SISTEMA DE ASISTENCIA Y MONITOREO PARA UN PARQUEADERO EN ENTORNO ABIERTO USANDO VISION ARTIFICIAL

SERGIO STEVEN CASTELLANOS ROJAS

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENÍERIA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ
2016

SISTEMA DE ASISTENCIA Y MONITOREO PARA UN PARQUEADERO EN ENTORNO ABIERTO USANDO VISION ARTIFICIAL

SERGIO STEVEN CASTELLANOS ROJAS

Monografía para optar al título de profesional en ingeniería electrónica

Asesor
ANDRÉS CAMILO JIMÉNEZ
Docente Académico

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ
2016

Nota de aceptación

_____
_____
_____

_____
Presidente del Jurado

_____
Jurado

_____
Jurado

Bogotá 01 de Agosto de 2016

Padres con todo
mi cariño, a mí
familia, a mí
hermano

AGRADECIMIENTOS

Quedan cortas las palabras para expresar el agradecimiento que tengo con mis padres por tan hermoso regalo que me han dado, a ustedes les dedico todos mis triunfos y éxitos en esta vida, a mi familia por su apoyo incondicional, a mis amigos y compañeros de carrera con quienes aprendí día a día, a mis docentes por ayudarme a ser la persona académica y profesionalmente formada quien soy hoy, a dios quien permitió que todo esto fuera posible, a la Universidad de Los Libertadores por brindarme sus recursos en estos años de formación.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ECUACIONES

LISTA DE ANEXOS

GLOSARIO

*Morfología: Es una rama de la biología que se enfoca en el estudio de las estructuras y formas tanto en plantas como en animales. Ahora bien la morfología matemática se puede definir como “una técnica no lineal de tratamiento de señales” que para este documento aplicaremos a imágenes.

*DSP: Se refiere a cualquier proceso digital que se aplica para modificar la representación digital de una señal, donde se ocupa de la representación, transformación y manipulación de señales discretas desde un punto de vista de la información que contienen”

*Python: Lenguaje de programación gratuito multiplataforma donde se pueden desarrollar programas con orientación a objetos y posee librerías capaces de permitirnos trabajar en DSP y morfología.

RESUMEN

El procesamiento digital de imágenes (por sus siglas en inglés, DIP), son las técnicas que se aplican a la figura con el fin de extraer información y mejorar su calidad visual, para esto se hace uso de la morfología matemática, es decir, el procesamiento interno de sus elementos, adoptando la imagen como un conjunto numérico organizado, a lo que se le denomina matriz.

El objetivo de este proyecto es ofrecer un control a los estacionamientos vehiculares abiertos, facilitando la ubicación de lugares libres a conductores y operarios, esto se logra mediante el uso de cámaras, dando como resultado la cantidad de lugares disponibles y la ruta que deben tomar los vehículos al ingresar.

1. INTRODUCCIÓN

En la actualidad existen diferentes sistemas para ayudar o realizar el control del ingreso de vehículos a los estacionamientos públicos, por ejemplo un control por medio de personal quienes están al tanto de los lugares que aún se encuentran disponibles, también se ha implementado el uso de foto resistencias o los sensores de infrarrojo instalados en los espacios vehiculares, los cuales al detectar un cambio en la iluminación cambian su estado indicando que el lugar ya se encuentra ocupado, pero este tipo de sistemas presenta muchos inconvenientes, como lo son la información con alto índice de incertidumbre, los elevados costos que generan en sus mantenimientos, teniendo en cuenta que por cada sitio de parqueo debe existir por lo menos un sensor instalado, además este tipo de soluciones solo es aplicable para estacionamientos cubiertos y teniendo en cuenta las extensas adecuaciones en tubería y cableado que se deben disponer para la instalación de los mismos. Es por estos motivos que se justifica el desarrollo de un sistema menos costoso y más confiable para realizar el control del ingreso vehicular y de la disponibilidad de lugares dentro de un estacionamiento.

Como solución se desarrolla un sistema electrónico fundamentado en el uso del procesamiento digital de imágenes y morfología matemática que mediante fotos en tiempo real obtiene información para suplir las necesidades del algoritmo.

El siguiente documento explica el paso a paso que se debe seguir y los fundamentos matemáticos necesarios para el desarrollo del proyecto anteriormente nombrado.

Cabe resaltar que cada subíndice del documento consta de una explicación matemática y teórica, de su respectiva relación con el desarrollo del proyecto y de un diagrama de flujo o un algoritmo matemático si es necesario, donde inicialmente se explica cómo se instancian las variables, a continuación se modifica la imagen matriz para depurar todo tipo de información que no sea necesario o cree ruido por medio de la morfología matemática, luego se procede a evaluar la cantidad y tamaño de elementos que aún existen en la imagen, obteniendo el número real de vehículos mediante los momentos de HU. Así finalmente al tener los lugares ocupados y la cantidad de espacios libres se crea un algoritmo que obtiene mediante una máscara los espacios vacíos y la ruta optima que debe seguir el conductor para llegar a su sitio final de parqueo. Para tener una mejor orientación sobre el algoritmo tenemos el siguiente diagrama de flujo.

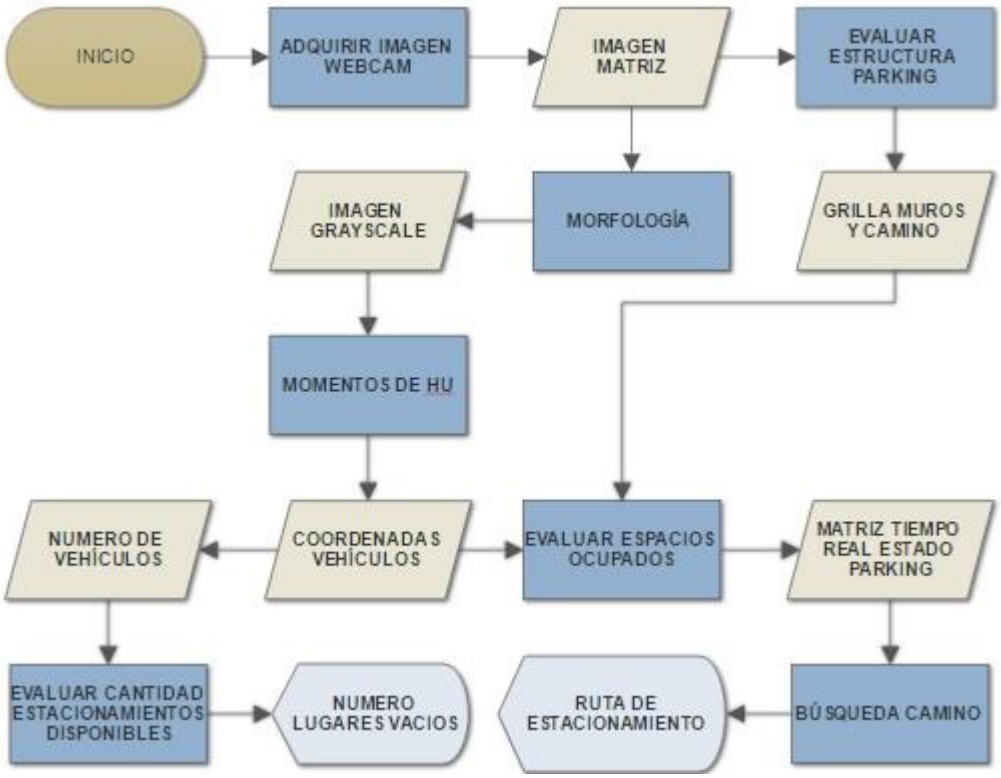


Ilustración 1. Diagrama de flujo para sistema de monitoreo y guía en parking

Fuente: Elaboración propia

1.1. OBJETIVOS

1.1.1 General

Desarrollar un sistema para guiar y controlar el flujo de vehículos en estacionamientos descubiertos por medio de procesamiento de imágenes.

1.1.2 Específicos

- Obtener el número de vehículos que se encuentran estacionados.
- Determinar la ubicación de cada vehículo estacionado.
- Establecer la ruta más eficaz para dirigirse a un estacionamiento vacío.
- Visualizar la ruta y el número de estacionamientos vacíos mediante una interfaz de usuario.

1.2. PREGUNTA DE INVESTIGACIÓN

¿Cómo desarrollar e implementar un sistema electrónico de guía y control para un estacionamiento descubierto y con iluminación fija mediante el procesamiento de imágenes obtenidas de cámaras de seguridad en un parqueadero a escala?

1.3. ANTECEDENTES

En la actualidad se cuenta con una muy limitada cantidad de soluciones para el control de lugares disponibles en estacionamientos descubiertos, ya que la única solución hoy en día es el uso de sensores infrarrojos los cuales efectúa una comparación entre la cantidad de vehículos que ingresan y el número de los mismos que salen y aún más pocas son las soluciones electrónicas que indican al conductor una ruta para que se dirija de la manera más rápida a un lugar vacío donde pueda estacionar. Normalmente se encuentran instalados en estacionamientos cubiertos un sistema de monitoreo por medio de sensores infrarrojo que evalúa si el lugar está ocupado o aún está disponible con un led como indicador para que el conductor pueda identificar de una manera más rápida el espacio disponible.

1.4. JUSTIFICACIÓN

Como se mencionó en los antecedentes son muy pocas las soluciones electrónicas que se tienen en la actualidad para ayudar a controlar estacionamientos y guiar los usuarios a lugares disponibles para aparcar. Además estas soluciones carecen de eficacia para identificar rápidamente una ruta para que el conductor se dirija rápidamente a un estacionamiento vacío, teniendo en cuenta que son soluciones que dependen de una infraestructura costosa y muy extensa, pues cada sensor infrarrojo solo soporta un estacionamiento y requiere de una conexión de datos y su respectiva alimentación. Por el contrario el proyecto en desarrollo permite con una sola fuente de información que nos brinda una cámara, obtener el número de vehículos que se encuentran dentro del estacionamiento, brindar exactamente qué lugares son los que se encuentran ocupados y otorgando la dirección de un estacionamiento vacío, creando una ruta óptima para que el usuario se dirija al lugar final de aparcamiento, evitando que este tenga que conducir por todo el lugar en busca de un espacio vacío, mencionando que con una sola cámara el algoritmo es capaz de monitorear un buen número de lugares de forma simultánea.

