Disponibilidad en zonas de estacionamiento mediante una cámara e IOT

Eder Arnulfo Sánchez Villabona y Adriana Yeslendy Suárez Saavedra

Trabajo de grado en la modalidad de investigación Presentado para optar por el título de

Ingeniero Electrónico

Director:

Jaime Guillermo Barrero Pérez

Ingeniero Electricista

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2020

Contenido

Pág.

Lista de Figuras

Pág.

Lista de Tablas

Pág.

Lista de Apéndices

Pág.

Resumen

Título: Disponibilidad en zonas de estacionamiento mediante una cámara e IOT[*]

Autores: Eder Arnulfo Sánchez Villabona, Adriana Yeslendy Suárez Saavedra.[**]

Palabras Claves: Aplicación *web*, procesamiento digital de imágenes, Raspberry

Descripción:

El presente trabajo de grado tuvo como propósito desarrollar e implementar un prototipo de un sistema que permitiera determinar la disponibilidad y ubicación de plazas en una zona de estacionamiento vehicular. Con el adecuado procesamiento de imágenes se logró presentar la información al usuario: zonas libres u ocupadas mediante una aplicación *web*. Además, se agregó un estado de alerta que se muestra cuando el usuario no estaciona adecuadamente su vehículo.

Inicialmente, se adquirieron los componentes necesarios: una cámara de 5Mpx ubicada de manera estratégica para capturar la totalidad del estacionamiento y una Raspberry Zero W encargada de obtener la imagen, procesarla y enviar la información necesaria a la aplicación web. Posterior a esto, se construyó una maqueta a escala de una zona de estacionamiento y los algoritmos necesarios para la identificación de las zonas disponibles, usando Python como lenguaje de programación y algunas librerías como Opencv y numpy.

Finalmente, usando HTML, css y JavaScript se realizó la aplicación web y la comunicación con la Raspberry para realizar las pruebas de funcionamiento. La aplicación es de fácil acceso y se encuentra de manera global, por lo tanto, se puede acceder a ella desde cualquier dispositivo: móvil u ordenador, sin realizar un registro previo.

[*] Proyecto de grado

[**] Facultad de Ingenierías Fisicomecánicas Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones Director: Jaime Guillermo Barrero Pérez Ingeniero Electricista

Abstract

Title: Availability in parking areas through a camera and IOT[*]

Autors: Eder Arnulfo Sánchez Villabona, Adriana Yeslendy Suárez Saavedra. [**]

Keywords: Web application, digital image processing, Raspberry.

Description:

The present degree project aimed to develop and implement a prototype of a system that would allow to determine the availability and location of parking spaces in a vehicular parking area. With the adequate image processing, the information was presented to the user: free or occupied areas, through a "web" application. In addition, an alert status was added that displays when the user does not park their vehicle properly.

Initially, the necessary components were acquired: a 5Mpx camera strategically located to capture the entire parking lot and a Raspberry Zero W in charge of obtaining the image, processing it and sending the necessary information to the "web" application. Subsequent to this, a scale model of a parking area and the necessary algorithms for the identification of the available areas were built, using Python as the programming language and some libraries such as Opencv and Numpy.

Finally, using HTML, css y Java Script, the “web” application and communication with the Raspberry were conducted to perform the functional tests. The application is easily accessible and found globally, therefore, it can be accessed from any device: mobile or computer, without prior registration.

[*] Proyecto de grado

[**] Facultad de Ingenierías Fisicomecánicas Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones Director: Jaime Guillermo Barrero Pérez Ingeniero Electricista

Introducción

El uso de tecnologías de la información y comunicación (TICs) ha permitido adicionar el término *smart* a ciertos dispositivos, incorporando a su definición la conexión a internet como elemento fundamental para su funcionamiento. Estas innovaciones permiten estar informados de lo que sucede a nuestro alrededor, mejoran la comunicación y facilitan labores cotidianas como la localización de lugares de nuestro interés.

Gracias a estos avances, las personas pueden conocer la ubicación de los lugares que deseen, sin embargo, no tienen información detallada de lo que sucede en el interior, por ejemplo, al llegar a una zona de estacionamiento no tienen certeza de encontrar una plaza disponible ni la posición exacta de ella. Algunos establecimientos cuentan con un sistema de sensores que indican, mediante leds, la disponibilidad de una plaza, pero esto solo se puede observar al ingresar al lugar.

Por esta razón se implementó un prototipo de un sistema, formado por un microprocesador y una cámara, capaz de suministrar dicha información a través de IoT. El canal de comunicación que se creó entre la zona de estacionamiento y la persona que necesita un sitio para dejar su automóvil es una aplicación web, esta permite observar la zona y las plazas disponibles antes de ingresar al lugar y durante la estadía en este. De esta manera, el interesado podrá decidir donde estacionar su vehículo, ahorrando tiempo, combustible y evitando que ingrese a una zona de estacionamiento sin lugares disponibles.

1. Generalidades

1.1 Objetivo General

Desarrollar e implementar un prototipo de un sistema que permita determinar la disponibilidad, cantidad y ubicación de plazas en una zona de estacionamiento vehicular.

1.2 Objetivos Específicos

- Capturar imágenes de la zona de estacionamiento mediante una cámara y un microprocesador.
- Desarrollar los algoritmos requeridos para procesar las imágenes en el microprocesador y de esta manera obtener la ubicación de los sitios disponibles para estacionar.
- Desarrollar una aplicación *web* que le permita al usuario conocer en tiempo real las zonas de estacionamiento disponibles.

2. Marco Teórico

2.1 Procesamiento digital de imágenes

Una imagen puede ser definida matemáticamente como una función bidimensional, f(x,y), donde x y y son coordenadas espaciales (en un plano), y f es la intensidad o nivel de gris de la imagen en ese punto. Esta coordenada con su respectiva intensidad se conoce como pixel.

El procesamiento digital de imágenes es un conjunto de técnicas aplicadas a una imagen y se clasifican en tres niveles:

Proceso de bajo nivel: Utilizan operaciones como el preprocesamiento de imagen para reducir el ruido, mejorar el contraste y filtros de enfoque. Se caracterizan porque sus entradas son imágenes y sus salidas también.

Proceso de nivel medio: Operaciones como segmentación y clasificación de objetos individuales. Se caracterizan porque sus entradas son generalmente imágenes, pero sus salidas son atributos extraídos de esas imágenes (contornos, bordes, identidad de objetos individuales).

Proceso de alto nivel: Implica el obtener algún significado de un conjunto de objetos reconocidos y, finalmente, realizar las funciones cognitivas asociadas con la vista. Por ejemplo, reconocimiento de señales de tránsito, matrículas de los vehículos, entre otros (Mejia, 2005).

Se aplicaron los siguientes procesos para obtener la información deseada:

Escala de grises: Una imagen en escala de grises se representa mediante una matriz de intensidad, para encontrar la intensidad de una imagen en formato RGB se halla la proyección del vector (R,G,B) sobre (1,1,1) que representa las diferentes tonalidades de gris:

$$Proy \equiv (R,G,B).(1,1,1) = R + G + B = \left|\vec{V}\right||\hat{n}|\cos(\emptyset) \qquad (1)$$

Donde $\boldsymbol{Proy}$ es la intensidad, $\vec{\boldsymbol{V}}$ es el vector que forma el punto (R,G,B), $\hat{\boldsymbol{n}}$ es el vector de proyección (1,1,1) y $\emptyset$ es el angulo que forma $\vec{\boldsymbol{V}}$ con $\hat{\boldsymbol{n}}$.

Por lo tanto $\boldsymbol{Proy} = \frac{\boldsymbol{R+G+B}}{\sqrt{\boldsymbol{3}}}$, si este valor es mayor a 255 se debe renormalizar obteniendo:

$$\boldsymbol{Proy} = \frac{\boldsymbol{R+G+B}}{\boldsymbol{3}}$$

Figura 1.

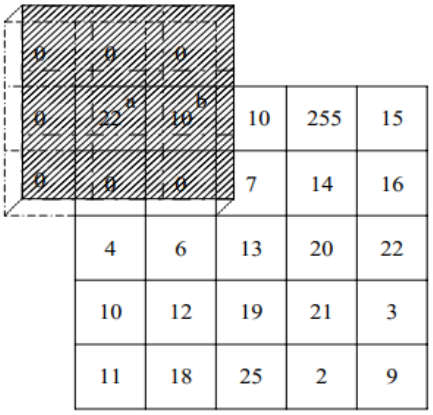
Transformación a escala de grises

Nota. Tomado de: Mordvintsev, A.; Abid, K. (2017) OpenCV-Python Tutorials Documentation. Recuperado de: https://readthedocs.org/projects/opencv-python-tutroals/downloads/pdf/latest/

Filtrado por la mediana: Es un filtro de desenfoque que permite suavizar la imagen y eliminar el ruido. Para este filtrado es necesario tomar el valor de los pixeles que forman la vencidad del pixel a modificar, se ordenan de menor a mayor y el nuevo valor del pixel es la mediana (Esqueda, 2002), es decir:

Figura 2.

Vecindad de un pixel



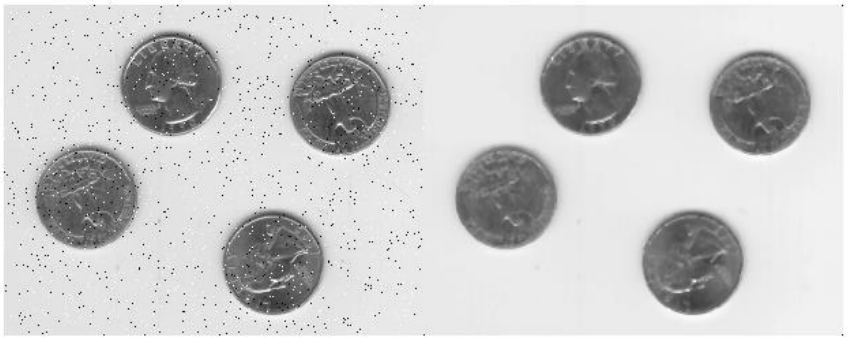
Nota. Tomado de: Esqueda, J. J. (2002) Fundamentos de Procesamiento de Imágenes. Universidad Autónoma de Baja California. Recuperado de: https://www.academia.edu/16801512/Fundamentos_de_procesamiento_de_im%C3%A1genes_digitales

La vecindad de a está formada por [0,0,0,0,22,10,0,0,0].

Ordenado de menor a mayor [0,0,0,0,0,0,0,10,22], donde el número subrayado es la mediana, este proceso se realiza para cada pixel y así se obtiene la nueva imagen.

Figura 3.

Filtrado por la mediana



Nota. Tomado de: Filtrado de mediana 2D - MATLAB medfilt2 (s.f.) MATLAB - El lenguaje del cálculo técnico Recuperado de: https://la.mathworks.com/help/images/ref/medfilt2.html.

Binarización: Este proceso se realiza mediante un umbral, el cual se compara con cada pixel de la imagen y, teniendo en cuenta si es menor, mayor o igual, se asigna 0 o 1, respectivamente, matemáticamente se puede expresar así:

$$g(x,y) = \begin{cases} 1, & f(x,y) \geq T \\ 0, & f(x,y) < T \end{cases}$$

Donde $f(x,y)$ es cada pixel de la imagen y T el umbral deseado (Fernandez, 2019).

Figura 4.

Binarización

Nota. Tomado de: Convertir imagen a imagen binaria, basada en umbral - MATLAB im2bw. (s.f.) MATLAB - El lenguaje del cálculo técnico Recuperado de https://la.mathworks.com/help/images/ref/im2bw.html

Operaciones morfológicas:

Erosión: Matemáticamente se define como:

$$A \ominus B = \{ z|(B)_z \subseteq A \} \quad (2)$$

Donde A es la imagen a erosionar, B es el elemento estructurante y z un punto (x,y).

De la ecuación (2) se tiene que la erosión de A por B es el conjunto de todos los puntos z de modo que B trasladado por z, $(B)_z$, está contenido en A (Gonzalez & Woods, 2008).

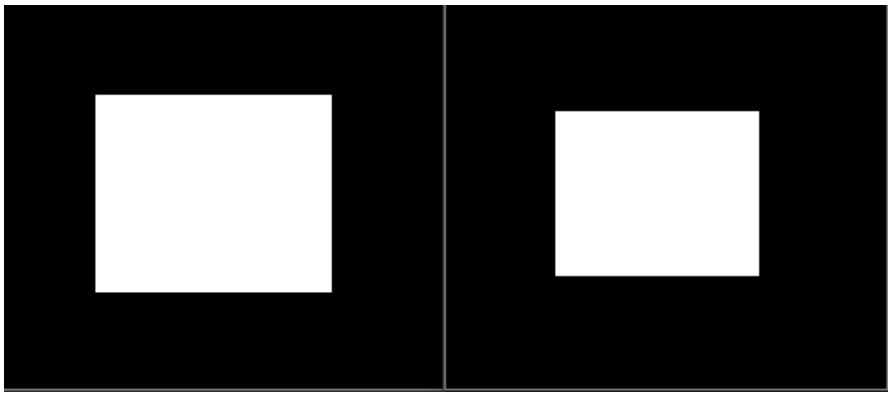
La afirmación que B debe estar contenido en A es equivalente a que B no comparta ningún elemento común con el fondo, de otra forma, la erosión se puede expresar como:

$$A \ominus B = \{z|(B)_z \cap A^c\} = \emptyset \quad (3)$$

Donde A^c es el complemento de A y $\emptyset$ es el conjunto vacío.

