INSTITUTO TECNOLÓGICO SUPERIOR DE TEZIUTLÁN

Tesis

"Desarrollo de algoritmo aplicando técnicas de visión artificial para identificar los movimientos en un diagrama estado-fase para la programación de PLC"

PRESENTA:

ANDREA VÁZQUEZ CATARINO

CON NÚMERO DE CONTROL

17TE0170

PARA OBTENER EL TÍTULO DE:

INGENIERA MECATRÓNICA

CLAVE DEL PROGRAMA ACADÉMICO

IMCT-2010-229

DIRECTOR (A) DE TESIS:

M.S.C. ALFREDO CARRASCO ARAOZ

"La Juventud de hoy, Tecnología del Mañana"

TEZIUTLÁN, PUEBLA, MARZO 2022

PRELIMINARES

Agradecimientos

A MI ASESOR

Gracias por el apoyo y al esfuerzo

brindado durante esta faceta, por ayudarme

a culminar esta gran importante etapa.

Gracias por darme la oportunidad

de creer en mis conocimientos y habilidades.

A MI PADRE

Eres lo más importante para mí, gracias

por darme ánimos, por guiarme en el

camino sin ti no lo lograría.

Dedicatoria

Te dedico este logro a ti ¡Gracias por todo papá!, sin ti no lo hubiera logrado, gracias por cada ánimo, por cada aliento de seguir adelante y jamás rendirme, gracias por estar conmigo en esta gran etapa de mis estudios universitarios, estoy y estaré agradecida por siempre, jamás me alcanzará la vida para poder darte las gracias por todo lo que hiciste por mí y caminar conmigo, ¡Gracias por todo papá, te quiero inmensamente!

Fuiste y serás un ejemplo de vida, de lucha, de perseguir por los sueños, gracias mi amor (Moisés) por ayudarme a nunca rendirme, por decirme cada palabra esperanzadora, por ayudarme a nunca dar la vuelta a una página sin haberla terminado perfectamente, te dedico mi trabajo mi amor, sin ti no lo hubiera logrado, ¡Gracias mi amor!

Resumen

Este proyecto describe la creación de un algoritmo en lenguaje Python para la visualización de movimientos en un diagrama estado-fase. El desarrollo de este algoritmo permite identificar los estados-fase para convertirlos en un diagrama de mando por medio de comandos propios del lenguaje Python. El diseño del código está basado en el procesamiento de imágenes, a partir de ello se permite tener una lectura del diagrama de estado-fase propuesto.

Introducción

La visión artificial es una tecnología que emula a la visión humana y permite obtener, procesar e interpretar imágenes tomadas durante los procesos de producción. Las máquinas con visión artificial analizan y decodifican la información obtenida durante los procesos de producción para tomar decisiones y actuar de la manera más conveniente, a través de un proceso automatizado. Para la problemática a evaluar presentada, las operaciones de filtrado para el tratamiento de imágenes son significativa ya que con ello podemos identificar aspectos y características no visibles en una imagen por la visión humana, es aquí donde tiene una gran importancia este tipo el tratamiento de imágenes.

Así, los sistemas de visión artificial aplicados a la robótica son muy útiles para realizar tareas de inspección de muestreo, supervisión y controles de calidad, con una precisión extrema. No obstante, esta tecnología día a día sigue evolucionando por ello, pensar en mejorar los procesos actuales en el monitoreo es una gran apuesta. Las áreas en las que se puede aplicar visión artificial son diversas, los cuales cabe destacar los distintos tipos de robots a nivel industrial, estos observan características cuantitativas, pues que son utilizados para mediciones con respecto al tamaño, grosores, ángulos, por otro lado, tenemos los robots adaptados al posicionamiento, reubicación y trayectorias de objetos. Todas estas adaptaciones antes mencionadas se aplican de cierta manera que se puedan visualizar características que la visión humana no tiene, ya que visualiza tendencias cualitativas de manera muy genérica por ello no cumple con los criterios específicos a evaluar en un proceso. A todo tipo de aplicación en un proceso de visión artificial de producción orientado al control automatizado es importante evaluar es la necesidad de tener un personal calificado el cual tenga un nivel de preparación adecuado, de lo contrario provoca mucho tiempo para lograr interpretar un diagrama estado-fase si este tiene cierto grado de complejidad. Por lo que a través de este proyecto se propone la lectura de una imagen de un diagrama estado-fase para poder visualizar la interpretación del mismo.

Índice

Índice de imágenes

Índice de tablas

CAPÍTULO I
GENERALIDADES DEL PROYECTO

1.1 Descripción de la empresa

1.1.1 Antecedentes

Teziutlán es históricamente un polo de desarrollo económico en la región Nororiental del Estado de Puebla, cuyos fundamentos se fincan, primero, en la industria minera y metalúrgica, posteriormente en la fruticultura y la ganadería, y más recientemente en la industria de la confección de ropa. Como es natural la actividad industrial siempre ha traído aparejado el crecimiento de otras actividades económicas, como son el comercio, el transporte, los servicios financieros y de manera muy especial, la educación.

El Instituto Tecnológico Superior de Teziutlán, atento a las demandas de la sociedad, y a los principios de la Ley de Educación del Estado de Puebla, se consolida como una Institución cuyo objetivo es lograr una educación de calidad, moderna y eficaz, orientada al servicio, acercándola a las necesidades e intereses de la población, que promueva el uso transparente y eficiente de los recursos humanos, materiales y financieros de que disponga, y que cumpla puntualmente con sus programas de trabajo.

Las carreras que se ofrecen actualmente en el Instituto Tecnológico Superior de Teziutlán son:

- Ingeniería en Gestión Empresarial.
- Ingeniería en Industrias Alimentarias.
- Ingeniería en Sistemas Computacionales.
- Ingeniería Industrial.
- Ingeniería Informática.
- Ingeniería Mecatrónica.

El egresado del área de ingeniería mecatrónica tiene la capacidad de construir, diseñar e innovar sistemas de control y automatizaciones industriales, con el

propósito de solventar las necesidades del sector productivo y de servicios, así como utilizar los recursos en forma segura, racional, eficiente, económica y sustentable.

1.2 Planteamiento del problema

Para un proceso de automatización industrial es necesario tomar en cuenta el diseño del circuito electrónico de control para así poder visualizar los distintos elementos de control que lo conforman, a partir de esto se construye un diagrama Ladder, el cual se especifican todos los requerimientos necesarios a través de una estructura normalizada ya establecida se desglosan entradas y salidas lógicas, dentro de estas especificaciones se asignan las activaciones de los actuadores involucrados, para llevar a cabo el proceso de activaciones se construye un diagrama estado-fase en este diagrama se enumeran los movimientos de los actuadores, se muestra de tal manera que se visualizan de forma lineal o no dependiendo de la secuencia, es decir el número de actuadores a intervenir y los tiempos de activación.

Uno de los principales problemas surge a partir de la construcción del diagrama Ladder, ya que se necesita interpretar de manera oportuna y clara un diagrama de estado-fase para su implementación con el PLC, cuando el diagrama contiene más de dos pistones se vuelve difícil la interpretación de la secuencia. La construcción de la secuencia a través del algoritmo es únicamente para ayudar al diseño previo y manual del mismo. En otros sectores de la industria esto representa es un gran problema ya que el desarrollo y la interpretación del mismo debe de realizarse de manera oportuna, por lo que con lleva a un aumento de tiempo considerable por lo que surgen afectaciones, si esto se enfoca no solo a nivel industrial, si no al proceso educacional de aprendizaje para facilitar la compresión de esta etapa de implementación de diagramas estado-fase en procesos de automatización industrial en los alumnos y así reducir los tiempos que conlleva esta etapa.

1.3 Preguntas de investigación

- ¿Cuál es el impacto del desarrollo de un algoritmo el cuál pueda procesar un diagrama de estado-fase para la mejora de implementación de tiempos de ejecución en un PLC?
- ¿La selección de la metodología para la lectura de contornos es la adecuada?
- ¿La segmentación de la imagen de entrada es la correcta?
- ¿Diseñar el algoritmo proporciona todos parámetros de un diagrama estado-fase?

1.4 Objetivos

Objetivo general

Diseñar un algoritmo aplicando técnicas de visión artificial para identificar los movimientos en un diagrama estado-fase para la programación de PLC

Objetivos específicos

- Analizar el proceso de implementación de un sistema de automatización mediante un PLC.
- Establecer los parámetros necesarios para la interpretación de un diagrama estado-fase.
- Seleccionar las principales librerías para el procesamiento de imágenes en la librería open CV.
- Establecer las operaciones básicas para lecturas del contorno en una imagen.
- Asignar los rangos límites del tamaño de la imagen para especificar la lectura de la imagen.
- Considerar los parámetros necesarios para la lectura de pixeles.
- Estructurar etiquetas de movimientos lógicos de los actuadores.

1.5 Justificación

Un problema que se presenta para el desarrollo y el proceso de un circuito electroneumático es la etapa posterior a la de asignaciones de movimientos lógicos de los actuadores, es decir aquella cuando se construye el diagrama estado-fase partiendo antes del diagrama Ladder, para la construcción del diagrama estado-fase es primordial comprender de manera significativa los movimientos, esto se vuelve tedioso cuando el número de los actuadores es grande en procesos industriales o en otro ámbitos por ello se propone un código con el lenguaje de programación Python, el cual utiliza técnicas de visión artificial por ello se utiliza open CV. En las distintas fuentes consultadas no existe ningún dispositivo que realicé la lectura de un diagrama estado-fase el cual sea implementado en el sector productivo o en otras áreas para mejora de tiempos de procesos en la automatización mediante PLC's, es por ello que no se ha empleado algún algoritmo implementado en un dispositivo para este objetivo.