Lo que finalmente propone una solución económica, que permite sin necesidad de hacer mayores adaptaciones un control confiable y eficaz para que los usuarios de un estacionamiento descubierto puedan dirigirse a su lugar final de aparcamiento sin mayor esfuerzo y stress, además de brindan al administrador información sobre la cantidad de vehículos que se encuentran dentro del lugar y de manera conjunta tener un sistema de vigilancia.

1.5. DELIMITACIONES

Esta solución opera de una manera óptima en un estacionamiento con las siguientes característica:

- Debe ser un estacionamiento descubierto ya que la cámara se ubica en un lugar alto para tener una imagen de toda el área.

- La iluminación debe ser fija ya que el algoritmo no cuenta con filtros para adecuar la imagen.

- El color de las delimitaciones de los lugares deben ser amarillas, ya que el procesamiento de la imagen tiene un filtro para eliminar reconocer este contorno en específico.

- Para implementar el algoritmo a otro tipo de distribución de estacionamiento, este debe ser acoplado al número de lugares y la distribución que se desea implementar.

2. TEORÍA DE CONJUNTOS

Las operaciones de conjuntos son fundamentales para la aplicación de la morfología de imágenes, debido a que con estos conceptos es posible realizar la transformación de las ilustraciones. A continuación se visualiza una figura a la que se adjunta su representación en matriz numérica. [1]

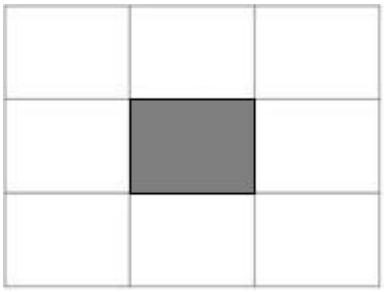


Ilustración 2.Imagen cuadrado

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

0	0	0
0	1	0
0	0	0

Ilustración 3. Cuadrado matriz

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

La imagen binaria (figura 1) corresponde al momento de realizar el procesamiento digital de imágenes a una matriz, donde podemos ver que “0” es la ausencia de color y “1” es el recuadro azul. Para imágenes a blanco y negro sucede lo mismo, solo que los intervalos no son tan drásticos como los binarios, sino que tendremos una gama de tonalidades desde “0” (blanco) hasta “255” (negro). De la misma forma existen diferentes formatos de imágenes como el RGB o HSV que también cuentan con intervalos de 0-255, pero que se componen de más de una matriz para tener color, contraste o brillo, dependiendo del formato de la imagen.

2.1. OPERACIONES BÁSICAS DE CONJUNTOS

En este grupo se encuentran las operaciones comunes entre conjuntos (unión, intersección, etc.), aplicada a matrices que al interactuar permiten que logremos filtros morfológicos.

2.1.1 Unión

La unión entre dos conjuntos de imágenes es fácil de visualizar, al igual que la unión entre dos matrices booleanas, por ello, a continuación mostraremos de cómo se realiza el proceso con un ejemplo.

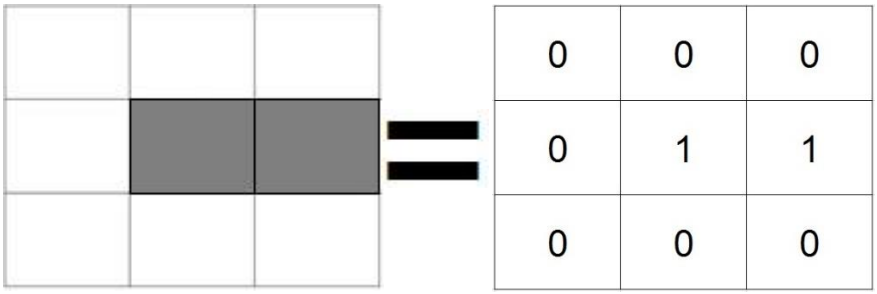


Ilustración 4. Imagen A

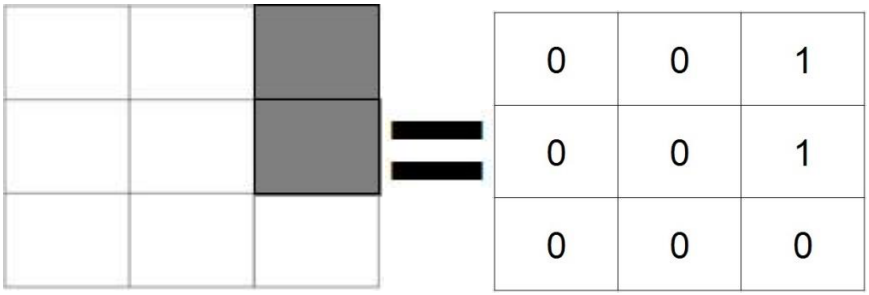


Ilustración 5. Imagen B

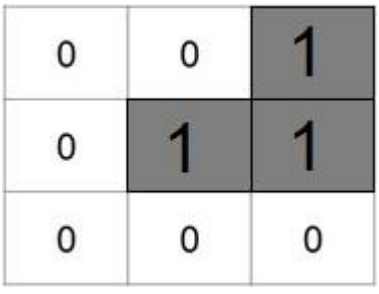


Ilustración 6. A unión B

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

En la Figura 4 se visualiza el resultado de la unión entre dos conjuntos A y B, donde están los valore de cada una de las matrices y las pigmentaciones de los pixeles. Esto mismo sucede con imágenes a grandes escalas o en otros formatos donde se toman los valores matriciales y se fusionan en un conjunto resultante “unión”.

$$\boldsymbol{A} \cup \boldsymbol{B} = \{\boldsymbol{x} | \boldsymbol{x} \in \boldsymbol{A} \vee \boldsymbol{x} \in \boldsymbol{B}\}$$

Ecuación 1. Unión

2.1.2 Intersección

La intersección entre conjuntos tiene como resultado obtener la parte en común entre las dos matrices. Como resultado se visualiza un tercer conjunto y matriz que muestra las similitudes de las dos matrices bases.

$$\boldsymbol{A} \cap \boldsymbol{B} = \{\boldsymbol{x} | \boldsymbol{x} \in \boldsymbol{A} \wedge \boldsymbol{x} \in \boldsymbol{B}\}$$

Ecuación 2. Intersección

0	0	0
0	0	1
0	0	0

Ilustración 7. A intersección B

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

2.1.3 Complemento

Al hablar del complemento de una matriz o un conjunto, se puede asimilar la idea de estar observando el negativo de la misma, es decir, que al obtener este tipo de propiedad en una imagen, sus valores se invierten. En una matriz booleana se puede apreciar el complemento de una imagen al realizar un cambio de 0-1 o viceversa, en cambio, para una matriz de imagen en otro tipo de formato es algo más complejo donde sus valores normalmente

de 0-255 se invierten relativamente, donde un pixel que podría valer “1” en su complemento puede valer “254”.

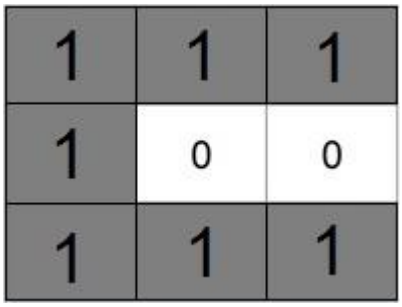


Ilustración 8.Complemento A

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

En la figura 6. Se evidencia lo que es el complemento del conjunto booleano A, donde sus valores correspondientes a cada posición de la matriz se invierten.

$$A^c = \{\boldsymbol{x} | \boldsymbol{x} \in \boldsymbol{U} \wedge \boldsymbol{x} \notin \boldsymbol{A}\}$$

Ecuación 3. Complemento

2.1.4 Diferencia simétrica

De la misma manera cómo podemos obtener las similitudes entre dos matrices, también existe una operación entre dos conjuntos que nos da la posibilidad de obtener las diferencias entre los mismos. En dicha operación las imágenes se sobreponen y eliminan sus partes similares dentro de las posiciones matriciales

$$\boldsymbol{A} \Delta \boldsymbol{B} = \boldsymbol{A} \cup \boldsymbol{B} \setminus \boldsymbol{A} \cap \boldsymbol{B} = (\boldsymbol{A} \setminus \boldsymbol{B}) \cup (\boldsymbol{B} \setminus \boldsymbol{A})$$

Ecuación 4. Diferencia Simétrica

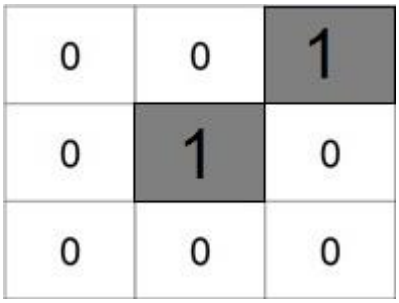


Ilustración 9. Diferencia simétrica A y B

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

3. MORFOLOGÍA

La morfología es el uso de operaciones básicas de conjuntos que permite obtener información centralizada de una imagen a partir de la interacción de la misma con una estructura. Dependiendo del tipo de estructura y la operación que se aplique, obtendremos una deformación de la imagen que nos permitirá extraer determinada información de la matriz.
Se conoce que en las matemáticas morfológicas las imágenes se estudian desde el punto de vista de un conjunto, debido a que es más sencillo poder manipular elementos individuales y flexibles. Por ello se afirma que el conjunto de las interrelaciones entre estos elementos den la estructura del objeto. Teniendo en cuenta estos aspectos de la morfología, se indicaran algunos usos de la misma.

- Pre-procesamiento de imágenes (supresión de ruidos, simplificación de formas).
- Destacar la estructura de los objetos (extraer el esqueleto, detección de objetos, envolvente convexa, ampliación, reducción).
- Descripción de objetos (área, perímetro).

