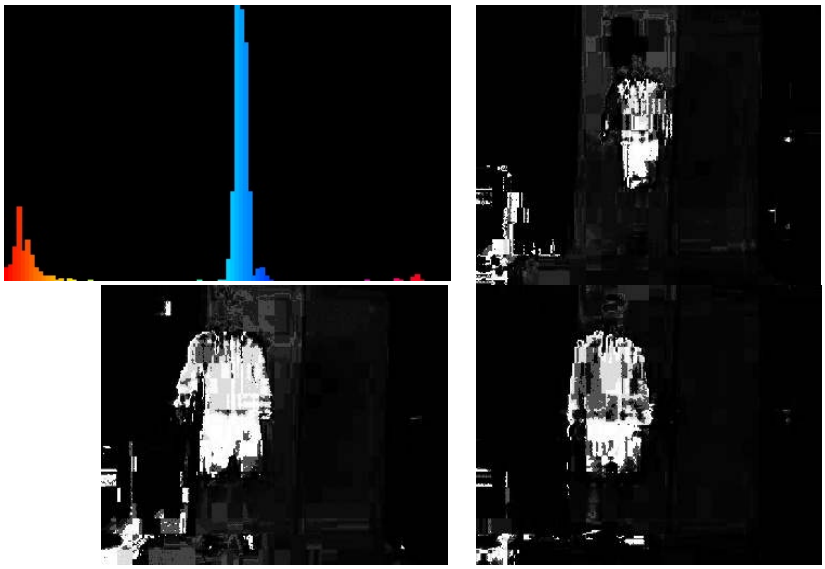


Gráfico 21: Histograma de colores utilizando 100 columnas y ejemplos de imágenes backprojection obtenidas.

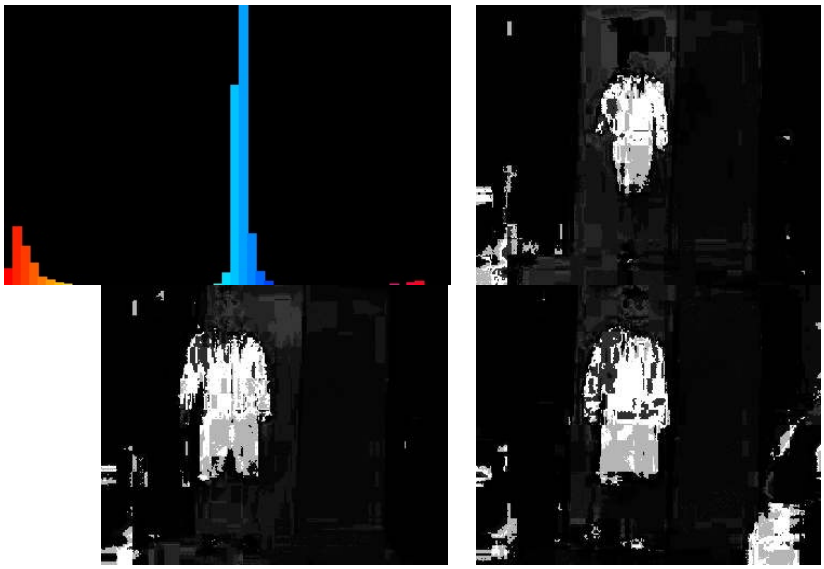


Gráfico 22: Histograma de colores utilizando 50 columnas y ejemplos de las imágenes backprojection obtenidas.

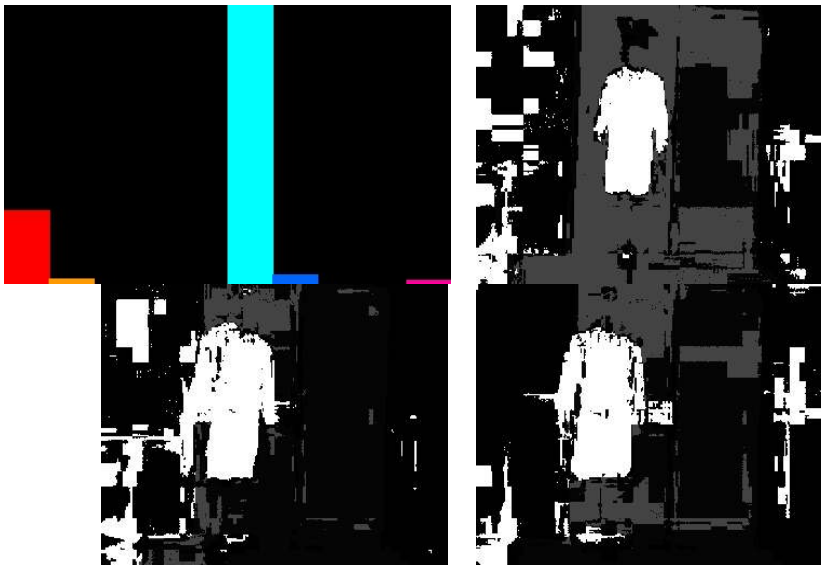


Gráfico 23: Histograma de colores utilizando 10 columnas y ejemplos de las imágenes backprojection obtenidas.

Nótese que a medida que reducimos el tamaño del histograma a partir del cual obtendremos nuestro *backprojection* las imágenes presentan máximos de intensidad más esparcidos por el entorno. (Gráficos 20, 21, 22 y 23). Teniendo en cuenta los resultados presentados, se decidió utilizar un tamaño de 25 columnas para el histograma de colores del objeto a seguir.

5.4.4.2.5 Predicción del estado del objeto a seguir

En la sección anterior (Gráficos 20, 21, 22 y 23), se mostraron varios ejemplos del espacio de características utilizado por *CAMShift.* Es sencillo notar que el fondo de la imagen puede cobrar importancia para el algoritmo *CAMShift,* dado que el color utilizado para modelar el objeto de seguimiento, puede aparecer en el fondo de la imagen, conduciendo los resultados del seguimiento a posicionamientos erróneos del algoritmo. Este comportamiento erróneo del algoritmo, se puede agravar en situaciones donde existen oclusiones del objeto que estamos siguiendo:

Figura 86: Backprojection de una imagen donde aparece una oclusión al objeto de seguimiento que presenta unos colores similares a los del objeto seguido.

Nótese que el objeto que se interponiendo con nuestro objeto de seguimiento, presenta un histograma de colores similar al objeto seguido. (Figura 86).

Con el fin de eliminar ruido del ambiente, suavizar las posiciones del objeto a seguir y hacer el algoritmo más robusto ante oclusiones, se decidió implementar un filtro de *Kalman.*

El filtro de *Kalman* es un algoritmo desarrollado por *Rudolf E. Kalman* en 1960, que sirve para poder identificar el estado oculto (no medible) de un sistema dinámico lineal. El filtro de *Kalman* es un método matemático que obtiene de forma recursiva un estimador lineal del estado de un proceso en un instante *t*, en base a sus observaciones en los instantes 1,...,*t*, y un error en la estimación asociado. Se dice que es recursivo, ya que el filtro se retroalimenta de su salida en el paso anterior para la realización de la nueva estimación.

En esta representación, el sistema es descrito por un conjunto de variables denominadas de estado. El estado contiene toda la información relativa al sistema en un cierto punto en el tiempo. Esta información debe permitir la inferencia del comportamiento pasado del sistema, con el objetivo de predecir su comportamiento futuro.

El proceso a ser estimado por el filtro de *Kalman* tiene como objetivo resolver el problema de estimar el estado $X \in Rn$, de un proceso controlado en

tiempo discreto, el cual es dominado por una ecuación lineal en diferencia estocástica de la siguiente forma:

$$X_t = AX_{t-1} + w_{t-1} \quad [2.5.18]$$

X_t: Estado en el momento t.
A: Matriz de transición del estado *t-1* al estado *t*
X_{t-1}: Estado en el momento t-1
w_{t-1}: Error del proceso en el momento t-1.

Con una observación Z:

$$Z_t = HX_t + v_t \quad [2.5.19]$$

Z_t: Medidas en el momento *t*.
H: Matriz de que relaciona el estado con la medición llamada matriz de observación.
X_t: Estado en el momento *t*.
v_t: Error de la medida.

Las variables w_{t-1} y v_t representan el error del proceso y de la medida respectivamente, se asume que son independientes entre ellas, que son ruido blanco y que siguen una distribución normal:

$$p(w) = N(0, Q) \quad [2.5.20]$$

$$p(v) = N(0, R) \quad [2.5.21]$$

Q: Matriz de covarianza de perturbación del proceso.
R: Matriz de covarianza de perturbación de la medida.

Desde el punto de vista de las ecuaciones matemáticas utilizadas para la implementación del filtro, se pueden dividir en dos grupos (Figura 87):

- Estimación del estado
- Corrección del estado

Las del primer grupo son responsables de la estimación del estado *t* en base al estado *t-1,* y de la actualización intermedia de la matriz de covarianza del error tomando en cuenta únicamente las *t-1* observaciones. Por otro lado, las ecuaciones de la fase de actualización, son las encargadas de actualizar las

estimaciones teniendo en cuenta la nueva información dada por la observación en el instante *t,* minimizando el error de la observación en el instante *t* de forma estadística.

