Universidad de Extremadura

Escuela Politécnica

Grado en Ingeniería de Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

'Desarrollo de un sistema de seguimiento de objetos veloces en una imagen en tiempo real'

Carlos Mourato Buenavida

Julio 2016

Universidad de Extremadura

Escuela Politécnica

Grado en Ingeniería de Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

'Desarrollo de un sistema de seguimiento de objetos veloces en una imagen en tiempo real'

Autor: Carlos Mourato Buenavida

Fdo:

Director: D. Antonio Gordillo Guerrero

Fdo:

Co-Director: D. Pedro Núñez Trujillo

Fdo:

Tribunal Calificador

Presidente: José Moreno del Pozo

Secretario: Ramón Gallardo Caballero

Vocal: Horacio Manuel González Velasco

CALIFICACIÓN:

Julio, 2016.

A mi familia.

"A veces la persona que nadie imagina capaz de nada
es la que hace cosas que nadie imagina"
- Alan Turing -

Abstract

This Project deals with a system of motion detection of objects throughout an image processing, in which the image is captured with the help of a CCD camera, that goes over to be processed with a reduced sheet, where a laser pointer points at the object in motion. Fort he assembly a solution of free hardware and software has been used, chosing Python as programation lenguage to encode the processing and control algorythms. This computer uses its exit gates to send information to some servomotors that indicates where the object in motion is within its range of gathering, pointing at it with a laser pointer.

The drawing and building tasks have been made in the RoboLab laboratory and Smart Open Lab at Technical College in Cáceres.

Keywords:
Image processing, automatic control, real time, servomotor.

Resumen

Este trabajo trata sobre un sistema de detección de movimiento de objetos mediante procesamiento de imagen, en el que con ayuda de una cámara se captura una imagen que pasa a ser procesada por un ordenador de placa reducida, en el que posteriormente un láser apunta al objeto en movimiento. Para el montaje se ha utilizado una solución de hardware y software libre, utilizando Python como lenguaje de programación para codificar los algoritmos de procesado y control. Dicho computador utiliza sus puertos de salida para mandar información a unos servomotores que indican donde se encuentra el objeto en movimiento dentro de su rango de captación, marcándolo con un láser.

Las tareas de diseño y construcción se han realizado en el laboratorio RoboLab y en el laboratorio Smart Open Lab de la Escuela Politécnica de Cáceres.

Índice general

Índice de figuras

Capítulo 1

Introducción

Gracias a las capacidades del procesamiento de imágenes que existen hoy día, se hace posible un proceso de monitorización y vigilancia en determinados casos, en los que se simula la percepción humana para una toma de decisiones automática en tiempo real. La cantidad de información que maneja e interpreta es bastante limitada para el ser humano, es por ello que es conveniente una maquinaria automática que pueda manejar mayor cantidad de información. No se pretende sustituir al ser humano, si no completar y ayudar en las funciones que este realizaría, ya que los sistemas artificiales actuales crean falsos positivos en los que hace necesario tener a un ser humano pendiente. [2]

El hecho de que el procesamiento de vídeo se pueda hacer en tiempo real y mediante iteracción sin mandos, hace que sea una línea de investigación actual y activa, intentando mejorar la velocidad de toma de datos y el rendimiento computacional.

Si la optimización de toma de decisiones en procesamiento de imágenes, es utilizada para mover motores o servomotores, hay que tener en cuenta que sus movimiento son lentos y éstos suelen tener bastante holgura. Si se quiere evitar esto hay que tener en cuenta que la calidad es proporcional al precio.

1.1. Motivaciones

Las motivaciones principales que existen para llevar a cabo este trabajo fin de estudios son las de investigación, innovación y desarrollo de productos.

El sistema a desarrollar se encuentra a la orden del día en cuanto a procesamiento de imágenes en visión artificial, ya que éste es utilizado por sistemas de seguridad, tráfico, prevención de accidentes, caza, etc.

La investigación y la innovación son conceptos que suelen ir juntos en este tipo de proyectos, es por ello que es muy importante saber desenvolverse en estas competencias.

Con este trabajo fin de estudios se pretende conocer un lenguaje de programación que queda aislado en el programa de la titulación, así como reforzar las competencias de procesamiento de imagen y electrónica vistas a lo largo del grado.

1.2. Objetivos

El objetivo principal de este proyecto es el desarrollo de un sistema que sea capaz de manera autónoma de seguir objetos mediante procesamiento de imágenes en tiempo real y apuntar al objeto que se está moviendo. Para ello conviene destacar los siguiente objetivos secundarios:

- Adquirir más conocimientos de tratamiento de imagen y electrónica.
- Profundizar en los algoritmos de procesamiento de imagen.
- Mejorar la compresión de sistemas operativos tipo Unix
- Realizar un proyecto innovador
- Aprender técnicas de diseño electrónico y de desarrollo de productos (diseño 3D)
- Aprender a manejar servomotores con los pines GPIO de la placa de desarrollo a través de código Python

1.3. Organización de la memoria

Este documento se divide 4 capítulos principales, 4 anexos y una bibliografía.

El primer capítulo analiza las distintas posibilidades para desarrollar el sistema, en el que veremos las distintas etapas que tiene el sistema de procesamiento de imagen y las posibilidades para montar el elemento hardware.

El segundo capítulo estará dividido en dos, diseño software y diseño hardware. En el primero trataremos de concretar los sistemas que hemos elegido para procesar la imagen y captar el objeto en movimiento, en el segundo trataremos la manera que se ha adoptado para realizar el proceso para apuntar al objeto en movimiento junto con algunas mejoras que se le han implementado.

En el tercer capítulo se exponen los resultados de cada una de las partes que se explican en el capítulo dos.

En el cuarto capítulo se realiza la conclusión de nuestros logros tras haber finalizado el prototipo y ver las limitaciones y futuras mejoras que puede tener.

Finalmente se añaden diversos anexos que ayudarán a comprender mejor el proyecto mostrando el código utilizado.

Capítulo 2

Estado del Arte

2.1. ¿Qué es el procesamiento de imagen?

Según indicaba Gregory A. Baxes [3] en 1994, el procesamiento de imagen se había convertido en algo común en su día. Actualmente es utilizado en el trabajo de la industria, en laboratorios, en uso comercial utilizando programas como editor de gráficos rasterizados, las cadenas de televisión y periódicos, y además, en la industria armamentística es implementado para bombas inteligentes y misiles autodirigidos. En los últimos años ha tomado un gran peso debido al auge de la visión artificial e iteracción sin mandos.

Habitualmente encontramos procesamiento de imagen en todo nuestro alrededor, por ejemplo, la manipulación de lentes de contacto, en las que se hace una corrección geométrica de la lente para predistorsionar la imagen y contrarrectar la distorsión del ojo damnificado. Otra forma de procesamiento de imagen se puede hallar en la reflexión de una imagen en un vaso de agua. La imagen se invierte geométricamente y, a menudo, distorsionada por el movimiento del agua.

En las televisiones comerciales también encontramos procesamiento de imágenes diariamente, ya que con los controles de ajuste podemos modificar el brillo, el contraste, el tinte y el color de la imagen que se muestra por la pantalla. Estos controles afectan al nivel de negro de la señal de vídeo, la ganancia, la señal de saturación, etc.

La manipulación y análisis de información gráfica se denomina, en términos generales, procesamiento de imagen.

El interés de los métodos de procesamiento de imagen provienen de dos principales áreas de aplicaciones [4]: mejora de la información gráfica para la interpretación humana y el procesamiento de datos de la escena para percepción de máquinas automatizadas (tema en el que nos concentraremos posteriormente)

Uno de los principales problemas en la mejora de la calidad visual de las primeras imágenes digitales estaba relacionado con la selección de procedimientos de presentación y distribución de niveles de brillo. Tras esto se siguió mejorando los métodos de procesamiento de imágenes digitales, en los que se consiguió realizar pruebas en ordenadores de gran escala y enfocar el potencial al procesamiento de

imágenes. Siguiendo así, mejorando día a día, se ha conseguido llegar a lo que hoy día tenemos como procesamiento digital de imágenes, tan importante en la industria de la informática o el cine. Tal es así, que se encuentra entre los supercomputadores más potentes del mundo el utilizado para realizar superproducciones de películas. Avatar, la película de James Cameron, necesitó una red compuesta por 40 000 microprocesadores y más de 104 terabytes de RAM para ser renderizada. Todo este hardware se encuentra dentro de 4.000 servidores BLADE BL2X220c fabricados por Hewlett-Packard, y distribuidos dentro de 34 racks.

Normalmente, en procesamiento de imágenes se sigue un patrón común, representado en la figura 2.1

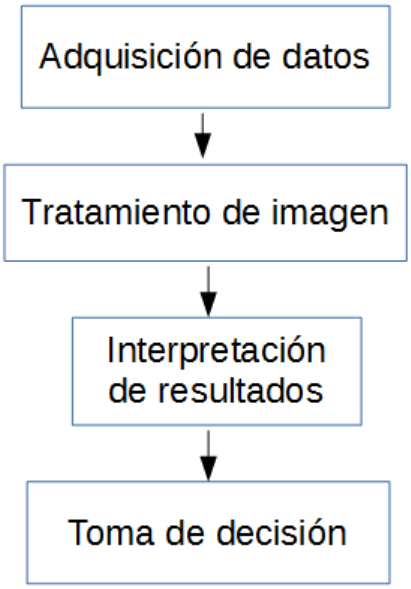


Figura 2.1: Diagrama de flujo de procesamiento de imágenes

2.2. Adquisición de datos

El mecanismo utilizado en visión artificial para sustituir al ojo humano es, por antonomasia, una cámara. La calidad de la cámara, entre otras cosas, hace que sea similar al ojo para poder interpretar ciertas cualidades. Existen diversos tipos de

cámaras, dependiendo del uso a realizar.

Uno de los objetivos de los métodos de adquisición de imágenes se encuentra en adaptar la iluminación conforme a como lo haría la retina, ya que el rango de niveles de intensidad de luz a la que el sistema visual humano puede adaptarse es enorme, del orden de 10^{10}, desde el umbral escotópico al límite del deslumbramiento. La capacidad del ojo para discriminar entre los cambios de intensidad de la luz en cualquier nivel de adaptación específica es también de gran interés. Cuando utilizamos una cámara para la obtención de imágenes se utiliza el objetivo de dicha cámara para adaptar el rango de nivel de luz, tal como haría la retina en el sistema visual humano. [3]

Dos fenómenos demuestran claramente que el brillo percibido no es una simple función de la intensidad; el primero se basa en el hecho de que el sistema visual tiende a la suboscilación o sobrepasar alrededor del límite de las regiones de diferentes intensidades; el segundo fenómeno, llamado contraste simultáneo, está relacionado con el hecho de que el brillo percibido de una región no depende simplemente de su intensidad.

Los tipos de imágenes en las que estamos interesados son generados por la combinación de una fuente de 'iluminación' y la reflexión o absorción de energía de esa fuente por los elementos de la 'escena'. Éstas generan una imagen 2-D utilizando un solo sensor, que tendrá desplazamientos relativos, tanto en las direcciones 'x' e 'y' entre el sensor y el área a explorar. Durante las últimas décadas, los dispositivos de adquisición de imágenes han experimentado un rápido desarrollo y han provocado la investigación sobre el registro automático de imágenes[5]

2.3. Tratamiento de la imagen

El principal objetivo de los métodos de tratamiento de imágenes es procesar una imagen para que el resultado sea más adecuado que la imagen original para una aplicación específica.