La morfología en imágenes posee operaciones básicas que nos permiten extraer la información que deseamos de la matriz base, como las que se expondrán a continuación.

3.1. PROPIEDADES DE LAS OPERACIONES MORFOLÓGICAS

Debido a que muchas de las operaciones morfológicas son irreversibles y pueden causar la perdida de información, esta maneja dicha perdida mediante sucesivas transformaciones. Por ello se explicaran algunas de las propiedades de las transformaciones morfológicas, relacionada a su capacidad de controlar la perdida de información en las imágenes.[2]

3.1.1 Creciente

Se asimila que una transformación morfológica está dentro de las propiedades crecientes, cuando esta mantiene la relación de inclusión de elementos conjuntos. Tal como lo expresa la siguiente ecuación.

$$X \subset \mathrm{Y} \Rightarrow \Psi(\mathrm{X}) \subset \Psi(\mathrm{Y})$$

Ecuación 5. Creciente

3.1.2 Antiextensividad

Una transformación se considera antiextensiva cuando esta contrae o reduce el conjunto, tal que.

$$\Psi(\mathrm{X}) \subset \mathrm{X}$$

Ecuación 6. Antiextensividad

3.1.3 Idempotencia

Se consideran Idempotencia a aquellas operaciones que al aplicarlas dos veces consecutivas no presentan cambios, así que.

$$\Psi[\Psi(\mathrm{X})] = \Psi(\mathrm{X})$$

Ecuación 7. Idempotencia

3.1.4 Homotopía

La homotopía se puede comparar a un árbol genealógico de elementos. A continuación se anexa una imagen para asociar mejor la teoría.

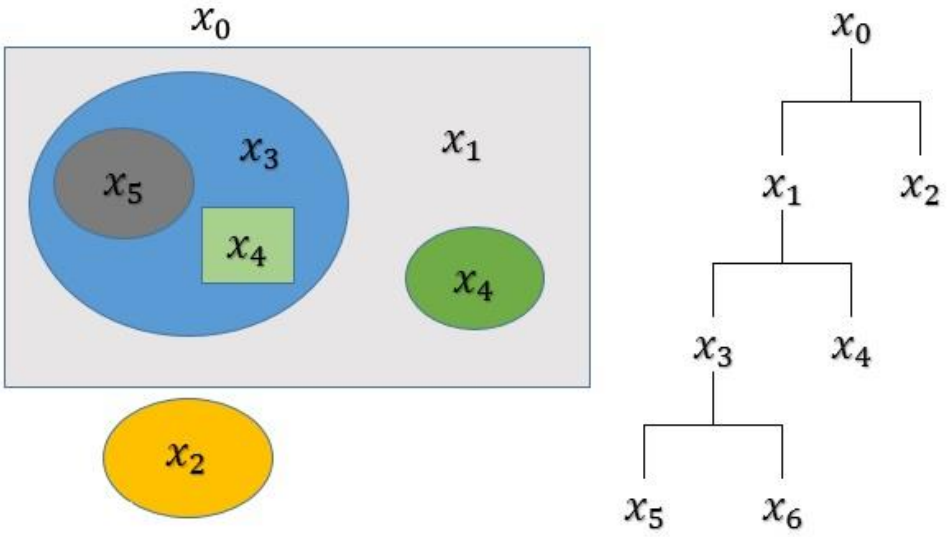


Ilustración 10. Homotopía

Fuente: Elaboración propia.

3.2. OPERACIONES MORFOLÓGICAS

Las transformaciones de imágenes mediante la morfología permiten que se resalten los aspectos e información que más interesan o llaman la atención del mismo. Como consecuencia, esto logra que el objeto en estudio después de ser sometido a alguna operación morfológica sea uno totalmente diferente al inicial, quizás más expresivo debido a que resalta el aspecto de interés.

Las operaciones morfológicas se pueden clasificar dependiendo de sus propiedades. A continuación se anexa un cuadro que permite relacionar cada operación a su propiedad.

CRITERIOS	PROPIEDADES			
	Creciente	Extensivo	Idempotente	Homotopía
Complementación, inclusión o exclusión, contorno	NO	NO	NO	NO
Erosión y dilatación (cuando el origen no pertenece al elemento estructural), filtrado medio	SI	NO	NO	NO
Adelgazamiento y engrosamiento	NO	SI	NO	NO
Erosión y dilatación (cuando el origen ∈ al elemento estructura)	SI	SI	NO	NO
Contorno del contorno	NO	NO	SI	NO
Proyección, filtrado morfológico		NO	SI	NO

Adelgazamiento secuencial, esqueleto, erosión final	NO	SI	SI	NO
Apertura, cierre, corteza convexa, distribución de tamaño, minimización, umbral.	SI	SI	SI	NO
Desplazamiento, similitud, afinidad, simetría	SI	NO	NO	SI
Adelgazamiento y engrosamiento homotópicos y secuenciales	NO	SI	SI	SI

Tabla 1. Propiedades de las operaciones morfológicas

Fuente: S. y Z. X. HARALICK, R., STERNBERG, “Image analysis using mathematical morphology,” pp. 69–72.

3.3. DILATACIÓN Y EROSIÓN

El objetivo de las transformaciones básicas de la morfología es la extracción de información utilizando un conjunto estructurante, el tamaño y forma del elemento es de elección dependiendo de la finalidad de la transformación.

3.3.1 Dilatación

La dilatación es una transformación que permite a dos conjuntos combinarse mediante el uso de la adición vectorial de sus elementos. Esta técnica es implementada para suavizar las imágenes, realizar extracción de formas y obtener sus parámetros[2].

Ahora bien existiendo los conjuntos A y B. Al desarrollar la dilatación del primero por el segundo, tendremos la siguiente definición matemática siendo A subconjunto de E^n y donde $x \epsilon E^n$. La translación de A por x se representa por $(A)_x$ y se define por:

$$(A)_x = \{c \epsilon E^n|_{c=a+x}\ para\ algún\ a \epsilon A\}$$

Ecuación 8. Dilatación definición 1

Sean A y B subconjuntos de E^n, la dilatación de A por B se puede mostrar como $A \oplus B$ y se puede definir como.

$$A \oplus B = \{c \epsilon E^n|_{c=a+b} para\ algún\ a \epsilon A\ y\ b \epsilon B\}$$

Ecuación 9. Dilatación definición 2

El utilizar la definición de la dilatación en términos generales no es muy productivo, como solución se presenta una alternativa que supone

considerar que la dilatación de A por B puede ser calculada como la unión de translaciones de A por los elementos de B:

$$A \oplus B = \bigcup_{b \epsilon\, a} (A)_b$$

Ecuación 10. Dilatación definición simplificada

Se puede demostrar el efecto de la dilatación en la siguiente imagen donde el elemento estructurante A aumenta la definición del objeto B. [3]

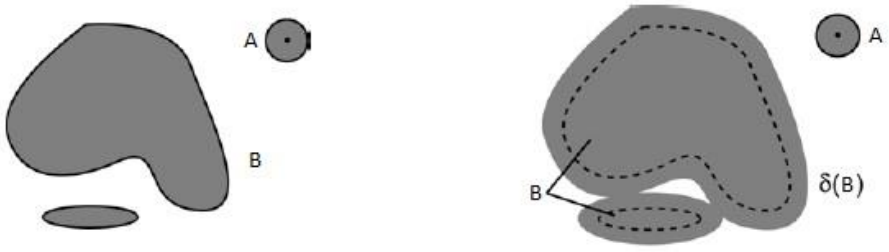


Ilustración 11. Dilatación

Fuente: P. I. D. Tema, “Tema 5 : Morfología Primera parte Morfología.”

De esta forma se aprecia como el elemento estructurante “A” con un valor de dilatación para el pixel (x, y) es el mayor valor de la imagen en la ventana de vecindad cuando su origen se posiciona en (x, y). [4]

Las siguientes ilustraciones explican de una manera más específica lo que sucede en la transformación de dilatación en una imagen en la escala de cada pixel.

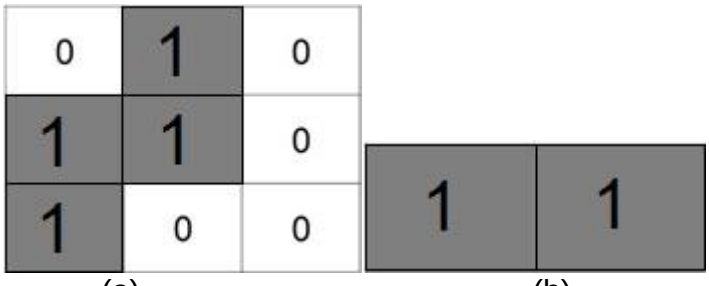


Ilustración 12. (a)Imagen (b) elemento estructurante

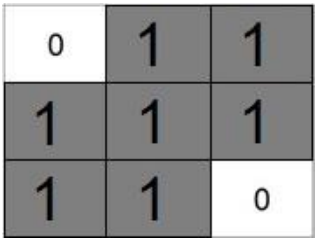


Ilustración 13. Respuesta Dilatación

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

3.3.2 Erosión

Se infiere que la erosión es la operación morfológica dual de la dilatación, ya que este tipo de transformación toma dos conjuntos y los combina utilizando el vector resta de sus elementos.

