DIMENSIONAMIENTO DE UNA OFICINA MEDIANTE UN SISTEMA DE VISIÓN UTILIZANDO LOS PATRONES CUADRADOS UBICADOS EN TECHOS DE TABLA ROCA

Blanca Arlette Serrato Espino
Tecnológico Nacional de México en Celaya
13031064@itcelaya.edu.mx

María del Carmen Tinajero Campos
Tecnológico Nacional de México en Celaya
13031063@itcelaya.edu.mx

Salvador Manuel Malagon Soldara
Tecnológico Nacional de México en Celaya
salvador.malagon@itcelaya.edu.mx

Resumen

Este trabajo propone una alternativa para el proceso de desplazamiento en una oficina mediante un sistema de visión, siendo éste el primer paso para un posterior dimensionamiento del área de trabajo.

Muchas oficinas cuentan con un techo de tabla roca cuyos marcos son de aluminio en una forma particularmente cuadrada. Es importante resaltar que se trata de un ambiente controlado, donde los factores como luz, sombra, ángulos de inclinación de la toma de imágenes, entre otros; tienen una influencia destacable en los resultados obtenidos.

Para el desarrollo de este trabajo se utilizó una R*aspberry Pi 3,* una cámara web, un monitor, un teclado, un mouse, un editor de programación (Ninja) y el compilador Python con las librerías *OpenCV* y *NumPy.*

Obteniendo resultados favorables, ya que se logró la correcta detección de las líneas presentes para el cálculo de los cuadros que dimensionan el área de trabajo.

Para la realización de este trabajo se siguió el método de investigación científico exploratorio, ya que se pretende dar una visión general de tipo aproximado respecto a una determinada realidad.

Palabra(s) Clave: Líneas, OpenCV, Python, Visión.

Abstract

This work puts forward an alternative for the process of displacement in an office through a vision system, being this the first step for the dimensioning of the work area.

Many offices have a sheetrock ceiling whose frames are made of aluminum in a particularly square shape. It is important to emphasize that it is a controlled environment, where factors such as light, shadow, inclination angles of the image taken, and so on; they have an outstanding influence on the results obtained.

For the development of this work we used a Raspberry Pi 3, a webcam, a monitor, a keyboard, a mouse, a programming editor (Ninja) and the Python compiler with the OpenCV libraries and NumPy.

Obtaining favorable results, since the correct detection of the present lines was obtained for the calculation of the tables that dimension the work area.

For the accomplishment of this work the method of exploratory scientific investigation was followed, since it is intended to give an approximate general view of a certain reality.

Keywords: *Lines, OpenCV, Python, Vision.*

1. Introducción

Las imágenes de los techos de las oficinas están segmentadas con líneas rectas horizontales y verticales que forman cuadrados, lo cual nos lleva a sugerir el utilizar comandos establecidos mediante un sistema de visión para lograr el dimensionamiento del área de trabajo; debido a que ya se había logrado implementar satisfactoriamente el uso de sistemas de visión para la detección de líneas por medio de comandos de programación ya aprendidos con anterioridad en experiencias pasadas.

"El procesamiento de imágenes es la primera etapa a la que se someten las imágenes de la escena analizada con objeto de intentar determinar de forma automática su contenido. En este sentido hay que considerar el procesamiento de imágenes como el conjunto de técnicas que permiten transformar una imagen digital, con el fin de facilitar las etapas posteriores de segmentación y reconocimiento" [7].

"La transformación de imágenes se refiere a todos aquellos procesos que permiten obtener una nueva imagen g(x, y) a partir de la imagen original f(x, y). Donde el objetivo es conseguir que la imagen de salida mejore en alguna de sus características" [6].

Por otra parte, para comenzar el procesamiento, es esencial conocer algunas de las características más importantes en la constitución de una imagen. Un borde es una frontera entre dos regiones con propiedades de nivel de gris o tono relativamente distintas, es necesario que las regiones en cuestión sean suficientemente homogéneas para que la transición entre dos de ellas se pueda determinar solamente sobre la base de estas discontinuidades [1].