Una de las primeras cosas que se suelen hacer al tratar con imágenes, para posteriormente procesarla, es utilizar las conversiones a distintos formatos de colores, como pueden ser el modelo RGB, modelo HSV, escala de grises, etc [6]

El tratamiento de la imagen se divide en dos grandes categorías: métodos en el dominio espacial y métodos en el dominio de la frecuencia. El término dominio espacial se refiere al propio plano de la imagen, se basa en la manipulación directa de los píxeles de una imagen. Las técnicas del procesamiento en el dominio de frecuencia se basan en la modificación de la transformada frecuencial de una imagen. [4]

Los métodos en el dominio espacial, como ya hemos dicho anteriormente, son los que se trabajan directamente sobre los píxeles de la imagen. Entre ellos se encuentran algunos con los que vamos a trabajar posteriormente, y que serán expuestos. Hay gran cantidad de métodos en el dominio espacial, pero cabe destacar los filtrados con máscaras, mejora del contraste, imágenes en negativo, compresión de rangos dinámicos, umbralización, ecualización del histograma, sustracción de imágenes, promediado de imágenes, filtrados espaciales, etc.

Los métodos en el dominio de la frecuencia, como hemos expuesto anteriormente, se basan en la modificación mediante transformadas en el dominio de la frecuencia. Según el uso que se le vaya a dar, existen múltiples transformadas, como la de Fourier, siendo la más conocida, la del Coseno Discreta, la de Hadamard, la de Hotelling, la de Walsh, etc. Las transformadas se usan para realizar algún tipo de compresión o para después ser pasadas por determinado filtro, como puede ser el filtro paso-bajo o el filtro paso-alto.

En la etapa de tratamiento de la imagen se suele realizar una extracción de características, que permiten representar y describir los patrones de interés de la imagen. Es un paso en el reconocimiento de patrones y segmentaciones en las que las medidas u observaciones son procesadas para encontrar características que puedan ser usadas para asignar objetos de determinada clase [7]. Los clasificadores son una herramienta esencial que se propone como otra posibilidad al utilizar tratamiento de imágenes. Su utilización facilita la elaboración y el acceso autodirigido de programas automatizados que tiene como fin una interpretación de resultados.

2.4. Interpretación de resultados

A la hora de interpretar resultados obtenemos imágenes ya preparadas para nuestro objetivo final, que será el de la toma de decisión. Según el proceso final que

hagamos después de la etapa de tratamiento de la imagen podemos clasificar dicha imagen, por ejemplo, para centrarnos en regiones concretas de la imagen o para identificar coches o personas.

En esta etapa, los clasificadores se encargan de asignar un conjunto de características dado a una clase con la que se encuentra una mayor similitud, de acuerdo a un modelo introducido anteriormente. Es frecuente utilizar la clasifición de patrones frente a otras técnicas cuando los objetos tienen características en común, pero sujetas a variaciones que se desconocen. Cuando las variaciones son pequeñas se suelen recurrir a otros métodos más sencillos, como por ejemplo, emparejamiento por plantilla. Los clasificadores se suelen usar para segmentación por imágenes (textura, color), reconocimiento de objetos, control de calidad, detección de cambios o defectos en objetos y reconocimiento óptico de caracteres.

2.5. Toma de decisión

Tras haber extraído las características y clasificado el objeto requerido se pasa a realizar una acción final. Por ejemplo, tras haber realizado un proceso de reconocimiento de huella dactilar, se interpreta el resultado de distintos tratamientos de imágenes, se clasifica y se relaciona con una persona concreta, a la que posteriormente se toma la decisión de abrir la puerta para que éste pueda pasar. Por lo que la toma de decisión, la acción final, es la apertura de la cerradura tras la decisión del microcontrolador o microprocesador que interpreta los datos

2.6. Detección de movimiento

Las líneas de investigación sobre la detección de movimiento mediante imágenes en tiempo real están bastante desarrolladas. Son muchos los métodos y modelos destacados para realizar tal tarea, pero nosotros intentaremos centrarnos en los más completos y actuales. Esta línea, como casi todas las realizadas mediante procesamiento de imágenes, siguen el esquema de la figura 2.1. La detección de movimiento se suele realizar con una cámara fija, como vamos a realizar en este proyecto, pero también se puede realizar con una cámara en movimiento, pero el procesamiento de la

imagen en este caso se vuelve más complejo y caro, computacionalmente hablando.

Este tipo de proyectos siempre está sujeto a factores negativos externos que producen variaciones en imágenes, como puede ser los cambios de luminosidad, sombras, o pequeños movimientos. Existen diversos algoritmos que intentan paliar estos defectos.

La detección de movimiento, al tener que hacerse en tiempo real, complica bastante el proceso, ya que se necesitan microprocesadores que realicen muchas operaciones por segundo. Es necesario elegir una buena computadora para asegurarnos una buena automatización del proyecto.

2.6.1. Segmentación de la imagen

Uno de los procesos claves es la segmentación de la secuencia de imágenes para poder detectar el movimiento de objetos móviles en regiones del espacio bidimensional. Alguno de los métodos más conocidos en la segmentación del movimiento son la sustracción del fondo, diferenciación temporal y flujo óptico [8]

- Con la segmentación de movimiento utilizando el método de sustracción del fondo, conseguimos diferenciar la zona de la imagen que está en movimiento, especialmente en tramas que se encuentran fondo estático. Consideramos una imagen del fondo como referencia y se realiza la diferencia píxel a píxel con la trama actual. Es uno de los métodos más populares, ya que es muy simple de realizar, pero tiene como inconveniente la gran sensibilidad de cambios de iluminación y pequeños movimientos del sistema de captación de la escena.

- Con el método de diferenciación temporal hacemos uso de las diferencias de las distintas tramas de imágenes consecutivas de una secuencia para extraer parámetros característicos del movimiento. Es muy utilizado para escenas dinámicas, pero es deficiente en la extracción de píxeles de movimiento, pudiendo llegar a faltar información al desaparecer éstos.

- El método de flujo óptico se basa en la extracción de características de vectores de flujo de objetos en movimiento para detectar objetos en movimiento en tramas de imágenes. Se utilizan los vectores de flujo basándose en los contornos

de los objetos para extraer características del movimiento. Tiene un bajo rendimiento computacional, por lo que lo hace muy difícil su utilización en tiempo real.

2.6.2. Seguimiento del objeto

La segmentación del movimiento va relacionada comúnmente con el seguimiento del objeto, por lo que muchos algoritmos de seguimiento interactúan constantemente con los algoritmos de detección del movimiento [8]. Entre los métodos más destacados de seguimientos de objetos se encuentran: seguimientos basados en la región, seguimientos basados en el contorno activo, seguimientos basados en las características y seguimientos basados en el modelo [9] . Las distintas técnicas se pueden combinar para hacer más efectivo el proceso.

- La técnica de seguimientos basados en la región contiene algoritmos que siguen los objetos según las variaciones de la imagen correspondiente a los objetos en movimiento.

- La técnica de seguimientos basados en el contorno activo realiza algoritmos de seguimientos basados en el contorno activo que tienen los objetos identificados como en movimiento y delimitados por su contorno. La actualización de los contornos se realiza de forma automática y sucesivamente entre las distintas tramas de la secuencia.

- La técnica de seguimientos basados en las características utiliza algoritmos que realizan el reconocimiento y seguimiento de objetos utilizando parámetros para extraer características de alto nivel y así compararlas entre las distintas tramas. Esta técnica se puede subdividir en varias categorías según su función: algoritmos basados en características globales, algoritmos basados en características locales y algoritmos basados en dependencia gráfica. Éstos pueden ser combinados entre sí.

- La técnica de seguimiento basados en el modelo se estructura en algoritmos de seguimiento mediante el método de la comparación de modelos previos de interés. Estos algoritmos se construyen con diversas herramientas de diseño.

2.6.3. Umbralización

La umbralización es una técnica importante en la segmentación de imágenes, usada para identificar y separar objetos del fondo, en base a la distribución de los niveles de gris o la textura de los objetos en las imágenes[10]. Hay muchos procedimientos para realizar la umbralización, pero no hay un método que se aplique a todas las imágenes, ésto depende del uso al que se va a dedicar y varios factores más. Las distintas técnicas se van a clasificar como propusieron Sezgin y Sankur[11]. Las técnicas se clasifican en métodos basados en la forma del histograma, métodos basados en la clusterización, métodos basados en la entropía, métodos basados en los atributos de la imagen, métodos basados en información espacial y métodos basados en características locales.

- Los métodos basados en la forma del histograma busca las diferentes propiedades del histograma, como picos, valles, etc. y así utilizar una envolvente convexa a su histograma, o curvatura.

- Los métodos basados en la clusterización modelan el histograma de la imagen como superposiciones de funciones gaussianas. Los algoritmos se dedican al análisis de segmentación de fondo y objetos superpuestos.

- Los métodos basados en la entropía buscan la entropía de los distintos niveles de gris de la imagen. Busca la máxima entropía, que será interpretada como la máxima información transmitida y lo usa como umbral óptimo.

- Los métodos basados en los atributos de la imagen utiliza técnicas de buscan un umbral mediante similitud de la imagen original y la binarizada.

- Los métodos basados en información espacial utiliza la información espacial de los píxeles, a diferencia de los métodos anteriores que utilizan la información de los niveles de gris.

- Los métodos basados en características locales adaptan el umbral en los distintos píxeles en función de características locales de la imagen, como varianza, media, rango, etc.

2.7. Raspberry Pi

Raspberry Pi es un ordenador de placa reducida de bajo coste realizado por la Fundación Raspberry Pi[12], con el objetivo de fomentar las ciencias de la computación en las escuelas. En la figura 2.2 se puede ver una figura de la placa Raspberry Pi.

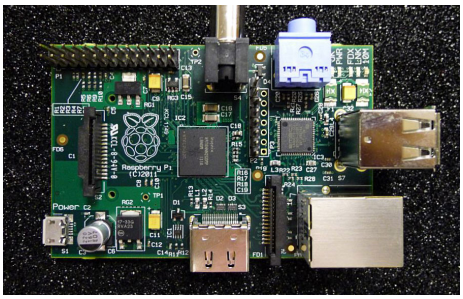


Figura 2.2: Placa de desarrollo Raspberry Pi

El microprocesador Raspberry utiliza principalmente sistemas operativos basados en código abierto, como son Linux, Ubuntu, Debian, etc. En 2012 se lanzó la distribución Raspbian, derivada de Debian, y que está optimizada principalmente para el hardware de Rapsberry Pi. Aunque hay muchas más distribuciones, la fundación Raspberry Pi recomienda la distribución Raspbian para iniciarse y es la que nosotros vamos a utilizar en el proyecto.

Entre las grandes características de la plataforma Raspberry se encuentra la gran cantidad de soporte online con el que cuenta, ya que existen gran cantidad de proyectos en internet con los que poder iniciarse fácilmente.

En este proyecto vamos a utilizar la placa de desarrollo Raspberry Pi 2 modelo B, que cuenta con las siguientes características técnicas[12]:

- Procesador: Quad-Core Cortex A7 a 900MHZ
- RAM: 1GB

- Puertos: 4 USB 2.0, 40 GPIO pin, HDMI, ethernet, combo audio/mic, interfaz de cámara (CSI), interfaz de Pantalla (DSI), micro SD y núcleo gráfico 3D.

2.8. Servomotores

En este proyecto vamos a utilizar servomotores que serán accionados por la placa de desarrollo Raspberry Pi a través de sus puertos GPIO.

Los servomotores son unos motores especiales de corriente continua que se caracterizan por posicionarse una posición concreta de forma rápida dentro de su intervalo de posición. Para ello el servomotor opera con un tren de pulsos que se corresponde con la posición a adoptar. Para ello utiliza la modulación PWM, que consiste en una técnica que modifica el ciclo de trabajo de una señal periódica. Un ejemplo de una señal PWM se muestra en la figura 2.3.

