Implementación de un Sistema que Permita Determinar la Disponibilidad de Puestos en una Zona de Aparcamiento Vehicular.

Erik Fabian Vera Ortiz

Trabajo de Grado para Optar el Título de Ingeniero Electrónico

Director
Jaime Guillermo Barrero Pérez
Maestría en potencia eléctrica

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.
Bucaramanga
2019

Erik Fabian Vera Ortiz

Agradecimientos

A Dios por permitirme llegar a este punto de mi vida.

A todas las personas que hicieron fuera posible la realización de este proyecto, aquellos que estuvieron allí conmigo y me apoyaron de una u otra manera: amigos y familiares.

Agradezco a mi tutor Jaime Guillermo Barrero, por estar atento de mi trabajo y ayudarme en cuanto a las dudas, correcciones y planes respecto el proyecto.

A mi madre Claudia Ortiz F. y hermanos Brayan Arley Vera, Angie T. Vera, Jeiner Luwing Ortiz, quienes son mi orgullo, y estuvieron apoyándome a lo largo de mi carrera universitaria.

Gracias a todos, y a la universidad por permitirme ser parte de esta

Tabla de Contenido

Pág.

Lista de Figuras

Lista de Tablas

Lista de Apéndices

"Ver apéndices adjuntos en el CD y pueden visualizarlos en base de datos de la Biblioteca UIS"

RESUMEN

TITULO: Implementación de un sistema que permita de aparcamiento vehicular.

AUTOR: Erik Fabian Vera Ortiz, kirefabi@gmail.com

PALABRAS CLAVE: Aparcamiento inteligente, servidor web

DESCRIPCIÓN:

Para determinar la disponibilidad de puestos en un parqueadero, se construyó una maqueta a escala y mediante el procesamiento de imágenes se logró identificar el estado (libre/ocupado) de puestos de parqueo. El presente proyecto se dividió en varias etapas que consisten en: capturar la imagen; enviarla a un sistema de procesamiento, en este caso un computador; una vez en el computador, realizar el respectivo procesamiento de la imagen para identificar el estado libre u ocupado de la plaza de aparcamiento vehicular; los datos obtenidos del procesamiento en el ordenador se envían y almacenan en una base de datos; luego, la información de la base de datos se transmite al usuario final en una aplicación web donde el usuario pueda ver el estado del aparcamiento; y por último, validar la información en un parqueadero que se realiza a escala (1:43) de las medidas que cada plaza debería tener un parqueadero real. Se utilizaron cámaras con resoluciones de 0,3 MPx (OV7670) y de 2 MPx (OV2640) para capturar la imagen y con la ayuda de la ESP32 se transmitió por medio del Wi-Fi a un equipo de cómputo para realizar su respectivo procesamiento, el lenguaje de programación utilizado para la identificación de la plaza es Python, donde se hace uso de diferentes librearías, de las cuales openCV es una de las más importantes para el desarrollo de este trabajo, en cuanto a la base de datos, se utilizó mysql y la aplicación web utiliza lenguaje html, php, css y javascript para mostrar los datos.

Trabajo de Grado

Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Jaime Guillermo B. MSc en potencia eléctrica

ABSTRACT

TITLE: Implementation of a system that allows to determine the availability of positions in a vehicular parking area.

AUTHOR: Erik Fabian Vera Ortiz, kirefabi@gmail.com

KEYWORDS: Smart parking, web server.

DESCRIPTION:

To determine the availability of parking spaces, a scale model was built and through image processing it was possible to identify the status (free / busy) of parking spaces.

This project was divided into several stages that consist of: capturing the image; send it to a processing system, in this case a computer; once on the computer, perform the respective image processing to identify the free or busy state of the vehicle parking space; the data obtained from the processing in the computer are sent and stored in a database; then, the information in the database is transmitted to the end user in a web application where the user can see the parking status; and finally, validate the information in a parking lot that is carried out at scale (1:43) of the measures that each place should have a real parking lot. Cameras with resolutions of 0.3 MPx (OV7670) and 2 MPx (OV2640) were used to capture the image and with the help of the ESP32 it was transmitted via Wi-Fi to a computer to perform their respective processing, The programming language used for the identification of the square is Python, where different libraries are used, of which openCV is one of the most important for the development of this work, as for the database, mysql was used and the web application uses html, php, css and javascript language to display the data.

Trabajo de Grado

Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Jaime Guillermo B. MSc en potencia eléctrica.

Introducción

De las recientes tecnologías en *Smart parking* la información relativa a estacionamientos se almacena en una base de datos usando la arquitectura de la nube a través de los sensores utilizados, permitiendo a los clientes obtener información de las zonas disponibles para parqueo. Existen planteamientos de solución donde ubican sensores en la entrada de un aparcamiento y la información es llevada a la nube y mostrada en una aplicación para teléfonos inteligentes (Talha Kiliç, 2017) . Otros métodos permiten localizar el número de plazas y cuáles se encuentran libres u ocupadas, reduciendo al usuario el tiempo dedicado al estacionamiento hasta en de un 40% y el nivel de CO_2 en un 30% (Radimer , 2017).

Algunos establecimientos en su interior tienen un área extensa con varias zonas de parqueo alejadas entre sí, por lo que el usuario se ve obligado a buscar donde parquear dando vueltas al interior del establecimiento sin la certeza de saber si el lugar al que se dirige tenga puestos de estacionamiento libres. Consecuentemente, en este trabajo se implementó un sistema que permita mitigar el problema que se evidencia en esta sección, permitiendo al usuario identificar el lugar donde ubicarse en cualquiera de los espacios libres de estacionamiento dentro del establecimiento, zona o campus en el cual se encuentra.

El estímulo principal para realizar este trabajo es poder implementar un sistema de bajo costo que permita mediante el procesamiento de imágenes identificar el estado(libre/ocupado) de puestos de parqueo en un establecimiento.

Para llevar este proyecto a cabo se realizó en una maqueta el parqueadero y se estableció la comunicación entre el usuario y una aplicación web donde se indica el estado de los puestos del parqueadero.

El principal interés es poder reconocer sí un vehículo se encuentra ocupando una plaza de la zona de aparcamiento. El sistema de procesamiento debe ser confiable de manera que permita discriminar objetos pequeños tales como piedras, hojas e incluso personas que pasan por el lugar.

El prototipo cuenta con ciertas restricciones y condiciones las cuales se mencionan a continuación:

Funcionalidad: El producto debe indicar de forma simple, cuando un puesto en una zona de aparcamiento se encuentra libre u ocupado mostrando las señales en color rojo y verde respectivamente. La cámara se encuentra en una posición fija y en la captura de la imagen se debe mostrar la cantidad de espacios de parqueo para realizar el proceso de reconocimiento.

Ensamble del producto: La cámara debe ubicarse a una determinada altura donde se pueda ver la zona de aparcamiento de forma clara, el diseño se llevará a modo maqueta donde se va a recrear el escenario de una zona de aparcamiento vehicular.

1. Objetivos

Objetivo General

Implementar un prototipo de un sistema que permita determinar la disponibilidad, cantidad y ubicación de puestos en una zona de aparcamiento vehicular.

Objetivos Específicos

El cumplimiento del objetivo general del trabajo de grado comprende:

Utilizar una cámara para realizar la captura de imagen de la zona de aparcamiento y enviarla a un sistema digital.

Utilizar un sistema digital que permita recibir información de una imagen capturada por la cámara y enviarla de manera inalámbrica a un computador central para su procesamiento.

Elaborar en el equipo central de procesamiento los algoritmos necesarios para obtener mediante el procesamiento de imágenes las coordenadas de los sitios disponibles para parquear un vehículo en un área previamente determinada.

Desarrollar una aplicación web que muestre el mapa de los lugares que se encuentren libres en una zona de aparcamiento de acuerdo con los resultados obtenidos en el proceso de reconocimiento permitiendo al usuario visualizar los puestos o zonas de parqueo.

Realizar un montaje en maqueta donde se pueda simular una zona de aparcamiento y de esta manera poder visualizar el funcionamiento del sistema prototipo que se ha desarrollado.

