Congreso Internacional en Ingeniería Electrónica. Mem. Electro 2015, Vol. 37, pp 402-407, Chihuahua, Chih. México
http://depi.itchihuahua.edu.mx/display/memorias_electro/MemoriaElectro2015
ISSN 1405-2172

MONITOREO DE ESPACIOS DE ESTACIONAMIENTO MEDIANTE CAMARA WEB

Rascón Madrigal Lidia Hortencia, Torres Herrera Carlos Daniel, Enríquez Aguilera Francisco, Ernesto Sifuentes de la Hoya.
Universidad Autónoma de Ciudad Juárez
Departamento de Ingeniería Eléctrica y Computación
Av. del Charro 450 Nte. Col. Partido Romero, Ciudad Juárez, Chihuahua.
Tel: 688-4841, email: lrascon@uacj.mx,
al77760@alumnos.uacj.mx, fenrique@uacj.mx, esifuent@uacj.mx

RESUMEN.

En este trabajo se presenta un sistema de monitoreo a un estacionamiento, el sistema es capaz de detectar los espacios disponibles mediante la detección de bordes aplicando el algoritmo Canny, el sistema combina la tecnología de computación en la nube y la interacción de máquina a máquina. El prototipo utiliza una cámara web para adquirir imágenes en un lugar de estacionamiento mediante una microcomputadora *Rasberry PI* y se almacena la información en un servicio de alojamiento de archivos multiplataforma en la nube, la cual es procesada y analizada en un programa basado en LabVIEW para detectar espacios disponibles. Se podría utilizar la información obtenida para canalizar los vehículos directamente hacia los espacios de estacionamiento actualmente disponibles e incluso reservar uno de ellos.

Palabras Clave: LabVIEW, Canny, *Raspberry PI*, Nube y Bordes.

ABSTRACT.

In this work, is presented a parking monitoring system, the system is able to detect the available space by detecting the edges using Canny algorithm, the system combines the technology of cloud computing and machine-to-machine interaction. The prototype uses a webcam to acquire images in a parking spot by a microcomputer Rasberry PI and the information is stored in a personal cloud storage multiplatform, such information is processed and analyzed in a system based on LabVIEW program to detect spaces available. Such data could be used to channel vehicles directly to the parking spaces currently available or even book one.

Keywords: LabVIEW, Canny algorithm, *Raspberry*, Cloud and Edge.

1. INTRODUCCIÓN

Los estacionamientos tienen una gran limitación en las grandes ciudades, un sistema de estacionamiento inteligente y seguro es costoso y limitado en la mayoría de los casos, se busca que en estos espacios la conducción sea fluida y sin realizar paradas que entorpezcan el flujo vehicular, lo que lograría minimizar el consumo de combustible y las emisiones de CO_2, además del estrés que se genera en el conductor [1]. En [2] y [3], se presenta el concepto genérico de utilizar la información en la nube para estacionamientos inteligentes, como una aplicación extendida del uso del Internet de las Cosas (*IoT Internet of Things*). Las investigaciones de estacionamientos inteligentes están relacionadas con los sistemas de transporte inteligentes, lo cual aplica para estacionamientos de empresas, donde el servicio no solo maneja la operación de estacionamientos de la fábrica, sino que debe diseñarse para ser compatible con un gran número de aspectos que pueden intervenir en los estacionamientos, una de las principales características de este tipo de servicios es el poder realizar reservaciones a través de internet como el sistema desarrollado por K. Inaba, et al [4], [5].

Otra utilidad que se le puede dar a un estacionamiento inteligente es el rastreo y/o registro de vehículos, siempre y cuando cuente con cámaras de visión, con lo cual se pueden desarrollar algoritmos de visión por computadora que hagan dicha tarea, la búsqueda se puede realizar por color y número de placa, esto último se convierte en una tarea difícil de realizar al 100% ya que es complicado obtener una buena imagen de la misma; sin embargo, se pueden complementar con el color del auto y con sólo obtener algunos caracteres de la placa [6], [7].