"Llamamos línea recta al lugar geométrico de los puntos tales que tomando dos puntos diferentes cualesquiera $P_1(x_1, y_1)$ y $P_2(x_2, y_2)$ del lugar, el valor de la pendiente *m* resulta siempre constante" [5]. Esta definición no puede aplicarse a una recta paralela al eje y, porque no tiene pendiente, tiende al infinito. La recta también se puede definir, analíticamente, como "una ecuación lineal o de primer grado con dos variables" [4].

2. Métodos

Para detectar las líneas horizontales y verticales, mostradas en la imagen en perspectiva, se utilizó el método de la Transformada Estándar de Hough.

"La transformada de Hough es un método de extracción de características patentado por Paul Hough en 1962, para la detección de bordes en imágenes parametrizadas, es decir, los objetos que se desean detectar se representan por medio de una ecuación conocida. La idea es transformar la imagen del espacio cartesiano xy a un espacio de parámetros y a través de un proceso de votos

identificar los puntos que describen una línea recta u otra formulación de bordes" [2].

"Los valores usados para los parámetros de la función HoughLines de OpenCV son: 1 para la resolución de ρ, π/360 radianes (0,5°) para la resolución de θ, y 193±5, para el umbral del acumulador. El método HoughLines devuelve una lista de la líneas detectadas en la imagen que se está procesando y cada una definida en coordenadas polares (ρ,θ) "[6].

Del conjunto de líneas obtenidas se etiquetan aquellas donde:

- 1.66 >θ< 1.48rad, como horizontales, paralelas al eje polar de referencia.
- −0.09rad >θ< 0.09rad, como verticales.
- Se trata de un rango cercano al tipo de recta que define la etiqueta.
- Los parámetros para el método *Canny* de *OpenCV* utilizados, según López Correa, fueron:
  - ✓ Umbral inferior para la histéresis = 50
  - ✓ Umbral superior para la histéresis = 150
  - ✓ Apertura para el operador de Sobel = 3
  - ✓ Los materiales utilizados fueron:
  - ✓ *Raspberry Pi 3*
  - ✓ Cámara *web*
  - ✓ Pantalla
  - ✓ Teclado
  - ✓ Mouse
  - ✓ Ambiente controlado

El proceso general se presenta en la figura 1. Éste se compone de varios pasos para lograr el objetivo que se plantea. Para generar el código primeramente fue necesario importar las librerías *cv2* y *NumPy*. Para realizar el código se utilizaron comandos especiales que se irán detallando poco a poco según su función en el programa [3].

Se propuso una función interna llamada *onMouse* para que cuando se oprimiera el clic derecho del mouse se realizara un evento y la condición cambiara a

verdadera. Luego se procedió a capturar el video utilizando el comando *cv2.VideoCapture(),* así como a nombrar la ventana para ser usada como un marcador de posición para imágenes y barras de seguimiento, donde fue referenciada por el nombre donde éste es el identificador utilizando el comando *cv2.namedWindow('MyWindowLAlala').* Posteriormente se utilizó el comando *cv2.setMouseCallback('MyWindowLAlala', onMouse)* que llamaba a la función interna declarada *onMouse* donde se estableció como controlador al mouse para una ventana especificada.

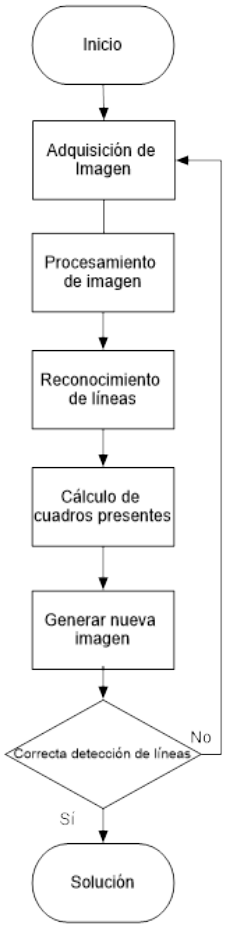

Figura 1 Proceso General.

