DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL MULTIAGENTE PARA LA SEGMENTACIÓN DE VIDRIO EN VENTANAS DE RASCACIELOS

Autor: Ing. Artur Mosquera Mykh

Director: PhD. Oscar Gualdrón Guerrero

Universidad de Pamplona

Facultad de Ingenierías y Arquitectura

Maestría en Controles Industriales

2019

DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL MULTIAGENTE PARA LA SEGMENTACIÓN DE VIDRIO EN VENTANAS DE RASCACIELOS

Trabajo de Grado para optar al título de
Magister en Controles Industriales

Autor: Ing. Artur Mosquera Mykh
Director: PhD. Oscar Gualdrón Guerrero

Universidad de Pamplona
Facultad de Ingenierías y Arquitectura
Maestría en Controles Industriales
2019

DEDICATORIA

A mi padre Jemay Mosquera, mi madre Eleonora Moskera y a mi hermano Daniel Mosquera, por darme el apoyo incondicional durante mis estudios de maestría. Mis logros se los debo a ustedes, gracias por los consejos, comprensión y la constante motivación para alcanzar mis metas.

AGRADECIMIENTOS

A mi director de tesis Oscar Gualdrón Guerrero por sus aportes y recomendaciones, a Jhon Arias y Doris Ospina por su apoyo al proyecto, a mis compañeros de maestría y al grupo de investigación de Automatización y Control.

CONTENIDO

LISTA DE FIGURAS

RESUMEN

El número de rascacielos construidos y en construcción en los últimos años evidencia la dinámica de crecimiento de los rascacielos y, con los requerimientos y tecnológicas empleadas para efectuar la limpieza de sus ventanas, se observa una necesidad de desarrollar nuevas alternativas que realicen esta tarea de forma más segura, rápida y precisa. Este trabajo presenta el desarrollo de un sistema de visión artificial multiagente para la segmentación de vidrio en ventanas de rascacielos, el cual está diseñado para ser integrado en el futuro a un prototipo de brazo robótico y un dron, con el fin de realizar el trabajo de limpieza de ventanas de rascacielos. La función del sistema desarrollado es la de segmentar el vidrio y, a partir de la segmentación, obtener los parámetros de la altura, el ancho y la posición en los ejes X, Y y Z del centro del vidrio con respecto a la posición de la cámara.

En la investigación se presenta una introducción teórica de los métodos empleados para el desarrollo del sistema mencionado, un estado del arte de la visión artificial, y la metodología y resultados que guían la selección de los métodos empleados. Los resultados incluyen la compilación de una base de datos de ventanas de rascacielos con una amplia variedad de condiciones físicas y de iluminación; la segmentación del vidrio evaluada de forma preliminar empleando algoritmos de detección de bordes, tales como Canny, Sobel, Prewitt, Roberts y Laplaciano de Gaussiano; las propiedades de región enfocadas a áreas rectangulares; y la detección de líneas utilizando transformada de Hough. Adicionalmente, se presenta la implementación de técnicas de inteligencia artificial como las redes neuronales convolucionales junto con segmentación semántica, y la integración de todos los métodos expuestos anteriormente en un sistema de visión artificial multiagente para la segmentación de vidrio empleando un árbol de decisión. Por último, se propone una forma de lograr la conversión de pixeles a sistema métrico de los parámetros entregados por el sistema multiagente, con el fin de demostrar que el sistema desarrollado puede ser implementado. Lo anterior se logra siguiendo las ecuaciones matemáticas de la relación entre la imagen tomada por la cámara, la medida estimada por un sensor de distancia y los parámetros intrínsecos de la cámara.

Palabras clave: rascacielos, segmentación, algoritmos de bordes, transformada de Hough, red neuronal convolucional, sistema multiagente.

ABSTRACT

The number of skyscrapers under construction and built in recent years demonstrates the growth dynamics of skyscrapers and, with the requirements and technologies used to clean their windows, show a clear need to develop new alternatives that perform this task in a safer, faster and more accurate way. This work presents the development of a multi-agent artificial vision system for glass segmentation in skyscraper windows, which is designed to be attached in the future into a robotic arm prototype and a drone, in order to perform the work of skyscraper window cleaning. The function of the developed system is to segment the glass and, from the segmentation, obtain the parameters of the height, width and position in the X, Y and Z axes of the glass's center in relation to the camera's position.

The research presents a theoretical introduction of the methods used for the development of the mentioned system, a state of the art of artificial vision, and the methodology and results that guided the selection of the methods used. The results include the compilation of a database of skyscraper windows with a wide variety of physical and lighting conditions; a preliminary glass segmentation program using edge detection algorithms such as Canny, Sobel, Prewitt, Roberts and Laplacian of the Gaussian, region properties focused on rectangular areas, and line detection using Hough transform. Additionally, the implementation of artificial intelligence techniques such as convolutional neural networks is presented along with semantic segmentation, and all the methods set forth above integrated into a multi-agent artificial vision system for glass segmentation using a decision tree. Lastly, a way to achieve the conversion of pixels into the metric system of the parameters delivered by the multi-agent system is proposed, in order to demonstrate that the developed system can be implemented. The above is achieved by following the mathematical equations of the relationship between the image taken by the camera, the measurement made by a distance sensor and the intrinsic parameters of the camera.

Keywords: skyscrapers, segmentation, edge algorithms, Hough transform, convolutional neural network, multi-agent system.

CAPÍTULO 1: INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

La dificultad de la limpieza de ventanas de rascacielos ha estado en aumento en los últimos años, debido principalmente a que estos son cada vez más altos, con fachadas más complejas, y el número que se construyen anualmente está en constante incremento. Los rascacielos a diferencia de los edificios convencionales, no permiten abrir sus ventanas para efectuar la limpieza desde adentro, por lo cual requieren que estas sean limpiadas desde el exterior. Teniendo en cuenta el aumento en los costos y tiempo que toma el proceso de limpieza convencional, así como el peligro al que se someten los limpiadores de ventanas de rascacielos, se evidencia que a lo largo de los años se han creado alternativas tecnológicas para facilitar la limpieza de estas ventanas.

Una de estas alternativas recientes, es el prototipo de brazo robótico que se está desarrollando con la compañía J&D Trading Group Inc, el cual está diseñado para ser montado sobre un octocóptero de tipo X8 y realizar el trabajo de limpieza de ventanas de rascacielos de una manera más fácil y segura aprovechando los avances tecnológicos de los drones. Este prototipo inicial requiere de un operario para realizar las tareas de limpieza, adicional al requerido por la Federal Aviation Administration (FAA) para volar el dron seleccionado. Debido a esto, luego de realizar el análisis de los costos de operación del modelo propuesto, se llegó a la conclusión de que es indispensable que este sea automatizado con el fin de evitar la necesidad de un operador adicional, y así convertir el prototipo inicial en una alternativa comercial eficiente, rápida y segura para la limpieza de ventanas de rascacielos.

Debido a la variedad de condiciones y características físicas de las ventanas de rascacielos, la parte más importante y extensa de esta automatización es el requerimiento de un sistema de visión artificial robusto y preciso, que sea capaz de indicarle al brazo robótico el lugar donde se encuentra el vidrio de la ventana del rascacielo con respecto al dron, y el tamaño de esta, sin importar el tipo de ventana, condiciones de iluminación u otros factores que influyan en su reconocimiento y la obtención de las variables requeridas.

En el marco de la visión artificial para la segmentación de vidrio, no ha habido un enfoque específico para esta tarea, a excepción de una investigación enfocada en el reconocimiento de ventanas completas para representación tridimensional de edificios, lo cual probablemente se debe a la falta de necesidad o aplicación del reconocimiento exacto del vidrio. Esto no quiere decir que una investigación para este requerimiento parte desde cero, ya que para lograr esta tarea se articulan bases teóricas de otras aplicaciones, tales como los vehículos autónomos, la detección de elementos en fachadas y la detección de señales de tráfico.

Este proyecto incluye la compilación de una base de datos amplia y variada de imágenes de ventanas de rascacielos, extraídas a partir de fotos tomadas por quadcopteros frente a los rascacielos o fotos de sus fachadas capturadas desde edificios opuestos. Esto se logra con la ayuda de páginas web de imágenes y videos de Stock completamente gratuitos o con vista previa gratuita, tales como GettyImages, iStock y ShutterStock. Entre los factores de búsqueda se encuentran las diferentes condiciones de iluminación, marcos de distintos tamaños, grosores y tonalidades, variación entre marcos en una sola ventana, vidrio que permite ver dentro del edificio y los objetos observados, y vidrio completamente reflectante y los objetos que se reflejan.

Posteriormente, se realiza un programa preliminar que cuenta con una comparación visual entre dos métodos para la segmentación de vidrio en ventanas de rascacielos; el primero de los cuales emplea algoritmos de detección de bordes como Canny, Sobel, Prewitt, Roberts y Laplaciano de Gaussiano, junto con filtros y propiedades de región, mientras que el segundo incorpora al primero la transformada de Hough para la detección de líneas.

A partir de los resultados, y con el fin de lograr un programa robusto, se presenta el desarrollo de un programa empleando redes neuronales convolucionales que aprovechan el aprendizaje transferido de una red neuronal existente y la segmentación semántica para realizar una clasificación precisa por pixel.

Todas las combinaciones que dan solución a distintas características encontradas en la base de datos de ventanas, se integran en un programa multiagente empleando un árbol

de decisión, con el fin de dar solución a un mayor número y tipos de ventanas de rascacielos.

Para la validación de los sistemas, se emplea el sistema de visión artificial multiagente resultante en cada imagen de ventanas de rascacielos perteneciente a la base de datos, evaluando la exactitud de los parámetros del vidrio mediante una matriz de validación creada durante la segmentación semántica.

Finalmente, se propone una forma de lograr la conversión de pixeles a sistema métrico de los parámetros entregados por el sistema multiagente, con el fin de demostrar la viabilidad de implementación del sistema, lo cual es realizado siguiendo las ecuaciones matemáticas de la relación entre la imagen tomada por la cámara, la medida estimada por un sensor de distancia y los parámetros intrínsecos de la cámara.

Es importante acotar, que los programas desarrollados fueron realizados empleando el software Mathworks Matlab con ayuda de los paquetes Image Processing Toolbox y Neural Network Toolbox.

Este trabajo de investigación contiene una metodología novedosa, en vista de que los métodos mencionados anteriormente no han sido aplicados en la segmentación de vidrio en ventanas de rascacielos, por lo cual se espera que el presente trabajo se convierta en una base para seguir nuevas investigaciones en este campo.

1.2 OBJETIVOS PLANTEADOS

OBJETIVO GENERAL

Desarrollar un sistema de visión artificial multiagente para la segmentación de vidrio en ventanas de rascacielos.

OBJETIVOS ESPECIFICOS

- Compilar una base de datos amplia y variada de imágenes de ventanas de rascacielos.

- Realizar un programa preliminar que cuente con una comparación visual entre métodos y procedimientos que permitan la segmentación de vidrio en ventanas de rascacielos.

- Identificar la mejor forma para calcular la distancia entre la cámara y el vidrio segmentado.

- Integrar el programa preliminar, métodos de inteligencia artificial y combinaciones de métodos encontrados para las distintas ventanas, en un programa multiagente empleando un árbol de decisión.

- Validar el sistema de visión artificial multiagente empleando la base de datos de imágenes y una ventana real de prueba para la distancia entre el vidrio de la ventana y la cámara.

CAPÍTULO 2: ESTADO DEL ARTE Y MARCO TEÓRICO

El presente capítulo abarca una introducción teórica que conceptualiza la justificación del sistema desarrollado, una introducción generalizada de la visión artificial, un estado del arte de la visión artificial, y un marco teórico enfocado solamente en los métodos, algoritmos y procedimientos empleados en el desarrollo del sistema de visión artificial multiagente para la segmentación de vidrio en ventanas de rascacielos.

2.1 Dinámica de crecimiento de los rascacielos

El número de rascacielos en el mundo está en constante incremento, sin embargo con los recientes avances en la optimización de sus estructuras (Aldwaik & Adeli, 2014), se puede apreciar un aumento considerable con respecto a años anteriores. El año 2017 fue el cuarto año consecutivo en el que se supera el record anterior de número de rascacielos mayores a 200m construidos, con 144 nuevos edificios completados en 2017 (Figura 1), comparado al record anterior de 127 completados en 2016. Esto llevó el número total de edificios de más de 200 metros de altura en el mundo a 1319, mostrando un incremento de 12.3% desde 2016 y marcando un incremento de 402% desde el año 2010. (Council on Tall Buildings and Urban Habitat, 2018).

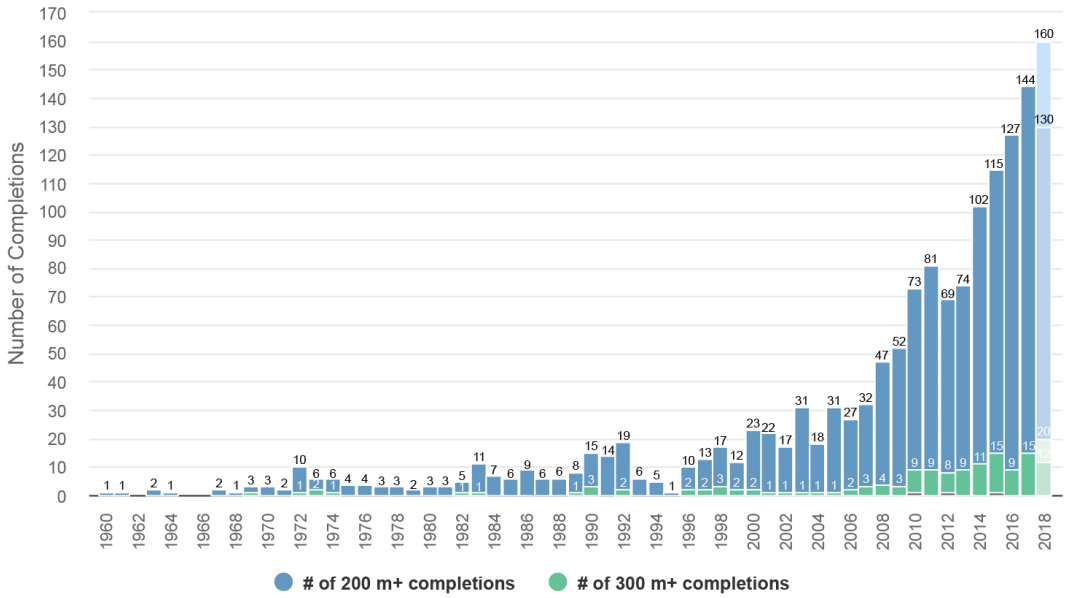

Figura 1. Edificios de más de 200m y 300m completados por año entre 1960 y 2017.

Fuente: Council on Tall Buildings and Urban Habitat, 2018.

Este aumento se debe a que las ciudades en países desarrollados buscan volverse lugares importantes desde los puntos de vista financiero, cultural, tecnológico o turístico, con el fin de atraer inversiones necesarias para crecer. Mientras las ciudades crecen, más personas son necesarias, lo que conlleva a más infraestructura para acomodarlas, culminando en el aumento de demanda de rascacielos (Guedes & Cantuária, 2017; Meissner, 2017).

Al caracterizar la dinámica de crecimiento de los rascacielos, evaluar los requerimientos para efectuar la limpieza de sus ventanas e identificar las tecnológicas empleadas en los últimos años para su limpieza, se hace evidente la necesidad de desarrollar nuevas alternativas que realicen esta tarea de forma más segura, rápida y precisa. Una posible alternativa involucra el aprovechamiento de los avances en el tiempo de vuelo y capacidad de carga de los drones para integrarlos con brazos robóticos diseñados con las limitantes de los drones en mente. Estas dos industrias paralelas de desarrollo rápido combinan sus fortalezas para lograr tareas que antes no eran posibles; los drones pueden llegar a varios lugares, pero están limitados en cuanto a las operaciones que pueden efectuar, mientras que los brazos robóticos tienen un área de trabajo y capacidades extensas, pero están limitados a un espacio en particular (Mosquera, 2017).

2.2 Introducción a la visión artificial

En la década de los cincuentas se presento el surgimiento de las computadoras y el interés de la comunidad científica por la posibilidad de enseñar a estas computadoras a realizar tareas comúnmente asociadas con la inteligencia humana, como la capacidad de resolver problemas, comprender lenguajes o analizar información visual (Poppe, 2010).

La necesidad de analizar la información visual permitió el surgimiento de la visión artificial, la cual es uno de los subcampos de la inteligencia artificial de mayor dinamismo, y su realización conlleva una dificultad importante para las computadoras. El concepto de funcionamiento es semejante al proceso asociado con la visión humana, su insumo básico de entrada es una imagen, la cual es analizada con distintas técnicas y algoritmos para lograr una interpretación o reconocimiento parecido a como lo haría un ser humano. Sin embargo, la mayor ventaja de la visión artificial es que, a diferencia de los ojos y su

capacidad de ver solo el espectro visible, esta no está limitada a una sola forma de capturar, recibir y percibir el entorno en una imagen (Sanabria & Archila, 2011).

Para poder realizar el procesamiento digital de la imagen, esta es representada por un arreglo o matriz de puntos, y se define como una función, *f(x,y)*, donde '*x*' y '*y*' son coordenados espaciales, y la amplitud *f* en cualquier par de coordenadas se llama intensidad o nivel de gris de la imagen en ese punto. Esta imagen digital está compuesta por un número finito de elementos comúnmente llamados pixeles, y cada uno de estos tiene una posición y valor específico (Gonzalez & Woods, 2008).

El procesamiento digital de la imagen se puede dividir en tres categorías: procesos de nivel bajo, medio y alto. Se consideran procesos de bajo nivel a operaciones como el preprocesamiento para la reducción de ruido o aumento de contraste, y se caracterizan por el hecho de que tanto la entrada como la salida resultante es una imagen. El procesamiento de nivel medio involucra tareas como la partición de una imagen en regiones u objetos (segmentación), descripción de esos objetos para reducirlos a una forma apropiada para que el computador los procese, y clasificación y reconocimiento de objetos individuales. Este tipo de procesamiento se caracteriza por el hecho que, aunque la entrada es una imagen, la salida son atributos extraídos de esas imágenes, como por ejemplo bordes o contornos. Por último, un procesamiento de nivel alto involucra un sistema que realice funciones cognitivas normalmente asociadas con la visión humana (Gonzalez, Woods & Eddins, 2004).

El procesamiento de alto nivel se basa en tener el conocimiento de la manera en que se pretende lograr la tarea, partiendo de un modelo formal del entorno, y luego, de que forma la realidad percibida por la imagen digital se compara al modelo. A partir de las diferencias que se presenten, se añaden nuevas tareas que permitan sobreponerse a estas diferencias. Con el fin de lograr lo anterior, se hace un cambio a procesamiento de bajo nivel para encontrar información que permita actualizar el modelo, y se itera el proceso resultante en un bucle donde el procesamiento de alto nivel genera tareas parciales para el procesamiento de bajo nivel (Sonka, Hlavac & Boyle, 2014).

Cuando se presenta un nuevo problema es necesario entender el funcionamiento de los métodos y algoritmos, con el fin de encontrar la metodología que lleve a una posible

solución. Por esta razón, la visión artificial se puede considerar una disciplina que da solución al cómo combinar imágenes derivadas de sensores, modelos previamente obtenidos de la escena y el conocimiento del procesamiento de imágenes, para construir una descripción explicita del modelo del entorno. Este modelo permite resolver una imagen que no ha sido interpretada, haciendo uso de la descomposición de la imagen en partes independientes y coherentes, para que el subsecuente análisis pueda dar solución a una colección de problemas pequeños en lugar de un solo problema complejo (Fischler & Firschein, 2014).

2.3 Tendencias emergentes en el uso de la visión artificial

De forma generalizada y simplificada, la visión artificial comprende los métodos de procesamiento digital de imágenes, clasificación y reconocimiento de patrones u objetos, junto con los sistemas y tecnologías a las que estos pueden ser aplicados. Debido a los avances tecnológicos en cuanto a poder de procesamiento y el tamaño de los procesadores, se ha generado un cambio dinámico en la visión artificial y las tendencias de su uso (Romero & Rolle, 2018).

Entre las aplicaciones más prominentes, y uno de los temas más estudiado en la literatura sobre visión artificial, se encuentra el reconocimiento de rostros, debido principalmente a su uso en la seguridad de la información. Este reconocimiento se define como un sistema biométrico usado para identificar y verificar a una persona a partir de una imagen digital, e involucra la extracción de las características de un rostro y su reconocimiento, sin importar las condiciones de iluminación, expresiones, edad, pose, o translación, rotación y escala de la imagen (Parmar & Mehta, 2014).

Lo anterior ha demostrado ser una tarea compleja que requiere una búsqueda y evaluación de distintos algoritmos y métodos que permitan mejorar la detección, dependiendo de las condiciones mencionadas que puedan presentarse durante la utilización de los sistemas de visión artificial (Cao, Wei, Wen & Sun, 2014; Lu & Zhang, 2016).

En lo referente al reconocimiento de rostros, uno de los métodos actuales más prominentes es el uso de redes neuronales. Sin embargo, debido a una clara falta de

bases de datos publicas a gran escala, la mayoría de los avances recientes siguen confinados a empresas como Facebook y Google. Por ejemplo, uno de los mejores métodos recientes para el reconocimiento de rostros fue logrado por Google mediante redes neuronales convolucionales, y empleando una base de datos de 200 millones de imágenes y 8 millones de identidades únicas (Schroff, Kalenichenko & Philbin, 2015).

Esto no ha detenido a otros investigadores a crear sus propias bases de datos, empleando una metodología de alineamiento afín en 2D para entrenar una red usando combinaciones de imágenes de fuentes disponibles en la red (Figura 2), como por ejemplo las bases de fotos de CelebFaces y WDRef, que cuentan con una base de datos de 2 millones de imágenes (Parkhi, Vedaldi & Zisserman, 2015).

Figura 2. Ejemplo de la base de datos de celebridades con seis identidades.
Fuente: Parkhi, Vedaldi & Zisserman, 2015.

En otros campos como la agricultura, el procesamiento digital de imágenes y la visión artificial han demostrado ser herramientas potentes. Con el paso de los años se han desarrollado más aplicaciones para la supervisión y gestión automática de los procesos hortícolas basadas en imágenes, cuyo objetivo es reducir los costes y aumentar la productividad de los cultivos. Esto se debe a que en la agricultura la calidad del producto es crucial, pero con la gran diversidad y variedad existente en los cultivos, nace la necesidad de aplicar técnicas de visión artificial para analizar imágenes y establecer relaciones entre cada cambio físico externo, y los cambios bioquímicos que acontecen internamente (Cubero, Lee, Aleixos, Albert & Blasco, 2016).

En el caso de la segmentación de suelos y cultivos (Figura 3), este se enfoca más en las condiciones que puedan presentarse en la imagen capturada, tales como sombras, ruido, saturación de pixeles, baja iluminación, variedades de cultivos o parámetros intrínsecos

de las cámaras (Hernández-Hernández, García-Mateos, González-Esquiva, Escarabajal-Henarejos, Ruiz-Canales & Molina-Martínez, 2016).

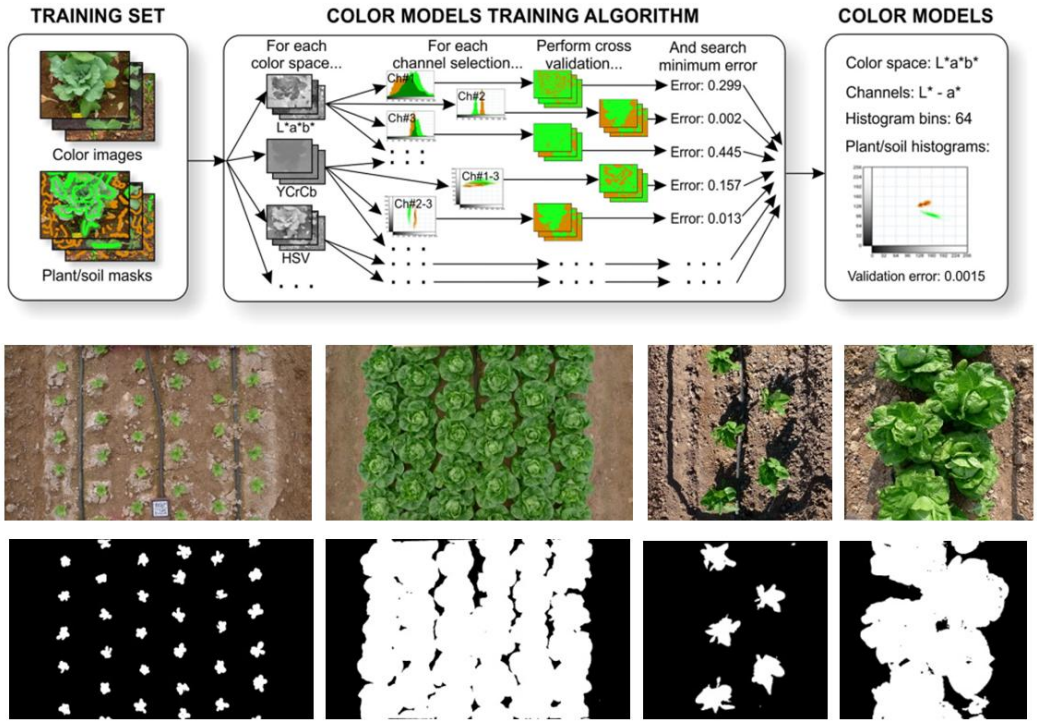

Figura 3. Diagrama del sistema y resultados de la segmentación automática de suelo y plantas.
Fuente: Hernández-Hernández et al., 2016.

La visión artificial en la agricultura, se considera un método rápido y consistente para la inspección de productos, puesto que permite generar una alternativa automática, no destructiva y económica para los requerimientos de calidad y producción que están en constante aumento. La presencia de defectos en la piel de la fruta es uno de los factores que impactan su precio, debido a que los consumidores asocian calidad con una buena apariencia y una total ausencia de defectos externos. Estos defectos son más difíciles de detectar, inspeccionar y categorizar, que el color, forma y tamaño (Li, Huang & Zhao, 2015).

Además de lo anterior, existen más aplicaciones de la visión artificial en la agricultura, como por ejemplo su uso para detectar enfermedades en los cultivos (Chung, Huang, Chen, Lai, Chen & Kuo, 2016), detectar líneas de cultivos para optimizar la irrigación (Diao, Zhao, Song, Wu, Wu, Qian & Wei, 2015), estimar el peso del producto cultivado para su venta (Gongal, Silwal, Amatya, Karkee, Zhang & Lewis, 2016) o para optimizar su

clasificación en la industria alimenticia (Constante, Gordon, Chang, Pruna, Acuna & Escobar, 2016)

Otra aplicación prominente de la visión artificial es la detección de señales de tráfico, placas de auto y otros objetos presentes en las carreteras; esta es empleada principalmente en los vehículos inteligentes y la seguridad vial. En esta aplicación se suelen usar técnicas de procesamiento de bajo nivel, porque se requiere que las regiones se calculen de forma rápida y económica, para que luego estas sean sujetas a métodos más sofisticados de clasificación. Recientemente, técnicas de etiquetado semántico usando redes neuronales convolucionales han logrado excelentes resultados en este campo (Ranft & Stiller, 2016).

Como ejemplo de lo anterior, el mapeo móvil del entorno suele emplearse para la creación de ciudades navegables en 3D, y para encontrar el posicionamiento y los tipos de señales de tráfico a lo largo del recorrido del vehículo, empleando criterios de color y formas. El resultado es una combinación de técnicas 2D y 3D (Figura 4) para acelerar y mejorar la detección del entorno. Sin embargo, la necesidad de emplear técnicas de respuesta rápida presenta una dificultad considerable en las señales de tráfico (Figura 5), donde las variaciones, condiciones de iluminación, oclusión de objetos, posición y orientación varían constantemente (Timofte, Zimmermann & Van Gool, 2014).

Figura 4. Mapeo 3D y detección de señales de tráfico
Fuente: Timofte, Zimmermann & Van Gool, 2014.

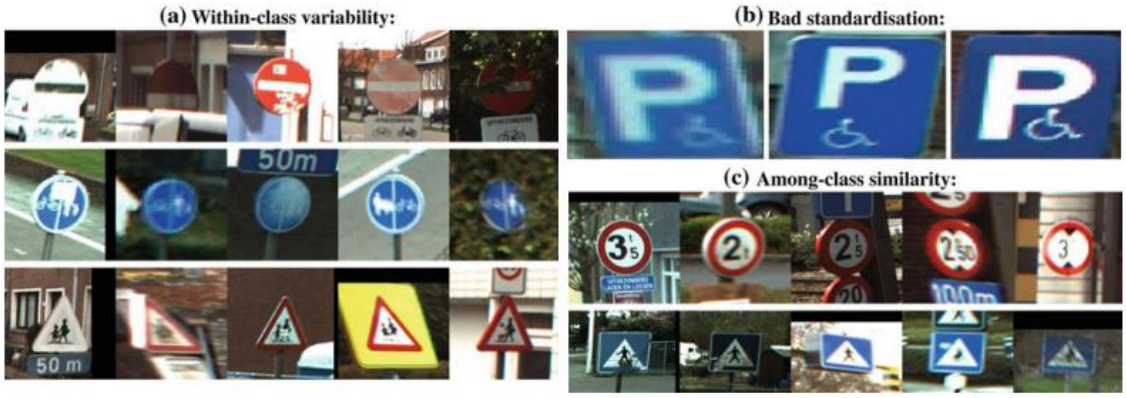

Figura 5. Dificultad en la detección de señales de tráfico. (a) variabilidad dentro de la misma clase. (b) mala estandarización. (c) similitudes entre clases.

Fuente: Timofte, Zimmermann & Van Gool, 2014.

Además de señales con símbolos, en años recientes las investigaciones se han enfocado en el reconocimiento de texto, tanto en señales de tráfico como en placas de autos. Esto se debe a la complejidad y retos que se presentan por la baja resolución de la imagen, efectos de distorsión, fondos complejos, diferentes tipos de letras, color y alineación de estos, cuando son extraídos de videos o imágenes tomadas en movimiento (Bhunia, Kumar, Roy, Balasubramanian & Pal, 2018).

Entre los métodos empleados actualmente para la tarea mencionada anteriormente, se puede encontrar el uso de maquinas de soporte vectorial (Gonzalez, Bergasa & Yebes, 2014) y combinaciones de algoritmos genéticos y redes neuronales (Quiros, Abad, Bedruz, Uy & Dadios, 2015).

A pesar de no ser tan prominente como las aplicaciones anteriores, vale la pena mencionar el uso de la visión artificial en el campo de la fotogrametría, centrándose en las diferentes características de la imagen como color, histograma y texturas, para clasificar los elementos de una fachada en distintas categorías de objetos como ventanas, puertas, muros, cielo, vegetación u otros elementos. Esto se realiza con el fin de mejorar los modelos 3D de los edificios, reconstruir edificios en 3D o realizar análisis de deformaciones a partir de imágenes en alta resolución (Yang, Förstner & Chai, 2012).

Existen bases de datos que ayudan en esta tarea como la de eTRIMS (Figura 6), la cual proporciona varias muestras de fachadas juntos con imágenes que indican la

segmentación de objetos, la clase a la que pertenece cada objeto y los bordes de los objetos detectados (Korč and Förstner, 2009).

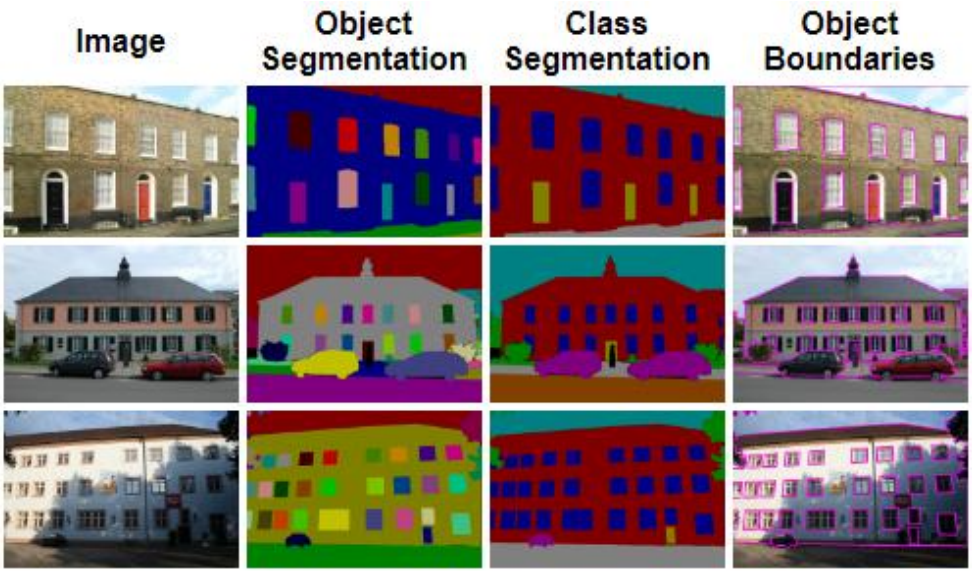

Figura 6. Ejemplo de imágenes en la base de datos eTRIMS.
Fuente: Korč and Förstner, 2009.

Generalmente para la detección de ventanas se emplean características de los bordes de los marcos, puesto que estas tienen una falta de características propias, cambiando según reflejos u objetos que se ven a través de ellas. Otro factor importante, consiste en el hecho de que en fachadas completas pueden presentarse objetos que generan respuestas en los detectores de bordes pero que no son ventanas, por lo cual es necesario conocer a cual elemento pertenece cada borde o sección detectada (Neuhausen, Koch & König, 2016).

La dificultad de lo anterior se hace evidente al emplear imágenes no rectificadas, es decir, imágenes tomadas desde un ángulo que distorsiona u oscurece algunas de las características buscadas (Neuhausen, Martin, Obel, Mark & König, 2017), o cuando se emplean imágenes aéreas oblicuas (Lin, Nex & Yang, 2018).