2. Características de la maqueta del parqueadero.

y a una altura fija se ubicó un ESP32 con su cámara, ver Figura 1 . Con esta estructura se simuló el ambiente de trabajo colocando y quitando objetos (personas, vehículos de diferente color, etc) bajo diferentes condiciones de iluminación. Las principales características que representa esta maqueta son: Área total = $40x40m^2$, 4 zonas de parqueo cada una de ellas con un área de $3x6\ m^2$, Las dimensiones del parqueadero se diseñaron para vehículos automotores de uso particular o público los cuales se encuentran en la clasificación de línea liviana por la norma técnica colombiana (ICONTEC, 2006).

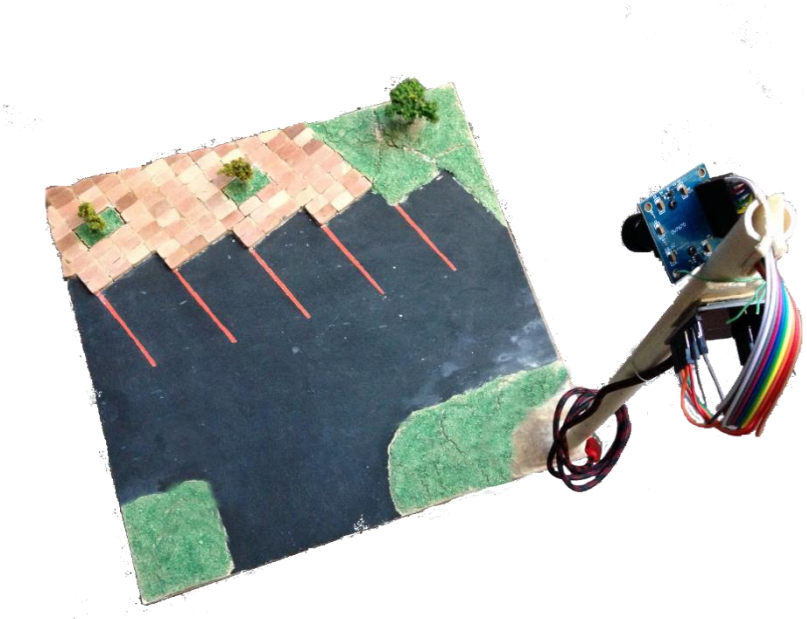


Figura 1. Aparcamiento vehicular a escala (1:43).

Para las pruebas se utilizaron dos cámaras con diferentes características. La cámara OV7670 con resolución de hasta 0,3 Mpx con ángulo de visión cercano a los 25° fue la primera que se utilizó para el desarrollo de este proyecto por su bajo costo y disponibilidad en el mercado local. Sin

embargo, a principios del 2019, empezaron a aparecer a nivel mundial, sistemas de desarrollo que integran en una misma tarjeta el ESP32 y su cámara. Una de estas opciones utiliza la cámara OV2640 de 2 Mpx con una lente que permite un ángulo de visión cercana a los 180°, esta opción se utilizó al final de este proyecto para validar el software desarrollado. El ángulo de visión está relacionado con la altura a la que se debe ubicar la cámara, entre más amplio sea el ángulo, mayor espacio abarca en la captura de la imagen y por lo tanto se necesita una menor altura para ubicar la cámara.

Áreas de objetos al interior del parqueadero.

Para calcular el área del objeto en una imagen, se necesita definir una proporción que relacione el número de pixeles con una métrica determinada, esta operación se conoce como pixeles por métrica. Para realizar las mediciones se utiliza un objeto de referencia con área conocida, se ubica en la zona de interés y se realiza la siguiente relación:

Pixeles por métrica = ancho de objeto (Pixeles) / ancho conocido (Metros).

En la Figura 2(a) se pueden observar 6 objetos cuadrados distribuidos a lo largo del parqueadero con el objetivo de encontrar las variaciones en el área de cada uno de los elementos. En la Figura 2(b) se encuentran estos mismos elementos etiquetados para poder hallar el área y encontrar la relación de píxel por métrica.

(a) (b)

Figura 2. En (a) se pueden ver los objetos a calcular la relación px/m. En (b) se pueden ver los mismos objetos etiquetados. La toma se realiza con la cámara OV7670, con resolución de $320x240\ px$.

Tabla 1.Relación píxel por métrica de objetos capturados con cámara OV7670.

Nombre de etiqueta	Dimensión en m^2	Píxel por métrica.
Object#1	$1x1\ m^2$	$14\ px/m$
Object#2	$2x2\ m^2$	$15.5\ px/m$
Object#3	$1x1\ m^2$	$16\ px/m$
Object#4	$1x1\ m^2$	$17\ px/m$
Object#5	$1x1\ m^2$	$18\ px/m$
Object#6	$2x2\ m^2$	$17\ px/m$

Como se puede observar en la

Tabla *1* la relación px/m en cada uno de los objetos que se encuentran distribuidos a lo largo del aparcamiento depende de sus coordenadas. Los objetos más cercanos a la cámara (objetos 5 y 6) tienen una mayor área en relación con los demás. Con base en esta información, se calculó un

factor de corrección, y cada vez que se realiza el reconocimiento de un objeto, su área será corregida en función de su posición, cabe resaltar que la cámara se ubica a una altura fija para que el factor de corrección k tenga validez.

(a) (b)

Figura 3. En (a) se pueden ver los objetos a calcular el área. En (b) se pueden ver los mismos objetos etiquetados. La toma se realiza con la cámara OV7670, con resolución de $320x240\ px$ a una altura fija.

La Tabla 2. muestra la cantidad de pixeles en cada uno de los rectángulos etiquetados en la Figura 3. Como se pudo ver en la

Tabla 1 el área tendrá un valor diferente cada vez que el objeto se encuentre más distante del otro.

Tabla 2. Cantidad de pixeles en el área de cada elemento.

Nombre de etiqueta	Dimensión en m^2	Píxeles en el área.	Factor de corrección k
Object#1	$5.3x\ 8.4\ m^2$	$3624\ px$	1.372
Object#2	$5.3x\ 8.4\ m^2$	$4236\ px$	1.173

Object#3	$5.3x\ 8.4\ m^2$	$4920\ px$	1.01
Object#4	$5.3x\ 8.4\ m^2$	$4972\ px$	1

$Object\#4 = k(Object\#i)\ con\ i = 1,2,3$

De los valores de la Tabla 2., la máxima diferencia se encuentra entre el objeto 1 y el objeto 2, esta diferencia es de $1348\ px$, para poder estimar el área a eliminar, se ha planteado realizar una corrección a partir de los valores encontrados realizando una conversión que permita mantener una misma área en cada objeto que se encuentra distribuido.

Se tomó como área de referencia el objeto#4 para corregir el resto de los elementos, para ello, todos los objetos deben mantener la misma relación de área por lo tanto, existe un factor k tal que al multiplicar por cada objeto se obtiene el área del objeto de referencia.

$Object\#4 = k(Object\#i)\ con\ i = 1,2,3$

Para el desarrollo del proyecto se realizó una consulta sobre los vehículos más vendidos en Colombia, de los cuales destacan el KIA Rio, el Renault sandero, el chévrolet tracker, entre otros (motor, 2017) de los cuales se eligió el tamaños de los carros que circulan en Colombia y se encontraron que en promedio estos son de alrededor de $4x1{,}8\ m^2$, la Figura 4 muestra las medidas de un vehículo promedio y un vehículo eléctrico pequeño.

Vehículo promedio Vehículo eléctrico pequeño

Figura 4. Dimensiones de un vehículo promedio vendido en Colombia y un vehículo eléctrico pequeño. Recuperado de www.medidasdecoches.com

3. Elementos de Hardware para transmitir la imagen por medio de Wi-Fi

La Figura 5, muestra los elementos utilizados en este proyecto, donde un ESP32 con cámara, captura la imagen y la envía directamente al equipo central de procesamiento conectado a la red (computador). En este proyecto, para no depender de ninguna otra red, se configuro el ESP32 para trabajar como “Access point”(La ESP32 puede trabajar como Wi-Fi Station y Access Point(AP)).