2. DESARROLLO

El prototipo realizado este trabajo utiliza una cámara web para adquirir imágenes en un lugar de estacionamiento mediante una computadora de placa única *Rasberry PI*, almacena la información (imagen) en un servicio de alojamiento de archivos multiplataforma en la nube, la cual es procesada y analizada en un programa basado en LabVIEW para detectar los espacios disponibles en el área de estacionamiento.

El prototipo funciona en tres bloques principales que se muestran en la figura 1. El primer bloque, captura de la imagen y la almacena en un servidor ftp con una IP pública http://45.55.183.22/; la imagen se archiva y se reescribe cada vez que se tome una fotografía en la *Raspberry PI*, con el fin de ahorrar espacio en el servidor. El segundo bloque, extrae la imagen del servidor, la almacena en la computadora y hace las conversiones necesarias para su análisis. El tercer bloque, analiza la imagen para determinar si existe un espacio disponible o no en el estacionamiento.

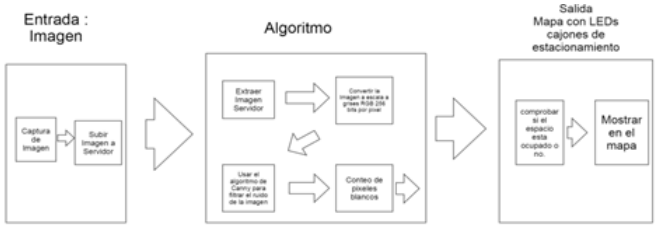


Figura 1. Diagrama de bloques del prototipo para ubicar un espacio de estacionamiento disponible.

2.1 Captura de la imagen

La imagen se obtiene mediante una cámara web conectada a una computadora de placa única *Raspberry PI* que transmite de forma inalámbrica (mediante un adaptador Wireless) las imágenes tomadas a un servidor remoto. El servidor ftp con una IP pública http://45.55.183.22/, almacena las imágenes y es un mediador entre el dispositivo de captura de imágenes y la aplicación de computadora que se encargará de su análisis.

2.1.1 La configuración de la *Raspberry Pi*

La configuración para la captura de imágenes requiere la instalación de *node-red* en la *Raspberry Pi*.

Node-Red es la herramienta que servirá para que la microcomputadora tome fotografías de forma automática y las almacene en el servidor [9]. Antes de instalar Node-Red se debe instalar la plataforma Node.Js esto se hace abriendo una terminal en la *Raspberry Pi* y escribiendo.

```
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install npm
```

Una vez que la instalación termina, la plataforma está instalada en la tarjeta, entonces se escribe la siguiente línea de comando en la terminal para instalar *Node-Red*.

```
sudo npm install -g node-red
```

Con *Node-Red* instalado en la *Raspberry Pi* se procede a crear un ciclo para que tome las fotografías y las suba al servidor ftp como se muestra en la figura 2. Se utilizan las herramientas del panel izquierdo, llamadas "Nodos" [9], la secuencia de pasos se presenta a continuación.

1. Añadir un nodo "*Inject*"

El nodo "*Inject*" (Inyectar) permite inyectar mensajes en un ciclo de flujo, ya sea haciendo clic en el botón del nodo, o estableciendo un intervalo de tiempo entre cada inyecta.

2. Añadir un nodo "exec"

El nodo "exec" permite introducir líneas de comando de forma síncrona, para llevar un control en el proceso de la toma de fotografías. Una vez agregado el nodo se conecta al nodo "*Inject*".

3. Añadir un segundo nodo "exec"

El nodo se encarga de subir las fotografías tomadas al servidor. Dentro del Nodo introducir el comando:

wput -B -nc /home/pi/camara/foto.jpg ftp://alex:020202@45.55.183.22/lastsnap.jpg

El comando usa la función *wput*, la función es una línea de comandos del tipo cliente-ftp similar a *wget* pero en lugar de la descarga de archivos, se encarga de subir archivos al servidor.

wput [etiquetas]... [archivo de subida]... [dirección del servidor receptor]...