La transformación de erosión está asociada con el condicional de sí el elemento estructurante está totalmente incluido dentro del conjunto, de lo contrario el resultado es un conjunto vacío. [5]

De esta forma tenemos un escenario donde sea el conjunto $A \subseteq E^n$ y su complemento representa la porción del espacio **euclidiano** no ocupado por dicho conjunto, entonces el complemento de A, representado por A^c, será:

$$A^c = \{ X \in E^n |_{x \notin A} \}$$

Ecuación 11. Definición erosión 1

El complemento de un conjunto para la morfología, representa que invierte la posición de sus elementos en cuanto a sus coordenadas de filas y columnas. Ahora considérese el conjunto $B \subseteq E^n$, se denota por $\bar{B}$ a su conjunto simétrico respecto al origen, donde será:

$$\bar{B} = \{X \mid para\ algún\ b \in B, x = -b\}$$

Ecuación 12. Definición erosión 2

Para denotar esta transformación ***Minkowski*** decide representarla mediante el símbolo”⊖”, y finalmente se puede reducir de la manera:

$$A \ominus B = \bigcap_{b \in B} (A)_{-b}$$

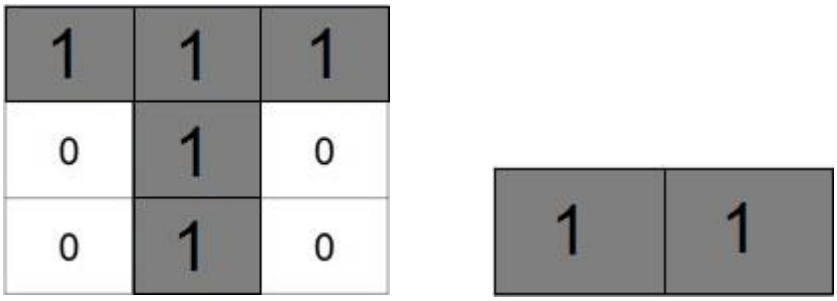
Ecuación 13. Definición Erosión simplificada

Se puede demostrar el efecto de la erosión en la siguiente imagen donde el elemento estructurante A disminuye la definición del objeto B. [3]

Ilustración 14. Erosión

Fuente: P. I. D. Tema, “Tema 5 : Morfología Primera parte Morfología.”

A continuación se explica de una forma gráfica, como interactúa una imagen y un elemento estructurante al realizar una transformación por erosión, siendo:



(a) (b)

Ilustración 15. (a) Imagen (b) elemento estructurante

1	1	0
0	0	0
0	0	0

Ilustración 16. Respuesta erosión

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

3.4. APERTURA Y CIERRE

En transformaciones básicas generalmente es muy difícil poder restaurar la imagen a su estado anterior, dicha acción solo reconstituye ciertas partes del conjunto base. El resultado se considera como la figura esencial y como consecuencia obtenemos realmente un subconjunto que es muy rico en propiedades morfológicas y distribución de tamaño. Aclarado esto aparecen dos operaciones importantes para el procesamiento morfológico.

3.4.1 Apertura

La apertura es una transformación morfológica que elimina elementos y suaviza la imagen, además de suprimir franjas o zonas de un objeto que sean más estrechas que el elemento estructurante.[4]

Se define una imagen B a la cual se le realiza una apertura por un elemento estructurante K $(B \circ K)$, donde se tiene en cuenta que el proceso requiere de una erosión seguida de una dilatación de forma que.

$$B \circ K = (B \ominus \overline{K}) \oplus K$$

Ecuación 14. Definición apertura 1

Ahora desde un punto geométrico, se infiere que: “La apertura de A por B es la unión de todas las translaciones de B que están contenidas en A”[6]. Por ende se tiene:

$$A \circ \mathrm{B} = \left\{ \mathrm{x} \in E^n |_{para\ algún\ Y}, x \in B_y \subseteq A \right\} = \bigcup_{[y|B_y \subseteq A]} B_y$$

Ecuación 15. Definición apertura simplificada

A continuación se anexa un ejemplo del proceso de una imagen A al ser sometida a una transformación de apertura por un elemento estructurante circular B.

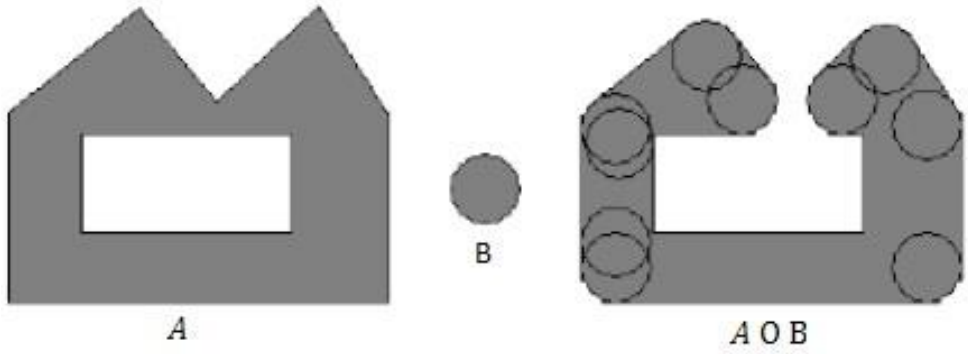
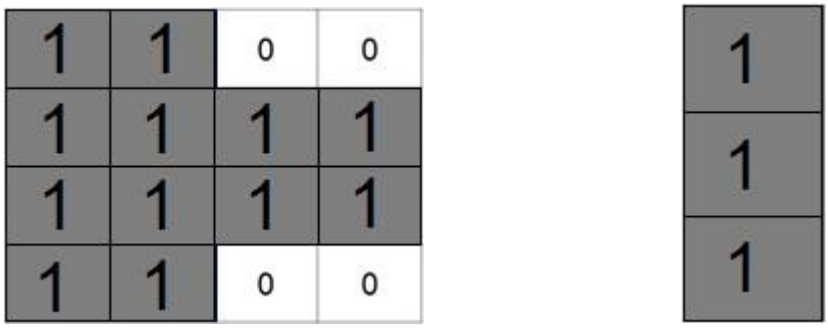


Ilustración 17. Apertura

Fuente: P. I. D. Tema, “Tema 5 : Morfología Primera parte Morfología.”

De forma que el proceso a escala más pequeña es representada por las siguientes figuras que simulan el mismo procedimiento desde una mirada de pixel a pixel.



(a) (b)

Ilustración 18. (a)Imagen (b) Elemento estructurante

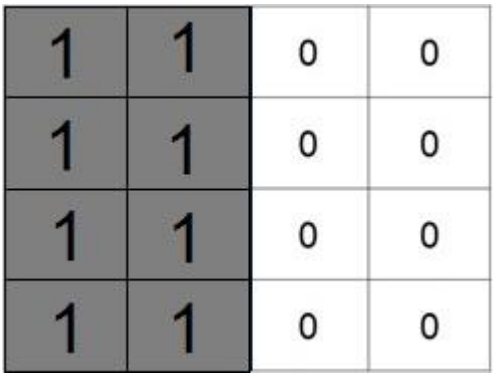


Ilustración 19. Respuesta apertura

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

3.4.2 Cierre

El cierre permite suavizar los contornos de las imágenes, funde roturas estrechas y alarga los golfos delgados, además de eliminar los orificios pequeños y reconstituir espacios en los contornos. Para obtener estos efectos sobre una imagen tendremos que el cierre se crea a partir de una dilatación seguida de una erosión, donde.

$$B \cdot K = (B \oplus \overline{K}) \ominus K$$

Ecuación 16. Definición cierre 1

Ahora bien a una ilustración A, se le aplica una transformación de cierre mediante el uso del elemento estructurante B y se evidencia que desde el punto de vista geométrico el cierre de A por B es la complementación de todas las translaciones de $\bar{B}$ que están contenidas en A^c. Definiendo que.

$$A \cdot B = [\bigcup_{\{y|B_y \subseteq A^c\}} \overline{B_y}]^c$$

Ecuación 17. Definición cierre simplificada

Aplicando este teoría a una imagen con un elemento estructurante circular obtendremos que.

Ilustración 20. Cierre

Fuente: P. I. D. Tema, “Tema 5 : Morfología Primera parte Morfología.”

De forma que el proceso a escala más pequeña es representada por las siguientes figuras que simulan el mismo procedimiento desde una mirada de pixel a pixel.

1	1	0	0
0	0	0	1
1	1	1	0
0	0	0	0

(a)

1
1

(b)

Ilustración 21. (a)Imagen (b) Elemento estructurante

1	1	0	0
1	1	0	0
1	1	0	0
0	0	0	0

Ilustración 22.Respuesta cierre

Fuente: G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.”

4. OpenCV

Las siglas OpenCV provienen de los términos “Open Source Computer Vision Library”. Por lo que se concluye que es una librería para diferentes plataformas que se fundamenta en el tratamiento de imágenes y además principalmente se usa en visión por computador en tiempo real.[7]

Esta librería se compone de 4 módulos.[8]

- Cv: Contiene las funciones principales de la biblioteca.
- Cvaux: contiene las funciones auxiliares (experimental).
- Cxcore: contiene las estructuras de datos y funciones de soporte para algebra lineal.
- Highgui: funciones para manejo de la GUI.

4.2. TIPOS DE DATOS EN OPENCV

OpenCV provee tipos de datos básicos para su uso, pero también posee otros que son introducidos como un plus para el programador, lo que permite que el acceso a determinada información sea más sencilla.

4.2.1 CvArr

Este es un tipo de dato denominado “metatype”, el cual es ficticio y se usa de forma genérica cuando describimos los parámetros de las funciones. Su nomenclatura tiene como objetivo informar que permite la utilización de cualquier tipo de arrays.

4.2.2 IpIImage

Es uno de los tipos de datos más utilizados, debido a que con él se representan todas las imágenes sin importar su extensión de archivo.

4.2.3 CvMat

CvMat se caracteriza por permitir almacenar los elementos como cualquier matriz y además ofrece la posibilidad de acceder a información adicional. Normalmente está asociado con la función cvCreateMat, la cual permite configurar la estructura matricial de una manera muy sencilla.

4.2.4 CvScalar

Esta estructura es un vector que se constituye de cuatro elementos y que es muy útil cuando se desea acceder a los pixeles de una imagen.

4.2.5 CvPoint y CvPoint2D32f

Son usadas para definir las coordenadas de un punto. Sus diferencias radican en que el primero de ellos se define con número entero, mientras que en el segundo se usan números en punto flotante.

4.2.6 CvSize

Es útil cuando se desea crear una imagen nueva, debido a que permite definir y crear las dimensiones de un rectángulo en pixeles.