Partiendo de lo antes mencionado se construye un algoritmo el cual solucione esta problemática, para este proceso se basa en el tratamiento de imágenes, ya que todo este proceso parte de una imagen el cual está plasmado un diagrama estado-fase.

CAPÍTULO II
MARCO TEÓRICO

2.1 Visión Artificial

La visión artificial es una rama de la inteligencia artificial, la cual hace uso de los diversos medios de tecnología para adquirir y procesar imágenes, este se basa en tres procesos básicos los cuales se mencionan a continuación:

Adquisición: Se captura la imagen a través de una cámara para llevarla al siguiente proceso.

Procesamiento: En esta etapa se realiza un preprocesamiento para mejorar algunas características de la imagen, para ello se deben aplicar diversas técnicas de filtrado de imagen.

Análisis: Se resaltan propiedades de la imagen, donde se procede a realizar operaciones propias de la visión artificial como detección de esquinas, bordes, formas, contornos etc.

Aplicaciones: El objeto final que se requiere controlar dependiendo de la información que se obtuvo anteriormente.

2.1.1 Tipos de visión artificial

Los sistemas de visión artificial son de las principales herramientas más usadas en la industria, es por ello que existen diferentes tipos de sistemas de visión artificial. Esto depende del proceso de sistemas que tengan cada uno, es decir las necesidades para solventar una línea de producción, de acuerdo a ello se clasifican en tres tipos de sistemas: cámaras, sensores y sistemas de visión avanzados.

Sensores: Estos se basan en sensores digitales que tienen características ópticas especializadas para la adquisición de imágenes para que un software procese los resultados obtenidos

Cámaras: Su capacidad de procesamiento de imágenes es notoria, ya que reside en la resolución de imagen

Sistemas de visión avanzados: son sistemas de visión integrados ya que el propósito de este es mejorar funciones relacionadas con el software y procesamiento de datos.

2.1.2 Aplicaciones de la visión artificial

Las aplicaciones de la visión artificial son diversas, la principal funcionalidad de estos son que permite mejorar procesos de producción con ello implica aumento de calidad de la producción, control de calidad preciso, reducción de pérdidas por ello disminuyen los costos de producción.

Por otra parte, tenemos la implementación de sistemas multisensoriales en el área de la navegación, donde la capacidad sensorial es amplia, este puede tener diversas aplicaciones, para un sistema de visión artificial hace uso de un sensor único.

Automoción: Enfocado al control de calidad de ensamblaje en piezas para vehículos.

Electrónica: Es utilizada para ensamble de piezas en placas electrónicas, para soldaduras y limpieza de la misma que es donde radica principalmente el uso de esta herramienta.

Medicina: Esta área radica el procesamiento de imágenes las cuales un claro ejemplo son radiografías, resonancias magnéticas etc., donde la interpretación de imágenes radica en herramientas de filtrado (como contrastes y gamas diferentes) para obtener resultados de salud del paciente.

Geología: El uso de visión artificial se usa para la detección de relieves naturales en distintas áreas del globo terráqueo, movimientos en placas tectónicas, y demás.

Bilogía: Son áreas donde la interpretación de imágenes se utiliza para la identificación de diversos tipos de relieves en áreas naturales, para indagar diferentes características propias de áreas naturales, como colores, temperaturas etc.

2.2 Open CV

Open CV (Open Source Computer Vision) es una librería libre de visión artificial, su objetivo es el tratamiento de imágenes destinadas para el uso de herramientas tecnológicas para uso de visión por computador. Open CV fue diseñada por la compañía Intel donde contiene más de 2500 algoritmos, esta librería esta desarrollada para distintos lenguajes de programación como Python, java y C++

La librería está formada por módulos los cuales son los siguientes:

CV: está conformada por funciones básicas.

Imgproc: contiene funciones para el control de imágenes.

Highgui: es un módulo para la lectura-escritura de imágenes estáticas y de video.

Features2d: funciones destinadas a la detección de puntos y descriptores.

Calib3d: su función es la calibración de la cámara que se está utilizando y características geométricas para vistas amplias y exactas.

Video: contiene funciones de movimiento en objetos en tiempo real

Objdetect: es un módulo que contiene funciones para la detección de objetos.

Core: el módulo contiene funciones básicas para el uso de arreglos y relacionadas con otros módulos determinados.

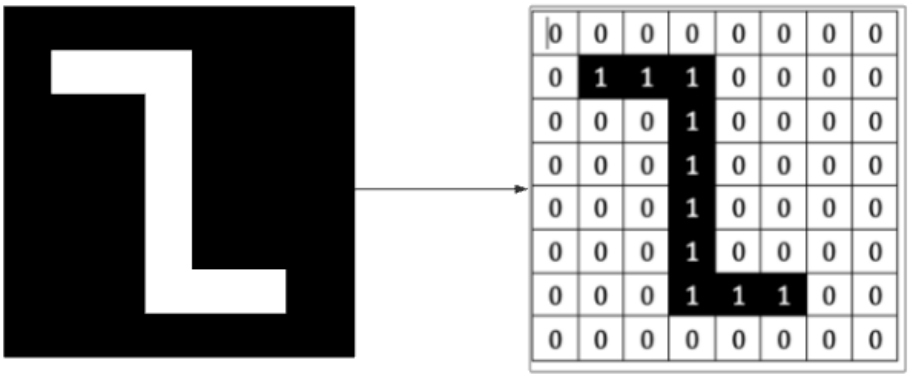
2.2.1 Tipos de imágenes

Para realizar una interpretación de una imagen es de vital importancia algún medio que proporcione esta, es por ello que la forma de interpretación de un ordenador no es la correcta, por ello se deben aplicar distintas técnicas de procesamiento para que la imagen tengan una función discontinua, es decir el nivel de luminosidad y nivel de grises de una imagen no debe ser consistente. En la librería de open CV es una matriz de datos es decir donde cada pixel es un elemento dentro de la misma. EN open CV contiene tres tipos de interpretación de imágenes.

Imágenes binarias: La interpretación para este tipo de imagen contiene de 1 bit/píxel es por ello que en este tipo solo se tiene dos tipos de colores, es decir blanco y negro, esta imagen puede ser almacenada en una matriz doble o una matriz uint8, esto va a depender del uso de memoria.

Figura 1

Representación imagen binaria

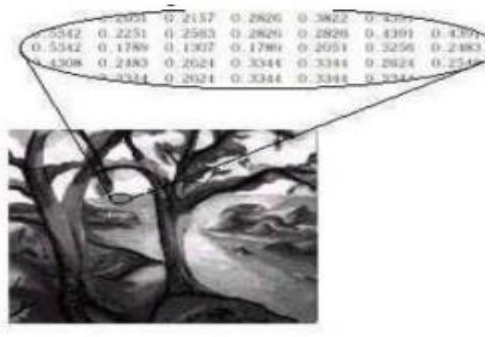


Fuente: https://revistadigital.inesem.es/informatica-y-tics/opencv/, Marín, R. 2020

Imágenes escala de grises: en esta imagen todas las intensidades conservan un mismo rango, la matriz puede ser de dos tipos, unit 8 es que va desde 0 a 255, en cambio para una matriz doble los valores de intervalo son de 0 a 1, donde estos son los tipos de intensidades asignando 0 es negro y 255 es blanco.

Figura 2

Representación imagen escala grises

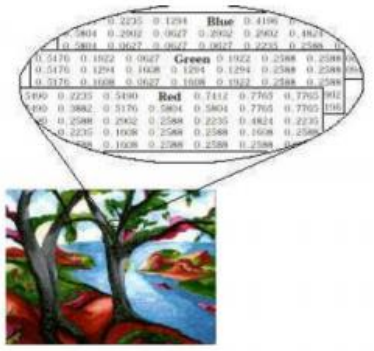


Fuente: Documento tesis SVA, Manso, R. M., 2014

Imágenes en color: son representadas por la combinación de tres colores, rojo, verde y azul, esta combinación es asignada para cada pixel, es decir la codificación es de 3 bytes, por lo cual se asigna un byte para cada color, el número de posibles combinaciones es 16 millones aproximadamente, cuando pasa esto la imagen se dice que es tipo multicanal. Existen también imágenes de cuatro canales, donde el cuarto canal mide el nivel de transparencia por cada pixel, estas son destinadas a aplicaciones de visión nocturna.

Figura 3

Representación color



Fuente: Documento tesis SVA, Manso, R. M., 2014

2.2.2 Operaciones comunes en imágenes

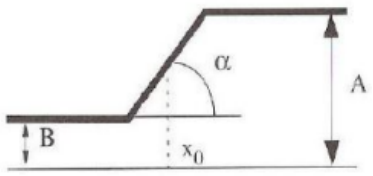
A continuación, se detallan algunas de las operaciones más usadas en imágenes las cuales son las siguientes:

2.2.2.1 Detección de contornos

Los contornos son un segmento de línea o ya sea de puntos que tienen la característica de ser continuos, es decir son como una frontera donde los niveles de gris son resaltantes. Esta funciona a la aplicación de máscaras, es decir son matrices que van filtrando cada pixel.