Se usan las etiquetas *–B* y *–nc*. La primera etiqueta (*-B*) especifica al comando subir los archivos en modo binario, para que la imagen suba completa sin pérdida de paquetes durante el proceso. La segunda etiqueta (*-nc*) es para sobrescribir la fotografía en el servidor ftp, diciéndole que si existe un archivo con el mismo nombre, lo ignore y lo sobre escriba.

Archivo de subida. Indica que el archivo que se va a subir se encuentra en el directorio /home/pi/camara/ que es la fotografía que fue tomada por el nodo anterior al ciclo.

Dirección del servidor receptor. Indica la dirección del servidor ftp, en este caso 45.55.183.22 con el nombre y usuarios establecidos en esa misma línea como usuario: alex, contraseña: 020202 que se guardara en el directorio raíz del servidor con el nombre lastsnap.jpg

4. Agregar un último nodo "exec"

Este último nodo se encarga de terminar el proceso de capturar y de almacenar las fotografías en el servidor para evitar un empalme entre procesos.

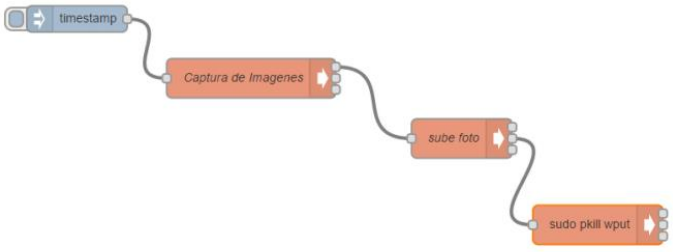


Figura 2.Flujo final para la captura y almacenamiento de fotografías.

2.1.2 Configuración de acceso *WebDAV* en el servidor público.

WebDAV es una extensión del protocolo HTTP que permite a los usuarios gestionar archivos en servidores. Una ventaja es la integración nativa en todos los sistemas operativos principales (Windows, Mac, Linux); no hay necesidad de instalar software de terceros para utilizar *WebDAV*. La *Raspberry* sube la fotografía mediante FTP pero la aplicación descarga la fotografía mediante *WebDAV* con el fin de agilizar el proceso de descarga y análisis por parte de la aplicación [10].

En la configuración de *WebDAV* se designa un directorio de trabajo, se habilitan los módulos necesarios y se configura.

Se designa un directorio de trabajo para usar *WebDAV*. Se crea el nuevo directorio en **/ var / www / *WebDAV***. Es necesario cambiar el propietario a **www-data** (el usuario de Apache) con el fin de permitir a Apache escribir en él.

Habilitar los Módulos. Los módulos de Apache se encuentran en ***/ etc / apache2 / mods-available***. Se crea un enlace simbólico desde ***/ etc / apache2 / mods-available*** *a* ***/etc / apache2 / mods-enabled.***

Configuración. Se crea un archivo de configuración en ***/etc/apache2/sites-available/000-default.conf*** usando un editor de texto en Linux,

En la primera línea, se agrega la configuración de la directiva DavLockDB:
DavLockDB /var/www/DavLock
Y el alias y el Directorio de directivas dentro de la sección VirtualHost:

Alias /WebDAV /var/www/WebDAV
<Directory /var/www/WebDAV>
DAV On
</Directory>

2.2 Algoritmo para análisis de la imagen.

Una vez que la imagen del estacionamiento (fotografía) está en el servidor, un programa realizado en LabVIEW, descarga la imagen. La aplicación se encarga de convertir la imagen de color en una imagen en blanco y negro con una escala de grises de 8 bits por pixel. Luego se le aplica el algoritmo de Canny a la imagen para eliminar el ruido que esta pueda contener y filtrar los bordes de los objetos que interesa analizar, en este caso los bordes de los automóviles. El programa cuenta los pixeles en blanco que estén en cada región de los espacios del estacionamiento mediante un histograma. El código del programa se muestra en la figura 3, y se explica a continuación.