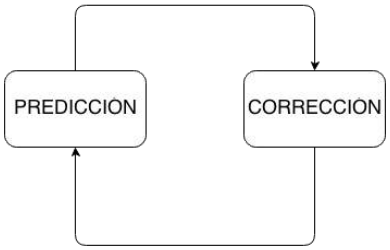


Figura 87: Diagrama de flujo del filtro de Kalman.

Predicción del estado:

$$\hat{X}_t^- = A\hat{X}_{t-1} \quad [2.5.22]$$

$\hat{X}_t^-$: Estimación del estado en el momento t.
A: Matriz de transición del estado *t-1* al estado *t.*
$\hat{X}_{t-1}$: Estado en el momento t-1.

$$P_t^- = AP_{t-1}A^T + Q \quad [2.5.23]$$

- P_t^- : Pronóstico de la covarianza del error en el momento *t*.
- A: Matriz de transición.
- P_{t-1}: Covarianza del error actualizada en el momento t-1.
- Q: Matriz de covarianza de la perturbación del proceso.

Corrección del estado

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \quad [2.5.24]$$

- K_t: Ganancia de Kalman en el momento t.
- P_t^-: Pronóstico de la covarianza del error en el momento *t*.
- H: Matriz de que relaciona el estado con la medición llamada matriz de observación.
- R: Matriz de covarianza de perturbación de la medida.

$$\hat{X}_t = \hat{X}_t^- + K_t(Z_t - H\hat{X}_t^-) \quad [2.5.25]$$

- $\hat{X}_t$: Actualización de la estimación del estado en base a los datos observados y al error de la estimación inicial.
- K_t: Ganancia del filtro.
- H: Matriz de que relaciona el estado con la medición llamada matriz de observación.
- Z_t : Observaciones en el momento t.

$$P_t = (I - K_t H)\, P_t^- \quad [2.5.26]$$

- P_t: Actualización de la covarianza en el momento *t.*
- I: Matriz identidad.
- K_t: Ganancia del filtro.
- P_t^-: Pronóstico de la covarianza del error en el momento *t*.
- H: Matriz de que relaciona el estado con la medición llamada matriz de observación.

5.4.4.2.5.1 Fusión entre el filtro de Kalman y CAMShift

A continuación, se describen los parámetros obtenidos, con el fin de implementar un filtro de Kalman.

Si suponemos que el objeto está enmarcado en un rectángulo con un centro localizado en [x, y], se denota un vector de estado para el rectángulo dado por:

$$X_t = \begin{bmatrix} x_t \\ y_t \\ \frac{x_t - x_{t-1}}{dt} \\ \frac{y_t - y_{t-1}}{dt} \end{bmatrix} \quad [2.5.27]$$

Con una matriz de transición:

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [2.5.28]$$

donde dt es igual 0.01 segundos, el tiempo medio que tarda el algoritmo en completar un bucle completo.

Las observaciones Z_t del proceso serán igual a:

$$Z_t = Hx_t + v_t \quad [2.5.29]$$

donde la matriz de observaciones H es igual a:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [2.5.30]$$

y la variable aleatoria v_t , que representa el ruido inherente al proceso de medición, viene dada por la covarianza de la matriz $R : p(v_t) = N(0, R_t)$, donde R representa la matriz de perturbación de la medida. Básicamente, esta matriz representa la varianza de nuestras observaciones del proceso:

$$R = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 \\ 0 & \sigma_y^2 & 0 & 0 \\ 0 & 0 & \sigma_{vx}^2 & 0 \\ 0 & 0 & 0 & \sigma_{vy}^2 \end{bmatrix} \quad [2.5.31]$$

Para obtener la varianza de cada una de las observaciones, utilizaremos la siguiente figura donde se muestra el error absoluto cometido por el algoritmo *CAMShift (Gráfico 24* :

Gráfico 24: Error absoluto cometido por el algoritmo CAMShift.

Utilizando los errores absolutos cometidos por nuestras observaciones, que en este caso vienen dadas por el algoritmo *CAMShift*, se pueden obtener las varianzas (Tabla 6) de cada una de las observaciones del sistema implementado:

σ_x^2	21,19
σ_y^2	13,85
σ_{vx}^2	15563,48
σ_{vy}^2	8945,20

Tabla 6: Varianzas de las observaciones dadas al filtro de Kalman.

En teoría la matriz de perturbación del proceso podría variar en función del estado del proceso, no obstante, para la realización de este proyecto se ha considerado *R* como una matriz diagonal constante:

$$R = \begin{bmatrix} p & 0 & 0 & 0 \\ 0 & p & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 0 & 0 & p \end{bmatrix} \quad [2.5.31]$$

Una vez definida la matriz de perturbación de la medida, se obtiene la variable p de perturbación del proceso de forma experimental. El Gráfico 25 muestra el error del filtro en la etapa de predicción, en función del parámetro p mostrado en la matriz superior:

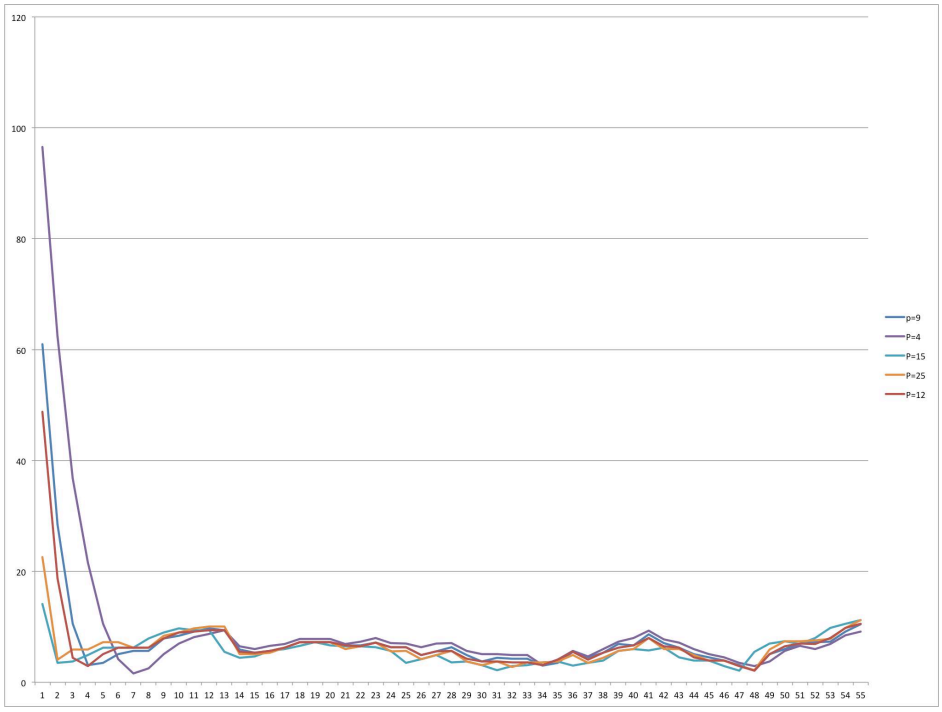


Gráfico 25: Error cometido por el filtro de Kalman respecto a la posición real del objeto en función de p en la fase de predicción.

Nótese como el filtro con una matriz R con *p=15* , tiene un error menor en la etapa de predicción, así como un ajuste más rápido a la posición inicial. En el Gráfico 26 se muestra el error absoluto cometido por el filtro con [2.5.32] y [2.5.33] como matrices de covarianza.

$$R = \begin{bmatrix} 21{,}19 & & & \\ & 21{,}85 & & \\ & & 15563{,}48 & \\ & & & 8945{,}20 \end{bmatrix} \quad [2.5.32]$$

$$R = \begin{bmatrix} 15 & & & \\ & 15 & & \\ & & 15 & \\ & & & 15 \end{bmatrix} \quad [2.5.33]$$