A continuación, se utilizó el comando *cv2.cameraCapture.read()* que sirve para grabar, decodificar y regresar un cuadro del video (*frame*) cuyo argumento fue el nombre de la ventana donde se obtendrá el *frame*. La cámara fue tomando un video con un ciclo *while* que iba revisando cada milisegundo si ya se había dado

un *clic* del mouse y se detendría hasta que ocurra dicho *clic* del mouse. También se utilizó el comando *cv2.imshow('MyWindowLAlala',frame)* para mostrar la imagen obtenida en una ventana específica, donde sus parámetros eran el nombre de la ventana establecida y la imagen fue mostrada. Enseguida, se utilizó el comando *cv2.imwrite('MyWindowLAlala.jpg',frame)* para salvar la imagen en un archivo especificado con un nombre y el formato del archivo (*jpg* o cualquier otro). Para destruir la ventana generada se utilizó el comando *cv2.destroyWindow('MyWindowLAlala').*

Después, se comenzó a procesar la imagen utilizando el comando *cv2.imread('MyWindowLAlala.jpg')* que cargaba una imagen de un archivo creado anteriormente. Para convertir una imagen de un espacio de color a otro se utilizó el comando *cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)* donde sus parámetros eran la entrada de la imagen, la salida de la imagen del mismo tamaño que el de la entrada y el código de conversión del espacio de color. Además, se utilizó el comando *cv2.Canny(gray,50,150,apertureSize=3)* para configurar los parámetros en el método *Canny* explicado anteriormente. Se prosiguió a encontrar las líneas en una imagen usando la transformación estándar de Hough, utilizando el comando *cv2.HoughLines(edges,1,np.pi/180*10,100)*, donde los parámetros eran la imagen que sacaba la función *cv2.canny(edges), Rho* que era igual a 1 representa la resolución de distancia del acumulador de pixeles*, np.pi/180*10* representa *theta* que es la resolución angular del acumulador en radianes, el 100 es el umbral del acumulador, el cual hace que solo se devuelvan las líneas que obtengan suficientes votos *(> threshold)*.

Para realizar el cálculo del número de líneas se utilizaron dos ciclos anidados *for,* donde se generaban dos puntos y su pendiente para generar una línea con el comando *cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)* y así ir contando el número de líneas que fueron creadas.

Se debe de tomar en cuenta que al detectar dos líneas paralelas significa que es una arista de un cuadrado. Por lo cual, se procedió a realizar operaciones aritméticas para calcular el número de cuadrados presentes en la imagen.

Se obtiene el total dividiendo el número de líneas detectadas entre cuatro (ecuación *1*), así como también se obtiene el residuo del número de líneas detectadas con la operación de módulo tomando en cuenta que cuatro líneas conforman un cuadrado (ecuación *2*), y este residuo se multiplicó por diez para no trabajar con decimales y luego dividirlo entre cuatro (euación *3*).

$$total = \frac{\text{num}}{4} \tag{1}$$

$$res = \text{num}\%4 \tag{2}$$

$$residuo = \frac{res*10}{4} \tag{3}$$

Según el residuo obtenido se procedió a realizar determinada operación, si el residuo era cero quería decir que el área total estaba conformada por el mismo número de líneas tanto horizontalmente como verticalmente, y como se mencionó que dos líneas representan un lado de un cuadrado es necesario restarle una unidad al *total* obtenido anteriormente (ecuación *4*) y luego se elevaba al cuadrado (ecuación *5*) obteniendo así el número total de cuadrados presentes.

$$total1 = total - 1 \tag{4}$$

$$fin = total1 * total1 \tag{5}$$

Si el residuo era cinco quería decir que el área total estaba conformada de tal manera que el número de líneas horizontales era mayor al número de cuadros verticales o viceversa, como se mencionó que dos líneas representan un lado de un cuadrado es necesario restarle una unidad al *total* obtenido anteriormente (ecuación *6*) y luego se multiplicaba el *total* obtenido primeramente por el *total2* obtenido de la resta (ecuación *7*) obteniendo así el número total de cuadrados presentes.

$$total2 = total - 1 \tag{6}$$

$$fin = total * total2 \tag{7}$$

Si el residuo era diferente a cualquier resultado de los mencionados anteriormente quería decir que había ocurrido alguna falla en la detección correcta de las líneas presentes.

Al realizar el procedimiento, primeramente, se tomó la fotografía con una cámara conectada a la Raspberry Pi 3 una vez que se haya dado un clic sobre la pantalla,

enseguida se guardó la imagen que fue procesada posteriormente para obtener el número de líneas que tiene la imagen, para después realizar cálculos y así obtener el número de cuadros detectados, además de generar una nueva imagen donde se pudieran observar las líneas que el programa había reconocido.