Figura 5.

Erosión de A por una matriz blanca



Dilatación: Matemáticamente se define como:

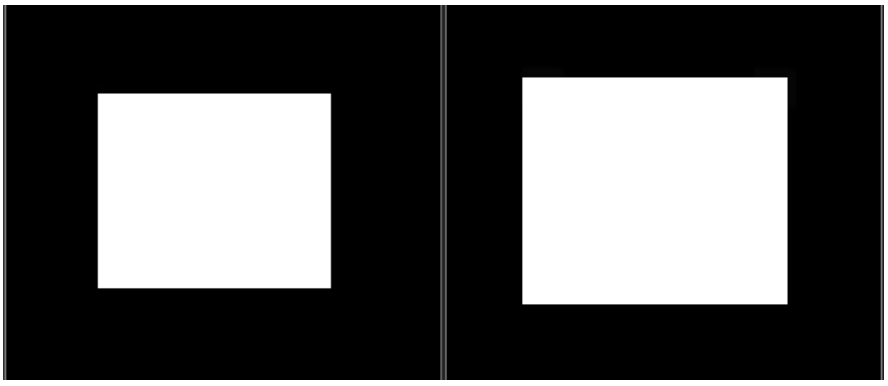
$$A \oplus B = \left\{z|(\hat{B})_z \cap A \neq \emptyset\right\} \qquad (4)$$

Donde A es la imagen a dilatar, B el elemento estructurante, z un punto (x,y), $\hat{B}$ es la reflexión de B respecto a su origen y $\emptyset$ el conjunto vacío.

La ecuación (4) se basa en reflejar el elemento estructurante, $\hat{B}$, sobre su origen y desplazarlo por z. La dilatación de A por B es el conjunto de todos los desplazamientos, z, de modo que, $\hat{B}$ y A se superponen por al menos un elemento. (Gonzalez & Woods, 2008)

Figura 6.

Dilatación de A por una matriz blanca



Los procesos mencionados se realizaron usando la Raspberry Zero W como herramienta de cómputo y Python como lenguaje de programación de propósito general.

Python permite trabajar rápidamente e integrar sistemas de manera más efectiva, dentro de sus aplicaciones tenemos desarrollo *web* e internet, cálculos numéricos, creación de interfaces gráficas, desarrollo de software, control, construcción de sistemas de comercio electrónico, entre otras (Aplicaciones para python, s.f.).

Para resolver problemas de visión por computadora python utiliza una biblioteca de enlaces conocida como OpenCV, que a su vez, hace uso de Numpy, que es una biblioteca optimizada para operaciones numéricas (Mordvintsev & Rahman, Introducción a OpenCV. OpenCV., s.f.).**2.2 Sistemas "IoT"**

"IoT" ("Internet of Things", en inglés) describe la red de objetos físicos que llevan sensores integrados, software y otras tecnologías con el fin de conectar e intercambiar datos con otros dispositivos y sistemas a través de Internet. Estos dispositivos abarcan desde objetos domésticos cotidianos hasta sofisticadas herramientas industriales (Oracle, s.f.).

Un sistema “IoT” está compuesto de cuatro elementos claves:

Hardware: Son los sensores y dispositivos que recopilan datos del entorno, por ejemplo, sensores de humedad, temperatura, cámaras, etc. El hardware también se encarga de realizar acciones específicas en función de los datos recolectados.

Conectividad: El hardware necesita transmitir los datos recolectados, generalmente a entornos de nube. También necesita recibir comandos desde la nube, asociados a la realización de acciones concretas.Algunos sistemas “IoT” utilizan un componente intermedio entre el hardware y la conexión a la nube, generalmente una puerta de enlace o un enrutador.

Software: El software es el responsable de analizar todos los datos recibidos y generar acciones concretas en función de estos datos.

Interfaz de usuario: Todo sistema “IoT” necesita una interfaz, a través de la cual los usuarios interactúan con el sistema “IoT”, por ejemplo, una aplicación “web” (News America Digital, 2019).

Una aplicación “web” es una versión de una página “web” que ha sido optimizada, normalmente por un equipo de desarrollo, para poder ser utilizada desde un computador o teléfono móvil.

Se caracterizan por ser amigables con el usuario, además, requieren un único desarrollo para cualquier dispositivo, no es necesario descargarlas y son accesibles desde cualquier navegador.

3. Procedimiento

El proceso de identificar la disponibilidad de las zonas de estacionamiento y enviar los datos necesarios para informar a través de una aplicación “web”, se realizó mediante diferentes etapas, inicialmente se adquirieron y procesaron las imágenes para detectar los vehículos, posteriormente se seleccionaron los datos a enviar y la forma de mostrarlos en la aplicación.

Para desarrollar cada etapa se utilizaron diferentes funciones, algunas de la librería OpenCV, las cuales se mencionan en su inicio como *cv2*, adicional a esto, se crearon otras funciones que se pueden reconocer por iniciar con las letras *ea_*. Por ejemplo cv2.Floodfill( ) y *ea_threshold( ).*

3.1 Adquisición de imágenes

Con el objetivo de recrear de la manera más realista posible un sistema de visión artificial, se construyó una maqueta de una zona de estacionamiento a escala 1:43 con cinco plazas para aparcar y las dimensiones de cada zona de estacionamiento se tomaron del POT realizado por la alcaldía de Bucaramanga (Alcaldía de Bucaramanga, Secretaría de planeación, 2014). Se ubicó una cámara a una altura de 33cm con el propósito de obtener, de la mejor forma, la totalidad de las plazas de estacionamiento.

Figura 7.

Maqueta vista frontal

Figura 8.

Maqueta vista superior

Para capturar la imagen se utilizó una PiCamera de 5Mpx cuya resolución mínima es de 64x64 y la máxima es de 2592x1944 para fotos fijas y 1920x1080 para grabación de video.

La obtención de las imágenes se realizó con una Raspberry Zero W y Python en su versión 3.7.3 como lenguaje de programación. El primer paso es inicializar la cámara y seleccionar la resolución deseada, en este caso se utilizó 720x480, antes de iniciar el proceso de adquisición se usó un time.sleep( ) de 2 segundos, esto para permitirle al sensor de la cámara adaptarse a los niveles de luz. Posteriormente se captura la imagen, esta se almacena en el directorio donde se encuentra el código principal y se lee en escala de grises. Este proceso se realiza como se muestra a continuación:

Figura 9.

Inicialización de la cámara, captura y lectura de la imagen en escala de grises

```
#Inicialización de la cámara
camara = picamera.PiCamera()
camara.resolution = (720,480)
time.sleep(2)

while(True):
    #Captura de la imagen
    camara.capture("park.jpg")
    #Cargar la imagen en escala de grises
    img = cv2.imread("park.jpg",0)
```

Figura 10.

Imagen obtenida en escala de grises

3.2 Procesamiento y segmentación de imágenes

En una imagen digital es común la presencia de ruido, por ello se deben considerar técnicas para mejorar la imagen, para detectar la presencia de un vehículo en una zona de estacionamiento es necesario realzar ciertas características de interés de la imagen.

Con el propósito de hacer robusto el sistema ante variaciones de luminosidad, se creó una función llamada *ea_threshold( )* para obtener el umbral de binarización, esta función tiene como parámetro la imagen en escala de grises y se calcula la intensidad promedio de esta con la librería **numpy**. Se realizaron pruebas para encontrar el umbral que mejor se adaptara a las necesidades. Las ecuaciones obtenidas con el software MATLAB (Apéndice A.) son:

$$umbral = \begin{cases} x \leq 70, & 125{,}6 * 10^{-9}x^5 - 29{,}57 * 10^{-6}x^4 + 2{,}66 * 10^{-3}x^3 - 0.113x^2 + 2.32x - 14.66 \\ x > 70, & 12{,}43 * 10^{-6}x^3 - 3{,}46 * 10^{-3}x^2 + 0.361x - 5.47 \end{cases}$$

Figura 11.

Función para encontrar el umbral de binarización

```
#Función para definir el umbral
def ea_threshold(img):
    b=np.mean(np.array(img))    #Intensidad

    #Ecuacion encontrada:
    if b<=70:
        th=(0.000000125602004*(b**5)-0.000029575682471*(b**4)+0.002663197779916*(b**3)-0.113316655954267*(b**2)+2.320655889996000*b-14.661622800268075)
    else:
        th=(0.000012435492327*(b**3)-0.003461966496095*(b**2)+0.361394110382705*b-5.476943956185723)
    th=int(th)
    return th
```

Para remover el ruido conocido como “sal y pimienta” se utiliza el filtrado por la mediana mediante cv2.blur( ), posteriormente para resaltar los bordes de los vehículos se realiza la diferencia entre las operaciones morfológicas dilatación y erosión, este resultado se dilata para finalmente binarizarse. Estos procesos se realizan en la función *ea_preProcessing( )* que tiene

como parámetros de entrada la imagen en escala de grises y el umbral que retorna la función *ea_threshold( )*.

Figura 12.

Código de la función ea_preProcessing( )

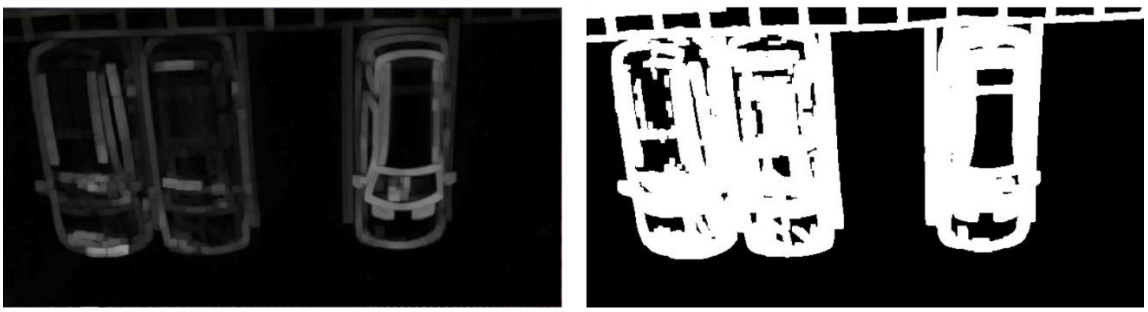
```
#Función de preprocesamiento
def ea_preProcessing(img,th):
    img_blur = cv2.blur(img,(3,3))  #Filtrado por la mediana
    kernel = np.array([ [1, 1, 1],
                        [1, 1, 1],
                        [1, 1, 1]],np.uint8)/9

    #Operaciones para encontrar y resaltar el borde
    borde = cv2.dilate(img_blur,kernel,iterations = 2) - cv2.erode(gris,kernel,iterations =2)
    dilate = cv2.dilate(borde,kernel,iterations=2)

    #Binarización
    _,binarizada = cv2.threshold(dilate,th,255,cv2.THRESH_BINARY)
    return binarizada
```

Figura 13.

Etapas de la función ea_preProcessing( )

a) Filtrado por la mediana

b) Operaciones morfológicas



c) Dilatación

d) Binarización

Con la binarización se busca que el fondo de la imagen sea negro y los carros se resalten en blanco, un umbral adecuado entrega una imagen como la que se mostró en la figura 13(d), donde se puede observar el contorno del vehículo. La función *ea_preProcessing( )* retorna dicha imagen.

Regiones de interés (ROI)

Para asignar los ROI a la imagen se utilizó la función cv2.warpPerspective( ), la cual permite realizar una transformación de perspectiva a una imagen.

Se aplicó cv2.warpPerspective( ) para elegir cada una de las plazas, en el código se incluyen las coordenadas tanto en X como en Y que representarán los vértices de la ROI , también el ancho y alto de la nueva imagen. En la Figura 14 se observa el proceso de asignación de las regiones de interés.

Figura 14.

Asignación de la región de interés para una plaza

```
#Tamaño de la nueva imagen:
w,h =100,170
ptsp = np.float32([[0,0],[w,0],[0,h],[w,h]])
#Regiones de interes
#Plaza 0
pts0 = np.float32([[19,36],[150,27],[47,295],[180,286]])
M0 = cv2.getPerspectiveTransform(pts0,ptsp)
ROE0 = cv2.warpPerspective(imgBina,M0,(w,h))
```

Con el propósito de visualizar las regiones de interés se aplicó el código de la figura 14 a la imagen original, este resultado se muestra en la figura 16.

Figura 15.

Zona de estacionamiento

Figura 16.

Regiones de interés de la imagen original

Para eliminar ruido y objetos no deseados se creó la función *ea_unwanted_areas( )* que tiene como parámetro la imagen binarizada.

Figura 17.