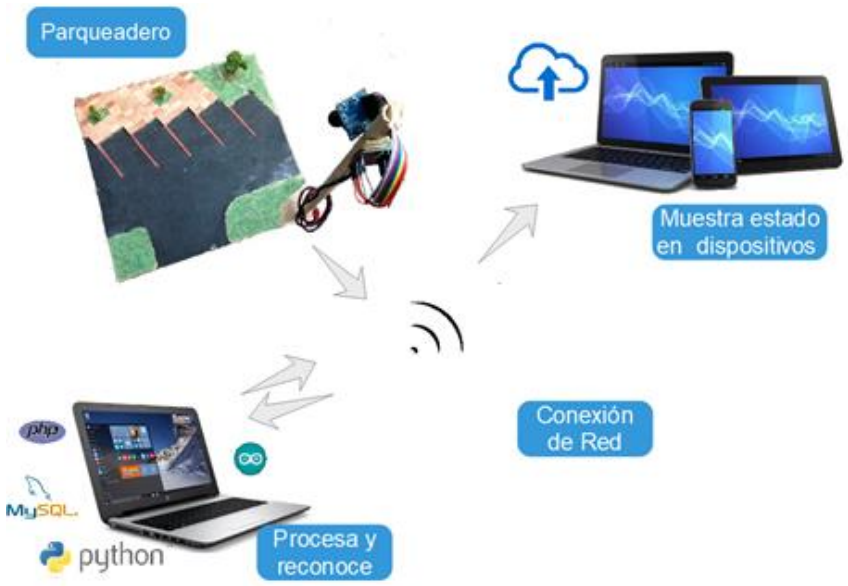


*Figura 5.*Esquema general del proyecto

Como sistema de sensor, se utilizó un módulo de cámara VGA OV7670 cuya máxima resolución posible es de 640x480 (Omnivision, 2005), La cámara se conectó al ESP32, el cual fué el encargado de capturar la imagen y enviarla por medio del Wi-Fi a un equipo de cómputo donde esta se almacena, se adecúa y se le realiza su respectivo procesamiento (Espressif, 2018).

Para enviar la imagen, se utilizó la IDE de Arduino y se hizo uso del repositorio ESP32CameraI2S de bitluni que se encuentra en github. (bitluni, 2017).

En el laboratorio se realizó el montaje del parqueadero a pequeña escala donde se instaló en un poste la cámara OV7670 junto con la tarjeta ESP32 encargada de enviar los datos de imagen. La Figura 6 muestra la tarjeta y cámara utilizadas.

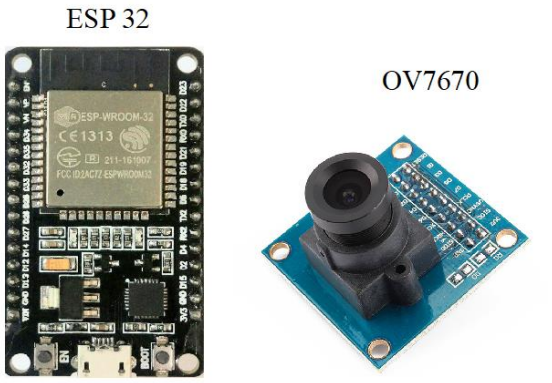


Figura 6. Elementos de hardware para la adquisición y transmisión de imagen.

La conexión entre el ESP32 y la cámara OV7670 se muestra en la Figura 7 .

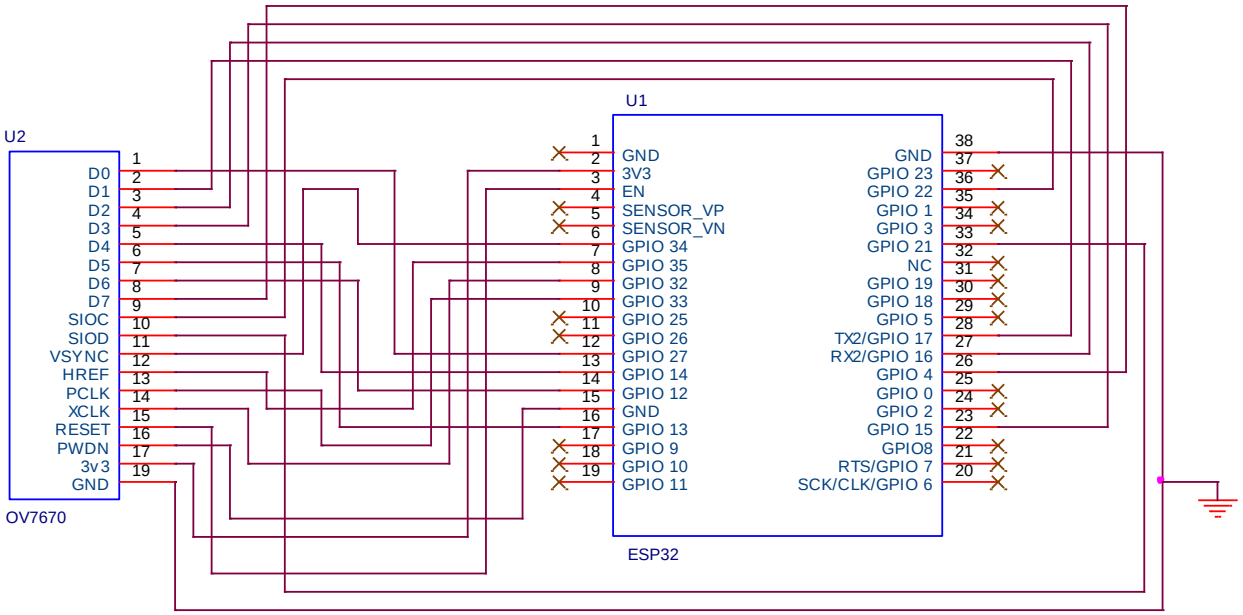


Figura 7. pines de conexión entre esp32 y cámara OV7670

4. Implementación del Software.

En la Figura 8 se puede observar el proceso general (algoritmo) que se realizó al momento de querer identificar un objeto en una imagen. se inicia con la adquisición de la imagen hasta la etapa final de reconocimiento, cada etapa se ha enumerado y en ella se menciona la acción realizada.

Figura 8. Proceso general de reconocimiento.

4.1 Adquisición y envío de la imagen

4.1.1 Transmisión de la imagen. El algoritmo que permite leer una cámara OV7670 con y sin memoria FIFO se encuentra en (bitluni, 2017). Allí se explica cómo haciendo uso de los diferentes periféricos I^2S, Wi-Fi y DAC, se puede leer y transmitir una imagen al computador trabajando el ESP32, parte de este código se puede ver en la Figura 9, donde las constantes “ssid”, nombre de red y “password”, clave, permiten conectar el computador a la red creada por el ESP32 donde se transmitirá la imagen. Si la clave se define como “0000”, la red queda de libre acceso (red abierta o sin clave) ver código completo en el apéndice A.

```
server.begin();         //inicializamos el servidor
WiFi.mode(WIFI_AP);     //Access point
WiFi.softAP(ssid,password);     //Red abierta
IPAddress local_ip(192, 168, 0, 42); //Modifica la dirección IP ?
```

Desde la misma ESP

```
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);
WiFi.softAPConfig(local_ip, gateway, subnet);
 server.begin();
```

*Figura 9.*Conexión del servidor de la esp32.

Se realizaron capturas de la imagen QQ-VGA con resoluciones de 160x120 y una velocidad de transmisión de 10fps y Q-VGA con resolución de 320x240 a una velocidad de transmisión de 5fps aproximadamente. Cabe resaltar que la máxima resolución posible que se puede obtener con la cámara OV7670 es de 640x480 pixeles.

La imagen fue capturada con extensión *.BMP (bit map picture) pero es posible trabajar con otros tipos de imágenes de mapas de bits tales como *.JPG, *.PNG y *.GIF se podría usar como alternativa el formato *.JPG el cual contiene características similares en cuanto a profundidad de bits (hasta 24 bits) y un menor peso de archivo, este formato se usó en la captura de imagen con la ESP32 TTGO Camera.

4.1.2 Recepción de la imagen. En esta etapa del proceso, se busca recibir la imagen proveniente del ESP32 y adecuarla para resaltar las regiones de interés (ROI). Una vez se ha recibido la imagen en el equipo de procesamiento, se realiza una adecuación para el proceso de reconocimiento.

A continuación, se muestran las líneas de código más relevantes que se desarrollaron las cuales permiten adquirir la imagen de la dirección url mediante la conexión Wi-Fi con el ESP32 usando Python como lenguaje de programación (Python 3.6.5), se puede consultar el código completo en el Apéndice B.

El código para guardar la imagen en el PC se muestra en la Figura 10 . Para esto fue necesario instalar la librería urllib de Python que permite obtener la imagen de una URL (Uniform Resource Locator) dada.