Figura 4

Niveles de gris



Fuente: Documento tesis SVA, Manso, R. M., 2014

En la imagen se muestran distintos niveles de grises, esto se debe a los cambios de intensidad de luminosidad, por ellos se emplean operaciones para contornos, los cuales eliminan estas dificultades como la existencia de sombras que se toman como bordes al igual que los cambios de textura, el uso de operadores más usados son los siguientes:

Canny: Detecta los bordes mediante las técnicas máscaras de convolución gaussiana, es se utiliza para suavizar la imagen que lee con el propósito de resaltar las secciones de la imagen donde se tenga derivadas espaciales, la dirección de la gradiente nos permite saber la dirección de los bordes, finalmente el operador realiza una supresión no máxima, el cual desplaza las crestas hasta los bordes por ende deja solo en cero los pixeles que no alcanzaron la parte superior de la cresta.

Robert: Detecta la posición más precisa, lleva todo el proceso al operador canny, pero con la diferencia de que este no necesita de un previo suavizado, por lo que es más sensible al ruido.

Prewitt: Calcula el valor de la gradiente, pero solo con dos núcleos, ya que el valor máximo es igual a un valor de salida.

Sobel: Calcula el valor que se obtiene de la gradiente a través de una diferencia de primer orden por medio de un valor absoluto.

2.2.2.2 Detección de rectas

El reconocimiento de patrones en una imagen permite visualizar la detección de formas como líneas, círculos, etc. El operador más usado es el siguiente:

Transformada de Hough: Detecta los puntos de la gradiente donde se visualiza cuáles son los cambios de intensidad más significativos, para ello es necesario s obtener una imagen binaria, por ejemplo, la aplicación más sencilla es la detección de una línea recta, el propósito es detectar si cada pixel forma parte de una línea, y encontrar todas las líneas forman parte del pixel.

2.3 Procesamiento de imágenes

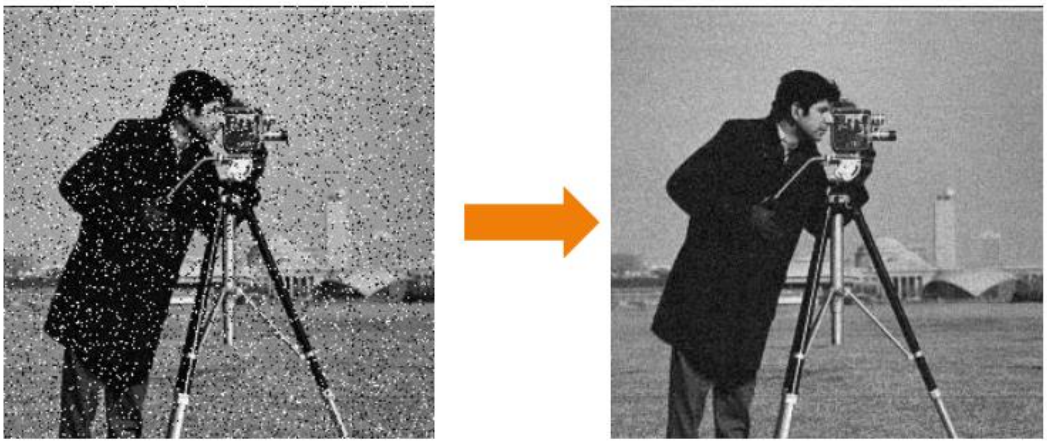
En algunas ocasiones las etapas para el procesado no se ejecutan ya que esto depende de los requerimientos necesarios para el usuario, las etapas son las siguientes:

Adquisición: los medios necesarios para obtener la captura de la imagen.

Preprocesado: se llevan a cabo técnicas de mejora para la mejora de procesamientos alternos o futuros, por ejemplos técnicas de eliminación de ruido, suavizados, realces, mejoras de contrastes.

Figura 5

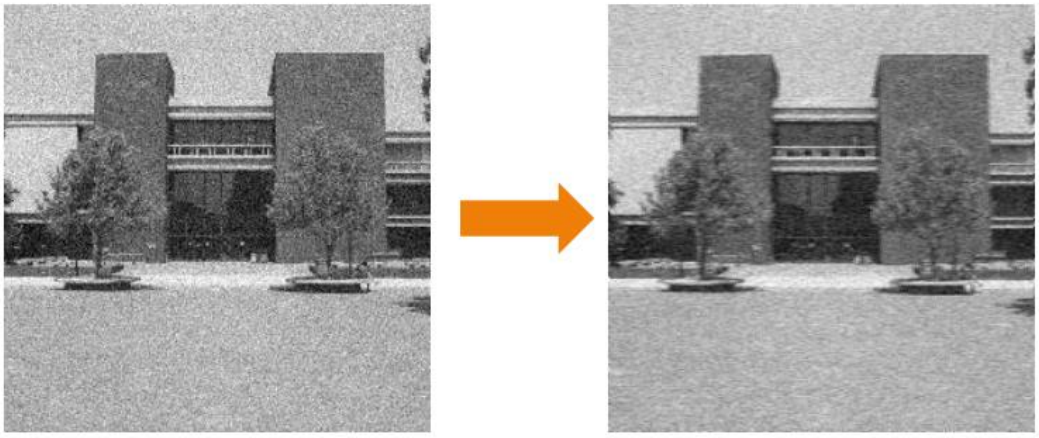
Representación eliminación de ruido



Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

Figura 6

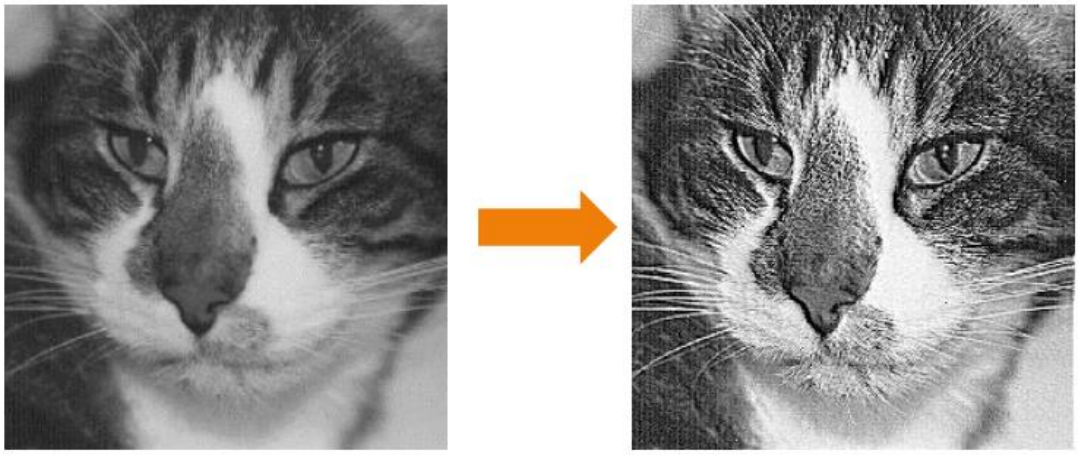
Representación eliminación de suavizado



Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

Figura 7

Representación eliminación de realce

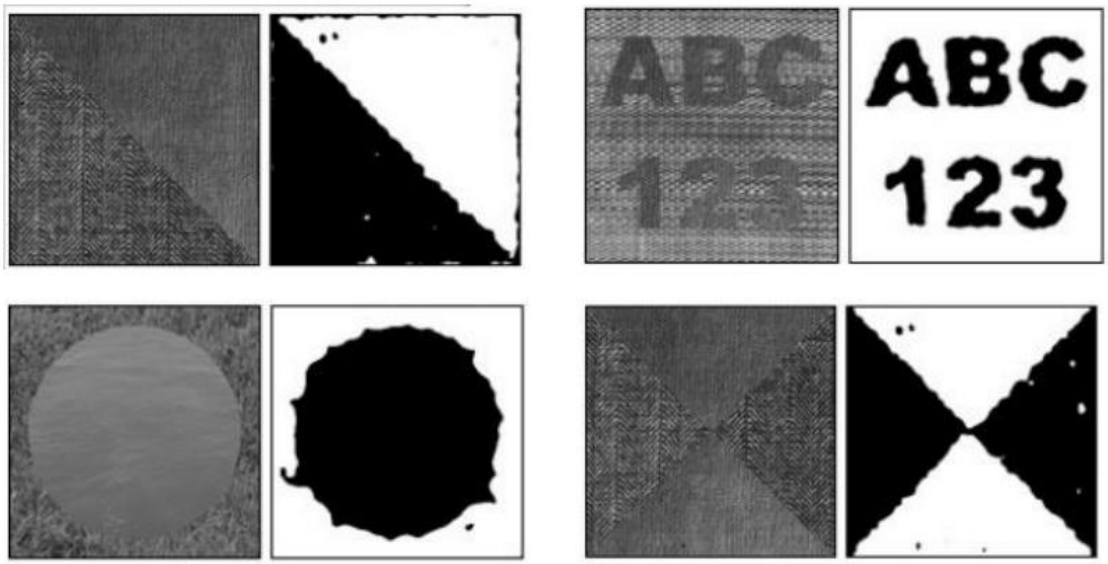


Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

Segmentación: se obtienen todas las secciones donde los pixeles comparten alguna característica determinada.