En la figura 2 se muestra la parte del código utilizado donde se adquiere la imagen y se realiza el procesamiento de la misma, así como en la figura 3 hace referencia a la parte del código donde se hace el reconocimiento de las líneas, el cálculo de cuadros y la generación de la nueva imagen resultante.

3. Resultados

La fotografía tomada en la figura 4 se realizó con una cámara conectada a la R*aspberry Pi 3* mostrada en la figura 5 bajo un ambiente controlado haciendo referencia en la figura 6, ya que los factores de luz, sombra y ángulo de inclinación de la fotografía tomada, entre otros, afectaban el resultado final.

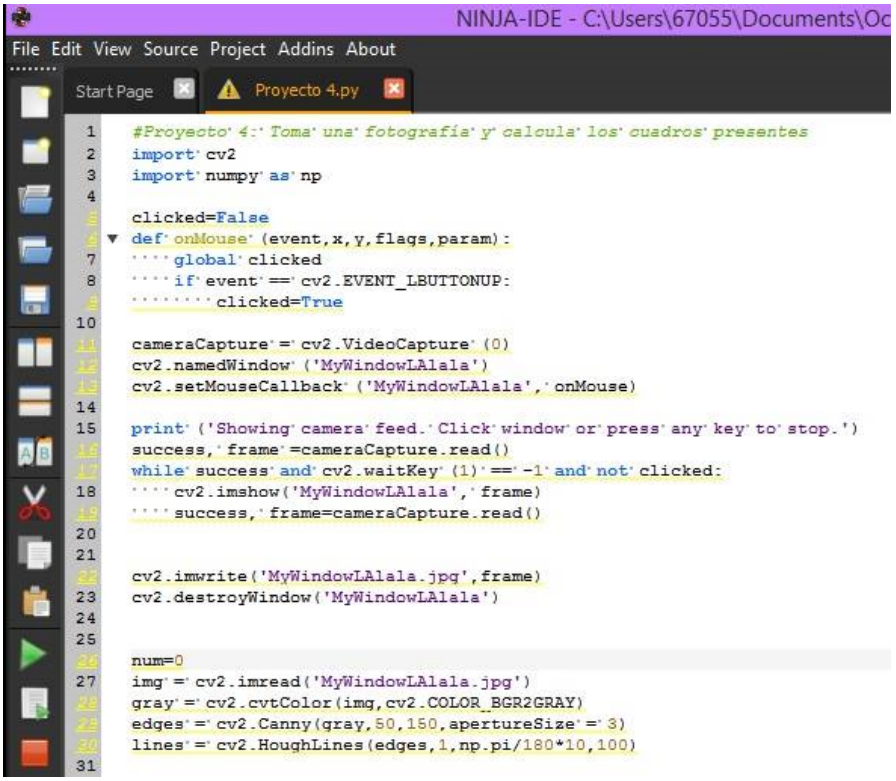

Figura 2 Primera parte del código.

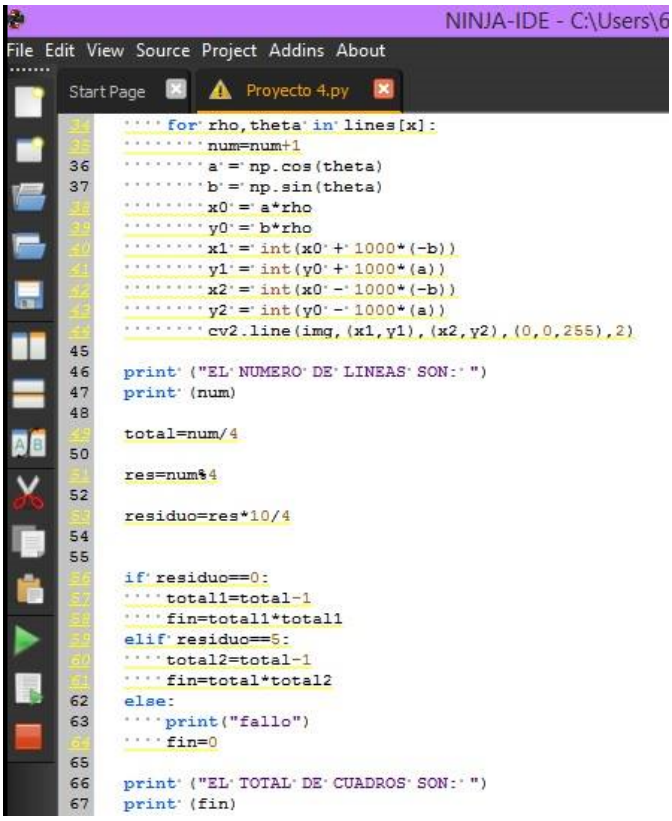

Figura 3 Segunda parte del código.