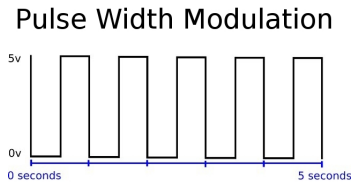


Figura 2.3: Ejemplo de modulación PWM

La construcción más común de un circuito para modulación PWM se realiza mediante un comparador con dos entradas y una salida. Una de las entradas se conecta a un oscilador de onda de dientes de sierra, mientras que la otra queda disponible para la señal moduladora. En la salida la frecuencia es generalmente igual a la de la señal dientes de sierra y el ciclo de trabajo está en función de la portadora.

En la figura 2.4 se muestra un diagrama de ejemplo de utilización de modulación PWM.

La principal desventaja que presentan los circuitos PWM es la posibilidad de que haya interferencias generadas por radiofrecuencia. Éstas pueden minimizarse ubicando el controlador cerca de la carga y realizando un filtrado de la fuente de

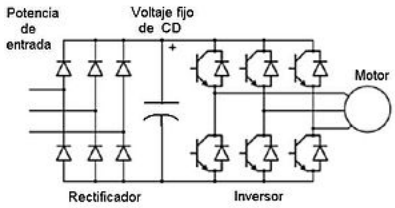


Figura 2.4: Diagrama de ejemplo de la utilización de la modulación de ancho de pulsos en un variador de frecuencia

alimentación.

Un servomotor está generalmente formado por un amplificador, un motor, un sistema reductor formado por ruedas dentadas y un circuito de realimentación, todo en un misma caja de pequeñas dimensiones. En la figura 2.5 se muestran varios servomotores con distintos tamaños.

Figura 2.5: Ejemplo de servomotores

Capítulo 3

Diseño

El diseño del prototipo va a constar de dos partes: diseño software y diseño hardware.

3.1. Diseño software

3.1.1. Herramientas software

En este punto vamos a realizar el proceso software de procesamiento de imagen en tiempo real. Para este tipo de proyectos se cuenta multitud de plataformas disponibles, nosotros nos hemos centrado en la placa de desarrollo Raspberry Pi, siendo éste un ordenador de placa reducida y de bajo coste, muy útil para los tipos de proyectos como el que se va a desarrollar.

Para poder realizar el proyecto propuesto se ha elegido 'Python' [13] como lenguaje de programación para realizar el algoritmo de procesamiento de imagen. Python en un lenguaje intuitivo, de fácil manejo y soporta orientación a objetos, programación imperativa y programación funcional. Una de las características más atractivas que tiene Python es la fácil implementación de la librería OpenCV[1], destinada a procesamiento de imagen. OpenCV es una de las librerías más completas, con un entorno de desarrollo fácil y con gran eficiencia. Además, para hacer más fácil la manipulación del código de nuestro proyecto utilizaremos la librería Numpy [14], siendo ésta una extensión de Pyhton que utiliza operaciones matemáticas de alto nivel para operar con vectores y matrices.

Para captar la imagen vamos a contar con una cámara CCD utilizada por la consola Play Station 2. Ésta se utiliza para juegos que la consola interactuaba con el ser humano a través de la cámara, por lo que se hizo esencialmente para procesar imágenes. En la figura 3.1 se muestra un ejemplo de una cámara EyeToy.

3.1.2. Desarrollo

El proyecto a desarrollar consiste en detectar movimiento según una cámara fija, en la que a posteriori, y tras un desarrollo de procesado de imagen, se va a conseguir unas coordenadas 'x' e 'y', azimut y elevación, que indicarán el punto medio del objeto que se está moviendo. Para ello hemos de analizar lo visto en el apartado de 2.6, en el que se explicaba brevemente algunos de los métodos a desarrollar para

Figura 3.1: Cámara EyeToy

detectar movimiento. Primeramente nos centraremos en el apartado 2.6.1, que explica diferentes métodos de segmentación del movimiento.

La diferenciación temporal está basada en las diferencias de las distintas tramas de una secuencia de imágenes, obteniendo así las características de los píxeles en movimiento. A groso modo se dividen en tres: Comparación de plantillas, seguimiento del núcleo y seguimiento de puntos [15].

El método de template matching, o comparación de plantillas, intenta detectar la silueta mediante filtros que captan el contorno, utilizando esta imagen de plantilla. En las siguientes secuencias de imágenes se realiza la comparación de la imagen actual con la plantilla que se ha obtenido en la detección, detectando una región en la que se situaría la silueta. Tiene como desventaja que si se modifica mucho la forma de la plantilla se pierde la silueta y por consiguiente no se apreciaría comparación. Este método no serviría para el proyecto ya que nuestro objeto a detectar no está definido previamente y no sabemos si podría cambiar su forma con forme atraviesa el rango de nuestro sistema de captación.

El método de seguimiento del núcleo, o tracking kernel, se basa en la correlación cruzada para medir similitud, por lo que es necesario una plantilla original. Igual que en el caso anterior, este método no serviría para nuestro proyecto ya que no dispondríamos una plantilla inicial. El método de tracking kernel, además, es muy sensible a cambios de iluminación.

El método de seguimiento de puntos utiliza algoritmos estadísticos para realizar el seguimiento del objeto. Este método tiene un problema de restricción de proximidad

debido a que el objeto no podría cambiar notablemente de una imagen al siguiente, ya que perdería su plantilla original [16]

Otro método de segmentación del movimiento indicado en el apartado 2.6.1, exponía el método del flujo óptico. Éste método es muy eficiente en cuanto a la capacidad de captar movimiento entre secuencias de imágenes. Se basa en el estudio de vectores de flujo en movimiento para detectar píxeles en movimiento. La restricción de este método viene a la hora de analizar su rendimiento computacional, ya que los algoritmos utilizados son muy pesados y lo hace inútil en procesamiento en tiempo real.

El método de sustracción del fondo consigue analizar si hay movimiento en el rango de captación del sistema realizando la diferencia de la imagen del fondo, captada previamente, junto con la imagen actual, refrescando continuamente. Este sistema se compatibiliza bien con nuestro proyecto inicial, ya que este método es propicio para fondos estáticos como el nuestro. Además el rendimiento computacional es alto debido a que los algoritmos de cálculo son bastante simples.

Sabemos que para que tengamos un algoritmo lo más eficiente posible necesitamos utilizar el método de segmentación de imagen realizando la sustracción del fondo.

Se propone captar un fondo dinámico y utilizarlo como referencia para posteriores cálculos. Por otro lado se captará en cada momento una imagen actual que será captada por nuestro sistema de adquisición de datos, una cámara CCD. Ambas imágenes son captadas en RGB. Posteriormente habría que realizar una diferencia absoluta entre la imagen del fondo y la imagen actual.

En el campo de la visión artificial hay un abanico de posibilidades en este punto, ya que la diferencia absoluta entre el fondo y la imagen actual, dependiendo del uso a realizar, se puede hacer de varias formas, nosotros nos centraremos en estas tres:

- Realizar un proceso de transformación de RGB a escala de grises, y un posterior filtrado por un filtro de Gauss para suavizar la imagen, y realizar la diferencia absoluta de ambas imágenes. Reflejado en la figura 3.2

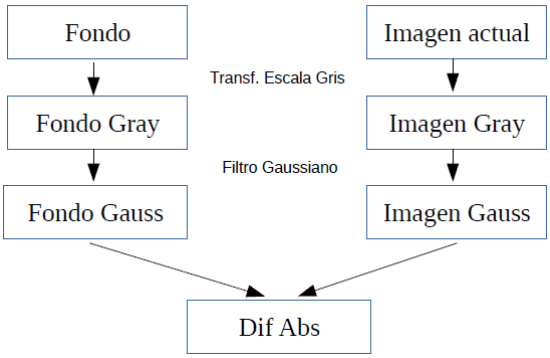


Figura 3.2: Caso 1 de sustracción del fondo

- Realizar un proceso de transformación de RGB a escala de grises, un proceso de umbralización y pasar a una imagen binaria y tras esto, realizar la diferencia absoluta de ambas imágenes. Reflejado en la figura 3.3

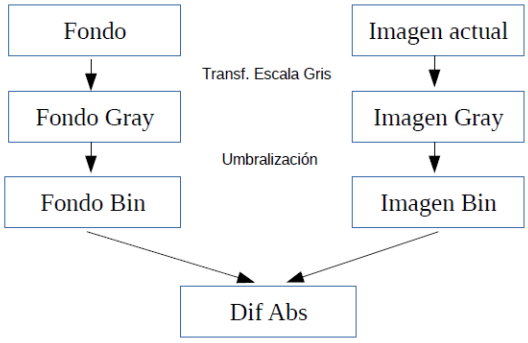


Figura 3.3: Caso 2 de sustracción del fondo

- Realizar la diferencia en RGB. Reflejado en la figura 3.4

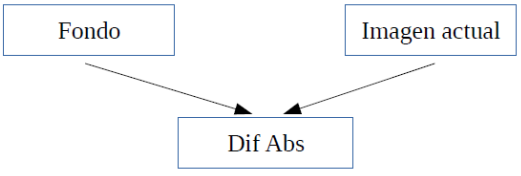


Figura 3.4: Caso 3 de sustracción del fondo

Hay que tener en cuenta que en este tipo de proyectos unos pequeños cambios de iluminación producen errores muy significativos, siendo importante paliar al máximo esta deficiencia.

En [15] y [8] realizan el primer caso, mientras que [2] utiliza el tercer caso. Tras realizar varias pruebas, comprobamos que al realizar el tercer caso produce una mayor eficiencia de acierto, ya que conseguimos reducir el error de cambios de la iluminación al realizar la diferencia absoluta en formato RGB.

Tras llegar a este paso vamos a analizar, según la figura 3.5, cómo se va a desarrollar el sistema.

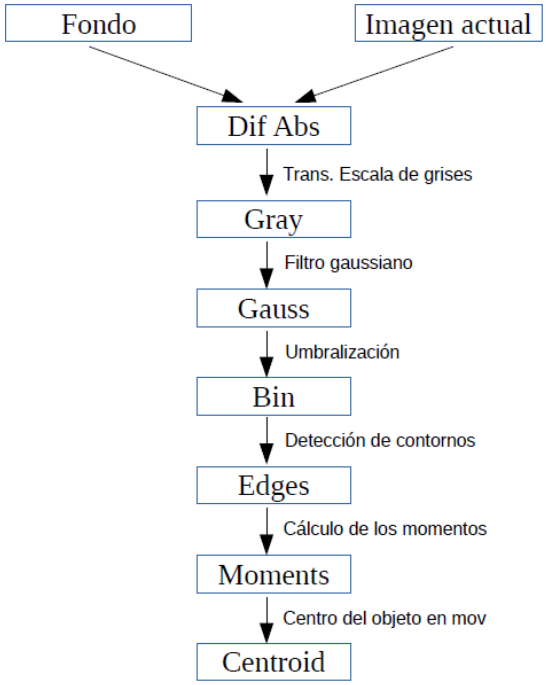


Figura 3.5: Diagrama de flujo del sistema

La captación del fondo se realizará en un inicio y se guardará como referencia. La imagen estará en formato RGB. La imagen del fondo se tendrá que ir renovando cuando no se detecte movimiento y la diferencia de una nueva imagen y la inicial sea pequeña, esto será debido a cambios progresivos de iluminación. Así conseguimos paliar, de alguna manera, esta deficiencia.

Posteriormente se irá captando la imagen, en formato RGB también, y se irá refrescando continuamente. Siendo ésta la imagen en tiempo real que será captada por la webcam.

En este momento se va a realizar la diferencia absoluta entre el fondo y la imagen en tiempo real, como hemos explicado anteriormente. Esta diferencia se realiza en formato RGB.

El siguiente paso a realizar será la transformación de la diferencia en RGB a escala de grises. En el apartado 2.3 se comentaba que había varios formatos para representar imágenes, como eran RGB, HSV, escala de grises, etc. El modelo de escala de grises consiste en identificar a cada píxel con un valor equivalente en una graduación de gris, sin embargo, el modelo de RGB consiste en identificar cada píxel en la composición de niveles de los tres colores primarios [4].