4.3. MOMENTOS DE HU

Los 7 momentos de Hu se basan en instantes donde la imagen es invariante a la rotación. Para explicar sus características se procede a reemplazar el momento geométrico por un *"momento geométrico central normalizado*" η_{pq} para el cual es necesario obtener un momento central desde uno geométrico μ_{pq} y un momento normalizado $\eta_{pq,}$ desde este momento central, de esta forma se obtiene la escala y las características invariantes de rotación.[9]

Se define en un momento geométrico general en dos dimensiones y para una imagen de tamaño M x N.

$$m_{pq} = \sum_{x=0}^{x=M-1} \sum_{y=0}^{y=N-1} (x)^p \cdot (y)^p \, f(x,y)$$

Ecuación 18. Momento Geométrico General

Ahora para una imagen binaria, una superficie o un contorno se lleva a cabo por m_{00} por lo que el centro del contorno puede calcularse a partir de.

$$\bar{x} = \frac{m_{10}}{m_{00}} \, \bar{y} = \frac{m_{01}}{m_{00}}$$

Ecuación 19. Centro del contorno

Donde X y Y son puntos en los ejes X y Y respectivamente y por consiguiente podemos transformar el momento mencionado anteriormente en un momento invariante por la replantación en un momento central, dado que.

$$\mu_{pq} = \sum_{x=0}^{x=M-1} \sum_{y=0}^{y=N-1} (x - \bar{x})^p \cdot (y - \bar{y})^q f(x, y)$$

Ecuación 20. Momento invariante

Además un momento central normalizado es una escala invariante, por lo tanto un momento central puede ser transformado en un momento normalizado ya que.

$$y = [(p + q)/2] + 1$$
$$\eta_{pq} = \mu_{pq}/\mu_{00}^{y}$$

Ecuación 21. Momento normalizado

Por ultimo para obtener un momento de rotación invariante aplicamos η_{pq} en siete características de Hu en momentos geométricos estándar.

Como conclusión Hu definió de M1 a M6 como invariantes absolutos ortogonales y M7 como una invariante ortogonal skew (útil para imágenes espejo). Las cuales son eficaces para reconocer objetos simples.

$$M_{pq} = \iint_{D} P_{pq}\,(x, y) f(x, y)\, dx\, dy$$

Ecuación 22. Característica Hu 1

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p\, y^q f(x, y)\, dx\, dy$$

Ecuación 23. Característica Hu 2

$$c_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x + iy)^p (x - iy)^q f(x, y)\, dx\, dy$$

Ecuación 24. Característica Hu 3

$$\iint_{\Omega} P_{pq}(x,y) \cdot P_{mn}(x,y)\, dx\, dy = 0$$

Ecuación 25. Característica Hu 4

$$\iint_{\Omega} w(x,y) \cdot P_{pq}(x,y) \cdot P_{mn}(x,y)\, dx\, dy = 0$$

Ecuación 26. Característica Hu 5

$$M_1 = (\eta_{20} + \eta_{02}),$$

$$M_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2,$$

$$M_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2,$$

$$M_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2,$$

$$M_5 = (\eta_{30} - 3\eta_{12})^2(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2],$$

$$M_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} - \eta_{12})(\eta_{21} - \eta_{03}) + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2],$$

$$M_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} + 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2],$$

Ecuación 27. Característica Hu 6

4.4. INSTALACIÓN DE OPENCV PARA SISTEMA LINUX

El primer paso es verificar que se cuenta con los siguientes pre-requisitos para continuar con la instalación de las librerías de OpenCV.

- GCC 4.4.x o posterior.
- CMake 2.6 o posterior.

- Git.
- GTK+2.x o posterior.
- Pkg-config.
- Python 2.6 o posterior y Numpy 1.5 o posterior con paquetes para desarrolladores.
- Ffmpeg o libav con paquetes para desarrolladores: libavcodec-dev, libavformat-dev, libswscale-dev.

Teniendo estos paquetes en el equipo, se procede a seguir una de las dos siguientes formas de instalación de OpenCV.

4.4.2 Instalación desde los repositorios de Ubuntu o Debian

Para realizar la instalación se debe insertar el siguiente comando en la terminal del equipo.

```
sudo apt-get install libopencv-dev python-opencv
```

Sin embargo, puede que por medio de este método no se descargue y compile la última versión de OpenCV y es probable que se pierdan algunas características.

4.4.3 Instalación desde el sitio oficial de OpenCV

Como primer paso para instalar la última versión de OpenCV, se debe retirar la biblioteca desde el repositorio con.

```
sudo apt-get autoremove libopencv-dev python-opencv
```

Y a continuación siga los siguientes pasos.

- Tener Ubuntu o Debian actualizado.

```
sudo apt-get update && sudo apt-get upgrade && sudo
apt-get dist-upgrade && sudo apt-get autoremove
```

- Instalar las dependencias.

Build tools:

```
sudo apt-get install build-essential cmake
```

GUI:

```
sudo apt-get install qt5-default libvtk6-dev
```

Media I/O:

```
sudo apt-get install zlib1g-dev libjpeg-dev libwebp-dev libpng-dev libtiff5-dev libjasper-dev libopenexr-dev libgdal-dev
```

Video I/O:

```
sudo apt-get install libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev yasm libopencore-amrnb-dev libopencore-amrwb-dev libv4l-dev libxine2-dev
```

Parallelism and linear algebra libraries:

```
sudo apt-get install libtbb-dev libeigen3-dev
```

Python:

```
sudo apt-get install python-dev python-tk python-numpy python3-dev python3-tk python3-numpy
```

Java:

```
sudo apt-get install ant default-jdk
```

Documentación:

```
sudo apt-get install doxygen
```

- Descargar y descompilar OpenCV

  Ingrese al sitio web oficial de OpenCV y descargue la última versión para Linux. Después descomprima el archivo.

- Compile e instale OpenCV

  Se debe ingresar al directorio de OpenCV y a continuación en el terminal se escriben los siguientes comandos.

```
mkdir build
```

```
cd build
```

```
cmake -DWITH_QT=ON -DWITH_OPENGL=ON -DFORCE_VTK=ON -DWITH_TBB=ON -
```

```
DWITH_GDAL=ON -DWITH_XINE=ON -
DBUILD_EXAMPLES=ON ..
```

```
make -j4
```

```
sudo make install
```

Ahora, se debe compilar OpenCV.

```
sudo ldconfig
```

5. Proyecto

Para el desarrollo del programa se debe tener en cuenta que está fundamentado mediante el lenguaje de programación “Python”

5.1. Librerías

Las Librerías son módulos que se incorporan en el encabezado del programa con el fin de poder hacer uso de funciones ya predefinidas. Cada librería contiene un número de funciones con un enfoque en particular y que podemos usar de la siguiente forma.

```
import cv2
import numpy as np
from tkinter import *
import tkinter
```

- Cv2: Las funciones de OpenCv nos permite realizar todo el procesamiento digital de imagen que requerimos para el desarrollo del proyecto. Es una librería enfocada en la manipulación de videos y fotos en una gran gama de formatos que nos brinda aproximadamente 2500 algoritmos optimizados.

- Numpy: Esta librería nos permite crear, modificar y operar arreglos de matrices. Numpy posee funciones que simulan operaciones de algebra lineal y transformadas de Fourier.

- Tkinter: Tkinter es un módulo compatible con Python que permite crear y desarrollar ventanas para visualizar un entorno grafico que interactúe con el usuario. Es posible mostrar tablas, imágenes, crear algún tipo de menú o también adquirir información por parte del operador.

5.2. Captura de video e imagen

OpenCv dispone de funciones para acceder e interactuar con salidas externas de video, es de esta forma que se obtiene el acceso a una WebCam, la cual está situada en la parte superior del parking y con la que se desea capturar las fotos que permiten hacer el test en tiempo real.

Para inicializar y visualizar la imagen de este dispositivo mediante el código se estructura el siguiente código:

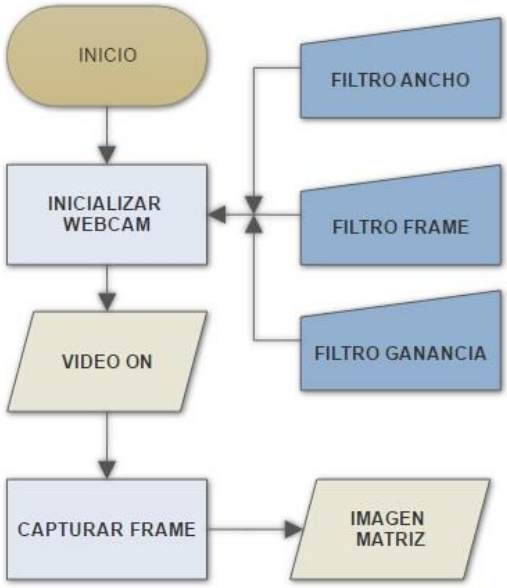


Ilustración 23. Diagrama de flujo adquirir imagen desde WebCam

Fuente: creado con herramienta online gliffy

```
cap = cv2.VideoCapture(1)#cargar WebCam.
cap.set(3,1280)          #filtro posición.
cap.set(4,1024)          #filtro ancho frames.
cap.set(15, 0.1)         #filtro ganancia imagen.
ret, img = cap.read()    #captura de frame
```

En primera instancia debemos asignar una variables “cap” que permite obtener acceso a la WebCam. Se asigna como variable de ingreso “1”, ya que la cámara se encuentra conectada mediante un puerto USB, de no ser externa el valor a ingresar debe ser “0”. – Además VideoCapture nos permite mediante una función anidada “set()” activar y calibrar filtros para la imagen, se decide hacer uso de 3 de ellos para mejorar la resolución y la calidad de la imagen con la que se trabajara todo el procesamiento.

Luego de obtener acceso y filtro en el video, se procede a capturar una imagen en un instante de tiempo (frame) con el fin de procesar y adquirir la información relevante mediante una foto estática. La función “read()” captura un instante del video para almacenarlo en la variable “img” la cual se almacena como una arreglo compuesto de 3 matrices que componen todos los parámetros de la imagen.

5.3. Morfología y detección de área

Además de ofrecernos la obtención de imagen por medio de una WebCam, OpenCv incluye funciones morfológicas que permiten adquirir información relevante de la imagen.