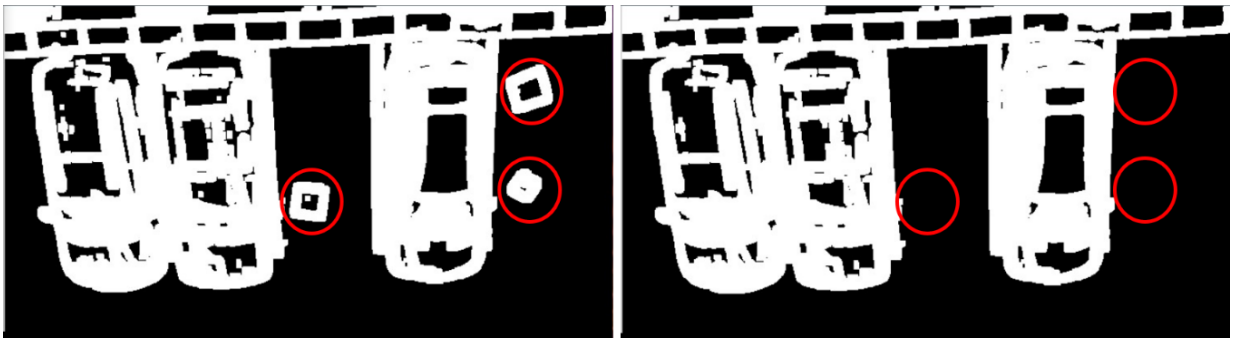
Función para eliminar áreas no deseadas

```
#Función para eliminar áreas no deseadas
def ea_unwanted_areas(img):
    nb_components,output,stats,centroids =cv2.connectedComponentsWithStats(img,connectivity=8)
    sizes = stats[1:,-1]; nb_components = nb_components-1
    min_size = 5589     #5589 equivale al 20% de la plaza de estacionamiento
    img_out = np.uint8(np.zeros((output.shape)))

    for i in range(0, nb_components):
        if sizes[i] >= min_size:
            img_out[output == i +1] = 255
    return img_out
```

Teniendo en cuenta que las zonas de estacionamiento están formadas en promedio por 27945 pixeles, se tomó el 20% que equivale a 5589 pixeles y las áreas de este valor o menos serán eliminadas. Las zonas con vehículos muestran más de 12000px por lo tanto no se verán afectadas al realizar este procedimiento.

Figura 18.

(a) Imagen binarizada, (b) Resultado de eliminar áreas no deseadas



Conociendo el proceso de selección de las regiones de interés, se extraen estas regiones de la imagen binarizada, con el fin de realizar los procesos restantes de forma individual. Esta imagen y sus ROI se muestran en la figura 19 y 20, respectivamente.

Figura 19.

Imagen binarizada

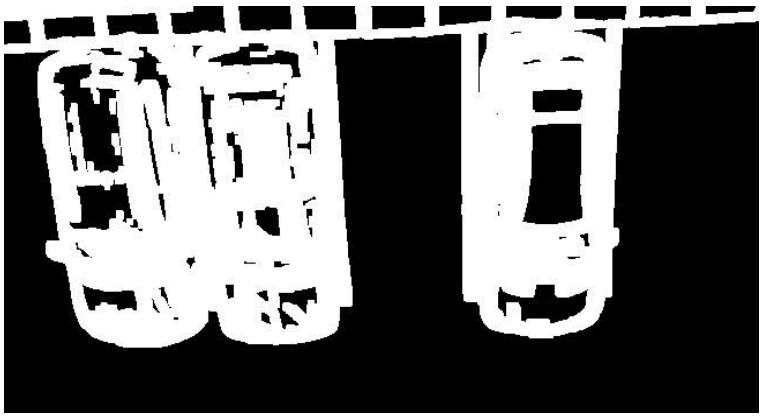
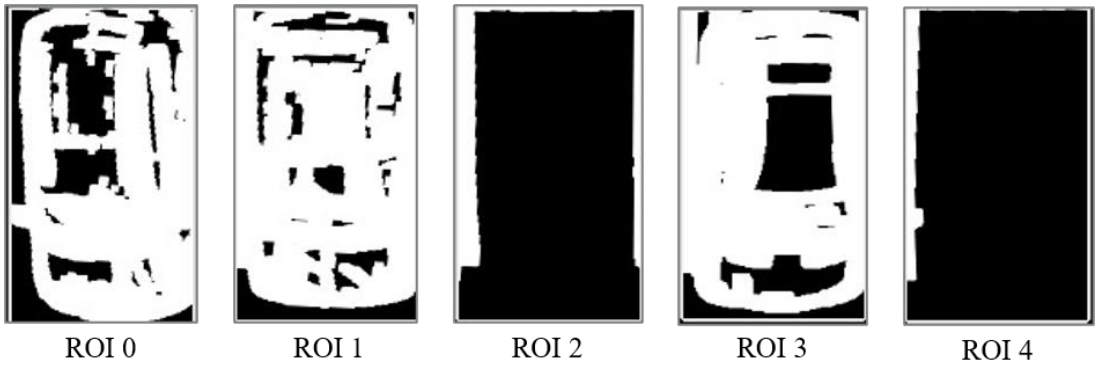


Figura 20.

Regiones de interés de la imagen binarizada



Si en una plaza hay un vehículo en su región de interés se muestra el contorno, lo que se busca es rellenarlo. Para este fin se creó la función *ea_fill_holes( )* que tiene como parámetro la imagen correspondiente a cada región de interés y sus dimensiones.

Figura 21.

Función ea_fill_holes( )

```python
#Función para rellenar
def ea_fill_holes(img,ancho,alto):
    im_th = img.copy()
    # Sumar rectángulo blanco a la imagen
    im_th = cv2.rectangle(im_th,(0,0),(ancho,alto),(255,255,255),3)
    im_th = cv2.bitwise_not(im_th)
    # Copiar la imagen umbralizada
    im_floodfill = im_th.copy()
    # Máscara usada para rellenar
    h, w = im_th.shape[:2]
    mask = np.zeros((h+2, w+2), np.uint8)
    # Rellenar desde el punto (0, 0)
    cv2.floodFill(im_floodfill, mask, (0,0), 255);
    # Invertir imagen
    im_floodfill_inv = cv2.bitwise_not(im_floodfill)
    # Sumar las dos imágenes
    im_out = im_th | im_floodfill_inv
    # Valor promedio
    mean = np.mean(np.array(im_out))
    return mean
```

Esta función rellena el vehículo en su totalidad, inicialmente dibuja un rectángulo blanco en cada región de interés, la invierte y mediante la función cv2.FloodFill( ) se cambia el fondo negro de la imagen por fondo blanco, esta se invierte y se suma con la imagen invertida antes de aplicar la función cv2.FloodFill( ). En la figura 22 se muestra este proceso para una plaza con vehículo y en la figura 23 para una plaza libre.

Figura 22.

Proceso de la función ea_fill_holes( ) para una plaza con vehículo

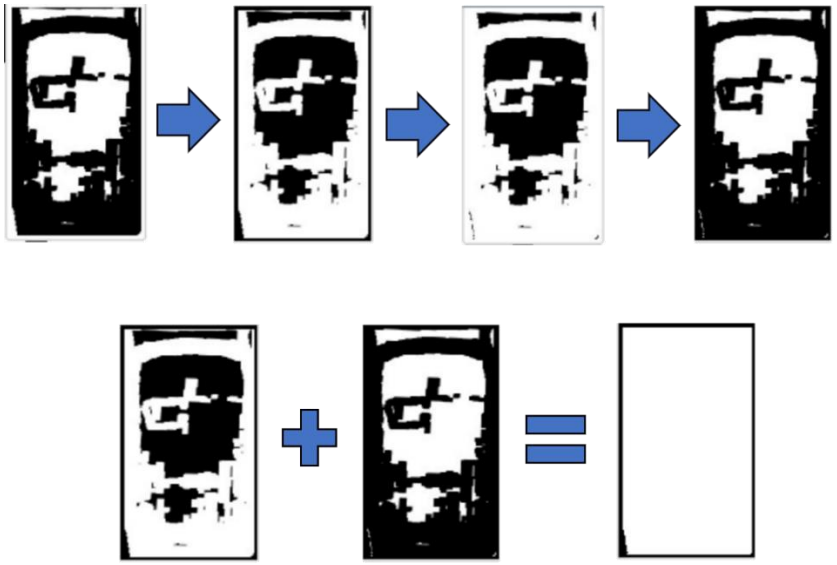
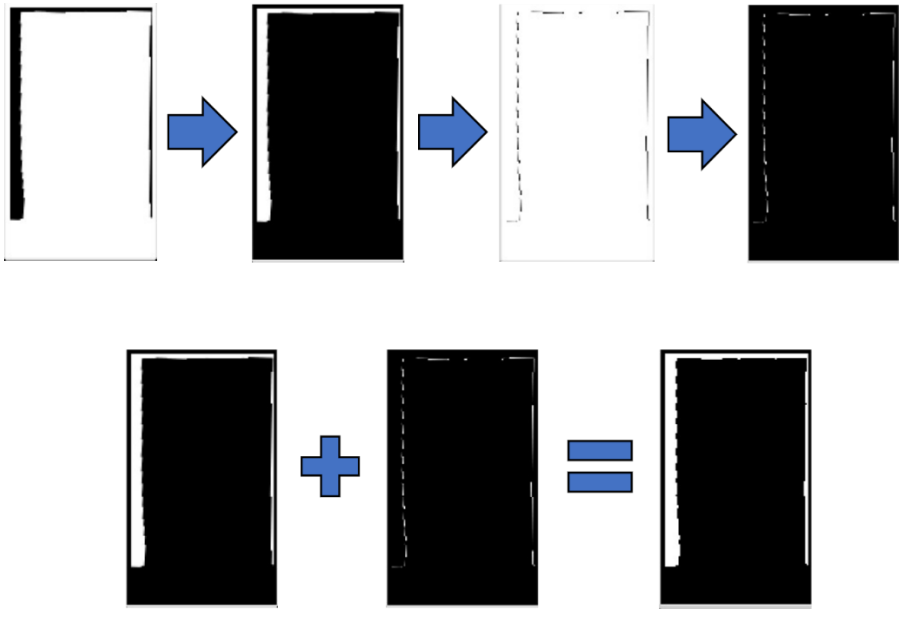


Figura 23.

Proceso de la función ea_fill_holes( ) para una plaza libre



Teniendo en cuenta que una imagen es una matriz con varios valores, se debe realizar un proceso que permita obtener un único valor numérico. Con el resultado obtenido anteriormente se calcula la media aritmética para cada región de interés mediante la librería **numpy** y su función **mean( )**. La función *ea_fill_holes( )* retorna dicho cálculo.

3.3 Detección de vehículos

El siguiente paso es detectar si una plaza de estacionamiento está libre u ocupada, En esta sección el algoritmo se basa en comparar la media aritmética de cada ROI con unos límites establecidos.

Para hallar los límites se realizaron pruebas basadas en calcular el promedio de cada región de interés, en presencia y ausencia de vehículo. Inicialmente, se tomaron objetos que representaran vehículos de diferentes dimensiones, teniendo en cuenta los más vendidos en Colombia (Revista Motor. , s.f.). Las ROI con un vehículo pequeño, como el Spark GT cuya longitud es 3.64m y su ancho es 1.597m (Medidas de coches Chevrolet., s.f.), arrojaron un promedio superior a 140. Finalmente, se calculó la media cuando no hay vehículo y el resultado no superó el valor de 30.

Una plaza se reconoce ocupada cuando el promedio supera el valor de 140, para esto se creó una función llamada *ea_color_text( )* que tiene como parámetro la media aritmética encontrada anteriormente y retorna un vector con dos cadenas de texto. Si el promedio sobrepasa este valor la función entrega ["red","Ocupado"]. Cuando un vehículo ocupa dos plazas, se obtiene que el promedio en cada ROI esta entre 60 y 140, para este caso la función retorna ["orange","Alerta"], si no hay vehículo la salida de la función será ["green","Libre"].

Figura 24.

Función para asignar color y texto

```
#Función para asignar el estado de cada plaza
def color_text(prom):
    if prom >= 140:
        return ["red","Ocupado"]
    elif  60< prom <140:
        return ["orange","Alerta"]
    else:
        return ["green","Libre"]
```

3.4 Base de datos y aplicación “web”

3.4.1 Creación de la base de datos

La base de datos utilizada es *firebase*. “Firebase Realtime Database” es una base de datos NoSQL alojada en la nube y tiene diferentes optimizaciones y funcionalidades en comparación con una base de datos relacional. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. La API de “Realtime Database” está diseñada para permitir solo operaciones que se puedan ejecutar rápidamente (Firebase, s.f.).

Para crear y modificar la base de datos desde un archivo python, se debe descargar la librería ***firebaseadmin*** y el enlace se realiza mediante:

Figura 25.

Enlace con la base de datos

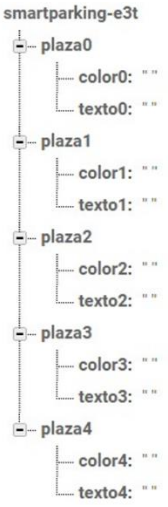
```
#Enlace con la base de datos
#Ruta de las redeciales json
PAHT_CRED = '/home/pi/Desktop/smart_parking/Json/cred.json'
#URL de la base de datos
URL_BD = 'https://smartparking-e3t.firebaseio.com/'

cred = credentials.Certificate(PAHT_CRED)
firebase_admin.initialize_app(cred, {
    'databaseURL': URL_BD
    })
```

El código utilizado para crear la base de datos se muestra en la figura 26(a) y el resultado se observa en la figura 26(b), donde se visualiza la información para las cinco plazas de estacionamiento.

Figura 26.