Figura 4 Fotografía tomada.

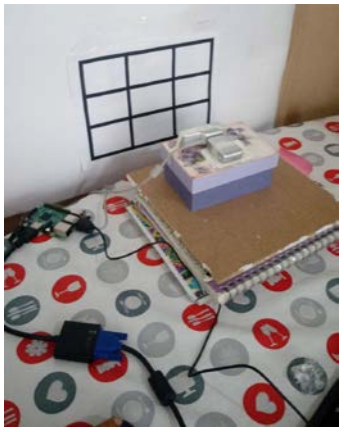

Figura 5 Cámara conectada a la Raspberry Pi 3.

Figura 6 Ambiente controlado.

El resultado mostrado en la figura 7 es el obtenido del programa al ejecutarse en Ninja fue que encontró ocho líneas por lo tanto es un cuadro.

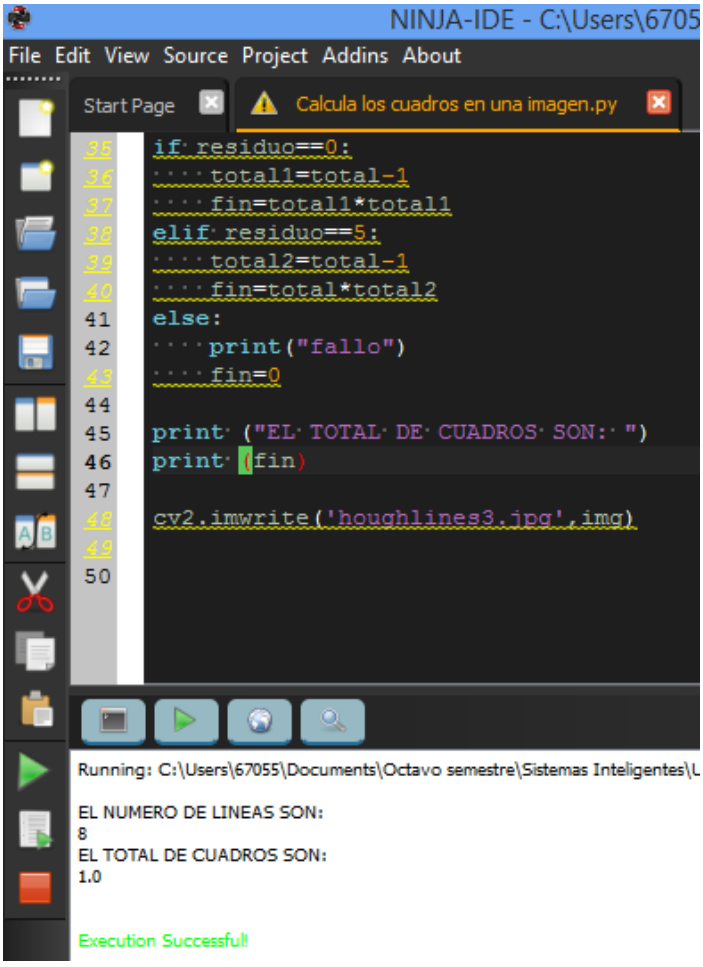

Figura 7 Resultados obtenidos.

Por lo cual se observó un correcto resultado, pero fue necesario generar la imagen con las líneas detectadas para saber que éstas fueron correctamente calculadas, así que con un comando especial se generó la imagen mostrada en la figura 8. Con lo cual se logró observar que el programa detectó correctamente cada línea presente en la imagen.

Se realizó el mismo procedimiento para otros casos con mayor número de líneas, tomando la fotografía mostrada en la figura 9, en la figura 10 se obtuvieron los resultados de Python y se corroboraron dichos resultados con la imagen generada en la figura 11.