En el caso del proyecto, no es necesaria la detección de la ventana sino la segmentación de la parte de la ventana que debe ser limpiada (vidrio), para lo cual no ha habido un enfoque específico. Esto puede deberse principalmente a la falta de necesidad o aplicación de este resultado. Sin embargo, el enfoque para lograr esta tarea comparte

bases teóricas con otras aplicaciones mencionadas anteriormente como el reconocimiento de placas de autos (Azad & Baghdadi, 2014), la detección de elementos en fachadas y la detección de señales de tráfico (Berkaya, Gunduz, Ozsen, Akinlar & Gunal, 2016), en especial las rectangulares (Pink & Eickeler, 2016). A pesar de la similitud, esto no asegura que las mismas técnicas y algoritmos funcionen de la misma forma en una superficie de vidrio, debido principalmente a la variedad de marcos o separaciones existentes entre ventanas de rascacielos, y las condiciones de iluminación, reflejos u objetos que se puedan observar a través. Lo anterior implica la necesidad de bases de datos de imágenes personalizadas y acondicionadas al nuevo sistema que se desee desarrollar.

2.4 Bases teóricas de la visión artificial

La segmentación de imágenes, la cual se refiere al proceso de dividir una imagen en regiones siguiendo criterios específicos relacionados a las características u objetos de interés presentes en dicha imagen, es uno de los pasos más importantes de la visión artificial, debido a que la exactitud de la segmentación es la que determina la exactitud final del análisis de la imagen. Estas técnicas de segmentación de imágenes suelen dividirse en tres categorías principales: técnicas basadas en discontinuidades, umbrales, y técnicas basadas en regiones (Chacón, 2016). En las bases teóricas para el proyecto se hace énfasis en los algoritmos de bordes pertenecientes a las técnicas basadas en discontinuidades, la transformada de Hough para la detección de líneas, las redes neuronales convolucionales para una segmentación inteligente y los arboles de decisión para la integración de múltiples métodos.

2.4.1 Algoritmos de bordes

En el análisis de objetos en imágenes, es esencial poder distinguir entre el objeto de interés y el resto de la imagen. La detección de bordes se refiere a obtener imágenes cuya salida muestre pixeles de mayor intensidad en los valores que detecten bordes, y en cuanto más rápido se produce el cambio de intensidad, el borde es más fuerte. Este borde detectado se define como un súbito cambio de intensidad en pixeles al compararlos a valores de pixeles contiguos, y es una de las técnicas mas usadas debido a que los bordes contienen una considerable cantidad de información sobre la imagen, y son

capaces de indicar la ubicación, el tamaño y la forma de diferentes objetos presentes en esta (Tyagi, 2018).

La elección de un algoritmo de bordes depende completamente de la aplicación, y cada método presenta diferentes ventajas y desventajas según el objeto a segmentar. Un buen proceso de detección de bordes facilita la elaboración de fronteras de objetos y simplifica el proceso de reconocimiento de objetos (Vadivambal & Jayas, 2015).

Al mencionar los algoritmos de bordes, se refiere a los operadores de detección de bordes, los cuales mediante el cálculo de primeras y segundas derivadas, permiten determinar las secciones más importantes necesarias para realizar las mediciones. En los métodos basados en la primera derivada o gradiente, se buscan picos, mientras que en la segunda derivada se buscan pasos de respuesta positiva a negativa o viceversa, los cuales también son conocidos como cruces por cero (Bautista, Medina & Marín, 2012).

2.4.1.1 Operador Roberts

El operador de Roberts es un operador de primera derivada o gradiente, el cual emplea una máscara convolucional de 2x2 (Figura 7), y en el cual el diferencial es calculado diagonalmente, es decir, 45° y 135°. Se utiliza principalmente por su velocidad de cómputo, su facilidad de uso al implicar un entorno de vecindad de 4 pixeles, y por solo emplear sumas y restas en los cálculos. No obstante, su sensibilidad al ruido es una de sus mayores desventajas (Fahrurozi, Madenda, & Kerami, 2016).

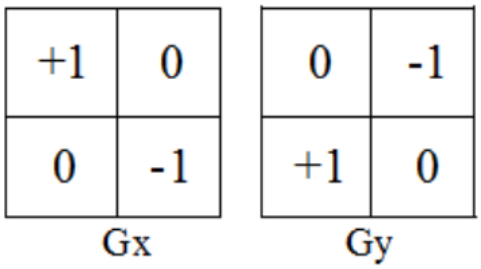

Figura 7. Máscaras del operador Roberts.
Fuente: Fahrurozi, Madenda, & Kerami, 2016.

2.4.1.2 Operador Sobel

El operador de Sobel es un operador de primera derivada o gradiente, el cual consiste en dos máscaras convolucionales de 3x3 (Figura 8). Estas están diseñadas para encontrar variaciones (bordes) verticales y horizontales, por lo cual pueden ser empleadas por separado en cada una de estas orientaciones, o combinadas para encontrar la magnitud absoluta del gradiente en cada punto y la orientación de este gradiente. Este método pone especial énfasis en los pixeles más cercanos al centro de la máscara y es uno de los operadores comúnmente empleado en la detección de bordes diagonales (Aperador-Chaparro, Bautista-Ruiz & Mejía, 2013).

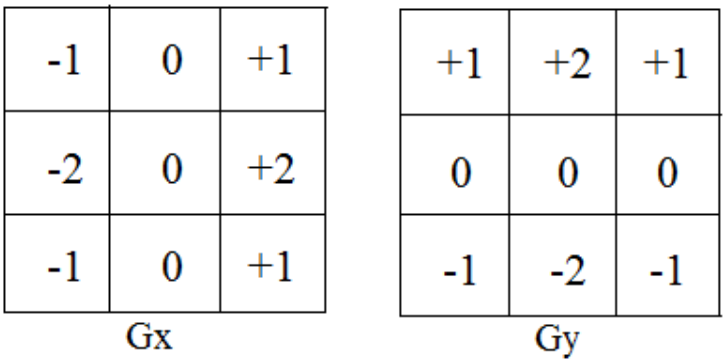

Figura 8. Máscaras del operador Sobel.
Fuente: Fahrurozi, Madenda, & Kerami, 2016.

2.4.1.3 Operador Prewitt

Al igual que con el operador de Sobel, el operador de Prewitt es un operador de primera derivada o gradiente que usa dos máscaras convolucionales de 3x3 (Figura 9), diseñadas para encontrar bordes verticales y horizontales. Estas máscaras pueden ser empleadas por separado en cada una de estas orientaciones, o combinadas para encontrar la magnitud absoluta del gradiente en cada punto y la orientación de este gradiente. A diferencia del operador Sobel, el operador Prewitt no asigna una importancia especial a pixeles cercanos al centro de la máscara, y expande la definición del gradiente en la máscara para ser más inmune al ruido y realizar una mejor detección de bordes horizontales y verticales (Zorrilla, Julián, Solano, Reyes & Calvo, 2016).

+1	0	-1
+1	0	-1
+1	0	-1

Gx

-1	-1	-1
0	0	0
+1	+1	+1

Gy

Figura 9. Máscaras del operador Prewitt.

Fuente: Fahrurozi, Madenda, & Kerami, 2016.

2.4.1.4 Algoritmo de Canny

Este algoritmo se basa en tres criterios: el primero es el de evitar eliminar bordes importantes y no suministrar bordes falsos, el segundo es el de minimizar la distancia entre la posición real y la localizada del borde, y el tercero es el de la integración de las respuestas múltiples correspondientes a un único borde. Siguiendo estos criterios se establecen tres etapas para la implementación de este algoritmo: la primera es la obtención del gradiente, donde se calcula la magnitud y orientación del vector gradiente de cada píxel, la segunda es la supresión no máxima, donde se adelgaza el ancho de los bordes hasta lograr bordes de un píxel de ancho, y por último la histéresis de umbral, donde se aplica una función de histéresis basada en dos umbrales para reducir los contornos falsos (Rebaza, 2007).

Como la primera etapa consiste en un filtrado de convolución de la primera derivada de una función gaussiana normalizada discreta sobre la imagen, esta realiza un suavizado de la imagen. Junto con la información normalizada no binaria de salida que indica la fuerza del borde a partir de un análisis del gradiente global entre pixeles vecinos, hace a este método uno de los más robustos ante la presencia de ruido (Jaramillo, Fernández & de Salazar, 2010).

2.4.1.5 Laplaciano de Gaussiano (LoG)

El Laplaciano de Gaussiano, como su nombre lo indica, combina el filtrado Gaussiano con el Laplaciano, y consiste en la realización de un filtrado, un realce y una detección. El

filtrado se emplea para suavizar la imagen, el realce es donde se aplica la segunda derivada, y el criterio de detección es la presencia de un cruce por cero en la segunda derivada con el correspondiente pico en la primera derivada (Amador González, 2004).

En este operador el coeficiente asociado al píxel del centro es positivo y los coeficientes asociados a los píxeles exteriores son negativos, haciendo que la suma de todos los coeficientes sea cero. Al hacer la convolución de una de las máscaras (Figura 10) con la imagen, el resultado es cero cuando el punto central tiene el mismo valor que los pixeles adyacentes (Duque & Ospina, 2004).

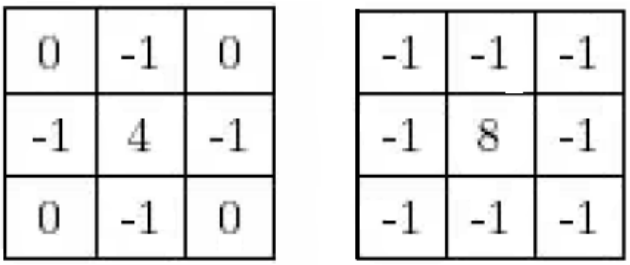

Figura 10. Operadores laplacianos.
Fuente: Duque & Ospina, 2004.

2.4.2 Transformada de Hough

Esta transformada se emplea con frecuencia para detectar líneas mediante la transformación de los puntos del plano cartesiano a un espacio de parámetros, y usualmente se clasifica en dos grupos dependiendo de la parametrización usada para representar las líneas (Guil, Villalba & Zapata, 1995).

El primer grupo emplea los parámetros de la pendiente a y el parámetro de intersección con la ordenada b, basándose en la representación de la recta dada por $y = ax + b$, la cual se satisface por infinitas rectas que pasan por cada punto del plano con diferentes valores de a y b. Al considerar el plano ab, denominado espacio de parámetros, se obtiene una única recta para cada punto (Gonzalez & Woods, 2008). Como se observa en la Figura 11, si se consideran dos puntos (x_i, y_i) y (x_j, y_j), cada punto tendrá asociada una única recta, la cual al intersectarse permite obtener los parámetros a y b de la recta que contiene los puntos colineales de los dos puntos considerados.

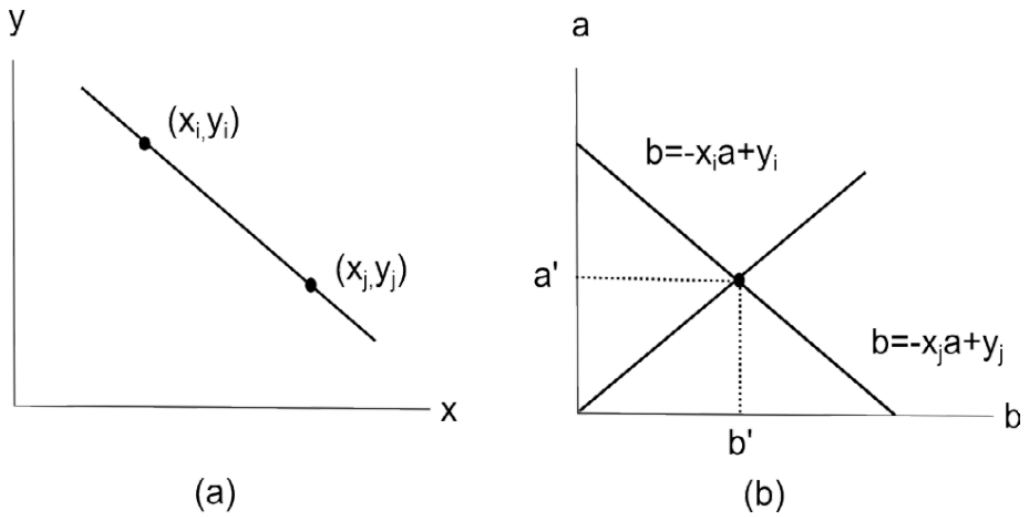

Figura 11. Transformada de Hough. (a) Recta en el plano xy; (b) espacio de parámetros.
Fuente: Diaz & Arceo, 2018.

Al implementar la transformada de Hough, se emplean celdas acumuladoras que dividen el espacio de parámetros, y que resultan en una malla que durante la fase de votación permite evaluar los posibles valores por cada punto y acumular los frecuentes. En la fase de localización de picos en la malla, valores altos en cada casilla, indican que se ha encontrado una línea recta. La mayor desventaja de emplear el espacio de parámetros es la dificultad para detectar líneas verticales, debido a que los parámetros pueden llegar a ser infinitos a medida que la línea se hace vertical (Gonzalez & Woods, 2008).

El segundo grupo utiliza coordenadas polares siguiendo la ecuación $\rho = xCos\theta + ySin\theta$, y está compuesto por el parámetro ρ, el cual es la distancia de la línea con el origen, y θ que es el ángulo entre el vector con respecto al eje de las abscisas. Se comporta de forma similar al primer grupo, diferenciándose que en lugar de rectas, por cada punto estará asociada una curva sinusoidal en el plano $\rho\theta$ (Figura 12). Utilizando las coordenadas polares es posible dar solución al inconveniente de las líneas verticales mencionado en el primer grupo (Diaz & Arceo, 2018).

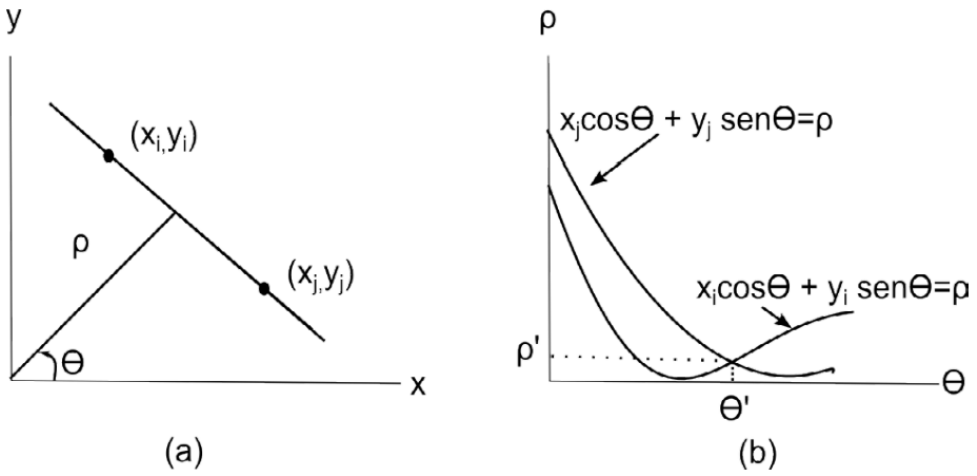

Figura 12. Transformada de Hough. (a) Parametrización de las líneas en el plano xy; (b) curvas sinusoidales en el plano ρθ.

Fuente: Diaz & Arceo, 2018.

Con el fin de visualizar mejor el concepto de la transformada de Hough utilizando coordenadas polares, un ejemplo simple de detección de líneas se puede observar en la Figura 13.

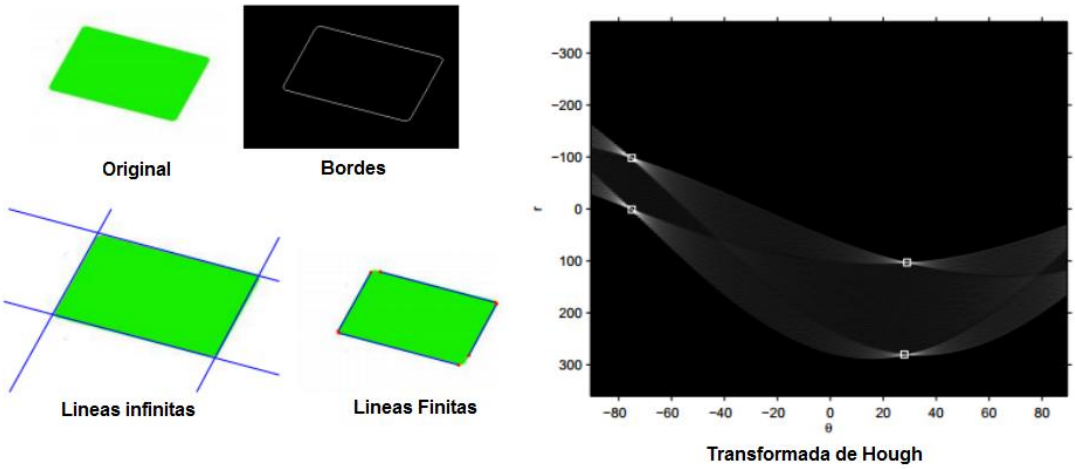

Figura 13. Ejemplo de la utilización de la transformada de Hough.

Fuente: Moreira & Sappa, 2015.

A pesar de enfocarse en las líneas rectas, la transformada de Hough no se encuentra restringida a la detección de estas, y puede ser empleada para detectar curvas. Utilizando la ecuación $(x-a)^2+(y-b)^2=r^2$ y un arreglo acumulador de tres dimensiones, es posible encontrar círculos en las imágenes suministradas a la transformada de Hough (Moreira & Sappa, 2015).

2.4.3 Redes neuronales convolucionales

Las redes neuronales convolucionales (Conv-Net o CNNs) son una clase de redes neuronales multicapa diseñadas para el tratamiento, clasificación y reconocimiento de imágenes. Son similares a las redes neuronales comunes, en vista de que cuentan con pesos, función de activación, función de perdida, entre otras similitudes. Sin embargo, aunque es posible tratar imágenes con redes neuronales estándar, en cuanto el tamaño y calidad de dicha imagen aumenta, se genera un rápido sobreajuste y un desperdicio de recursos debido a las neuronas completamente conectadas. En el caso de las redes neuronales convolucionales, estas dividen y modelan la información en partes más pequeñas, combinando esta información en las capas más profundas de la red (Antona Cortés, 2017).

2.4.3.1 Arquitectura de una red neuronal convolucional

La estructura de la redes neuronales convolucionales está compuesta por varios tipos de capas, las más importantes siendo las capas de convolución, las ReLU, las de submuestreo o pooling, y las clasificadoras completamente conectadas (Figura 14).

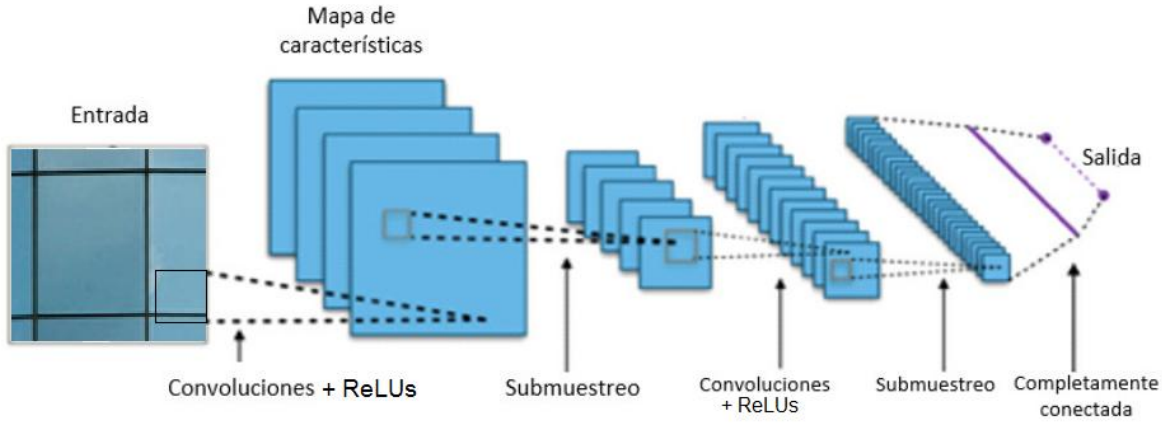

Figura 14. Arquitectura de una red neuronal convolucional.

Fuente: Fausett, 1994. Editado por el autor.

La operación de convolución recibe como entrada una imagen sobre la cual se aplica un filtro o kernel que resulta en un mapa de características (Figura 15). De esta forma, se

logra reducir el tamaño de los parámetros, mejorar la eficiencia del sistema, optimizar los cálculos, y permitir al sistema trabajar con entradas de tamaño variable (Isasi Viñuela & Leon, 2004).

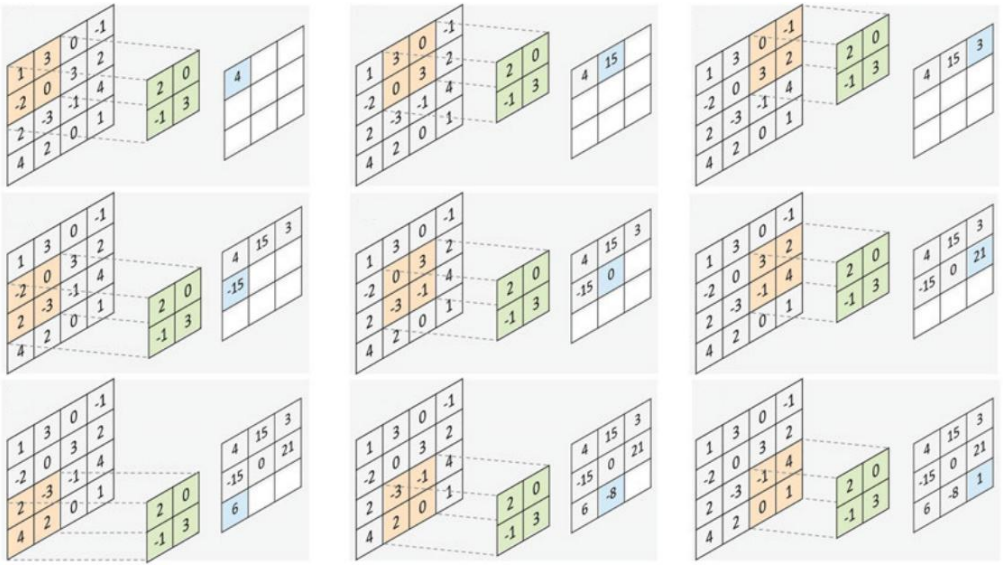

Figura 15. Ejemplo de aplicación de una convolución.
Fuente: Khan, Rahmani, Shah & Bennamoun, 2018.

Las capas ReLU o rectified linear unit, se utilizan en la mayoría de las redes neuronales actuales, debido principalmente a que las redes neuronales que cuentan con varias capas se vuelven difíciles de entrenar cuando se da el problema de desaparición de gradiente. Esta función de activación que se expresa como R(z)=max(0,z), permite el paso de todos los valores positivos sin modificarlos, pero asigna todos los valores negativos a 0. Por razones de practicidad, esta función rectificadora es una de las más utilizadas en aprendizaje profundo, superando otras funciones como la logística sigmoide o el de la tangente (Dahl, Sainath & Hinton, 2013).

La capa de submuestreo o pooling, es colocada generalmente luego de la capa convolucional o la ReLU, y se encarga de reducir la cantidad de parámetros a analizar mediante la reducción de las dimensiones del mapa de características (Figura 16). Lo anterior implica una pérdida de información, pero permite reducir el sobreajuste y facilitar el cálculo de las próximas capas de la red. En esta capa se suele aplicar la función denominada “max-pooling”, la cual divide la matriz en un conjunto y guarda el valor máximo de este (Scherer, Müller & Behnke, 2010).

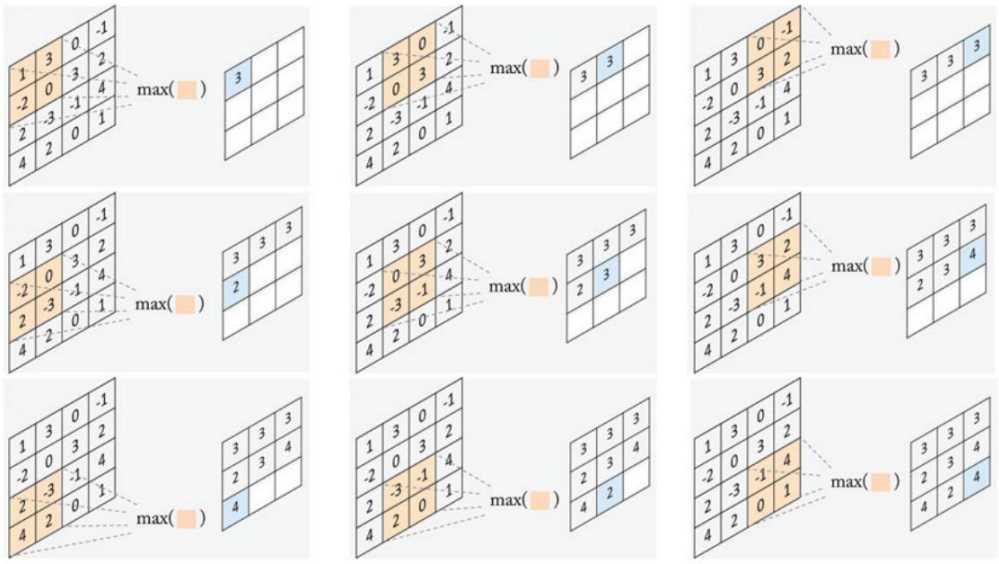

Figura 16. Ejemplo de aplicación de un max pooling.
Fuente: Khan, Rahmani, Shah & Bennamoun, 2018.

Por último, la capa clasificador total, contiene una o más capas completamente conectadas que se encargan de tomar las características y producir probabilidades de clases o puntajes. Estas neuronas funcionan de forma parecida a las perceptrón multicapa, pues cada salida se calcula multiplicando la salida de la capa anterior por el peso de la conexión, y aplicándole a este una función de activación (Zeiler & Fergus, 2014).

Es importante señalar que las redes neuronal convolucionales han evolucionado con los años debido a la necesidad de herramientas especializadas para la extracción de características en imágenes, y las arquitecturas más eficaces y populares han sido reconocidas por la ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Entre estas arquitecturas se encuentra la LeNet, la AlexNet, la ZF Net, la GoogLeNet, la VGGNet y la ResNet (Patterson & Gibson, 2017).

2.4.3.2 Segmentación semántica

La función de las redes neuronales convolucionales se puede dividir en clasificación, localización, detección y segmentación (Figura 17), las cuales se seleccionan y se emplean dependiendo de la aplicación y la necesidad de la tarea a efectuar. En el caso de

la segmentación, estas redes pueden ser adaptadas para aplicaciones que requieren predicciones por pixel, es decir, cuando es necesario encontrar datos precisos o posiciones exactas de objetos (Khan, Rahmani, Shah & Bennamoun, 2018).

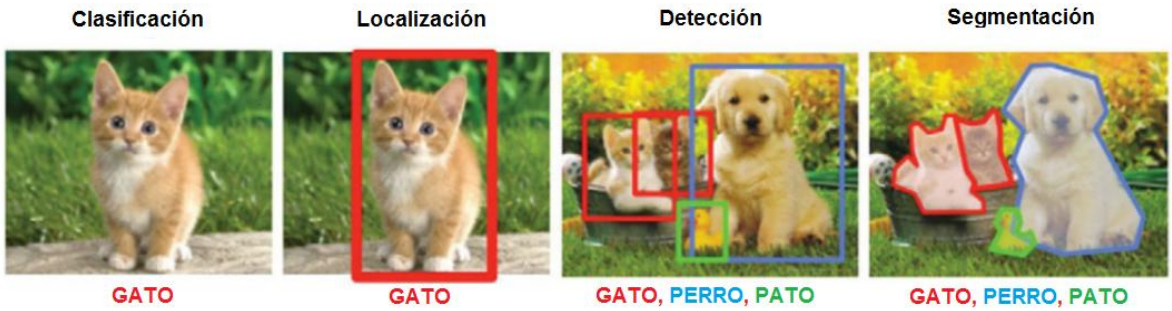

Figura 17. Diferencia entre clasificación, localización, detección y segmentación.

Fuente: Khan, Rahmani, Shah & Bennamoun, 2018.

En el caso de la segmentación semántica, esta consiste en realizar un etiquetado semántico de cada pixel perteneciente a una imagen, y luego emplear la clasificación de los pixeles para entrenar una red neuronal convolucional. Esta red puede ser una red totalmente convolucional, como la vista anteriormente, o una red deconvolucional profunda (Figura 18), la cual además de la parte convolucional contiene un módulo deconvolucional que es el reciproco de la red convolucional (Patterson & Gibson, 2017).

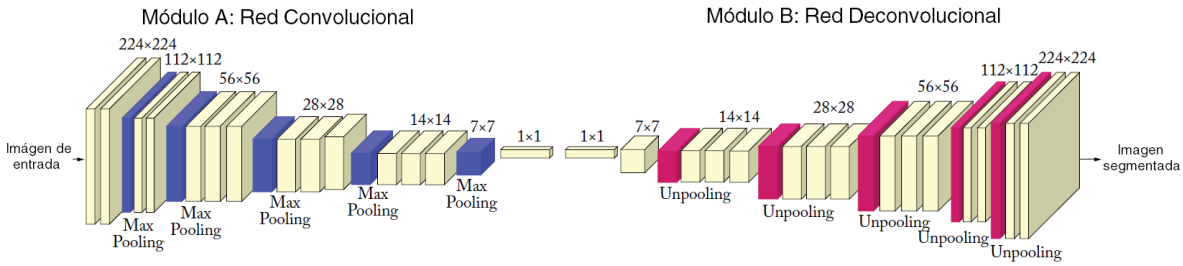

Figura 18. Arquitectura de una red deconvolucional profunda.

Fuente: Patterson & Gibson, 2017.

El módulo de convolución actúa como un extractor de características representando la imagen de entrada como un mapa de características; y el módulo de deconvolución utiliza estos mapas de características extraídos para generar mapas de predicción de clase con el mismo tamaño espacial que la imagen de entrada, y que representan la probabilidad de que cada píxel pertenezca a diferentes clases (Noh, Hong & Han, 2015).

La red deconvolucional profunda utiliza una red VGG-16 para la parte convolucional, en la cual la última capa completamente conectada es removida y las dos últimas capas completamente conectadas son convertidas a capas de convolución. Debido al uso de múltiples capas de convolución, la red requiere de una gran cantidad de datos para el entrenamiento, y de memoria y tiempo adicional de entrenamiento (Patterson & Gibson, 2017).

Sin embargo, una ventaja de las redes neuronales convolucionales y las redes neuronales profundas en general que se ha estudiado en los años recientes, es el aprendizaje transferido (Noroozi, Vinjimoor, Favaro & Pirsiavash, 2018). Este se define como la mejora del aprendizaje en una nueva tarea a través de la transferencia de conocimiento de una tarea relacionada que ya se ha aprendido (Donahue, Jia, Vinyals, Hoffman, Zhang, Tzeng & Darrell, 2014).

Por ejemplo, al entrenar imágenes, la primera capa de estas redes tiende a aprender características que se asemejan a los filtros de Gabor o las manchas de colores. Este fenómeno ocurre no solo en diferentes conjuntos de datos, sino incluso en objetivos de entrenamiento completamente diferentes, incluso en la clasificación de imágenes supervisadas. Estas características se suelen llamar características generales, y pueden ocurrir independientemente de la función y la base de datos de imágenes; por otro lado las características calculadas por la última capa de una red dependen en gran medida del conjunto de datos elegido y la tarea a efectuar (Yosinski, Clune, Bengio & Lipson, 2014).

CAPITULO 3: METODOLOGÍA Y RESULTADOS

El presente capítulo abarca tanto la metodología como los resultados obtenidos en cada fase del desarrollo del proyecto. Se decidió organizar de esta forma con el fin de explicar el razonamiento de las decisiones tomadas durante el proyecto, así como exponer como cada fase y sus respectivos resultados, influyen en la selección de los métodos empleados en las fases siguientes. Como se muestra en la Figura 19, el desarrollo se compone de cinco fases principales: la creación de la base de datos, el desarrollo del sistema de segmentación preliminar, el desarrollo del sistema empleando redes neuronales convolucionales, la integración de los métodos anteriores en un árbol de decisión para crear un sistema multiagente y, por último, la fotogrametría aplicada al sistema para demostrar la posibilidad de una implementación real del sistema de visión artificial multiagente para la segmentación de vidrio en ventanas de rascacielos.

Figura 19. Diagrama de metodología y resultados.

Fuente: Elaboración propia, 2019.

3.1 Compilación de la base de datos

El desarrollo del sistema inició con la búsqueda de distintas fotos de ventanas de rascacielos que más se asemejaran a las que tomaría una cámara montada sobre un dron, es decir, directamente en frente de la ventana con el vidrio a segmentar. En este caso, fotos tomadas por drones volando cerca de ventanas de rascacielos o fotos de las fachadas tomadas desde edificios opuestos, fueron la prioridad de la búsqueda.

Se enfocó en evitar fotos tomadas desde el suelo, desde un ángulo o con distorsiones; sin embargo, algunas de estas fueron agregadas a la base de datos para hacer al sistema más robusto y analizar su impacto en el sistema. Un ejemplo de distorsión presentada, es

la distorsión de ángulo ancho u ojo de pez, la cual a pesar de utilizarse comúnmente en fotografía por su ángulo de visión amplio, no se empleó en gran medida en el sistema con el fin de evitar la necesidad de corregir los efectos mencionados y lograr una segmentación más precisa del vidrio.

El alcance del programa de segmentación abarca solamente la segmentación de ventanas de rascacielos rectangulares, puesto que estas conforman la mayoría de los tipos de ventanas que se pueden encontrar en los rascacielos actuales. No obstante, durante el desarrollo se evaluó la posibilidad y los requerimientos para implementar el sistema de segmentación en ventanas con distintas formas e inclinaciones, las cuales han comenzado a ser incluidas en el diseño y construcción de rascacielos modernos.

A partir de los requerimientos mencionados, se realizó una búsqueda exhaustiva de fotos de rascacielos en páginas web de imágenes y videos de stock, completamente gratuitos o que contaban con una vista previa gratuita. GettyImages, ShutterStock, iStock, Pond5, dreamstime, son algunos ejemplos de las páginas en las que se realizó la búsqueda, y algunas de las fotos recopiladas se pueden apreciar en la Figura 20.