1. **Aplicar el algoritmo de Canny.**
   Descarga la Imagen del servidor y genera un archivo IMAQ con el algoritmo de Canny ya aplicado.
2. **Analiza espacios.**
   Obtiene las regiones de interés de cada espacio del estacionamiento, las regiones de interés se extraen de una base de datos hecha previamente para este programa.
   Analizar cada zona de interés para decidir si hay un automóvil o no en cada cajón o espacio del estacionamiento.
3. **Detección de espacios disponibles.**
   El programa recibe tres valores. Uno son valores booleanos sobre si existe o no un automóvil previamente analizado, y recibe un clúster con las regiones de interés de cada rectángulo en el estacionamiento, también recibe la imagen en la que se dibujaran los rectángulos.

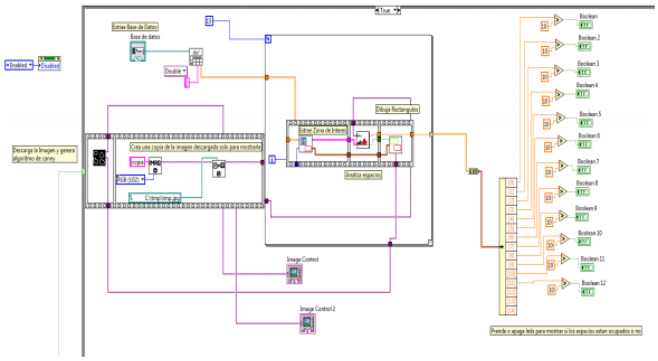


Figura 3. Código del programa completo realizado en LabVIEW.

2.2.1 Aplicar el algoritmo de Canny.

El Algoritmo de Canny para detección de bordes es un operador que utiliza un algoritmo multi-etapa para determinar una amplia gama de bordes en una imagen con ruido. Utiliza un filtro gaussiano *(m,n)* para suavizar una imagen *f(m,n)*. Esto reducirá el ruido o los detalles y texturas no deseadas como se muestra la ecuación 1 [8].

$$g(m,n) = G\sigma(m,n) * f(m,n) \quad (1)$$

Donde $G\sigma(m,n)$ es dado por la ecuación 2.

$$G\sigma(m,n) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[-\frac{m^2+n^2}{2\sigma^2}\right] \quad (2)$$

Calculando la gradiente de $g(m, n)$ usando cualquiera de los operadores de gradiente para llegar a la ecuación 3 y 4.

$$M(m,n) = \sqrt{g_m^2(m,n) + g_n^2(m,n)} \quad (3)$$

$$\theta(m,n) = tan^{-1}[g_n(m,n)/g_m(m,n)] \quad (4)$$

El valor del umbral *M* es dado por la ecuación 5

ISSN 1405-2172

$$M_T(m,n) = \begin{cases} M(m,n) & si\ M(m,n) > T \\ 0 & De\ \ otro\ modo \end{cases} \quad (5)$$

Donde T se elige de tal manera que todos los elementos de borde se mantienen y la mayor parte del ruido es suprimido [8].

La función ***CannyEdgeDetection*** utiliza el algoritmo de Canny, un método especializado de detección de bordes para estimar con precisión la ubicación de los bordes incluso bajo condiciones pobres de relaciones de señal a ruido.

En el programa realizado en LabVIEW, el primer paso es extraer la imagen del servidor que se encuentra en un folder temporal. El siguiente paso es crear un archivo IMAQ, se le aplica a la imagen la función ***IMAQ CannyEdgeDetection*** *VI* que viene incluida en la librería IMAQ de LabVIEW [11].

La función ***CannyEdgeDetection*** requiere los valores *Sigma, High Threshold, Low Threshold y WindowSiz*e.