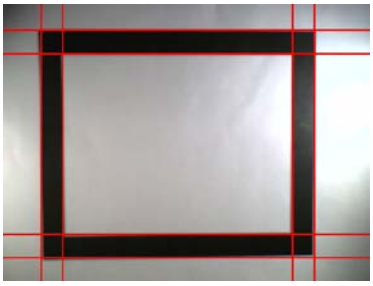

Figura 8 Imagen generada.

Figura 9 Fotografía tomada.

4. Discusión

Se obtuvieron los resultados deseados usando la técnica de transformación de línea de Hough aplicándola con la función *cv2.HoughLines*. Esta técnica se aplicó a imágenes de cuadrados impresos en tinta negra, aunque no es una fotografía de los patrones cuadrados ubicados en techos de tabla roca en oficinas, esta investigación presenta los principios básicos para la futura aplicación en imágenes tomadas en oficinas.

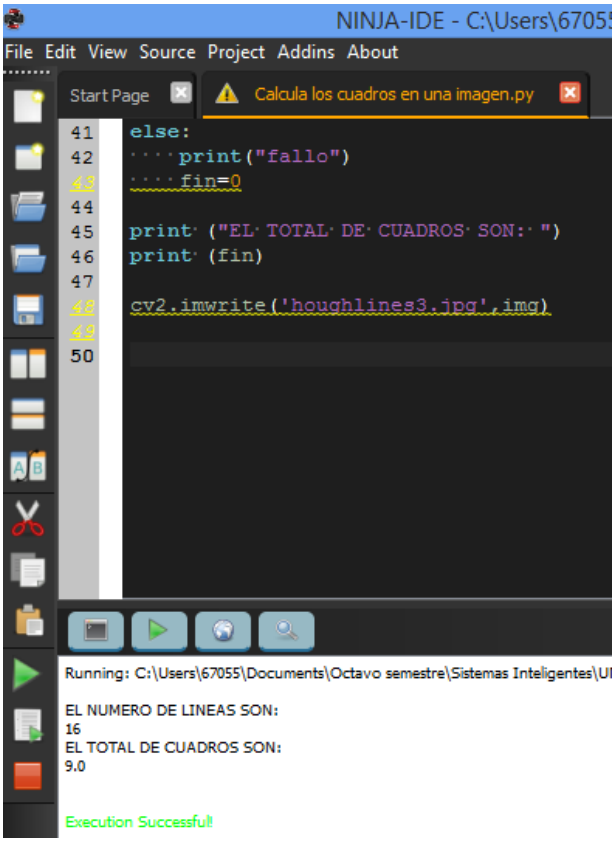

Figura 10 Resultados obtenidos.

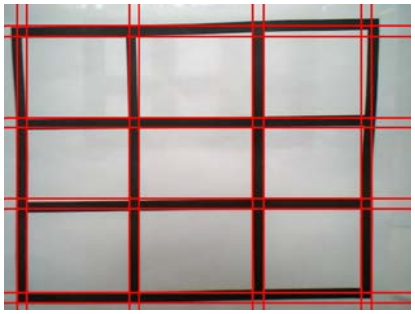

Figura 11 Imagen generada.

5. Bibliografía y Referencias

[1] González, R. & Woods, R. (2002). *Digital Image Processing* (2a. ed.). *USA:* Prentice Hall. ISBN 9780201180756.

[2] Hough, P. (18 de Dic de 1962). *US Patente n° 3069654.*

[3] Howse, J. (2003). *OpenCV Computer Vision with Python*. USA: Packt Publishing.

[4] Kindle, J. (1989). *Teoría y problemas de geometría analítica plana y del espacio*. Compendios Schaum. McGraw-Hill. ISBN 9789684229488.

[5] Lehmann, C. (1989). *Geometría Analítica*. Limusa. ISBN 9681811763.

[6] López Correa, Ma. (2015). *Ayuda a la localización de un robot móvil autónomo en los pasillos de un supermercado por medio de métodos de visión. Tesis para optar al título de Maestra en Inteligencia Artificial.* México: Universidad Veracruzana.

[7] Sánchez, J. (2002). *Avances en robótica y visión por computador. Ciencia y técnica*. España: Ediciones de la Universidad de Castilla-La Mancha: http://books.google.com.mx/books?id=VeXwzEIngQC.