Figura 20. Ejemplos de fotos de fachadas de rascacielos recopiladas.

Fuente: Elaboración propia, a partir de diversas páginas mencionadas en el texto, 2019.

Las fotos originales son de fachadas de rascacielos que abarcan varias ventanas en una sola imagen, sin embargo durante la futura implementación y funcionamiento del sistema, el dron estará lo más cerca posible de la ventana a limpiar, tanto para mejorar la exactitud de la segmentación como para el funcionamiento de los sensores de distancia. Por esta razón, se recortaron ventanas individuales o un pequeño conjunto de ventanas por imagen, procurando mantener la cantidad incluida de ventanas distorsionadas o inclinadas al mínimo, y priorizando la selección de ventanas cercanas al centro de la foto (Figura 21).

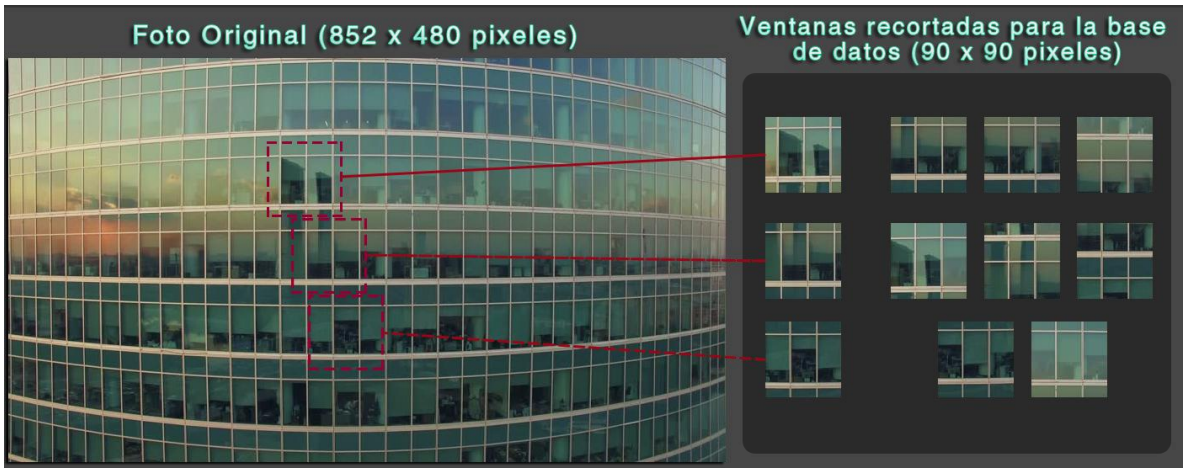

Figura 21. Ejemplo de la construcción de la base de datos de ventanas de rascacielos.

Fuente: Elaboración propia, 2019.

En general, la prioridad fue conseguir la mayor variedad factible de ventanas que se puedan encontrar en los rascacielos en cuanto a condiciones del vidrio y tipologías, a partir de lo cual se pudo apreciar que estas pueden ser divididas en dos categorías principales. La primera categoría abarca el vidrio que permite ver dentro del edificio y los objetos que se ven a través, tales como mesas, sillas, columnas, persianas, escritorios, entre otros. Y la segunda categoría abarca el vidrio completamente reflectante en el que se proyectan elementos de otros edificios, grúas de construcción, cielo despejado o nublado, el horizonte y el sol en los marcos o el vidrio, entre otros elementos. Dentro de estas dos categorías (Figura 22), también se encontró un extenso número de variaciones, como por ejemplo iluminación dependiente del tiempo (soleado, ocaso, nublado, noche); marcos de distintos tamaños, grosores y tonalidades; variación entre marcos en una sola ventana en cuanto a diferencias entre la parte horizontal y vertical; y distorsiones en los reflejos.

Figura 22. Ejemplos de las dos categorías principales de la base de datos de ventanas de rascacielos y sus variaciones.

Fuente: Elaboración propia, 2019.

Cabe destacar que el grado de dificultad alto en la segmentación de las imágenes fue considerado como un aspecto positivo, puesto que permitió garantizar la consolidación de un sistema más robusto. En algunos casos se seleccionaron casos de ventanas a las que el dron nunca se vería expuesto en su operación, con el único fin de evaluar y mejorar las capacidades del sistema desarrollado.

Siguiendo los parámetros establecidos, se seleccionaron y se recortaron imágenes de ventanas de rascacielos (Figura 23), las cuales se emplearon como la base de datos para todas las pruebas del sistema de visión artificial, así como para el entrenamiento de la red neuronal convolucional.

Figura 23. Base de datos compuesta por 400 imágenes de ventanas de rascacielos.
Fuente: Elaboración propia, 2019.

De esta manera, la base de datos resultante está compuesta por 400 imágenes en formato .jpg, y un tamaño de 90x90 pixeles en cada imagen. Esta resolución fue escogida para facilitar la implementación y entrenamiento de maquinas de soporte vectorial y redes neuronales convolucionales, permitir una segmentación más rápida del vidrio durante la investigación y demostrar que el sistema funciona en imágenes de baja resolución. Al respecto, el hecho de que las fotos originales fueron de vista previa con una resolución baja también influyó en la resolución de la base de datos.

En cuanto a la nomenclatura definida para las imágenes, esta contiene la letra “w” junto con el número correspondiente al orden en que fueron adicionadas a la base de datos. De tal forma, facilita la aplicación de los programas desarrollados a todas las imágenes pertenecientes a la base de datos y, al mismo tiempo, permite identificar cuáles son los números específicos de las imágenes que lograron una segmentación exitosa o fallida. Esto último posibilitó la optimización de los umbrales de cada método empleado durante la investigación, así como la creación de las ramas del árbol de decisión.

3.2 Desarrollo del sistema de segmentación preliminar

Es importante recalcar que la función del sistema de visión artificial desarrollado es la de indicar el tamaño del vidrio y la posición de su centro con respecto al centro del dron o de la cámara. Estos parámetros fueron pensados para ser utilizados a futuro en un brazo robótico ensamblado en el dron, y serán usados para posicionarlo correctamente en la ventana a limpiar y reconocer la superficie que debe ser limpiada. Con esto en mente, se procedió a analizar las diferentes formas y métodos que posibilitan la obtención de los parámetros mencionados.

Como se mencionó durante la creación de la base de datos, los vidrios de las ventanas de rascacielos pueden ser reflectantes o permitir ver a través de ellos, lo cual significa que el vidrio tiene una falta de características propias, y lo que se observa son atributos de otros objetos. Esta es la razón principal por la cual los marcos fueron escogidos como el enfoque inicial del sistema para lograr la segmentación del vidrió.

En esta sección del desarrollo del proyecto se realizó un programa preliminar de segmentación de vidrio en ventanas de rascacielos, el cual se hizo, no solo para demostrar que es posible obtener los resultados esperados en la investigación, sino para evaluar el potencial de los distintos métodos y algoritmos en esta tarea. Esta evaluación fue imprescindible para posteriormente proceder a realizar métodos de inteligencia artificial y asegurar el mejoramiento del sistema empleando el árbol de decisión.

El programa preliminar para lograr la segmentación del vidrio consiste en encontrar los bordes de los marcos mediante algoritmos de detección de bordes con operadores Canny, Sobel, Prewitt, Roberts y Laplaciano de Gaussiano; dilatación de los bordes detectados, creación de áreas a partir de los bordes detectados; adición de filtros para eliminar áreas que estén en contacto con los márgenes de la imagen (otras ventanas incompletas) y, por último, propiedades de región para detectar formas rectangulares usando elementos de estructurado morfológico. Al mismo tiempo, se creó un segundo programa de segmentación, en el cual se usan los mismos métodos anteriores, pero incorporando la transformada de Hough junto con filtros adicionales propios de este método.

Estos programas realizados en Matlab 2018a, incluyen la creación de una interfaz gráfica de usuario elaborada con el GUIDE de Matlab (Figura 24), en la cual la mitad superior corresponde al programa preliminar con transformada de Hough, y la mitad inferior al programa donde solamente se emplearon los algoritmos de bordes y las propiedades de región.

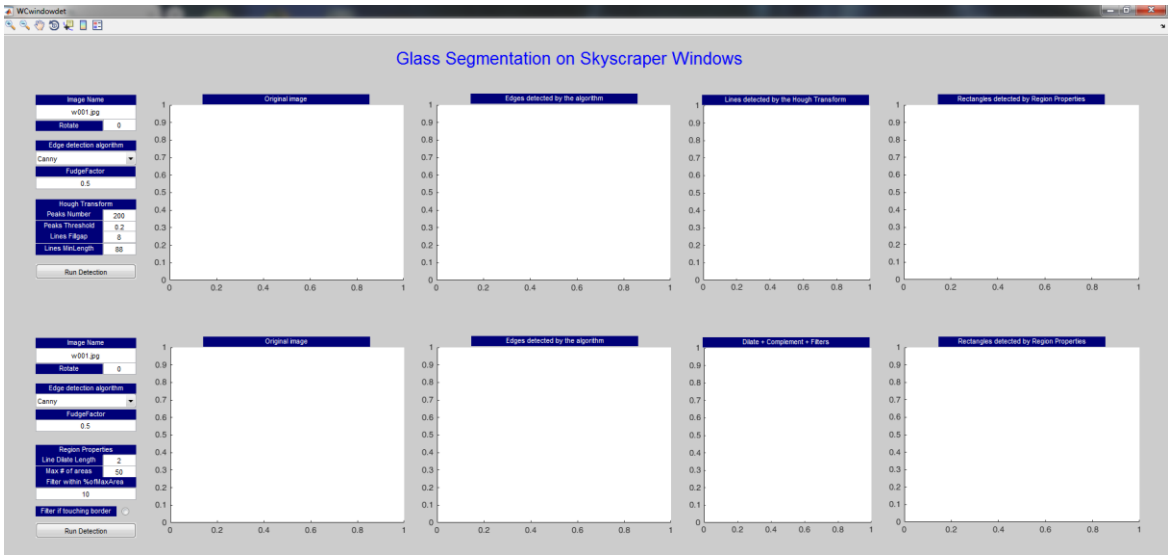

Figura 24. Interfaz gráfica del programa preliminar para la prueba de distintos algoritmos.
Fuente: Elaboración propia, 2019.

Las opciones del programa preliminar se realizaron de tal forma que permitiesen seleccionar el algoritmo de bordes, cambiar umbrales, y modificar filtros para las propiedades de región. Cada una de estas opciones se puede apreciar en más detalle en la Figura 25.

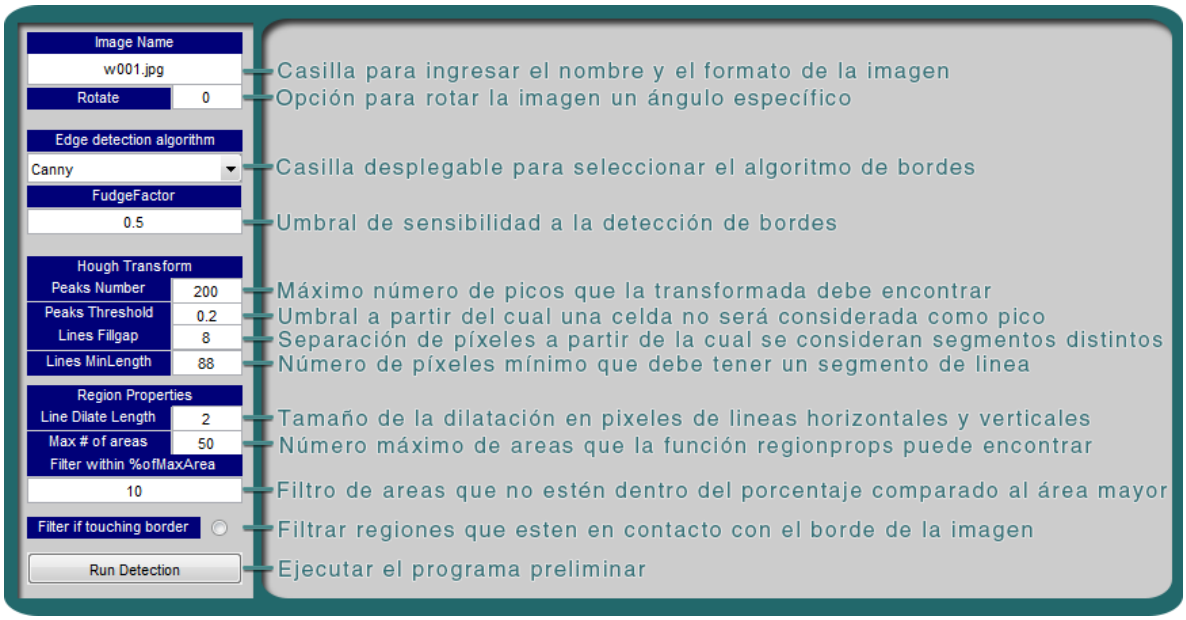

Figura 25. Opciones de la interfaz gráfica del programa preliminar.

Fuente: Elaboración propia, 2019.

3.2.1 Primer programa de segmentación preliminar empleando algoritmos de bordes, filtros y propiedades de región

Para el funcionamiento del primer programa preliminar, inicialmente se ingresa el nombre de la imagen en la cual se desea segmentar el vidrio, se transforma la imagen a escala de grises, se le aplica la función de bordes de Matlab con el operador o algoritmo escogido para encontrar el umbral específico de la imagen, y luego se aplica nuevamente la misma función de bordes escogida, pero esta vez con un umbral que es el producto del umbral anterior por otro umbral llamado fudgefactor. Este umbral corresponde a un valor entre 0 y 1 que influye en la sensibilidad del algoritmo a la detección de bordes; de la siguiente manera: cuando el valor es cercano a 0, este detecta más bordes suaves, es decir que encuentra variaciones menores entre pixeles adyacentes, mientras que para valores cercanos a 1, detecta bordes definidos o variaciones altas de valores entre pixeles.

A la imagen resultante, correspondiente a una imagen en blanco y negro con los bordes que el algoritmo seleccionado detectó, se le aplica una dilatación de líneas de 0 y 90 grados, es decir, se incrementa el grosor de las líneas horizontales y verticales que se encuentren en la imagen de bordes. A esta imagen dilatada se le invierten los colores blanco y negro para acomodarla a las necesidades de las propiedades de región, y se le

aplica un filtro para eliminar cualquier región que esté en contacto con los bordes de la imagen. Esto último se empleó para que no se tengan en cuenta los vidrios incompletos que se puedan presentar en la imagen.

En el siguiente paso se aplican filtros adicionales de áreas; el primero para reducir la cantidad de regiones encontradas a un valor deseado, el segundo para eliminar regiones que tengan un tamaño de área por fuera de un rango específico con respecto al área más grande y, el tercero, para encontrar la región que se encuentre más cerca al centro de la imagen mediante un filtro adicional.

Este último filtro se realizó teniendo en cuenta que el operario posicionará el dron frente a la ventana a limpiar; sin embargo, esto no asegura que en la imagen solo aparezca una sola ventana con marcos completos, ya que pueden presentarse casos de varias ventanas pequeñas o secciones de ventanas en el cuadro de la foto. De esta manera, enfocarse solo en la región más cercana al centro, permitirá al operario seleccionar de manera más sencilla el vidrio a limpiar y facilitará los requerimientos de posicionamiento.

Es necesario destacar que la región segmentada final, la cual es considerada como la del vidrio a limpiar, puede resultar irregular debido a errores durante la detección de bordes u otros filtros empleados en el programa. En este sentido, como el sistema está enfocado a ventanas rectangulares, se empleó la propiedad BoundingBox de la función de propiedades de región de Matlab, la cual permite obtener el rectángulo más pequeño que contiene la región segmentada, devuelto como un vector con la posición de la esquina superior izquierda, el ancho y el alto del rectángulo delimitador.

Con la función anterior se obtiene directamente el ancho y el alto de la región resultante, y con la posición de la esquina superior izquierda se calcula el centroide de la región. Al mismo tiempo, a pesar de que estos son los parámetros buscados por el sistema, su representación está dada en pixeles y requiere ser transformada a medidas reales que permitan implementar el sistema de visión artificial. En el caso del programa preliminar y para propósitos de validación y comparación se emplearon los resultados en pixeles.

Adicionalmente, para una visualización mejor de los resultados en la interfaz, se posicionó un rectángulo de color rojo en el centroide de la región segmentada, y se incluyó de forma textual los parámetros conseguidos al realizar la segmentación.

3.2.2 Resultados del primer programa de segmentación preliminar

Empleando el primer programa preliminar, se compararon los algoritmos de bordes mencionados anteriormente, ejecutando los cinco algoritmos de detección de bordes a cada una de las imágenes de la base de datos. Uno de los casos en que todos los algoritmos resultaron en una segmentación correcta se puede ver en la Figura 26, donde también se aprecian las diferencias de detección entre algoritmos de bordes.

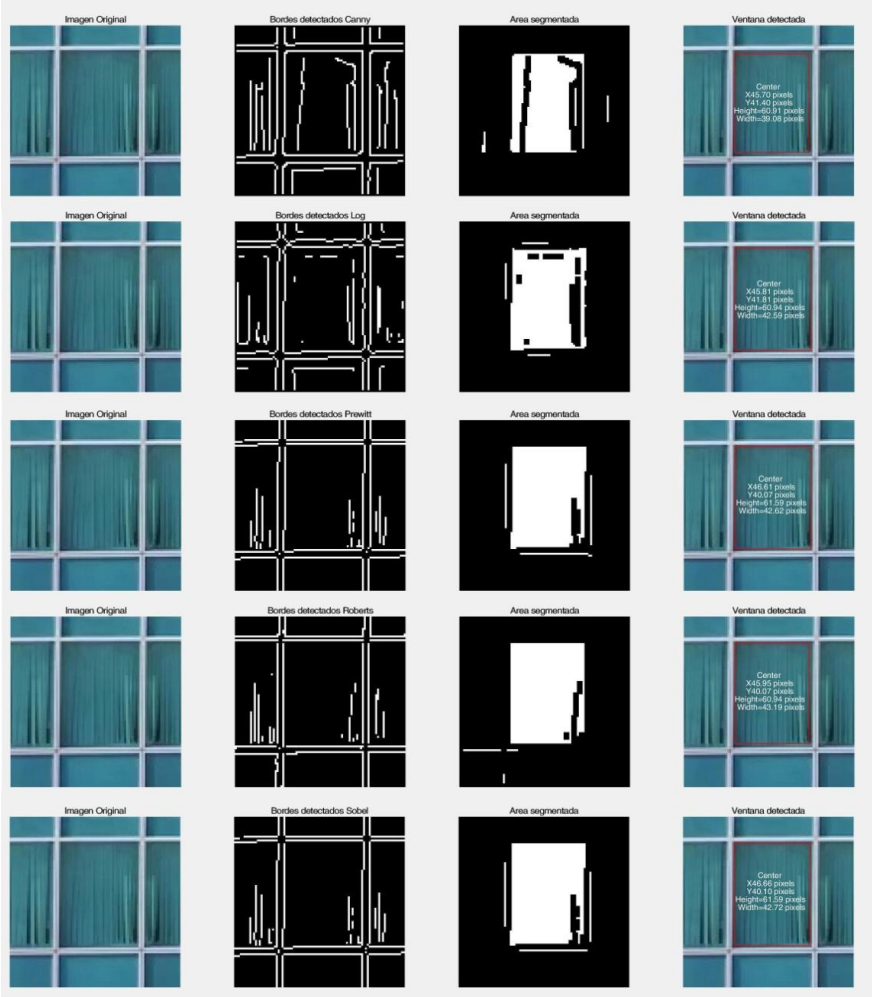

Figura 26. Ejemplo de la segmentación de vidrio empleando distintos algoritmos de bordes.
Fuente: Elaboración propia, 2019.

Para lograr lo anterior, se implementó un programa adicional que permitió aplicar el código de segmentado preliminar a cada una de las 400 imágenes de la base de datos, manteniendo un valor fijo en los umbrales de detección. Cabe mencionar, que al ser cuatro los datos entregados por el programa, el tamaño de la matriz de resultados fue de 400 filas y 4 columnas. Estos datos contienen el ancho del vidrio en pixeles, el alto del vidrio en pixeles y la posición del centro del vidrio en los ejes X y Y tomada a partir de la esquina superior izquierda.

A pesar de que inicialmente se realizó una evaluación preliminar aproximada, durante la realización de la red neuronal convolucional se creó una matriz de validación que contiene los cuatro valores verdaderos marcados y segmentados por el investigador, para cada una de las 400 imágenes que componen la base de datos.

Esta matriz de validación se comparó con los datos entregados por el primer programa preliminar, obteniendo así el error en pixeles para cada uno de los cuatro parámetros del vidrio. A continuación, empleando el máximo error en pixeles de los cuatro valores, se dividieron los resultados en dos categorías; segmentación fallida si el error máximo es mayor a 5 pixeles, y segmentación correcta si el error máximo es menor o igual a 5 pixeles. Al mismo tiempo, se elaboró un contador para calcular las segmentaciones correctas con un error de 1, 2, 3, 4 y 5 pixeles. Los resultados obtenidos al emplear el primer programa preliminar en las 400 imágenes de la base de datos se pueden ver en la Figura 27.

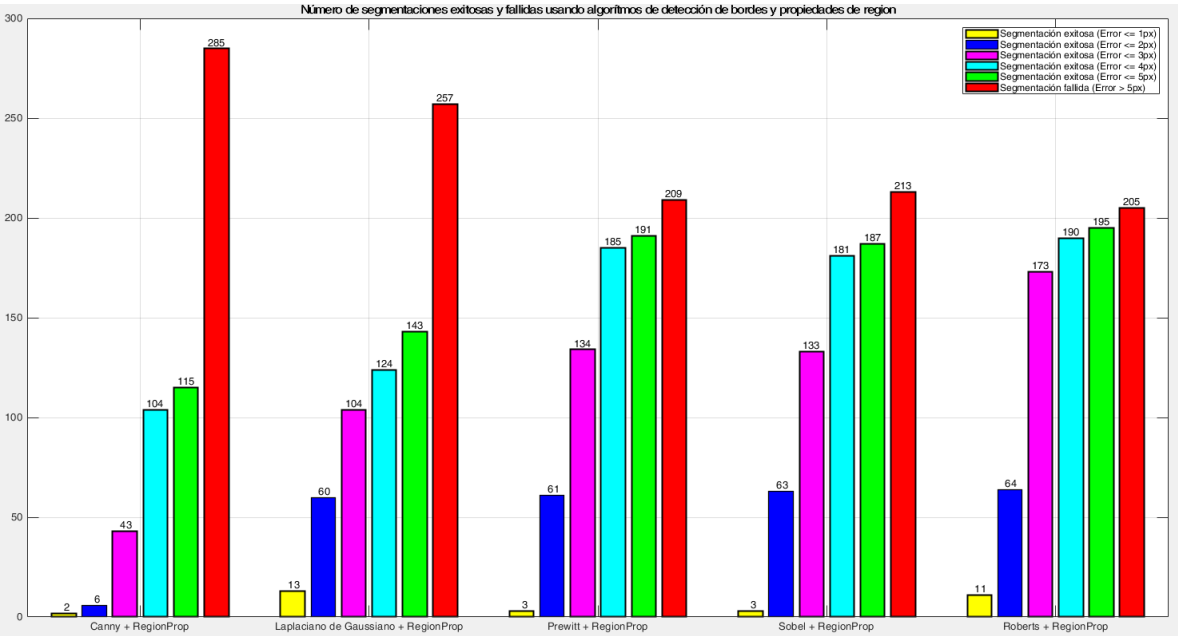

Figura 27. Resultados del primer programa preliminar para cada uno de los algoritmos de bordes.
Fuente: Elaboración propia, 2019.

Al comparar los resultados obtenidos, el operador Roberts mostró los mejores resultados en cuanto a la exactitud de los bordes encontrados y la cantidad de segmentaciones exitosas, logrando segmentar 195 de las 400 ventanas de la base de datos con un error menor o igual a 5 pixeles, y 173 con un error menor a 3 pixeles. Otros operadores que sobresalieron fueron el de Prewitt y el de Sobel, mientras que los de Canny y Laplaciano de Gaussiano presentaron los peores resultados con 115 de 400 y 143 de 400 segmentaciones correctas, respectivamente.

Fue interesante observar que, en el caso de este sistema, el algoritmo de Canny presenta los peores resultados, a pesar de ser uno de los algoritmos más utilizados en la visión artificial por su sensibilidad a la detección de bordes y tolerancia al ruido. Sin embargo, en este caso, sus ventajas hacen que se detecten con mayor exactitud los reflejos y objetos que se ven a través del vidrio, dificultando así la segmentación.

Para visualizar mejor las segmentaciones exitosas de los algoritmos de detección de bordes, en la Figura 28 se exponen algunos resultados obtenidos con los operadores Roberts, Sobel y Prewitt.

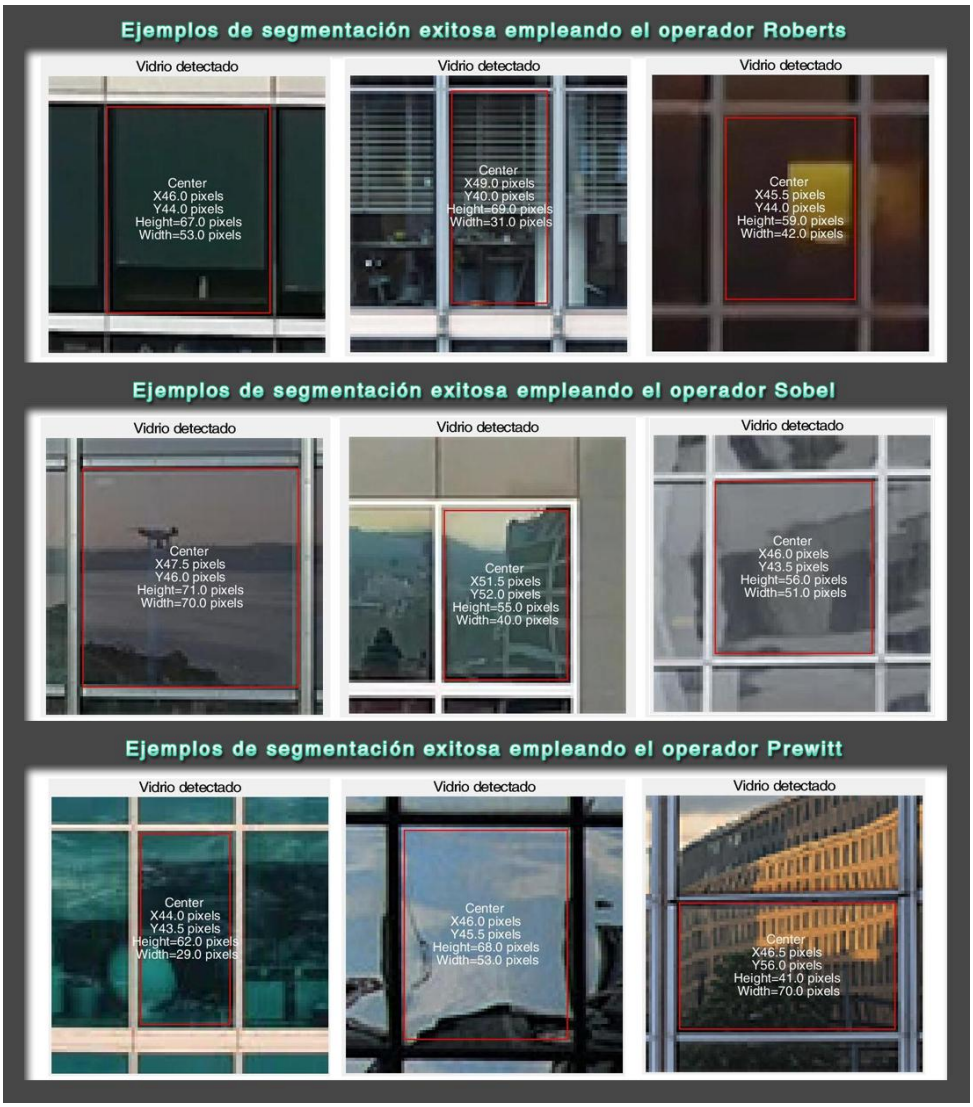

Figura 28. Ejemplos de segmentación exitosa con los algoritmos de detección de bordes que sobresalieron en la segmentación de vidrio.

Fuente: Elaboración propia, 2019.

A partir de estos resultados se seleccionaron los algoritmos de detección de bordes de Sobel y de Roberts como los principales a emplear en pruebas de optimización y en el árbol de decisión.

A pesar de la segmentación exitosa en varias imágenes de la base de datos, se evidenció que, empleando solamente los algoritmos de bordes, la segmentación tiende a fallar en las ventanas con marcos poco definidos, cuando se ven objetos reflejados en el vidrio o cuando se presentan demasiados objetos dentro del edificio que son vistos a través del vidrio.

Por otro lado, al disminuir los umbrales de los algoritmos de bordes se observó que se aumenta la sensibilidad de la detección, y se solucionan los problemas asociados a bordes poco definidos, pero al mismo tiempo se afecta la segmentación de regiones con los nuevos bordes detectados. Por consiguiente, la cantidad de bordes detectados se convierten en varias áreas pequeñas segmentadas en lugar del vidrio deseado, lo que hace fallar la segmentación sin importar que operador o algoritmo se empleara (Figura 29).

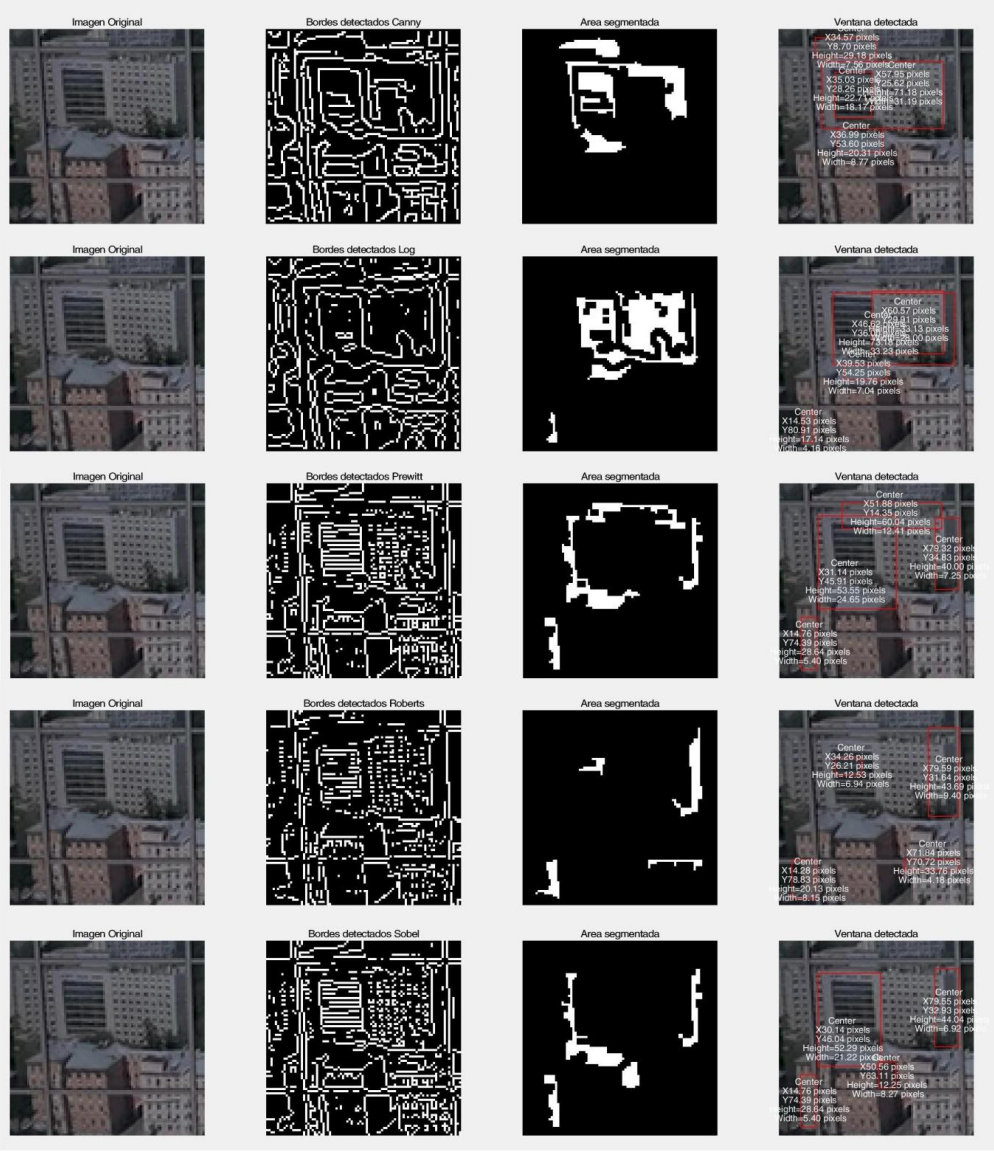

Figura 29. Problemas de segmentación en las ventanas con reflejos de edificios.

Fuente: Elaboración propia, 2019.

Para solucionar el problema expuesto, se optó por emplear la transformada de Hough, la cual detecta líneas entre los bordes encontrados y, mediante el uso de sus parámetros, permite obviar discontinuidades entre líneas, filtrar por la longitud y el ángulo de las líneas detectadas.

3.2.3 Segundo programa de segmentación preliminar incorporando transformada de Hough

Este segundo programa preliminar está estructurado de la misma forma que el de algoritmos de bordes y propiedades de región, es decir, inicialmente se ingresa el nombre de la imagen, se transforma a escala de grises y luego se aplica dos veces la función de bordes de Matlab con el algoritmo escogido para ajustar la sensibilidad de detección. Sin embargo, antes de proceder con la dilatación de líneas se añade la transformada de Hough para detectar líneas entre los bordes encontrados.