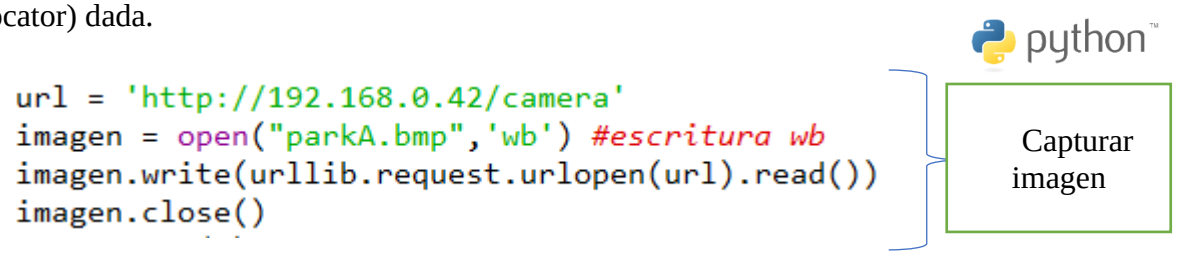


Figura 10. Guardar imagen de url al equipo usando Python.

4.2 Preprocesamiento de la imagen.

En esta etapa se realiza una adecuación, la imagen de entrada (con resolución de 320 x 240 pixeles) se convierte a escala de grises y luego se aplica un suavizado aplicándole un filtro de mediana. Este filtro no es más que una convolución de la imagen con una máscara determinada, esta máscara tendrá un tamaño de NxN con N impar para que exista un pixel central. La Figura 11 muestra este procesamiento.

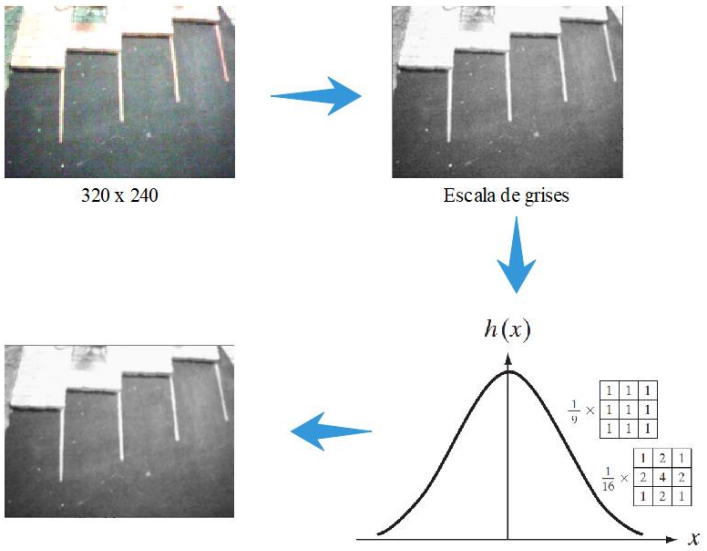


Figura 11. Adquisición y filtrado de la imagen.

La Figura 12, muestra parte del código en Python utilizado para realizar este procedimiento. Primero se convierte a escala de grises, luego se realiza un filtrado a la imagen que permite un suavizado y la eliminación de componentes de ruido que pueda presentar la imagen se puede consultar el apéndice B.

```
imgOr = cv.imread('D:\A_linux\Proyecto grado_linux\parkA.bmp')
img = cv.cvtColor(imgOr, cv.COLOR_BGR2GRAY) # escala de grises
img = cv.medianBlur(img,3) #filtrado de la imagen
```

Figura 12. Filtro y convolución con máscara normalizada mediana sobre la imagen.

4.3 Segmentación de la imagen.

Una vez se ha realizado el preprocesamiento de la imagen, se le realizan operaciones morfológicas las cuales se encuentras desarrolladas en librerías de Python con un enfoque al procesamiento

digital de imágenes. Estas operaciones van a permitir resaltar mejor los bordes de la imagen y visualizar claramente las cuatro zonas de aparcamiento vehicular (la región de interés ROI).

A continuación, se va a describir el algoritmo que se desarrolló para reconocer las plazas de parqueo. En primer lugar, a la imagen se le va a erosionar, luego esta imagen erosionada se resta de la original para resaltar sus bordes. Seguidamente se realiza una operación de "closing" y luego una operación de "filling". Finalmente, se hace uso de unas mascaras creadas previamente; *mask, y maskref*, y con operaciones de multiplicación y suma se identifica la disponibilidad del puesto de parqueo.

La erosión básicamente consiste en tomar la imagen y reducir o erosionar las zonas donde la imagen es más pronunciada, esta operación se define de la forma:

$$A \ominus B = \{z|(B)_z \subseteq A\} \tag{2.2.1}$$

La ecuación 2.2.1 indica que la erosión de A por B es el conjunto de todos los puntos z, de modo que B este contenido en A cuando tome los valores z. Este proceso sería análogo a un producto lógico and en sistemas digitales. La Figura 13 muestra un ejemplo gráfico de este proceso.

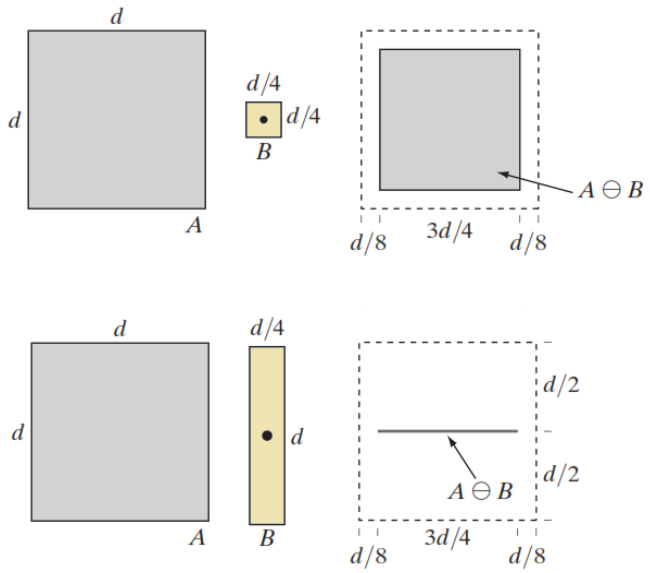


Figura 13. Erosión de A por B, modificado de Digital Image Processing, (Rafael C. Gonzales, 2008)

Para obtener la erosión, se usa un *kernel* o máscara (elemento estructurante) que se mueve sobre la imagen como sí se tratase de una convolución, donde el pixel central en la máscara solo toma el valor de 1 sí todos los pixeles o valores vecinos son igual a 1, de lo contrario el valor que toma es de cero (en el caso de una imagen binaria), estas operaciones morfológicas también son válidas en imágenes a escala de grises y con varias capas, en este trabajo, se realiza esta operación en una imagen en escala de grises.

Por otro lado, la dilatación se describe matemáticamente como:

$$A \oplus B = \{z|[(\hat{B})_z \cap A] \subseteq A\} \qquad (2.2.2)$$

Esta ecuación se basa en reflejar B sobre su origen y desplazar esta reflexión por z, entonces la dilatación de A por B es entonces el conjunto de todos los desplazamientos z de tal manera que B

y A se superponen con al menos un elemento. Se puede asociar digitalmente como sí se tratase de una compuerta or donde el resultado es 1 sí al menos uno de los valores de entrada es 1. La Figura 14 muestra un ejemplo gráfico de este proceso.

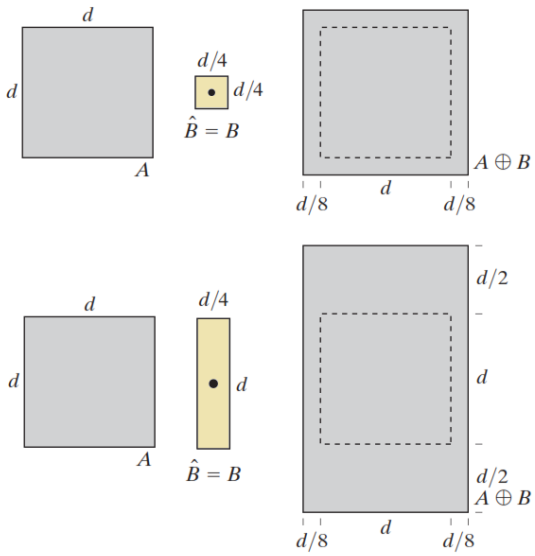


Figura 14. Dilatación de A por B, modificado de Digital Image Processing, (Rafael C. Gonzales, 2008)

Para resaltar los bordes, la imagen del parqueadero, en escala de grises, se erosiona y luego se le resta la imagen original, este resultado se visualiza en la Figura 15.