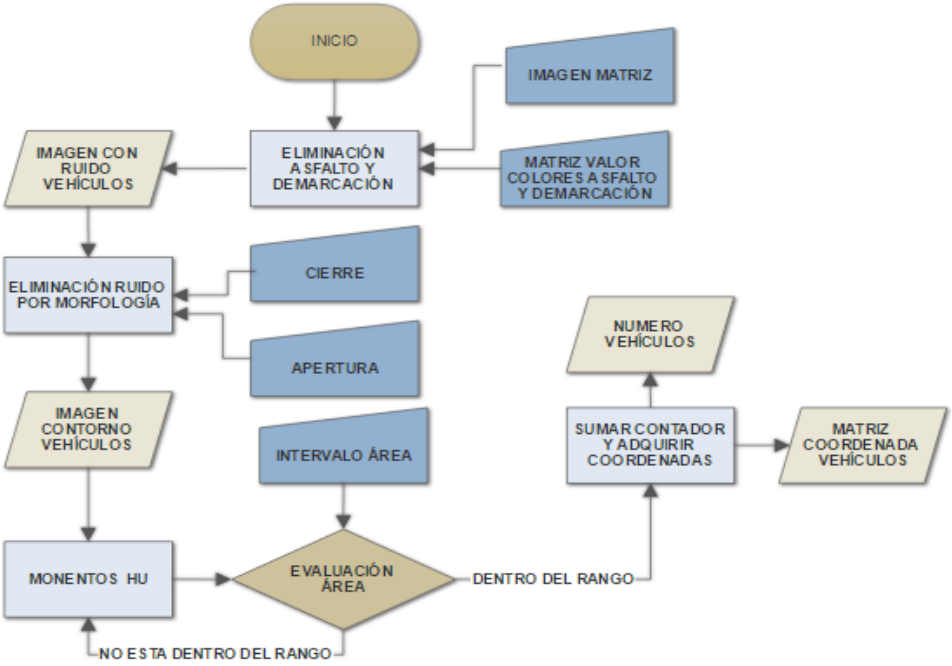


Ilustración 24. Diagrama flujo morfología y momentos de HU

Fuente: creado con herramienta online gliffy

Con el fin de determinar qué cantidad de vehículos se encuentran dentro del parking, se hace uso de dos operaciones morfológicas (closeth y open) lo que permite depurar las señalizaciones de parqueo, el asfalto y algún otro tipo de objeto que pueda generar interferencia en la detección de los contornos de los vehículos, además se deben crear arreglos de matrices para generar mascaras de la imagen, OpenCv no dispone de este tipo de variables por lo que se debe utilizar las funciones arrays que nos ofrece la librería Numpy. Es de esta forma que se posee:

```
#matriz para detección de colores del asfalto
boundaries = [
    ([20, 20, 20], [80, 80, 80])
]

#direccionamiento de matriz
(lower, upper) = boundaries[0]

# crea los arreglos para detectar colores
lower = np.array(lower, dtype="uint8")
upper = np.array(upper, dtype="uint8")

# crea la máscara dependiendo del umbral seleccionado, en
este caso el color gris del suelo
mask = cv2.inRange(image, lower, upper)
cv2.bitwise_not(mask, mask)

# Arreglos morfológicos de cerrado y apertura para eliminar
el ruido
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (35, 35))
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)

#crea una copia de mask e image para obtener el contorno de
cada vehículo
thresh = mask.copy()
image2 = image.copy()
_, contours, _ = cv2.findContours(thresh,cv2.RETR_TREE,
                 cv2.CHAIN_APPROX_SIMPLE)

#momento de la imagen, detección del área
area = cv2.contourArea(cnt)

#filtro de contornos, depende del tamaño del area (X*Y
pixels)
if area < 3000 or area > 5300:
    continue
# Puntos necesarios para el rectángulo o elipse
if len(cnt) < 4:
    continue

#evalúa y crea un rectángulo con la posición en coordenadas
(x,y) y alto-ancho (w,h) del momento
x, y, w, h = cv2.boundingRect(cnt)
cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

De esta forma tenemos que para una imagen base.

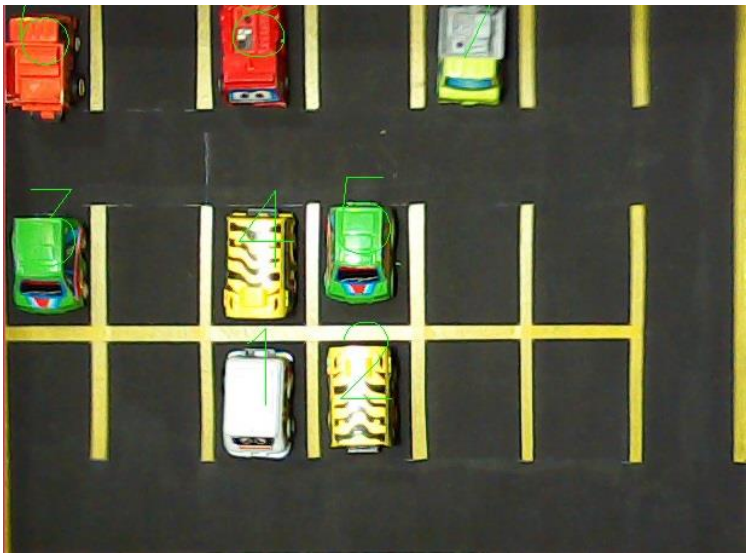


Ilustración 25. Imagen base parking

Fuente: obtenida por medio de webcam, maqueta a escala.

Aplicando los filtros para objetos no necesarios y las operaciones morfológicas en los momentos de la imagen, obtendremos como resultado.

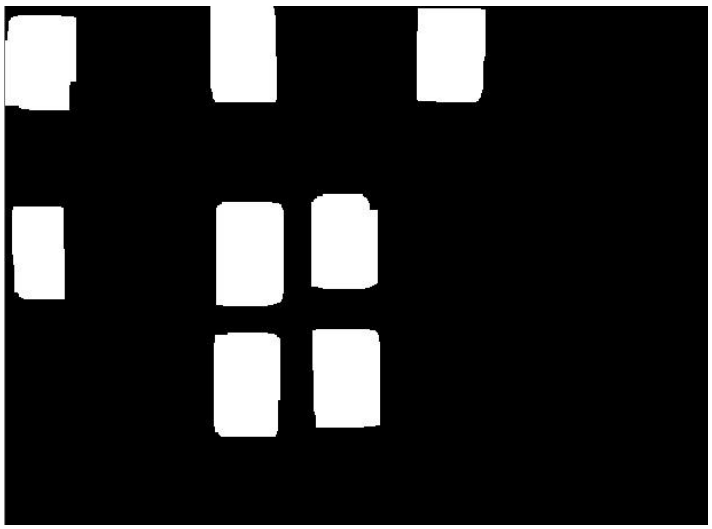


Ilustración 26. Morfología de la imagen

Fuente: obtenida por medio de herramienta cv2.imshow() Spyder

Posterior a obtener la máscara de los contornos, se verifica que estos pertenezcan a los vehículos que se encuentran dentro del parking.

Ilustración 27. Mascara

Fuente: obtenida por medio de herramienta cv2.imshow() Spyder

Finalmente ya que uno de los objetivos del procesamiento es determinar cuántos autos están estacionados, mediante la aplicación de los momentos de la imagen se logra detectar una a una las áreas activas dentro de la imagen asignándola a una variable que actúa como un contador y nos guardara el dato exacto de los vehículos.

5.4. Búsqueda de lugares disponible y ruta de parqueo

Luego de tener definido un contorno sobre los vehículos, es posible crear una máscara binaria para representar lo lugares disponibles y aquellos que ya se encuentran llenos, con el fin de crear un barrido que permita identificar qué espacio aún está disponible y cuál es la ruta más cercana para que el conductor llegue a ella.

Creado el arreglo matricial anterior, se puede proceder a desarrollar el siguiente algoritmo que permite identificar las coordenadas de inicio y llegada para el conductor, y el paso a paso para que este llegue a su destino de la siguiente manera.

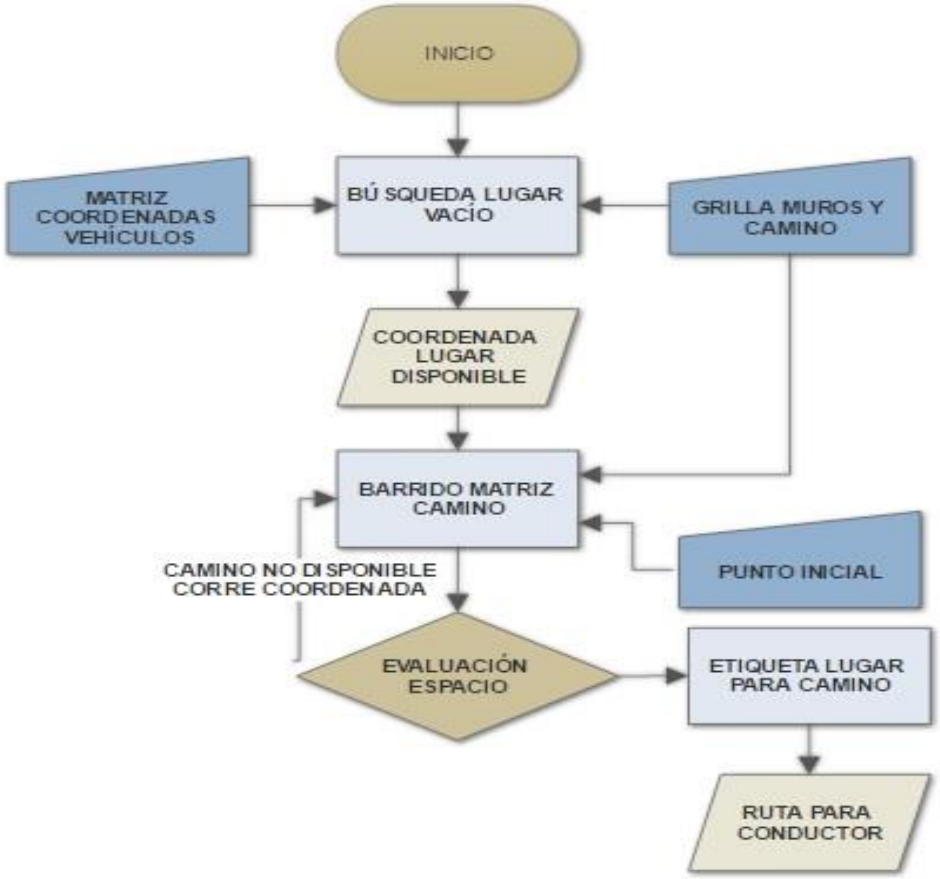


Ilustración 28. Diagrama flujo búsqueda lugar disponible y camino.

Fuente: obtenida por medio de herramienta online gliffy