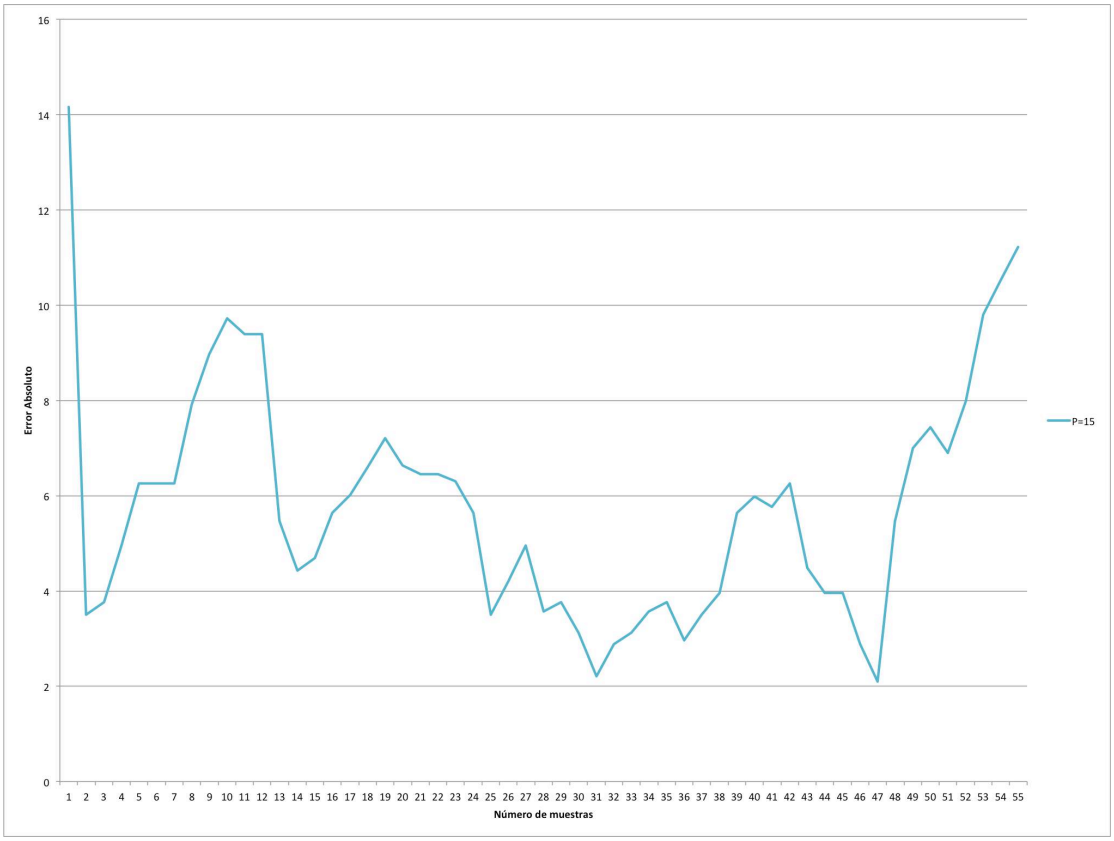


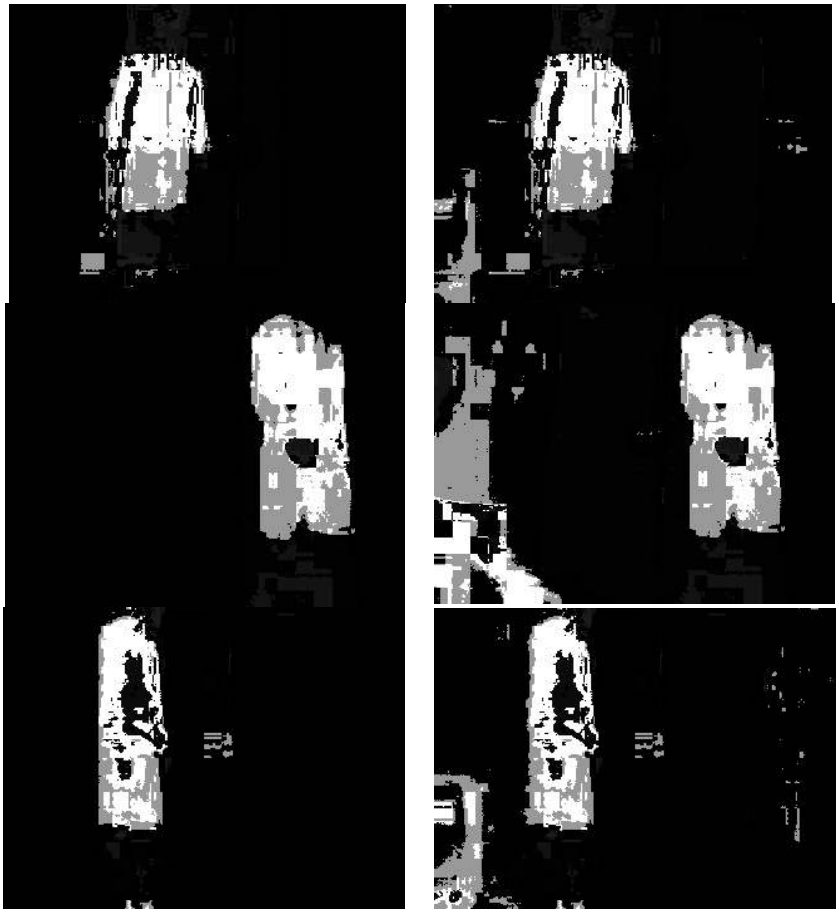
Gráfico 26: Error del filtro de Kalman utilizado respecto a la posición real del objeto en la fase de predicción.

5.4.4.2.5.1 Resultados de la fusión entre el filtro de Kalman y CAMShift

En los Gráficos 20, 21, 22 y 23 se puede apreciar como incluso eligiendo correctamente el tamaño de el histograma que representa a nuestro objeto en una secuencia de imágenes, los elementos el fondo cobran importancia en nuestra imagen *backprojection.* Con el fin de reducir el ruido ambiental y limitar las zonas de búsqueda del algoritmo *CAMShift,* se propone la utilización de un filtro de *Kalman,* con el fin de mejorar los resultados del algoritmo *CAMShift*.

La mejora significativa que podemos conseguir con la utilización de un filtro de *Kalman*, se basa en la capacidad de predecir las zonas donde se debe encontrar nuestro objeto de seguimiento, eliminando las zonas menos probables de contener nuestro objeto de seguimiento.

Para conseguir este objetivo, se propone la utilización de la última región rectangular, encontrada por el algoritmo *CAMShift* (Figura 88). Esta región de búsqueda se deslizará por la imagen de tal forma que todo lo que no se encuentre dentro de ella, dejará de tener relevancia para el algoritmo *CAMShift.* Nótese como con la técnica aplicada el ruido del fondo de la imagen se reduce considerablemente.



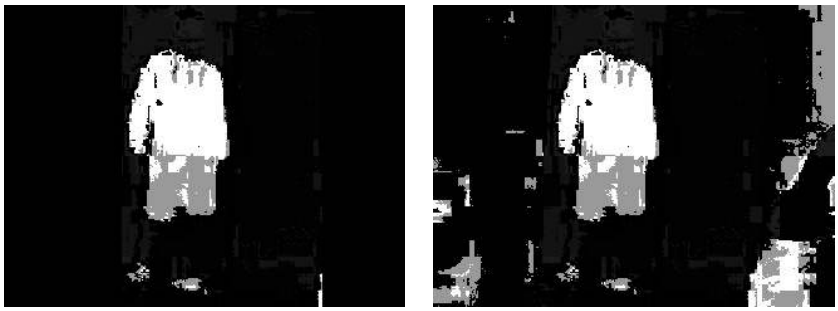


Figura 88: Reducción del ruido introducido al algoritmo CAMShift utilizando un filtro de Kalman.

5.4.5 Control de la rotación de la cámara

Ya se ha comentado en el Capítulo 5.1 que el sistema desarrollado cuenta con dos servo-motores con el fin de poder seguir personas a través del entorno. Para poder seguir a las personas que entran en el entorno de seguridad, es necesario conocer el ángulo de rotación necesario para alcanzar el objetivo.

Generalmente, el seguimiento de objetos a través de rotaciones de la cámara, es más preciso con el uso de cámaras estéreo (permiten conocer la distancia al objeto). No obstante, para el proyecto realizado, se ha utilizado una única cámara.

Para poder hacer una estimación del ángulo que debe rotar a cámara, se desarrolla una aproximación lineal entre los píxeles de la imagen y el incremento de tiempo de la señal *ton* (Figura 89), del control PWM de los servo-motores (Capítulo 5.1) en milisegundos. Se obtiene:

$$\Delta 1\ píxel = \Delta 1{,}7\ ms \quad [2.5.34]$$

$$\Delta ton = 1{,}7\ {}^{ms}/_{píxel} \quad [2.5.35]$$

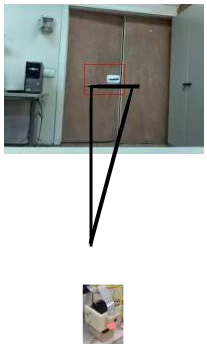


Figura 89: Aproximación lineal entre el ángulo rotado y la diferencia en píxeles respecto al centro de la imagen.

Una vez conocida la relación entre los píxeles de la imagen y la rotación de la cámara, se propone el uso del conocido teorema de Pitágoras (nótese que tanto la ecuación [2.5.35] como el teorema de Pitágoras, suponen una relación lineal que no existe entre los píxeles de la imagen y la rotación de la cámara):

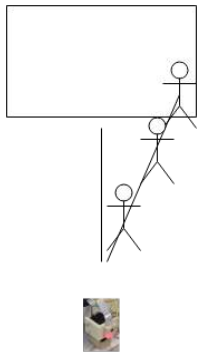


Figura 90: Demostración gráfica del Teorema de Pitágoras.

Tal y como muestra la Figura 90, si hacemos una suposición lineal entre el píxel central de la persona seguida y la distancia de este píxel al píxel central de la imagen, se puede conocer el incremento (o decremento) de tiempo (*ton*), que necesitamos en la señal de control PWM de los servo-motores.

5.4.5.1 Control en función de la posición del objeto seguido

En un primer momento se decidió controlar la rotación de la cámara del sistema únicamente utilizando [2.5.35]. El resultado fue un comportamiento inestable de los servo-motores del sistema, con continuas vibraciones de los mismos, y por consiguiente de la imagen procesada. Las continuas vibraciones de la imagen se traducían en pérdidas del objeto seguido.

Con el fin de controlar la rotación del sistema, se propone la división de la imagen en una serie de cuadrículas (Figura 91). En función de la cuadrícula donde se encuentre el centro de masas del objeto seguido, se pondera el incremento en *milisegundos* del tiempo *ton* del señal PWM de control de los servo-motores.