(a) Código para crear la base de datos, (b) Base de datos creada

```
# Referencias a cada una de las plazas en firebase
# Plaza 0
ref0 = db.reference('plaza0')
refcolor0 = ref0.child('color0')
reftexto0 = ref0.child('texto0')
# Plaza 1
ref1 = db.reference('plaza1')
refcolor1 = ref1.child('color1')
reftexto1 = ref1.child('texto1')
# Plaza 2
ref2 = db.reference('plaza2')
refcolor2 = ref2.child('color2')
reftexto2 = ref2.child('texto2')
# Plaza 3
ref3 = db.reference('plaza3')
refcolor3 = ref3.child('color3')
reftexto3 = ref3.child('texto3')
# Plaza 4
ref4 = db.reference('plaza4')
refcolor4 = ref4.child('color4')
reftexto4 = ref4.child('texto4')
```



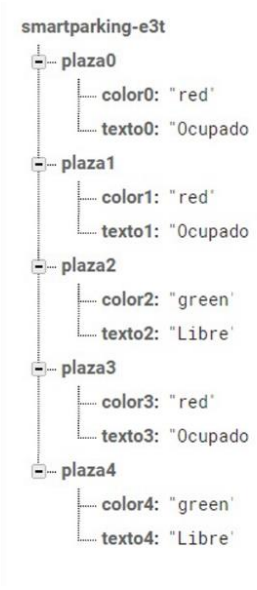
3.4.2 Envío de información a la base de datos

Se creó una función *ea_up( )* por cada plaza para enviar a la base de datos la información correspondiente, esta función tiene como parámetro el promedio de cada región de interés y ejecuta la función *ea_color_text( )* la cual asigna el estado de la plaza mediante el color y el texto.

Figura 27.

(a) Funciones para enviar información, (b) Base de datos

```
#Funciones para enviar la información a la base de datos
#Plaza 0
def ea_up0(i):
    refcolor0.set(color_text(i)[0])
    reftexto0.set(color_text(i)[1])
    return
#Plaza 1
def ea_up1(i):
    refcolor1.set(color_text(i)[0])
    reftexto1.set(color_text(i)[1])
    return
#Plaza 2
def ea_up2(i):
    refcolor2.set(color_text(i)[0])
    reftexto2.set(color_text(i)[1])
    return
#Plaza 3
def ea_up3(i):
    refcolor3.set(color_text(i)[0])
    reftexto3.set(color_text(i)[1])
    return
#Plaza 4
def ea_up4(i):
    refcolor4.set(color_text(i)[0])
    reftexto4.set(color_text(i)[1])
    return
```



Con el fin de minimizar el tiempo de envío de información, se realizó programación por hilos para ejecutar las funciones *ea_up( )*, un hilo es un proceso del sistema operativo que permite realizar varias tareas al mismo tiempo. Los hilos comparten memoria y recursos pero ocupan una dirección diferente en la memoria (Fernandez, Hilos en Python. , 2019), este procedimiento se muestra a continuación.

Figura 28.

Programación por hilos

```
#Definición de los hilos
t0 = threading.Thread(name = "hilo_0",target=ea_up0 ,args=(ea_fill_holes(ROE0,w,h),))
t1 = threading.Thread(name = "hilo_1",target=ea_up1 ,args=(ea_fill_holes(ROE1,w,h),))
t2 = threading.Thread(name = "hilo_2",target=ea_up2 ,args=(ea_fill_holes(ROE2,w,h),))
t3 = threading.Thread(name = "hilo_3",target=ea_up3 ,args=(ea_fill_holes(ROE3,w,h),))
t4 = threading.Thread(name = "hilo_4",target=ea_up4 ,args=(ea_fill_holes(ROE4,w,h),))
#Ejecución de los hilos
t0.start()
t1.start()
t2.start()
t3.start()
t4.start()
```

3.4.3 Aplicación "web"

Para definir el sentido y la estructura del contenido de la aplicación "web" se usó HTML y para describir la apariencia se utilizó CSS. Ver Apéndice C. Para comunicar la base de datos con la aplicación "web" se utilizó Java Script y se muestra a continuación el código para una sola plaza:

Figura 29.

Comunicación entre la base de datos y la aplicación para plaza 0

```
//plaza0
//Enlace con el Id de HTML
var elem0 = document.getElementById('plaza0');
//Enlace a la base de datos en firebase
var plaza0Ref = firebase.database().ref('plaza0')
var color0Ref = plaza0Ref.child('color0')
var texto0Ref = plaza0Ref.child('texto0')
//cambia el texto en el ID seleccionado
texto0Ref.on('value', snap => elem0.innerHTML = snap.val());
//cambia el color en el ID seleccionado
color0Ref.on('value', snap => elem0.style.backgroundColor = snap.val());
```

La aplicación se subió al *hosting* de *firebase*, "*firebase hosting*" permite implementar aplicaciones "*web*" y entregar contenido dinámico y estático en una CDN (red de distribución de contenidos) global rápidamente (Firebase, s.f.).

La URL para acceder a la aplicación es: https://smartparking-e3t.web.app/

Figura 30.

Vehículos en la maqueta

Figura 31.

Aplicación "web"

3.5 Automatización

Con el propósito de ejecutar el programa en el momento de encender la Raspberry, se instaló **supervisor,** que es un sistema cliente/servidor que permite a sus usuarios monitorear y controlar un número de procesos en sistemas operativos typo UNIX (Medina, s.f.).

Se creó un archivo de configuración (Figura 32), que contiene el nombre del programa, el comando que permite ejecutarlo, la ruta completa del archivo y el comportamiento automatizado que deberá mantener **supervisor**.

Figura 32.

Archivo de configuración

4. Pruebas

Para verificar el funcionamiento adecuado del sistema, se realizaron diferentes pruebas con el propósito de encontrar la cantidad de aciertos o fallos al detectar los diferentes estados, (libre, alarma y ocupado), para variaciones en los niveles de iluminación.

Se realizaron las siguientes pruebas:

- Vehículos estacionados de manera adecuada. Apéndice D
- Vehículos que ocupan dos zonas de estacionamiento. Apéndice E
- Zonas de estacionamiento con objetos diferentes a un carro. Apéndice F

- Zona de estacionamiento real. Apéndice G

La primera prueba se realizó para intensidades en el rango de 23 a 53 y los resultados son los siguientes:

Tabla 1.

Pruebas con intensidad menor a 70

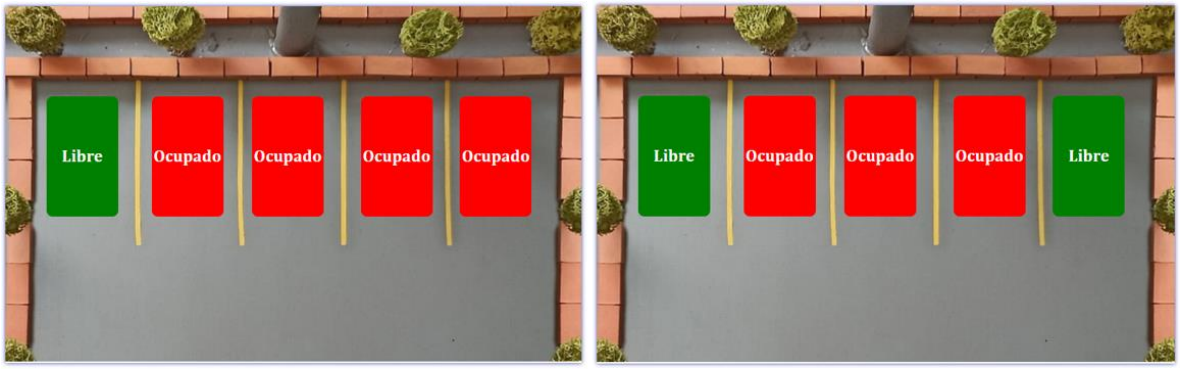
Un vehículo	Dos vehículos	Tres vehículos	Cuatro vehículos	Cinco vehículos
100%	100%	100%	100%	100%

Figura 33.

(a) Intensidad de 23, (b) Intensidad de 53

Figura 34.

Aplicación "web"



La siguiente prueba se realizó para intensidades en el rango de 90 a 120 y los resultados son los siguientes:

Tabla 2.

Pruebas con intensidad mayor a 70

Un vehículo	Dos vehículos	Tres vehículos	Cuatro vehículos	Cinco vehículos
100%	100%	100%	100%	100%

Figura 35.

(a) Intensidad de 90, (b) Intensidad de 120

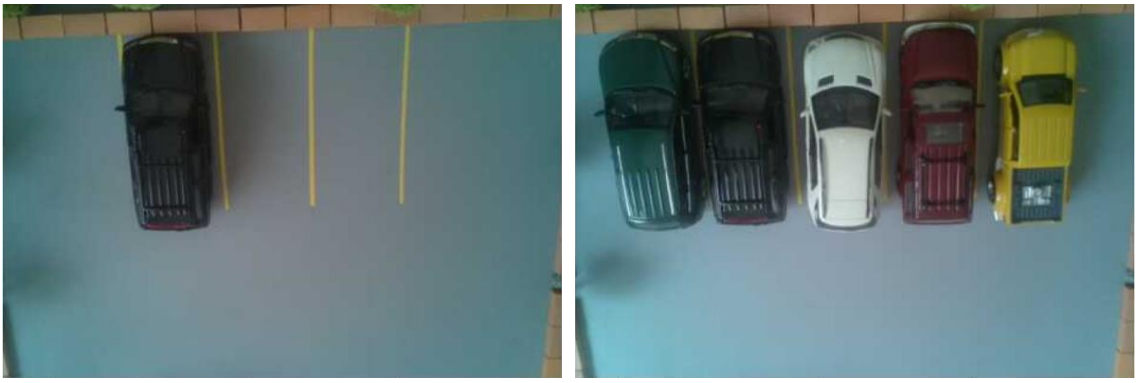
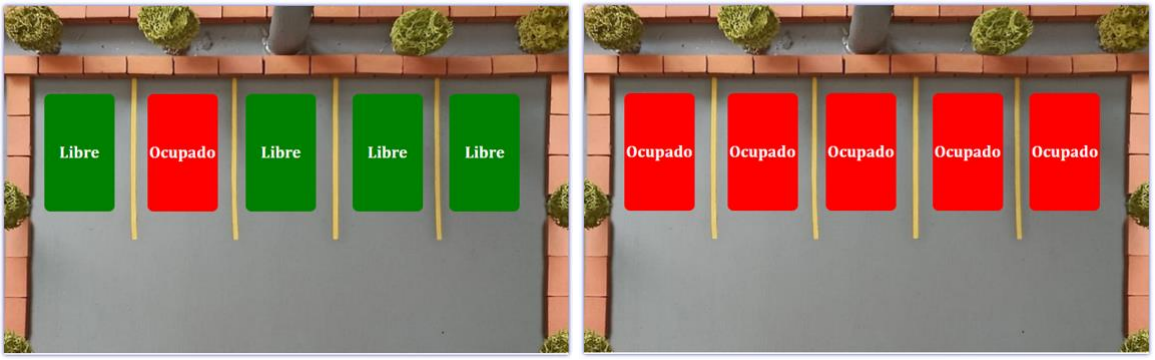


Figura 36.

Aplicación "web"



Pruebas para el estado “Alerta”

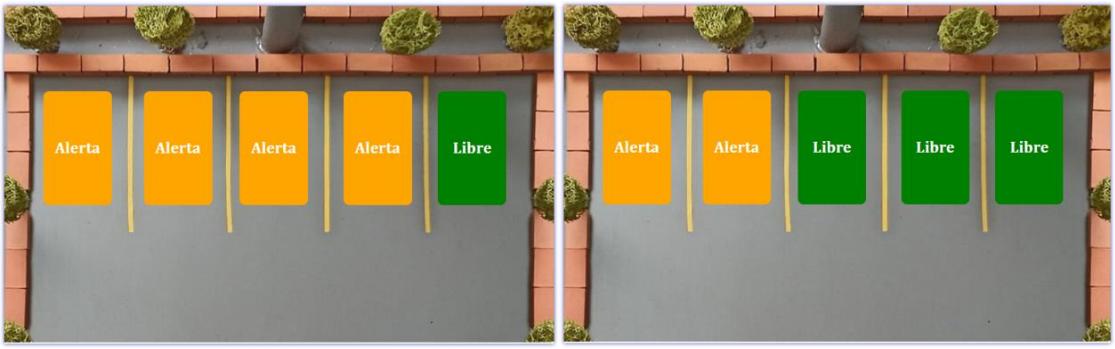
Se ubicaron vehículos de manera que ocupen dos plazas de estacionamiento y se realizaron pruebas para diferentes intensidades. En este caso, la intensidad está entre 76 y 84.

Figura 37.

(a) Intensidad de 76, (b) Intensidad de 84

Figura 38.

Aplicación "web"



Para el caso de la figura 37(a) y para el de la figura 37(b) el acierto fue de 100%

De igual forma se realizaron pruebas con vehículos mal estacionados para intensidades entre 43 y 50

Figura 39.

(a) Intensidad de 43, (b) Intensidad de 50

Figura 40.