Tras el proceso de transformación a escala de grises se va a realizar un filtro de suavizado que expande ligeramente los niveles de gris de los píxeles que estén vecinos el uno al otro, como se puede observar la máscara de ejemplo de la figura 3.6

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Figura 3.6: Ejemplo de máscara filtro de suavizado gaussiano.

En el siguiente bloque debemos realizar un sistema de umbralización para poder pasar nuestra matriz de imagen en escala de grises a una matriz binaria. Como hemos visto en el apartado 2.6.3, se proponen varias técnicas para desarrollar la umbralización[17].

Hemos visto que el método basado en la forma del histograma busca picos y valles de una envolvente del histograma, lo que la búsqueda de estas características del histograma hace que sea un cálculo lento para un proceso en tiempo real.

El método basado en la entropía usa los niveles de gris en una imagen, éste mide la incertidumbre que describe la información contenida en una fuente. Lo que hace es aplicar un criterio que se basa en encontrar el umbral óptimo que maximice la entropía entre las dos clases, por lo que hace que sea lento en procesos en tiempo real, por ello este método no se ha utilizado.

El método basado en los atributos de la imagen consiste en técnicas que seleccionan un valor de umbral basado en atributos que miden la similitud entre la imagen original y la imagen binarizada. Se produce un retardo considerado alto para nuestro proyecto.

El método basado en características locales calcula el umbral por píxel, teniendo en cuenta el valor de los píxeles vecinos, lo que hace que haya un retardo grande en el cálculo, ya que hay que recorrer toda la matriz analizando uno por uno cada píxel de la imagen.

El método basado en la clusterización modela imágenes superpuestas, realizando una segmentación de fondo y de objetos superpuestos. Esto hace que sea un modelo bastante exitoso en cuanto a encontrar un umbral óptimo que diferencie bien la región del fondo del objeto superpuesto, en el histograma. En este método se encuentra el famoso cálculo de Otsu[17], que es uno de lo métodos más exitosos de segmentación de imágenes.

El método basado en información espacial utiliza la información global de los píxeles, utilizando la información de niveles de gris. Es un método simple y fácil de implementar [17].

Tras esto, vemos que el método basado en la clusterización y el basado en información espacial son los que más se adaptarían a nuestro proyecto. Se han realizado pruebas con ambas técnicas y resulta que da mejor resultados, en cuanto a tiempo de computación y objetivo final de nuestro proyecto, el método basado en información espacial, ya que la imagen que nos llega después de todo el proceso de tratamiento de imágenes es una imagen sencilla de umbralizar y tampoco se necesita demasiada precisión. Por lo que utilizamos un umbral intermedio entre el valor 0 de nivel de gris y el nivel máximo de 255 de nivel de gris, por lo que tenemos un umbral de 127 de nivel de gris. Con todo esto ganamos rapidez en el cálculo, pero perdemos una eficiencia al realizar el cálculo de umbralización de la imagen, ya que el método ba-

sado en la clusterización calcula un umbral óptimo bastante bueno y potente, como es el método Otsu.

En la figura 3.7 se ve un ejemplo de una imagen umbralizada.

Figura 3.7: Ejemplo de umbralización de una imagen

Teniendo ya la imagen binarizada procedemos a ver qué técnica se ajusta mejor al seguimiento del objeto. Estas técnicas son tan importantes como la segmentación del movimiento, ya que van ligadas, y para este tipo de proyectos se complementan. Como hemos visto en el apartado 2.6.2 hay varios métodos para seguir el objeto en movimiento.

La técnica de seguimiento basado en las características utiliza ciertas características del objeto para hacer su seguimiento, como curvas, rectas, vértices, etc. En nuestro caso no sabemos cómo será el objeto o si se puede deformar, por lo que éste será un método que no podremos utilizar.

La técnica de seguimiento basado en el modelo realiza el seguimiento basándose en la comparación de un modelo previo. Como hemos comentado anteriormente, nosotros no disponemos de un modelo fijo del objeto que necesitamos seguir, por lo que este método no se implementará.

Las técnicas de seguimientos basados en la región y en el contorno activo son los métodos que más se aproximan a nuestro proyecto. El método basado en las regiones utiliza las variaciones de las regiones de la imagen correspondiente a los objetos en movimiento, mientras que el método basado en el contorno activo utiliza las delimitaciones de los contornos, pero éste describe los objetos de formas más sencilla y con mayor rendimiento, por lo que es muy eficaz en coste computacional [9].

El método de cálculo de contornos pueden explicarse simplemente como una curva que une todos los puntos continuos (a lo largo de la frontera), que tiene el mismo color o intensidad. Los contornos son una herramienta útil para el análisis de la forma y la detección de objetos y reconocimiento [1].

Por ello, utilizamos filtros Canny para detectar los contornos, y así poder separar bien el fondo del objeto en movimiento, reduciendo así la matriz de puntos de la imagen a procesar. Dicho filtro utiliza un algoritmo que permite una localización precisa de los bordes. Esta opción tiene una mayor eficiencia computacional y exactitud, ya que evita posible rupturas o discontinuidades en los bordes, reduce la cantidad de delimitaciones ruidosas y es capaz de cerrar contornos.

A la hora de implementar la detección de contornos en el código de Python, OpenCV daba varias posibilidades. Una de ellas era una función propia que indicaba 'encontrar contornos', en el que devolvía una matriz con un procesamiento de contornos. Otra opción era utilizar el filtro Canny que tiene OpenCV por defecto. La opción de utilizar el filtro Canny directamente tiene mayor capacidad de procesamiento, ya que la función de 'encontrar contornos' obviaba ciertos objetos y no mostraba la realidad.

En la figura 3.8 tenemos la imagen de 'Lenna', tan famosa en procesamiento de imagen, y a su lado dicha imagen tras un filtrado con el algoritmo Canny.

Figura 3.8: Ejemplo de utilizar Filtro Canny

El siguiente proceso a realizar se centra en calcular los momentos de una imagen con contornos. Nos apoyamos en un proceso propio de la librería de procesamiento de imagen 'OpenCV', de la que ya tratamos en el apartado 3.1.1. Dicha función calcula

los momentos, hasta el 3er orden, de una forma vectorial o una forma rasterizada [1]. El resultado que devuelve la función 'momentos' viene almacenado en varias variables. Eligiendo una u otra, podemos calcular varias formas poligonales en la imagen. En nuestro caso, para calcular el centro del objeto utilizamos el cálculo del centroide de la imagen, que nos indicará el centro de masa del objeto en movimiento. En la figura 3.9 se puede observar como el centroide indica el centro de masas de la imagen tras un procesamiento de cálculo de contornos.

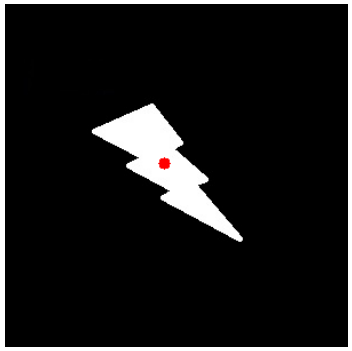


Figura 3.9: Ejemplo de figura con el centroide en el centro de masas[1]

Tras todo este proceso, se implementa una mejora, comentada anteriormente, para que el fondo se vaya actualizando si hay un cambio uniforme de luz. En concreto, se medirá la diferencia del fondo con la imagen actual y si es menor de un determinado umbral asimilamos que son parecidos y actualizamos el fondo con la imagen actual. Claro está, que en esta imagen actual no habrá movimiento y podrá ser tomada como fondo para la próxima iteración.

Todo lo expuesto anteriormente viene especificado en el Anexo A.

3.2. Diseño hardware

3.2.1. Desarrollo

En el diseño hardware vamos a realizar el proceso de construcción de circuitería y el proceso de movimiento de los servomotores para poder apuntar con el láser al punto medio del objeto en movimiento, que se desarrolla en el apartado 3.1.2.

Como hemos visto, vamos a utilizar el microprocesador de placa de desarrollo Raspberry Pi[12]. En ella se ha implementado el código para localizar el objeto en movimiento y encontrar el centroide. El siguiente punto consistirá en manipular los pines de la placa para poder mover los servomotores hasta el punto donde queremos. Vamos a instalar la librería GPIO [18], que nos ayudará a utilizar los puertos de salida y entrada en Python. En la figura 3.10 vienen indicado la numeración de los pines.

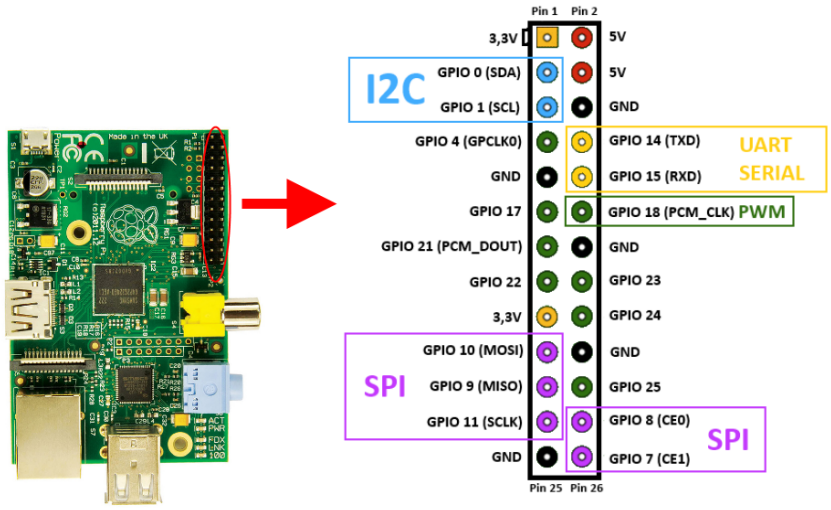


Figura 3.10: Puertos GPIO de Raspberry Pi

Hay que tener cuidado a la hora de declarar las entradas y salidas, y las manipulación de los puertos. Estos puertos trabajan a 3,3 V, y no a 5 V, como solemos estar acostumbrados al utilizar placas de microprocesadores y microcontroladores. Entre los pines se disponen de varios puntos de alimentación de 3,3 V y de 5 V, pero no es recomendable utilizar estos pines para conectar demasiada carga, ya que podemos saturar a la placa y quedarla inutilizada.

Para apuntar con el láser vamos a usar dos servomotores, uno para apuntar en el eje 'x', o azimut, y otro en el eje 'y', o elevación. Se ha elegido un modelo de servomotor comercial de bajo coste es el microservo TowerPro de 9g y modelo SG90, como el de la figura 3.11. El microservo necesita una alimentación de 5 V, pero ésta será alimentada de manera externa, ya que sería mucha carga para la placa del

microprocesador. Dicho servomotor tiene una capacidad de rotación de 180º, 90º para cada dirección, para ello utiliza una modulación por ancho de pulsos (PWM) para controlar el ángulo de giro, como hemos visto en el apartado 2.8. Tal como indica en su hoja de características[19] necesita una frecuencia de 50 Hz, que corresponde a un periodo de 20ms, y un ciclo de trabajo de 1,5 ms para que el servomotor se quede en el eje central, 2 ms para que se mueva 90º y 1 ms para que se mueva -90º.

Figura 3.11: Microservo SG90 de TowerPro

En la implantación en Python nos vamos a apoyar en la librería GPIO, instalada previamente. En ella da opción de introducir, a la salida de un pin, una modulación PWM de determinado tiempo y ciclo de trabajo. Para ello primero declaramos el pin y su exclusividad de querer hacer modulación PWM a una frecuencia de 50 Hz y posteriormente ayudándonos de las características de Python para utilizar el método de variables objeto para introducir un ciclo de trabajo concreto y modificarlo si queremos cambiar de posición, éstos serán especificados en porcentaje de ciclo en alto. En la práctica hemos comprobado que los ciclos de trabajo especificados en la hoja de características no son los mismos, es por ello que tras diversas pruebas se ha comprobado que con un ciclo de trabajo de un 7,5 % (1,5 ms) el servomotor se mueve a su sitio central, sin embargo con un ciclo de trabajo de un 10 % (2 ms)

el servomotor se mueve a 45º hacia a la derecha, y trabajo de un 5 % (1 ms) el servomotor se mueve a 45º hacia a la izquierda.