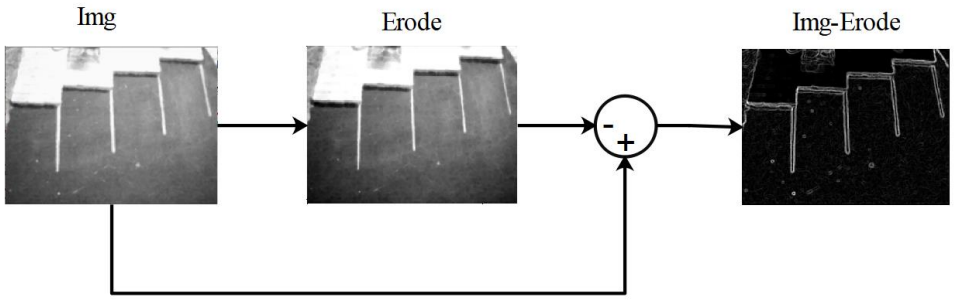


Figura 15. Resaltar bordes de la imagen mediante un proceso de erosión.

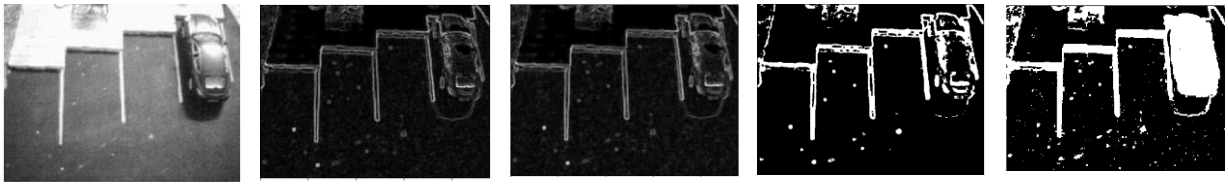
A la salida de la imagen en la Figura 15 se le realiza una operación de *closing* que consiste en realizar una dilatación entre A y B, luego al resultado de esta dilatación se aplica una erosión por B como se muestra en la ecuación 2.2.3 permitiendo que los bordes sean mejor definidos y continuos. Al terminar el proceso se binariza la imagen.

$$A \cdot B = (A \oplus B) \ominus B \qquad (2.2.3)$$

Luego de haber realizado la operación de *closing* y *binarizado* a la imagen, se procede a realizar una operación de *filling* o relleno, la operación de relleno utiliza recursivamente la ecuación 2.2.4, donde A^c representa el complemento de A. La función o la operación iterativa de la ecuación se detiene cuando se cumple que $X_k = X_{k-1}$ de lo contrario mientras no exista este criterio de parada, se sigue ejecutando.

$$X_k = (X_{k-1} \oplus B) \cap A^c \qquad (2.2.4)$$

En la Figura 16 se visualiza la imagen después de haber pasado por cada una de las etapas de procesamiento descritas anteriormente.



(a) (b) (c) (d) (e)

Figura 16. (a) imagen en escala de grises. (b) imagen con la detección de bordes usando erosión. (c) proceso de *closing* en la imagen. (d) binarización y (e) relleno o *filling* de la imagen.

Una vez se ha rellenado la imagen como se muestra en la Figura 16(e) se aplican las operaciones de multiplicación y suma con las mascaras guardadas en las variables *mask* y *maskref* como se muestra en la Figura 17 para de esta manera, obtener la región de interés ROI.

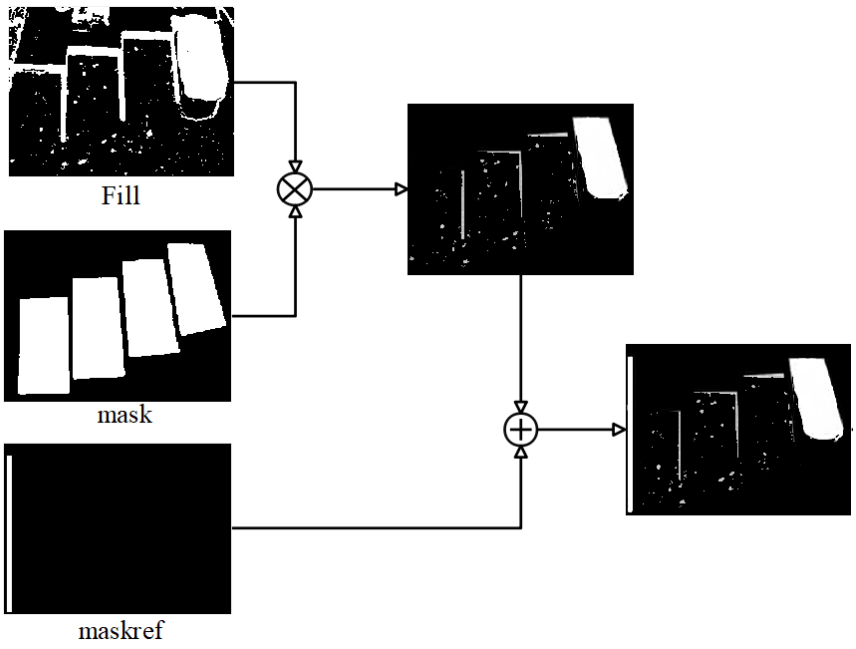


Figura 17. Imagen procesada y lista para realizar su identificación.

Se espera que al operar la imagen con la máscara *mask* se eliminen todos objetos que se encuentran fuera de la zona de parqueo (piedras, árboles, etc.). Finalmente, la operación con la máscara *maskref* permite obtener una imagen con la información relacionada con la disponibilidad o no de zonas de parqueo (coordenadas, áreas, centro de masa, etc).

Parte del código implementado en Python para realizar todas estas operaciones se muestra en la Figura 18 .

```python
#Crear un kernel de '1' de 3x3
kernel = np.array([ [1, 1, 1],
                    [1, 1, 1],
                    [1, 1, 1]],np.uint8)/9

#Se aplica la transformacion: Morphological
transformacion = img - cv.erode(img,kernel,iterations =1)
#Se aplica la transformacion: closing
transformacion = cv.morphologyEx(transformacion,cv.MORPH_CLOSE,kernel)


#se vuelve binario
ret,img = cv.threshold(transformacion,28,1,cv.THRESH_BINARY)

'''relleno de la imagen'''

seed = np.copy(img)
seed[1:-1, 1:-1] = img.max()

fill = np.uint8(reconstruction(seed, img, method='dilation'))

fill=fill*mask
fill=fill+maskref
```

Rellena la imagen

Máscaras útiles para el proceso de reconocimiento (mask, maskref)

Figura 18. Parte del código utilizado para la segmentación de la imagen.

4.4 Clasificación y reconocimiento de objetos

En este apartado se procede a identificar el estado en el que se encuentra (libre/ocupado) cada zona de aparcamiento vehicular. El algoritmo primero analiza los objetos que se encuentran en la zona de parqueo y discrimina por áreas (*min_size)* si estos corresponden o no a un vehículo. vehículo, para ello se utilizó la librería de Python openCV con la herramienta "connectedComponentsWithStats" (esta función permite encontrar y conectar todos los componentes o regiones blancas en la imagen para luego eliminar áreas menores a 3500 pixeles). Luego, con el método de detección Canny [uno de los mejores métodos de detección de contornos mediante máscaras de convolución. (Rebaza, 2007)], se detecta el contorno de cada imagen y

luego el algoritmo obtiene las coordenadas de cada uno de los vehículos detectados (las imágenes se ordenan de izquierda a derecha), (Rosebrock, 2016).

El resultado del procesamiento descrito se muestra en la Figura 19, de donde se ha modificado usando parte del código encontrado en (stackoverflow, 2017).