Aplicación "web"

Para el caso de la figura 39(a) y para el de la figura 39(b) el acierto fue de 100%

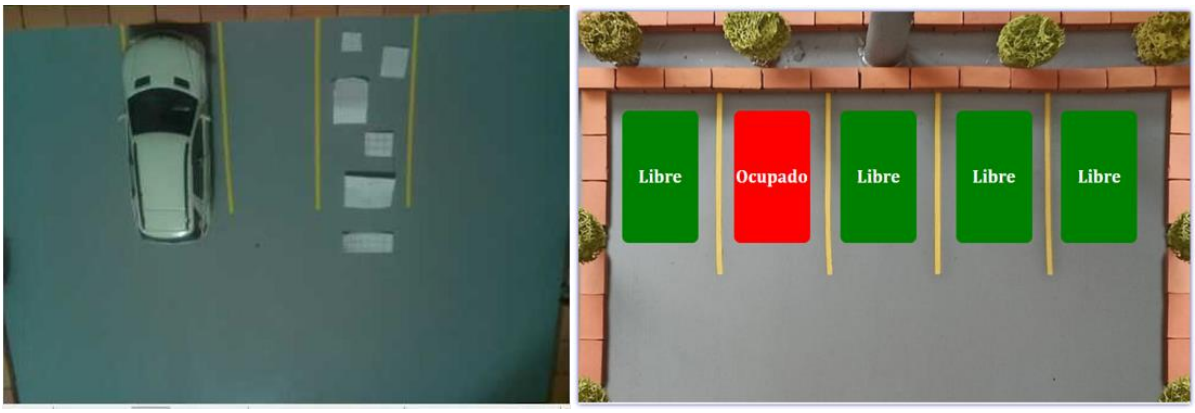
Pruebas con objetos diferentes a un vehículo

Cuando hay objetos diferentes a un vehículo en las zonas de estacionamiento la aplicación debe mostrar que la zona está disponible, se ubicaron papelitos distribuidos en la zona de estacionamiento y se realizaron pruebas para diferentes intensidades.

Para intensidades entre 84 y 86:

Figura 41.

Prueba con objetos diferentes a un vehículo



Para intensidades entre 32 y 36:

Figura 42.

Prueba con objetos diferentes a un vehículo

Para el caso de la figura 41 y para el de la figura 42 el acierto fue de 100%

5. Conclusiones

Las imágenes se capturaron mediante una PiCamera y el uso de Python como lenguaje de programación, el cual facilitó su obtención, ya que contiene librerías especializadas para la adquisición y procesamiento de imágenes.

Durante el desarrollo del algoritmo, se concluyó que para el correcto funcionamiento del prototipo es esencial seleccionar adecuadamente las regiones de interés (ROI), puesto que se debe garantizar que el vehículo se capture en su totalidad.

La resolución seleccionada permitió obtener porcentajes de acierto elevados, con una resolución menor se dificulta la elección de las regiones de interés, se pierde exactitud al reducir la calidad de la imagen y el tiempo de procesamiento no disminuye considerablemente. Una resolución mayor aumenta el tiempo de ejecución y no muestra gran variación en cuanto a los resultados.

El tamaño elegido para las regiones de interés permitió extraer la información requerida de cada zona de estacionamiento, una región de menor tamaño podría ocasionar errores en los resultados, por el contrario, una más amplia podría aumentar el tiempo de procesamiento.

Se realizaron pruebas con un video tomado de una zona de estacionamiento real, para esto fue necesario adaptar las regiones de interés y realizar el procedimiento ya mencionado, los resultados fueron satisfactorios ya que fue posible identificar el estado de cada zona.

Con base al alto porcentaje de acierto obtenido al realizar las pruebas en varias ocasiones, se ha logrado evidenciar la eficacia del algoritmo desarrollado.

Se desarrolló la aplicación “web” utilizando herramientas como HTML, CSS, JavaScript y *firebase*. Esta aplicación permite visualizar la información sobre la disponibilidad de estacionamiento a cualquier usuario que necesite realizar una consulta.

La aplicación es de fácil acceso, es decir, los usuarios no deben registrarse para visualizar el contenido, ya que desde el momento de ingreso se puede observar la disponibilidad de zonas.

En este tipo de aplicaciones el uso de la visión artificial es muy viable, dado que reduce la cantidad de sensores, lo cual se ve reflejado en costos y en reducción de consumo energético.

6. Recomendaciones

Como siguiente paso a este proyecto se recomienda identificar un vehículo en movimiento e informar si esta ingresando o saliendo de la plaza de estacionamiento.

Se recomienda buscar alternativas de microprocesadores con el fin de disminuir el costo del prototipo, en el mercado se pueden encontrar placas de desarrollo como Banana pi M2 zero, Orange pi one, Sipeed MAIX, entre otras.

Referencias Bibliográficas

Alcaldía de Bucaramanga, Secretaría de planeación. (2014). *Plan de Ordenamiento Territorial de Bucaramanga (Artículo 360o).* Obtenido de https://www.bucaramanga.gov.co/la-ruta/download/acuerdo/POT-2014-2027.pdf

Aplicaciones para python. (s.f.). *Python.* Obtenido de https://www.python.org/about/apps/

Convertir imagen a imagen binaria, basada en umbral - MATLAB im2bw. (s.f.). *MATLAB - El lenguaje del cálculo técnico.* Obtenido de https://la.mathworks.com/help/images/ref/im2bw.html

Esqueda, J. J. (noviembre de 2002). *Fundamentos de Procesamiento de Imágenes. Universidad Autónoma de Baja California.* Obtenido de https://www.academia.edu/16801512/Fundamentos_de_procesamiento_de_im%C3%A1genes_digitales

Fernandez, R. (2019). *Hilos en Python.* . Obtenido de https://unipython.com/hilos-python/

Fernandez, R. (30 de agosto de 2019). *Umbralización de una Imagen.* Obtenido de https://unipython.com/umbralizacion-una-imagen/

Filtrado de mediana 2D - MATLAB medfilt2. (s.f.). *MATLAB - El lenguaje del cálculo técnico.* Obtenido de https://la.mathworks.com/help/images/ref/medfilt2.html

Firebase. (s.f.). *Firebase Realtime Database.* Obtenido de https://firebase.google.com/docs/database?hl=es-419

Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing (3th Edition).* Obtenido de http://sdeuoc.ac.in/sites/default/files/sde_videos/Digital%20Image%20Processing%203rd%20ed.%20-%20R.%20Gonzalez%2C%20R.%20Woods-ilovepdf-compressed.pdf

Medidas de coches Chevrolet. (s.f.). *Medidas y dimensiones de coches.* Obtenido de https://www.medidasdecoches.com/medidas-coches-chevrolet.html

Medina, L. A. (s.f.). *Como Instalar supervisor en Ubuntu y Debian. Como instalar Linux.* Obtenido de https://www.comoinstalarlinux.com/como-instalar-supervisor-en-ubuntu-y-debian/

Mejia, J. R. (2005). *Procesamiento Digital de Imágenes.* Obtenido de http://laurence.com.ar/artes/comun/Apuntes%20procesamiento%20digital%20de%20imagenes.pdf

Mordvintsev, A., & Abid, K. (noviembre de 2017). *OpenCV-Python Tutorials Documentation.* . Obtenido de https://readthedocs.org/projects/opencv-python-tutroals/downloads/pdf/latest/

Mordvintsev, A., & Rahman, A. (s.f.). *Introducción a OpenCV. OpenCV.* Obtenido de https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

News America Digital. (16 de enero de 2019). *Plataformas IoT: Qué son y Por qué implementarlas.* Obtenido de https://news.america-digital.com/iot-plataformas-implementacion-exitosa/

Oracle. (s.f.). *¿Qué es IoT?* Obtenido de https://www.oracle.com/co/internet-of-things/what-is-iot.html

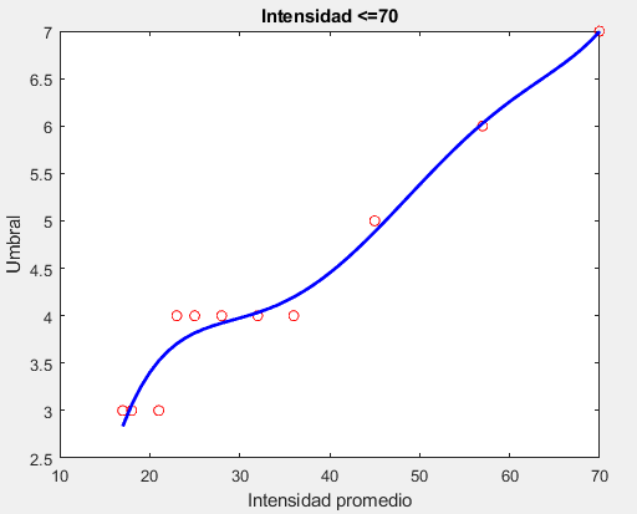
Revista Motor. . (s.f.). *Top 10 de los carros más vendidos en Colombia.* Obtenido de https://www.motor.com.co/fotos/top-10-carros-vendidos-colombia/27469

APéndices

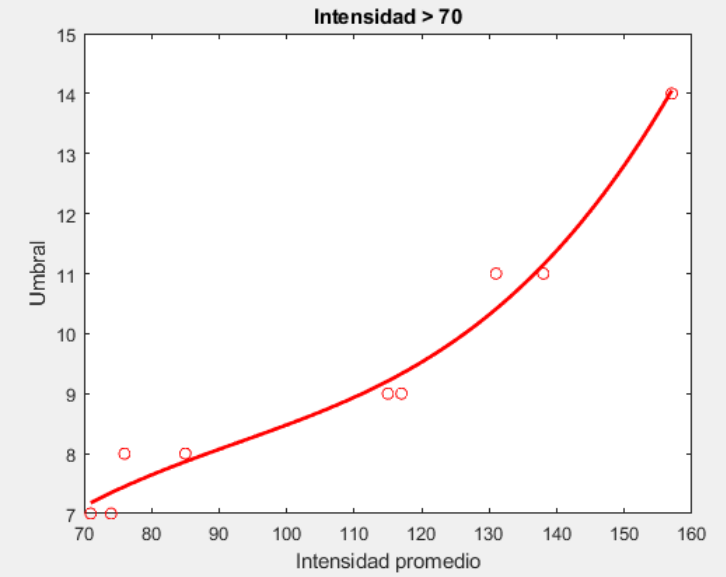
Apéndice A. Código Matlab para encontrar las ecuaciones del umbral.

Se hallaron polinomios de diferente orden para encontrar el que mejor se adaptara a los puntos requeridos, este polinomio no debe ser exacto, solo una aproximación a los puntos, ya que el umbral puede variar entre el número anterior y el siguiente.

```
#Pruebas realizadas intensidad <=70

Intensidad_a=[17,18,21,23,25,28,32,36,45,57,70];
Umbral_a=[3,3,3,4,4,4,4,4,5,6,7];

p=polyfit(Intensidad_a, umbral_a,5)
xp=17:1:70;
vector_1=polyval(p,xp);

plot(Intensidad_a, umbral_a,'ro')
hold on
plot(xp,vector_1,'b','linewidth',2)
title('Intensidad <=70')
xlabel('Intensidad promedio')
ylabel('Umbral')
```




```
#Pruebas realizadas intensidad >70

Intensidad_b=[71,74,76,85,115,117,131,138,157];
Umbral_b=[7,7,8,8,9,9,11,11,14];

p2=polyfit(Intensidad_b, Umbral_b,3)
xp2=71:1:157;
vector_2=polyval(p2,xp2);
figure
plot(Intensidad_b, Umbral_b,'ro')
hold on
plot(xp2,vector_2,'r','linewidth',2)
title('Intensidad > 70')
xlabel('Intensidad promedio')
ylabel('Umbral')
```


Intensidad > 70
Umbral
Intensidad promedio
15
14
13
12
11
10
9
8
7
70
80
90
100
110
120
130
140
150
160