Figura 8

Representación de segmentación

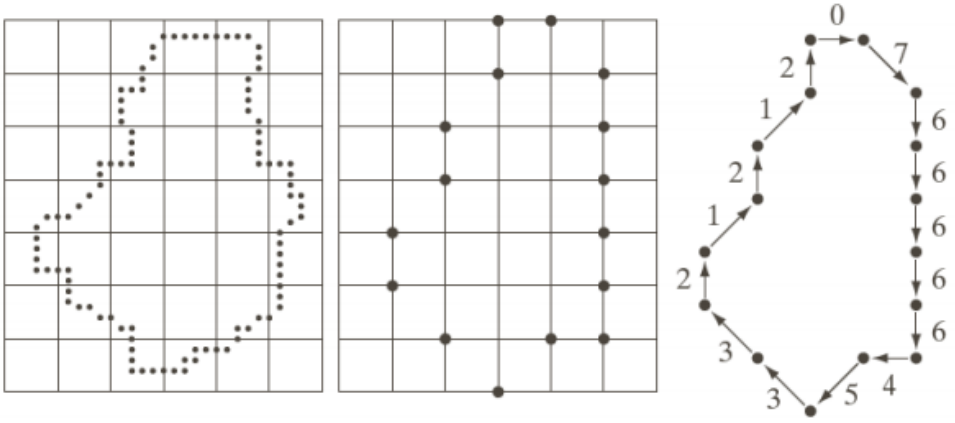


Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

Representación: se convierten los pixeles en una forma correcta para el procesamiento, los pixeles se pueden representar de distintas maneras por ejemplos círculos, líneas etc.

Figura 9

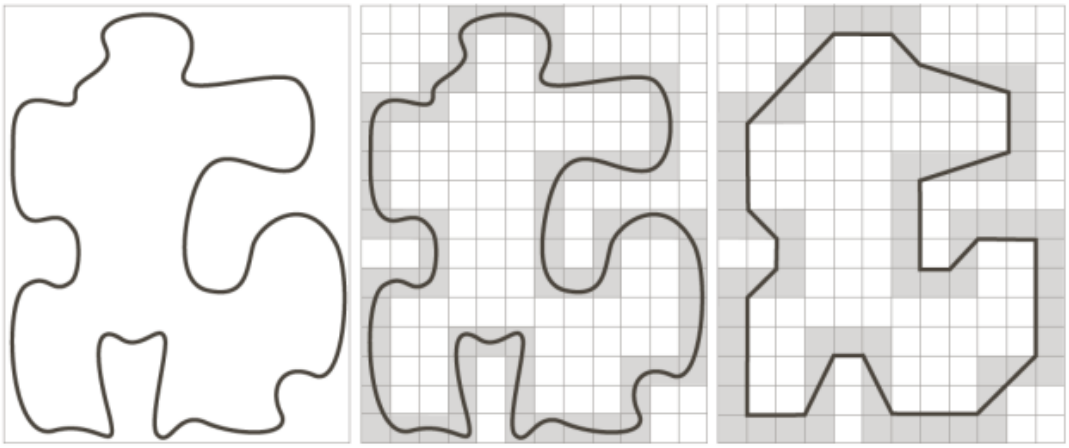
Representación localización de puntos



Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

Figura 10

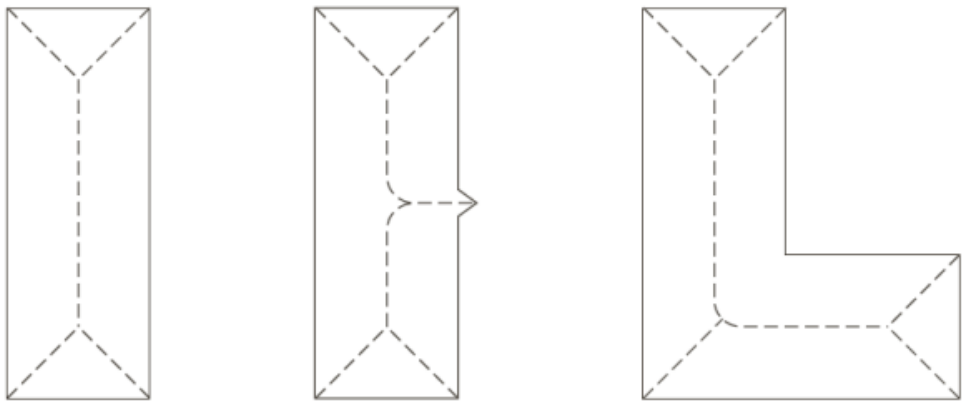
Representación localización de curvas



Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

Figura 11

Representación localización de rectas



Fuente: Documento educativo procesamiento de imágenes, Rodríguez, M. J., 2017

Descripción: se toman características de tipo cuantitativas con el propósito de marcar diferencias de tipos de clases.

Reconocimiento-Interpretación: se interpreta la imagen de los objetos ya procesados es decir los que ya fueron reconocidos, en esta etapa se puede reducir la cantidad de datos primordiales y eliminar datos que no son necesarios.

Figura 12

Representación eliminación de objetos innecesarios

Fuente: Documento educativo procesamiento de imagen, Rodríguez, M. J., 2017

2.4 Circuitos neumáticos

El uso de sistemas neumáticos se hace uso de aire comprimido para la transmisión de energía para poder funcionar, ya que se aplica aire comprimido al hacer uso de una fuerza esta se comprime y esta compresión se mantiene hasta que se devuelve la misma energía es decir se expande.

2.4.1 Solución de circuitos neumáticos

Para la solución de circuitos neumáticos existen diversos métodos los cuales son los siguientes:

Método intuitivo: La señal que se tiene al final de cada movimiento de cada pistón se marca al siguiente movimiento consecutivo. Este método es aplicado a circuitos que no tiene gran dificultad para interpretarse.

Método gracet: Este método se basa etapas donde se plasman a través de diagramas donde se visualiza las secuencias y duraciones, es decir el tiempo en son ejecutables.

Método tabla: El método se basa en lógica combinacional, es decir los valores de las señales de entrada y salida se basan en números de base binaria, para aplicación de este método es necesario que los elementos donde se conserva la memoria sean de tipo binario o biestables, es decir se conservan solo dos direcciones que se pueden leer.

Tabla 1

Transición de estados

Estado presente	Entrada presente {S1,1S1,1S2,2S1,2S2}			Salidas	
	11010	00110	00101	*B*	*A*
E1	E2	E1	E1	0	0
E2	E2	E3	E2	0	1
E3	E3	E3	E4	1	1
E4	E4	E1	E4	0	1

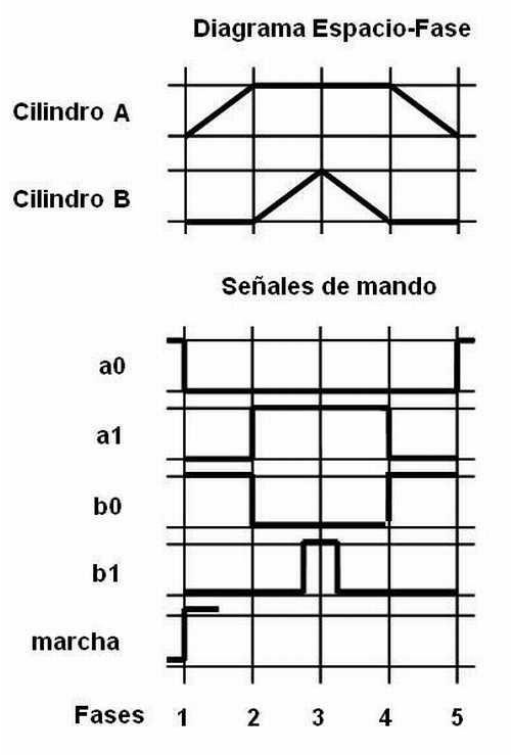
Fuente: Documento Tesis, Édison de Jesús Henao Castañeda, M. M., 2012

2.4.2 Diagrama estado-fase

Esta representación gráfica se visualiza el funcionamiento de un actuador con respecto a las fases y los cambios de estado con otros actuadores. Este se interpreta semejante a un plano cartesiano utilizando las coordenadas (x, y), es decir abscisas y ordenadas, para las primeras representan las fases, por el contrario, para las segundas son las acciones.

Figura 13

Representación gráfica estado-fase



Fuente: https://ikastaroak.ulhi.net/edu/es/PPFM/PSAFM/PSAFM02/es_PPFM_PSAFM02_Contenidos/website_421_representacin_grfica_diagramas_espaciofase.html, PSAFM02, 2020

Este diagrama representa la secuencia A+B+B-A-, el cual se tiene dos finales de carrera para cada actuador ya que solo son dos direcciones, A+ es asignado a1, para A- es a0, y de la misma manera funciona con B+ es b.

En un principio los dos pistones están contraídos, se inicia la secuencia de manera que cada final de carrera se activa, cuando los movimientos de compresión y expansión de los actuadores son censados.

CAPÍTULO III
METODOLOGÍA Y DESARROLLO

3.1 Alcance y enfoque de la investigación

- Visualización del diagrama estado-fase.
- Desarrollo de conteo de contornos en una imagen.
- Obtención de las gradientes en las coordenadas x-y para la convolución de la misma.
- Desarrollo de los límites de las coordenadas para llevar a cabo el reconocimiento de los estados lógicos.
- Obtención de filtros necesarios para el reconocimiento de pixeles necesarios.
- Construcción algoritmo para la lectura de la imagen entrante.
- Obtención de movimientos lógicos para una imagen de salida final.

3.2 Hipótesis

El uso de operadores de filtrado básicos para el tratamiento de imágenes proporciona la obtención de propiedades geométricas en un diagrama de estado-fase para la interpretación de secuencias neumáticas.