En este sección del programa se emplea la función de Hough tres veces, la primera para encontrar líneas verticales entre -0,5 y 0,5 grados, la segunda para las líneas horizontales entre -90 y -89,5 grados, y la tercera para las líneas horizontales entre 89,5 y 89,9 grados. Esto se realizó para facilitar la búsqueda de las líneas de los marcos, aprovechando el hecho de que el sistema está enfocado a ventanas rectangulares y cuadradas. Cabe resaltar que, en el caso de necesitar adaptar el sistema a otras formas de ventanas, como por ejemplo las triangulares, es necesario hacer modificaciones en estos filtros de ángulos.

Para cada ejecución de la función anterior, primero se ingresa el máximo número de picos que la transformada debe encontrar, y el umbral a partir del cual una celda no será considerada como pico. Con los picos obtenidos al emplear la función de picos, se asigna el parámetro de la separación de pixeles a partir de la cual se considera que un segmento de línea es distinto, y el parámetro del número de pixeles mínimo que debe tener un segmento de línea.

Con los valores anteriores se emplea la función de líneas para finalmente obtener las líneas buscadas; no obstante, estas son dadas como coordenadas del punto inicial y el

punto final de la línea. Para poder aplicar las propiedades de regiones en las líneas, es necesario dibujar las líneas encontradas en una matriz de ceros del mismo tamaño que la imagen original. De esta forma, asignando el número de puntos, y empleando índices lineales a partir de los índices de filas y columnas que se obtuvieron del tamaño de la imagen, resulta una imagen en blanco y negro de las líneas encontradas por la transformada de Hough.

Luego de encontrar las líneas para los tres rangos de ángulos, se continua de la misma manera que en el primer programa preliminar que emplea solo algoritmos de bordes y propiedades de región, pero con la diferencia de que en lugar de dilatar las líneas de los bordes, en este caso se dilatan las líneas encontradas por la transformada de Hough. En resumen, se emplean los mismos filtros de regiones y parámetros de las propiedades de región, para encontrar el rectángulo más pequeño que contiene la región segmentada, obteniendo así los parámetros requeridos de ancho, alto y las coordenadas en X y Y del centroide del vidrio segmentado.

3.2.4 Resultados del segundo programa de segmentación preliminar

Con el segundo programa preliminar terminado y adaptado a la interfaz de usuario, se procedió a su ejecución en la misma imagen que no pudo ser segmentada correctamente utilizando solamente los algoritmos de bordes y, como se observa en la Figura 30, con la ayuda de la transformada de Hough, se logró una segmentación correcta del vidrio a pesar de los reflejos definidos.

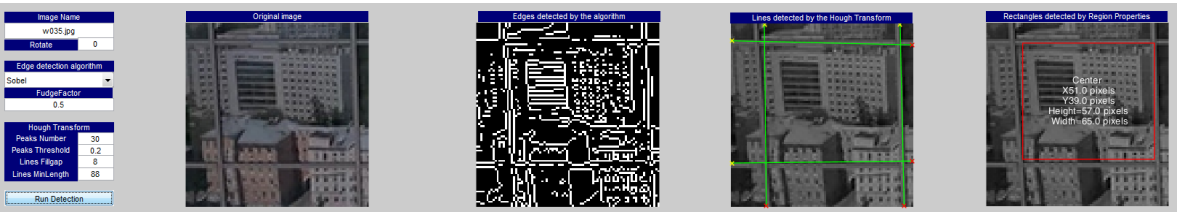

Figura 30. Segmentación exitosa empleando transformada de Hough.

Fuente: Elaboración propia, 2019.

Otro ejemplo de las mejoras en este segundo programa preliminar se puede ver en la Figura 31, donde se aprecia que, al emplear solamente algoritmos de bordes, el reflejo presente en el vidrio es detectado como borde, fragmentando las regiones del vidrio y

dando como resultado una segmentación incorrecta; sin embargo, al emplear la transformada de Hough y encontrar las líneas entre los bordes, se logra una segmentación exitosa del vidrio.

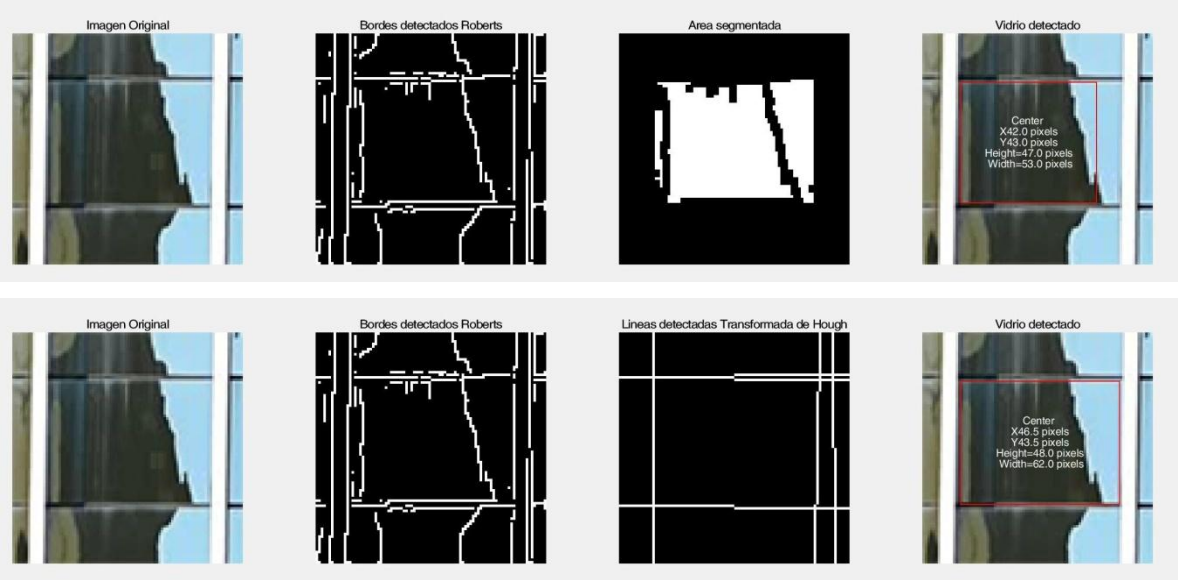

Figura 31. Ejemplo de las ventajas de usar la transformada de Hough para segmentar el vidrio en ventanas de rascacielos con reflejos.

Fuente: Elaboración propia, 2019.

De igual manera que con el primer programa preliminar, la validación de este segundo programa se realizó inicialmente por el investigador, determinando de forma aproximada si el programa realizó una segmentación correcta o no. Sin embargo, luego de realizar la red neuronal y obtener la matriz de validación, se regresó a este programa para realizar una validación más precisa.

La matriz de validación se comparó con los datos del segundo programa preliminar, obteniendo así el error en pixeles para cada uno de los cuatro parámetros del vidrio. Seguidamente, al emplear el máximo error en pixeles de los cuatro valores, se separaron los resultados en dos categorías; segmentación fallida si el error máximo es mayor a 5 pixeles, y segmentación correcta si el error máximo es menor o igual a 5 pixeles. Al mismo tiempo se elaboró un contador para calcular las segmentaciones correctas con un error de 1, 2, 3, 4 y 5 pixeles. Los resultados de emplear el segundo programa preliminar en las 400 imágenes de la base de datos se pueden ver en la Figura 32.

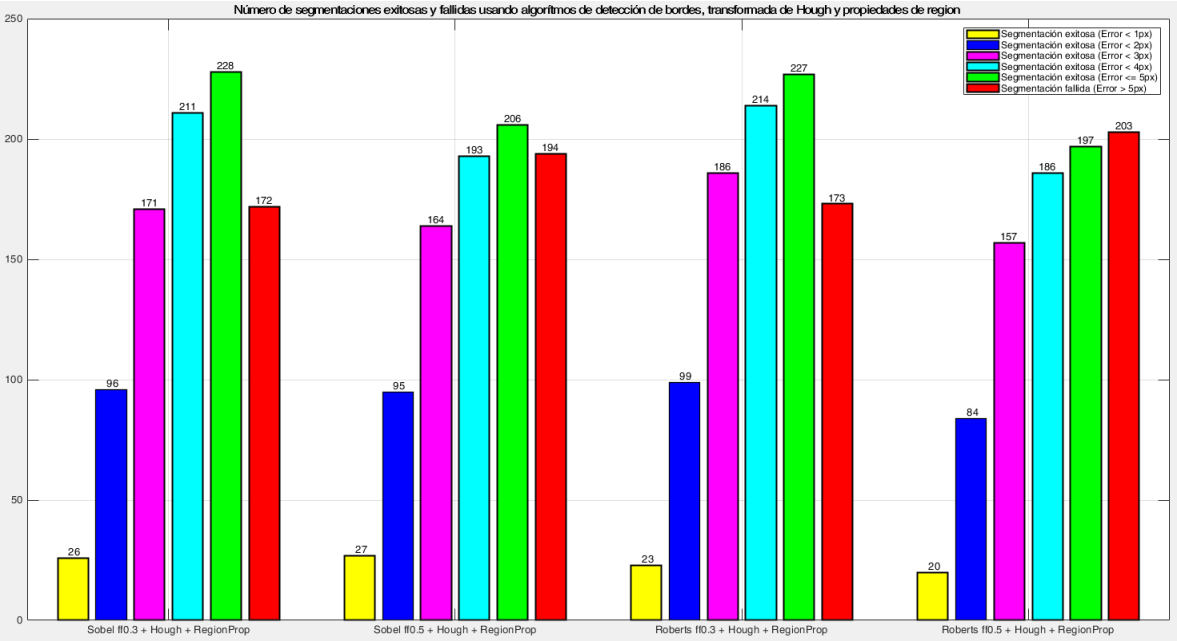

Figura 32. Resultados del segundo programa preliminar, empleando algoritmos de bordes, transformada de Hough y propiedades de región.

Fuente: Elaboración propia, 2019.

Al utilizar los dos algoritmos de bordes escogidos a partir de los resultados del primer programa preliminar, se obtuvieron resultados con dos valores de fudgefactor (0,5 y 0,3) distintos, con el fin de evaluar el mejor valor fijo a emplear. En cuanto a los parámetros de la transformada de Hough, se asignó un número de picos a detectar de 30, un umbral de picos de 0,2, una separación a partir de la cual se consideran segmentos de líneas distintos de 7 pixeles (≈ 8%) y un filtro de longitud mínima de línea de 83 pixeles (≈ 92%).

Los resultados del segundo programa preliminar, en el cual se incluyó la transformada de Hough con los parámetros fijos escogidos, mostraron una clara mejora en la cantidad de segmentaciones correctas, con 228 de 400 al emplear el algoritmo de bordes de Sobel y 227 de 400 al emplear el de Roberts.

Estos resultados evidenciaron la necesidad de incrementar el nivel de fiabilidad hasta uno que sea aceptable para el sistema que se va a emplear. Por esta razón, el siguiente paso realizado fue el de evaluar el potencial de emplear la metodología expuesta, y analizar cuantas segmentaciones correctas pueden ser logradas si se efectúa un ajuste manual de los parámetros de la transformada de Hough, hasta lograr una segmentación correcta a criterio del investigador. Esto se realizó con el fin de conocer los ajustes automáticos

necesarios para el programa multiagente final, así como las distintas características y estrategias que pudiesen ser incorporadas a este.

Utilizando la interfaz creada (Figura 33), se llevo a cabo la variación manual de todos los parámetros disponibles hasta lograr una segmentación correcta, y a partir de estos resultados se registraron las observaciones y los cambios de parámetros necesarios para lograr una segmentación correcta del vidrio.

Figura 33. Ejemplo de variación manual de parámetros en la interfaz.
Fuente: Elaboración propia, 2019.

Al realizar un ajuste manual de parámetros, se logró una segmentación exitosa en 364 de las 400 imágenes de la base de datos, demostrando el potencial de la transformada de Hough con parámetros variables, así como la dificultad presente en algunas ventanas añadidas a la base de datos. Cabe mencionar que el número elevado de segmentaciones exitosas se debe en gran medida a que se validó visualmente cada cambio de parámetro hasta lograr el resultado deseado. Lograr lo mismo pero de forma automática es más difícil, sin embargo, mediante el análisis de las soluciones que se encontraron, fue posible crear posteriormente nuevas estrategias para mejorar la fiabilidad de la segmentación en el programa multiagente final.

Una de los hallazgos más importantes, está representado en el hecho que, al extender el filtro de líneas a 88 pixeles, es decir, a casi todo el tamaño de la imagen (90x90), se

obtiene una estrategia efectiva para segmentar vidrio en ventanas continuas de rascacielos, tanto fachadas de vidrio continuo como marcos continuos. Sin embargo, no es recomendable extender el filtro de líneas para todos los casos, puesto que se pueden presentar marcos que no son completamente continuos, ya sea por un muro a un lado del marco, o porque la ventana está al borde lateral o superior del rascacielos.

Se observó también, que en imágenes nocturnas o imágenes que son brillantes, los marcos suelen mezclarse con el vidrio, lo que hace necesario detectar bordes menos definidos y disminuir el filtro de tamaño de líneas. No obstante, en algunos casos, se pueden detectar líneas en los reflejos si no se escoge la sensibilidad correcta del algoritmo de detección de bordes, lo cual exige la reducción del parámetro de separación a partir de la cual se consideran segmentos de líneas distintos.

Otro resultado significativo corresponde a la existencia de varios casos de ventanas con reflejos de sol, donde no solo es importante saber si el reflejo está posicionado en el vidrio o en el marco, sino también la posición del reflejo del sol en la imagen. Esto se debe a que el parámetro de la separación a partir de la cual se consideran segmentos de líneas distintos es usado para llenar una discontinuidad entre dos líneas. En ese sentido, si el reflejo del sol está en la parte central de la imagen, es posible llenar esta discontinuidad incrementando el parámetro mencionado, mientras que, si el reflejo del sol está ocultando la parte superior del marco, en esta sección no se detecta borde y no se encuentra la segunda línea a la que se extiende el segmento.

Con estas observaciones, hallazgos y ajustes fue posible lograr segmentaciones de ventanas con tipos específicos de reflejos de sol, marcos poco definidos, vidrios con varios objetos que se ven a través y reflejos en distintas condiciones de iluminación (Figura 34).

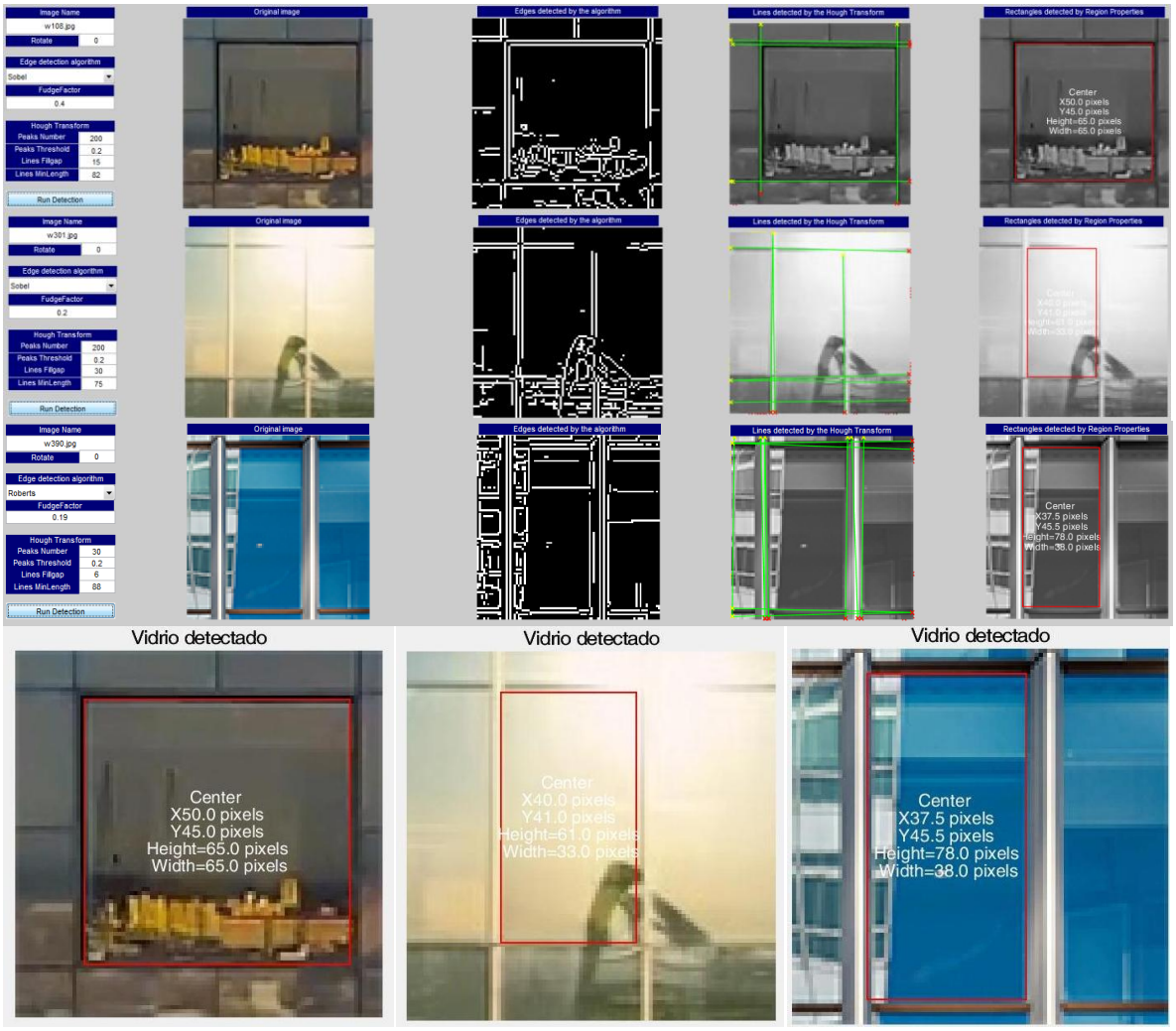

Figura 34. Segmentación exitosa empleando transformada de Hough con umbrales variados manualmente.

Fuente: Elaboración propia, 2019.

Cabe resaltar nuevamente que, en varios casos, los errores de exactitud se debieron a que algunas imágenes de la base de datos tienen ventanas que no están rectificadas. Por lo tanto y, a pesar de que esta situación no se presentará en la operación del dron, se incluyeron imágenes con pequeñas inclinaciones y distorsiones para que el sistema multiagente final fuese más robusto.

Finalmente, teniendo en cuenta la existencia de varios casos en los que ningún ajuste manual logró la segmentación correcta, se procedió a implementar métodos de inteligencia artificial para dar una posible solución a estos casos más complejos.

3.3 Segmentación de vidrio en ventanas de rascacielos empleando redes neuronales convolucionales

Luego de analizar los métodos potenciales de la inteligencia artificial a utilizar en la segmentación de vidrio en ventanas de rascacielos, se seleccionaron dos métodos de aprendizaje supervisado como los más indicados para el proyecto: las Maquinas de Soporte Vectorial (SVM) y las Redes Neuronales Convolucionales (CNN). Sin embargo, luego de realizar las pruebas iniciales con las SVM, se decidió priorizar la implementación y entrenamiento de la CNN en razón a que, en el caso de las necesidades del sistema objeto de estudio, no se busca saber si lo observado es una ventana o no, sino que parte de la imagen corresponde al vidrio de la ventana del rascacielo. Lo anterior puede ser logrado encontrando el vidrio o encontrando el marco que delimita dicho vidrio. Analizando estos requerimientos se decidió emplear la segmentación semántica mediante redes neuronales convolucionales, para etiquetar los pixeles de la imagen en distintas categorías o etiquetas de clase.

3.3.1 Image Labeler y etiquetado semántico

Antes de realizar la red neuronal, es necesario conseguir el parámetro u objeto requerido para su entrenamiento supervisado, normalmente denominado ground truth, y el cual se obtiene asignando etiquetas semánticas o categorías a cada pixel perteneciente a las imágenes de la base de datos. Para realizar esta tarea se empleó el Image Labeler de Matlab (Figura 35), en el cual se cargaron las 400 imágenes de ventanas de rascacielos pertenecientes a la base de datos y, empleando las herramientas que este contiene, se realizó el etiquetado semántico requerido.

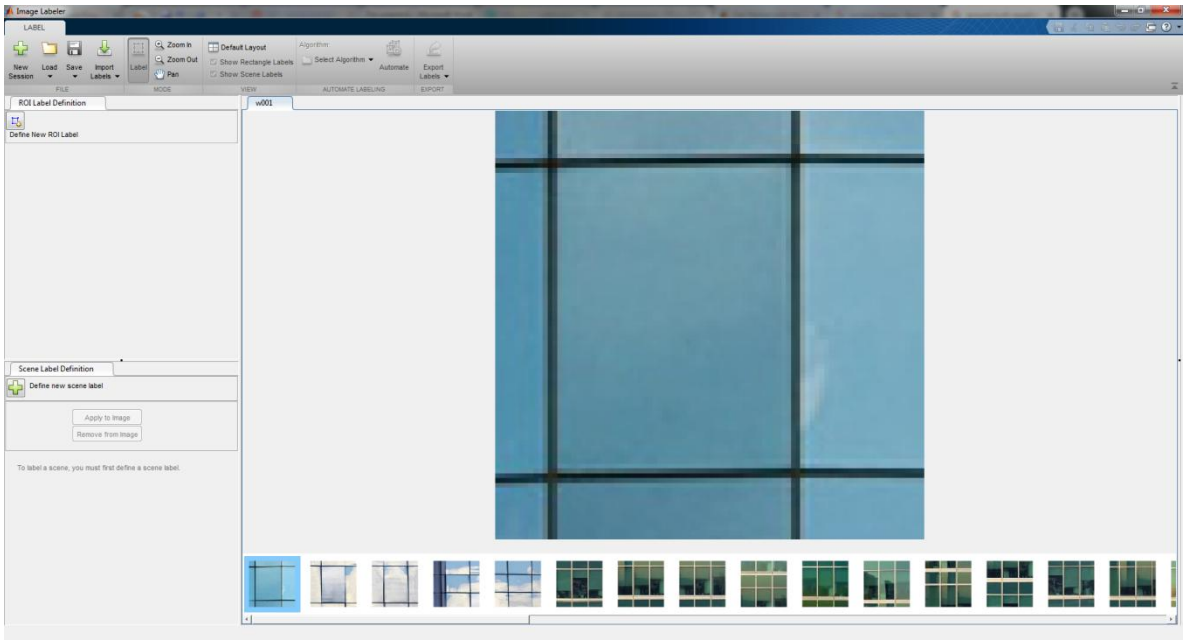

Figura 35. Image Labeler de Matlab con la base de datos añadida.

Fuente: Elaboración propia, 2019.

Para la clasificación se crearon tres etiquetas de pixeles correspondientes a los objetos que se requieren identificar o segmentar en la imagen, “Glass” para la parte correspondiente al vidrio, “Frame” para el marco de la ventana y “Wall” para muros o separaciones entre ventanas. Al añadirlas, a cada una de estas etiquetas se le asignó un color específico, en este caso rojo para la etiqueta “Glass”, azul para la etiqueta “Frame” y amarillo para la etiqueta “Wall”.

Con las etiquetas creadas se procedió a asignarle una de estas a cada uno de los pixeles correspondientes a cada imagen de la base de datos, lo cual se realizó empleando la brocha del programa de etiquetado para pintar sobre cada imagen según el criterio del investigador (Figura 36). Al mismo tiempo, se empleó el slider de opacidad para facilitar las tareas de asignación de etiquetas, evaluación de secciones faltantes por pintar y, en general, optimización de la precisión del etiquetado.

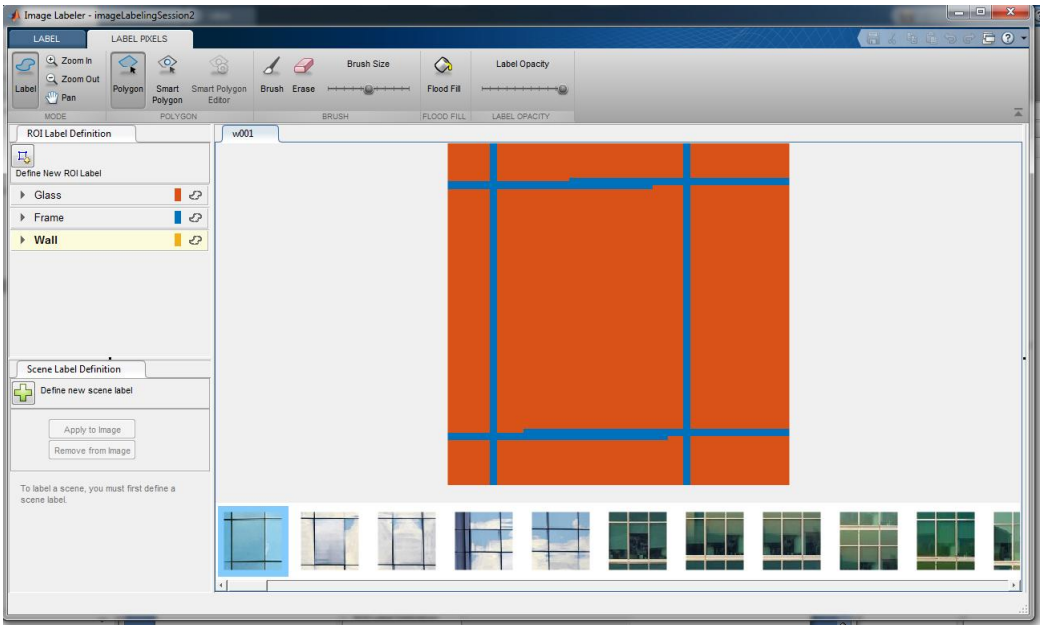

Figura 36. Ejemplo de asignación de etiquetas en el Image Labeler de Matlab.

Fuente: Elaboración propia, 2019.

Un ejemplo de imagen que contiene las tres etiquetas mencionadas y asignadas a una imagen de la base de datos se puede ver en la Figura 37, en la cual se escogió una opacidad cercana al 50%, con el fin de que se aprecien los detalles tanto de la imagen original como del etiquetado realizado.

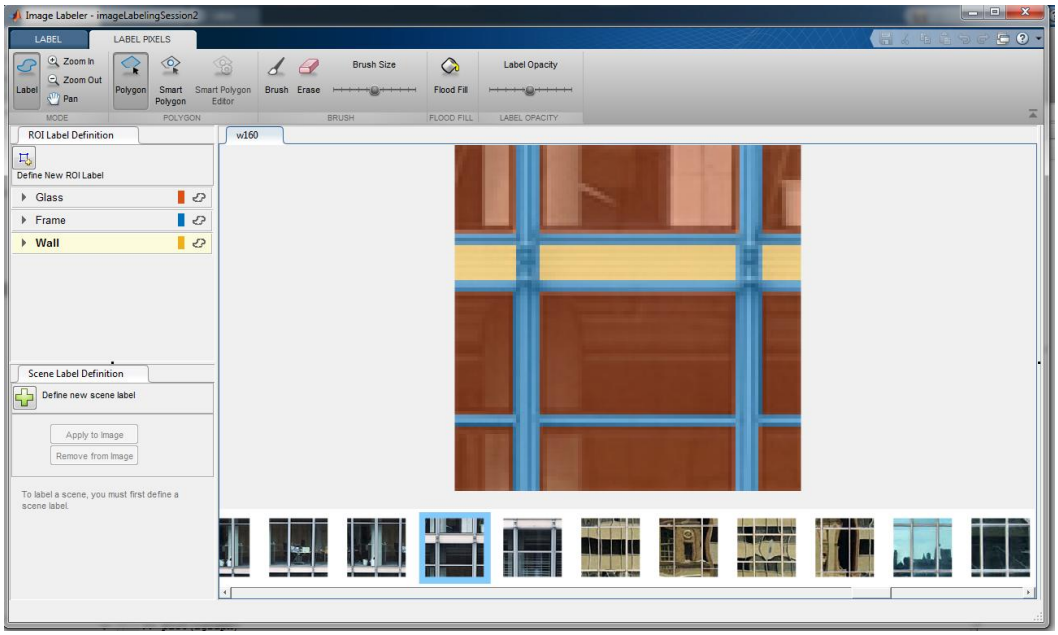

Figura 37. Ejemplo con transparencia de las etiquetas en el Image Labeler de Matlab.

Fuente: Elaboración propia, 2019.

Luego de realizar la asignación de las etiquetas a las 400 imágenes de la base de datos, se guardaron todos los archivos necesarios, resultando así un archivo ground truth y una carpeta que contiene las 400 etiquetas finales. Al mismo tiempo, como se mencionó durante el programa de segmentación preliminar, a partir de las imágenes etiquetadas se creó una matriz de validación que contiene los cuatro parámetros verdaderos que se requieren de cada imagen de la base de datos. Esto se logró aplicando propiedades de regiones y los filtros seleccionados durante el programa preliminar, para realizar la segmentación en cada una de las 400 imágenes etiquetadas.

Con estos datos obtenidos se procedió a la creación de la red neuronal, para la cual se decidió aprovechar el aprendizaje transferido de una red neuronal existente, y se procedió a su modificación de acuerdo con los requerimientos del sistema.

3.3.2 Creación de la red neuronal convolucional

Para comenzar, se descargó e instaló el add-on de Matlab llamado “Deep Learning Toolbox Model for VGG-16 Network”, y luego se cargó la red neuronal convolucional preentrenada VGG-16. Esta red fue previamente entrenada con aproximadamente 1,2 millones de imágenes del ImageNet Dataset por el grupo de geometría visual de la Universidad de Oxford, para la clasificación de objetos. Lo anterior, con el fin de aprovechar el aprendizaje transferido para facilitar la construcción de la arquitectura de la red neuronal y su entrenamiento.

Al ingresar el directorio en donde se encuentra la base de datos, y con la función imageDatastore de Matlab, se creó el objeto donde se almacenaron las imágenes con las etiquetas asociadas y el ground truth obtenido anteriormente con el Image Labeler.

Antes de proceder con la arquitectura de la red neuronal, se crearon las clases con los nombres de las etiquetas empleadas, y se realizó un conteo del número de pixeles por etiqueta en toda la base de datos, con el fin de analizar la frecuencia con que estas se presentan.

Idealmente, todas las clases deberían tener un número cercano de observaciones; sin embargo, la desigualdad en el número de observaciones por etiqueta es un problema común de las bases de datos. Dicha inconsistencia también se presentó en el caso de la segmentación de vidrio en ventanas de rascacielos, en donde las imágenes de la base de datos presentan una mayor cantidad de pixeles asignados al vidrio que al marco o al muro (Figura 38), debido a que este cubre más área en la imagen. Cabe resaltar que este desequilibrio puede ser perjudicial en el proceso de entrenamiento, puesto que la red neuronal estará predispuesta a favorecer la clase predominante.

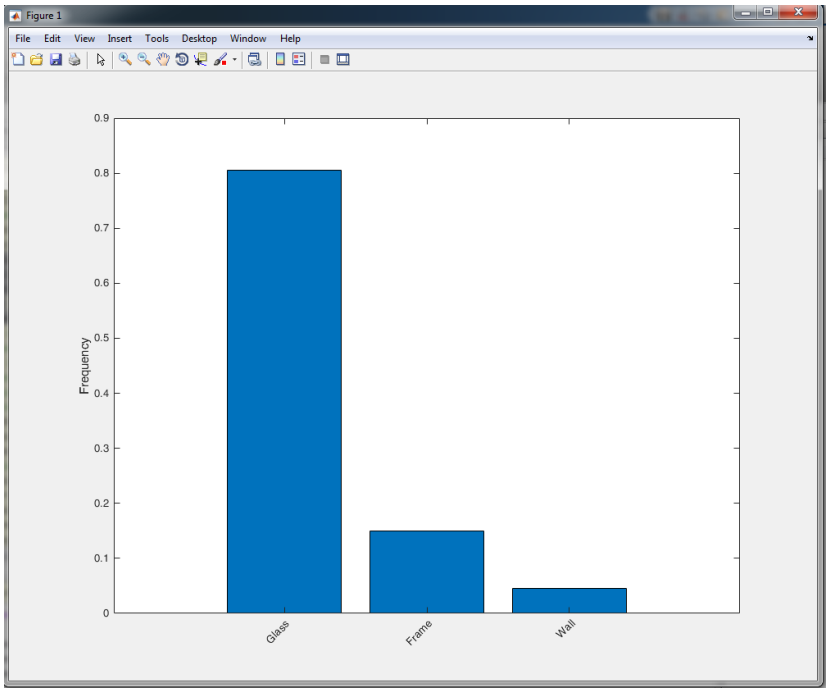

Figura 38. Frecuencia con que se presentan las etiquetas semánticas en la base de datos.
Fuente: Elaboración propia, 2019.

De acuerdo con lo enunciado, se realizó un balance de clases modificando los pesos de las clases según la frecuencia en que estos se presentan.

A continuación, con las clases balanceadas, se procedió a realizar las modificaciones necesarias a la arquitectura de la red neuronal de la siguiente manera. Primero se eliminó la última capa de la red VGG-16, la cual contenía las etiquetas preentrenadas, para luego insertar una nueva capa que contiene los pixeles de clasificación balanceados que se crearon para el sistema. Por último, se conectaron las capas para finalizar la arquitectura de la red neuronal convolucional (Figura 39).