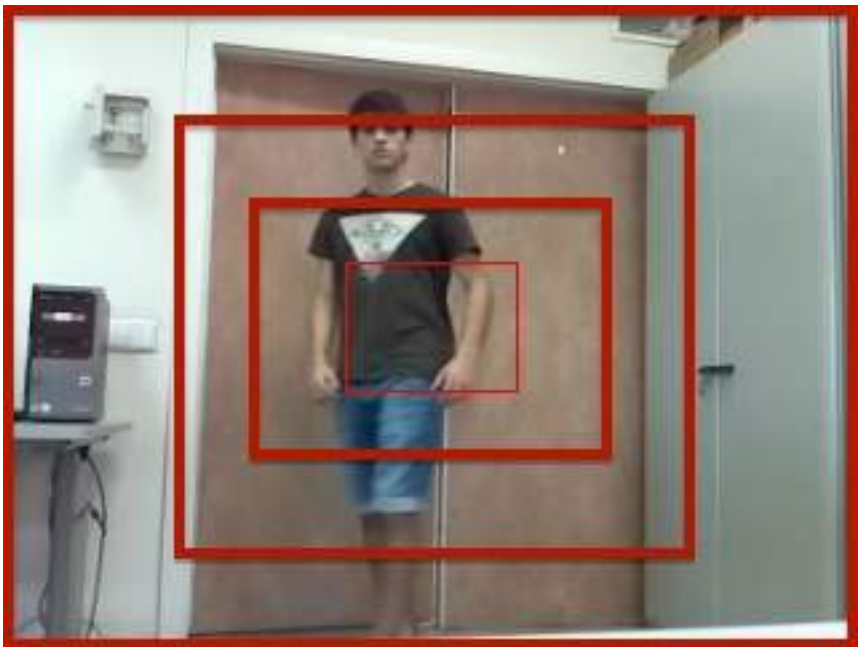


Figura 91: División de la imagen en un sistema de cuadrículas.

Este control en función de la cuadrícula donde se encuentre el centro de masas del objeto, se puede hacer incluso más robusto si obtenemos una ecuación que satisfaga las diferentes ponderaciones de las cuadrículas. Para obtener esta ecuación se utiliza un método de regresión cúbica (Gráfico 27).

Si se toma en consideración que el algoritmo de procesamiento de imagen desarrollado, trabaja con imágenes de 320 columnas por 240 filas, la máxima distancia que podemos encontrar al centro de la imagen es de 160 píxeles en el eje *x*, y 120 píxeles en el eje *y*. Se puede encontrar el incremento de *ton* para cada uno de los servo-motores con [2.5.36] y [2.5.37].

$$\Delta ton_{rotacion\ horizontal} = \begin{cases} x < 160: & -\ f(|160 - x|) \cdot |160 - x| \\ x > 160: & f(|160 - x|) \cdot |160 - x| \end{cases} \quad [2.5.36]$$

$$\Delta ton_{rotación\ vertical} = \begin{cases} y < 120: & -\ f(|120 - y|) \cdot |120 - y| \\ y > 120: & f(|120 - y|) \cdot |120 - y| \end{cases} \quad [2.5.37]$$

donde f es un polinomio de tercer grado de la forma:

$$f(x) = Ax^3 + Bx^2 + Cx + D \quad [2.5.38]$$

donde:

$$A = 7{,}51322736292314 \cdot 10^{-7}$$

$$B = -1{,}06761589170358 \cdot 10^{-4}$$

$$C = 1{,}03543852453928 \cdot 10^{-2}$$

$$D = -1{,}65368462186532 \cdot 10^{-3}$$

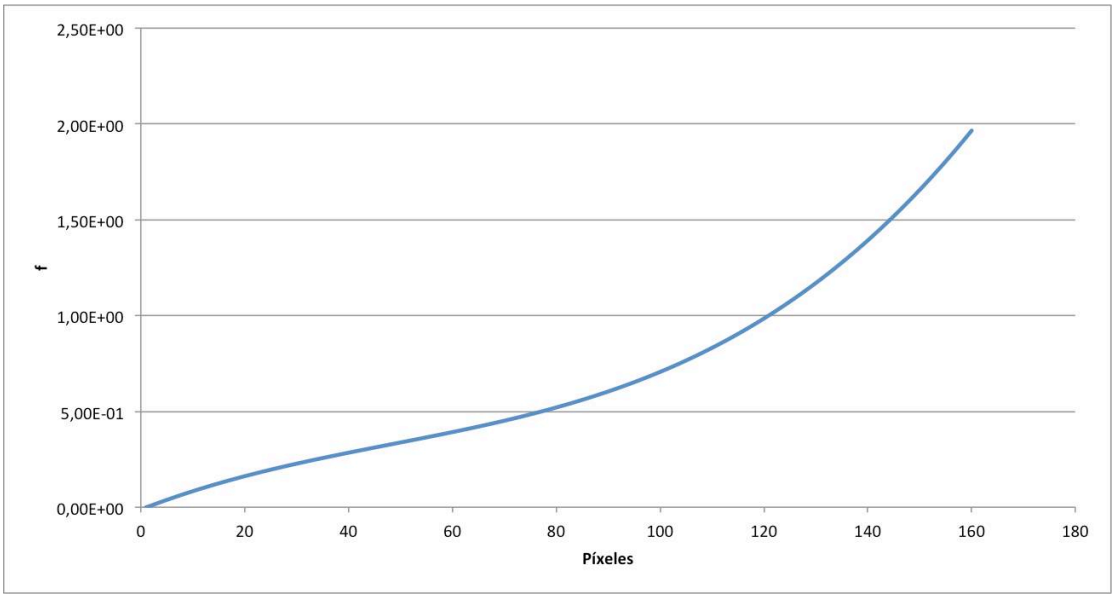


Gráfico 27: Función de control de la rotación de los servo-motores [2.5.38].

5.4.5.2 Implementación del control de la rotación

En el Capítulo 5.2.1.1 se expuso el proceso seguido para implementar un servidor *socket* TCP/IP. Dado que el ordenador donde se ejecuta el algoritmo de procesamiento de imagen comentado en este capítulo, debe estar conectado a la *Raspberry PI* a través de un cable *Ethernet*, es necesario implementar un *socket* cliente con el que se envíen los incrementos requeridos en el tiempo *ton* de la señal PWM.

Este *socket* cliente, Capítulo 4.6.1, se implementa en lenguaje C. Para crear este *socket* cliente:

1. Definimos la dirección IP en la que el servidor *socket* escuchará las peticiones del cliente.

2. Definimos un puerto en el que el *socket* servidor escuchará las peticiones del cliente.

3. El *socket* cliente intenta conectarse con el *socket* servidor.

4. El *socket* cliente envía un mensaje y espera la respuesta del servidor.

En el Gráfico 1 del Capítulo 5.2.6 se muestran los tiempos de comunicación entre la *Raspberry PI* y este *socket* cliente.

5.4.5 Diagrama de flujo del proceso de seguimiento

En la Figura 92 se muestra un diagrama de flujo del proceso de seguimiento de una persona con todas las etapas del algoritmo comentadas en el Capítulo 5.4.