3.3 Diseño de la metodología

Para el desarrollo de la metodología se centra en las siguientes etapas

- Interpretación del diagrama de estado-fase.
- Determinación de la secuencia neumática.
- Uso de operaciones de tratamiento de imágenes.
- Selección de operaciones básicas en imágenes.
- Selección del método más adecuado para la lectura de contornos.
- Determinación de condicionamiento para la lectura de pixeles en una imagen.

3.4 Procedimiento

Para la construcción del algoritmo fue de vital importancia tomar en cuenta distintos aspectos neumáticos para la interpretación de un diagrama estado-fase.

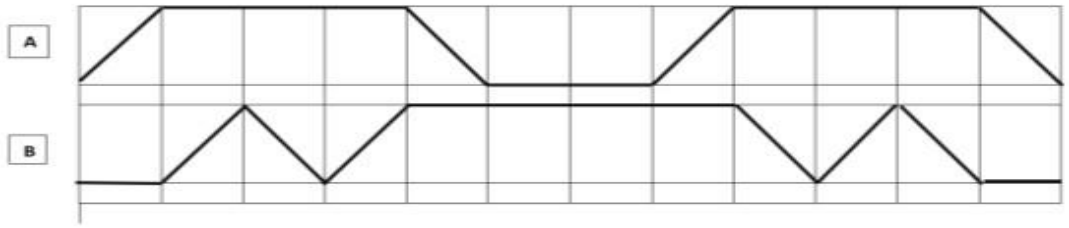
3.4.1 Análisis neumático

Diagrama estado-fase Esta representación gráfica se visualiza el funcionamiento de un actuador con respecto a las fases y los cambios de estado con otros actuadores.

Para el desarrollo del previo algoritmo, es primordial obtener la secuencia neumática para partir de este mismo, para ello se basó en el siguiente diagrama estado-fase mostrado en la siguiente imagen.

Figura 14

Diagrama estado fase



Fuente: Elaboración propia,2021

Podemos visualizar que en la figura existen dos cilindros en este caso lo llamaremos cilindro A y B, a continuación, la secuencia neumática es la siguiente:

Figura 15

Secuencia neumática

A+ B+ B- B+ A- A+ B- B+ B- A-

Fuente: Elaboración propia,2021

Estos movimientos correspondientes a los dos cilindros se plasmaron en un gráfico de sistema de mando el propósito de ello, para ello se utiliza una lógica binaria siendo las líneas horizontales de abajo indica que ese elemento de mando no está activado es decir que no está pulsado o detectado por lo que su estado lógico es cero en cambio la línea horizontal de arriba indica que ese elemento de mando si está activo (se encuentra pulsado es decir está detectando este mismo.

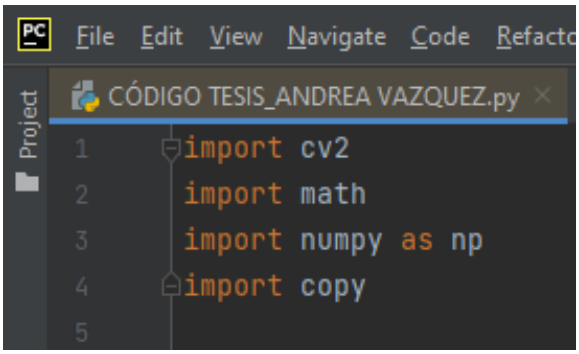
3.4.2 Construcción de algoritmo

Para la creación de código se detectó el número de cilindros que trabajarán para dicha secuencia en este caso son dos cilindros a través del operador canny se va a detectar el número de objetos, el proceso es el siguiente:

Importamos las siguientes librerías:

Figura 16

Importación de librerías



Fuente: Elaboración propia, 2021

EL primer paso es cargar la imagen con el comando cv2.imread("imagen1.jpg") este estará dentro de la ruta del archivo de la creación del proyecto.

El segundo comando cv2.imshow("original", original) se utiliza para mostrar la imagen en una ventana esta misma se ajusta al tamaño de la imagen.

Para la conversión a escala de grises en la imagen se guardó en una variable determinada con el siguiente formato "gris=cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)".

Figura 17

Lectura y conversión escala grises

```
# Lectura de la imagen
original = cv2.imread("imagen1.jpg")
cv2.imshow("original", original)

# Conversion escala de grises
gris = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)
```

Fuente: Elaboración propia, 2021

Se realiza el suavizado de la imagen a través de la operación de un filtro gaussiano donde el propósito es aplicar un filtro para el desenfoque de la imagen para convolucionar la imagen en el cual solo se utiliza el cambio de píxeles de alta frecuencia, la sintaxis es la siguiente:

Figura 18

Sintaxis filtro gaussiano

Fuente: https://hetpro-store.com/TUTORIALES/opencv-gaussianblur/, Marmolejo, D. R., 2021

Donde:

Matriz original: Matriz de Mat de n número de dimensiones.

Método destino: Matriz de mismas dimensiones igual la matriz Mat.

Valor impar: Es valor asignado para el ancho y el alto de la ventana para dimensionar el tamaño del Kernell Gaussiano.

Sigmas XY: Desviación estándar en las dos direcciones asignadas de los valores impares.

Tipo de borde: Método de extrapolación del filtro para la operación de desenfoque.

Figura 19

Aplicación filtro gaussiano

```
# Aplicar eliminación de ruido y filtro Gaussiano
gauss = cv2.GaussianBlur(gris, (5, 5), 0)
```

Fuente: Elaboración propia ,2021

La imagen que se muestra a continuación visualiza la ecuación que se genera por el filtro gaussiano

El filtro se guardó en la variable "gauss = cv2.GaussianBlur(gris, (5, 5), 0)", donde las sigmas para x-y son de tamaño cinco, también se utilizó la variable "gris" con el fin de usar la misma matriz de escala de grises.

Después de realizar todas las operaciones anteriores se procede a detectar los bordes con canny, la sintaxis es la siguiente:

Figura 20

Sintaxis canny

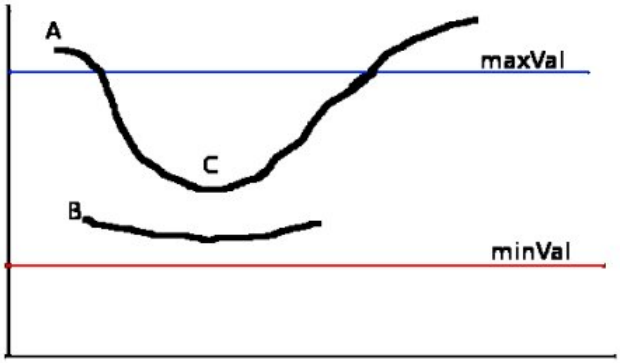
```
cv2.Canny(image, minVal, maxVal)
```

Fuente: https://likegeeks.com/es/procesar-de-imagenes-en-python/, Mokhtar Ebrahim, 2019

La imagen de entrada es a la que se le aplicó el borde gaussiano, los valores máximos-mínimos son los primeros y segundos umbrales como lo muestra en la imagen siguiente:

Figura 21

Clasificación de bordes



Fuente: https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html, CV, O., 2010

Lo que se pretende con esta operación es seleccionar cuales son los bordes que se requieren usar y cuáles no, por ello es el uso de valores máximos y mínimos, en este caso se le asigno un valor de 50 para el valor mínimo y al contrario para el valor máximo 150, es decir va a detectar cualquier borde de la gradiente mayor a 150 y aquellos inferiores al valor mínimo de 50 no los va a detectar.

Figura 22

Detección de bordes

```
17    # Detectamos los bordes con Canny
18    canny = cv2.Canny(gauss, 50, 150)
19    cv2.imshow("canny", canny)
```

Fuente: Elaboración propia, 2021

Con la siguiente operación se encuentran los contornos, donde se aplica el uso del método de recuperación, existen tres tipos: cv2.RETR_EXTERNAL, cv2.RETR_LIST, cv2.RETR_CCOMP y cv2.RETR_TREE. En nuestro caso haremos uso del primero, para ello se jerarquiza los contornos es decir que un contorno este dentro de otro y así sucesivamente, un contorno externo lo llamaremos "Padre" y por el contrario al interno "Hijo", por lo que el modo de recuperación selecciona todos los contornos solamente externos, es decir los que son mayores en dicha clasificación.

Figura 23

Detección de contornos

```
22  (contornos, _) = cv2.findContours(canny.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
23  print("Se detectaron {} pistones".format(len(contornos)))
```

Fuente: Elaboración propia, 2021

Aparte de la selección adecuada del método de recuperación, es necesario hacer uso del método de aproximación del contorno, en este caso existen dos: cv2.CHAIN_APPROX_NONE y cv2.CHAIN_APPROX_SIMPLE, la similitud en estos métodos es el almacenamiento de puntos encontrados respecto a cada contorno hallado, y la diferencia es la cantidad de puntos que se almacenan en los contornos por ello se selecciona la segunda metodología cv2.CHAIN_APPROX_SIMPLE donde guarda los puntos de los segmentos de los contornos generados, es decir suprime los segmentos verticales, horizontales y diagonales para descartar puntos redundante.

En la siguiente línea de código se dibujan los contornos, la sintaxis es la siguiente:

Figura 24

Sintaxis dibujos de contornos

```
cv2.drawContours(image, contours, contourIdx, color, thickness)
```

Fuente: https://docs.opencv.org/3.4/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc, CV, O., 2010

Donde

Contours: Proporciona los contornos en forma de lista

ContourIdx: Índice de los contornos.