En nuestro caso se va a asumir que nuestra cámara tiene una relación de aspecto de 4:3 y que dispone de un ángulo de visión de 90º en azimut, por lo que en elevación tendrá un ángulo de visión de 67,5º, que vamos a aproximar a 70º. Es por ello, que en nuestro código para mover el servomotor vamos a realizar un código en el que de entrada tenemos el centro de masas de la imagen en movimiento, que su cálculo se ha explicado en el apartado 3.1.2. Se ha calculado el largo y ancho de la imagen y el centro, que en webcam comerciales el ancho es de 640 píxeles y el alto de 480 píxeles, por lo que el centro se situaría en los píxeles 320 y 240, respectivamente. En este momento tendríamos que tratar por separado cada una de las partes implicadas, por un lado el azimut y por otro la elevación.

Para conexionar los servomotores de azimut y de elevación colocaremos uno encima de otra de manera que sean solidarios. En el que para el rango de azimut disponemos del esquema de la figura 3.12.

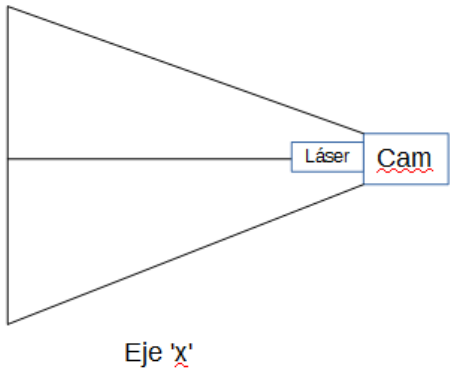


Figura 3.12: Imagen del servomotor en rango de azimut

Para el rango de elevación disponemos del esquema de la figura 3.13.

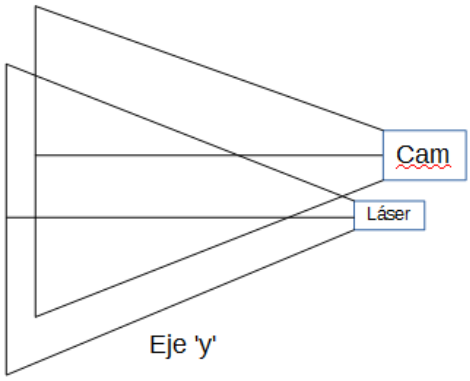


Figura 3.13: Imagen del servomotor en rango de elevación

Para tratar el rango de azimut primeramente dividiremos en tres casos, cuando sea igual al centro del eje, cuando sea superior al centro y cuando sea inferior al centro.

- Cuando sea igual al centro de la imagen utilizaremos un ciclo de trabajo de 1,5 ms, o lo que sería igual, un 7,5 %.
- Cuando la variable sea superior al centro de la imagen, tomaremos una regla de tres simple dividiendo la variable entre la longitud del eje, y multiplicaremos el ángulo máximo de emisión en el punto de mayor longitud del rango de azimut, en nuestro caso 45º, para hallar en los grados respecto al origen se encuentra el objeto en movimiento. Un ejemplo en la expresión 3.1

$$grados = (\frac{cx}{longitud}) * 45^{o} \tag{3.1}$$

  Seguidamente utilizaremos el ángulo hallado para calcular el ciclo de trabajo para activar el servomotor a su sitio correspondiente. Para ello vamos a realizar otra regla de tres, en la que partimos del ciclo de trabajo del centro, que es 7,5 %, y le restaremos el máximo de 45º, que es 5 %, que ésto corresponde a 45º, por otro lado tenemos la variable grados, con la que hallaremos su ciclo de trabajo correspondiente restando el valor a 7,5 %. Un ejemplo en la expresión 3.2

$$Ciclo = (\frac{grados * (7{,}5 - 5)}{45}) \tag{3.2}$$

Con este paso tendríamos ya el ciclo de trabajo para mover el servomotor en el caso de que el punto sea superior al punto central.

- Cuando la variable inferior al centro del eje realizaremos la ecuación de la expresión 3.1, pero en la variable 'cx' antes hay que restarle el valor del píxel central para poder igual y reajustar la situación. Posteriormente utilizaremos una expresión parecida a 3.2, pero para nuestro caso realizaremos un proceso distinto, ya que para el máximo punto de 45º corresponde un porcentaje de ciclo de trabajo de 10 %, por lo que a éste le restaremos el punto intermedio que será de 7,5 %, éste valor será multiplicado por la variable grados de nuestro caso, y dividido por 45º. Después, para obtener el valor del ciclo de trabajo exacto hay que sumar 7,5 al valor que nos había de realizar la operación. En la expresión 3.3 podemos ver un ejemplo de la ecuación utilizada.

$$Ciclo = (\frac{grados * (10 - 7{,}5)}{45}) \tag{3.3}$$

Para el cálculo en el rango de elevación se realiza de igual modo que lo calculado en el rango de azimut, pero teniendo en cuenta que el ángulo máximo es de 35º.

Lo que se ha explicado anteriormente se ha implementado en el código del proyecto, indicado en el Anexo A.

Para apuntar al objeto en movimiento se ha utilizado un láser de diodo común, como puede ser el láser KY-008 Keyes, como el que adjunta en la figura 3.14. Dicho láser está preparado para trabajar con un voltaje de 5 V y una resistencia en serie de 680 Ω, con una intensidad de 2,5 mA, por lo que el voltaje que llega al diodo es de 3,3 V, dado que nuestra placa Raspberry Pi da un voltaje de salida en sus pines de 3,3 V, no haría falta utilizar una resistencia [20]

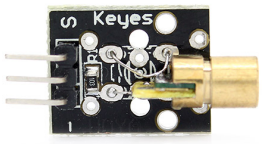


Figura 3.14: Láser KY-008 Keyes

Después del proceso de montaje de los servomotores y el láser, el sistema se muestra en la figura 3.15

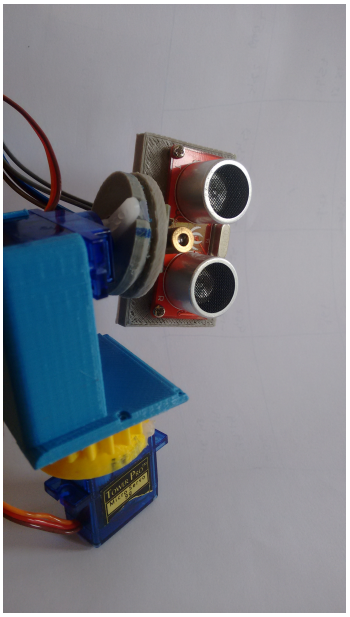


Figura 3.15: Montaje de servomotores y láser.

3.2.2. Mejoras

La placa de desarrollo Raspberry Pi tiene ciertas desventajas, entre las que está la obligatoriedad de manipulación de la placa mediante interfaz gráfica. Es por ello, que se va a desarrollar una botonera para poder manipular la placa sin necesidad de recurrir a los periféricos típicos de un ordenador común. Como se observa en el

Apéndice E, hay varios botones utilizando una resistencia pull-down para activar los pines GPIO de la placa reducida.

Se ha implementado un código en Pyhton para poder iniciar el programa que hemos desarrollado sin recurrir a la terminal de la Raspberry. Dicho script se llama 'Inicio.py' y se muestra en el Anexo B.

Dentro de las desventajas de la placa también está el apagado de ésta, ya que sin manipulación con la interfaz gráfica no se podría realizar un apagado seguro. Por ello hemos desarrollado un script llamado 'shutdown.py' que da la posibilidad de apagar de manera segura la placa mediante un pulsador, como se muestra en el Anexo C.

Seguidamente, tras haber realizado los dos script anteriores, vamos a manipular la terminal de la Raspberry Pi para poder iniciar éstos al iniciar el Sistema Operativo Raspbian. Este proceso hay que realizarlo para cada uno de los script. Para ello hay que seguir las instrucciones siguientes cambiando la palabra 'script' por el nombre del código, en nuestro caso 'Inicio' y 'Shutdown'.

Para empezar vamos a introducir en la terminal:

>>sudo nano /etc/init.d/script-init

Posteriormente copiamos el texto mostrado en el Anexo D y guardamos. Seguidamente haremos el script ejecutable escribiendo en la terminal:

>> sudo chmod 755 /etc/init.d/script−init

Por último activaremos el arranque automático escribiendo en la terminal:

>> sudo update-rc.d script-init defaults

Reiniciamos la placa y en el próximo arranque dispondremos del funcionamiento de la botonera para controlar el inicio de nuestro programa y el apagado seguro sin necesidad de interactuar con la interfaz gráfica.

Otra mejora que se ha implementado es la disposición de un diodo led para indicar que el programa está funcionando. Consistirá en un diodo led verde que, al iniciar

el programa, parpadeará 3 veces para indicar que la cámara está captando el fondo. Posteriormente se quedará fija, esto indicará que el programa ya está funcionando y está realizando los cálculos oportunos para captar el objeto en movimiento.

Como se puede observar en la figura 3.15, el láser tiene solidario a él un sensor de ultrasonidos que calcula la distancia al objeto al que apunta. El sensor de ultrasonido es igual que el que se muestra en la figura 3.16

Figura 3.16: Sensor de ultrasonidos

Como indica su hoja de características [21], éste tiene 3 pines de interacción. Uno se refiere a conexión de alimentación de 5V, otro a GND, y el otro sirve de emisor y receptor a la vez, es decir, la placa Raspberry Pi manda una señal PWM de pulso en alto de 10 microsegundos al módulo ultrasonidos, y posteriormente calcula la distancia y es emitida desde el ultrasonidos a la placa en forma de pulso PWM, en el que según el tiempo que la señal esté en alto corresponde con la distancia a la que se encuentra el objeto, según la ecuación 3.4

$$distacia = (34300 * duracion\_eco(segundos))/2 \tag{3.4}$$

Teniendo la distancia de dos muestras consecutivas y el ángulo de giro del azimut podemos averiguar de la distancia recorrida recurriendo al teorema de pitágoras generalizado, un ejemplo en la figura 3.17, haciendo uso de la ecuación del teorema del coseno, explicado en la ecuación 3.5

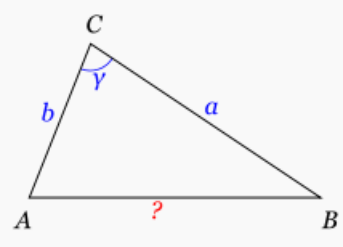


Figura 3.17: Ejemplo de triángulo

$$c = sqrt(a^2 + b^2 - 2abcos(\gamma)) \tag{3.5}$$

Tras tener la distancia recorrida podemos calcular la velocidad a la que se ha movido el objeto, ya que podemos controlar el tiempo que tarda en tomar cada muestra, tan solo habría que dividir la distancia recorrida entre el tiempo entre muestras consecutivas.

Hay que tener en cuenta que la señal PWM que emite el sensor de ultrasonido funciona a 5V, y los GPIO de la placa Raspberry Pi admiten voltajes máximos de 3.3V, por lo que se ha realizado un divisor de tensión para poder guardar la seguridad de voltaje de nuestro sistema. Esto viene indicado en nuestro esquema electrónico en el Anexo E.

Para mostrar la velocidad a la que está el objeto hemos utilizado una pantalla LCD de las que tenían los móviles de la marca Nokia, que se encontraban en los modelos 5510 y 3310. Como la que se muestra en la figura 3.18.