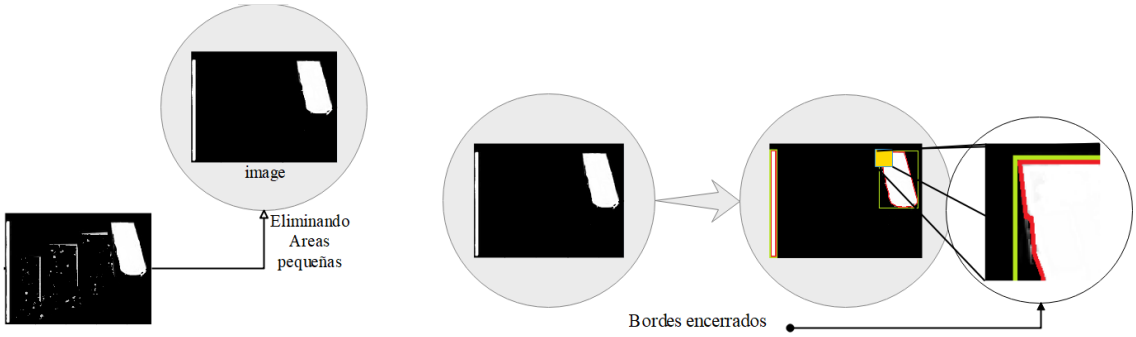


Figura 19.Eliminar áreas pequeñas en la imagen y encontrar contornos

Una vez se tiene el contorno de imagen, se procede a calcular la distancia entre cada uno de los elementos encontrados, para ello el valor de la distancia se calcula de la forma:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (2.2.5)$$

Donde P_1 sería el punto medio o centro de masa del rectángulo correspondiente con el objeto de referencia, P_2 el centro de masa de cualesquiera de los de los otros objetos que se corresponden con los vehículos encontrados y d_E hace referencia a la distancia encontrada entre el objeto de referencia y el vehículo encontrado.

Para la correcta identificación del puesto de aparcamiento, se aplicó la ecuación 2.2.5, con esta fórmula se determina la distancia del objeto detectado (vehículo) y por ende la zona de parqueo ocupada, ver Figura 20.

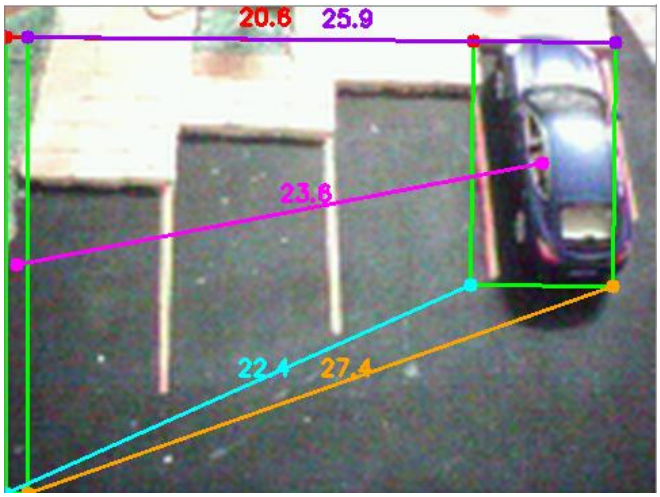


Figura 20. Distancias entre el objeto de referencia y el automóvil.

En la Figura 21 y Figura 22 se puede ver parte del código necesario para el procesamiento que se ha descrito anteriormente. En la Figura 21 se puede observar como se discriminan las áreas no deseadas para el proceso de identificación.

```
'''Eliminar areas no deseadas'''
#find all your connected components (white blobs in your image)
nb_components,output,stats,centroids=cv.connectedComponentsWithStats(fill,connectivity=8)
sizes = stats[1:, -1]; nb_components=nb_components - 1

# minimum size of particles we want to keep (number of pixels)
min_size = 3500

#your answer image
image = np.uint8(np.zeros((output.shape)))
#for every component in the image, you keep it only if it's above min_size
for i in range(0, nb_components):
    if sizes[i] >= min_size:
        image[output == i + 1] = 255
```

Figura 21. Código usado para eliminar las áreas pequeñas en la imagen.

Como se puede ver en la Figura 22, para conocer el puesto de aparcamiento en el cual se encuentra un determinado vehículo, se ha dejado un rango de valores que puede tomar las distancias medidas desde el punto de referencia, de acuerdo con estos valores encontrados, se indica la ubicación o puesto al que pertenece el objeto tomando como valor 1 cuando se encuentra un objeto y tomando 0 cuando no se encuentra y de esta forma poder indicar el estado libre u ocupado.

```
#Distancia euclidiana entre las coordenadas
D = dist.euclidean((xA, yA), (xB, yB)) / refObj[2]
(mX, mY) = midpoint((xA, yA), (xB, yB))
cv.putText(orig, "{:.1f}in".format(D), (int(mX), int(mY - 10)),
   cv.FONT_HERSHEY_SIMPLEX, 0.55, color, 2)

 #Almacenar valores
d+=[D]

#Almacenar las distancias y criterio de selección
 for i in range(4,len(d),5):
     dd+=[d[i]]

 for i in range(len(dd)):
     if dd[i]<=13:
         puesto1 = 1
     if 13<dd[i]<=23:
         puesto2 = 1
     if 23<dd[i]<=33:
         puesto3 = 1
     if 33<dd[i]<=42:
         puesto4 = 1
```

Se determina el puesto de acuerdo con la distancia entre el objeto y el punto de referencia de maskref.

Figura 22. Cálculo de distancias entre un objeto y su referencia e identificación del puesto.

5. Creación de la base de datos

Con base en el procesamiento de imágenes presentado ya se conoce el estado de cada puesto de parqueo y con esta información se crea una base de datos (se escogió MySQL phpMyAdmin) para poder visualizarla en una aplicación web.

Por medio de phpMyAdmin, se crea la base de datos “park1” y dentro de esta se genera una tabla llamada “aparcamiento” que va a almacenar la información actualizada del parqueadero (para este proyecto tiene cuatro puestos), ver Figura 23.

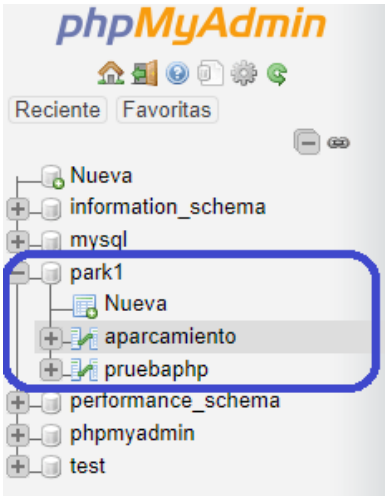


Figura 23. Creación de base de datos park1 en phpMyadmin.

La tabla contiene cuatro columnas llamadas “puesto1”, “puesto2”, “puesto3” y “puesto4” y en cada una de ellas se escribirá un “1” sí la zona se encuentra ocupada o un “0” sí se encuentra libre, ver Figura 24.

Figura 24. Creación de la tabla con los puestos del aparcamiento.

Para colocar esta información en “park1”, el algoritmo que se muestra en la Figura 25 explica cómo se conecta un servidor “host” a la base de datos. Allí se crea un cursor para actualizar la base de datos directamente desde el software de procesamiento de imágenes desarrollado en Python. Para una revisión más detallada se puede consultar directamente en la página (Pymysql, 2016).

```
conn = pymysql.connect(host='localhost',
                       user='root',
                       password='',
                       port=3306,
                       bind_address ='127.0.0.1',
                       database='park1')
cursor = conn.cursor()
```

Figura 25. Conexión a la base de datos desde Python usando pymysql.

Para poder actualizar la base de datos, se implementó el código que se muestra en la figura 27. MySQL debe comprender las indicaciones dadas desde Python. La información de escritura se debe enviar como si se estuviese escribiendo directamente en MySQL utilizando el siguiente comando:

cursor.execute(sql) donde sql es una variable tipo string para Python que realizar modificaciones a la base de datos de MySQL.

Algunos comandos básicos SQL son:

- SELECT se utiliza cuando se quiere leer (o seleccionar) los datos.
- INSERT se utiliza cuando se quiere añadir (o insertar) nuevos datos.
- UPDATE se utiliza cuando se quiere cambiar (o actualizar) datos existentes.
- DELETE se utiliza cuando se quiere eliminar (o borrar) datos existentes.
- REPLACE se utiliza cuando se va a cambiar (o reemplazar) datos nuevos o ya existentes.
- TRUNCATE se utiliza cuando se quiere vaciar (o borrar) todos los datos de la plantilla.