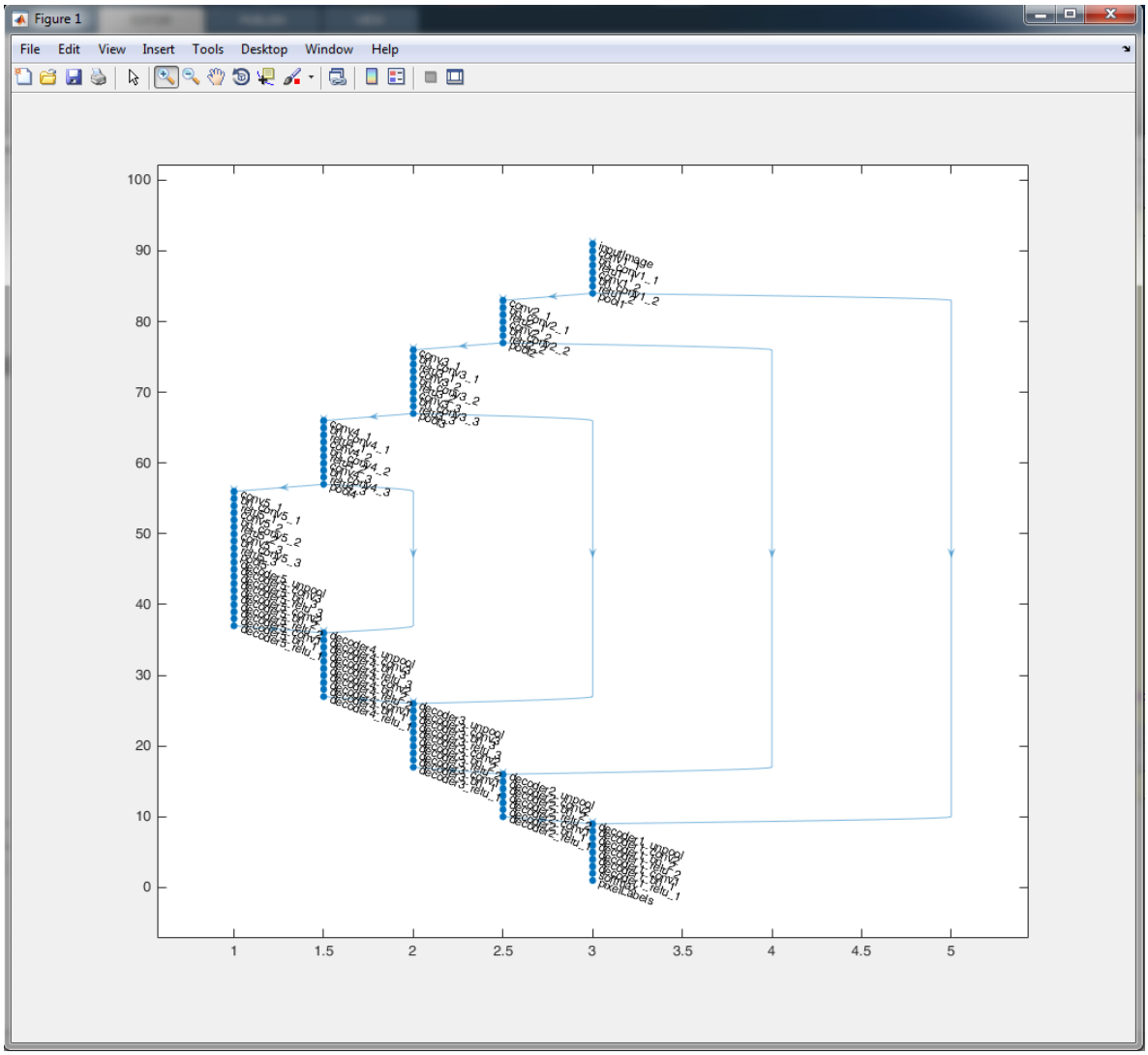

Figura 39. Arquitectura de la red neuronal convolucional empleada en el proyecto.
Fuente: Elaboración propia, 2019.

3.3.3 Entrenamiento de la red neuronal convolucional

Antes de realizar el entrenamiento de la red, se seleccionaron las opciones de entrenamiento comúnmente recomendadas en la segmentación semántica enfocada a vehículos de conducción autónoma, puesto que en estos casos se suelen segmentar varios objetos como peatones, vehículos, señales de tráfico y otros elementos de la carretera. Estos parámetros se emplearon como un punto de partida para el ajuste y evaluación de los más adecuados para el caso de la segmentación de vidrio en ventanas de rascacielos.

Otro paso adicional realizado, consistió en emplear la técnica de aumento de datos (data augmentation) para solucionar el problema de la cantidad de datos requerida para el entrenamiento de modelos, que es tan común en algoritmos de aprendizaje profundo. Lo anterior se realizó introduciendo perturbaciones en los datos originales, específicamente tres para las primeras pruebas realizadas: un reflejo en el eje X y dos traslaciones en los ejes X y Y.

Finalmente, se procedió con el entrenamiento de la red neuronal empleando un conjunto de entrenamiento, correspondiente a las primeras 200 imágenes de la base de datos, y un algoritmo de optimización de entrenamiento llamado descenso de gradientes estocástico con momento.

Las 200 imágenes restantes, es decir, las imágenes con nomenclatura entre 201 y 400, se emplearon como el conjunto de validación en futuras redes entrenadas. Cabe mencionar que normalmente se valida de forma directa el resultado con el porcentaje de pixeles correctamente clasificados en este conjunto, no obstante, en el caso de la presente investigación, se tuvo en cuenta que el requerimiento estaba enfocado en encontrar los parámetros de altura y ancho del vidrio, así como la posición del centroide, lo cual requiere de una validación específica que se explica más adelante.

Luego de realizar todos los pasos previos, se aplicó la primera red neuronal entrenada a todas las imágenes de la base de datos y se compararon los resultados con las imágenes originales. Debido a que esta fue la primera red neuronal entrenada, solamente se realizó una comparación visual a criterio del investigador.

Los resultados de la primera red neuronal permitieron evidenciar que, a pesar de emplear un balanceo de clases, en el caso de la clase “Wall”, no se realizaba un reconocimiento correcto debido a la falta de referencias empleadas durante el entrenamiento, resultando en pixeles individuales que la red consideraba un muro, pero que en realidad estaban ubicados en los reflejos del vidrio (Figura 40). Al mismo tiempo, con los resultados obtenidos se demostró que no es necesario el reconocimiento de los muros para los propósitos de segmentación del vidrio, razón por la cual se decidió no emplear esta clase en las siguientes redes neuronales entrenadas.

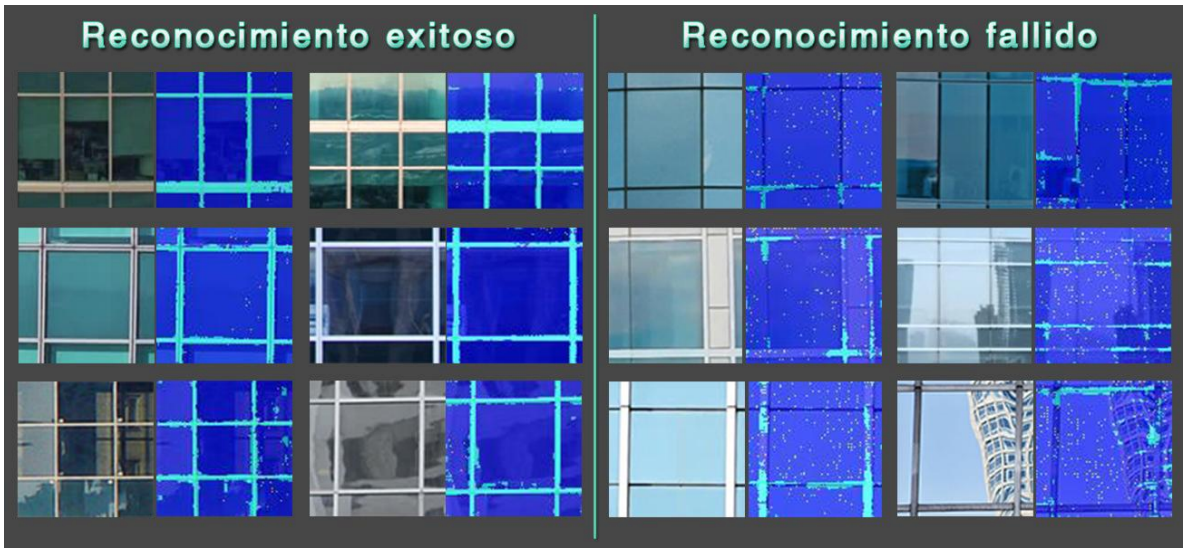

Figura 40. Ejemplos de los resultados obtenidos con la primera red neuronal convolucional.
Fuente: Elaboración propia, 2019.

Es necesario resaltar que la mitad de los ejemplos ilustrados de cada categoría pertenecen al conjunto de prueba y la otra mitad al conjunto de entrenamiento. Lo anterior se realizó para demostrar que el hecho de haber entrenado la red con una imagen no asegura que esta aprendió todas las imágenes del conjunto de entrenamiento, pero al mismo tiempo demuestra un aprendizaje por su capacidad de reconocer correctamente marcos y vidrios en imágenes a las que la red no fue expuesta durante el entrenamiento.

No se realizó una validación completa de esta red neuronal en la medida que esta fue una red de prueba para ajustes; sin embargo, se pudo observar que algunos de los errores que se presentan en el reconocimiento realizado por la red neuronal pueden ser tolerados, siempre y cuando estos errores en pixeles no impacten la segmentación del vidrio y la obtención de los parámetros requeridos. Adicionalmente, se observó que es posible emplear la transformada de Hough en los resultados presentados por la red neuronal para reducir el impacto de los errores de exactitud.

En las siguientes redes neuronales convolucionales entrenadas, se decidió analizar el efecto del tiempo de entrenamiento, el impacto de las opciones de entrenamiento en los resultados y, al mismo tiempo, la exactitud de estos resultados. Cabe destacar que la mayoría de estas redes no presentaron una mejora debido a la cantidad de épocas

asignadas para el entrenamiento, así como el valor de algunas de las opciones de entrenamiento, lo cual puede verse en la Figura 41, que corresponde a la red neuronal convolucional entrenada por 10 épocas.

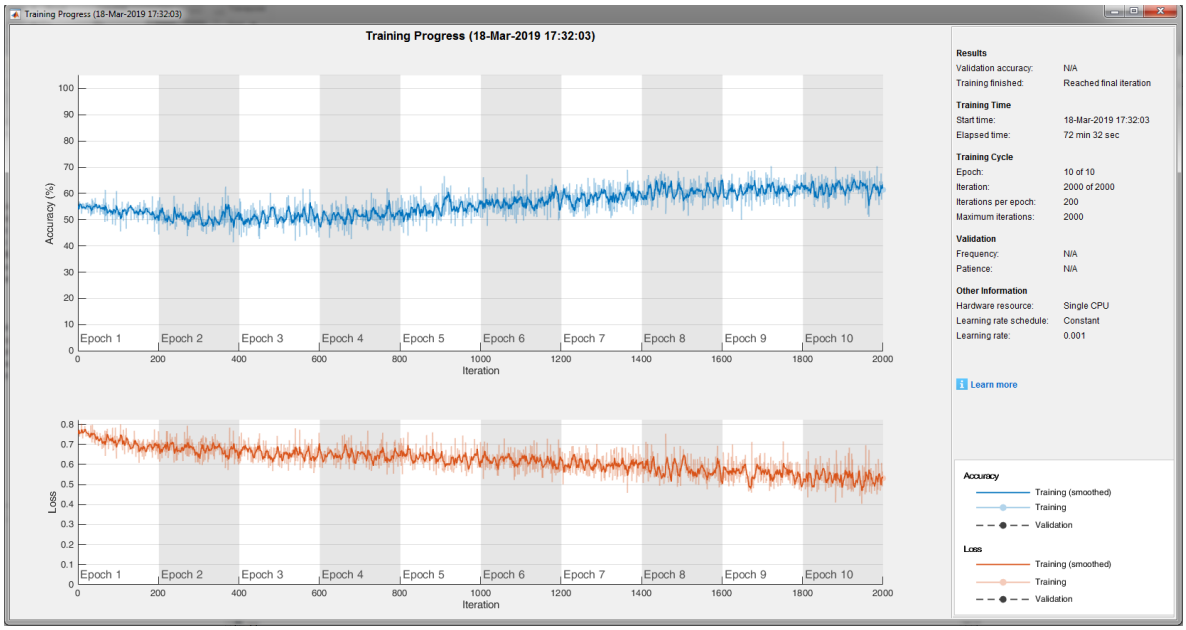

Figura 41. Progreso del entrenamiento de una red neuronal convolucional de prueba.
Fuente: Elaboración propia, 2019.

Lo anterior demostró la necesidad de aumentar la cantidad de épocas a utilizar en el entrenamiento y que, por consiguiente, este requiere más tiempo para completarse en futuros entrenamientos.

Otro factor que se evaluó, es el de incrementar el tamaño del minilote, haciendo que la red realice el entrenamiento con paquetes de imágenes en lugar de cada imagen individual. Para ello se aprovechó la evaluación del gradiente de la función de perdida para actualizar los pesos. Al realizar un entrenamiento de red neuronal, incrementando el minilote a 10 y el número de épocas a 40, se observó que los resultados no muestran mejoras en la exactitud, lo cual pudo ser debido a la influencia de la utilización de un minilote mayor en el número de iteraciones (Figura 42).

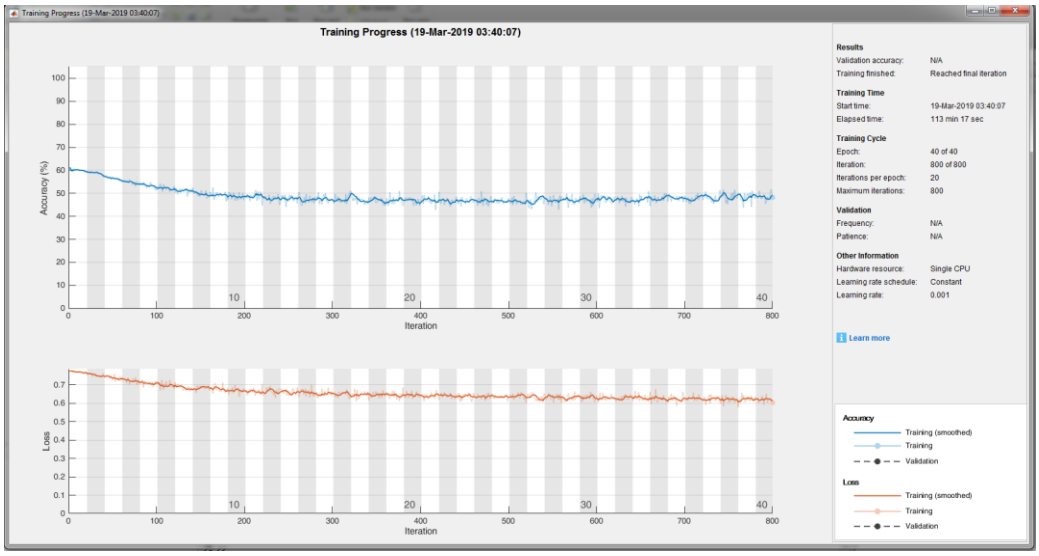

Figura 42. Progreso del entrenamiento de una red neuronal convolucional fallida.

Fuente: Elaboración propia, 2019.

Posteriormente, se realizaron dos redes neuronales convolucionales adicionales con los mismos parámetros mencionados, pero con un mayor número de épocas. Con un entrenamiento que duró alrededor de 5 horas, estos no presentaron mejoras en la exactitud del reconocimiento, y la mayoría de las imágenes de la base de datos fallaron en la segmentación del vidrio.

Como se mencionó anteriormente, si bien la exactitud indica el porcentaje de pixeles que se clasificaron correctamente, en consideración a la presencia elevada de vidrio en las imágenes de ventanas de rascacielos, un porcentaje aproximado a 60% no es suficiente para realizar una segmentación correcta de los objetos buscados (Figura 43).

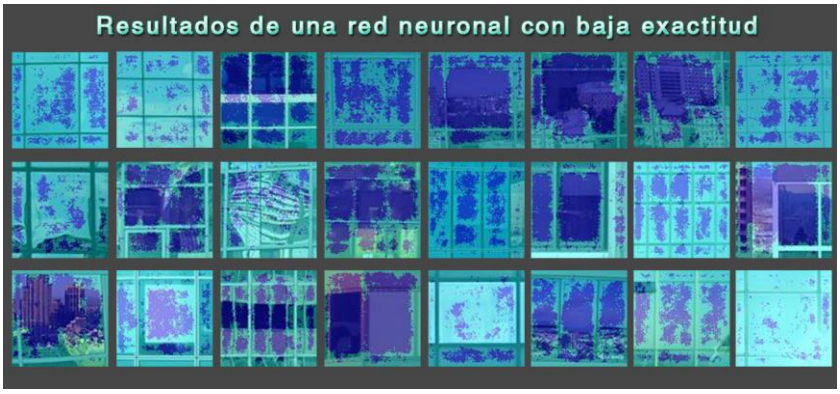

Figura 43. Ejemplos con transparencia de los resultados presentados por las redes neuronales fallidas.

Fuente: Elaboración propia, 2019.

Para el último entrenamiento se decidió reducir el minilote a 1 y aumentar el número de épocas a 120. El entrenamiento de esta red neuronal duró alrededor de 15 horas y permitió obtener una clara mejora en la exactitud, la cual alcanzó valores alrededor de 90% (Figura 44).

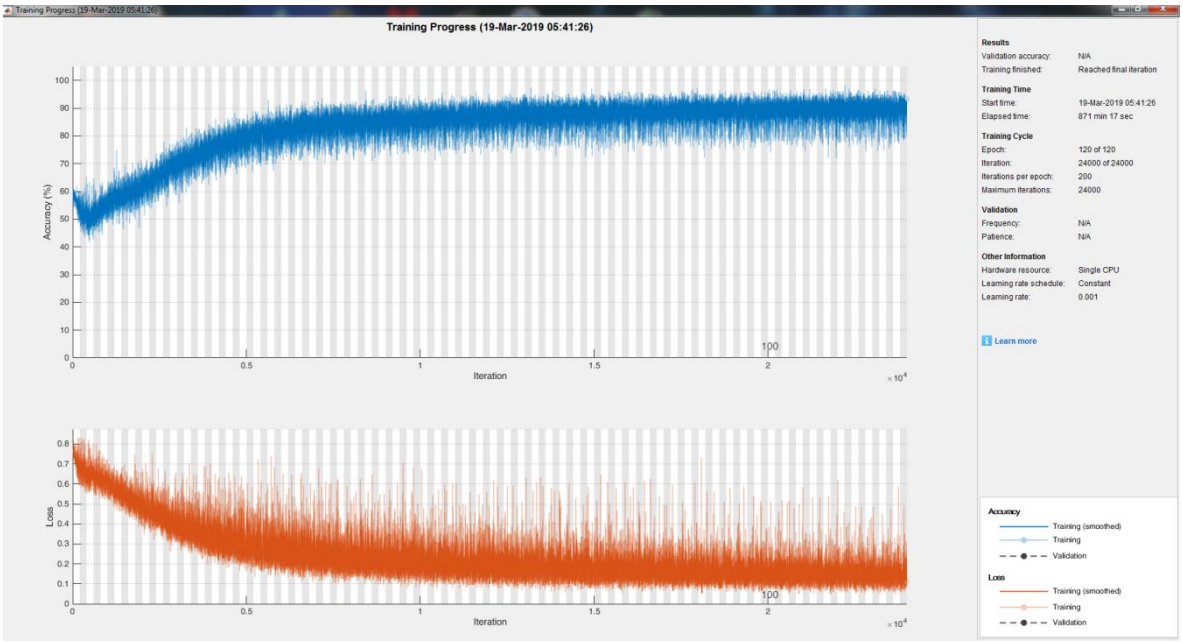

Figura 44. Progreso del entrenamiento de la red neuronal convolucional final.
Fuente: Elaboración propia, 2019.

3.3.4 Resultados obtenidos por la red neuronal convolucional

La red neuronal creada se aplicó primeramente a las 200 imágenes del conjunto de entrenamiento para evaluar la segmentación del vidrio más cercano al centro y los parámetros resultantes. Posteriormente, esta red neuronal también fue aplicada al conjunto de prueba, el cual contiene imágenes de ventanas de rascacielos que no estuvieron presentes en el entrenamiento.

Al comparar los datos resultantes con la matriz de validación se obtuvo el error en pixeles para cada uno de los cuatro parámetros del vidrio. Se calculó el máximo error en pixeles de los cuatro valores y los resultados fueron divididos en dos categorías; segmentación fallida si el error máximo es mayor a 5 pixeles, y segmentación exitosa si el error máximo

es menor o igual a 5 pixeles. Al mismo tiempo u de igual forma a lo que se realizó con el programa preliminar, se elaboró un contador para calcular las segmentaciones correctas con un error de 1, 2, 3, 4 y 5 pixeles. Los resultados de aplicar la red neuronal convolucional en el conjunto de entrenamiento y en el conjunto de validación se pueden ver en la Figura 45.

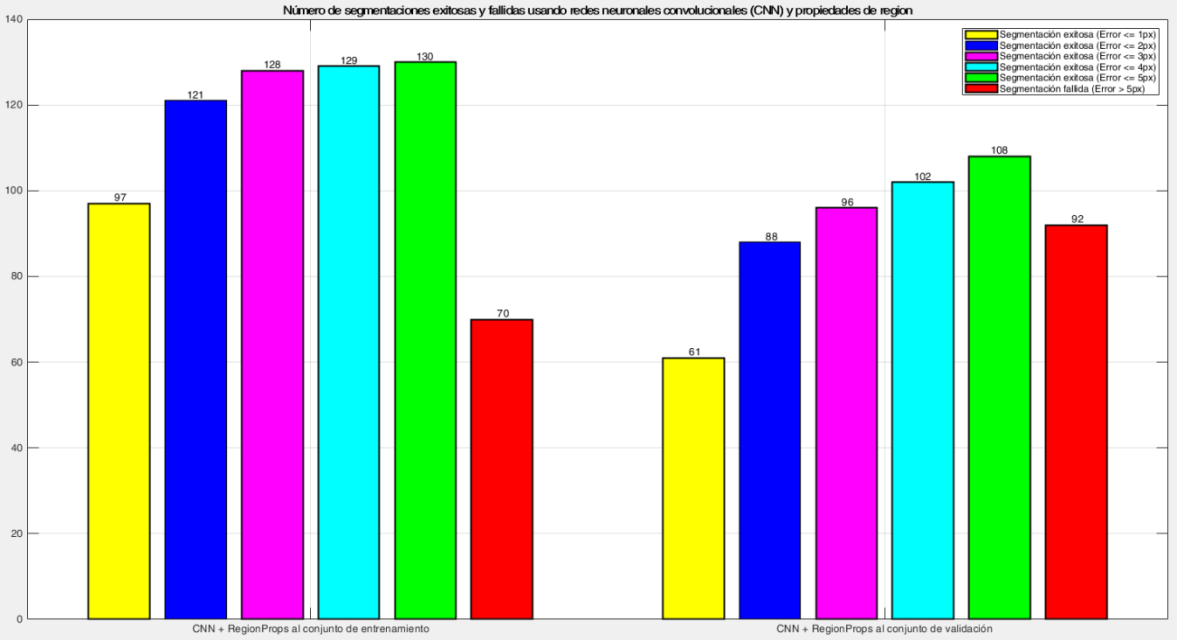

Figura 45. Resultados al emplear la red neuronal con propiedades de región para segmentar el vidrio en el conjunto de entrenamiento y el conjunto de validación.

Fuente: Elaboración propia, 2019.

Lo que más resaltó de estos resultados fue la exactitud de la segmentación del vidrio, tanto en imágenes con las que entrenó la red, como en imágenes a las que la red neuronal no fue expuesta. También es importante destacar la cantidad de segmentaciones exitosas, considerando que la red neuronal convolucional fue entrenada con solamente 200 imágenes, lo cual demuestra el alto potencial de este método con una base de datos ampliada.

Otra observación que se pudo apreciar, corresponde a que, a pesar del buen trabajo realizado por la red neuronal en el reconocimiento del vidrio y los marcos, se obtuvieron varias segmentaciones fallidas, debido a que los errores de reconocimiento en algunos pixeles crean discontinuidades en los marcos, evitando que las propiedades de regiones segmenten correctamente el vidrio. Además, adicional al caso común de alta exactitud en

la red neuronal y segmentación exitosa, se pueden presentar otros tres casos diferentes, los cuales se pueden visualizar en la Figura 46. Esto demostró la importancia de validar con la exactitud del sistema que segmenta los parámetros buscados y no con la exactitud de la red neuronal asociada a la clasificación de los pixeles.

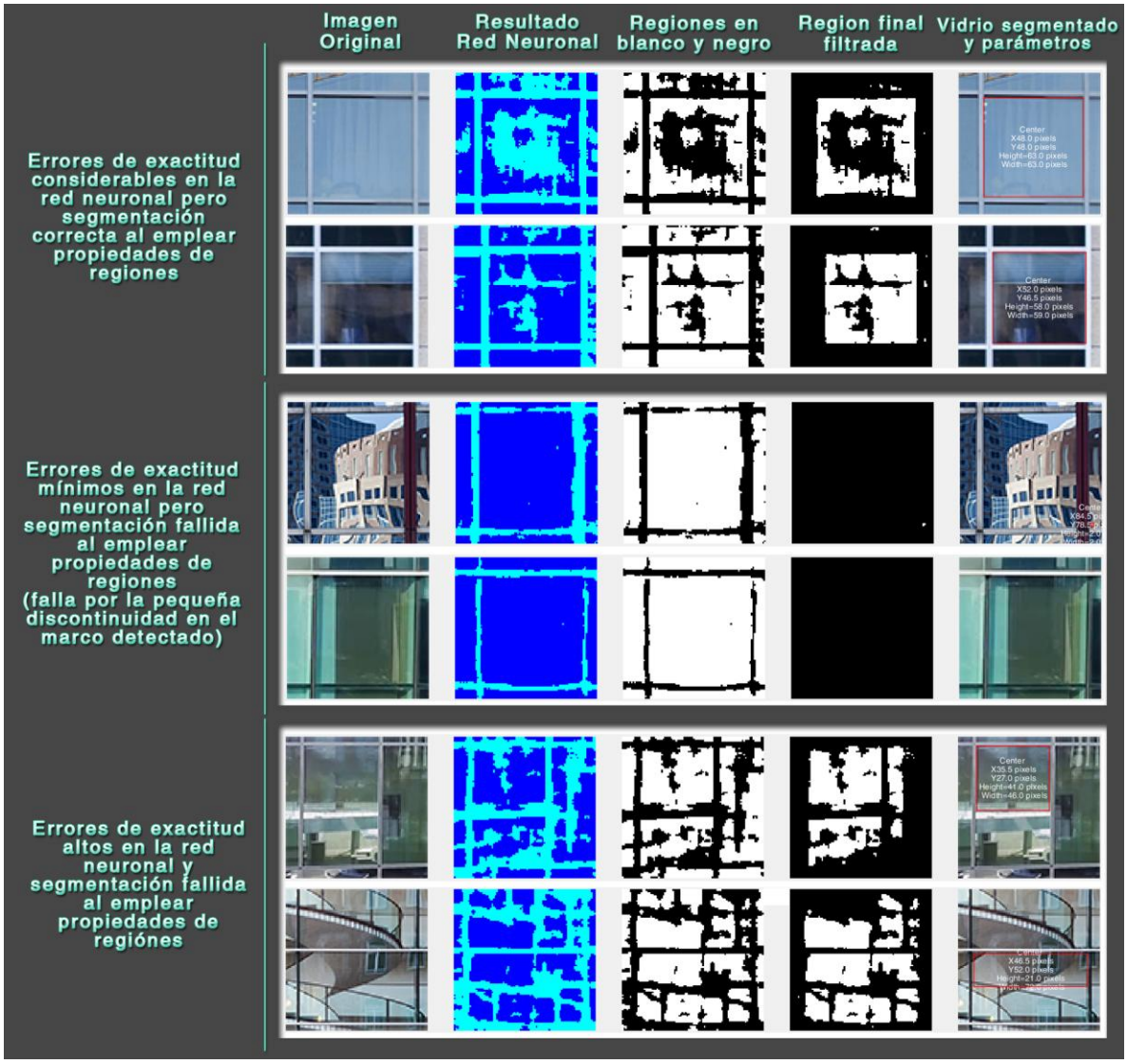

Figura 46. Ejemplos de los resultados de la red neuronal para visualizar la diferencia entre exactitud de la red neuronal y la exactitud de la segmentación del vidrio.

Fuente: Elaboración propia, 2019.

Finalmente, la comparación de los resultados presentados por la red neuronal y el programa preliminar, permitió evidenciar que la red neuronal tuvo un mayor número de segmentaciones exitosas, en la medida que logró segmentar correctamente 238 de las

400 imágenes de la base de datos. Adicional a lo enunciado, la aplicación de la transformada de Hough a los resultados presentados por la red neuronal antes de emplear las propiedades de región para segmentar el vidrio, posibilitó aumentar la segmentación correcta a 257 imágenes de la base de datos, aunque al mismo tiempo se presentó una reducción en la exactitud en pixeles de los parámetros conseguidos mediante la segmentación (Figura 47).

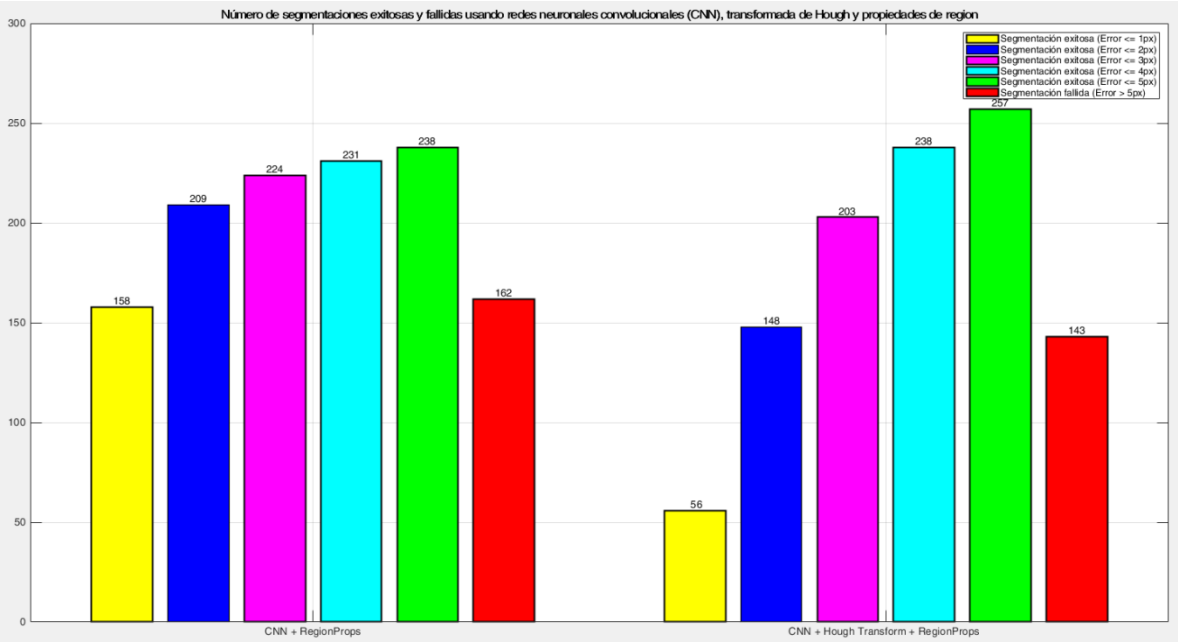

Figura 47. Resultados al emplear la red neuronal, transformada de Hough y propiedades de región para segmentar el vidrio en las imágenes de la base de datos.

Fuente: Elaboración propia, 2019.

Como resultado, la última red neuronal entrenada fue escogida para ser empleada en el sistema multiagente final, teniendo siempre presente que los resultados observados presentaron varias maneras y posibilidades de optimización de la red neuronal convolucional. De esta manera, teniendo en cuenta que se obtuvieron resultados aceptables con la red entrenada, se decidió proceder con la creación del árbol de decisión y la integración de la red neuronal convolucional al sistema multiagente mencionado.

3.4 Árbol de decisión y sistema multiagente final

Habiendo evaluado varios métodos para la segmentación de vidrio en ventanas de rascacielos, quedó evidenciado que emplear un solo método no es suficiente para una segmentación de vidrio confiable en ventanas de rascacielos, exceptuando posiblemente una futura mejora de la red neuronal convolucional con una base de datos más amplia para su entrenamiento.

Lo anterior se debe a la gran cantidad de variables que influyen en las ventanas, así como a las características propias de estas, producto de lo cual suele ocurrir un conflicto en los algoritmos empleados, en la medida que, si bien un cambio de variable en los filtros empleados permite segmentar el vidrio en un tipo concreto de ventana, al mismo tiempo dificulta o imposibilita la segmentación en otro tipo de ventana. Por esta razón, se escogió emplear un árbol de decisión para integrar las distintas técnicas, algoritmos y estrategias que fueron evaluadas durante la investigación, y así crear un sistema de visión artificial multiagente para la segmentación de vidrio en ventanas de rascacielos.

Para continuar, es pertinente presentar una breve relación de los resultados obtenidos anteriormente: i) la evaluación de los algoritmos de bordes, evidenció que los algoritmos de Sobel y Roberts demostraron ser los más apropiados para la detección de bordes en ventanas de rascacielos; ii) la transformada de Hough, demostró ser un método efectivo para encontrar los marcos de una ventana y diferenciarlos con los reflejos u objetos que se ven a través del vidrio, al emplear sus filtros de tamaño de líneas y su capacidad de obviar discontinuidades entre líneas; iii) las propiedades de región, resultaron ser la forma más apropiada para filtrar las áreas resultantes y, con el rectángulo delimitador, facilitan la obtención de los parámetros del vidrio; y por último iv) la red neuronal convolucional demostró ser el mejor método evaluado en cuanto a la exactitud de la segmentación y la cantidad de imágenes de la base de datos en las que se segmentó correctamente el vidrio, además de ser el método con más potencial de mejora.

A partir de todos los resultados obtenidos, fue posible elaborar diferentes estrategias para la creación del árbol de decisión, asegurando que este brinde solución a un número mayor de imágenes de la base de datos.

3.4.1 Estrategias para el desarrollo del árbol de decisión

La primera estrategia surgió al observar la diferencia entre la cantidad de pixeles detectados como bordes, los cuales se presentan tanto en las ventanas con reflejos u objetos que se ven a través del vidrio, como en las que esto no ocurre en gran medida. Lo anterior indicó que, en los vidrios con pocos objetos o reflejos de objetos, es posible emplear directamente algoritmos de bordes y propiedades de regiones para lograr la segmentación del vidrio y que, al mismo tiempo, una mayor presencia de bordes indica la necesidad de emplear transformada de Hough o métodos de inteligencia artificial para detectar correctamente los bordes de los marcos. Por esta razón, se implementó un algoritmo para la contabilización de pixeles de bordes, detectados a partir de los resultados del algoritmo de bordes seleccionado, y lo que permitió a su vez separar estos dos casos en el árbol de decisión.