El valor Sigma es el sigma del filtro de suavizado gaussiano que el VI aplica a la imagen antes de realizar la detección de bordes, *Hight Threshold* define el porcentaje superior de los valores de píxel en la imagen, el algoritmo de detección de bordes elige el punto de un segmento de borde de semillas o de partida, *Low Threshold* se multiplica por el valor de *Hight Threshold* para definir un umbral más bajo para todos los pixeles de un segmento de borde, *WindowSize* define el tamaño del filtro gaussiano que el VI se aplica a la imagen. El tamaño debe ser un número impar. Se aplicaron los siguientes valores para la eliminación de ruido:

1.000, 0.900, 0.365, 11.000

La imagen de la figura 4, muestra la imagen con la detección de bordes después de que se aplicó el algoritmo de Canny.

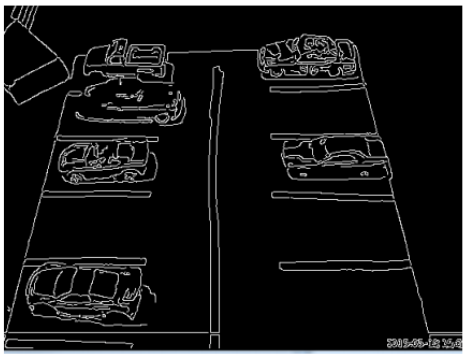


Figura 4. Imagen con el algoritmo de Canny aplicado.

2.2.2 Análisis de Espacios.

El análisis de la imagen utiliza funciones de la librería IMAQ. Se utiliza la función ***IMAQ Extract Tetragon VI***, extrae un rectángulo de la imagen previamente procesada con el algoritmo de Canny que corresponde al espacio ocupado por un cajón de estacionamiento. El rectángulo se obtiene utilizando los valores alimentados por el SubVI de **Extraer zonas de interés**, éste último, contiene las coordenadas en pixeles de los cajones de estacionamiento.

Para detectar si existe un automóvil en el espacio de estacionamiento se calcula el histograma; el rectángulo obtenido pasa por la función ***IMAQ Histogram VI*** que calcula el histograma de la imagen. Los valores entregados por el histograma cuando no existe un vehículo y cuando si existe, se muestran en la figura 5.

Figura 5. Análisis de los espacios de estacionamiento mediante histograma.

2.2.3 Detección de espacios disponibles

El último segmento del programa, se encarga de recibir los datos calculados por el histograma para cada espacio de estacionamiento y determina si existe o no un automóvil en cada espacio, además marca un rectángulo de color rojo los espacios disponibles. En la figura 6 se muestra la interfaz de usuario del programa en funcionamiento. Se observan dos imágenes, del lado izquierdo, la distribución del estacionamiento con los espacios de estacionamiento numerados e identificados con indicadores visuales, que se encienden para indicar que el espacio está ocupado y se apagan para indicar que el espacio esta disponible para estacionarse. A la derecha se muestra la imagen del estacionamiento con los contornos de los autos o un cuadro negro que indica el espacio vacío; además se han dibujado rectángulos, de color verde en

los espacios ocupados y de color rojo en los espacios disponibles.

Para dibujar los rectángulos se utiliza la función IMAQ *Overlay Rectangle VI* que se encarga de sacar las posiciones de un rectángulo en una imagen. Esta función recibe tres valores, la imagen en la que se van a dibujar los rectángulos, el clúster con los valores de cada zona de interés y el color en que se desea pintar dicho rectángulo. En las primeras pruebas del prototipo se creó una maqueta de estacionamiento y se utilizaron pequeños autos de juguete. Los espacios están enumerados para una mejor identificación. En la imagen de la izquierda en la figura 6 se observa, que los espacios 1,2,3,5,7,9 se encuentran ocupados (indicador visual encendido) mientras que los espacios 4,6,8,10,11,12 se encuentran disponibles (indicador visual apagado). En la imagen de la derecha se muestra la imagen del estacionamiento identificando los automóviles y los espacios disponibles que se remarcan con un rectángulo rojo.

Figura 6. Interfaz de usuario del programa para la detección de espacios disponibles en el estacionamiento.