```
#punto de partida
init = [5, 12]

#coordenadas de movimiento
cost = 1
delta = [[-1, 0], # arriba
         [ 0,-1], # izquierda
         [ 1, 0], # abajo
         [ 0, 1]] # derecha

#etiquetas Sting de los movimientos
delta_name = ['^', '<', 'v', '>']

#camino
grid = [[0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0],
        [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0],
        [0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

```
#reinicio de matriz
test3 = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

#lugares de parqueo
test3 =[[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

#inserta etiquetas en la matriz
def show(matrix):
    rows = ['[' + ']['.join(r) + ']' for r in matrix]
    print (' ' + ',\n '.join(rows) )

#barrio para encontrar espacio disponible
def search(grid,init,goal,cost):

    lim_Y = len(grid)
    lim_X = len(grid[0])
    openList = []
    openList.append([0]+init)
    expand = [[-1 for row in range(len(grid[0]))] for col in
              range(len(grid))]
    expand[init[0]][init[1]] = 0
    estado = True
    rot = [[ ' ' for row in range(len(grid[0]))] for col in
            range(len(grid))]
    contador = 0

    while(estado):
        selec1 = 99
        selec2 = 0

        for i in range (len(openList)):
            if (openList[i][0]<selec1):
                selec1 = openList[i][0]
                selec2 = i

        cord_X = openList[selec2][2]
        cord_Y = openList[selec2][1]
        grid[cord_Y][cord_X] = 3
        costS = openList[selec2][0]+cost
        openList.pop(selec2)
        expand[cord_Y][cord_X] = contador
        contador += 1
```

```
        for i in range (len(delta)):
            tcord_Y = delta[i][0] + cord_Y
            tcord_X = delta[i][1] + cord_X

            if((tcord_Y >= 0) and (tcord_Y < lim_Y) and
               (tcord_X >= 0) and (tcord_X < lim_X)and
               (grid[tcord_Y][tcord_X] == 0)):
                if([tcord_Y,tcord_X] == goal):
                    rot[tcord_Y][tcord_X] = '*'
                    expand[tcord_Y][tcord_X] = 99
                    estado = False
                openList.append([costS,tcord_Y,tcord_X])
                grid[tcord_Y][tcord_X] = 3

        if (len(openList) == 0):
            return '0cupos'

    estado = 0
    cord_Y = goal[0]
    cord_X = goal[1]
    min_E = 99
    max_E = 0

    while(estado<costS):
        for i in range (len(delta)):
            tcord_Y = delta[i][0] + cord_Y
            tcord_X = delta[i][1] + cord_X

            if((tcord_Y >= 0) and (tcord_Y < lim_Y) and
(tcord_X >= 0) and (tcord_X < lim_X) and
(expand[tcord_Y][tcord_X] > -1)):
                if (min_E > expand[tcord_Y][tcord_X]):
                    tempO = [tcord_Y] +[tcord_X]
                    min_E = expand[tcord_Y][tcord_X]
                    max_E = (i+2)%4

        rot[tempO[0]][tempO[1]] = delta_name[max_E]
        cord_Y = tempO[0]
        cord_X = tempO[1]
        min_E = 99
        max_E = 0
        estado += 1
    return rot
    state = 0
    posY = goal[0]
    posX = goal[1]
    minE = 99
    maxE = 0

    while(state<costS):
        for i in range (len(delta)):
            tPosY = delta[i][0] + posY
            tPosX = delta[i][1] + posX

            if((tPosY >= 0) and (tPosY < maxY) and (tPosX >=
```

```
                0)and (tPosX < maxX) and
                (expand[tPosY][tPosX] > -1)):
                if (minE > expand[tPosY][tPosX]):
                    tempO = [tPosY] +[tPosX]
                    minE = expand[tPosY][tPosX]
                    maxE = (i+2)%4

        route[tempO[0]][tempO[1]] = delta_name[maxE]
        posY = tempO[0]
        posX = tempO[1]
        minE = 99
        maxE = 0
        state += 1
    return route
```

5.5. Interfaz de usuario

Toda la parte visual y de interacción con el usuario se desarrolla con el uso de la librería Tkinter, la cual permite crear y editar ventanas visuales en el sistema operativo. De esta forma se instancia una matriz que representa el modelo del parqueadero y muestra mediante flechas la ruta que debe seguir el conductor para llegar al destino de parqueo, además se anexa en la parte inferior de la visualización la cantidad de lugares disponibles que existe en tiempo real dentro del parking.

Para crear un mapa más entendible para el conductor se crea también una función que permite evaluar si la posición de la matriz corresponde a un muro, el lugar de destino, el camino o asfalto y asigna un background para cada tipo de variable.

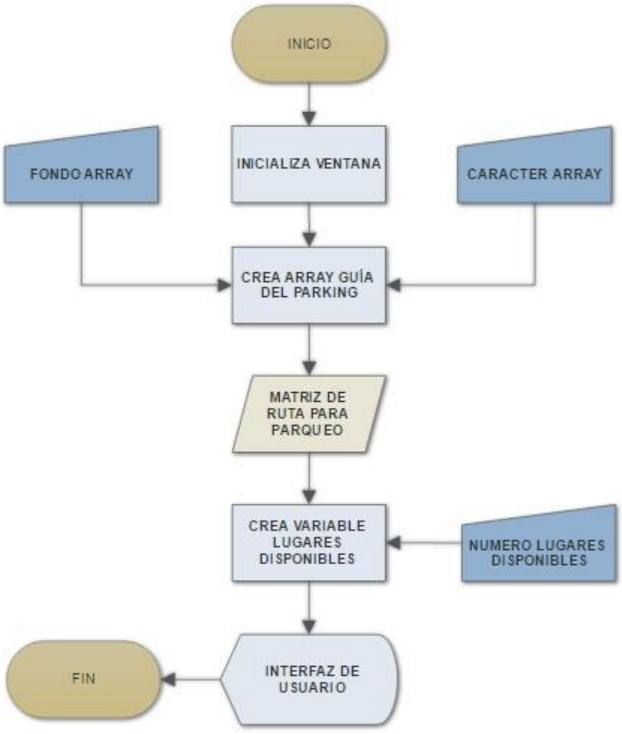


Ilustración 29. Diagrama flujo interfaz de usuario

Fuente: obtenida por medio de herramienta online gliffy

```
#asigna background a las casillas de visualización
def b_g (y,x):
    if camino[y,x] == "*":
        return "green"
    elif camino[y,x] != " ":
        return "white"
    elif y == 3:
        return "yellow"
    elif x%2 != 0  and y%2 == 0:
        return "yellow"
    elif camino[y,x] == " ":
        return "gray"

root = tkinter.Tk( )
for r in range(6):
    for c in range(13):
        #escribe la casilla
        tkinter.Label(root, text=camino[r,c], relief=RAISED
                      , bg = b_g(r,c), borderwidth=10
                      ).grid(row=r,column=c, padx=1, pady=1)

#crea una variable tipo String
var = StringVar()
label = Label( root, textvariable=var, relief=RAISED
              ).grid(row=6, column=4 ,columnspan=6,
              sticky=W+E+N+S, padx=5, pady=10)
```

```
#instancia el mensaje a visualizar en var
var.set("Lugares Disponibles = " + str(18 - contador))
```

De esta manera tendremos como resultado una ventana como la siguiente que representa en color amarillo todas las sub-divisiones correspondientes a las cintas guías de parqueo, gris el suelo del parking, blanco la ruta a seguir por el conductor para llegar a su destino y color verde el punto final donde se encuentra un lugar disponible de parqueo. La estructura debe ser igual a la imagen real del parqueadero, con un número real de estacionamientos y caminos disponibles para el conductor. Puesto que es la referencia del usuario con el parking

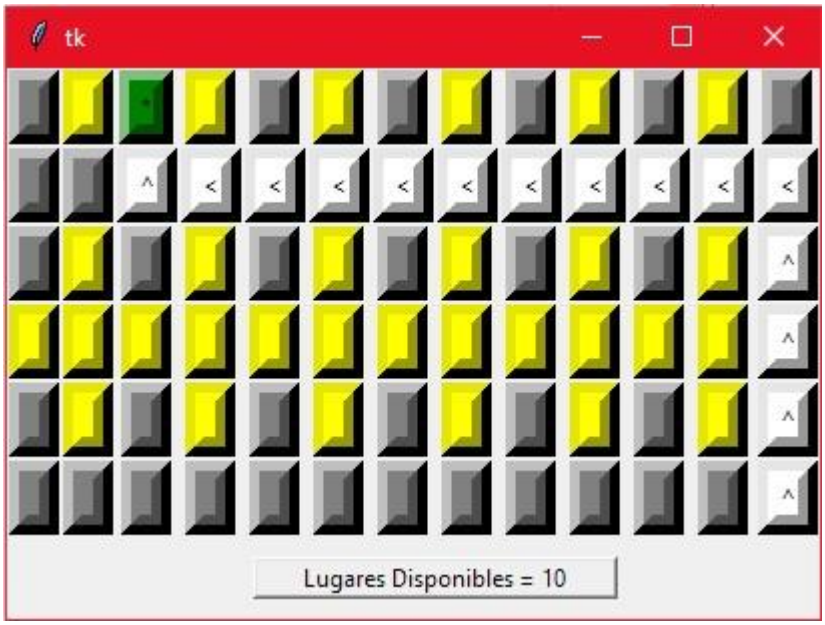


Ilustración 30. Interfaz

Fuente: obtenida por medio de herramienta Tkinter en programa Spyder

6. ANEXOS

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 12 15:12:39 2016

@author: steven
"""
import cv2
import numpy as np
from tkinter import *
import tkinter
import time