Cabe resaltar que la realización de un solo conteo no asegura una clasificación correcta, debido principalmente a la variable fudgefactor, que como se mencionó anteriormente se modifica para variar la sensibilidad del algoritmo de detección de bordes. Este factor hace que haya casos en los que no se detecten los reflejos por la baja sensibilidad o debido a que son fotos de ventanas tomadas de noche o con poca iluminación. Para solucionar lo anterior, se empleó una doble verificación de conteo de bordes, la primera con un fudgefactor de 0,5 y luego otra con un fudgefactor de 0,2, es decir, con mayor sensibilidad del algoritmo de detección de bordes. Al respecto, se determinó que, si el porcentaje de bordes no varía considerablemente al aumentar la sensibilidad, esto indica que no hay un reflejo en el vidrio u objetos que fueron obviados por la sensibilidad de detección, y que se pueden aplicar directamente el algoritmo de detección de bordes y los filtros de propiedades de región.

La segunda estrategia surge de observar los resultados dados por dos tipos predominantes de marcos que se pueden presentar en las ventanas de rascacielos, es decir, marcos continuos entre secciones de vidrio y marcos discontinuos separados comúnmente por un muro o cuando la ventana se encuentra al borde del edificio.

Durante el desarrollo del programa preliminar se expuso que, si bien es posible filtrar por tamaño de líneas empleando la transformada de Hough, si se asigna un filtro de línea fijo,

uno de los dos casos mencionados anteriormente será filtrado automáticamente, presentando así la necesidad de adaptar el filtro de líneas según el tipo de marco.

En esta estrategia se utiliza la transformada de Hough con filtros de longitud de línea y de llenado de discontinuidades entre líneas con porcentajes bajos (entre 10% y 30%), con el fin de encontrar la mayor cantidad de líneas entre los bordes detectados. A partir de los resultados, se encuentra la línea horizontal y vertical de mayor tamaño, las cuales se convierten a porcentajes con respecto a la imagen original, se reducen un 5% y, por último, se convierten en el porcentaje del filtro de longitud de línea la segunda vez que se emplea la transformada de Hough.

La tercera estrategia surgió al evaluar los resultados en las ramas del árbol de decisión, lo que permitió evidenciar que, además de una segmentación exitosa, se presentaban dos casos importantes de diferenciar y a los cuales se decidió hacer referencia como error de segmentación y fallo de segmentación.

Un error de segmentación se refiere a cuando el sistema multiagente considera que encontró una solución para la segmentación del vidrio, pero la exactitud en pixeles indica un error mayor a 5 pixeles comparado con la matriz de validación. Esta matriz que fue creada por el investigador para la validación, no estará presente en las nuevas ventanas que se le presentarán al sistema durante su operación. Por esta razón, cuando se detecta un error de segmentación, este resultado no puede ser empleado como un condicional para generar nuevas ramas en el árbol de decisión.

Por otro lado, un fallo de segmentación sucede cuando se emplean las propiedades de región, y estas no resultan en la segmentaron de un área, haciendo que los parámetros entregados por el programa sean un vector de ceros. Este caso puede ser medido en cada rama del árbol de decisión y ser empleado como un condicional para la creación de una nueva rama que de solución a estos fallos.

La cuarta estrategia consistió en emplear una comparación entre dos algoritmos de detección de bordes con distintos valores en la variable fudgefactor, con el fin de validar los resultados entre ellos y asegurar que los valores de las variables del vidrio entregados por cada algoritmo estén dentro de un umbral específico.

Sin embargo, se presenta una desventaja al emplear esta estrategia, en la medida que, si ambos métodos generan un error parecido en la segmentación, el sistema asume que la segmentación fue correcta, a pesar de que al comparar el resultado con la matriz de validación se expone que fue un error de segmentación.

La quinta y última estrategia se enfoca en el potencial de la red neuronal convolucional a mejorar con una base de datos más amplia y con un tiempo de entrenamiento mayor, y se refiere a la ubicación de redes neuronales en las ramas finales del árbol de decisión. Esto permite dar solución a las ventanas más complejas, donde ningún método anterior fue capaz de realizar una segmentación de vidrio correcta, empleando la red neuronal convolucional entrenada como último recurso.

Al mismo tiempo, si llega a ser necesario, el árbol de decisión permite acomodar varias redes neuronales convolucionales para dar solución a distintos tipos de casos que normalmente influirían entre sí dificultando la segmentación. Esto se puede realizar, tanto en las últimas ramas del árbol de decisión, como cualquier rama que se haya creado para separar un conjunto característico de ventanas de rascacielos. Por ejemplo, una de las ramas finales del árbol de decisión está enfocada en ventanas de rascacielos con reflejos de sol, lo que permite entrenar una red neuronal convolucional específicamente para realizar la segmentación en este tipo de ventanas e incluirla en dicha rama.

Finalmente, las cinco estrategias identificadas permiten optimizar el árbol de decisión para lograr categorizar algunos tipos de ventanas de rascacielos y darle solución a un mayor número de segmentaciones.

3.4.2 Árbol de decisión y diagrama del sistema multiagente

El sistema multiagente para la segmentación de vidrio en ventanas de rascacielos, en el cual se emplean las cinco estrategias mencionadas, se puede observar en la Figura 48. Cabe reiterar que los parámetros de vidrio son el vector final que entrega cada método, el cual contiene cuatro parámetros: la coordenada en X del centro del vidrio, la coordenada en Y del centro del vidrio, la altura del vidrio y el ancho del vidrio, todos estos en pixeles.

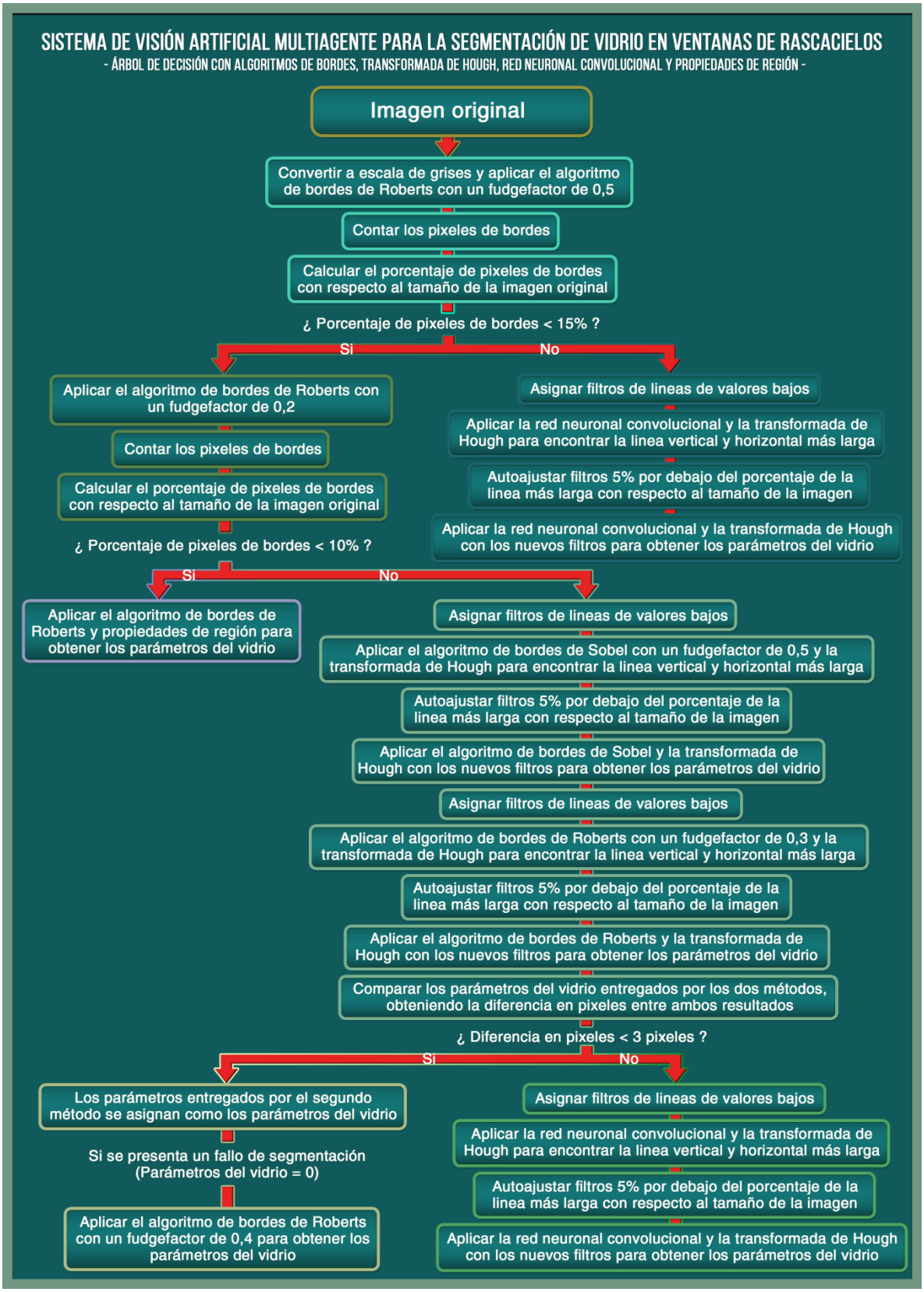

Figura 48. Sistema de visión artificial multiagente para la segmentación de vidrio en ventanas de rascacielos.

Fuente: Elaboración propia, 2019.

3.4.3 Adaptabilidad del sistema multiagente a distintas resoluciones

En el árbol de decisión se puede observar que se emplearon valores específicos en los porcentajes de los condicionales, los cuales fueron seleccionados con el tamaño de las imágenes de la base de datos en mente (90 x 90 pixeles). Lo mismo aplica a los filtros de líneas, cuyos valores asignados fueron entre 10% y 30% del tamaño de la imagen original. Cabe resaltar que se emplearon porcentajes en lugar de valores fijos en pixeles para permitir al sistema multiagente adaptarse a distintas resoluciones de imágenes y evitar su limitación a la resolución de la base de datos. Esta capacidad de adaptación se puede observar en la Figura 49.

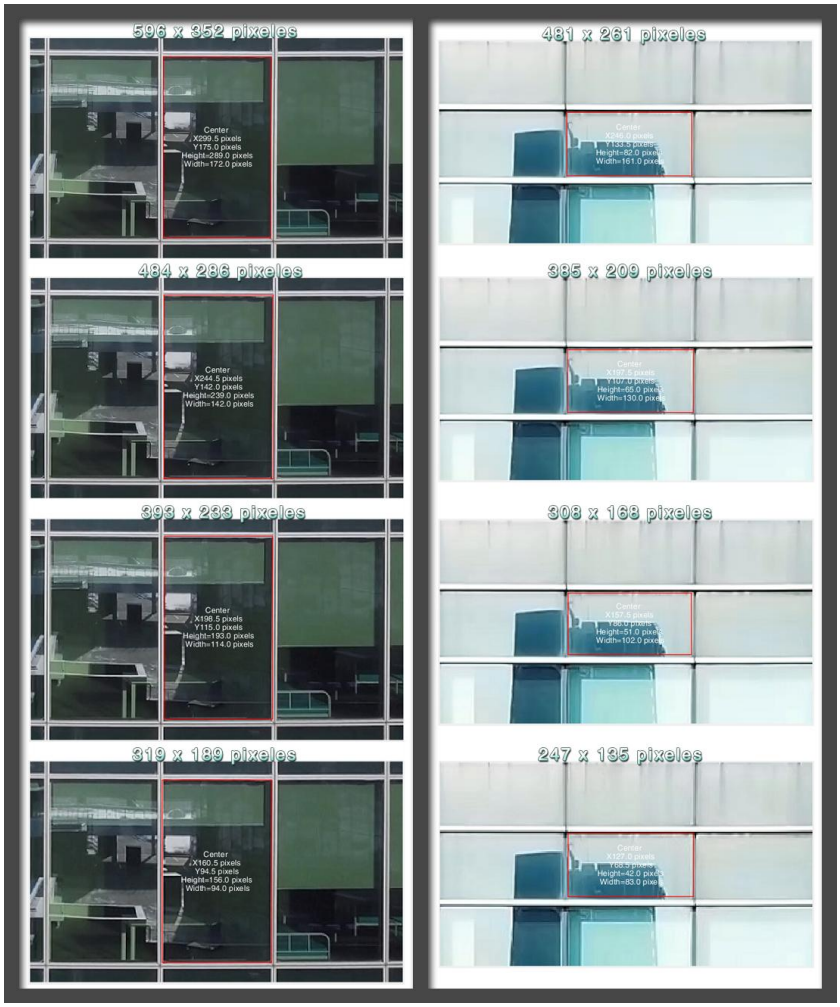

Figura 49. Resultados de segmentación al emplear el sistema de visión artificial multiagente en imágenes de distintas resoluciones.

Fuente: Elaboración propia, 2019.

Una limitante que se pudo observar al realizar pruebas con distintas resoluciones, radica en que para emplear la red neuronal convolucional, la imagen debe tener una resolución mínima correspondiente a la resolución de las imágenes empleadas en el entrenamiento de dicha red, es decir, que la resolución debe ser mayor a 90 pixeles de alto y 90 pixeles de ancho.

3.4.4 Resultados del sistema multiagente para la segmentación de vidrio en ventanas de rascacielos

Finalmente y al igual que con métodos anteriores, se realizó un programa para aplicar el sistema multiagente a todas las imágenes de la base de datos y comparar los resultados con la matriz de validación. Los resultados obtenidos al emplear el sistema multiagente para la segmentación de vidrio en ventanas de rascacielos se pueden apreciar en la Figura 50.

En los resultados se observa que se logró realizar una segmentación exitosa del vidrio en 300 de las 400 imágenes de la base de datos, es decir, que uno o más parámetros del vidrio tuvieron un error menor o igual a 5 pixeles. Cabe mencionar que el impacto de este error disminuye en imágenes de mayor resolución, debido principalmente a que este surge de las técnicas de dilatación de líneas y la detección de líneas contiguas en resoluciones de menor tamaño.

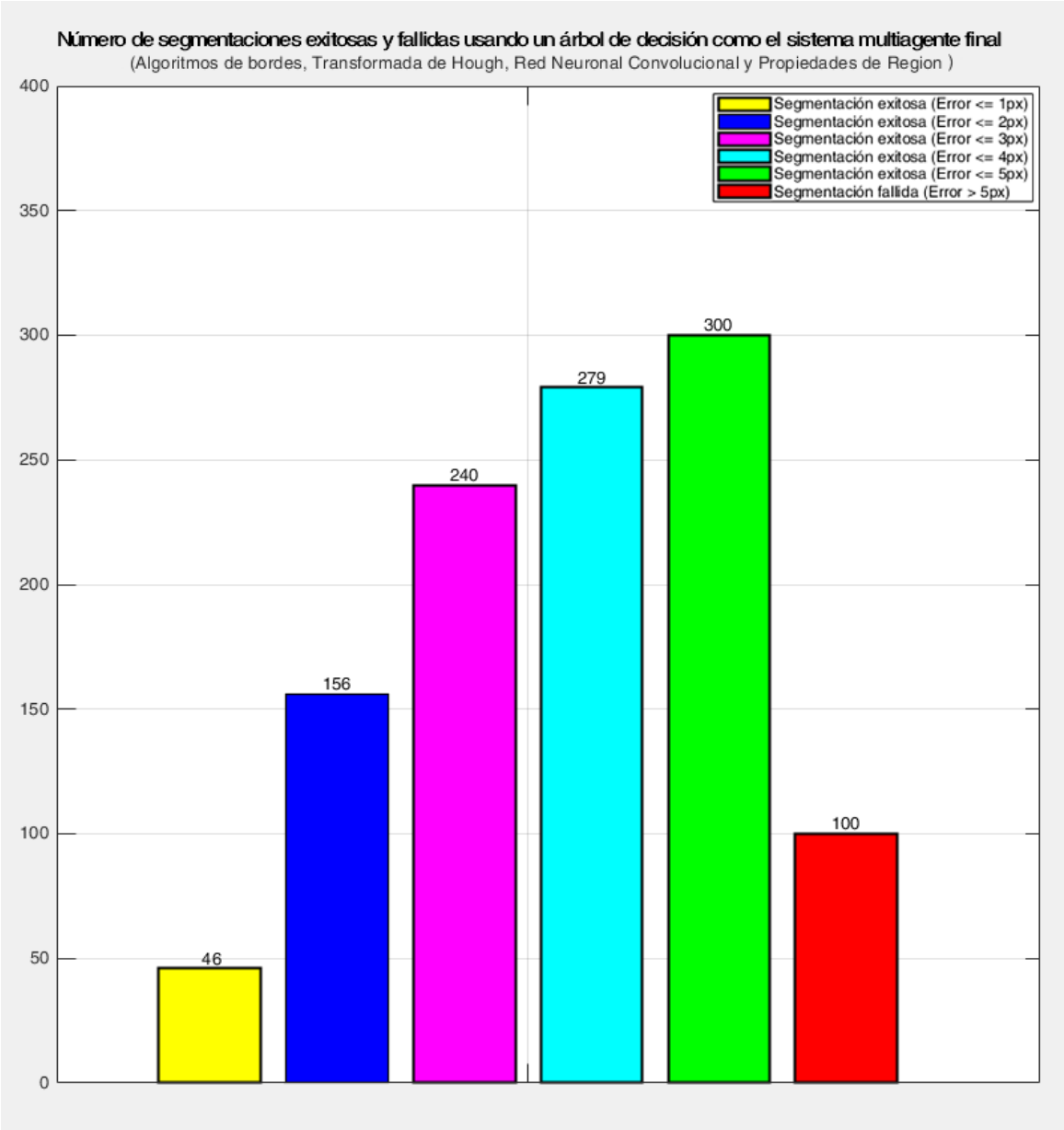

Figura 50. Resultados al emplear el sistema de visión artificial multiagente para segmentar el vidrio en las imágenes de la base de datos.

Fuente: Elaboración propia, 2019.

Por último, solamente luego de convertir los parámetros del vidrio segmentado a medidas reales y haber seleccionando la resolución de la cámara a emplear, es cuando se pueden realizar los ajustes finales y saber el impacto real de los errores de exactitud de la segmentación.

3.5 Conversión de pixeles a sistema métrico

A lo largo del desarrollo del sistema, la evaluación y validación de los métodos empleados fue realizada con los parámetros obtenidos al segmentar el vidrio de la ventana del rascacielos, los cuales han sido presentados en pixeles con el propósito de facilitar el desarrollo de dichos métodos y las pruebas efectuadas, sin embargo al implementar este tipo de sistemas se requiere que estos parámetros estén dados en medidas reales.

La posibilidad de la conversión de pixeles a sistema métrico fue evaluada antes de proceder con la implementación de la red neuronal convolucional, puesto que fue indispensable saber si la implementación del sistema era viable y si los requerimientos para esta conversión pudiesen ser logrados.

Determinar el tamaño de un objeto en una imagen puede ser logrado empleando técnicas de fotogrametría, las cuales permiten obtener información cuantificable a partir de fotos obtenidas por diferentes tipos de sensores y la fotointerpretación. En el caso del sistema desarrollado, esto se logra ya sea con dos cámaras en posiciones fijas conocidas; con una cámara calibrada tomando dos fotos con un movimiento de cámara conocido entre las dos imágenes; comparando el objetivo con otro objeto de tamaño conocido, o mediante una cámara calibrada y un sensor de distancia.

Por facilidad, usualmente se emplea un objeto de un tamaño conocido en la imagen, el cual al ser segmentado permite conocer tanto el tamaño real de dicho objeto como su tamaño en pixeles en la imagen. Con esta relación y el otro objeto estando en el mismo plano se puede conseguir el tamaño real del objetivo; no obstante, en el caso de las ventanas de rascacielos y los drones, es difícil tener un objeto fijo como referencia. Por esta razón se hizo énfasis en la definición de un método que, no solo facilitara hallar el tamaño real de la ventana del rascacielo, sino que lo efectuara de la forma más precisa posible, evitando inferir medidas, y considerando además que este está diseñado para ir montado sobre un dron.

Teniendo en cuenta el constante movimiento del dron al volar en una posición, además de las inclinaciones a las que se ve sometido durante el vuelo, se eligió utilizar una sola

cámara para facilitar la aplicación del sistema de visión artificial y un sensor de distancia para una medida más precisa del vidrio segmentado.

3.5.1 Ecuaciones para la conversión de pixeles a medidas reales

En el caso de la imagen de la ventana, esta se obtiene mediante una cámara, la cual proyecta una escena tridimensional en un sensor bidimensional mediante un lente (Figura 51). Por esta razón, la relación entre el tamaño real del objeto y el tamaño del objeto en el sensor de la cámara, es la misma que la relación entre la distancia al objeto y la distancia focal (Figura 52).

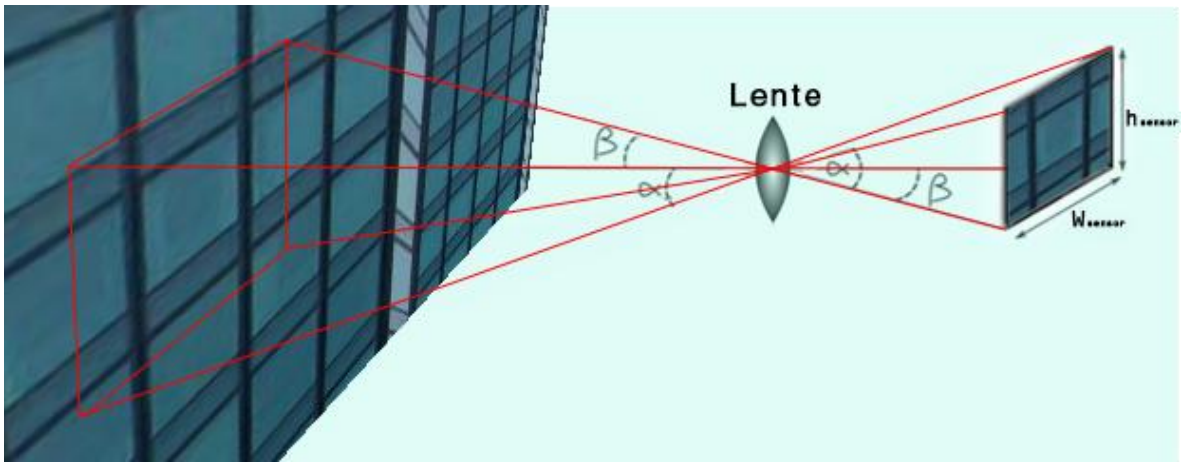

Figura 51. Imagen proyectada al sensor bidimensional de una cámara.
Fuente: Elaboración propia, 2019.

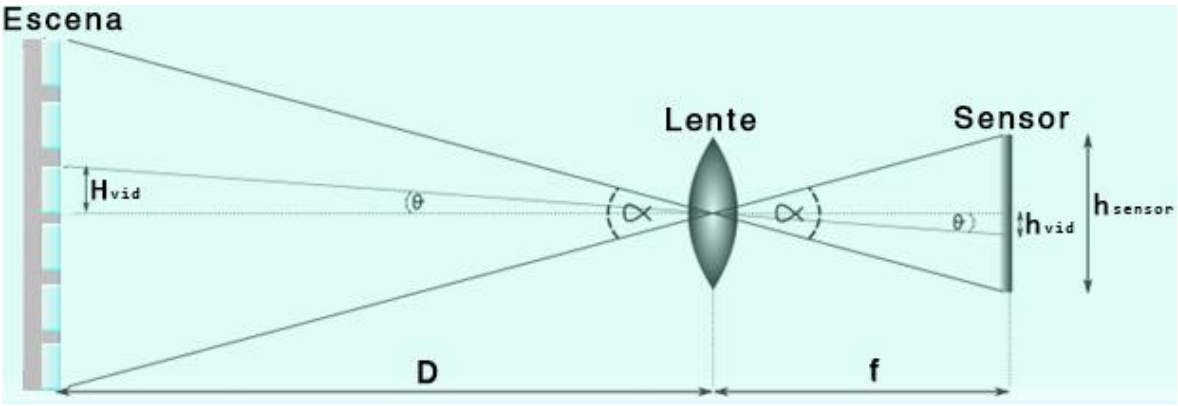

Figura 52. Visualización de la relación entre la altura real del objeto y la percibida por el sensor de la cámara.
Fuente: Elaboración propia, 2019.

A partir de lo anterior, se establecen las ecuaciones para el cálculo de la altura del vidrio, donde al expresar los ángulos se establece la siguiente relación:

$$D \cdot \theta = H_{vid} \quad (1)$$

$$f \cdot \theta = h_{vid} \quad (2)$$

A partir de (1) y (2) se obtiene:

$$H_{vid} = \frac{D}{f} \cdot h_{vid} \quad (3)$$

Donde f es la distancia focal del lente, D es la distancia entre el lente y el vidrio, h_{vid} es la altura del vidrio en el sensor y H_{vid} es la altura del vidrio. Sin embargo, como h_{vid} está expresado en pixeles y H_{vid} en el sistema métrico, es necesario hacer la siguiente conversión:

$$h_{vid}\,(mm) = \frac{h_{vid}\,(pixeles\ )}{h_{sensor}\ (pixeles\ )} \cdot h_{sensor}\ (mm) \quad (4)$$

Reemplazando (4) en (3) se obtiene la ecuación para encontrar la altura real del vidrio:

$$H_{vid} = D \cdot \frac{h_{vid}\,(pixeles\ )}{h_{sensor}\ (pixeles\ )} \cdot \frac{h_{sensor}\ (mm\,)}{f(mm\,)} \quad (5)$$

De la misma forma se obtiene la ecuación para calcular el ancho del vidrio:

$$W_{vid} = D \cdot \frac{w_{vid}\,(pixeles\ )}{w_{sensor}\ (pixeles\ )} \cdot \frac{w_{sensor}\ (mm\,)}{f(mm\,)} \quad (6)$$

Como se pudo observar al emplear este método, hallar el tamaño real del vidrio requiere variables como el tamaño del vidrio segmentado en pixeles, la distancia entre el lente de la cámara y el vidrio, y algunos parámetros intrínsecos de la cámara. El primer requerimiento se obtiene a partir del sistema multiagente desarrollado, el segundo empleando un sensor de distancia y el tercero mediante la calibración de la cámara o la hoja característica de la cámara.

El primer requerimiento fue logrado con anterioridad por medio del sistema multiagente para la segmentación de vidrio en ventanas de rascacielos, por lo que se procedió con la selección del sensor de distancia, mediante un proceso en el que fueron evaluados los sensores de distancia infrarrojos y los ultrasónicos.

3.5.2 Selección del sensor de distancia

En el caso de los sensores de distancia infrarrojos, estos suelen generar mediciones incorrectas en la detección de objetos transparentes o brillantes, como por ejemplo vidrios y espejos. Además, debido a que el objeto a detectar por el sistema es el vidrio, los sensores infrarrojos no lograban detectar el vidrio sino al objeto que se encontraba después de este. Por esta razón, estos tipos de sensores fueron descartados, y se enfocó completamente en los sensores de distancia ultrasónicos.

Luego de evaluar varios sensores potenciales, se seleccionaron los sensores de ultrasonido HC-SR04 y HRLV-MaxSonar-EZ4 (MB1043) como los potenciales a emplear, y cuyas características básicas se pueden observar en la Figura 53 y Figura 54, respectivamente. Las características prioritarias que se tuvieron en cuenta corresponden a la resolución reducida para que el error de la medición no impacte considerablemente la medición del tamaño del vidrio, y el rango de detección, por la distancia de seguridad durante el vuelo del dron.

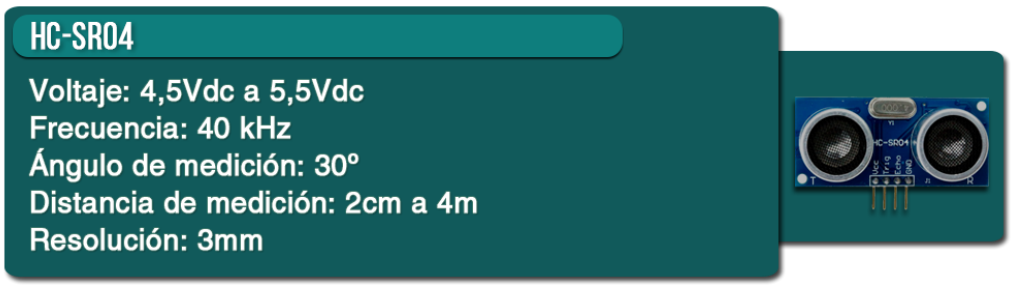

Figura 53. Características del sensor ultrasónico HC-SR04.

Fuente: Elaboración propia, 2019.

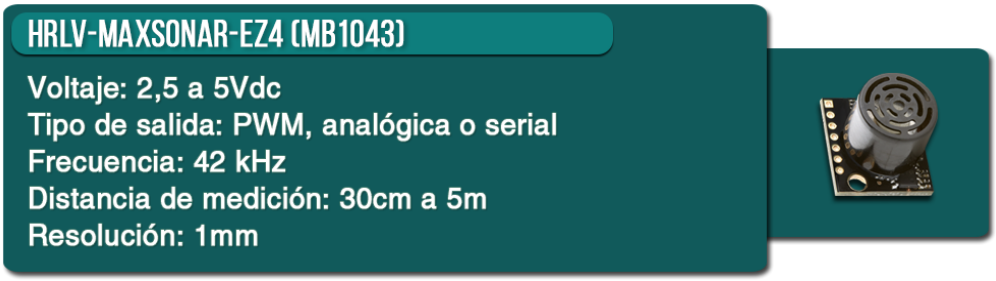

Figura 54. Características del sensor ultrasónico HRLV-MaxSonar-EZ4 (MB1043).
Fuente: Elaboración propia, 2019.

Otras características importantes que se tuvieron en cuenta, teniendo presente que los sensores deberán ser adaptados a un dron, fueron la tolerancia al ruido y la zona o arco de medición. Al respecto, se determinó que una zona más estrecha permite enfocar la medición en el vidrio en lugar de accidentalmente medir un marco que sobresalga, y la tolerancia al ruido permite que el ruido generado por el dron durante el vuelo no impacte la medición de la onda de sonido del sensor ultrasónico.

Considerando todas las características mencionadas anteriormente, se decidió emplear el sensor ultrasónico HRLV-MaxSonar-EZ4 (MB1043) para obtener la distancia del lente al vidrio y así lograr la conversión de pixeles a medidas reales.

3.5.3 Selección de la cámara

En cuanto a la cámara, se escogió una cámara web con parámetros intrínsecos conocidos, que fuese económica y se aseguró que esta no tuviese la opción de autoenfoque, con el fin de evitar la variación de la distancia focal durante su funcionamiento. Luego de una comparación de distintas cámaras, se seleccionó la webcam Logitech C270, la cual cuenta con una resolución máxima de 1280x720 pixeles, un tamaño de pixel de 2,8um, un sensor de 3,58mm de ancho por 2,02mm de alto y una distancia focal de 4,2mm.

3.5.4 Cálculo y resultados de la conversión de pixeles a medidas reales

Con todos los parámetros obtenidos para la conversión a medidas reales, se realizó una prueba inicial de la cámara web con una distancia fija hasta una ventana a escala cuyo

vidrio es de 13cm de alto por 13cm de ancho. Al ubicar la cámara a una distancia aproximada de 42,5cm, se tomó una captura de imagen y se aplicó el sistema de visión multiagente con código adicional para la conversión a medidas reales, luego de lo cual se obtuvo el resultado que se observa en la Figura 55.

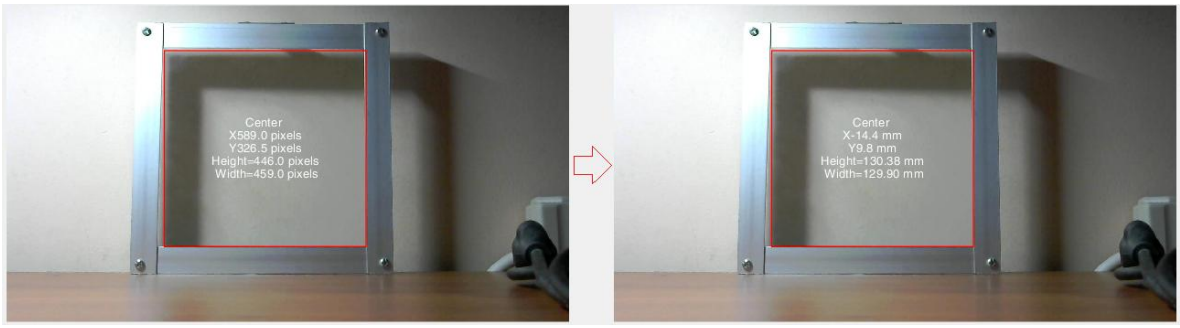

Figura 55. Prueba inicial para la conversión de pixeles a milímetros del vidrio segmentado.
Fuente: Elaboración propia, 2019.

A diferencia del centroide de los resultados en pixeles, el cual es medido a partir de la esquina superior izquierda, el centroide del vidrio en los resultados en medidas reales se calculó desde el centro de la imagen. Esto se realizó con el fin de conocer la ubicación del vidrio en el espacio a partir de la posición de la cámara.

Luego de que las pruebas demostraron que se está realizando una conversión correcta, se integró la cámara web, el sensor de distancia y el sistema multiagente desarrollado, para crear un sistema automatizado para el cálculo del tamaño del vidrio. Cabe resaltar que, en este momento del proceso, es importante tener presente la posición cero del sensor durante el montaje, es decir, el lugar físico desde donde las medidas entregadas por el sensor hacen la medición de la distancia (Figura 56).

Figura 56. Posición cero del sensor ultrasónico HRLV-MaxSonar-EZ4 (MB1043).
Fuente: Elaboración propia, 2019.

A partir de la ubicación y ajuste de todos los elementos en una posición fija, se llevó a cabo la conexión del sensor ultrasónico a una tarjeta Arduino Mega 2560 para efectuar la lectura de los datos y el envío de estos al sistema multiagente en Matlab.