Al final se cierra la conexión desde Python escribiendo el comando conn.close() .

```
if puesto1 == 1:
    print("ocupado")
    sql = "UPDATE `aparcamiento` SET `puesto1`='1';"
else:
    print("Libre")
    sql = "UPDATE `aparcamiento` SET `puesto1`='0';"
cursor.execute(sql)
if puesto2 == 1:
    print("ocupado")
    sql = "UPDATE `aparcamiento` SET `puesto2`='1';"
else:
    print("Libre")
    sql = "UPDATE `aparcamiento` SET `puesto2`='0';"
cursor.execute(sql)
if puesto3 == 1:
    print("ocupado")
    sql = "UPDATE `aparcamiento` SET `puesto3`='1';"
else:
    print("Libre")
    sql = "UPDATE `aparcamiento` SET `puesto3`='0';"
cursor.execute(sql)
if puesto4 == 1:
    print("ocupado")
    sql = "UPDATE `aparcamiento` SET `puesto4`='1';"
else:
    print("Libre")
    sql = "UPDATE `aparcamiento` SET `puesto4`='0';"
cursor.execute(sql)
```

Figura 26. Actualizar datos existentes en cada puesto del aparcamiento.

6. Desarrollo de aplicación web

Para la visualización del estado libre/ocupado por parte del usuario se desarrolló, con la herramienta HTML, PHP y estilos CSS, una aplicación con el objetivo de permitirle al usuario acceder a esta información desde un terminal móvil o un equipo de cómputo, ver Figura 27.

(a) (b)

Figura 27. Visualización del diseño de la interfaz de usuario. En a) se observa la ventana de inicio de la aplicación web y en b) Información en tiempo real de zonas de parqueo disponibles.

Acerca de las herramientas usadas para la creación de la página web, HTML (Hyper Text Markup Languaje) es un lenguaje de programación que se utiliza para desarrollo de páginas de internet generalmente usado para contenido estático, mientras tanto PHP (Hypertext Preprocessor) es un lenguaje de código abierto para desarrollo *web* basado en contenido dinámico ofreciendo la posibilidad de conectar directamente con la base de datos creada previamente en Mysql. Por ultimo y no menos importante, CSS (Cascading Style sheets) es un lenguaje exclusivamente de diseño gráfico que permite crear la presentación, forma, color y los estilos en general del documento web, su uso tiene una gran importancia visual, su versatilidad ofrece un gran margen de diseños y estilos en páginas web con facilidad de uso en los contenidos.

En este trabajo se ha utilizado estas tres herramientas de diseño web, que combinadas con la base de datos de Mysql y las instrucciones de Python ofrecen una potente herramienta para desarrollo de proyectos de ingeniería basado en el internet de las cosas IoT.

En el apéndice C, se encuentra el código utilizado para la elaboración de esta aplicación.

7. Pruebas y Resultados

Para la validación del hardware y del software de reconocimiento de imágenes, se realizaron una serie de pruebas en donde se indican la cantidad de aciertos y fallos al identificar si la zona de parqueo está libre u ocupada. Con la maqueta se realizaron más de 400 pruebas con imágenes de diferentes niveles de iluminación, en términos generales estas se pueden resumir así:

- Vehículos de diferentes colores ubicados correctamente en las zonas de parqueo.
- Vehículos de diferentes colores ocupando más de un puesto de estacionados.
- Zonas de parqueo con objetos diferentes a un vehículo.
- Corrección frente a corrimiento de la cámara (ver apéndice D).
- Pruebas realizadas con el ESP32 TTGO-Camera (ver apéndice E).
- Pruebas realizadas a un parqueadero real (ver apéndice F).

La Figura 28 muestra algunas imágenes de captura para la validación de resultados y de esta manera comprobar su efectividad. (diferentes niveles de iluminación, personas, hojas, carros mal parqueados)

Figura 28. Muestras de toma de imagen de algunos vehículos para la validación de resultados.

Se realizaron pruebas teniendo en cuenta diferentes niveles de luminosidad en la imagen, para conocer el nivel de luminosidad sobre el cual se trabaja la imagen se convierte al espacio de color

Lab, donde dicho espacio fue modelado en base a una teoría de color oponente que establece que dos colores no pueden ser rojo y verde al mismo tiempo o amarillo y azul al mismo tiempo. L*indica la luminosidad y a* y b* son las coordenadas cromáticas.

La Figura 29 muestra diferentes niveles de luminosidad con los cuales se realizaron pruebas. El nivel de luminosidad no está dado en lux (unidad derivada del Sistema Internacional de Unidades para la iluminancia o nivel de iluminación) este nivel corresponde al promedio de la luminosidad sacada del espacio de color Lab de la imagen obtenida en una cámara específica sin calibrar, en este caso la cámara OV7670.

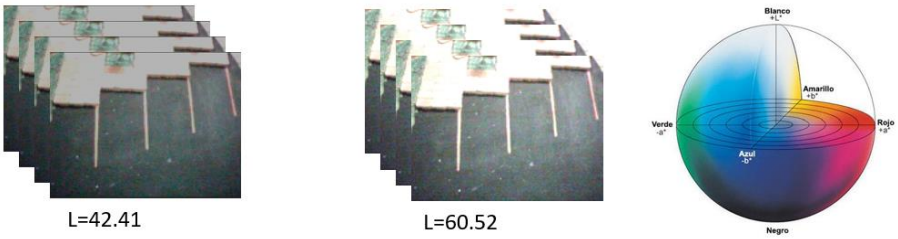


Figura 29. Niveles de luminosidad capturados con la cámara OV7670.

En la *Tabla 3* se muestran las gráficas del acierto en porcentaje para vehículos correctamente ubicados en la zona de parqueo, a cada uno de los vehículos se les ha realizado 100 tomas con baja iluminación.

Tabla 3. Resultados obtenidos con los niveles de luminosidad de la figura 29.

Luminosidad	Un vehículo	Dos vehículos	Tres vehículos	Cuatro Vehículos
42.41	99%	97%	99%	100%
60.52	98%	98%	96%	99%

Porcentaje de aciertos en base a 100 muestras.

Las pruebas anteriores se repitieron con vehículos parqueados ocupando cada uno de ellos más de un sitio de parqueo. En la Figura 30 se muestra un caso en el que un vehículo se ubica en medio de un par de puestos de parqueo, se espera que cuando un vehículo se encuentre justo en medio de las líneas que dividen las plazas de parqueo el software indique ambos puestos como ocupados. Los resultados correspondientes con esta situación se observan en la Tabla 4.

Figura 30. Muestra de imágenes de vehículos parqueados en medio de dos plazas.

Tabla 4 .Pruebas para vehículos ubicados sobre la línea.

Descripción	Resultados
1 vehículo estacionado sobre la línea de división	Se reconocen ocupados 2 puestos de parqueo el 80% de las veces
2 vehículos estacionados cada uno sobre una línea de división	Se reconocen ocupados 4 puestos de parqueo el 90% de las veces
3 vehículos estacionados cada uno sobre una línea de división	Se reconocen ocupados 4 puestos de parqueo el 90% de las veces
4 vehículos estacionados cada uno sobre una línea de división	Se reconocen ocupados 4 puestos de parqueo el 100% de las veces.

Se espera que, al estacionarse entre dos puestos, estos se muestren ocupados en la aplicación web.

Finalmente se analizó la situación que se presenta en el parqueadero cuando se toma una imagen de carros en su búsqueda o abandono del sitio de parqueo. La Figura 31 muestra algunas posiciones posibles en las que se puede encontrar estos vehículos y en la Figura 32 se muestra los resultados vistos por el usuario en la aplicación web de algunas de estas ubicaciones.

(a) (b) (c)

Figura 31. Vehículos ocupando más de un puesto ingresando o saliendo del parqueadero.

Figura 32. Vehículos ingresando o saliendo del aparcamiento vistos desde la aplicación web.

En los apéndices G se pueden ver algunos de los resultados obtenidos y como estos mismos se visualizan en la aplicación web.

En la Figura 33 se pueden observar elementos distribuidos en un puesto de aparcamiento. Las áreas distribuidas en el aparcamiento vehicular son discriminadas permitiendo que la plaza del estacionamiento se muestre libre, esto es con el fin de reducir lecturas erróneas debido a piedras, hojas otro objeto ubicado en la plaza del parqueadero.

Figura 33. Regiones que no se identifican como un puesto ocupado.