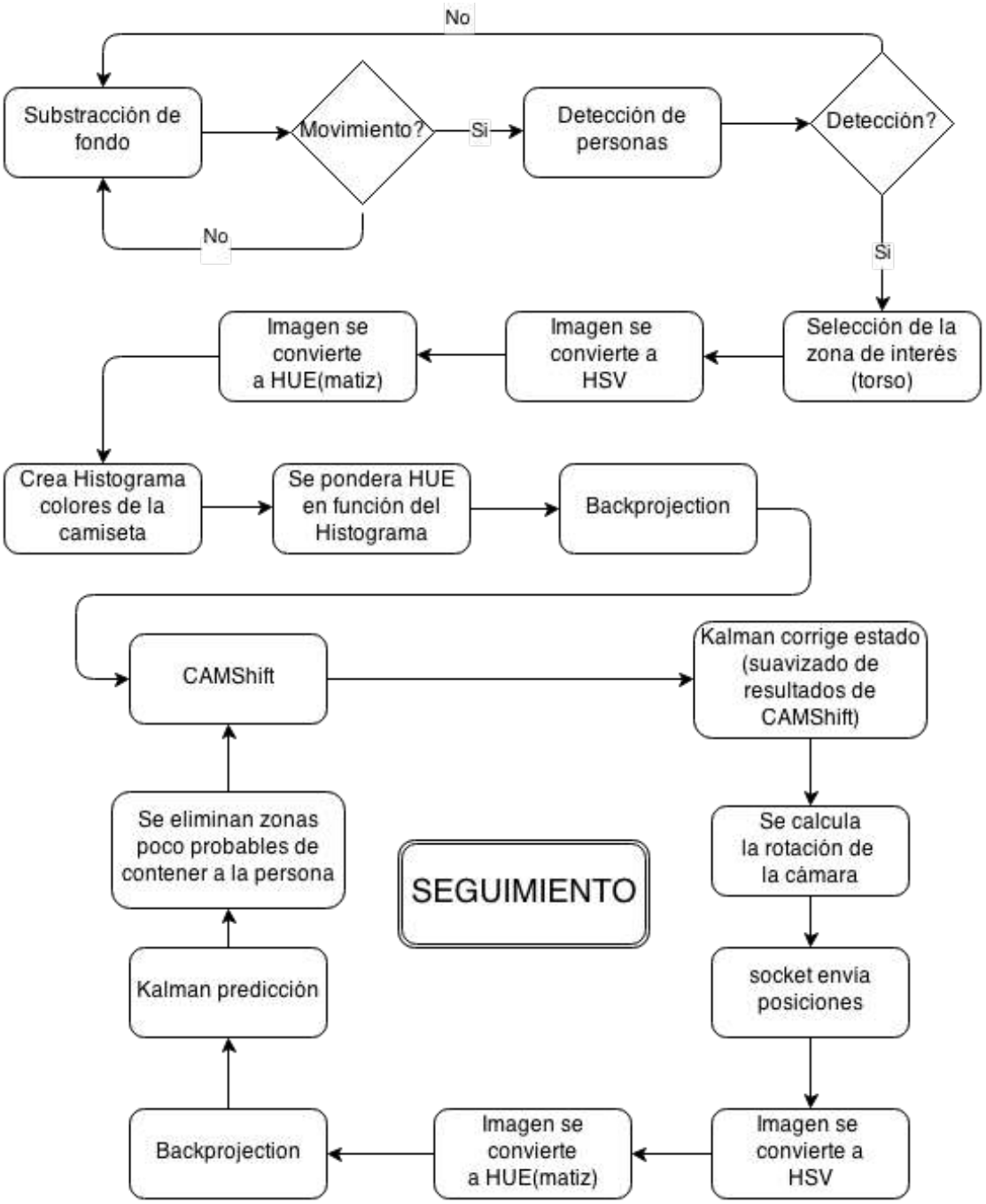


Figura 92: Diagrama de flujo completo del proceso de seguimiento de una persona.

5.5 Sistema propuesto

El sistema propuesto (Figura 93) consta de tres procesos que se llevan a cabo de forma paralela:

- Servidor Web.
- Procesamiento de imagen.
- Control de servo-motores.

Tanto el proceso de procesamiento de imagen como el servidor Web, se comunican a través de *sockets* con el proceso de control de servo-motores. El proceso de control de los servo-motores se mantiene a la escucha de los otros dos procesos paralelos, donde el proceso de procesamiento de imagen tiene prioridad sobre el servidor Web. Es sencillo apreciar que cuando aparece una persona en el entorno de vigilancia, es más importante rotar la cámara para seguir a la persona, que atender a las peticiones del usuario de rotación de servo-motores. Una vez el proceso de procesamiento decide que la persona seguida no se encuentra en el entorno de vigilancia, el usuario recupera el control de los servo-motores a través de la interfaz Web.

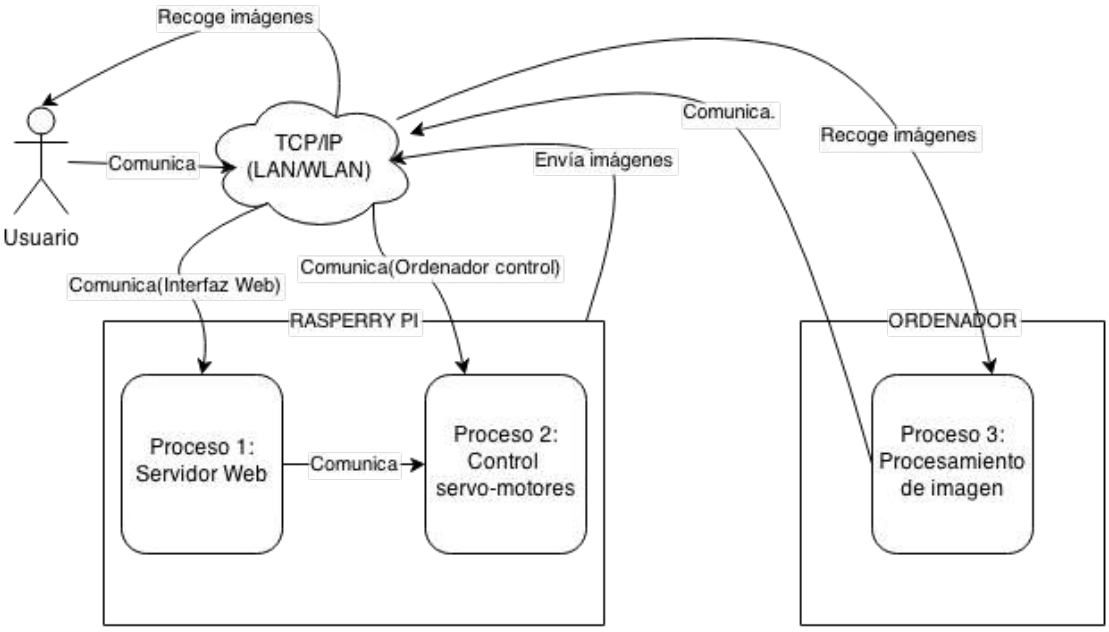


Figura 93: Sistema propuesto.

En las Figuras 94 y 95 se muestra la conexión física entre el *Hardware* del sistema propuesto.

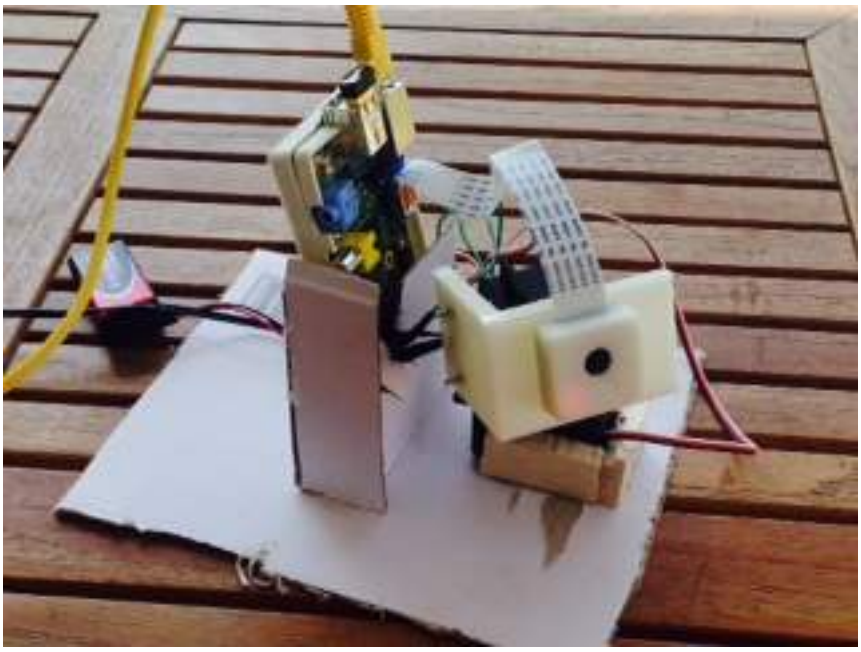


Figura 94: Cámara IP desarrollada.

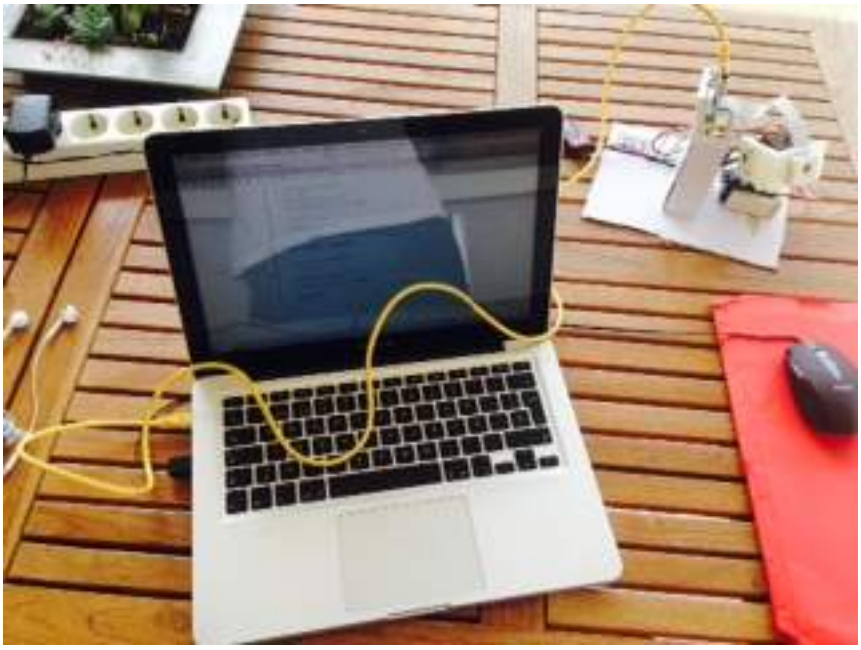


Figura 95: Conexionado entre el ordenador de control y la cámara IP.

6 RESULTADOS

En este capítulo se mostrarán los resultados obtenidos del seguimiento de una persona con el sistema propuesto.