Con el sistema conectado, se realizó una prueba final en una ventana de 63cm de ancho y 86cm de alto, adicionando de forma textual la distancia calculada por el sensor como la coordenada Z del centro del vidrio segmentado (Figura 57).

Figura 57. Prueba final para la conversión de pixeles a milímetros del vidrio segmentado.
Fuente: Elaboración propia, 2019.

En el caso de la última prueba, la altura presentó un error de exactitud menor a 1 mm, mientras que el ancho presentó un error de 12mm. Lo anterior puedo deberse a errores en la segmentación, al tamaño del vidrio, a la distancia a la que este está ubicado de la cámara o por distorsiones por la inclinación de la cámara. También se observó que, mientras más espacio ocupe la ventana en la foto y más cerca esté la ventana de la cámara, el error de la segmentación del vidrio será reducido.

Estas pruebas finales permitieron demostrar que es posible implementar el sistema multiagente para la segmentación de vidrio en ventanas de rascacielos desarrollado, y que los parámetros calculados y expresados en el sistema métrico, pueden ser empleados para un futuro posicionamiento de un brazo robótico sobre el vidrio o cualquier otra aplicación que requiere conocer el tamaño del vidrio y su posición en el espacio con respecto a la posición de la cámara.

CONCLUSIONES

El acelerado incremento en el número de rascacielos en construcción, así como la cantidad actual existente, exige el desarrollo de nuevas tecnologías con la capacidad de realizar la limpieza de sus ventanas de forma segura, rápida, confiable y precisa.

La fundamentación teórica permitió evidenciar las similitudes en el proceso de identificación de técnicas a ser empleadas en diversas aplicaciones de la visión artificial, así como las generalidades de su uso.

Los vidrios de las ventanas de rascacielos pueden ser reflectantes o permitir ver a través de ellos, lo cual significa que el vidrio tiene una falta de características propias, y lo que se observa son atributos de otros objetos. Sumado a esto, la variedad de condiciones físicas de los marcos de las ventanas, así como las condiciones externas de iluminación, evidencian la dificultad en la segmentación del vidrio.

En el proceso de segmentación del vidrio mediante la utilización de técnicas tradicionales de visión artificial, tales como los algoritmos de detección de bordes; en lugar de detectar el vidrio, es necesario enfocarse en las formas de detectar los marcos de las ventanas que lo rodean, en razón de la falta de características propias del vidrio.

Los algoritmos de detección de bordes de Sobel y Roberts presentaron los mejores resultados en la detección de los marcos y la consiguiente segmentación de vidrio, mientras que el algoritmo de Canny presentó los peores resultados, debido a que, a pesar de su sensibilidad a la detección de bordes y su tolerancia al ruido, su habilidad de detectar con mayor exactitud los reflejos y objetos que se ven a través del vidrio, dificulta la segmentación.

En vidrios con varios reflejos u objetos vistos a través, la cantidad de bordes detectados da como resultado la segmentación de varias áreas pequeñas en lugar del vidrio deseado, generando un fallo en la segmentación sin importar que operador o algoritmo de bordes se emplee. Con esto presente, la transformada de Hough demostró ser un método efectivo para detectar entre los bordes las líneas que componen el marco, mediante la utilización de sus filtros de tamaño y ángulo de líneas.

Las redes neuronales convolucionales y la segmentación semántica presentaron el mayor potencial para la segmentación de vidrio en ventanas de rascacielos, tanto por la exactitud de la segmentación, como por la cantidad de imágenes de la base de datos en las que se segmentó correctamente el vidrio; lo anterior, teniendo en consideración que la red no fue expuesta a la mitad de las imágenes de la base de datos en su entrenamiento.

En investigaciones con requerimientos y enfoques específicos, la validación de la red neuronal se debe llevar a cabo con los parámetros buscados, en lugar de validar directamente el resultado con el porcentaje de pixeles correctamente clasificados por la red. Esto se debe a que la exactitud del sistema que segmenta los parámetros buscados, difiere de la exactitud asociada a la clasificación de los pixeles en una red neuronal.

Se evidenció la insuficiencia de emplear un solo método para la segmentación confiable del vidrio en ventanas de rascacielos, exceptuando el caso en que se realice una posible mejora de la red neuronal convolucional. En ese sentido, se establecen cinco estrategias para integrar diferentes métodos en un árbol de decisión: conteo de pixeles de bordes, comparación de resultados entre dos métodos, posicionamiento estratégico de métodos en las ramas del árbol de decisión, filtro de líneas adaptable en la transformada de Hough y aprovechamiento de la condición de fallo cuando las propiedades de región no encuentran un área segmentada.

Siguiendo las ecuaciones matemáticas de la relación entre lo observado por la cámara, lo estimado por el sensor de distancia y los parámetros intrínsecos de la cámara, es posible realizar una conversión de pixeles a sistema métrico, que permita conocer con exactitud la posición real del objeto segmentado con respecto a la ubicación de la cámara.

RECOMENDACIONES

Los resultados expuestos por la transformada de Hough con filtros y parámetros ajustados de forma manual, indicaron el potencial que puede tener dicho método en caso de lograr encontrar una forma automática para ajustarlos a las diferentes necesidades de cada proyecto. Por lo tanto, para continuar en trabajos futuros se recomienda buscar formas de automatizar el ajuste de los parámetros y filtros, empleando diferentes técnicas de inteligencia artificial.

Es importante evaluar el efecto del incremento de la cantidad de imágenes pertenecientes a la base de datos, y el impacto de su uso en el entrenamiento de nuevas redes neuronales convolucionales. Al mismo tiempo, es conveniente realizar un análisis más detallado del impacto de los parámetros de entrenamiento en la exactitud de la segmentación y el progreso del entrenamiento de la red.

Se podría efectuar una comparación detallada entre distintas arquitecturas de redes neuronales como LeNet, la AlexNet, la ZF Net, la GoogLeNet, y la ResNet.

Continuando el desarrollo expuesto en la investigación, se sugiere la implementación del sistema de visión artificial en un dron, con el propósito de encontrar los ajustes adicionales necesarios para óptimo funcionamiento, así como de evaluar la compensación que se debe aplicar en el sensor de distancia y la cámara, debido a las inclinaciones y movimientos del dron.

REFERENCIAS BIBLIOGRÁFICAS

Aldwaik, M. & Adeli, H. (2014). Advances in optimization of highrise building structures. Structural and Multidisciplinary Optimization, 50(6), 899-919.

Antona Cortés, C. (2017). *Herramientas modernas en redes neuronales: la librería Keras* (Bachelor's thesis).

Aperador-Chaparro, W., Bautista-Ruiz, J. H., & Mejía, A. S. (2013). Determinacion por vision artificial del factor de degradacion en aleaciones biocompatibles. *Información tecnológica*, *24*(2), 109-120.

Azad, R. & Baghdadi, M. (2014). Novel and Fast Algorithm for Extracting License Plate Location Based on Edge Analysis. arXiv preprint arXiv:1407.6496.

Bautista, B. M., Medina, J. A. P., & Marín, F. J. S. (2012, July). Vision sens. In International Conference on Computers for Handicapped Persons (pp. 490-496). Springer, Berlin, Heidelberg.

Berkaya, S. K., Gunduz, H., Ozsen, O., Akinlar, C. & Gunal, S. (2016). On circular traffic sign detection and recognition. Expert Systems with Applications, 48, 67-75.

Bhunia, A. K., Kumar, G., Roy, P. P., Balasubramanian, R., & Pal, U. (2018). Text recognition in scene image and video frame using Color Channel selection. Multimedia Tools and Applications, 77(7), 8551-8578.

Cao, X., Wei, Y., Wen, F., & Sun, J. (2014). Face alignment by explicit shape regression. International Journal of Computer Vision, 107(2), 177-190.

Chung, C. L., Huang, K. J., Chen, S. Y., Lai, M. H., Chen, Y. C., & Kuo, Y. F. (2016). Detecting Bakanae disease in rice seedlings by machine vision. Computers and electronics in agriculture, 121, 404-411.

Constante, P., Gordon, A., Chang, O., Pruna, E., Acuna, F., & Escobar, I. (2016). Artificial Vision Techniques to Optimize Strawberry's Industrial Classification. IEEE Latin America Transactions, 14(6), 2576-2581.

Council on Tall Buildings and Urban Habitat (2018). Another Record-Breaker for Skyscraper Completions. Recuperado de: http://www.ctbuh.org/

Cubero, S., Lee, W. S., Aleixos, N., Albert, F., & Blasco, J. (2016). Automated systems based on machine vision for inspecting citrus fruits from the field to postharvest—a review. Food and Bioprocess Technology, 9(10), 1623-1639.

Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013, May). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 8609-8613). IEEE.

Diao, Z., Zhao, M., Song, Y., Wu, B., Wu, Y., Qian, X., & Wei, Y. (2015). Crop line recognition algorithm and realization in precision pesticide system based on machine vision. Transactions of the Chinese Society of Agricultural Engineering, 31(7), 47-52.

Diaz, L. E. N., & Arceo, L. E. C. (2018). Algoritmo rápido de la transformada de Hough para detección de líneas rectas en una imagen.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2014, January). Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning* (pp. 647-655).

Duque, J. P. U., & Ospina, E. (2004). IMPLEMENTACIÓN DE LA TRANSFORMADA DE HOUGH PARA LA DETECCIÓN DE LÍNEAS PARA UN SISTEMA DE VISIÓN DE BAJO NIVEL. *Scientia et technica*, *1*(24), 79-84.

Fahrurozi, A., Madenda, S., & Kerami, D. (2016). Wood Classification Based on Edge Detections and Texture Features Selection. *International Journal of Electrical & Computer Engineering (2088-8708)*, *6*(5).

Fausett, L. (1994). *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc..

Fischler, M. A., & Firschein, O. (Eds.). (2014). Readings in computer vision: issues, problem, principles, and paradigms. Elsevier.

Gongal, A., Silwal, A., Amatya, S., Karkee, M., Zhang, Q., & Lewis, K. (2016). Apple crop-load estimation with over-the-row machine vision system. Computers and Electronics in Agriculture, 120, 26-35.

Gonzalez, A., Bergasa, L. M., & Yebes, J. J. (2014). Text detection and recognition on traffic panels from street-level imagery using visual appearance. IEEE Transactions on Intelligent Transportation Systems, 15(1), 228-238.

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2004). Digital image processing using MATLAB (Vol. 624). Upper Saddle River, New Jersey: Pearson-Prentice-Hall.

Gonzalez, R. C., & Woods, R. E. (2008). Digital image processing: Pearson prentice hall. *Upper Saddle River, NJ*, *1*.

Guedes, M. C. & Cantuária, G. (2017). The Increasing Demand on High-Rise Buildings and Their History. In Sustainable High Rise Buildings in Urban Zones (pp. 93-102). Springer International Publishing.

Guil, N., Villalba, J., & Zapata, E. L. (1995). A fast Hough transform for segment detection. *IEEE Transactions on Image Processing*, *4*(11), 1541-1548.

Hernández-Hernández, J. L., García-Mateos, G., González-Esquiva, J. M., Escarabajal-Henarejos, D., Ruiz-Canales, A., & Molina-Martínez, J. M. (2016). Optimal color space selection method for plant/soil segmentation in agriculture. Computers and Electronics in Agriculture, 122, 124-132.

Isasi Viñuela, P., & Leon, G. (2004). Redes de neuronas artificiales: un enfoque práctico.

Jaramillo, M. A., Fernández, J. A., & de Salazar, E. M. (2010). Implementación del detector de bordes de Canny sobre redes neuronales celulares. *Universidad de Extremadura*.

Khan, S., Rahmani, H., Shah, S. A. A., & Bennamoun, M. (2018). A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, *8*(1), 1-207.

Korč, F., Förstner, W. (2009). eTRIMS Image Database for interpreting images of man-made scenes. Technical report, Department of Photogrammetry, University of Bonn.

Li, J. B., Huang, W. Q., & Zhao, C. J. (2015). Machine vision technology for detecting the external defects of fruits—a review. The Imaging Science Journal, 63(5), 241-251.

Lu, Z., & Zhang, L. (2016). Face recognition algorithm based on discriminative dictionary learning and sparse representation. Neurocomputing, 174, 749-755.

Meissner, M. (2017). Setting the Scene: Financial Spaces and Architectures. In Narrating the Global Financial Crisis (pp. 41-82). Springer International Publishing.

Moreira, G. A., & Sappa, A. (2015). Correspondencia Multiespectral en el espacio de HOUGH. *Proyecto de fin de Carrera, Escuela Superior Politécnica del Litoral, Ecuador*.

Mosquera, A. (2017). Limpieza de Ventanas de Rascacielos y Alternativas Tecnológicas Emergentes. Revista Colombiana de Tecnologías de Avanzada ISSN: 1692-7257 - Volumen 2 – Número 30. Universidad de Pamplona. Pamplona, (pp.109-118).

Neuhausen, M., Koch, C., & König, M. (2016). Image-based window detection: an overview.

Neuhausen, M., Martin, A., Obel, M., Mark, P., & König, M. (2017). A Cascaded Classifier Approach to Window Detection in Facade Images. In ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction (Vol. 34). Vilnius Gediminas Technical University, Department of Construction Economics & Property.

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision* (pp. 1520-1528).

Noroozi, M., Vinjimoor, A., Favaro, P., & Pirsiavash, H. (2018). Boosting self-supervised learning via knowledge transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 9359-9367).

Parkhi, O. M., Vedaldi, A., & Zisserman, A. (2015, September). Deep face recognition. In BMVC (Vol. 1, No. 3, p. 6).

Parmar, D. N., & Mehta, B. B. (2014). Face recognition methods & applications. arXiv preprint arXiv:1403.0485.

Patterson, J., & Gibson, A. (2017). *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.".

Pink, L. & Eickeler, S. (2016). Performance Enhancements for the Detection of Rectangular Traffic Signs. In Advanced Microsystems for Automotive Applications 2016 (pp. 113-123). Springer International Publishing.

Poppe, R. (2010). A survey on vision-based human action recognition. Image and vision computing, 28(6), 976-990.

Quiros, A. R. F., Abad, A., Bedruz, R. A., Uy, A. C., & Dadios, E. P. (2015, December). A genetic algorithm and artificial neural network-based approach for the machine vision of plate segmentation and character recognition. In Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2015 International Conference on (pp. 1-6). IEEE.

Ranft, B., & Stiller, C. (2016). The role of machine vision for intelligent vehicles. IEEE Transactions on Intelligent Vehicles, 1(1), 8-19.

Rebaza, J. V. (2007). Detección de bordes mediante el algoritmo de Canny. *Escuela Académico Profesional di Informática. Universidad Nacional de Trujillo*.

Romero, O. D., & Rolle, J. L. C. (2018). INTELIGENCIA ARTIFICIAL EN LA INGENIERÍA: PASADO, PRESENTE Y FUTURO. DYNA, 93(4), 350-352.

Sanabria, J. J., & Archila, J. F. (2011). Detección y análisis de movimiento usando visión artificial. Scientia et technica, 16(49).

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 815-823).

Scherer, D., Müller, A., & Behnke, S. (2010, September). Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks* (pp. 92-101). Springer, Berlin, Heidelberg.

Sonka, M., Hlavac, V., & Boyle, R. (2014). Image processing, analysis, and machine vision. Cengage Learning.

Timofte, R., Zimmermann, K., & Van Gool, L. (2014). Multi-view traffic sign detection, recognition, and 3D localisation. Machine vision and applications, 25(3), 633-647.

Yang, M. Y., Förstner, W., & Chai, D. (2012). Feature evaluation for building facade images-an empirical study. In International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences:[XXII ISPRS Congress, Technical Commission I] 39 (2012), Nr. B3 (Vol. 39, No. B3, pp. 513-518). Göttingen: Copernicus GmbH.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. In *Advances in neural information processing systems* (pp. 3320-3328).

Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

Zorrilla, V. M. S., Julián, F. G. C., Solano, M. Á. P., Reyes, M. V., & Calvo, E. R. (2016). DETECCIÓN DE BORDES DE UNA IMAGEN USANDO MATLAB. *Pistas Educativas*, *38*(122).

ANEXOS

ANEXO 1: Programa preliminar para la segmentación de vidrio.

```
function varargout = WCwindowdet(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @WCwindowdet_OpeningFcn, ...
                   'gui_OutputFcn',  @WCwindowdet_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WCwindowdet is made visible.
function WCwindowdet_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WCwindowdet (see VARARGIN)
% Choose default command line output for WCwindowdet
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes WCwindowdet wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = WCwindowdet_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in rundetection1.
%Rundetection1 uses Edges Algorithms + Hough Transform
function rundetection1_Callback(hObject, eventdata, handles)
% hObject    handle to rundetection1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%-----------Edge detection algorithm-------------------------------------
```

```
imname = get(handles.imageinput1,'String');% Reads the image name
fudgefactor = get(handles.edgeparam1,'String'); %Lower threshold range
for the edge detection algorithm
fudgefactor = str2num(fudgefactor);
switch get(handles.edgealgorithm1,'Value') % Reads the number of the
selected item in the popup menu
case 1; algname = 'Canny';
case 2; algname = 'Sobel';
case 3; algname = 'Prewitt';
case 4; algname = 'Roberts';
case 5; algname = 'Log';
end
hold off
I = imread(imname);
imrot1 = get(handles.imagerotate1,'String');
imrot1 = str2num(imrot1);
I = imrotate(I,imrot1);
axes(handles.axeswindow1)
imshow(I);
I  = rgb2gray(I);
axes(handles.axeswindow3)
imshow(I);
hold on
axes(handles.axeswindow4)
imshow(I);
hold on
[~, threshold] = edge(I,algname);
BW = edge(I,algname,threshold * fudgefactor); %BW =
edge(I,'Canny',threshold,sigma) %'Sobel' 'Prewitt' 'Roberts' 'log'
%BW = bwareaopen(BW,8,4); %Removes small objects from binary image
axes(handles.axeswindow2)
%--------------------------------------------------------------------
%--------------------Hough Transform-----------------------------------
%--------------------Vertical Lines------------------------------------
imshow(BW);
hold on
[H,T,R] = hough(BW,'RhoResolution',4,'Theta',-0.5:.1:0.5);
% Detect peaks
peaksnum = get(handles.houghparam1,'String');
peaksnum = str2num(peaksnum);
peaksth = get(handles.houghparam2,'String');
peaksth = str2num(peaksth);
P  = houghpeaks(H,peaksnum,'threshold',ceil(peaksth*max(H(:))));
%# Detect lines and overlay on top of image
axes(handles.axeswindow3)
linesfill = get(handles.houghparam3,'String');
linesfill = str2num(linesfill);
linesminlen = get(handles.houghparam4,'String');
linesminlen = str2num(linesminlen);
lines = houghlines(BW,T,R,P,'FillGap',linesfill,'MinLength',linesminlen);
[f,c]=size(BW); % Size of the original image
ImFin=zeros(f,c); % Creates a new matrix of zeros, the size of the
original image
axes(handles.axeswindow3)
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'g.-', 'LineWidth',1);
```

```
    %plot beginnings and ends of lines
    plot(xy(1,1),xy(1,2),'x','LineWidth',1,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',1,'Color','red');
    %-------------Draws a line on the zeros matrix----------------------
    x=[xy(1,1) xy(2,1)];
    y=[xy(1,2) xy(2,2)];
    nPoints = max(abs(diff(x)), abs(diff(y)))+1;  % Number of points in
line
    rIndex = round(linspace(y(1), y(2), nPoints));  % Row indices
    cIndex = round(linspace(x(1), x(2), nPoints));  % Column indices
    index = sub2ind(size(ImFin), rIndex, cIndex);     % Linear indices
    ImFin(index) = 255; % Zeros matrix with lines drawn on it
end
%--------------------Horizontal Lines-----------------------------------
[H,T,R] = hough(BW,'RhoResolution',4,'Theta',-90:.1:-89.5); %
hold on
% Detect peaks
peaksnum = get(handles.houghparam1,'String');
peaksnum = str2num(peaksnum);
peaksth = get(handles.houghparam2,'String');
peaksth = str2num(peaksth);
P = houghpeaks(H,peaksnum,'threshold',ceil(peaksth*max(H(:))));
% Detect lines and overlay on top of image
axes(handles.axeswindow3)
linesfill = get(handles.houghparam3,'String');
linesfill = str2num(linesfill);
linesminlen = get(handles.houghparam4,'String');
linesminlen = str2num(linesminlen);
lines = houghlines(BW,T,R,P,'FillGap',linesfill,'MinLength',linesminlen);
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'g.-', 'LineWidth',1);
    %plot beginnings and ends of lines
    plot(xy(1,1),xy(1,2),'x','LineWidth',1,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',1,'Color','red');
    %-------------Draws a line on the zeros matrix----------------------
    x=[xy(1,1) xy(2,1)];
    y=[xy(1,2) xy(2,2)];
    nPoints = max(abs(diff(x)), abs(diff(y)))+1;  % Number of points in
line
    rIndex = round(linspace(y(1), y(2), nPoints));  % Row indices
    cIndex = round(linspace(x(1), x(2), nPoints));  % Column indices
    index = sub2ind(size(ImFin), rIndex, cIndex);     % Linear indices
    ImFin(index) = 255;% Zeros matrix with lines drawn on it
end
%--------------------Horizontal Lines2----------------------------------
[H,T,R] = hough(BW,'RhoResolution',4,'Theta',89.5:.1:89.9); %
hold on
% Detect peaks
peaksnum = get(handles.houghparam1,'String');
peaksnum = str2num(peaksnum);
peaksth = get(handles.houghparam2,'String');
peaksth = str2num(peaksth);
P = houghpeaks(H,peaksnum,'threshold',ceil(peaksth*max(H(:))));
% Detect lines and overlay on top of image
axes(handles.axeswindow3)
linesfill = get(handles.houghparam3,'String');
```

```
linesfill = str2num(linesfill);
linesminlen = get(handles.houghparam4,'String');
linesminlen = str2num(linesminlen);
lines = houghlines(BW,T,R,P,'FillGap',linesfill,'MinLength',linesminlen);
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1), xy(:,2), 'g.-', 'LineWidth',1);
    %plot beginnings and ends of lines
    plot(xy(1,1),xy(1,2),'x','LineWidth',1,'Color','yellow');
    plot(xy(2,1),xy(2,2),'x','LineWidth',1,'Color','red');
    %-------------Draws a line on the zeros matrix-----------------------
    x=[xy(1,1) xy(2,1)];
    y=[xy(1,2) xy(2,2)];
    nPoints = max(abs(diff(x)), abs(diff(y)))+1;  % Number of points in
line
    rIndex = round(linspace(y(1), y(2), nPoints));  % Row indices
    cIndex = round(linspace(x(1), x(2), nPoints));  % Column indices
    index = sub2ind(size(ImFin), rIndex, cIndex);     % Linear indices
    ImFin(index) = 255;% Zeros matrix with lines drawn on it
end
%--------------------Line dilate-------------------------------------------
linelen = get(handles.regionparam1,'String');
linelen = str2num(linelen);
ste90 = strel('line', linelen, 90);
ste0 = strel('line', linelen, 0);
BWdil = imdilate(ImFin, [ste90 ste0]);
%------------------Imcomplement image-----------------------------------
BWim = imbinarize(BWdil);
BWim = imcomplement(BWim);
if get(handles.regionparam5,'Value') == 1
BWim = imclearborder(BWim); %clean if touching the border
end
%-------------------------Area filter-------------------------------------
maxarnum = get(handles.regionparam2,'String');
maxarnum = str2num(maxarnum);
BW = bwareafilt(BWim, maxarnum);%filter the max number of regionprops
areas to find
axes(handles.axeswindow4)
L=bwlabel(BW);
prop=regionprops(L,'all'); %Funcion para analisis de regiones en matlab
%--------------------Filter detected areas by area %-----------------
withinfilter = get(handles.regionparam3,'String');
withinfilter = str2num(withinfilter);
withinfilter = withinfilter/100; %from percent to decimal
maxarea = max( [prop.Area] ); %finds the biggest area found
sizeim=maxarea+(maxarea*withinfilter); % biggest area + withinfilter
percentage of it, to detect more areas within that range
sizeim2=maxarea-(maxarea*withinfilter);% biggest area - withinfilter
percentage of it, to detect more areas within that range
idx = find([prop.Area] > sizeim2 & [prop.Area] < sizeim); %Finds the
areas within the range
L = ismember(L,idx);
%-------Utilizacion de "Extrema" para encontrar el número de bordes------
S = regionprops(L,'Extrema','Centroid');
S2 = regionprops(L,'BoundingBox','Area');
se = strel('disk',5);
axes(handles.axeswindow4)
```

```
hold on
for j = 1:length(S)
temp = round(S(j).Extrema);
temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
mask = zeros(max(temp(:,1)),max(temp(:,2)));
for k = 1:8
mask(temp(k,1),temp(k,2))=1;
end
mask2 = imdilate(mask,se);
[~,numbordes(j)] = bwlabel(mask2,8);
end
%--------------------Boundingbox height width and centroid-------------
bb = regionprops(L,'BoundingBox');
bbMatrix = vertcat(bb(:).BoundingBox); %matrix of the boundingbox [Left,
Top, Width, Height]
nodetect=isempty(bbMatrix);
if nodetect==0
width=bbMatrix(:,3);
height=bbMatrix(:,4);
centerx=bbMatrix(:,1)+ bbMatrix(:,3)/2;
centery=bbMatrix(:,2)+ bbMatrix(:,4)/2;
end
%-------------------------------------------------------------------------
%-----Closest region to the center, in case more than 1 window not
touching the border enters the photo frame and at the same time they are
all the same size (tall windows case).
[rows,columns]=size(BW);
centerimagex=round(columns/2); %center of the image, pixel position in X
coordinate
centerimagey=round(rows/2); %center of the image, pixel position in Y
coordinate
distance_cenx=zeros(1,j);
distance_ceny=zeros(1,j);
for g=1:j
distance_cenx(g) = abs(centerimagex - centerx(g)); %how close each area
is to the center of the image, in x coordinate
distance_ceny(g) = abs(centerimagey - centery(g)); %how close each area
is to the center of the image, in y coordinate
end
if nodetect==0
distance_cen=distance_cenx+distance_ceny;
[min_val,min_pos]=min(distance_cen);
%-------------------------------------------------------------------------
%-------Only show and place text on the area closest to the center
temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
pixels',centerx(min_pos),centery(min_pos),height(min_pos),width(min_pos))
;
text(centerx(min_pos),centery(min_pos),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
WindowData = [centerx(min_pos), centery(min_pos), height(min_pos),
width(min_pos)];
bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-WindowData(3)/2
WindowData(4) WindowData(3)]; %bounding_box=[centerx-width/2 centery-
height/2 width height];
rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
```

```
else
%sprintf ('Glass area not detected, segmentation failed')
WindowData = [0 0 0 0];
end
num = sscanf(imname,'w%f'); %Gets the number of the image, used to access
the validation matrix
load('ValidationMatrix.mat'); %Loads the validation matrix created with
the data used for the pixellabels
Correct=ValidationMatrix(num,:);
Error=[abs(WindowData(1)-Correct(1)) abs(WindowData(2)-Correct(2))
abs(WindowData(3)-Correct(3)) abs(WindowData(4)-Correct(4))]
%--------------------------------------------------------------------------
%--------------------------------------------------------------------------

% --- Executes on button press in rundetection2.
function rundetection2_Callback(hObject, eventdata, handles)
% hObject    handle to rundetection2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%-----------Edge detection algorithm-----------------------------------
imname2 = get(handles.imageinput2,'String');% Reads the image name
fudgefactor = get(handles.edgeparam2,'String'); %Lower threshold range
for the edge detection algorithm
fudgefactor = str2num(fudgefactor);
switch get(handles.edgealgorithm2,'Value') % Reads the number of the
selected item in the popup menu
case 1; algname = 'Canny';
case 2; algname = 'Sobel';
case 3; algname = 'Prewitt';
case 4; algname = 'Roberts';
case 5; algname = 'Log';
end
hold off
I2 = imread(imname2);
imrot2 = get(handles.imagerotate2,'String');
imrot2 = str2num(imrot2);
I2 = imrotate(I2,imrot2);
axes(handles.axeswindow5)
imshow(I2);
I2  = rgb2gray(I2);
sizei=size(I2);
axes(handles.axeswindow8)
imshow(I2);
hold on
[~, threshold] = edge(I2,algname);
BW = edge(I2,algname,threshold * fudgefactor); %BW =
axes(handles.axeswindow6)
imshow(BW);
%--------------------Line dilate-----------------------------------------
linelen = get(handles.regionparam1,'String');
linelen = str2num(linelen);
ste90 = strel('line', linelen, 90);
ste0 = strel('line', linelen, 0);
BWdil = imdilate(BW, [ste90 ste0]);
%------------------Imcomplement image----------------------------------
BWim = imcomplement(BWdil);
if get(handles.regionparam5,'Value') == 1
```

```
BWim = imclearborder(BWim); %clean if touching the border
end
%-------------------------Area filter-----------------------------------
maxarnum = get(handles.regionparam2,'String');
maxarnum = str2num(maxarnum);
BW = bwareafilt(BWim, maxarnum);%filter the max number of regionprops
areas to find
axes(handles.axeswindow7)
imshow(BW);
L=bwlabel(BW);
prop=regionprops(L,'all'); %Funcion para analisis de regiones en matlab
%--------------------Filter detected areas by area %-----------------
withinfilter = get(handles.regionparam3,'String');
withinfilter = str2num(withinfilter);
withinfilter = withinfilter/100; %from percent to decimal
maxarea = max( [prop.Area] ); %finds the biggest area found
sizeim=maxarea+(maxarea*withinfilter); % biggest area + withinfilter
percentage of it, to detect more areas within that range
sizeim2=maxarea-(maxarea*withinfilter);% biggest area - withinfilter
percentage of it, to detect more areas within that range
idx = find([prop.Area] > sizeim2 & [prop.Area] < sizeim); %Finds the
areas within the range
L = ismember(L,idx);
%-------Utilizacion de "Extrema" para encontrar el número de bordes------
S = regionprops(L,'Extrema','Centroid');
S2 = regionprops(L,'BoundingBox','Area');
se = strel('disk',5);
axes(handles.axeswindow8)
for j = 1:length(S)
temp = round(S(j).Extrema);
temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
mask = zeros(max(temp(:,1)),max(temp(:,2)));
for k = 1:8
mask(temp(k,1),temp(k,2))=1;
end
mask2 = imdilate(mask,se);
[~,numbordes(j)] = bwlabel(mask2,8);
end
%---------------------Boundingbox height width and centroid--------------
bb = regionprops(L,'BoundingBox');
bbMatrix = vertcat(bb(:).BoundingBox); %matrix of the boundingbox [Left,
Top, Width, Height]
nodetect=isempty(bbMatrix);
if nodetect==0
width=bbMatrix(:,3);
height=bbMatrix(:,4);
centerx=bbMatrix(:,1)+ bbMatrix(:,3)/2;
centery=bbMatrix(:,2)+ bbMatrix(:,4)/2;
end
%-------------------------------------------------------------------------
%-----Closest region to the center, in case more than 1 window not
touching the border enters the photo frame and at the same time they are
all the same size (tall windows case).
[rows,columns]=size(BW);
centerimagex=round(columns/2); %center of the image, pixel position in X
coordinate
```

```
centerimagey=round(rows/2); %center of the image, pixel position in Y
coordinate
distance_cenx=zeros(1,j);
distance_ceny=zeros(1,j);
for g=1:j
distance_cenx(g) = abs(centerimagex - centerx(g)); %how close each area
is to the center of the image, in x coordinate
distance_ceny(g) = abs(centerimagey - centery(g)); %how close each area
is to the center of the image, in y coordinate
end
if nodetect==0
distance_cen=distance_cenx+distance_ceny;
[min_val,min_pos]=min(distance_cen);
%-------------------------------------------------------------------------
%-------Only show and place text on the area closest to the center
temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
pixels',centerx(min_pos),centery(min_pos),height(min_pos),width(min_pos))
;
text(centerx(min_pos),centery(min_pos),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
WindowData = [centerx(min_pos), centery(min_pos), height(min_pos),
width(min_pos)];
bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-WindowData(3)/2
WindowData(4) WindowData(3)]; %bounding_box=[centerx-width/2 centery-
height/2 width height];
rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
else
%sprintf ('Glass area not detected, segmentation failed')
WindowData = [0 0 0 0];
end
%-------------------------------------------------------------------------
```

ANEXO 2: Entrenamiento red neuronal convolucional

```
%----------Load the pretrained network for transfer learning-------------
net = vgg16; %load the pretrained VGG-16 network. The output net is a
SeriesNetwork object
net.Layers %View the network architecture. The network has 41 layers. 16
layers with learnable weights: 13 convolutional layers, and 3 fully
connected layers.
%------Load this project's Image Dataset and labels----------------------
imgDir = fullfile('C:\Users\Artur\ImageDataset'); %Location of all the
images
imds = imageDatastore(imgDir); %The imageDatastore enables you to
efficiently load a large collection of images on disk.
I = readimage(imds,1); imshow(I) %Check the first image to make sure it
added the images properly
load('C:\Users\Artur\Image Labeling\gTruth.mat')%load the gTruth.mat file
created by image labeling session
pxds = pixelLabelDatastore(gTruth);
gTruth.LabelDefinitions %View the definitions
%-------Analyze Dataset Statistics------------
tbl = countEachLabel(pxds)% Counts the number of pixels by class label
classes = tbl.Name;
```