Apéndice B. Código Principal

```
import cv2
import picamera
import numpy as np
import time
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
import logging
import threading

#Enlace con la base de datos
#Ruta de las redeciales json
PAHT_CRED = '/home/pi/Desktop/smart_parking/Json/cred.json'
#URL de la base de datos
URL_BD = 'https://smartparking-e3t.firebaseio.com/'

cred = credentials.Certificate(PAHT_CRED)
firebase_admin.initialize_app(cred, {
    'databaseURL': URL_BD
    })

# Referencias a cada una de las plazas en firebase

# Plaza 0
ref0 = db.reference('plaza0')
refcolor0 = ref0.child('color0')
```

```
reftexto0 = ref0.child('texto0')
# Plaza 1
ref1 = db.reference('plaza1')
refcolor1 = ref1.child('color1')
reftexto1 = ref1.child('texto1')
# Plaza 2
ref2 = db.reference('plaza2')
refcolor2 = ref2.child('color2')
reftexto2 = ref2.child('texto2')
# Plaza 3
ref3 = db.reference('plaza3')
refcolor3 = ref3.child('color3')
reftexto3 = ref3.child('texto3')
# Plaza 4
ref4 = db.reference('plaza4')
refcolor4 = ref4.child('color4')
reftexto4 = ref4.child('texto4')

#### FUNCIONES

#Función para definir el umbral
def ea_threshold(img):
    b=np.mean(np.array(img))    #Intensidad

    #Ecuacion encontrada:
    if b<=70:
        th=(0.000000125602004*(b**5)-0.000029575682471*(b**4)+
        0.002663197779916 *(b**3)-0.113316655954267*(b**2)+
         2.320655889996000*b-14.661622800268075)
    else:
        th=(0.000012435492327*(b**3)-0.003461966496095*(b**2)+
```

```
        0.361394110382705*b-5.476943956185723)
    th=int(th)
    return th

#Función de preprocesamiento
def ea_preProcessing(img,th):
    img_blur = cv2.blur(img,(3,3))  #Filtrado por la mediana
    kernel = np.array([ [1, 1, 1],
                        [1, 1, 1],
                        [1, 1, 1]],np.uint8)/9

    #Operaciones para encontrar y resaltar el borde
    borde = cv2.dilate(img_blur,kernel,iterations = 2) -
    cv2.erode(gris,kernel,iterations =2)
    dilate = cv2.dilate(borde,kernel,iterations=2)

    #Binarización
    _,binarizada = cv2.threshold(dilate,th,255,cv2.THRESH_BINARY)
    return binarizada



#Función para eliminar áreas no deseadas
def ea_unwanted_areas(img):
    nb_components,output,stats,centroids =
    cv2.connectedComponentsWithStats(img,connectivity=8)
    sizes = stats[1:,-1]; nb_components = nb_components-1
    min_size = 5589     #Equivale al 20% de la plaza de estacionamiento
    img_out = np.uint8(np.zeros((output.shape)))

    for i in range(0, nb_components):
        if sizes[i] >= min_size:
```

```
            img_out[output == i +1] = 255
    return img_out



#Función para rellenar
def ea_fill_holes(img,ancho,alto):
    im_th = img.copy()
    # Sumar rectángulo blanco a la imagen
    im_th = cv2.rectangle(im_th,(0,0),(ancho,alto),(255,255,255),3)
    im_th = cv2.bitwise_not(im_th)
    # Copiar la imagen umbralizada
    im_floodfill = im_th.copy()
    # Máscara usada para rellenar
    h, w = im_th.shape[:2]
    mask = np.zeros((h+2, w+2), np.uint8)
    # Rellenar desde el punto (0, 0)
    cv2.floodFill(im_floodfill, mask, (0,0), 255);
    # Invertir imagen
    im_floodfill_inv = cv2.bitwise_not(im_floodfill)
    # Sumar las dos imágenes
    im_out = im_th | im_floodfill_inv
    # Valor promedio
    mean = np.mean(np.array(im_out))
    return mean



#Función para asignar el estado de cada plaza
def ea_color_text(prom):
    if prom >= 140:
        return ["red","Ocupado"]
    elif  60< prom <140:
```

```
        return ["orange","Alerta"]
    else:
        return ["green","Libre"]


#Funciones para enviar la información a la base de datos
#Plaza 0
def ea_up0(i):
    refcolor0.set(color_text(i)[0])
    reftexto0.set(color_text(i)[1])
    return
#Plaza 1
def ea_up1(i):
    refcolor1.set(color_text(i)[0])
    reftexto1.set(color_text(i)[1])
    return
#Plaza 2
def ea_up2(i):
    refcolor2.set(color_text(i)[0])
    reftexto2.set(color_text(i)[1])
    return
#Plaza 3
def ea_up3(i):
    refcolor3.set(color_text(i)[0])
    reftexto3.set(color_text(i)[1])
    return
#Plaza 4
def ea_up4(i):
    refcolor4.set(color_text(i)[0])
    reftexto4.set(color_text(i)[1])
    return
```

```
#Tamaño de la nueva imagen:
w,h =100,170
ptsp = np.float32([[0,0],[w,0],[0,h],[w,h]])
#Regiones de interés
#Plaza 0
pts0 = np.float32([[19,36],[150,27],[47,295],[180,286]])
M0 = cv2.getPerspectiveTransform(pts0,ptsp)
#Plaza 1
pts1 = np.float32([[155,28],[278,20],[185,292],[295,288]])
M1 = cv2.getPerspectiveTransform(pts1,ptsp)
#Plaza 2
pts2 = np.float32([[278,20],[402,16],[297,289],[408,292]])
M2 = cv2.getPerspectiveTransform(pts2,ptsp)
#Plaza 3
pts3 = np.float32([[404,14],[525,11],[406,282],[518,276]])
M3 = cv2.getPerspectiveTransform(pts3,ptsp)
#Plaza 4
pts4 = np.float32([[530,11],[648,6],[523,278],[648,278]])
M4 = cv2.getPerspectiveTransform(pts4,ptsp)

### Inicio del proceso ###
#Inicialización de la cámara
camara = picamera.PiCamera()
camara.resolution = (720,480)
time.sleep(2)

while(True):
    #Captura de la imagen
    camara.capture("park.jpg")
    #Cargar la imagen en escala de grises
    img = cv2.imread("park.jpg",0)
```

```
    #Obtener el umbral
    th = ea_threshold(img)
    #Preprocesamiento
    imgBina = ea_preProcessing(img,th)
    #Eliminar areas no deseadas
    imgBina = ea_unwanted_areas(imgBina)
    imgBina = cv2.bitwise_not(imgBina)
    #Separar en las regiones de interez
    ROE0 = cv2.warpPerspective(imgBina,M0,(w,h))
    ROE1 = cv2.warpPerspective(imgBina,M1,(w,h))
    ROE2 = cv2.warpPerspective(imgBina,M2,(w,h))
    ROE3 = cv2.warpPerspective(imgBina,M3,(w,h))
    ROE4 = cv2.warpPerspective(imgBina,M4,(w,h))

    #Definición de los hilos
    t0 = threading.Thread(name = "hilo_0",target=ea_up0 ,
    args=(ea_fill_holes(ROE0,w,h),))
    t1 = threading.Thread(name = "hilo_1",target=ea_up1 ,
    args=(ea_fill_holes(ROE1,w,h),))
    t2 = threading.Thread(name = "hilo_2",target=ea_up2 ,
    args=(ea_fill_holes(ROE2,w,h),))
    t3 = threading.Thread(name = "hilo_3",target=ea_up3 ,
    args=(ea_fill_holes(ROE3,w,h),))
    t4 = threading.Thread(name = "hilo_4",target=ea_up4 ,
    args=(ea_fill_holes(ROE4,w,h),))
    #Ejecución de los hilos
    t0.start()
    t1.start()
    t2.start()
    t3.start()
    t4.start()
```

Apéndice C. Aplicación "WEB"

Archivo HTML

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>Smart Parking</title>
                <script src="https://www.gstatic.com/firebasejs/7.14.3/firebase-app.js"></script>
                <script src="https://www.gstatic.com/firebasejs/7.14.3/firebase-
                database.js"></script>
                <script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-
                QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>
                <link href="static/css/estilo.css"  rel="stylesheet" type="text/css">
                </head>
  <body>
                <div class ="contenedor">
                        <header>
                        <img class="izquierda" src="static/img/uis.jpg" > <img class="derecha"
                         align="right" src="static/img/logoe3t.png" height="70"> <p
                         class= "texto">Smart Parking</p>
                        </header>
                        <div class="wrap">
                                <div class="widget">
                                        <div class="fecha">
                                                <p id="diaSemana" ></p>
                                                <p id="dia" ></p>
                                                <p>de </p>
                                                <p id="mes" ></p>
                                                <p>del </p>
                                                <p id="year" ></p>
                                        </div>
                                        <div class="reloj">
```

```
                                <p id="horas" ></p>
                                <p>:</p>
                                <p id="minutos"></p>
                                <p>:</p>
                                <div class="caja-segundos">
                                        <p id="ampm" class="ampm"></p>
                                        <p id="segundos" class="segundos"></p>
                                </div>
                        </div>
                </div>
        </div>
<section>
                        <img src="static/img/zona.jpg">
                        <div id = "plaza0"  class = "plaza cero"> </div>
                        <div id = "plaza1"  class = "plaza uno"> </div>
                        <div id = "plaza2"  class = "plaza dos"> </div>
                        <div id = "plaza3"  class = "plaza tres">  </div>
                        <div id = "plaza4"  class = "plaza cuatro"> </div>
</section>

        <script src ="javaScrip/index.js">              </script>
        <script src ="javaScrip/reloj.js">              </script>

        <footer>
                <h1>
                Proyecto de grado<br>
                Realizado por: Adriana Suárez & Eder Sánchez
                <p> UNIVERSIDAD INDUSTRIAL DE SANTANDER<br>
                        FACULTAD DE INGENIERÍAS FISICOMECÁNICAS<br>
                        ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE
                        TELECOMUNICACIONES<br>
                        BUCARAMANGA<br>
                        2020</p>
                </h1>
        </footer>
</div>
```

```
  </body>
</html>
```

Archivo Java Script para el enlace con la base de datos

```
    // Configuración app web firebase
  var firebaseConfig = {
    apiKey: "AIzaSyDDBrjPxLpjoXJ5sb-9JiBm7d2B8NU6un8",
    authDomain: "smartparking-e3t.firebaseapp.com",
    databaseURL: "https://smartparking-e3t.firebaseio.com",
    projectId: "smartparking-e3t",
    storageBucket: "smartparking-e3t.appspot.com",
    messagingSenderId: "784684163945",
    appId: "1:784684163945:web:8df29895f84ae15b2293f8",
    measurementId: "G-133NHP910L"
  };
  // Inicializar Firebase
  firebase.initializeApp(firebaseConfig);


      //plaza0
      //Enlace con el Id de HTML
      var elem0 = document.getElementById('plaza0');
      //Enlace a la base de datos en firebase
      var plaza0Ref = firebase.database().ref('plaza0')
      var color0Ref = plaza0Ref.child('color0')
      var texto0Ref = plaza0Ref.child('texto0')
      //cambia el texto en el ID seleccionado
      texto0Ref.on('value', snap => elem0.innerHTML = snap.val());
      //cambia el color en el ID seleccionado
      color0Ref.on('value', snap => elem0.style.backgroundColor = snap.val());


      //plaza1
```

```
var elem1 = document.getElementById('plaza1');
var plaza1Ref = firebase.database().ref('plaza1')
var color1Ref = plaza1Ref.child('color1')
var texto1Ref = plaza1Ref.child('texto1')
texto1Ref.on('value', snap => elem1.innerHTML = snap.val());
color1Ref.on('value', snap => elem1.style.backgroundColor = snap.val());

//plaza2
var elem2 = document.getElementById('plaza2');
var plaza2Ref = firebase.database().ref('plaza2')
var color2Ref = plaza2Ref.child('color2')
var texto2Ref = plaza2Ref.child('texto2')
texto2Ref.on('value', snap => elem2.innerHTML = snap.val());
color2Ref.on('value', snap => elem2.style.backgroundColor = snap.val());

//plaza3
var elem3 = document.getElementById('plaza3');
var plaza3Ref = firebase.database().ref('plaza3')
var color3Ref = plaza3Ref.child('color3')
var texto3Ref = plaza3Ref.child('texto3')
texto3Ref.on('value', snap => elem3.innerHTML = snap.val());
color3Ref.on('value', snap => elem3.style.backgroundColor = snap.val());

//plaza4
var elem4 = document.getElementById('plaza4');
var plaza4Ref = firebase.database().ref('plaza4')
var color4Ref = plaza4Ref.child('color4')
var texto4Ref = plaza4Ref.child('texto4')
texto4Ref.on('value', snap => elem4.innerHTML = snap.val());
color4Ref.on('value', snap => elem4.style.backgroundColor = snap.val());
```

Archivo Java Script para el reloj

```
(function(){

	var actualizarHora = function(){

		// Obtenemos la fecha actual, incluyendo las horas, minutos, segundos, dia de la semana, dia del mes, mes y año;

		var fecha = new Date(),

			horas = fecha.getHours(),

			ampm,

			minutos = fecha.getMinutes(),

			segundos = fecha.getSeconds(),

			diaSemana = fecha.getDay(),

			dia = fecha.getDate(),

			mes = fecha.getMonth(),

			year = fecha.getFullYear();


		// Accedemos a los elementos del DOM para agregar mas adelante sus correspondientes valores

		var pHoras = document.getElementById('horas'),

			pAMPM = document.getElementById('ampm'),

			pMinutos = document.getElementById('minutos'),

			pSegundos = document.getElementById('segundos'),

			pDiaSemana = document.getElementById('diaSemana'),

			pDia = document.getElementById('dia'),

			pMes = document.getElementById('mes'),

			pYear = document.getElementById('year');


		// Obtenemos el dia se la semana y lo mostramos

		var semana = ['Domingo', 'Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sabado'];

		pDiaSemana.textContent = semana[diaSemana];


		// Obtenemos el dia del mes
```

```
            pDia.textContent = dia;


            // Obtenemos el Mes y año y lo mostramos
            var meses = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio',
'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre']
            pMes.textContent = meses[mes];
            pYear.textContent = year;


            // Cambiamos las hora de 24 a 12 horas y establecemos si es AM o PM
            if (horas >= 12) {
                  horas = horas - 12;
                  ampm = 'PM';
            } else {
                  ampm = 'AM';
            }


            // Detectamos cuando sean las 0 AM y transformamos a 12 AM
            if (horas == 0 ){
                  horas = 12;
            }
            pHoras.textContent = horas;
            pAMPM.textContent = ampm;


            // Minutos y Segundos
            if (minutos < 10){ minutos = "0" + minutos; }
            if (segundos < 10){ segundos = "0" + segundos; }
            pMinutos.textContent = minutos;
            pSegundos.textContent = segundos;
      };
      actualizarHora();
      var intervalo = setInterval(actualizarHora, 1000);
}())
```

Archivo CSS

```
*{
    margin: 0px;
    padding: 0px;
}

/*VARIABLES*/
:root{
      --posicion_superior: 170px; /*posicionar los divs de las plazas*/
      }