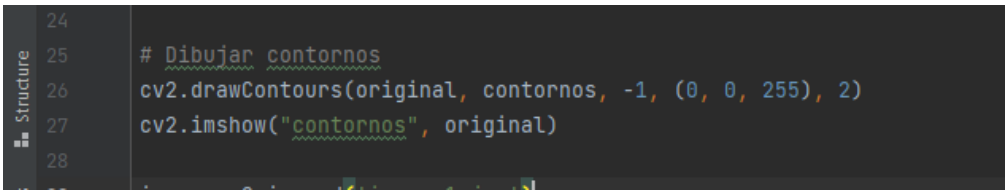
Color: Asignación de color para dibujar los contornos.

Thickness: Asignación para el grosor de la línea.

En este caso el índice de contornos especificado fue asignado -1 con el propósito de dibujar todos los contornos, para el color de los contornos fue rojo (0,0,255) y el grosor de dos.

Figura 25

Dibujo de contornos



Fuente: Elaboración propia, 2021

El siguiente proceso siguiente es localizar los estados lógicos, se hizo uso de puntos en cada vértice del diagrama, la lectura para cada pistón se asigna un color diferente en los vértices, en este caso tenemos dos pistones A y B, por ello se guardan en dos variables diferentes cada una con un color distinto haciendo uso de la librería Numpy de Python donde una de sus herramientas es proporcionar arreglos de tipo estático y homogéneo, en este caso se guardó el color en un arreglo (uno para cada color) la lógica es la siguiente:

$$\boldsymbol{Número\ de\ colores\ guardados = Número\ \ pistones}$$

El arreglo donde se almaceno son arreglos de tipos entero sin que contenga negativos, el intervalo va de 0 a 255.

Para el caso del color azul que es A, son asignados dos arreglos para los rangos dos rangos de color que pueda detectar el vértice, para el color rojo son cuatro rangos de colores.

Figura 26

Asignación de colores

```
322
323     azulBajo = np.array([100, 100, 29], np.uint8)
324     azulAlto = np.array([125, 255, 255], np.uint8)
325
326     rojoBajo1 = np.array([0, 100, 20], np.uint8)
327     rojoAlto1 = np.array([10, 255, 255], np.uint8)
328     rojoBajo2 = np.array([175, 100, 20], np.uint8)
329     rojoAlto2 = np.array([180, 255, 255], np.uint8)
330
```

Fuente: Elaboración propia, 2021

Posterior a lo antes mencionado se detectan los colores, se lee la imagen y se guarda en la variable "HSV" para después detectar los colores.

Para las siguientes líneas se guardan los rangos de los colores de los arreglos que fueron guardadas anteriormente con cv2.Range

Figura 27

Detección de colores

```
330
331     imagen = cv2.imread('qw.jpg')
332     imagenHSV = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)
333
334     #Detectando colores
335     maskazul = cv2.inRange(imagenHSV, azulBajo, azulAlto)
336     maskRojo1 = cv2.inRange(imagenHSV, rojoBajo1, rojoAlto1)
337     maskRojo2 = cv2.inRange(imagenHSV, rojoBajo2, rojoAlto2)
338     maskRojo =  cv2.add(maskRojo1, maskRojo2)
339
```

Fuente: Elaboración propia, 2021

Después se localizan los contornos con el método de recuperación, existen tres tipos: cv2.RETR_EXTERNAL y con el método de aproximación cv2.CHAIN_APPROX_SIMPLE.

Figura 28

Detección de contornos

```
344     contornosazul = cv2.findContours(maskazul, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
345     contornosRojo = cv2.findContours(maskRojo, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
```

Fuente: Elaboración propia, 2021

Para ejecutar el etiquetado con los vértices, se elaboró una función con el nombre de "dibujarcontorno", se hizo uso de cv2.FONT_HERSHEY_COMPLEX donde se especifica el tipo de letra.

Tabla 2

Tipo de letra

cv2.FONT_HERSHEY_SIMPLEX = 0
cv2.FONT_HERSHEY_PLAIN = 1
cv2.FONT_HERSHEY_DUPLEX = 2
cv2.FONT_HERSHEY_COMPLEX = 3
cv2.FONT_HERSHEY_TRIPLEX = 4
cv2.FONT_HERSHEY_COMPLEX_SMALL = 5
cv2.FONT_HERSHEY_SCRIPT_SIMPLEX = 6
cv2.FONT_HERSHEY_SCRIPT_COMPLEX = 7
cv2.FONT_ITALIC = 16

Fuente: https://noemioocc.github.io/posts/Añadir-texto-a-una-imagen-openCV-python/, Mimi, 2021

En la función se manejan dos variables a leer (contornos y color), dentro de esta misma tenemos un ciclo for con el propósito de recorrer cada contorno, en las líneas siguientes se utiliza los momentos que genera cada contorno tanto en X como en Y, se calculan las coordenadas de los momentos obtenidos, se colocan las etiquetas (texto) de los movimientos de los pistones A+A- etc.

Figura 29

Localización de centroides

```
font = cv2.FONT_HERSHEY_COMPLEX
def dibujarContorno(contornos, color):
    for (i, c) in enumerate(contornos):
        M = cv2.moments(c)
        if (M["m00"]==0): M["m00"]==1
        x = int(M["m10"]/M["m00"])
        y = int(M["m01"]/M["m00"])

        cv2.drawContours(imagen, [c], 0, color, 2)
        #cv2.putText(imagen, str(i + 1), (x - 10, y + 10), 1, 2, (0, 0, 0), 2)
        cv2.putText(imagen, str(x), (x-20,y+8), 1, 0.6,(0,0,0),1)
        cv2.putText(imagen, str(y), (x - 20, y -5), 1, 0.6, (0, 0, 0), 1)
        cv2.putText(imagen,str(len(contornosRojo)),(30,160), 1, 2,(0,0,0),2)
```

Fuente: Elaboración propia, 2021

En la variable g se suman los contornos, en la variable "g" se guardan los vértices detectados, y dentro de cada if (asignado un vértice) se encuentra otro if para asignar límites de pixeles en el eje y, con ello se colocan el texto (el movimiento del pistón) (x+20, y+20) es la ubicación donde se va a colocar el texto.

Figura 30

Enumeración de vértices

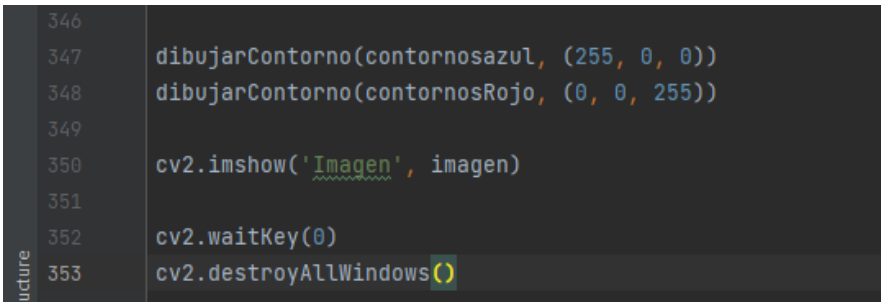
```
        g = len(contornosRojo)
        if g == 1:
            if y < 50:
                f = cv2.putText(imagen, 'A+', (x + 20, y + 20), 1, 1, (0, 0, 0), 2)
                print(f)
            if y < 300:
                if y > 200:
                    e = cv2.putText(imagen, 'A-', (x + 20, y + 10), 1, 1, (0, 0, 0), 2)
                    print(e)
            if y > 300:
                if y < 400:
                    e = cv2.putText(imagen, 'B+', (x + 20, y + 10), 1, 1, (0, 0, 0), 2)
                    print(e)
            if y > 500:
                e = cv2.putText(imagen, 'B-', (x + 20, y + 10), 1, 1, (0, 0, 0), 2)
                print(e)
        if g == 2:
            if y < 50:
                f = cv2.putText(imagen, 'A+', (x + 20, y + 20), 1, 1, (0, 0, 0), 2)
                print(f)
            if y < 300:
                if y > 200:
                    e = cv2.putText(imagen, 'A-', (x + 20, y + 10), 1, 1, (0, 0, 0), 2)
                    print(e)
            if y > 300:
                if y < 400:
                    e = cv2.putText(imagen, 'B+', (x + 20, y + 10), 1, 1, (0, 0, 0), 2)
```

Fuente: Elaboración propia, 2021

Finalmente se manda a llamar la función evaluando a los contornos encontrados.