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);
bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
%Ideally, all classes would have an equal number of observations.
%A common issue in many datasets are the diference in number of
observations per label.
% In the case of glass segmentation in skyscrapers, such scenes have more
glass than frame or wall pixels because it covers more area in the image.
%If not handled correctly, this imbalance can be detrimental to the
learning process because the learning is biased in favor of the dominant
classes. Class weighting is used to handle this issue.
%--------Create the Network--------
imageSize = [90 90 3];
numClasses = numel(classes);
lgraph = segnetLayers(imageSize,numClasses,'vgg16')
%----------Balance Classes Using Class Weighting----------------
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq;
pxLayer =
pixelClassificationLayer('Name','labels','ClassNames',tbl.Name,'ClassWeig
hts',classWeights)
lgraph = removeLayers(lgraph,'pixelLabels'); %Update the SegNet layer
lgraph = addLayers(lgraph, pxLayer);
lgraph = connectLayers(lgraph,'softmax','labels');
plot(lgraph)
%-----Select Training Options---------
options = trainingOptions('sgdm', ...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.0005, ...
    'MaxEpochs',120, ...
    'MiniBatchSize',1, ...
    'Shuffle','every-epoch', ...
    'ExecutionEnvironment','auto', ...
    'VerboseFrequency',20,...
    'Plots','training-progress');
%------Data Augmentation--------
augmenter =
imageDataAugmenter('RandXReflection',true,'RandXTranslation',[-10
10],'RandYTranslation',[-10 10]);
pximds = pixelLabelImageDatastore(imds,pxds,
'DataAugmentation',augmenter);
%---------Train Network----------
[net, info] = trainNetwork(pximds,lgraph,options);
arturnet4 = net;
save arturnet4
%% Run the net on all the images
load arturnet4.mat
for i=001:400
imname=sprintf('w%03d.jpg',i);
I = imread(imname);
C = semanticseg(I, arturnet4);
cmap = jet(numel(classes));
B = labeloverlay(I,C,'Colormap',cmap,'Transparency',0.5);
```

```
imshow(B)
h=sprintf('w%03dCNN4.jpg',i);
imwrite(B,h);
end
expectedResult = read(pxds);
actual = uint8(C);
expected = uint8(expectedResult);
imshowpair(actual, expected,'montage')
%% loop and save
F = figure (1);
for i=001:400
imname=sprintf('w%03d.jpg',i);
I = imread(imname);
C = semanticseg(I, arturnet4);
actual = uint8(C);
imshowpair(I, actual,'montage')
hold on
set(gcf, 'InvertHardCopy', 'off');
saveas(F,'test.jpg');
imagen = imread('test.jpg');
cortado = imcrop(imagen,[125 35 290 150]);
imshow(cortado)
h=sprintf('w%03dCNN4Montage.jpg',i);
imwrite(cortado,h);
end
```

ANEXO 3: Creación de la matriz de validación

```
%% Load imagedatastore and the label definition created
imgDir = fullfile('C:\Users\Artur\Image Labeling\ImageDataset');
%Location of all the images
imds = imageDatastore(imgDir); %The imageDatastore enables you to
efficiently load a large collection of images on disk.
I = readimage(imds,400); imshow(I) %Check the first image to make sure it
added the images properly
load('C:\Users\Artur\Image Labeling\gTruth.mat')%load the gTruth.mat file
created by image labeling session
pxds = pixelLabelDatastore(gTruth);
%%
for i=1:400
I = readimage(imds,i);
C = readimage(pxds,i);
B = labeloverlay(I,C, 'Transparency',0);
B = im2bw(B);
imshow(B)
%hold on
[rows,columns]=size(B);
BWim = imcomplement(B);
BWim = imclearborder(BWim);
maxarnum = 10;
BW = bwareafilt(BWim, maxarnum);
L=bwlabel(BW);
prop=regionprops(L,'all');
%--------------------Filter detected areas by area %-----------------
maxarea = max( [prop.Area] ); %finds the biggest area found
```

```
sizeim=maxarea+(maxarea*0.3); % biggest area + 20% of it, to detect more
areas within that range
sizeim2=maxarea-(maxarea*0.3);% biggest area - 20% of it, to detect more
areas within that range
idx = find([prop.Area] > sizeim2 & [prop.Area] < sizeim); %Finds the
areas within the range
L = ismember(L,idx);
%-------Utilizacion de "Extrema" para encontrar el numero de bordes------
S = regionprops(L,'Extrema','Centroid');
S2 = regionprops(L,'BoundingBox','Area');
se = strel('disk',5);
for j = 1:length(S)
temp = round(S(j).Extrema);
temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
mask = zeros(max(temp(:,1)),max(temp(:,2)));
for k = 1:8
mask(temp(k,1),temp(k,2))=1;
end
mask2 = imdilate(mask,se);
[~,numbordes(j)] = bwlabel(mask2,8);
end
%---------------------Boundingbox height, width and centroid-------------
bb = regionprops(L,'BoundingBox');
bbMatrix = vertcat(bb(:).BoundingBox); %matrix of the boundingbox [Left,
Top, Width, Height]
nodetect=isempty(bbMatrix);
if nodetect==0 % 1 if empty, 0 if not empty, avoids an empty matrix crash
width=bbMatrix(:,3);
height=bbMatrix(:,4);
centerx=bbMatrix(:,1)+ bbMatrix(:,3)/2;
centery=bbMatrix(:,2)+ bbMatrix(:,4)/2;
end
%-------------------------------------------------------------------------
%-----Closest region to the center, in case more than 1 window not
touching the border enters the photo frame and at the same time they are
all the same size (tall windows case).
centerimagex=round(columns/2); %center of the image, pixel position in X
coordinate
centerimagey=round(rows/2); %center of the image, pixel position in Y
coordinate
distance_cenx=zeros(1,j);
distance_ceny=zeros(1,j);
for g=1:j
distance_cenx(g) = abs(centerimagex - centerx(g)); %how close each area
is to the center of the image, in x coordinate
distance_ceny(g) = abs(centerimagey - centery(g)); %how close each area
is to the center of the image, in y coordinate
end
if nodetect==0
distance_cen=distance_cenx+distance_ceny;
[min_val,min_pos]=min(distance_cen);
%-------------------------------------------------------------------------
%-------Only show and place text on the area closest to the center
% temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
```

```
pixels',centerx(min_pos),centery(min_pos),height(min_pos),width(min_pos))
;
% text(centerx(min_pos),centery(min_pos),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
%--------------------------------------------------------------------------
%---------------Data for the machine vision program----------------------
WindowData = [centerx(min_pos), centery(min_pos), height(min_pos),
width(min_pos)];
bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-WindowData(3)/2
WindowData(4) WindowData(3)]; %bounding_box=[centerx-width/2 centery-
height/2 width height];
rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
else
%sprintf ('Glass area not detected, segmentation failed')
WindowData = 0;
end
WindowData;
% Creates a validation matrix where each array is the correct segmented
% glass of the image on the database [centerx cetery height width]
ValidationMatrix (i,1) = WindowData (1)
ValidationMatrix (i,2) = WindowData (2)
ValidationMatrix (i,3) = WindowData (3)
ValidationMatrix (i,4) = WindowData (4)
end
%% Saving the validation matrix for later use
save('ValidationMatrix.mat','ValidationMatrix');
```

ANEXO 4: Programa final, sistema multiagente para la segmentación de vidrio y conversión de pixeles a sistema métrico.

```
%% Load all the convolutional neural networks and the validation matrix
load arturnet4.mat
load('ValidationMatrix.mat');
%% Initializa camera
info = imaqhwinfo('winvideo')
info.DeviceInfo.SupportedFormats
cam = videoinput('winvideo',1,'YUY2_1280x720');
preview(cam)
set(cam,'ReturnedColorSpace', 'RGB');
frame = getsnapshot(cam);
imshow(frame);
%% Guardar la imagen
baseFileName = 'w503.jpg';
someFolder='D:/Artur';
fullFileName = fullfile(someFolder, baseFileName);
imwrite(frame, fullFileName);
%% Aplicar el arbol de decision
counter = 601;
im_name=sprintf('w%03d.jpg',counter);
im_original = imread(im_name);
%subplot (1,2,1); imshow(im_original);
im_number = sscanf(im_name,'w%f');
rotation = 0; %Rotate the image
im_rot = imrotate(im_original,rotation);
im_grayscale = rgb2gray(im_rot); % Grayscale image
```

```
%im_grayscale = imguidedfilter(im_grayscale); % Guided filtering
[rows,columns]=size(im_grayscale); % Size of the original image
%-----------Edge detection algorithm--------------------------------------
fudgefactor=0.5; algname= 'Roberts';
[~, threshold] = edge(im_grayscale,algname); %Best threshold for the
image with the specific algorithm
im_edges = edge(im_grayscale,algname,threshold * fudgefactor); %BW =
edge(I,'Canny',threshold,sigma) %'Sobel' 'Prewitt' 'Roberts' 'log'
%subplot (1,3,1); imshow(im_edges);
numPixels = numel(im_edges);
numPixelsEdges = sum(im_edges(:));
percPixelsEdges = numPixelsEdges * 100 / numPixels;
%Conditional and testing
if percPixelsEdges < 15
    fudgefactor=0.2; algname= 'Roberts';
    [~, threshold] = edge(im_grayscale,algname); %Best threshold for the
image with the specific algorithm
    im_edges = edge(im_grayscale,algname,threshold * fudgefactor); %BW =
edge(I,'Canny',threshold,sigma) %'Sobel' 'Prewitt' 'Roberts' 'log'
    numPixels = numel(im_edges);
    numPixelsEdges = sum(im_edges(:));
    percPixelsEdges = numPixelsEdges * 100 / numPixels;
    if percPixelsEdges < 10
        [WindowData]=DetectWindowEdges (im_original, fudgefactor,
algname);
        if  sum(WindowData) == 0
        aFail(1)=aFail(1)+1;
        end
    else
    fillvert_perc=20; minlenvert_perc=10; fillhor_perc=20;
minlenhor_perc=10;
    fudgefactor=0.5; algname= 'Sobel';
    [~, max_len_horizontal_perc, max_len_vertical_perc]=DetectWindowHough
(im_original, fudgefactor, algname, fillvert_perc, minlenvert_perc,
fillhor_perc, minlenhor_perc);
    minlenvert_perc=max_len_vertical_perc-5;
minlenhor_perc=max_len_horizontal_perc-5; % Longest line found - 2%, used
to filter small lines
    [WindowData, ~, ~]=DetectWindowHough (im_original, fudgefactor,
algname, fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
    TempWinData1=WindowData;
    fillvert_perc=10; minlenvert_perc=10; fillhor_perc=10;
minlenhor_perc=10;
    fudgefactor=0.3; algname= 'Roberts';
    [~, max_len_horizontal_perc, max_len_vertical_perc]=DetectWindowHough
(im_original, fudgefactor, algname, fillvert_perc, minlenvert_perc,
fillhor_perc, minlenhor_perc);
    minlenvert_perc=max_len_vertical_perc-5;
minlenhor_perc=max_len_horizontal_perc-5; % Longest line found - 2%, used
to filter small lines
    [WindowData, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowHough (im_original, fudgefactor,
algname, fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
    TempWinData2=WindowData;
    ErrorTempWinData=abs(TempWinData1-TempWinData2);
    TempCountWinData = sum(ErrorTempWinData(:) > 3);
        if TempCountWinData >= 1
```

```
            fillvert_perc=10; minlenvert_perc=10; fillhor_perc=10;
minlenhor_perc=10;
            [~, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet4, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
            minlenvert_perc=max_len_vertical_perc-5;
minlenhor_perc=max_len_horizontal_perc-5; % Longest line found - 2%, used
to filter small lines
            [WindowData, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet4, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
            if  sum(WindowData) == 0
            aFail(3)=aFail(3)+1;
            end
        else
        WindowData  = TempWinData2;
        if  sum(WindowData) == 0
        aFail(2)=aFail(2)+1;
        end
        end
    if  sum(WindowData) == 0
        fillvert_perc=20; minlenvert_perc=10; fillhor_perc=20;
minlenhor_perc=10;
        [~, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet4, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
        minlenvert_perc=max_len_vertical_perc-5;
minlenhor_perc=max_len_horizontal_perc-5; % Longest line found - 2%, used
to filter small lines
        [WindowData, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet4, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
            if  sum(WindowData) == 0
            aFail(4)=aFail(4)+1;
                fudgefactor=0.4; algname= 'Roberts';
                [WindowData]=DetectWindowEdges (im_original, fudgefactor,
algname);
                if  sum(WindowData) == 0
                aFail(5)=aFail(5)+1;
                end
            end
    end
    end
else
    %fudgefactor=0.2; algname= 'Roberts';
    %fillvert_perc=8; minlenvert_perc=10; fillhor_perc=8;
minlenhor_perc=10;
    %[~, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowHough (im_original, fudgefactor,
algname, fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
    %minlenvert_perc=max_len_vertical_perc-5;
minlenhor_perc=max_len_horizontal_perc-5; % Longest line found - 2%, used
to filter small lines
    %[WindowData, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowHough (im_original, fudgefactor,
algname, fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
```

```
        fillvert_perc=30; minlenvert_perc=10; fillhor_perc=30;
minlenhor_perc=10;
        [~, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet4, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
        minlenvert_perc=max_len_vertical_perc-5;
minlenhor_perc=max_len_horizontal_perc-5; % Longest line found - 2%, used
to filter small lines
        [WindowData, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet4, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc);
        if  sum(WindowData) == 0
        aFail(6)=aFail(6)+1;
        end
end
bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-WindowData(3)/2
WindowData(4) WindowData(3)]; %bounding_box=[centerx-width/2 centery-
height/2 width height];
%figure (2)
subplot (2,1,1); % In pixels
im_original = imread(im_name); imshow(im_original); hold on
set(gcf, 'Position', get(0, 'Screensize'));
rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
pixels',WindowData(1),WindowData(2),WindowData(3),WindowData(4));
text(WindowData(1),WindowData(2),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
subplot (2,1,2); %Real size
im_original = imread(im_name); imshow(im_original); hold on
set(gcf, 'Position', get(0, 'Screensize'));
rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
Distance = 2090; %Distancia entre el objeto y el lente (en mm)
FocalDis = 4.2; % Distancia focal en mm
SensorSizeXmm = 3.58; % Ancho sensor en mm
SensorSizeYmm = 2.08; % Alto sensor en mm
SensorSizeXpx = 1280; % Ancho sensor en pixeles
SensorSizeYpx = 720; % Alto sensor en pixeles
WindowDataReal(3)=Distance * (WindowData(3)/SensorSizeYpx) *
(SensorSizeYmm/FocalDis); %Height
WindowDataReal(4)=Distance * (WindowData(4)/SensorSizeXpx) *
(SensorSizeXmm/FocalDis); %Width
PixelSizeY=WindowDataReal(3)/WindowData(3);
PixelSizeX=WindowDataReal(4)/WindowData(4);
WindowDataReal(1)=(WindowData(1)-(SensorSizeXpx/2))*PixelSizeX; %Center X
WindowDataReal(2)=-(WindowData(2)-(SensorSizeYpx/2))*PixelSizeY; %Center
Y
temptext=sprintf ('Center \n X%0.1f mm \n Y%0.1f mm \n Z %0.1f mm \n
Height=%0.2f mm \n Width=%0.2f
mm',WindowDataReal(1),WindowDataReal(2),Distance,
WindowDataReal(3),WindowDataReal(4));
text(WindowData(1),WindowData(2),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);


function [WindowData]=DetectWindowCNN(arturnet, im_original)
%---------------Loading the image (photo taken)-------------------------
```

```
C = semanticseg(im_original, arturnet);
%cmap = jet(numel(classes));
im_CNN = labeloverlay(im_original,C,'Transparency',0);
rotation = 0; %Rotate the image
im_rot = imrotate(im_CNN,rotation);
im_grayscale = rgb2gray(im_rot); % Grayscale image
[rows,columns]=size(im_grayscale); % Size of the original image
im_bw=im2bw(im_rot);
%------------------Imcomplement image-----------------------------------
BWim = imcomplement(im_bw);
BWim = imclearborder(BWim); %clean if touching the border
%-------------------------Area filter-------------------------------------
maxarnum = 10;
BW = bwareafilt(BWim, maxarnum);%filter the max number of regionprops
areas to find
L=bwlabel(BW); %Devuelve una matriz matrix L, del mismo tamaño que BW2,
etiquetando las cosas que estan unidas entre si.
prop=regionprops(L,'all'); %Funcion para analisis de regiones en matlab
%--------------------Filter detected areas by area %-----------------
maxarea = max( [prop.Area] ); %finds the biggest area found
sizeim=maxarea+(maxarea*0.3); % biggest area + 20% of it, to detect more
areas within that range
sizeim2=maxarea-(maxarea*0.3);% biggest area - 20% of it, to detect more
areas within that range
idx = find([prop.Area] > sizeim2 & [prop.Area] < sizeim); %Finds the
areas within the range
L = ismember(L,idx);
%-------Utilizacion de "Extrema" para encontrar el número de bordes------
% set(gcf, 'Position', get(0, 'Screensize'));
% subplot(2,2,1), imshow(im_original), title ('Imagen Original')
% subplot(2,2,2), imshow(im_edges),title ('Bordes detectados Roberts')
% subplot(2,2,3), imshow(ImFin),title ('Lineas detectadas Transformada de
Hough')
% subplot(2,2,4), imshow(im_original),title ('Vidrio detectado')
S = regionprops(L,'Extrema','Centroid');
S2 = regionprops(L,'BoundingBox','Area');
se = strel('disk',5);
for j = 1:length(S)
temp = round(S(j).Extrema);
temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
mask = zeros(max(temp(:,1)),max(temp(:,2)));
for k = 1:8
mask(temp(k,1),temp(k,2))=1;
end
mask2 = imdilate(mask,se);
[~,numbordes(j)] = bwlabel(mask2,8);
end
%--------------------Boundingbox height, width and centroid------------
bb = regionprops(L,'BoundingBox');
bbMatrix = vertcat(bb(:).BoundingBox); %matrix of the boundingbox [Left,
Top, Width, Height]
nodetect=isempty(bbMatrix);
if nodetect==0 % 1 if empty, 0 if not empty, avoids an empty matrix crash
width=bbMatrix(:,3);
height=bbMatrix(:,4);
centerx=bbMatrix(:,1)+ bbMatrix(:,3)/2;
```

```
centery=bbMatrix(:,2)+ bbMatrix(:,4)/2;
end
%-------------------------------------------------------------------------
%-----Closest region to the center, in case more than 1 window not
touching the
%-----border enters the photo frame and at the same time they are all the
same size (tall windows case).
centerimagex=round(columns/2); %center of the image, pixel position in X
coordinate
centerimagey=round(rows/2); %center of the image, pixel position in Y
coordinate
distance_cenx=zeros(1,j);
distance_ceny=zeros(1,j);
for g=1:j
distance_cenx(g) = abs(centerimagex - centerx(g)); %how close each area
is to the center of the image, in x coordinate
distance_ceny(g) = abs(centerimagey - centery(g)); %how close each area
is to the center of the image, in y coordinate
end
if nodetect==0
distance_cen=distance_cenx+distance_ceny;
[min_val,min_pos]=min(distance_cen);
%-------------------------------------------------------------------------
%-------Only show and place text on the area closest to the center
% temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
pixels',centerx(min_pos),centery(min_pos),height(min_pos),width(min_pos))
;
% text(centerx(min_pos),centery(min_pos),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
%-------------------------------------------------------------------------
%---------------Data for the machine vision program----------------------
WindowData = [centerx(min_pos), centery(min_pos), height(min_pos),
width(min_pos)]
%bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-
WindowData(3)/2 WindowData(4) WindowData(3)]; %bounding_box=[centerx-
width/2 centery-height/2 width height];
%rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
else
%sprintf ('Glass area not detected, segmentation failed')
WindowData = [0 0 0 0];
end
```

```
function [WindowData, max_len_horizontal_perc,
max_len_vertical_perc]=DetectWindowCNNHough(arturnet, im_original,
fillvert_perc, minlenvert_perc, fillhor_perc, minlenhor_perc)
%---------------Loading the image (photo taken)-------------------------
C = semanticseg(im_original, arturnet);
%cmap = jet(numel(classes));
im_CNN = labeloverlay(im_original,C,'Transparency',0);
rotation = 0; %Rotate the image
im_rot = imrotate(im_CNN,rotation);
im_grayscale = rgb2gray(im_rot); % Grayscale image
```

```
im_bw=im2bw(im_rot);
%im_bw=imcomplement(im_bw);
%imshow(im_bw)
%hold on
%--------------------Hough Transform-------------------------------------
%--------------------Vertical Lines--------------------------------------
[H,T,R] = hough(im_bw,'RhoResolution',4,'Theta',-0.5:.1:0.5);
%# Detect peaks
peaksnum = 30;
peaksth = 0.2;
P  = houghpeaks(H,peaksnum,'threshold',ceil(peaksth*max(H(:))));
%# Detect lines and overlay on top of image
[rows,columns]=size(im_grayscale); % Size of the original image
ImFin=zeros(rows,columns); % Creates a new matrix of zeros, the size of
the original image
max_len_vertical=0; %declaring variable
max_len_horizontal=0; %declaring variable
max_len_vertical_perc=0;
max_len_horizontal_perc=0;
linesfillvertical = (rows/100)*fillvert_perc; % fills lines gaps,
percentage of the height of the image
linesminlenvertical = (rows/100)*minlenvert_perc;% minimum line lenght in
pixels, percentage of the height of the image
lines =
houghlines(im_bw,T,R,P,'FillGap',linesfillvertical,'MinLength',linesminle
nvertical);
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    % Longest line segment size and determine the endpoints of it
         len = norm(lines(k).point1 - lines(k).point2);
         if ( len > max_len_vertical)
           max_len_vertical = len;
           max_len_vertical_perc = (max_len_vertical*100)/rows;
           xy_long_vertical = xy;
         end
    %-------------Draws a line on the zeros matrix-----------------------
    %Done to make a black and white image with the lines drawn into and
    %not just the initial and endpoints coordinates
    x=[xy(1,1) xy(2,1)];
    y=[xy(1,2) xy(2,2)];
    nPoints = max(abs(diff(x)), abs(diff(y)))+1;  % Number of points in
line
    rIndex = round(linspace(y(1), y(2), nPoints));  % Row indices
    cIndex = round(linspace(x(1), x(2), nPoints));  % Column indices
    index = sub2ind(size(ImFin), rIndex, cIndex);     % Linear indices
    ImFin(index) = 255; % Zeros matrix with lines drawn on it
end
%--------------------Horizontal Lines-----------------------------------
[H,T,R] = hough(im_bw,'RhoResolution',4,'Theta',-90:.1:-89.5); %
%# Detect peaks
%peaksnum = 30;
%peaksth = 0.2;
P  = houghpeaks(H,peaksnum,'threshold',ceil(peaksth*max(H(:))));
%# Detect lines and overlay on top of image
linesfillhorizontal = (columns/100)*fillhor_perc; % percentage of the
width of the image
```

```
linesminlenhorizontal = (columns/100)*minlenhor_perc; % percentage of the
wight of the image
lines =
houghlines(im_bw,T,R,P,'FillGap',linesfillhorizontal,'MinLength',linesmin
lenhorizontal);
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    % Longest line segment size and determine the endpoints of it
         len = norm(lines(k).point1 - lines(k).point2);
         if ( len > max_len_horizontal)
           max_len_horizontal = len;
           max_len_horizontal_perc = (max_len_horizontal*100)/columns;
           xy_long_horizontal = xy;
         end
    %-------------Draws a line on the zeros matrix-----------------------
    x=[xy(1,1) xy(2,1)];
    y=[xy(1,2) xy(2,2)];
    nPoints = max(abs(diff(x)), abs(diff(y)))+1;  % Number of points in
line
    rIndex = round(linspace(y(1), y(2), nPoints));  % Row indices
    cIndex = round(linspace(x(1), x(2), nPoints));  % Column indices
    index = sub2ind(size(ImFin), rIndex, cIndex);     % Linear indices
    ImFin(index) = 255;% Zeros matrix with lines drawn on it
end
%--------------------Horizontal Lines2-----------------------------------
[H,T,R] = hough(im_bw,'RhoResolution',4,'Theta',89.5:.1:89.9); %
hold on
%# Detect peaks
%peaksnum = 30;
%peaksth = 0.2;
P  = houghpeaks(H,peaksnum,'threshold',ceil(peaksth*max(H(:))));
%# Detect lines and overlay on top of image
%linesfillhorizontal = 8;
%linesminlenhorizontal = 75;
lines =
houghlines(im_bw,T,R,P,'FillGap',linesfillhorizontal,'MinLength',linesmin
lenhorizontal);
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    % Longest line segment size and determine the endpoints of it
         len = norm(lines(k).point1 - lines(k).point2);
         if ( len > max_len_horizontal)
           max_len_horizontal = len;
           max_len_horizontal_perc = (max_len_horizontal*100)/columns;
           xy_long_horizontal = xy;
         end

%plot(xy_long_vertical(:,1),xy_long_vertical(:,2),'LineWidth',2,'Color','
cyan');

%plot(xy_long_horizontal(:,1),xy_long_horizontal(:,2),'LineWidth',2,'Colo
r','green');
    %-------------Draws a line on the zeros matrix-----------------------
    x=[xy(1,1) xy(2,1)];
    y=[xy(1,2) xy(2,2)];
    nPoints = max(abs(diff(x)), abs(diff(y)))+1;  % Number of points in
line
```

```
    rIndex = round(linspace(y(1), y(2), nPoints));  % Row index
    cIndex = round(linspace(x(1), x(2), nPoints));  % Column index
    index = sub2ind(size(ImFin), rIndex, cIndex);     % Linear index
    ImFin(index) = 255;% Zeros matrix with white lines drawn on it
end
%--------------------Line dilate-----------------------------------------
linelen = 2;
ste90 = strel('line', linelen, 90);
ste0 = strel('line', linelen, 0);
BWdil = imdilate(ImFin, [ste90 ste0]);
BWdil = imbinarize(BWdil);
%------------------Imcomplement image-----------------------------------
BWim = imcomplement(BWdil);
BWim = imclearborder(BWim); %clean if touching the border
%-------------------------Area filter-------------------------------------
maxarnum = 10;
BW = bwareafilt(BWim, maxarnum);%filter the max number of regionprops
areas to find
L=bwlabel(BW); %Devuelve una matriz matrix L, del mismo tamaño que BW2,
etiquetando las cosas que estan unidas entre si.
prop=regionprops(L,'all'); %Funcion para analisis de regiones en matlab
%--------------------Filter detected areas by area %-----------------
maxarea = max( [prop.Area] ); %finds the biggest area found
sizeim=maxarea+(maxarea*0.3); % biggest area + 20% of it, to detect more
areas within that range
sizeim2=maxarea-(maxarea*0.3);% biggest area - 20% of it, to detect more
areas within that range
idx = find([prop.Area] > sizeim2 & [prop.Area] < sizeim); %Finds the
areas within the range
L = ismember(L,idx);
%-------Utilizacion de "Extrema" para encontrar el numero de bordes------
% set(gcf, 'Position', get(0, 'Screensize'));
% subplot(2,2,1), imshow(im_original), title ('Imagen Original')
% subplot(2,2,2), imshow(im_edges),title ('Bordes detectados Roberts')
% subplot(2,2,3), imshow(ImFin),title ('Lineas detectadas Transformada de
Hough')
% subplot(2,2,4), imshow(im_original),title ('Vidrio detectado')
S = regionprops(L,'Extrema','Centroid');
S2 = regionprops(L,'BoundingBox','Area');
se = strel('disk',5);
for j = 1:length(S)
temp = round(S(j).Extrema);
temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
mask = zeros(max(temp(:,1)),max(temp(:,2)));
for k = 1:8
mask(temp(k,1),temp(k,2))=1; end
mask2 = imdilate(mask,se);
[~,numbordes(j)] = bwlabel(mask2,8);
end
%--------------------Boundingbox height, width and centroid------------
bb = regionprops(L,'BoundingBox');
bbMatrix = vertcat(bb(:).BoundingBox); %matrix of the boundingbox [Left,
Top, Width, Height]
nodetect=isempty(bbMatrix);
if nodetect==0 % 1 if empty, 0 if not empty, avoids an empty matrix crash
width=bbMatrix(:,3);
```

```
height=bbMatrix(:,4);
centerx=bbMatrix(:,1)+ bbMatrix(:,3)/2;
centery=bbMatrix(:,2)+ bbMatrix(:,4)/2;
end
%-------------------------------------------------------------------------
%-----Closest region to the center, in case more than 1 window not
touching the border enters the photo frame and at the same time they are
all the same size (tall windows case).
centerimagex=round(columns/2); %center of the image, pixel position in X
coordinate
centerimagey=round(rows/2); %center of the image, pixel position in Y
coordinate
distance_cenx=zeros(1,j);
distance_ceny=zeros(1,j);
for g=1:j
distance_cenx(g) = abs(centerimagex - centerx(g)); %how close each area
is to the center of the image, in x coordinate
distance_ceny(g) = abs(centerimagey - centery(g)); %how close each area
is to the center of the image, in y coordinate
end
if nodetect==0
distance_cen=distance_cenx+distance_ceny;
[min_val,min_pos]=min(distance_cen);
%-------------------------------------------------------------------------
%-------Only show and place text on the area closest to the center
% temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
pixels',centerx(min_pos),centery(min_pos),height(min_pos),width(min_pos))
;
% text(centerx(min_pos),centery(min_pos),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
%-------------------------------------------------------------------------
%---------------Data for the machine vision program----------------------
WindowData = [centerx(min_pos), centery(min_pos), height(min_pos),
width(min_pos)]
%bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-
WindowData(3)/2 WindowData(4) WindowData(3)]; %bounding_box=[centerx-
width/2 centery-height/2 width height];
%rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
else
%sprintf ('Glass area not detected, segmentation failed')
WindowData = [0 0 0 0];
end
function [WindowData]=DetectWindowEdges (im_original, fudgefactor,
algname)
%---------------Loading the image (photo taken)-------------------------
rotation = 0; %Rotate the image
im_rot = imrotate(im_original,rotation);
im_grayscale = rgb2gray(im_rot); % Grayscale image
%im_grayscale = imguidedfilter(im_grayscale); % Guided filtering
[rows,columns]=size(im_grayscale); % Size of the original image
%-----------Edge detection algorithm--------------------------------------
[~, threshold] = edge(im_grayscale,algname); %Best threshold for the
image with the specific algorithm
im_edges = edge(im_grayscale,algname,threshold * fudgefactor); %BW =
edge(I,'Canny',threshold,sigma) %'Sobel' 'Prewitt' 'Roberts' 'log'
```

```
%im_edges = bwareaopen(im_edges,4,4); %Removes small objects from binary
image
%--------------------Line dilate-----------------------------------------
linelen = 2;
ste90 = strel('line', linelen, 90);
ste0 = strel('line', linelen, 0);
BWdil = imdilate(im_edges, [ste90 ste0]);
%BWdil = imbinarize(BWdil);
%------------------Imcomplement image-----------------------------------
BWim = imcomplement(BWdil);
BWim = imclearborder(BWim); %clean if touching the border
%-------------------------Area filter-----------------------------------
maxarnum = 10;
BW = bwareafilt(BWim, maxarnum);%filter the max number of regionprops
areas to find
L=bwlabel(BW);
prop=regionprops(L,'all');
%--------------------Filter detected areas by area %-----------------
maxarea = max( [prop.Area] ); %finds the biggest area found
sizeim=maxarea+(maxarea*0.3); % biggest area + 20% of it, to detect more
areas within that range
sizeim2=maxarea-(maxarea*0.3);% biggest area - 20% of it, to detect more
areas within that range
idx = find([prop.Area] > sizeim2 & [prop.Area] < sizeim); %Finds the
areas within the range
L = ismember(L,idx);
%-------Utilizacion de "Extrema" para encontrar el número de bordes------
S = regionprops(L,'Extrema','Centroid');
S2 = regionprops(L,'BoundingBox','Area');
se = strel('disk',5);
for j = 1:length(S)
temp = round(S(j).Extrema);
temp(:,1) = temp(:,1) - min(temp(:,1)) + 1;
temp(:,2) = temp(:,2) - min(temp(:,2)) + 1;
mask = zeros(max(temp(:,1)),max(temp(:,2)));
for k = 1:8
mask(temp(k,1),temp(k,2))=1;
end
mask2 = imdilate(mask,se);
[~,numbordes(j)] = bwlabel(mask2,8);
end
%---------------------Boundingbox height, width and centroid-------------
bb = regionprops(L,'BoundingBox');
bbMatrix = vertcat(bb(:).BoundingBox); %matrix of the boundingbox [Left,
Top, Width, Height]
nodetect=isempty(bbMatrix);
if nodetect==0 % 1 if empty, 0 if not empty, avoids an empty matrix crash
width=bbMatrix(:,3);
height=bbMatrix(:,4);
centerx=bbMatrix(:,1)+ bbMatrix(:,3)/2;
centery=bbMatrix(:,2)+ bbMatrix(:,4)/2;
end
%-------------------------------------------------------------------------
%-----Closest region to the center, in case more than 1 window not
touching the border enters the photo frame and at the same time they are
all the same size (tall windows case).
```

```
centerimagex=round(columns/2); %center of the image, pixel position in X
coordinate
centerimagey=round(rows/2); %center of the image, pixel position in Y
coordinate
distance_cenx=zeros(1,j);
distance_ceny=zeros(1,j);
for g=1:j
distance_cenx(g) = abs(centerimagex - centerx(g)); %how close each area
is to the center of the image, in x coordinate
distance_ceny(g) = abs(centerimagey - centery(g)); %how close each area
is to the center of the image, in y coordinate
end
if nodetect==0
distance_cen=distance_cenx+distance_ceny;
[min_val,min_pos]=min(distance_cen);
%-------------------------------------------------------------------------
%-------Only show and place text on the area closest to the center
% temptext=sprintf ('Center \n X%0.1f pixels \n Y%0.1f pixels \n
Height=%0.1f pixels \n Width=%0.1f
pixels',centerx(min_pos),centery(min_pos),height(min_pos),width(min_pos))
;
% text(centerx(min_pos),centery(min_pos),
temptext,'HorizontalAlignment','center','color', 'w','fontsize', 10);
%-------------------------------------------------------------------------
%---------------Data for the machine vision program----------------------
WindowData = [centerx(min_pos), centery(min_pos), height(min_pos),
width(min_pos)];
%bounding_box=[WindowData(1)-WindowData(4)/2 WindowData(2)-
WindowData(3)/2 WindowData(4) WindowData(3)]; %bounding_box=[centerx-
width/2 centery-height/2 width height];
%rectangle('Position',bounding_box,'LineWidth',1,'EdgeColor','r')
else
%sprintf ('Glass area not detected, segmentation failed')
WindowData = [0 0 0 0];
end
%-------------------------------------------------------------------------
end
```