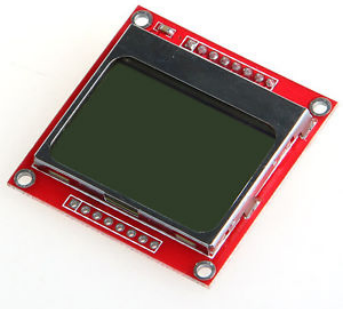


Figura 3.18: Pantalla Nokia 5510.

Para utilizar dicha pantalla nos ayudamos de la librería 'Adafruit Nokia LCD', que nos facilitará la manera de sacar por pantalla nuestras características más importantes, y de 'Adafruit GPIO SPI', que nos facilitará el manejo de los puertos SPI de los GPIO de la placa Raspberry Pi.

Para poder realizar correctamente las conexiones y quedar todo bien colocado y organizado, se ha diseñado e impreso en 3D una plataforma para colocar los servomotores, el soporte para el láser en el servomotor, la cámara y la Raspberry Pi.

En la imagen 3.19 se muestra la plataforma donde se van a ir colocando los diversos elementos hardware. Ésta dispone de encajes para conectar directamente el servomotor y otros encajes para ser colocadas correctamente las paredes de éstas, que se muestran en la figura. 3.20. Encima de éstas paredes se va a colocar una plataforma para la cámara, que se expone en la figura 3.21.

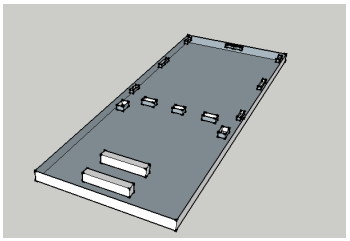


Figura 3.19: Plataforma

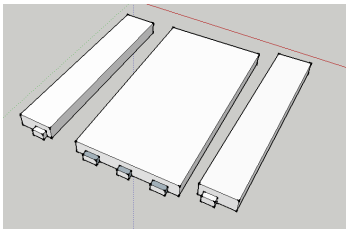


Figura 3.20: Paredes de la plataforma

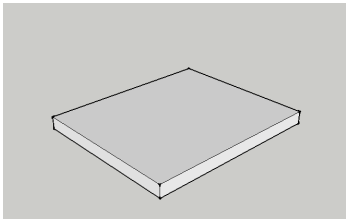


Figura 3.21: Soporte de la cámara

Para colocar el colocar el servomotor que servirá para mover en el eje 'y', hemos diseñado una plataforma para colocarla encima del servomotor que nos moverá en el eje 'x', que será el que se nos muestra en la figura 3.22

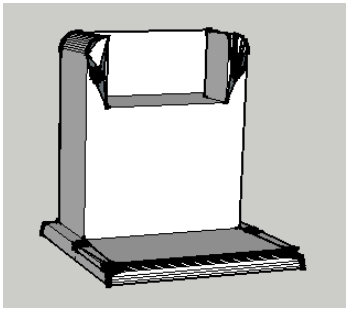


Figura 3.22: Encajes de servomotores

Posteriormente para conectar el servomotor del eje 'y' con el láser se ha diseñado otra plataforma, que se expone en la figura 3.23

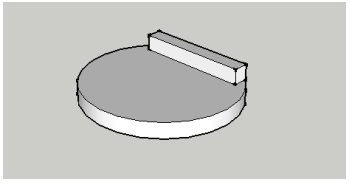


Figura 3.23: Soporte para el láser

Para quedar encajada la placa Raspberry Pi con la plataforma se ha diseñado unas tapas laterales con los agujeros para las distintas salidas, como se muestra en la figura 3.24

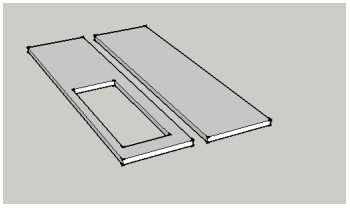


Figura 3.24: Tapas laterales para Raspberryy Pi

Por último, se ha desarrollado unas tapas con con agujeros para controlar la placa Raspberry Pi mediante los botones de los que hemos hablado anteriormente, en los cuales también se han diseñado para su impresión 3D. En la figura 3.25 se muestra la caja, y en la figura 3.26 los botones para la caja.

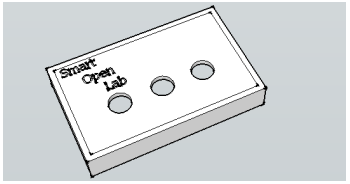


Figura 3.25: Caja de la interfaz

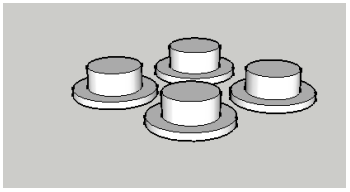


Figura 3.26: Botones para la interfaz

El diseño final queda como se muestra en la figura 3.27

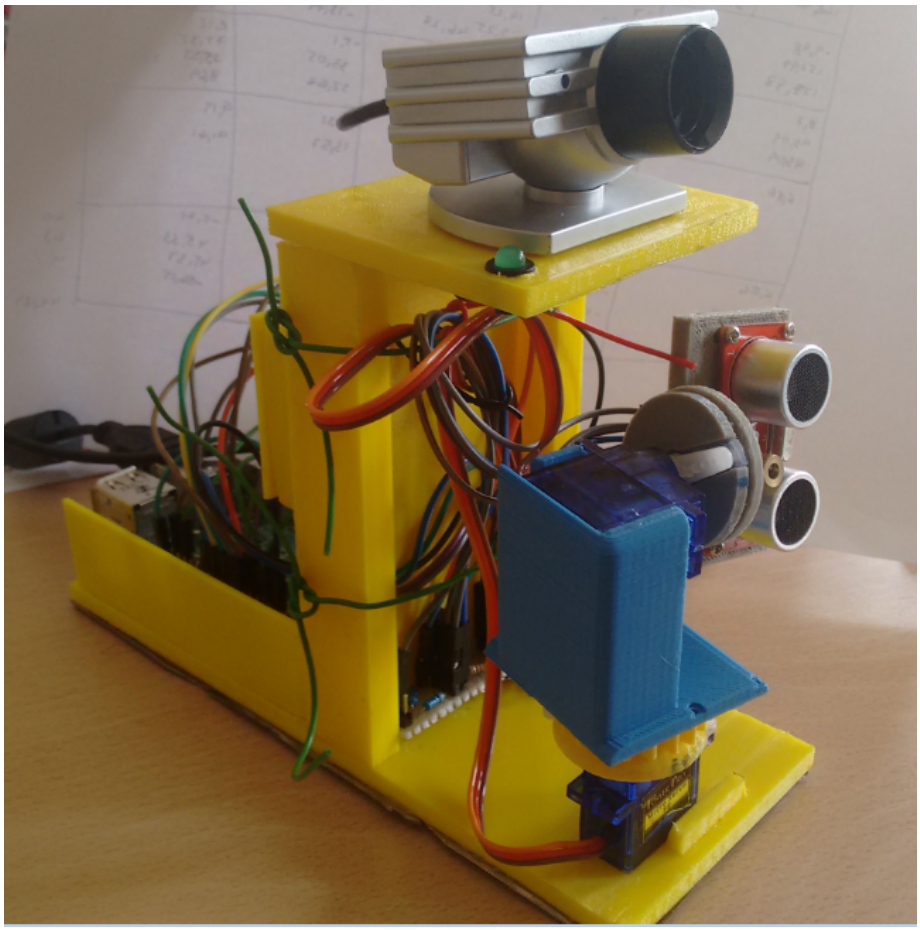


Figura 3.27: Prototitpo final del sistema

Capítulo 4

Resultados

Vamos a describir los resultados obtenidos tras la realización del proyecto. Para ello vamos a empezar por el proceso software tratado en el apartado 3.1.

Vamos a recordar el diagrama del desarrollo del sistema, representado en la figura 3.5, que consistía en captar fondo e imagen actual, realizar la diferencia absoluta de dichas imágenes, transformar a escala de grises, realizar un filtro gaussiano de suavizado, binarizar la imagen resultante, detectar los contornos, calcular los momentos de la imagen y identificar el centroide de la imagen en movimiento.

El ejemplo para mostrar el resultado se ha obtenido realizando una secuencia de vídeo en la calle y se han extraído las imágenes de el paso de dos objetos por delante de la cámara, como es el de un coche y una persona.

La imagen de fondo obtenida se representa en la figura 4.1

Figura 4.1: Imagen de fondo captada

En la figura 4.2 se muestra un ejemplo de una persona y un coche pasando por delante de la cámara.

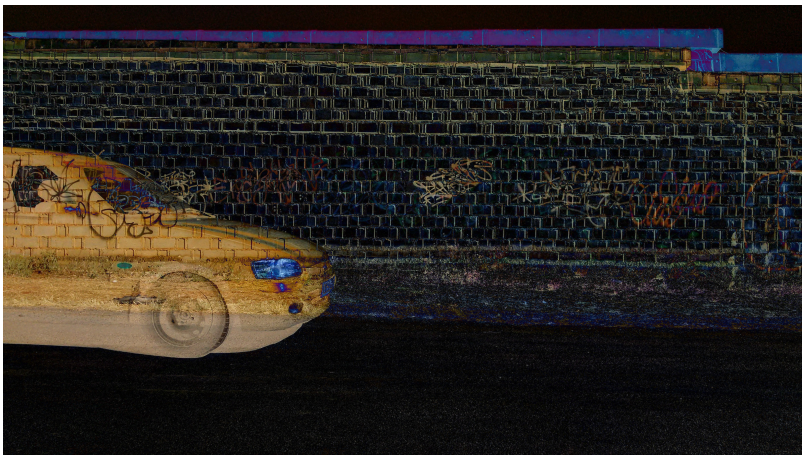
(a) Persona

(b) Coche

Figura 4.2: Imagen de objeto en movimiento

Tras tener las imágenes correspondientes vamos a proceder a realizar la diferencia en valor absoluto de nuestra imagen de la persona y el coche con la imagen del fondo, independientemente. Como se muestra en las figuras 4.3

(a) Persona



(b) Coche

Figura 4.3: Diferencia de los valores RGB en valor absoluto

El proceso siguiente es el de pasar los valores obtenidos a escala de grises, que se muestra en la figura 4.4

(a) Persona

(b) Coche

Figura 4.4: Escala de grises de las imágenes

Seguidamente se realizará un filtrado gaussiano para suavizar la imagen, como se expone en la figura 4.5

(a) Persona

(b) Coche

Figura 4.5: Filtrado gaussiano de la imágenes

Posteriormente pasamos a realizar la umbralización de los píxeles de la imagen, como se muestra en la figura 4.6

(a) Persona

(b) Coche

Figura 4.6: Binarización de la imágenes

De seguido vamos a realizar un proceso de detecćon de contornos, mostrado en la figura 4.7

(a) Persona

(b) Coche

Figura 4.7: Detección de contornos de la imágenes

Posteriormente se calculan los momentos y se averigua el centroide, por lo que nos quedará una imagen final en la que estará la imagen original junto con un círculo indicando el centro del objeto en movimiento. La figura 4.8 muestra el proceso final de todo el desarrollo software, en la que se nos indica el valor de azimut y elevación de donde se encuentra el objeto para poder continuar con el siguiente proceso hardware.

(a) Persona

(b) Coche

Figura 4.8: Muestra final de los objetos en movimiento

Como hemos visto anteriormente, el proceso realizado es satisfactorio para nuestro proyecto. Conseguimos colocar el centroide en el centro de masas del objeto en

movimiento. Hay varias variables que no podemos controlar que son fundamentales en la localización del objeto, como pueden ser las sombras, la iluminación de ciertos sectores, la uniformidad del fondo, distancia del fondo y el objeto de la cámara, etc.