Figura 31

Ejecución función



```
346
347  dibujarContorno(contornosazul, (255, 0, 0))
348  dibujarContorno(contornosRojo, (0, 0, 255))
349
350  cv2.imshow('Imagen', imagen)
351
352  cv2.waitKey(0)
353  cv2.destroyAllWindows()
```

Fuente: Elaboración propia, 2021

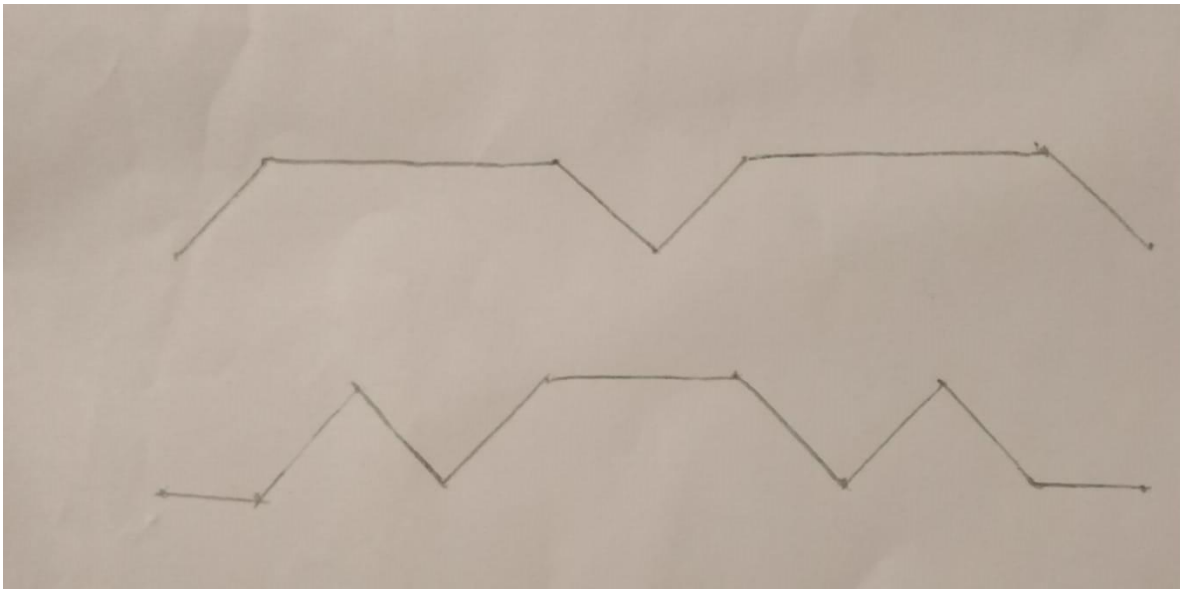
CAPÍTULO IV
RESULTADOS

4.1 Resultados lectura diagrama estado-fase

A continuación, se muestra el diagrama captado por la cámara la cual es segmentada, leída para llevar a cabo el proceso de filtrado.

Figura 32

Imagen de entrada

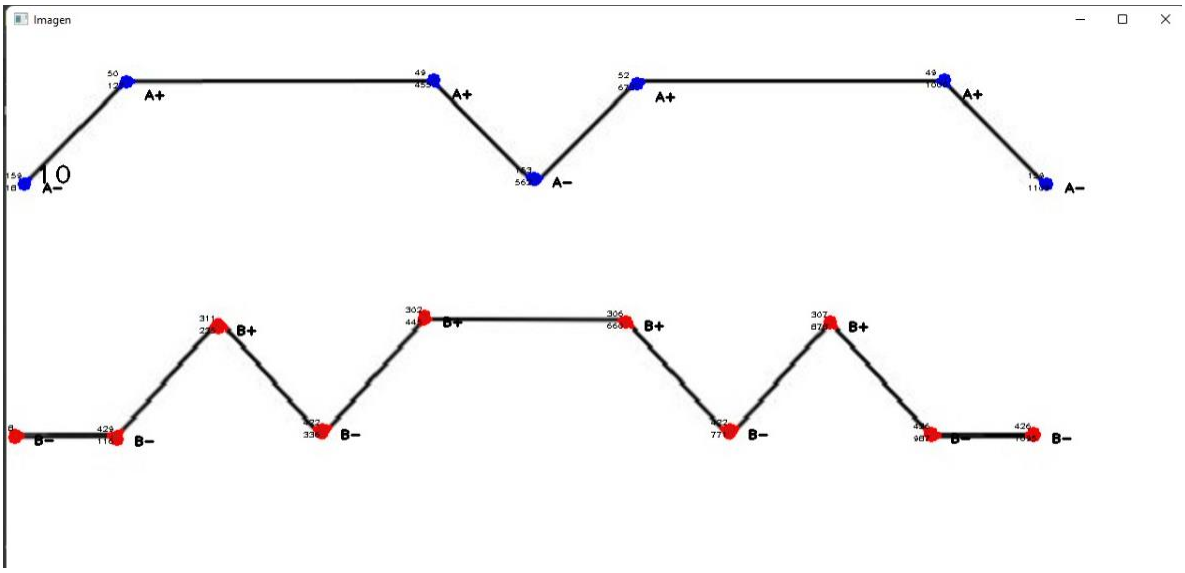


Fuente: Elaboración propia, 2021

Se muestra las distintas operaciones de filtrado para el tratamiento de la imagen ejecutadas en el software PyCharm, cada operación se muestra en las siguientes imágenes presentadas.

Figura 33

Resultado lectura de diagrama 1 principal



Fuente: Elaboración propia, 2021

Figura 34

Resultado lectura de secuencia y número de pistones diagrama 1

```
Se detectaron 2 pistones
A- B- A+  B- B+ B- A+ B+ A- A+ B+ B- B+ A+ B- B-

Process finished with exit code 0
```

Fuente: Elaboración propia, 2021

El procedimiento para la lectura del diagrama estado-fase es la siguiente:

- Lectura de la imagen original (es decir la imagen que no contiene alguna operación de filtrado).
- Aplicación de filtrado gaussiano-eliminación de ruido.
- Detección de bordes con el operador canny.
- Detección de contorno (muestra de número de objetos, es decir número de pistones).
- Dibujo de contornos.

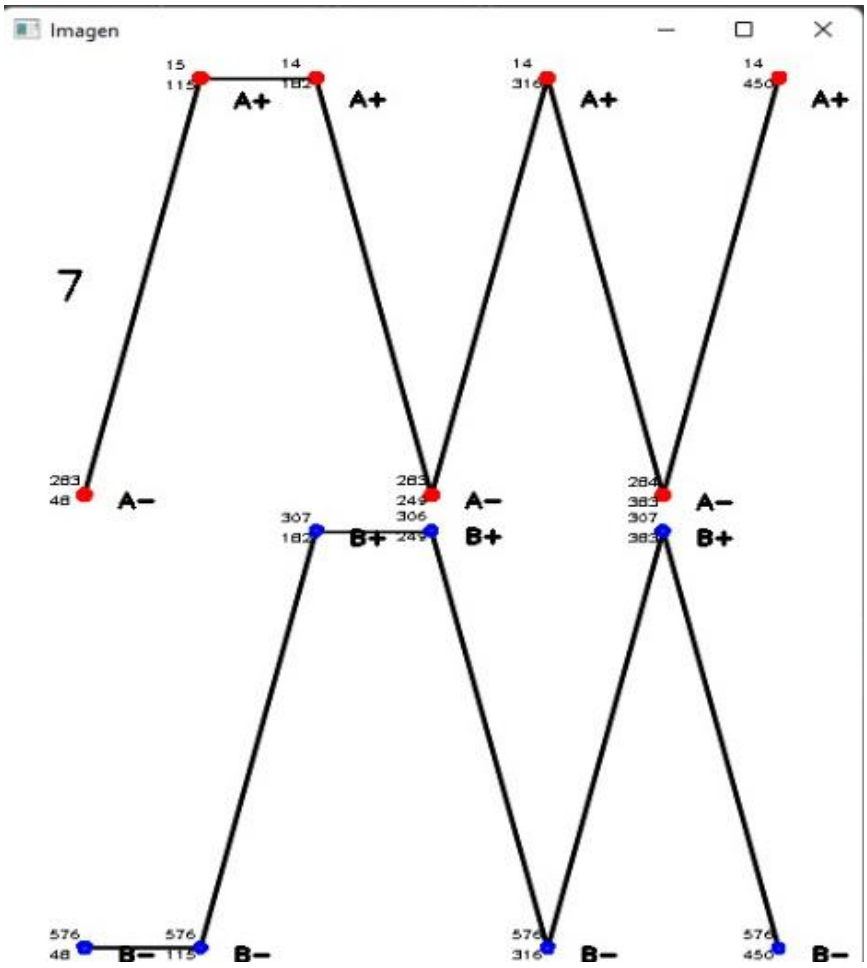
- Muestra de resultados.
- Lectura de imagen con los vértices de los contornos mostrados.
- Lectura de rangos de colores aceptados para la lectura.
- Lectura de los límites de los pixeles para el etiquetado de movimiento de estados lógicos.

A continuación, se presenta de otras muestras con distintas secuencias

La presentación de la imagen mostrada a continuación visualiza el número de segmentos definido, en la terminal va a mostrar el número de pistones es decir el número de contornos encontrados y la secuencia de los pistones encontrados.