6.1 Resultados en un entorno interior

Para obtener los resultados comentados, se ha utilizado una secuencia de imágenes grabadas en tiempo real por la *Raspberry PI* en un entorno interior con una intensidad de luz media (Figura 96):

Figura 96: Resultados del seguimiento de una persona por un entorno con una intensidad de luz media. El rectángulo verde viene dado por CAMShift. El rectángulo rojo grande viene dado por la región donde aparece el backprojection. El rectángulo rojo central representa el centro de la imagen. El círculo verde representa el estado del filtro de Kalman. El círculo rojo representa la predicción del filtro de Kalman.

En los Gráficos 28, 29 y 30, se muestra el centro de la persona, en cada uno de los ejes, obtenidos desde la secuencia mostrada en la figura superior, nótese que las imágenes obtenidas tienen un tamaño de 320x240 píxeles.

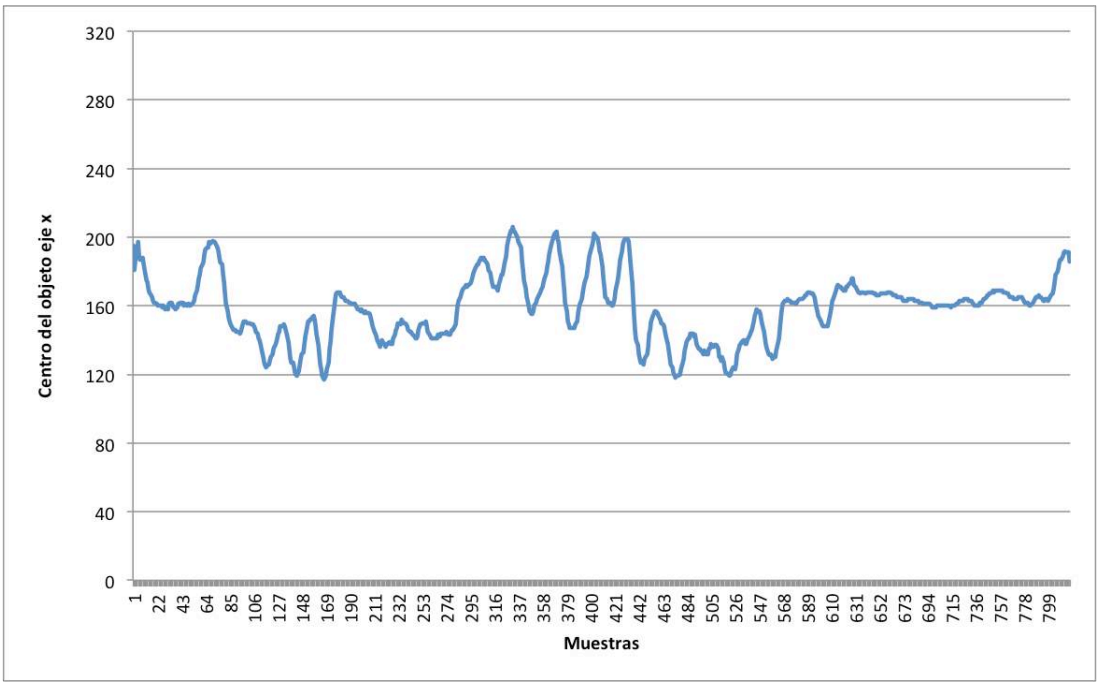


Gráfico 28: Centro de una persona en movimiento [Figura 93] en el eje x (columnas).

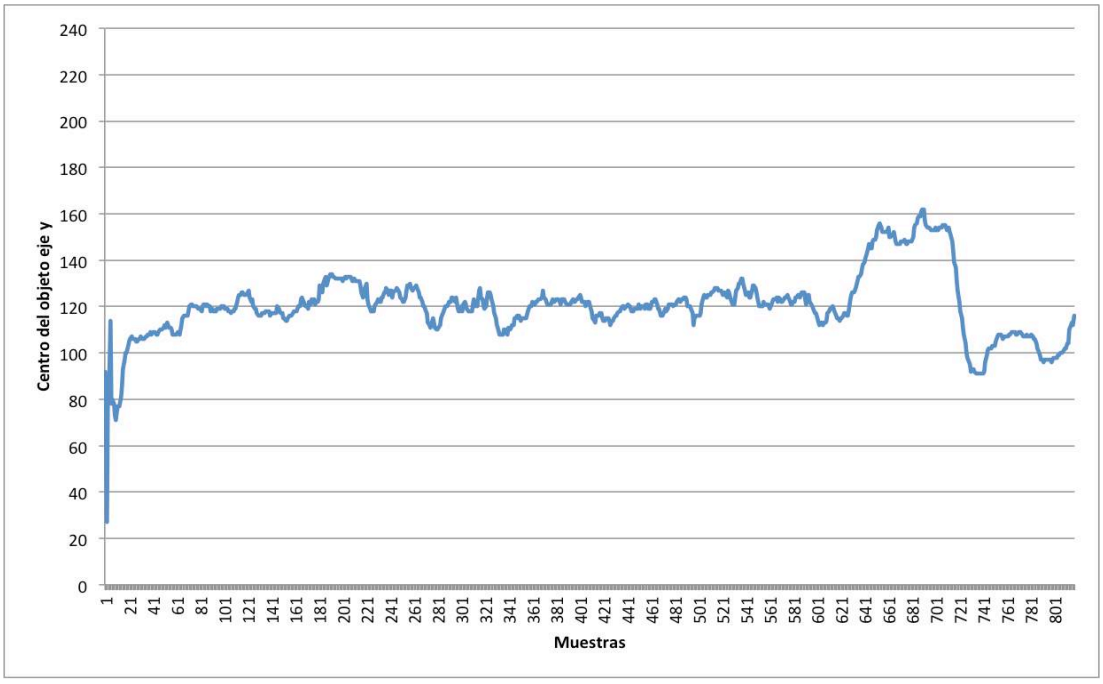


Gráfico 29: Centro de una persona en movimiento [Figura 93] en el eje y (filas).

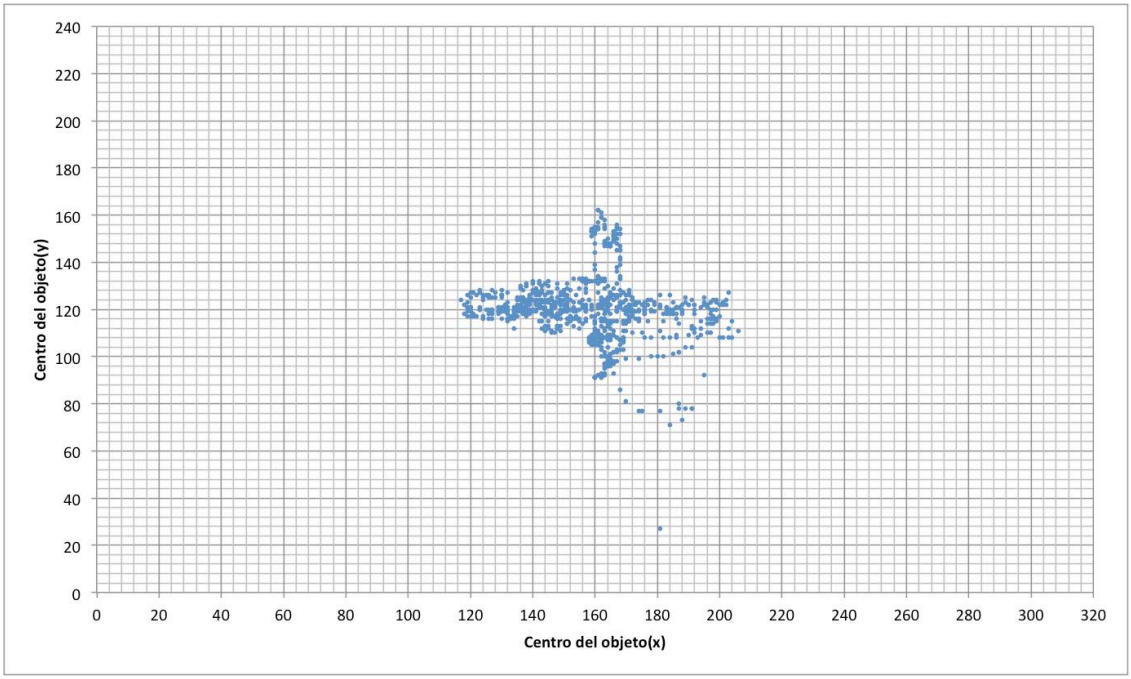
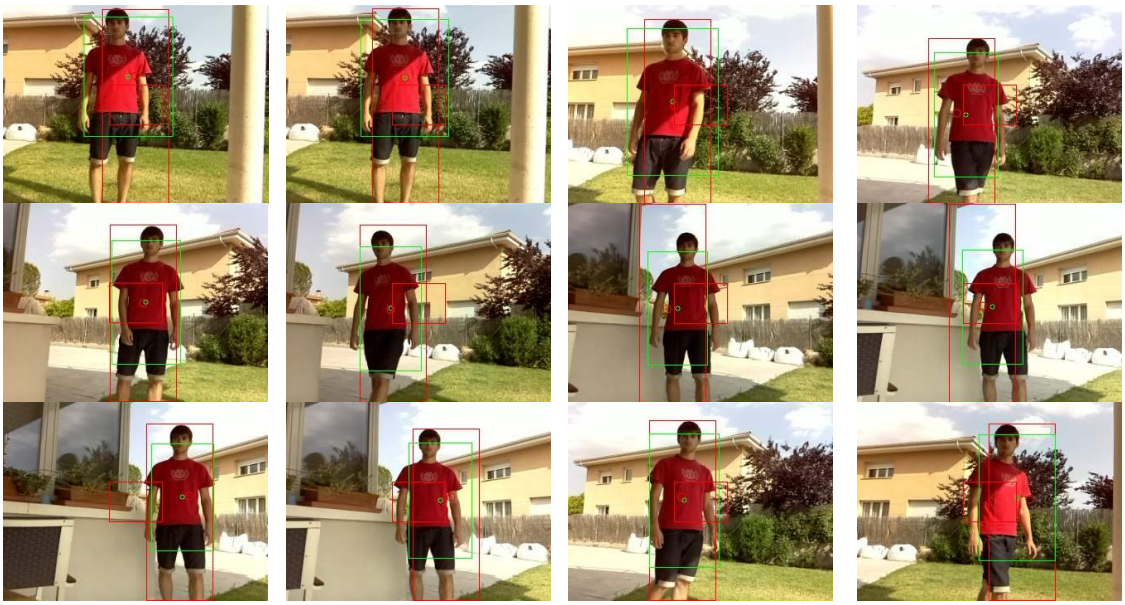