#matriz para detección de colores del asfalto
boundaries = [
    ([20, 20, 20], [80, 80, 80])
]
#punto de partida
init = [5, 12]
#direccionamiento de matriz
(lower, upper) = boundaries[0]
#coordenadas de movimiento
cost = 1
delta = [[-1, 0], # arriba
         [ 0,-1], # izquiera
         [ 1, 0], # abajo
         [ 0, 1]] # derecha

#etiquetas Sting de los movimientos
delta name = ['^', '<', 'v', '>']
#cargar WebCam.
cap = cv2.VideoCapture(1)
#filtro posición.
cap.set(3,1280)
#filtro ancho frames.
cap.set(4,1024)
#filtro ganancia imagen.
cap.set(15, 0.1)


#inserta etiquetas en la matriz
def show(matrix):
    rows = ['[' + '][' .join(r) + ']' for r in matrix]
    print (' ' + ',\n '.join(rows) )

#barrio para encontrar espacio disponible
def search(grid,init,goal,cost):

    lim_Y = len(grid)
    lim_X = len(grid[0])
    openList = []
    openList.append([0]+init)
    expand = [[-1 for row in range(len(grid[0]))] for col in  range(len(grid))]
    expand[init[0]][init[1]] = 0
    estado = True
    rot = [[ ' ' for row in range(len(grid[0]))] for col in range(len(grid))]
    contador = 0

    while(estado):
        selec1 = 99
        selec2 = 0

        for i in range (len(openList)):
            if (openList[i][0]<selec1):
                selec1 = openList[i][0]
```

```
                selec2 = i

        cord_X = openList[selec2][2]
        cord_Y = openList[selec2][1]
        grid[cord_Y][cord_X] = 3
        costS = openList[selec2][0]+cost
        openList.pop(selec2)
        expand[cord_Y][cord_X] = contador
        contador += 1

        for i in range (len(delta)):
            tcord_Y = delta[i][0] + cord_Y
            tcord_X = delta[i][1] + cord_X

            if((tcord_Y >= 0) and (tcord_Y < lim_Y) and
               (tcord_X >= 0) and (tcord_X < lim_X)and
               (grid[tcord_Y][tcord_X] == 0)):
                if([tcord_Y,tcord_X] == goal):
                    rot[tcord_Y][tcord_X] = '*'
                    expand[tcord_Y][tcord_X] = 99
                    estado = False
                openList.append([costS,tcord_Y,tcord_X])
                grid[tcord_Y][tcord_X] = 3

        if (len(openList) == 0):
            return '0cupos'

    state = 0
    posY = goal[0]
    posX = goal[1]
    minE = 99
    maxE = 0

    while(state<costS):
        for i in range (len(delta)):
            tPosY = delta[i][0] + posY
            tPosX = delta[i][1] + posX

            if((tPosY >= 0) and (tPosY < maxY) and (tPosX >= 0) and (tPosX < maxX) and (expand[tPosY][tPosX] > -1)):
                if (minE > expand[tPosY][tPosX]):
                    tempO = [tPosY] +[tPosX]
                    minE = expand[tPosY][tPosX]
                    maxE = (i+2)%4

        route[tempO[0]][tempO[1]] = delta name[maxE]
        posY = tempO[0]
        posX = tempO[1]
        minE = 99
        maxE = 0
        state += 1
    return route


#asigna background a las casillas de visualización
def b_g (y,x):
    if camino[y,x] == "*":
        return "green"
    elif camino[y,x] != " ":
        return "white"
    elif y == 3:
        return "yellow"
    elif x%2 != 0  and y%2 == 0:
        return "yellow"
    elif camino[y,x] == " ":
        return "gray"


#barrido en la matriz test para encontrar el valor "0"
def buscarElemento(lista, elemento):
```

```
    for i in range(0,len(lista)):
        lista = np.asarray(lista)
        for j in range(0,len(lista.transpose())):
            if(lista[i][j] == elemento):
                return i,j
    return 5,12

while True:
    #captura de frame
    ret, img = cap.read()
    #muestra img
    cv2.imshow("input", img)
    #guarda img como "test.jpeg"
    cv2.imwrite('test.jpeg', img)
    key = cv2.waitKey(10)

    if key == 27:
        #lee y carga "test.jpeg"
       image = cv2.imread("test.jpeg")
        #recorta image
       image = image[25:478,26:639]
        # crea los arreglos para detectar los colores
        lower = np.array(lower, dtype="uint8")
        upper = np.array(upper, dtype="uint8")
        # crea la máscara dependiendo del umbral seleccionado, en este caso el color gris del suelo
        mask = cv2.inRange(image, lower, upper)
        cv2.bitwise not(mask, mask)
        # Arreglos morfológicos de cerrado y apertura para eliminar el ruido
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
        kernel = cv2.getStructuringElement(cv2.MORPH RECT, (35, 35))
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
        #crea una copia de mask e image para obtener el contorno de cada vehículo
        thresh = mask.copy()
        image2 = image.copy()
        _, contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        contador = 0

        for cnt in contours:

            #momento de la imagen, detección del area
            area = cv2.contourArea(cnt)
            # Detección de contornos, depende del tamaño del area X*Y pixels
            if area < 3000 or area > 5300:
                continue
            # Puntos necesarios para el rectángulo o elipse
            if len(cnt) < 4:
                continue
            contador += 1
           #evalúa y crea un rectángulo con la posición en coordenadas (x,y) y alto-ancho (w,h) del momento
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(image, str(contador), (x, y + int(h/2)), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0))
           #función para coordenadas de los momentos de la imagen
            xx = int(int(x)/80)*2
            yy = int(int(y)/100)*2
            test3[yy][xx] = 1

        posiciony, posicionx = buscarElemento(test3, 0)
        goal = [posiciony, posicionx]
        camino = np.array(search(grid,init,goal,cost))
        #inicializa la ventana
        root = tkinter.Tk( )
        for r in range(6):
            for c in range(13):
               #escribe la casilla
               tkinter.Label(root, text=camino[r,c], relief=RAISED , bg = b_g(r,c),
                             borderwidth=10 ).grid(row=r,column=c, padx=1, pady=1)
```

```
        #crea una variable tipo String
        var = StringVar()
        label = Label( root, textvariable=var, relief=RAISED ).grid(row=6, column=4
                        ,columnspan=6, sticky=W+E+N+S, padx=5, pady=10)
       #instancia el mensaje a visualizar en var
        var.set("Lugares Disponibles = " + str(18 - contador))
        break

cv2.destroyAllWindows()
cv2.VideoCapture(0).release()
root.mainloop( )
```

Anexo 1. Código final.

Anexo 2. Maqueta a escala Parking

Anexo 3. Soporte WebCam.

Anexo 4. Maqueta completa.

7. CONCLUSIONES

- Es posible manipular imágenes al comprender su relación y estructura matricial.
- La morfología matemática permite obtener datos puntuales de la imagen pues es una herramienta muy efectiva para depurar e identificar segmentos de la matriz a partir de operaciones entre conjuntos matriciales.
- Python posee los recursos necesarios para efectuar un procesamiento digital de imagen con métodos actualizados y eficiencia.
- Los momentos de HU permiten complementar la búsqueda de datos para imágenes variantes en su posición y tamaño, facilitando el proceso en ilustraciones móviles y/o rotatorias.

- La iluminación ambiente para el procesamiento de la imagen es una variable indispensable que de no ser valorada en el algoritmo causa incertidumbre en los resultados esperados.

8. RECOMENDACIONES

- Complementar el desarrollo del proyecto en un escenario nocturno, ya que el presente solo se estipula en un entorno con luz día.
- Tener presente en el momento de desarrollar la simulación los parámetros iniciales para reconocimiento de color del asfalto, debido a que varían según el nivel de iluminación ambiente. Si no se tiene encuentra esta variable, el algoritmo podría no reconocer los vehículos.
- Para obtener una imagen optima que abarque sin mayor altura la totalidad de los lugares de parqueo, se debe estudiar el tipo de lente que debe tener la cámara con la que se dispone hacer el proyecto.
- Se debe tener presente que la detección del área en el código es variable dependiendo de la resolución de la imagen, si no se calibra es probable que los resultados no sean verídicos.
- Tener presente para la evaluación del área que para esta simulación se utilizó una cámara con resolución de 640x480 pixeles.

9. BIBLIOGRAFIA

[1] G. V. Emmanuelle Gouillart, “Manipulación y procesamiento de imágenes usando Numpy y Scipy.” [Online]. Available: http://pybonacci.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html.

[2] G. Aguilar, “Procesamiento Digital De Imágenes Utilizando Filtros Morfológicos,” 1995.

[3] Francisco Gabriel Ortiz Zamora, “Procesamiento Morfológico de Imágenes en Color . Aplicación a la Reconstrucción Tesis Doctoral,” 2002.

[4] P. I. D. Tema, “Tema 5 : Morfología Primera parte Morfología.”

[5] C. Platero, “Capítulo 6 : Procesamiento morfológico Procesamiento Morfológico Necesidad del post-procesado Geometría y forma Conocimiento a priori Teoría de Conjuntos No lineal.”

[6] S. y Z. X. HARALICK, R., STERNBERG, “Image analysis using mathematical morphology,” pp. 69–72.

[7] C. Medrano, “Tutorial de OpenCV.”

[8] A. Furfaro, “Manejo de Bibliotecas Opencv Primer Ejemplo .,” 2010.

[9] R. Sabhara, “Comparative Study of Hu Moments and Zernike Moments in Object Recognition,” *Smart Comput. Rev.*, vol. 3, no. 3, pp. 166–173, 2013.