Figura 35

Resultado lectura de diagrama 2



Fuente: Elaboración propia, 2021

Figura 36

Resultado lectura de secuencia y número de pistones diagrama 2

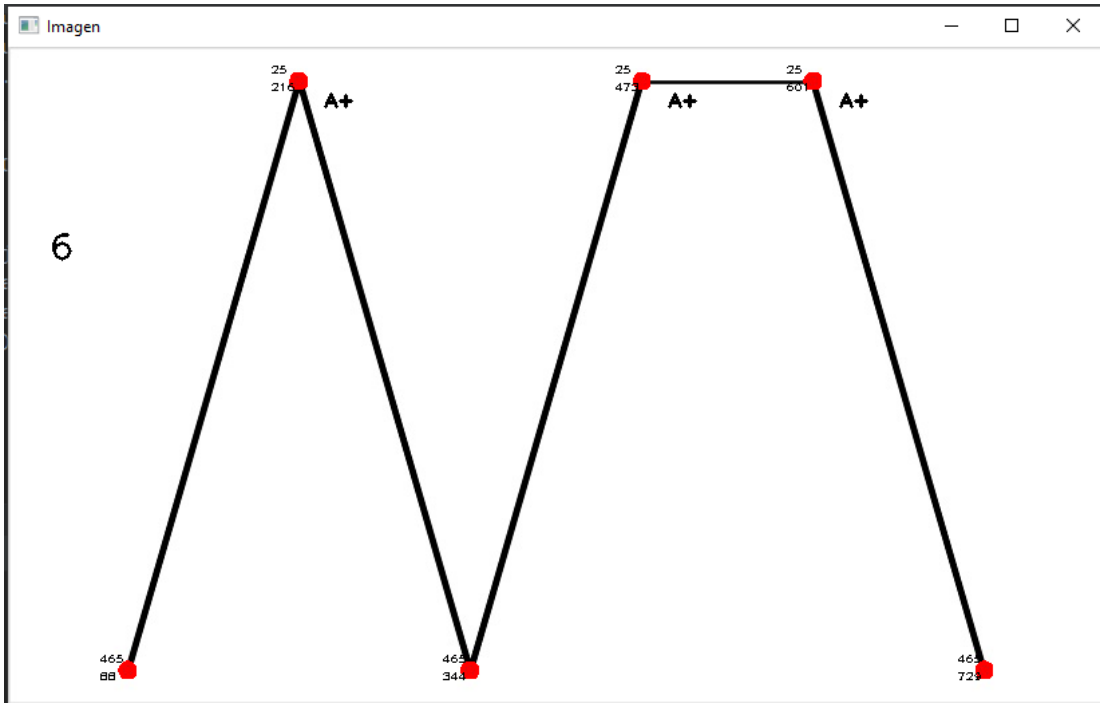
```
Se detectaron 2 pistones
A- B- A+ B- A+ B+ A- B+ A+ B- A- B+ A+ B-

Process finished with exit code 0
```

Fuente: Elaboración propia, 2021

Figura 37

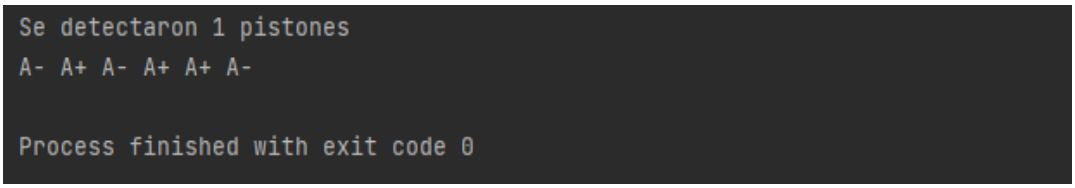
Resultado lectura de diagrama 3



Fuente: Elaboración propia, 2021

Figura 38

Resultado de lectura de secuencia y numero de pistones diagrama 3



Fuente: Elaboración propia, 2021

CAPÍTULO V
CONCLUSIONES

5.1 Conclusiones

- La identificación del número de pistones se realizó a través de la lógica de detectar el número de objetos, es decir mediante la técnica de detección de contornos, seleccionando el operador más adecuado, teniendo en cuenta diversos parámetros para descartar las condiciones innecesarias para el filtrado de la imagen seleccionando el operador canny.
- El método de recuperación más apto se seleccionó a través de la jerarquización de contornos descartando contornos internos.
- Para descartar puntos redundantes que se almacenan en contornos se hizo uso del método de aproximación del contorno.
- La lectura de la imagen de los puntos verticales de los contornos es mediante la vectorización de colores.
- El etiquetado en texto de la imagen se llevó a cabo con la localización de los pixeles dentro de los rangos seleccionados.

5.2 Recomendaciones

Para llevar a cabo el reconocimiento de la imagen

- Para los vértices es necesario incluir más colores dentro de los arreglos y declarar más pistones dentro de un diagrama estado-fase.
- Modificar los rangos de lectura de pixeles en el eje Y de los momentos generados dentro del ciclo for para incluir la segmentación de imágenes con resolución menor a 592 pixeles.

5.3 Trabajo a futuro

Se espera que el algoritmo construido más adelante se adapte para el funcionamiento e implementación en un PLC a través de posibles conexiones seriales haciendo uso de librerías de Python para así poder convertir los datos de tipo string del etiquetado de los vértices (estados lógicos) a datos de tipo binario para que pueda procesar esta información en comunicación serial.

CAPÍTULO VI

COMPETENCIAS DESARROLLADAS

- Desarrollo e interpretación de técnicas de reconocimiento de objetos en una imagen.
- Selección de métodos de recuperación y jerarquización de contornos en una imagen dependiendo de los contornos a utilizar.
- Selección del operador más apto para el reconocimiento de contorno con relación a los pixeles de filtrado procesados anteriormente.
- Desarrollo de técnicas para el reconocimiento de vértices en contornos.
- Determina los parámetros necesarios para el etiquetado de estados lógicos en un pistón.
- Adaptación de métodos de lectura de pixeles en una imagen.

CAPÍTULO VII
ANEXOS

CARTA DE AUTORIZACIÓN DEL(LA) AUTOR(A) PARA LA CONSULTA Y PUBLICACIÓN ELECTRÓNICA DEL TRABAJO DE INVESTIGACIÓN

El que suscribe:

ANDREA	VÁZQUEZ	CATARINO

Con Número de Control	**17TE0170**
Perteneciente al Programa Educativo	**INGENIERÍA MECATRÓNICA**

Por este conducto me permito informar que he dado mi autorización para la consulta y publicación electrónica del trabajo de investigación en los repositorios académicos.

Registrado con el producto:	**TESIS**

Cuyo Tema es:

DEASRRROLLO DE ALGORITMO APLICANDO TÉCNICAS DE VISIÓN ARTIFICIAL PARA IDENTIFICAR LOS MOVIMIENTOS EN UN DIAGRAMA ESTADO-FASE PARA LA PROGRAMACIÓN DE UN PLC

Correspondiente al periodo:

AGOSTO 2021-MARZO 2022

Y cuyo(a) director(a) de tesis es:

M.S.C. ALFREDO CARRASCO ARAOZ

ATENTAMENTE

ANDREA VÁZQUEZ CATARINO

Nombre y firma

Fecha de emisión: **31/03/2022**

c.c.p. Subdirección Académica

CAPÍTULO VIII
FUENTES DE INFORMACIÓN

7.1 Bibliografía

- Blanca Arlette Serrato Espino, M. d. (octubre de 2017). Dimensionamiento de una oficina mediante un sistema de visión utilizando los patrones cuadrados ubicados en los techos de tabla roca . Celaya.
- CV, O. (enero de 2010). *Open source computer vision*. Obtenido de https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html
- CV, O. (2010). *Open source computer vision*. Obtenido de https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a
- CV, O. (2010). *Open-source computer vision*. Obtenido de https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html
- CV, O. (2010). *Open-source computer vision*. Obtenido de https://docs.opencv.org/3.4/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc
- Édison de Jesús Henao Castañeda, M. M. (agosto de 2012). Método tabla de secuencias para la solución de circuitos neumáticos e hidráulicos. Maracaibo, Colombia.
- Enrique Alegre, G. P. (junio de 2016). Conceptos y métodos en visión por computador . Barcelona, España.
- Hernandez Sampieri, R. (2014). *Metodología de la investigación.* México: McGRAW-HILL.

- Holzer, R. (febrero de 2020). OpenCV tutorial Documentation.
- elo VDI2221 frente al modelo metodológicoI+P+D3. *Revista Espacios*, 22.
- Mimi. (octubre de 2021). Kipunaec Electrónica, matemática Ecuador. Obtenido de https://noemioocc.github.io/posts/Añadir-texto-a-una-imagen-openCV-python/
- klette, R. (2014). Concise computer vision. Londres.
- Lathiya, K. (3 de noviembre de 2021). *AppDividend*. Obtenido de https://appdividend.com/2020/09/09/python-cv2-image-size-how-to-get-image-size-in-python/
- Manso, R. M. (julio de 2014). Sistema de visión artificial para la detección y lectura de matrículas. Valladolid.
- Marmolejo, D. R. (2021). *Hetpro*. Obtenido de https://hetpro-store.com/TUTORIALES/opencv-findcontours/
- Marmolejo, D. R. (2021). *Hetpro*. Obtenido de https://hetpro-store.com/TUTORIALES/opencv-gaussianblur/
- Marín, R. (12 de febrero de 2020). Revista digital INESEM. Obtenido de https://revistadigital.inesem.es/informatica-y-tics/opencv/
- PSAFM02. (2020). Tecnología neumática. Obtenido de https://ikastaroak.ulhi.net/edu/es/PPFM/PSAFM/PSAFM02/es_PPFM_PSAFM02_Contenidos/website_index.html
- Rafael C. Gonzalez, R. E. (2008). *Digital Image Processing.* Pearson Education.
- Rodríguez, M. J. (2017). Procesamiento de imágenes digitales. Sevilla, España.

- Sebastian Amaya Zapata, D. P. (enero de 2016). Desarrollo e implementación de un sistema de visión artificial basado en lenguajes de uso libre para un sistema seleccionador de productos de un centro integrado de manufactura. Medellin , Colombia.
- Unipython. (5 de abril de 2018). *Unipython*. Obtenido de https://unipython.com/propiedades-los-contornos/