.contenedor{
      width: 1024px;
      height: 0 auto;
      background: white;
      margin: 0em auto;  /*centra el contenedor*/
      padding: 0.5em;
      -webkit-box-shadow: 0px 0px 12px 0px rgba(46,55,158,1);
      -moz-box-shadow: 0px 0px 12px 0px rgba(46,55,158,1);
      box-shadow: 0px 0px 7px 0px rgba(0,0,0,1);
}

header{
      font-weight: bold;
      text-align: center;
      height: 130px;
      background:blue;
      color: white;
      padding: 0.5em;
}
```

```
img.izquierda {
	float: left;
	width: 140px;
	height: 65;
}
img.derecha {
	float: right;
}

.texto {
	color: #FFF;
	margin: 0 auto;
	text-align: center;
	font-weight: 75px;
	font: italic bold 96px Georgia, Serif;
	text-shadow: -4px 3px 0 #34d16e, -14px 7px 0 #0a0e27;
}

body {
	font-family: Cambria, Courier New, Helvetica, sans-serif;
	font-size: 12px!important;
	color: #ffffff;
	background-color: #EDECEC;
}

section {  /*Seccion que contiene la img del parqueadero*/
	position : relative;
	width: 94%;
	height: 640px;
	margin: 1.2em auto;
	-webkit-box-shadow: 0px 0px 12px 0px rgba(46,55,158,1);
```

```
      -moz-box-shadow: 0px 0px 12px 0px rgba(46,55,158,1);
      box-shadow: 0px 0px 7px 0px rgba(46,55,158,1);
}

section img{
      width: 100%;
      height: 100%;
}

.plaza {  /*tamaño y màs caracteristicas de la plazas en la pagina*/
      display: flex;
      justify-content: center;
      align-content: center;
      flex-direction: column;
      border-radius: 10px;
      text-align: center;
      font-weight: bold;
      line-height: 1.5;
      height : 210px;
      width: 133px;
      font-size:28px;
}

  /* posicionamiento de los div's*/
.cero {
      position : absolute;  /*absolute no tiene espacio en la pagina*/
      left: 68px;
      top:  var(--posicion_superior);
      z-index:2
}
```

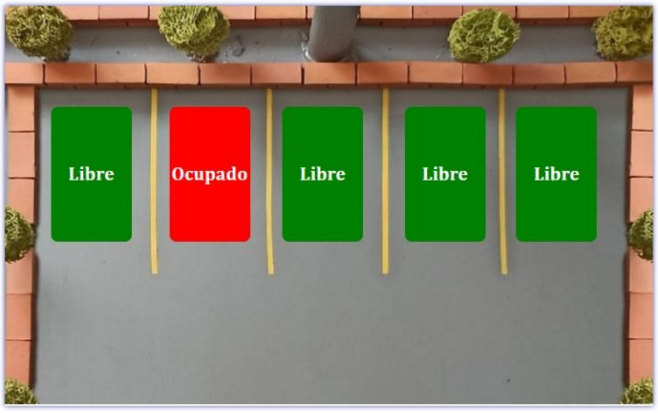
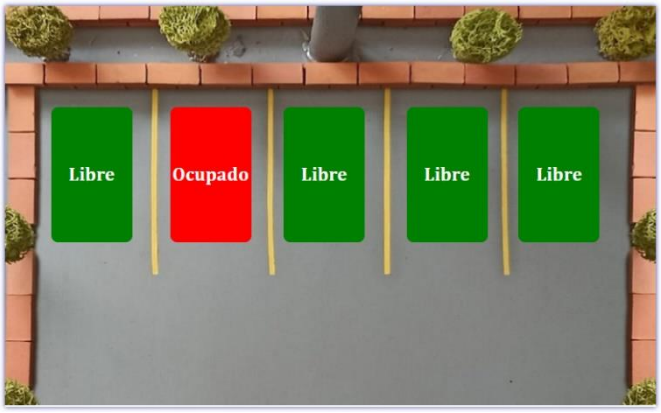
```
.uno {
	position : absolute;
	left: 244px;
	top: var(--posicion_superior);
	z-index:2;
}

.dos {
	position : absolute;
	left: 412px;
	top:  var(--posicion_superior);
	z-index:2;
}

.tres {
	position : absolute;
	left: 590px;
	top:  var(--posicion_superior);
	z-index:2;
}

.cuatro {
	position : absolute;
	left: 760px;
	top:  var(--posicion_superior);
	z-index:2;
}

footer{
	text-align: center;
	font-size: 17px!important;
```

```
	width: auto;
	height: 185px;
	background: #2980B9;
	margin-top: 44px;}

footer p{
	background: #5DADE2;
	font-size: 16px;
	padding: 5px;
}

/* estilo de Fecha y Hora*/
.wrap {
	width: 100%;
	max-width: 1000px;
	margin:15px;
}

.widget {
	width: 40%;
	margin:auto;
}

.widget p {
	color: rgb(0,0,0);
	display: inline-block;
	line-height: 1em;
}
```

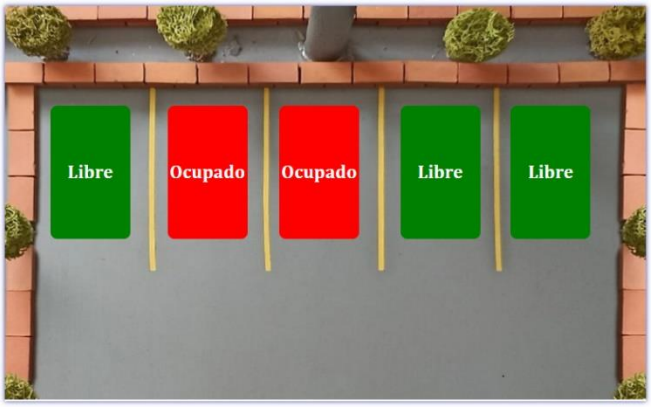
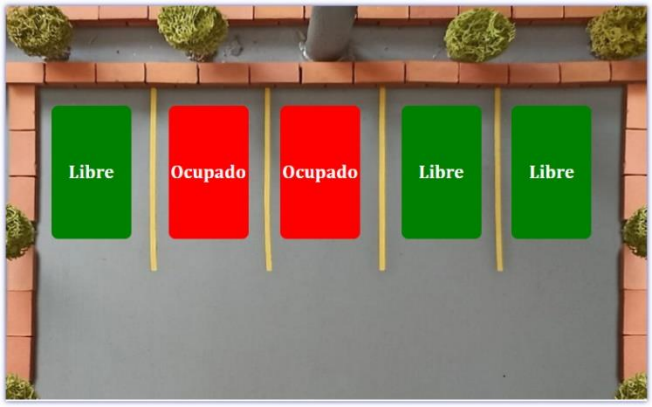
```
.fecha {
	text-align: center;
	font-size:2.5em;
	margin-bottom: 0rem;
	padding: 2px;
	width: 100%;
}

.reloj {
	font-family: Cambria, Arial;
	width: 100%;
	padding: 5px;
	font-size: 4em;
	text-align: center;
}