El sistema, aunque se ha implementado bien, va algo lento debido a la cantidad de procesos que tienen que realizarse de manera secuencial. Por cada iteración tarda 0,5 segundos, por lo que el sistema realiza el cálculo de 2 imágenes por segundo.

En cuanto al desarrollo hardware, visto en el apartado 3.1.2, el resultado es, a groso modo, aceptable. Los servomotores no apuntan exactamente al punto donde se encuentra el objeto. Como hemos visto en las figuras 3.12 y 3.13, en el eje 'x' el láser coincide en el mismo lugar que la cámara, sin embargo, en el eje 'y', la posición no el mismo punto que el de la cámara, por lo tanto el ángulo de visión es distinto, y por ello en el eje 'y' es difícil colocar el punto en su sitio correcto.

La obtención de la velocidad, mediante el sensor ultrasonidos y tras un proceso de cálculo mediante trigonometría, calcula bien los parámetros, pero debido a que el sistema es lento el objeto se puede mover más, o menos, de lo que es la realidad. Además, como el ángulo de captación del sensor de ultrasonido es de 30º puede captar más objetos, o el fondo, puede calcular datos erróneos.

Capítulo 5

Conclusiones y trabajos futuros

Tras haber mostrado los resultados y comprobado su funcionamiento, llegamos a la conclusión de que el sistema, a pesar de tener algunos defectos, funciona bastante bien. El sistema capta bien el movimiento del objeto, por lo que el procesamiento de la imagen se hacer correctamente, sin embargo al apuntar el láser mediante el movimiento de los servos se hace algo lento. Una posible solución es cambiar la placa de desarrollo, y/o paralelizar los procesos.

Se ha conseguido realizar unas mejoras que hace que el sistema funcione mejor, como puede ser el led que avisa cuando está captando la imagen del fondo para así tener cuidado de que no haya ningún objeto que no sea estático delante de la cámara durante ese tiempo de parpadeo.

Se ha conseguido realizar también una interfaz de tres botones que consigue manipular la placa de desarrollo correctamente sin necesidad de intervenir en la interfaz gráfica utilizando diversos periféricos propios de un computador.

Mediante el sensor de ultrasonidos hemos conseguido calcular la distancia que hay al objeto en movimiento y mostrarla por la pantalla LCD, al igual que la velocidad instantánea del objeto en ese momento.

5.1. Limitaciones

Este sistema tiene la limitación de la variabilidad gradual de la iluminación, ya que dependiendo de lo rápido que cambie y de a qué parte de la imagen afecte, nos puede molestar a la hora de realizar los cálculos.

Otra variable que nos puede molestar mucho a la hora del cálculo son las sombras de ciertos objetos o el reflejo de cristales, ya que nos pueden dar falsos positivos al creer el sistema que puede haber algún objeto en movimiento.

El fondo es muy importante, ya que si al terminar de parpadear el diodo led que hemos implementado, hay algún objeto en movimiento, todo el proceso del cálculo va a tener ese objeto de referencia y estaríamos haciéndolo mal. El fondo debería de ser homogéneo y posiblemente liso, ya que puede dar falsos positivos con algunos contornos, como nos podría haber pasado en el ejemplo que hemos mostrado en el apartado 4, que tenemos un fondo que no es liso y que contiene pintadas en la pared, por lo tanto tampoco sería homogéneo.

Debido a la cantidad de datos que tiene que tratar la placa Raspberry Pi hace que vaya muy lento el procesamiento de datos, a dos o tres imágenes por segundo. Es por ello que es muy difícil de captar movimientos rápidos.

Los servomotores utilizados son de baja calidad, que aunque dan una respuesta útil para un proyecto de nuestra altura, si se necesitara mayor precisión se deberían utilizar otros de mayor calidad. Es por ello que no siempre apuntan al sitio que se desea. Éstos funcionan a través de modulación PWM, que es una modulación con poca inmunidad al ruido externo, lo que hace que una pequeña interferencia pueda modificar nuestra posición requerida.

El sensor ultrasonidos tiene un ángulo de captación grande, por lo que no es muy directivo, y los datos que proporciona pueden variar. El sensor también se comunica con la placa mediante PWM, y como hemos comentado anteriormente, éste sistema es débil en cuanto a resistencia de ruido externo.

5.2. Trabajos futuros

El sistema podría no acabar aquí y seguir estudiándose futuras mejoras que se le pueden implementar. Por ejemplo, para poder tratar los datos de una manera más óptima y con mayor calidad, se podría realizar un cálculo paralelo utilizando al máximo los 4 núcleos que posee el microprocesador. Así podría ir más rápido y poder realizar más procesos por segundo que los que hace actualmente. Otra posibilidad está también en adquirir un microprocesador más potente que el utilizado

Para solventar el problema de las sombras en la imagen se han desarrollado últimamente algunos algoritmos que pueden hacer desaparecer estas sombras para que no afecten a nuestro sistema, pero computacionalmente sale bastante caro.

Otra mejora consistiría en cambiar los servomotores por otros de más calidad y que tengan una precisión mayor a la actual.

Dependiendo la aplicación que se le vaya a dar al sistema convendría mejorar el sistema de captación de la distancia del sistema al objeto.

5.3. Aplicaciones

Este sistema puede resultar útil en varias aplicaciones como pueden ser:

- Videovigilancia, ya que puedes seguir el movimiento de cualquier objeto que se muestre por medio del sistema de captación.
- Radares para tráfico, ya que se puede calcular la velocidad a través del movimiento del objeto por la pantalla.
- Cálculo de personas o automóviles, en los que se puede discriminar a través de modelos previos.

Capítulo 6

Agradecimientos

Quisiera agradecer este trabajo fin de estudios a Antonio Gordillo por haber sido mi tutor durante todo este año, y todas la complicaciones que han ido surgiendo, con el que aprendido mucho en Smart Open Lab.

Agradecer a Pedro Núñez por coger este trabajo fin de estudios como co-tutor y enseñarme muchas cosas en RoboLab.

Dar las gracias a toda mi familia por todo el apoyo incondicional que ha mostrado durante estos cuatro años y sus intentos de motivación cuando había tanta fustración.

Agradecer a todos mis amigos y compañeros de clases que de una manera u otra me han ayudado a superarme cada día y conseguir que hoy haya llegado hasta aquí.

Bibliografía

[1] Intel, "Opencv." http://opencv.org/, 1999.

[2] A. F. Granados and J. I. M. H., "Detección de flujo vehicular basado en visión artificial," *Scientia et Technica. Universidad Tecnológica de Pereira*, vol. 20, pp. 163–168, Agosto 2007.

[3] G. A. Baxes, *Digital Image Processing: Principles and applications.* 1994.

[4] R. C. Gonzalez and R. E. Woods, *Digital Image Proccessing.* 1992.

[5] P. V. K. S. Bhavna B. Khajone, Rohan B. Kokate, "A survey of image registration techniques," *IJRIT International Journal of Research in Information Technology*, vol. 2, pp. 554 – 560, Aril 2014.

[6] M. F. Zanuy, *Tratamiento digital de voz e imagen y aplicación a la multimedia.* 2000.

[7] W. A. U. Marta Lucía Guevara, Julian David Echeverry, "Detección de rostros en imágenes digitales usando clasificadores en cascada," *Scientia et Technica, Universidad Tecnológica de Pereira*, vol. 1, pp. 1 – 5, Junio 2008.

[8] J. P. Portillo, *Detección de Movimientos de Objetos mediante secuencias de vídeo.* PhD thesis, Escuela Superior de Ingeniería Mecánica y Eléctrica Unidad Zacatenco, Mayo 2012.

[9] O. Javed and M. Shah, "Tracking and object classification for automated surveillance," *7th European Conference on Computer Vision Copenhagen, Denmark*, vol. 2353, pp. 343–357, Mayo 2002.

[10] M. J. J. J. J.F.Vazquez, C.A.Luna *Seminario Anual de Automática, Electrónica Industrial e Instrumentación.*

[11] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, p. 146–165, Enero 2004.

[12] R. P. Fundation, "Raspberry pi." https://www.raspberrypi.org/, 2011.

[13] P. S. Foundation, "Python." https://www.python.org/, 1991.

[14] J. Hugunin, "Numpy." http://www.numpy.org/, 2005.

[15] I. P. Adrian G. Bors, "Prediction and tracking of moving objects in image sequences," *IEEE Transactions on Image Processing*, vol. 9, pp. 1441 – 1445, Agosto 2000.

[16] H. L. Paredes, *Detección y seguimiento de Objetos con cámaras en movimiento.* PhD thesis, Escuela Politécnica Superior de la Universidad Autómata de Madrid, Septiembre 2011.

[17] A. I. R. A. C. H. E. M. B. Carlos A. Cattaneo, Ledda I. Larcher, "Métodos de umbralización de imágenes digitales basados en entriopía de shannon y otros," *Asociación Argentina de Mecánica Computacional*, vol. 30, pp. 2785–2805, Noviembre 2011.

[18] R. P. Foundation, "Documentation gpio." https://www.raspberrypi.org/documentation/usage/
2011.

[19] TowerPro, "Microservo towerpro sg90 de 9g." http://www.micropik.com/PDF/SG90Servo.pdf.

[20] Keyes, "Laser keyes ky-008." http://en.keyes-robot.com/productshow.aspx?id=166.

[21] S. S. Works, "Seeed ultrasonic sensor." http://www.seeedstudio.com/depot/datasheet/Seeed