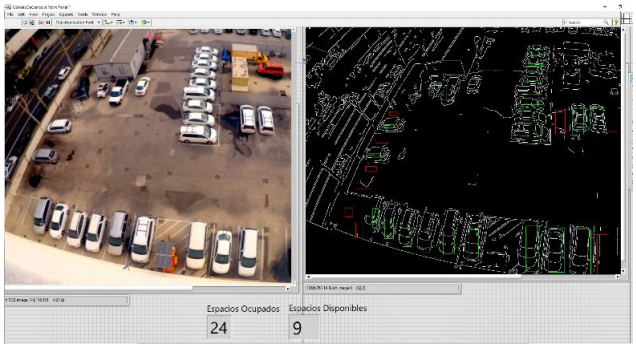
3 RESULTADOS.

Para realizar pruebas al sistema propuesto, se diseñó una maqueta y con esto lograr estresarlo en condiciones que se pudieran tener en la vida real, se pudo detectar que en un ambiente real sería difícil cambiar la posición de la cámara para lograr que abarque todos los cajones de forma equitativa (como en la maqueta), así que se optó por hacer las regiones de interés más pequeñas en los cajones más lejanos y con esto se logró que el sistema funcionara adecuadamente. Se hicieron varias pruebas en varias posiciones y los resultados se presentan en la Tabla 1.

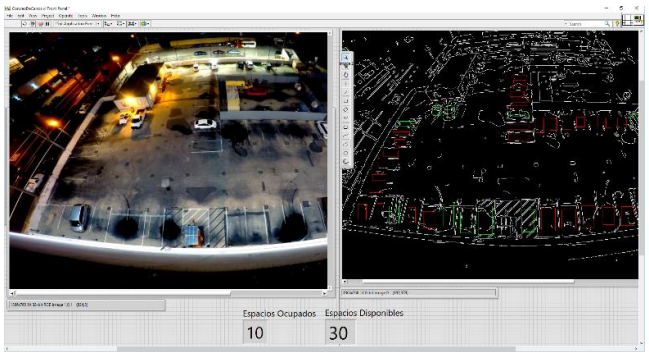
Tabla 1. Resultados de las pruebas realizadas con el programa.

Carros Reales	Detectados	Notas
12	12	
10	10	
8	8	
6	6	
9	8	Había un empalme en las regiones de interés
8	7	Detectaba el espacio con carro y el espacio vacío como activo, por empalme
6	6	
3	3	
10	9	Empalme en zonas de interés
12	12	
10	10	
9	9	
9	8	Empalme
4	4	
0	0	

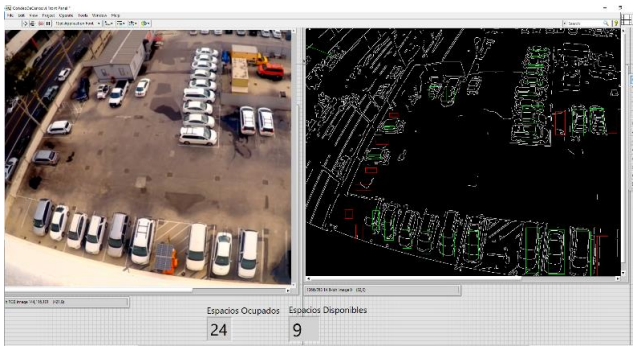
Una vez depurado el software con los posibles errores encontrados en la maqueta, se procedió a realizar pruebas a fotos de estacionamiento en diferente horario, para poder tener variación de luminosidad del ambiente (a las 12pm, 6pm y 10pm), los resultados se presentan en las figuras 7 (a), (b) y (c).



(a)



(b)



(c)

Figura 7. Pruebas en imágenes de estacionamiento real a diferente horario a) A las 12pm, b) a las 6pm y c) En un horario de 10pm.

En esta estudió se pudo comprobar que el programa de detección de espacios funciona con diferente utilización de los espacios del estacionamiento y a diferente intensidad de luminosidad.