.reloj .caja-segundos {
	display: inline-block;
}

.reloj .segundos,
.reloj .ampm {
	font-size: 1.4rem;
	display: block;
}
```

Apéndice D. Pruebas con vehículos estacionados correctamente

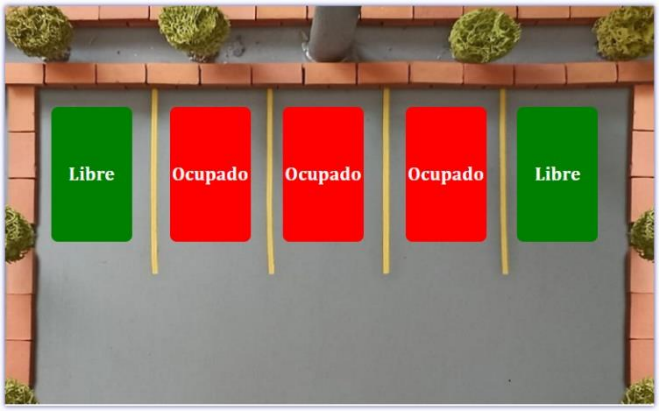
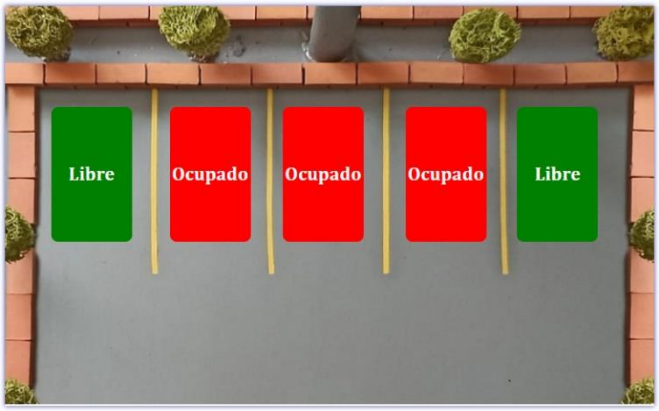
PARA 1 VEHÍCULO

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 10 Plaza 1: 232 Plaza 2: 22 Plaza 3: 22 Plaza 4: 0	Plaza 0: 15 Plaza 1: 233 Plaza 2: 26 Plaza 3: 11 Plaza 4: 2

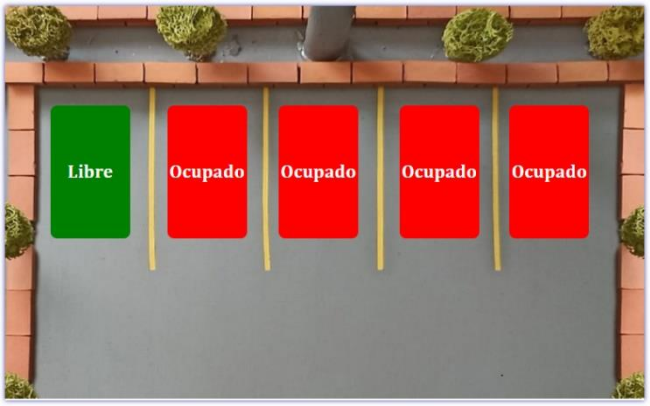
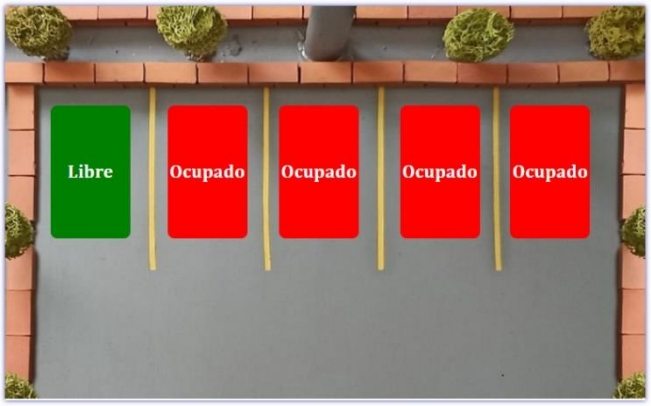
PARA 2 VEHÍCULOS

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 12 Plaza 1: 233 Plaza 2: 228 Plaza 3: 14 Plaza 4: 9	Plaza 0: 13 Plaza 1: 233 Plaza 2: 230 Plaza 3: 12 Plaza 4: 9

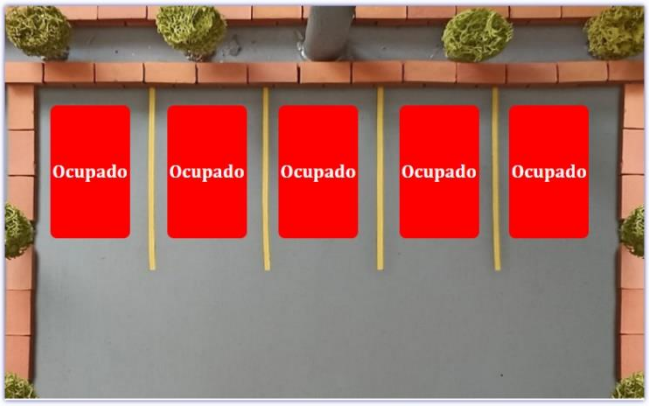
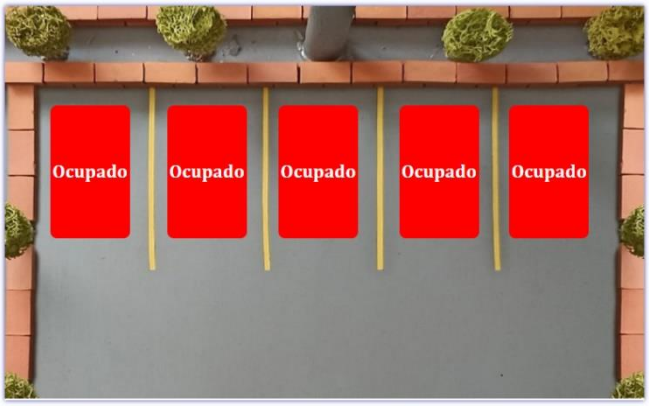
PARA 3 VEHÍCULOS

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 9 Plaza 1: 233 Plaza 2: 232 Plaza 3: 233 Plaza 4: 15	Plaza 0: 7 Plaza 1: 232 Plaza 2: 225 Plaza 3: 232 Plaza 4: 11

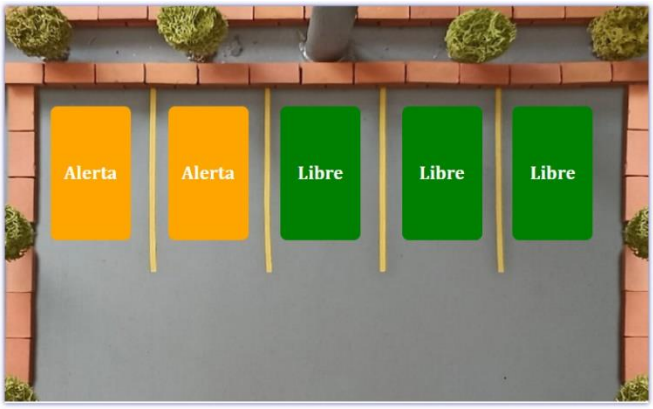
PARA 4 VEHÍCULOS

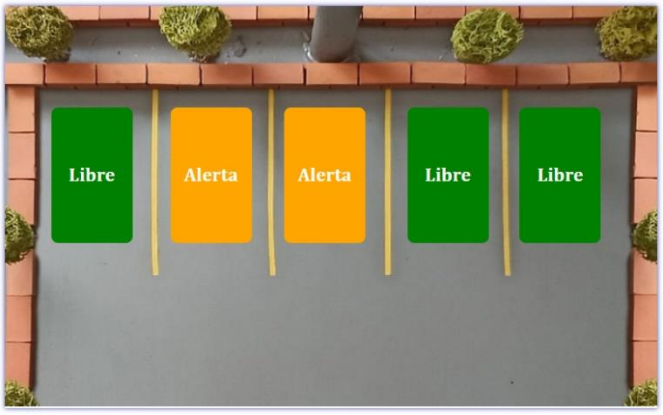
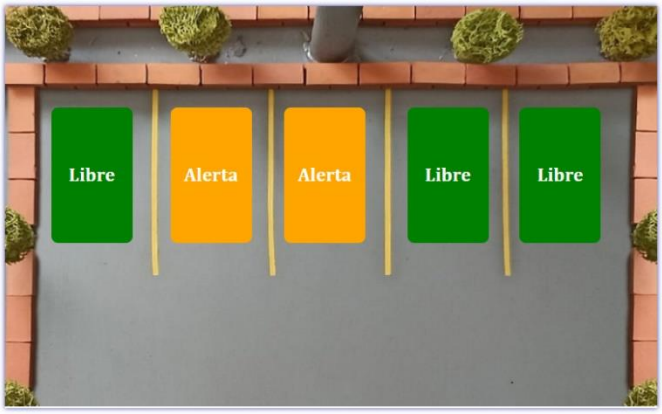
	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 4 Plaza 1: 229 Plaza 2: 228 Plaza 3: 229 Plaza 4: 205	Plaza 0: 8 Plaza 1: 232 Plaza 2: 230 Plaza 3: 232 Plaza 4: 230

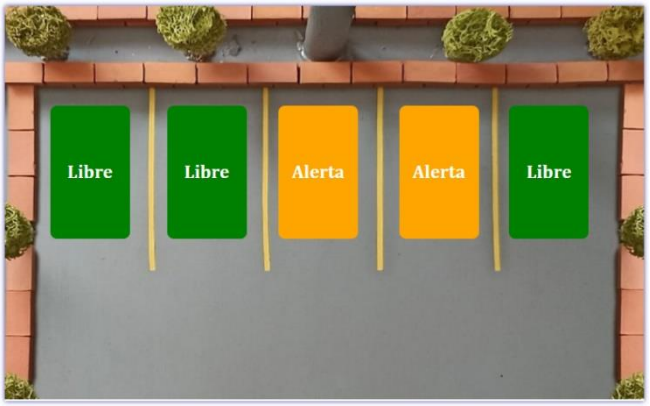
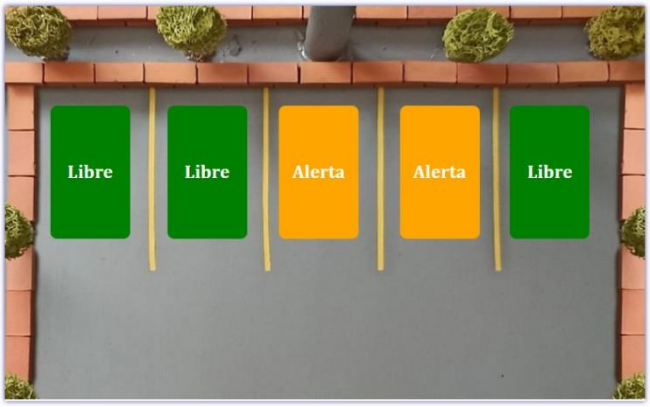
PARA 5 VEHÍCULOS

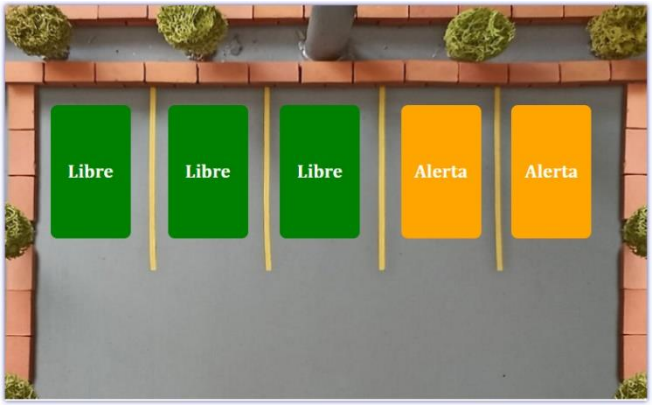
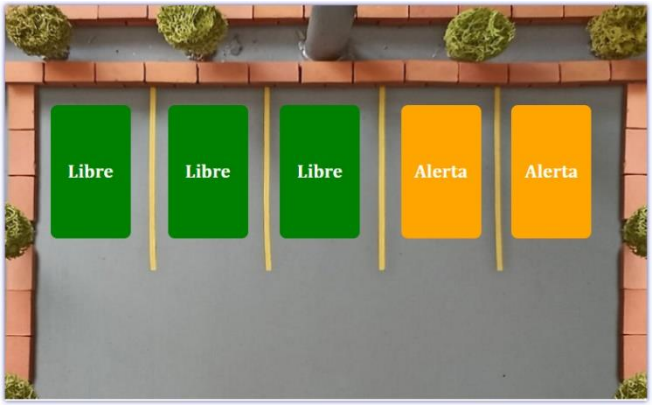
	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 227 Plaza 1: 231 Plaza 2: 224 Plaza 3: 226 Plaza 4: 205	Plaza 0: 234 Plaza 1: 229 Plaza 2: 223 Plaza 3: 229 Plaza 4: 209

Apéndice E. Pruebas con vehículos que ocupan dos plazas

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 110 Plaza 1: 98 Plaza 2: 25 Plaza 3: 24 Plaza 4: 13	Plaza 0: 87 Plaza 1: 103 Plaza 2: 0 Plaza 3: 0 Plaza 4: 0

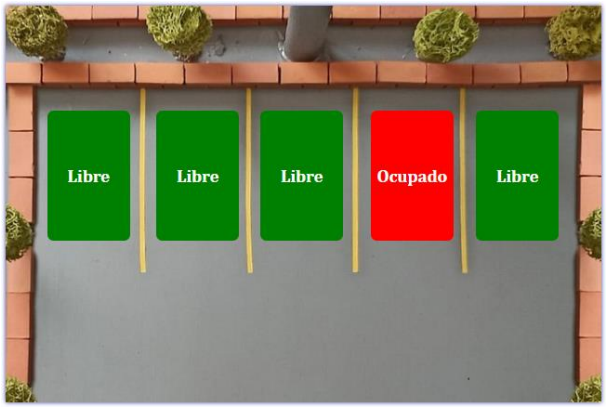
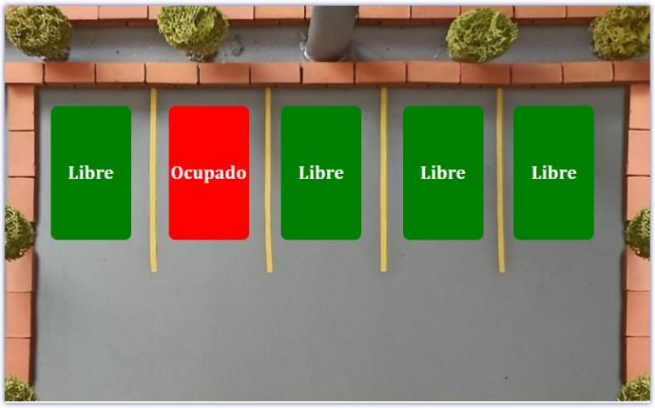
	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 12 Plaza 1: 113 Plaza 2: 89 Plaza 3: 22 Plaza 4: 14	Plaza 0: 0 Plaza 1: 75 Plaza 2: 93 Plaza 3: 0 Plaza 4: 0

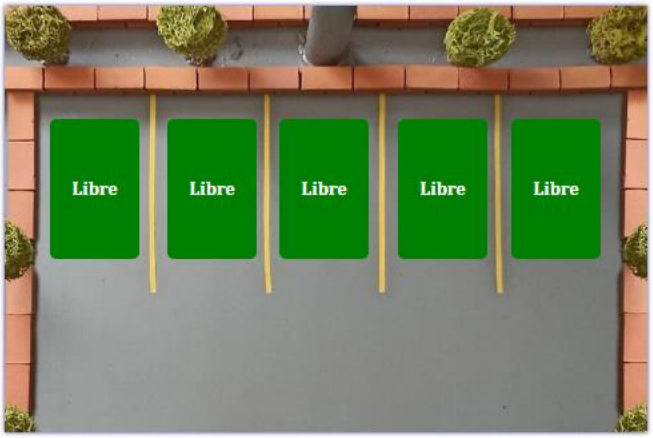
	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 12 Plaza 1: 29 Plaza 2: 104 Plaza 3: 90 Plaza 4: 14	Plaza 0: 0 Plaza 1: 0 Plaza 2: 82 Plaza 3: 85 Plaza 4: 0

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 11 Plaza 1: 29 Plaza 2: 24 Plaza 3: 96 Plaza 4: 77	Plaza 0: 0 Plaza 1: 0 Plaza 2: 0 Plaza 3: 87 Plaza 4: 76

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 17 Plaza 1: 112 Plaza 2: 119 Plaza 3: 107 Plaza 4: 93	Plaza 0: 132 Plaza 1: 121 Plaza 2: 126 Plaza 3: 122 Plaza 4: 2

Apéndice F. Pruebas con objetos diferentes a un vehículo

	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 0 Plaza 1: 0 Plaza 2: 27 Plaza 3: 208 Plaza 4: 13	Plaza 0: 11 Plaza 1: 226 Plaza 2: 15 Plaza 3: 17 Plaza 4: 4

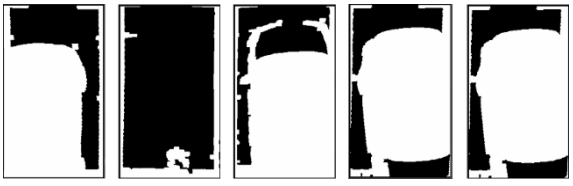
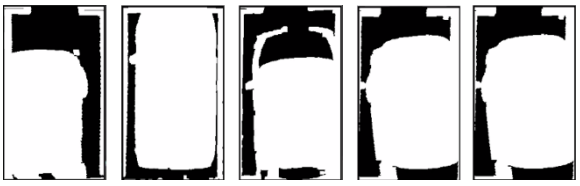
	INTENSIDAD MENOR A 70	INTENSIDAD MAYOR A 70
IMAGEN ORIGINAL		
VISTA EN LA APLICACION		
PROMEDIO	Plaza 0: 0 Plaza 1: 25 Plaza 2: 20 Plaza 3: 18 Plaza 4: 18	Plaza 0: 0 Plaza 1: 24 Plaza 2: 18 Plaza 3: 17 Plaza 4: 4

Apéndice G. Pruebas en un estacionamiento real

Con el fin de realizar pruebas en una zona de estacionamiento real se tomó un video de YouTube. Para verificar el algoritmo creado fue necesario adaptar las regiones de interés de las cinco plazas, seleccionar el umbral de binarización adecuado y crear una función adicional para visualizar el estado de cada plaza.

Se tomaron dos instantes del video, el primero muestra cuatro plazas ocupadas y una disponible, el segundo tiene las cinco plazas ocupadas.

Los resultados obtenidos se visualizan en la siguiente tabla:

	Cuatro plazas ocupadas y una disponible	Cinco plazas ocupadas
Estacionamiento real		
Resultado del algoritmo		

	Cuatro plazas ocupadas y una disponible	Cinco plazas ocupadas
Promedio	promedio0: 168.1575 promedio1: 41.20125 promedio2: 176.6625 promedio3: 176.0025 promedio4: 176.0025	promedio0: 162.09 promedio1: 214.41 promedio2: 163.215 promedio3: 177.74625 promedio4: 177.74625
Estados de cada plaza	Estado0: ['red', 'Ocupado', 1] Estado1: ['green', 'Libre', 0] Estado2: ['red', 'Ocupado', 1] Estado3: ['red', 'Ocupado', 1] Estado4: ['red', 'Ocupado', 1]	Estado0: ['red', 'Ocupado', 1] Estado1: ['red', 'Ocupado', 1] Estado2: ['red', 'Ocupado', 1] Estado3: ['red', 'Ocupado', 1] Estado4: ['red', 'Ocupado', 1]