Gráfico 30: Localizaciones de los centros de la persona seguida [Figura 93].

6.2 Resultados en un entorno exterior

Para obtener los resultados en un entorno exterior, se ha utilizado una secuencia de imágenes grabadas en tiempo real por la *Raspberry PI* en un entorno exterior con una intensidad de luz alta (Figura 97).



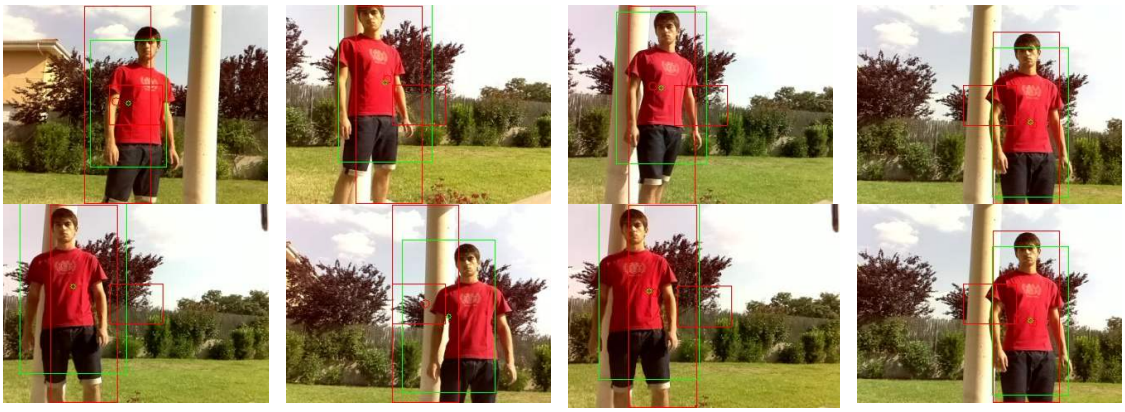


Figura 97: Resultados del seguimiento de una persona por un entorno con una intensidad de luz alta. El rectángulo verde viene dado por CAMShift. El rectángulo rojo grande viene dado por la región donde aparece el backprojection. El rectángulo rojo central representa el centro de la imagen. El círculo verde representa el estado del filtro de Kalman. El círculo rojo representa la predicción del filtro de Kalman.

En los Gráficos 31,32 y 33 se muestra el centro de la persona, en cada uno de los ejes, obtenidos desde la secuencia mostrada en la figura superior, nótese que las imágenes obtenidas tienen un tamaño de 320x240 píxeles.

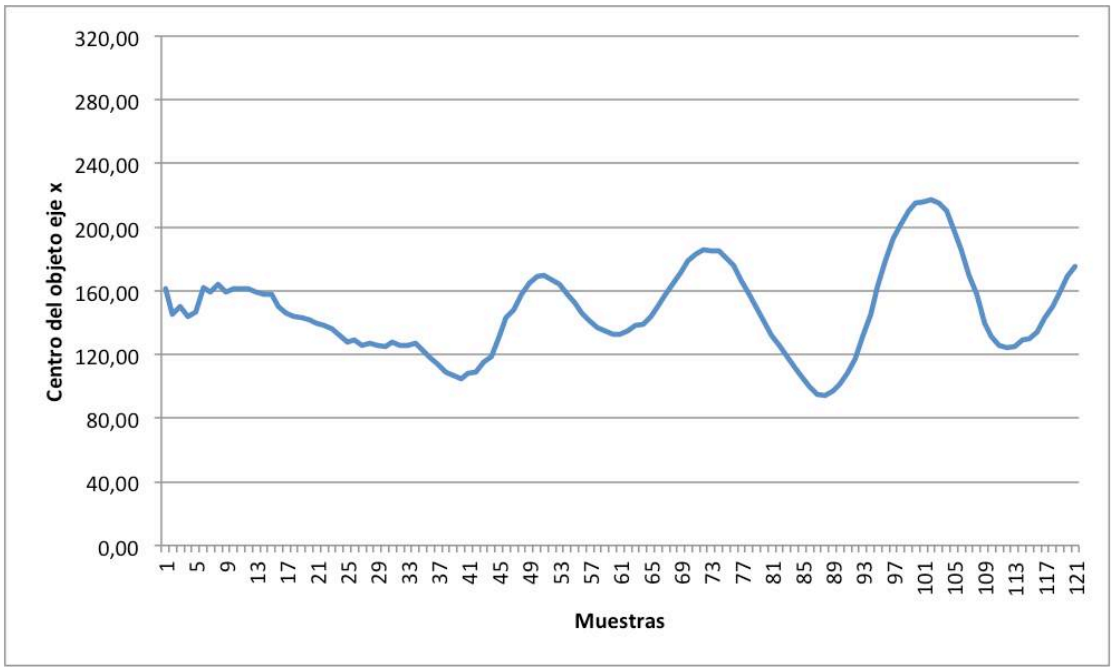


Gráfico 31: Centro de una persona en movimiento [Figura 94] en el eje x (columnas).

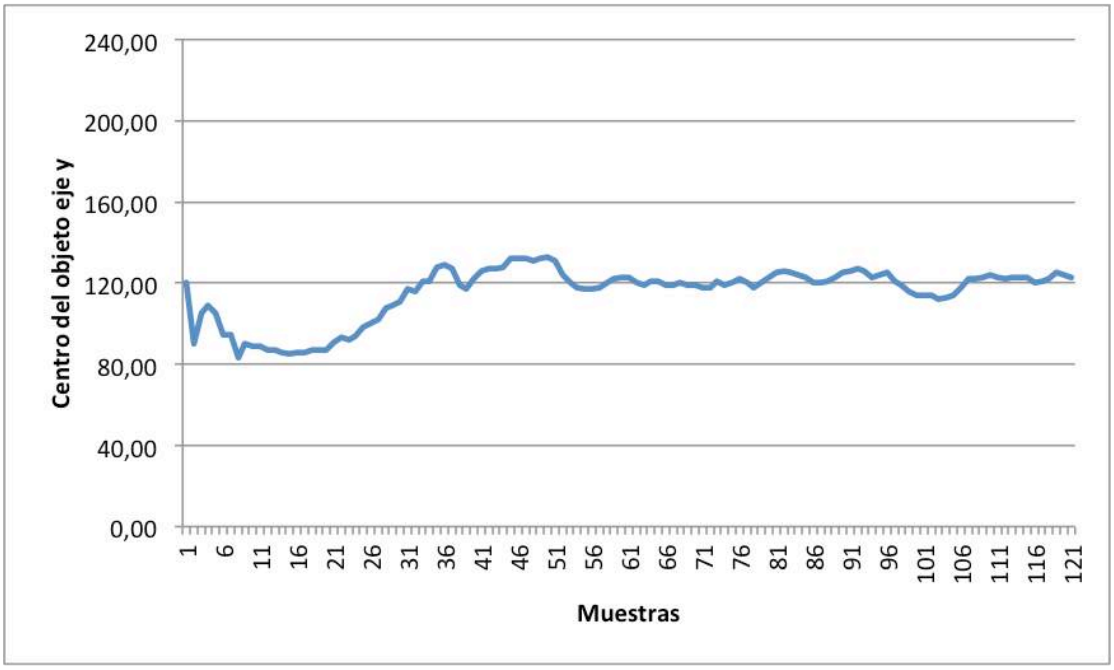


Gráfico 32: Centro de una persona en movimiento [Figura 94] en el eje y (filas).