4 CONCLUSIONES.

Después de hacer varias pruebas con distintos números de automóviles se demostró la efectividad del programa detector de espacios ocupados y desocupados, los errores que se pudieran generar pueden ser por error del programador al no definir mejor las zonas de interés, esto se puede solucionar re-calibrando la imagen con una mejor zona de interés. Otra falla que puede haber en el sistema es la detección de espacios que estén invadidos por usuarios mal estacionados lo cual sería una mejora a realizar.

En este trabajo se ha probado la efectividad de los algoritmos de cálculo de bordes para ayudar a la detección de autos en un estacionamiento, esto se logró utilizando imágenes de estacionamientos reales, y con la maqueta para observar su efectividad trabajando. Sin embargo, aún falta realizar un análisis estadístico utilizando imágenes reales del estacionamiento y se pudiera probar el algoritmo propuesto en [12] para comprobar si se mejora el tiempo de procesamiento para calcular los bordes y completar el trabajo con sensores de presencia de autos en cada espacio, detección de placas de cada auto para poder completar lo que se llama un estacionamiento inteligente.

5 REFERENCIAS.

[1] G. Yan, W. Yang, D. B. Rawat, and S. Olariu, “SmartParking: A secure and intelligent parking system: Observations from the IEEE IV 2010 symposium,” IEEE Intell. Transp. Syst. Mag., vol. 3, no. 1, pp. 18–30, 2011.

[2] Z. Ji, I. Ganchev, and X. Zhang, “A Cloud-Based Intelligent Car Parking Services for Smart Cities 2 . The Top-Down Design of the Intelligent Car Parking Service 3 . Sample Car Parking Service for University Campus,” pp. 2–5, 2014.

[3] N. Li, P. Fan, and H. Lv, “The Construction of Cloud Data Analysis Platform and Its Application in Intelligent Industrial Park,” pp. 860–863, 2012.

[4] K. Inaba, M. Shibui, T. Naganawa, M. Ogiwara, and N. Yoshikai, “Intelligent parking reservation service on the Internet,” Proc. 2001 Symp. Appl. Internet Work. (Cat. No.01PR0945), pp. 159–164, 2001.

[5] L. Wenhong, X. Fanghua, and L. Fasheng, “Design of the inner intelligent parking guidance system,” Proc. Int. Conf. Inf. Manag. Int. Conf. Inf. Manag. Innov. Manag. Ind. Eng. ICIII 2008, vol. 3, pp. 90–93, 2008.

[6] H. C. Tan, J. Zhang, X. C. Ye, H. Z. Li, P. Zhu, and Q. H. Zhao, “Ntelligent car-searching system for large park,” Proc. 2009 Int. Conf. Mach. Learn. Cybern., vol. 6, no. July, pp. 3134–3138, 2009.

[7] E. C. Neto, E. S. Rebouças, J. L. Moraes, S. L. Gomes, and P. P. R. Filho, “Techniques Digital Image Processing And Applied Computational Intelligence,” vol. 13, no. 1, pp. 272–276, 2015.

[8] J. Canny, “A Computational Approach to Edge Detection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-8, no. 6, 1986.

[9] Emerging Technology. (s.f.). Getting Started with Node-Red., [en línea] http://nodered.org/docs/getting-started/installation.html. Recuperado el 20 de Marzo de 2015.

[10] Wong, L. Z. (22 de Septiembre de 2014). How To Configure *WebDAV* Access with Apache on Ubuntu 14.04, [en línea] disponible https://www.digitalocean.com/community/tutorials/how-to-configure-*WebDAV*-access-with-apache-on-ubuntu-12-04.

[11] Relf, C. G. Image acquisition and processing with LabVIEW, Boca Raton: CRC Press. 2013

[12] A. N. Acosta, F. Javier, E. Aguilera, and J. Cota-ruiz, “Detección de Círculos Basada en la Orientación de Bordes y Triángulos Rectángulos,” no. 108, pp. 1251–1266, 2014.