Apéndice A

Anexo A

```
>> #!/usr/bin/python
>> # − ∗− coding: utf−8 − ∗−

>>def movx(x):      #Movimiento servomotor eje 'x'
>>    g=0
>>    p=GPIO.PWM(27,50)
>>    if x<320:
>>        x=float(x)
>>        g=x/320
>>        g=g*45
>>        poc= ((10 − 7,5) ∗ g)/45
>>        porc=poc+7.5

>>        p.start(porc)
>>        sleep(0.1)
>>        p.stop()

>>    elif x == 320: #Punto central

>>        p.start(7.5)
>>        sleep(0.1)
```

```
>>        p.stop()

>>    elif x >320:
>>        x=float(x)
>>        x=x-320
>>        g=((x/320)) * 45
>>        poc=((7,5 - 5) * g)/45
>>        porc=7.5-poc
 >>        p.start(porc)
>>        sleep(0.1)
>>        p.stop()

>>    poc=0
>>    porc=0
>>    return g

>>def movy(y):#Movimiento servomotor eje 'y'

>>    q=GPIO.PWM(22,50)

>>    if y<240:
>>        y=float(y)
>>        g=y/240
>>        g=g*35
>>        poc=((10 - 7,5) * g)/35
>>        porc=poc+7.5

>>        q.start(porc)
>>        sleep(0.1)
>>        q.stop()

>>    elif y == 240: #Punto central
```

```
>>        q.start(7.5)
>>        sleep(0.1)
>>        q.stop()

>>    elif y >240:
>>        y=float(y)
>>        y=y-240
>>        g=((y/240)) * 35
>>        poc=((7,5 - 5) * g)/45
>>        porc=7.5-poc

>>        q.start(porc)
>>        sleep(0.1)
>>        q.stop()

>>    g=0
>>    poc=0
>>    porc=0

>>def ultra(): #Detectar distancia al objeto (Ultrasonidos)

>> #    Mandar un pulso de 10 useg
>>    GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
>>    GPIO.output(GPIO_TRIGGER, GPIO.HIGH)
>>    sleep(0.00001)
>>    GPIO.output(GPIO_TRIGGER, GPIO.LOW)

>>    #Esperar a detectar Eco. Pulso en alto.
>>    while True:
>>        GPIO.setup(GPIO_ECHO, GPIO.IN)
```

```
>>      pulso_inicio = time()
>>      if GPIO.input(GPIO_ECHO) == GPIO.HIGH:
>>          break

>>  #Esperar a acabar Eco. Pulso en bajo.
>>  while True:
>>      pulso_fin = time()
>>      if GPIO.input(GPIO_ECHO) == GPIO.LOW:
>>          break

>>  # La duración del eco se mide en segundos.
>>  duracion_eco = pulso_fin - pulso_inicio

>>  #Cálculo de la distancia
>>  distancia = (34300 * duracion_eco)/2

>>  return distancia

>>def salida(texto,d): #Sacar texto por pantalla LCD NOKIA

>>  import Adafruit_Nokia_LCD as LCD
>>  import Adafruit_GPIO.SPI as SPI

>>  from PIL import Image
>>  from PIL import ImageDraw
>>  from PIL import ImageFont

>>  # Raspberry Pi hardware SPI config:
>>  DC = 23
>>  RST = 24
>>  SPI_PORT = 0
>>  SPI_DEVICE = 0
```

```
>>  texto=str(texto) #Convertir variable en string para sacar por pantalla
>>  d=str(d)

>>  disp = LCD.PCD8544(DC, RST, spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE))

>>  disp.begin(contrast=60)

>>  # Limpiar pantalla.
>>  disp.clear()
>>  disp.display()

>>  # Crear imagen en blanco para dibujar.

>>  image = Image.new('1', (LCD.LCDWIDTH, LCD.LCDHEIGHT))

>>  draw = ImageDraw.Draw(image)

>>  # Dibujar cuadro en blanco para la imagen
>>  draw.rectangle((0,0,LCD.LCDWIDTH,LCD.LCDHEIGHT), outline=255,
fill=255)

>>  font = ImageFont.load_default()

>>  # Escribir el texto.
>>  draw.text((1,1), 'Velocidad:', font=font) #Indica el corner de arriba a la
izquierda
>>  draw.text((1,10), texto, font=font)
>>  draw.text((30,10), 'cm/s', font=font)

>>  draw.text((1,20), 'Distancia:', font=font) #Indica el corner de arriba a la
izquierda
```

```
>>    draw.text((1,30), d, font=font)
>>    draw.text((30,30), 'cm', font=font)

>>    # Mostrar imagen
>>    disp.image(image)
>>    disp.display()

>> #Cargar las librerías
>>from cv2 import VideoCapture, absdiff, cvtColor, GaussianBlur, threshold,
Canny, findContours, moments, circle, imshow, COLOR_RGB2GRAY, THRESH_BINARY

>>from math import sqrt, cos

>>import numpy as np

>>import RPi.GPIO as GPIO

>>from time import sleep, time #sleep(0.5) # Time in seconds.

>> #Declaración de variables

>>laser=26
>>salir=19
>>luz=21

>>video_capture = VideoCapture(0)
>>prim=0
>>enc=0
>>distancia1=0
>>inicio=0

>> #Declaración de los pines
```

```
>>GPIO_TRIGGER = 7 #Mismo pin para mandar y recibir
>>GPIO_ECHO = 7

>>GPIO.setmode(GPIO.BCM)
>>GPIO.setup(laser,GPIO.OUT)
>>GPIO.setup(luz,GPIO.OUT)

>>GPIO.setup(27,GPIO.OUT)
>>GPIO.setup(22,GPIO.OUT)

>>GPIO.setup(salir,GPIO.IN)

>>sleep(0.5)

>> #Captura de imagen de fondo
>>while prim<7:
>>    ret,img_fija=video_capture.read()
>>    if enc==0:
>>        GPIO.output(luz,GPIO.HIGH)
>>        enc=1
>>    else:
>>        GPIO.output(luz,GPIO.LOW)
>>        enc=0
>>    sleep(0.5)
>>prim=prim+1

>>height = np.size(img_fija, 0)
 >>width = np.size(img_fija, 1)

>>while(1):
>>    #Captura de imagen actual
```

```
>>  ret, img = video_capture.read()

>>  cx=width/2 #Punto medio de la imagen  >>  cy=height/2

>>  #Diferencia de imagen de fondo e imagen actual
>>  img_dif=absdiff(img,img_fija)

>>  #Conversión a escala de grises
>>  img_gray=cvtColor(img_dif, COLOR_RGB2GRAY)

>>  #Filtro gaussiano
>>  img_gray=GaussianBlur(img_gray,(5,5),0)

>>  #Umbralización
>>  ret,img_bin = threshold(img_gray,127,255,THRESH_BINARY)

>>  #Detección de contornos
>>  edges = Canny(img_bin,100,200)

>>  cont=img_bin

>>  contours,hierarchy = findContours(cont, 1, 2)

>>  long=np.size(contours)

>>  #Momentos
>>  M = moments(edges)

>>  if(long > 0):#Encuentra movimiento
>>      GPIO.output(laser,GPIO.HIGH) #Activar Láser

>>      #Evitar las divisiones con ceros
```

```
>>      if M['m00']==0:
>>          M['m00']=0.01

>>      if M['m10']==0:
>>          M['m10']=0.01

>>      if M['m01']==0:
>>          M['m01']=0.01

>>      #Centro del objeto en movimiento
>>      cx = int(M['m10']/M['m00'])
>>      cy = int(M['m01']/M['m00'])

>>      center = (cx,cy)
>>      radius =int( width*0.05)

>>      #Crear centroide
>>      circle(img,center,radius,(0,255,0),2)

>>      #Controlar el tiempo por cada detección de movimiento
>>      final=time()

>>      tiempo=(final-inicio)

>>      inicio=time()
>>      velocidad=1 #Activar para calcular velocidad

>>  if (long == 0 ):
>>      GPIO.output(laser,GPIO.LOW)
>>      velocidad=0 #Desactivar para calcular velocidad

>>  gx=movx(cx) #Mover servomotor en eje 'x'
```

```
>>    movy(cx) #Mover servomotor en eje 'y'

>>    if velocidad==1: #Calcular velocidad
>>        distancia=ultra() #Obtener distancia recorrida del objeto
>>        #Calcular velocidad
>>        vel=(sqrt(abs((distancia1 * *2) + (distancia * *2) - 2 * distancia1 *
distancia * cos(gx))))/tiempo * 10

>>        distancia1=distancia
>>    else:
>>        vel=0
>>        distancia=ultra()

>>    vel=int(vel)
>>    distancia=int(distancia)
>>    salida(vel,distancia) #Sacar por pantalla

>>    if GPIO.input(19)==1: #Botón de detención
>>        break

>>GPIO.output(luz,GPIO.LOW)
>>GPIO.cleanup()
```

Apéndice B

Script 'Inicio.py'

Código para iniciar el programa 'Deteccion' mediante la interfaz de la botonera:

```
>> #!/usr/bin/python
>> # -*- coding: utf-8 -*-
>>import os
>>import time
>>import RPi.GPIO as GPIO
>>GPIO.setmode(GPIO.BCM)
>>GPIO.setup(14,GPIO.IN)
>>prev_input = 0
>>while True:
>>      input = GPIO.input(14)
>>      if ((not prev_input) and input):
>>            os.system("sudo python /home/pi/Desktop/Deteccion.py")
>>      prev_input = input
>>      #Pequeña pausa
>>      time.sleep(0.05)
```

Apéndice C

Script 'shutdown.py'

Código para apagar la placa Raspberry Pi de manera segura mediante la interfaz de la botonera:

```
>> #!/usr/bin/python
>> # -*- coding: utf-8 -*-
>>import os
>>import time
>>import RPi.GPIO as GPIO
>>GPIO.setmode(GPIO.BCM)
>>GPIO.setup(4,GPIO.IN)
>>prev_input = 0
>>while True:
>>      input = GPIO.input(4)
>>      if ((not prev_input) and input):
>>            os.system("sudo halt")
>>      prev_input = input
>>      #Pequeña pausa
>>      time.sleep(0.05)
```

Apéndice D

Iniciar Scrpit

Código a escribir en la terminal de Raspberry Pi para poder iniciar los script de manera automática cuando se inicie el Sistema Operativo Raspbian:

```
>> #! /bin/sh
>> # /etc/init.d/scriptinit
>> ### BEGIN INIT INFO
>> # Provides: script-init
>> # Short-Description: Script de ejemplo de arranque automático
>> ### END INIT INFO
>>case '$1' in
>>   start)
>>      echo 'Arrancando script-init'
>>       # Aquí hay que poner el programa que quieras arrancar automáticamente
>>      /usr/bin/python /home/pi/script.py
>>      ;;
>>   stop)
>>      echo 'Deteniendo script-init'
>>      ;;
>>   *)
>>      echo 'Modo de uso: /etc/init.d/script-init start|stop'
```

```
>>      exit 1
>>      ;;
>>esac
>>exit 0
```

Apéndice E

Esquema electrónico

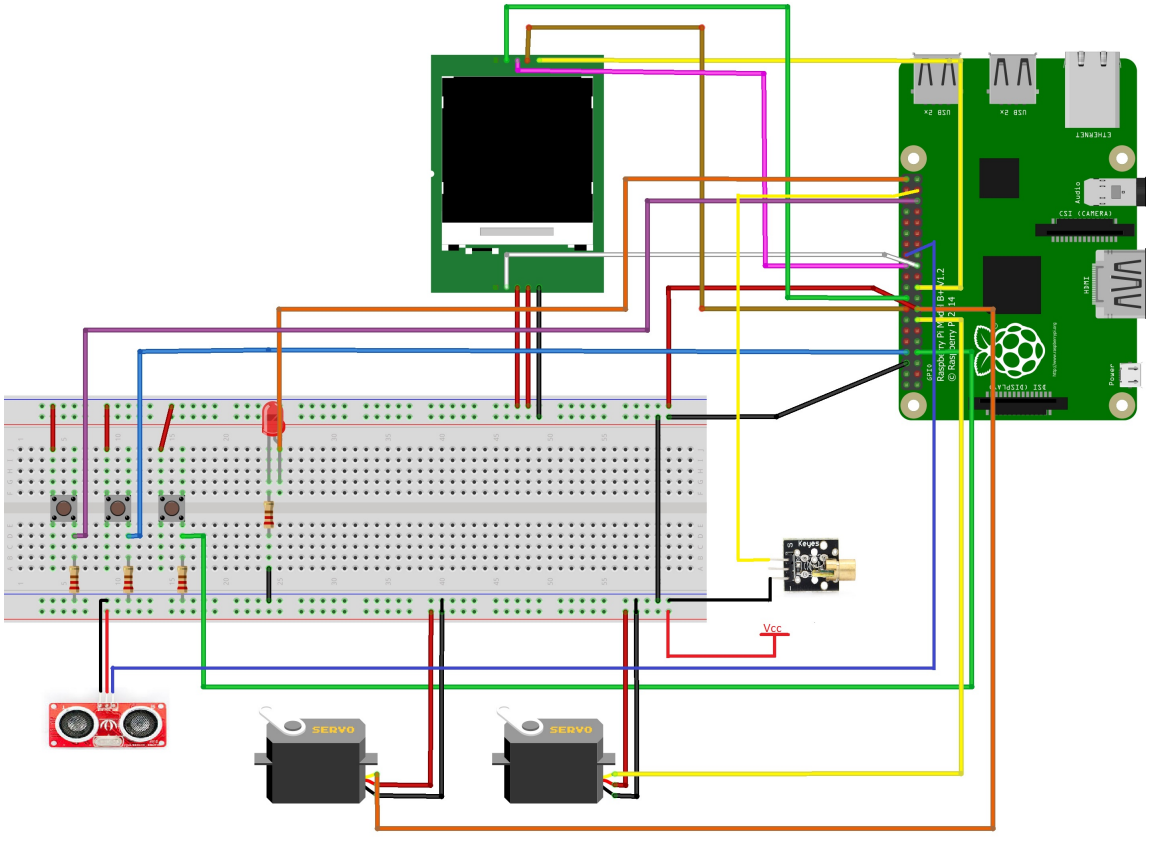


Figura E.1: Esquema electrónico