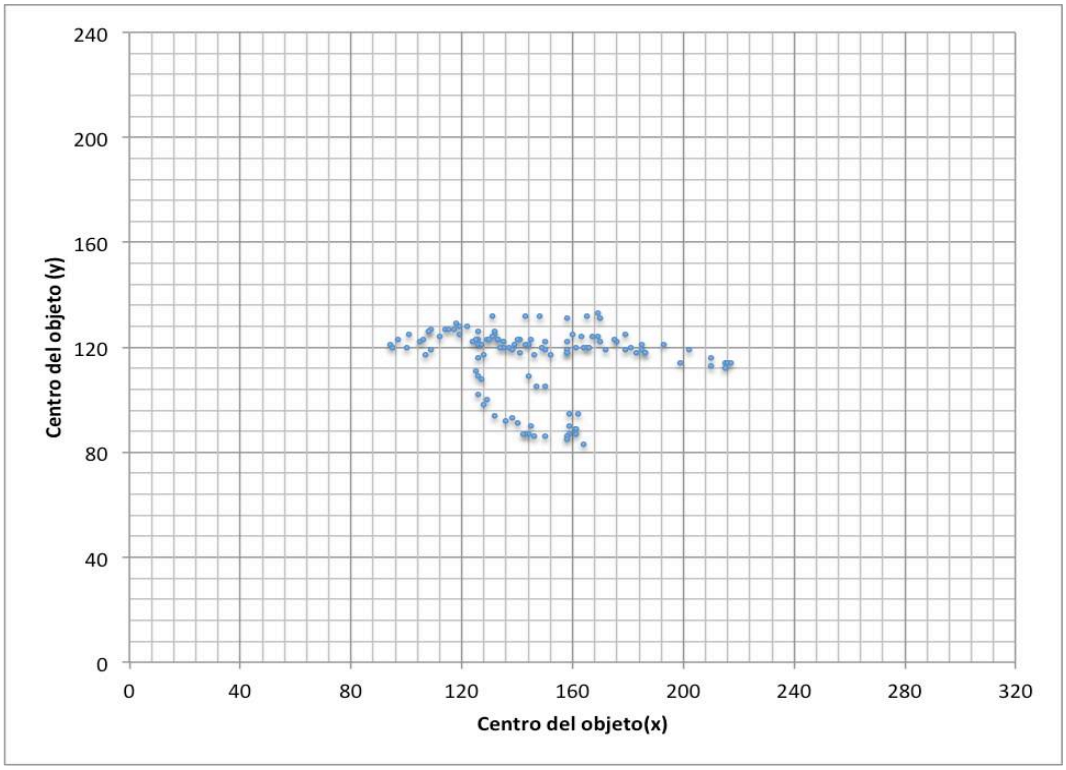


Gráfico 33: : Localizaciones de los centros de la persona seguida [Figura 94].

7 TRABAJOS FUTUROS

El principal inconveniente del algoritmo de procesamiento de imagen, está relacionado con la detección de personas. La detección de personas se lleva a cabo con un método desarrollado por la librería OpenCV, donde se utiliza una base de datos (con positivos y falsos por defecto) implementada por la librería. Por lo tanto, no tiene por qué ajustarse a las condiciones del entorno donde se ha probado el sistema, ni al tamaño de las personas detectadas o ni a la cámara utilizada. Una mejora significativa de este método, estaría relacionada con la implementación de una base de datos propia del entorno donde se espera que funcione el sistema, y con la cámara utilizada para el sistema. El motivo por el cual no se ha implementado esta base de datos radica en el elevado número de imágenes que se deben analizar que, generalmente, oscila entre las 1000 y 1500 imágenes. Además, esta base de datos debe contener un elevado número de sujetos, así como distintas situaciones de detección. Considerando el tiempo que habría llevado la cumplimentación de esta base de datos, así como la disponibilidad de personas dispuestas a emplear su tiempo en el desarrollo de esta base de datos, se decidió utilizar el método por defecto de la librería.

Otra mejora significativa de los resultados estaría relacionada con el uso de cámaras infrarrojas. *Mean Shift* no tiene por qué trabajar con un espacio de características basado en el color de la imagen (nótese que este espacio de características depende enormemente de la iluminación del entorno). Un espacio de características más robusto se basa en el uso de imágenes obtenidas desde una cámara infrarroja (Figura 98). De este modo, los *clusters* que utiliza *Mean Shift,* dependerían del calor desprendido por los objetos de seguimiento. En la figura inferior se muestra una imagen infrarroja:

Figura 98: Imagen infrarroja.

Es sencillo notar como el fondo de la imagen pierde relevancia en la imagen mostrada. Además, el espacio de luz infrarroja, varía mucho menos que un espacio de características basado en el color.

Otro problema significativo del sistema, está relacionado con el reducido grado de confianza que existe sobre la relación entre los píxeles de la imagen y la rotación de los servo-motores. En el Capítulo 5.4.5 se comentó la asunción de linealidad entre los píxeles de la imagen y la rotación de los servo-motores. Esta

relación lineal no es veraz. El modelo de una cámara es mucho más complejo pero, dado que en el sistema propuesto no existe ningún método con el cual se pueda conocer la distancia real al objeto, se decidió asumir un relación lineal entre la imagen y la rotación llevada a cabo. Esta rotación se puede mejorar ampliamente utilizando cámaras estéreo donde se puede conocer la distancia real al objeto. Utilizando cámaras estéreo se puede calibrar un cierto grado de rotación para cada distancia y, a través de una regresión, obtener una ecuación más fiable de la rotación necesaria para cada distancia.

Como medida para mejorar el servidor implementado en la *Raspberry PI*, podría ser una buena idea encriptar los mensajes enviados desde el ordenador de procesamiento de imagen a la *Raspberry PI*v (nótese que cualquier equipo conectado a la misma red local puede recibir los paquetes enviados). Además, sería necesario incrementar la seguridad del servidor Web desarrollado, permitir una única sesión en el servidor al mismo tiempo, y prevenir de posibles errores ante la petición de varios clientes al mismo tiempo.

8 PRESUPUESTO

Equipos	**Unidades**	**Precio**
Raspberry Pi(suministro y colocación)	1	30€
Futaba s3303 (suministro y colocación)	2	20€
Estructura PAN-TILT(suministro y colocación)	1	10€
Batería 9 voltios (suministro y colocación)	1	4 €
Cable Etherne 1,5 m (suministro y colocación)	1	5 €
Raspberry Camera Module (suministro y colocación)	1	30 €
Ordenador de control *-2 GB RAM* *-Procesodor Intel core I3* *(suministro y colocación)*	1	400 €
TOTAL		499 €

Tabla 7: Presupuesto del sistema propuesto.

Bibliografía

1. *Object recognition from local scale-invariant features.* **Lowe, David G.** September de 1999, International Conference on Computer Vision.

2. *A combined corner and edge detector.* **Harris, C.** 1988, Alvey Vision Conference.

3. *Rapid Object Detection using a Boosted Cascade of Simple Features.* **Viola, Paul y Jones, Michael.** 2001. CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION.

4. *Background subtraction techniques: a review*.* **Piccardi, Massimo.** 2004. IEEE International Conference on Systems, Man and Cybernetics .

5. *Adaptive background mixture models for real-time tracking .* **Stauffer , Chris y Grimson, W.E.L.** 1999, Proc. of CVPR 1999 .

6. *Background Modeling using Mixture of Gaussians for Foreground Detection - A Survey.* **Bouwmans, T., El Baf, F. y Vachon , B.** Recent Patents on Computer Science 1, 3 (2008) 219-237.

7. *Histograms of Oriented Gradients for Human Detection.* **Dalal, Navneet y Triggs, Bill.** 2005. Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) 1063-6919/05.

8. *Fast Human Detection Using Motion Detection and Histogram of Oriented Gradients.* **Beiping , Hou y Wen , Zhu .** 2011, JOURNAL OF COMPUTERS, VOL. 6, NO. 8 .

9. *The estimation of the gradient of a density function, with applications in pattern recognition .* **Fukunaga, K. y Hostetler, L.** 1975 . Information Theory, IEEE Transactions on, 21(1):32–40 .

10. *Computer video face tracking for use in a perceptual user interface.* **Bradski, G.R.** 1998, Intel Technology Journal .

11. *Tracking of Humans Using Masked Histograms and Mean Shift.* **Ben-Israel , Elad .** 2007.

12. *Distinctive Image Features from Scale-Invariant Keypoints.* **Lowe, David G.** s.l. : International Journal of Computer Vision, 2004.

13. *People Tracking via a Modified CAMSHIFT Algorithm.* **Fahad Fazal Elahi , Guraya , Bayle , Pierre-Yves y Alaya Cheikh , Faouzi .**

14. *Doctorado en Tecnologías de las Comunicaciones - Procesado Digital de Señales en Comunicaciones (Curso 2003/04).* [En línea] http://www.tsc.uc3m.es/~mlazaro/Docencia/Doctorado/FiltAdapt/Kalman.pdf.

15. **kalman, Obtención de las ecuaciones del filtro del.** [En línea] http://www.depeca.uah.es/depeca/repositorio/asignaturas/32328/Tema4.pdf.

16. **I, Universidad Jaume.** Seguimiento – Filtro de Kalman. [En línea] http://www.vision.uji.es/courses/Doctorado/TAVC/TAVC-seguimiento.pdf.

17. **Bradski, G.R. y Kaehler, A.** *Learning OpenCV .* s.l. : O'Reilly Media.

18. OpenCV (Open Source Computer Vision) Wiki. [En línea] http://code.opencv.org/projects/opencv/wiki .

19. **Laganière, Robert.** *OpenCV 2 Computer Vision Application Programming CookBook.*

20. issues 1-19. *The MagPi. http://www.themagpi.com/es/*