Con el fin de validar la calidad, se restringe el área que debería ocupar un vehículo discriminando aquellas de dimensiones inferiores al 20% del área que ocupa un vehículo promedio, en cuanto al cálculo del área a discriminar se calculó de la siguiente manera:

$$Area_{disc} = Area_{vehículo} - Area_{vehiculo} * (0.20) \qquad (2.5.1)$$

8. Conclusiones

Se creó una maqueta de un parqueadero con una escala de 1:48 y allí se realizaron las pruebas con vehículos y objetos de diferentes tamaños y en ambientes de iluminación controlados. Las pruebas realizadas al software de reconocimiento mostraron altos porcentajes de éxito.

Para la transmisión Wi-Fi de las imágenes desde el ESP32 al equipo de cómputo, se utilizaron diferentes cámaras y se ensayaron diferentes resoluciones. Se puede concluir que con resoluciones de mínimo 320-240 el proceso de reconocimiento es exitoso y la velocidad de transmisión de la imagen es menor a 1 segundo.

Se logró transmitir mediante Wi-Fi la imagen desde el ESP32 al equipo central de procesamiento(computador) y en este se realizó con un porcentaje de aciertos superior al 97%, el proceso de reconocimiento de zonas de parqueo disponibles usando Python como lenguaje de programación.

Se desarrolló una aplicación web usando las herramientas HTML, PHP y CSS que le permiten al usuario, visualizar de forma gráfica y mediante indicación de dos colores (verde y rojo) el estado del puesto de aparcamiento como libre y ocupado.

Se realizaron pruebas en campo, tomando un grupo de imágenes correspondientes a un parqueadero y se realizaron pruebas con esas imágenes adaptando las máscaras para su

funcionamiento y se obtuvieron buenos resultados, donde se pudo identificar el número de vehículos en la zona de aparcamiento.

El algoritmo desarrollado se ensayó con imágenes con resolución de hasta $1366x768$px de un parqueadero real sin que se afecte la aplicación, esto es porque el tiempo de transmisión de la imagen es más rápido que el tiempo que se necesita para actualizar la información de la aplicación el cual se ha definido de 5 segundos.

El software puede realizar el proceso de reconocimiento aún si la cámara no se encuentra en una posición fija, pero exige una mejor resolución de la cámara para evitar errores al momento de procesar la imagen, con esto el sistema es más robusto y funciona aún si la cámara sufre un desplazamiento que no sea crítico, pero para ello, se requiere una mejor calidad de imagen.

Se obtuvieron buenos resultados incluso con bajas resoluciones, lo cual significa el uso de cámaras de bajo costo. Como la aplicación realiza cambios cada 5 segundos, no es necesario que el número de fotogramas por segundo (FPS) sea alto, bastaría con capturas de imagen de menos del tiempo de la aplicación.

El tiempo que dura el software en realizar el proceso de reconocimiento es de alrededor 1.5s, donde un segundo se deja de espera mientras la imagen de la URL es descargada al computador y el resto de tiempo corresponde con el proceso de reconocimiento. Los valores son tomados de un equipo de cómputo con procesador Intel core i5 con memoria ram de 4Gb.

La cantidad de memoria necesaria para almacenar el programa junto con las imágenes de referencia es alrededor de 72.0 KB. El tamaño puede verse afectado por la cantidad de imágenes y la resolución de estas mismas.

9. Recomendaciones

Como siguiente paso a este proyecto se recomienda realizar en el mismo ESP32, el procesamiento de la imagen (aprovechando que es un proceso lento el parqueo) y enviar por medio de una aplicación web, el número de plazas disponibles. Igualmente se propone ajustar el algoritmo de reconocimiento a un número de puestos de parqueaderos variable.

Se recomienda desarrollar algoritmos que identifiquen y alerten al administrador del parqueadero de vehículos u objetos mal estacionados, también es posible hacer que puedan identificar placas y hacer cobro por tiempo de parqueo.

Es posible realizar algunos cambios como usar elementos de hardware que permitan realizar el procesamiento directamente en el lugar donde se encuentra la cámara, para este trabajo como se trata de un software implementado en Python, se podría utilizar la Raspberry y adaptarle una cámara, de esta manera se evita usar un computador externo. Se puede revisar otras opciones de miniordenadores en el mercado como Olimex, Cubie tech, Banana Pi y Orange pi. Igualmente se puede hacer una revisión de cámaras dedicadas al procesamiento como lo sería la Pixy2.

Se recomienda una resolución mayor a $320x240$ pixeles para aumentar el éxito del procesamiento, es posible que se presente un mayor error si se toma una resolución inferior.

Garantizar la iluminación adecuada mediante sistemas de visión nocturna podría ser una solución para la detección en un ambiente oscuro, se puede consultar en el mercado los diferentes tipos de cámaras de los distintos desarrolladores.

Es posible hacer que la cámara siga realizando el proceso de reconocimiento aún si esta esta en movimiento, para ello se recomienda consultar sobre las transformaciones proyectivas que se puede realizar a una imagen para ajustarla a una de referencia. Consultar geometría proyectiva.

Otra posible solución consiste en encontrar la correlación entre un par de imágenes que permite medir las deformaciones o desplazamientos en una imagen, con esto se podría corregir sí existe un corrimiento en la imagen para ajustarla con la de referencia y de esta manera no afectar el resultado debido al corrimiento.

Es muy importante, que al momento de subir a un servidor web se puedan modificar los datos desde el equipo central de computo mediante Python como lenguaje de programación, esto es debido a que no todos los servidores en la nube permiten realizar esta tarea, algunos son más restringidos que otros y no permiten modificaciones desde otro aplicativo, como ejemplo sería un gestor de base de datos phpMyAdmin de Atspace (ATSPACE, 2018) el cual no permite escribir desde Python, mientras tanto el gestor de base de datos que ofrece Freesqldatabase (FreeSQLdatabase, 2017) funciona correctamente y permite la escritura externa.

Referencias Bibliográficas

Anaconda. (2019). *Anaconda Distribution*. Obtenido de https://www.anaconda.com/distribution/

ATSPACE. (2018). Obtenido de ATSPACE: https://www.atspace.com/

Auto Alfre. (3 de Julio de 2015). *Youtube*. Obtenido de entrada hacia atrás ESTACIONAMIENTO BATERIA RECTO: https://www.youtube.com/watch?v=Ya1nvXNm2VE

bitluni. (2017). *Github*. Obtenido de ESP32 I2S Camera: https://github.com/bitluni/ESP32CameraI2S

Espressif. (2018). *ESP32 series Datasheet.* Obtenido de https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Espressif EYE. (2018). *ESP-EYE Development Board.* Obtenido de https://www.espressif.com/en/products/hardware/esp-eye/overview

FreeSQLdatabase. (2017). *Free sql database.* Obtenido de https://www.freesqldatabase.com/

ICONTEC. (15 de 12 de 2006). Norma tecnica colombiana NTC, centros de diagnóstico automotor. Bogotá, Colombia.

Mallick, S. (9 de marzo de 2018). *Learn OpenCV*. Obtenido de GitHub: https://github.com/spmallick/learnopencv

motor. (2017). *top 10 de los carros más vendidos en Colombia.* Obtenido de https://www.motor.com.co/fotos/top-10-carros-vendidos-colombia/27469

Omnivision. (8 de 7 de 2005). *OV7670/OV7171 CMOS VGA (640x480) camera Chip^tm whit omnipixel tecnology.* Obtenido de https://www.voti.nl/docs/OV7670.pdf

Pymysql. (2016). *Objeto de conexión*. Obtenido de https://pymysql.readthedocs.io/en/latest/modules/connections.html

Radimer . (2017). *Grupo Radimer*. Obtenido de smart parking con camaras sensorizadas : https://www.radimer.es/servicios/solucion-lpr/

Rafael C. Gonzales, R. E. (2008). *Digital Imagen Processing* . New Jersey: Pearson Education, Inc.

Rebaza, J. V. (2007). *Detección de bordes mediante el algoritmo de canny.*

Rosebrock, A. (4 de abril de 2016). *Image Processing*. Obtenido de Pyimagesearch: https://www.pyimagesearch.com/2016/04/04/measuring-distance-between-objects-in-an-image-with-opencv/

stackoverflow. (2017). Obtenido de stackoverflow: https://stackoverflow.com/questions/42798659/how-to-remove-small-connected-objects-using-opencv

Talha Kiliç, T. T. (2017). Smart city application: android based smart parking system. *International Artificial Intelligence and Data Processing Symposium (IDAP).* IEEE conference publication.