UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO DE UN ALGORITMO DE PROCESAMIENTO DE IMÁGENES DEL SISTEMA DE PESAJE PARA EL CONTROL AUTOMÁTICO DE UNA FAJA TRANSPORTADORA EN LA UNIDAD MINERA MALLAY

TESIS

PRESENTADA POR:

RODOLFO CHINO CATARI

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PUNO – PERÚ

2019

UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

DISEÑO DE UN ALGORITMO DE PROCESAMIENTO DE IMÁGENES DEL SISTEMA DE PESAJE PARA EL CONTROL AUTOMÁTICO DE UNA FAJA TRANSPORTADORA EN LA UNIDAD MINERA MALLAY

TESIS PRESENTADA POR:

RODOLFO CHINO CATARI

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

APROVADO POR EL JURADO REVISOR CONFORMADO POR:

PRESIDENTE : .....
Dr. IVAN DELGADO HUAYTA

PRIMER MIEMBRO : .....
Dr. MIDWAR ELIAS VALENCIA VILCA

SEGUNDO MIEMBRO : .....
M.Sc. DAVID SALINAS MENDOZA

DIRECTOR/ASESOR : .....
Dr. MARCO ANTONIO QUISPE BARRA

Área : Control y Automatización
Tema : Reconocimiento de Imágenes

FECHA DE SUSTENTACION 03 DE JULIO DEL 2019

DEDICATORIA

Con infinito amor y gratitud a mis queridos padres Domingo G. Chino M. y María Catari R, a quienes admiro por sus ejemplos de honestidad y fortaleza, que alienta siempre mi corazón, por su inmenso amor, paciencia, esfuerzo, sacrificio que permitió la culminación de mi carrera profesional.

A mis queridos hermanos Oscar, Ever, Alex, Susan y Zoraida, quienes siempre me apoyaron de manera incondicional y entereza, por sus invaluables consejos que fortalecieron mi alma y encaminaron mi vida, hacia la culminación de mis estudios y poder llegar a la cima del éxito.

A mis tíos Guillermo, Lucio, María, por su gran apoyo incondicional, que a pesar de las dificultades que pasaron en la vida, son los que me motivaron y me inculcaron el espíritu de lucha y perseverancia para lograr mis metas.

Amigos y compañeros, por los gratos momentos compartidos durante mi vida universitaria.

"La única forma de aprender es seguir intentando desarrollar objetivos y buscar una alternativa de solución"

Rodolfo Chino Catari

AGRADECIMIENTOS

Agradezco a Dios por cuidarme, darme la fuerza y guiarme al camino correcto, de darme la oportunidad de vivir en este mundo maravilloso durante mi formación profesional.

A mi Alma Mater la Universidad Nacional del Altiplano Puno, por haberme brindado la oportunidad de forjarme como profesional. Lugar, en donde construí mi proyecto de vida.

A los docentes de la escuela profesional de ingeniería electrónica que me inculcaron, enseñaron y compartieron sus experiencias para afrontar los desafíos de la vida, por sus consejos y sugerencias en mi investigación de tesis y formación profesional.

A mis jefes y compañeros de trabajo por compartir sus experiencias laborales y enseñanzas que contribuyen mi formación profesional.

Finalmente, mi entero agradecimiento a mis queridos padres, tíos, hermanos y amigos que me enseñaron ser perseverante y por su apoyo incondicional que me brindaron en el mejor momento que solicité.

ÍNDICE GENERAL

Resumen 12

Abstract 13

CAPITULO I 14

1.1. Introducción 14

1.2. Planetamiento del Problema 15

1.3. Formulación del Problema 15

1.3.1. Problemas Específicos 15

1.4. Hipótesis de la Investigación 16

1.4.1. Hipótesis General 16

1.4.2. Hipótesis Específicas 16

1.5. Objetivos de la Investigación 16

1.5.1. Objetivo General 16

1.5.2. Objetivos Específicos 16

CAPITULO II 17

Revisión de Literatura 17

2.1. Antecedentes del Proyecto 17

2.2. Marco Teórico 22

2.2.1. Lenguaje de Programación 22

2.2.2. Python 24

2.2.3. Opencv 26

2.2.4. Algoritmo 27

2.2.5. Procesamiento de Imágenes 28

2.2.6. Faja Transportadora 30

2.3. Visión por Computadora 31

2.3.1. Elementos del sistema de Visión por Computadora 33

2.3.2. Pixel 36

2.3.3. Espacios de Color 38
2.4. Transformaciones morfológicas en OpenCV 40
2.4.1. Dilatación 40
2.4.2. Erosión 41
2.4.3. Apertura y Cierre 41
2.5. Bordes 42
2.5.1. Detección de Bordes 43
2.5.2. Detección de borde Canny 44
2.6. Numpy 45
2.6.1. Usando Numpy 46
2.7. Matplotlib 47
2.8. Umbral o Delimitado 49
2.8.1. Umbral Adaptativo 50
2.9. Histograma 51
CAPITULO III 56
Materiales y Métodos 56
3.1. Materiales 56
3.1.1. Hardware 56
3.1.2. Software. 56
3.2. Diseño, nivel y tipo de la Investigación 56
3.2.1. Diseño de la Investigación 56
3.2.2. Nivel de la Investigación 57
3.3. Población y muestra de la Investigación 57
3.3.1. Población 57
3.3.2. Muestra 57
3.4. Ubicación y descripción de la Investigación 57
3.4.1. Ubicación 57

3.4.2. Descripción de la Investigación 58
3.5. Técnicas e instrumentos de recoleccion de Datos 59
3.5.1. Técnicas 59
3.6. Desarrollo del Algoritmo 59
3.6.1. Adquisición de las Imágenes 59
3.6.2. Imagen para la primera muestra 60
3.6.3. Imagen para la segunda muestra 78
3.7. Diagrama de bloques del Algoritmo 96
3.7.1. Descripcion del diagrama de bloques 97
CAPITULO IV 98
Resultados y Discusión 98
4.1. Resultados obtenidos en el experimento 98
4.1.1. Muestra 1 98
4.1.2. Muestra 2 100
4.1.3. Muestra 3 101
4.1.4. Muestra 4 103
4.2. Discusión 106
CONCLUSIONES 108
RECOMENDACIONES 110
REFERENCIAS BIBLIOGRÁFICAS 111
ANEXOS 115

ÍNDICE DE FIGURAS

Figura 2.1: Espacios de Color 39
Figura 2.2: Imagen Binario antes (izquierda) y después de la dilatación (derecha)..... 40
Figura 2.3: Binario antes (Izquierda) y después de Erosión (Derecha)..... 41
Figura 2.4: Apertura y Cierre para obtener resultados requeridos 42
Figura 2.5: Detección de Bordes 43
Figura 2.6: Detección de bordes Canny 45
Figura 2.7: Grafica generado por las líneas de código 49
Figura 2.8: Tipos de Umbral 50
Figura 2.9: Comparativa de Umbral Binario y Adaptativo 51
Figura 2.10: Ilustración de ecualización de histogramas..... 53
Figura 2.11: Histograma RGB..... 55
Figura 3.1: Adaptación de la cámara a la faja transportadora 60
Figura 3.2: Imagen para la primera muestra..... 60
Figura 3.3: Gráfica del Histograma 62
Figura 3.4: Reducción de la Imagen..... 62
Figura 3.5: Conversión de la imagen RGB a Grises 63
Figura 3.6: Gráfica del histograma de la imagen en grises 63
Figura 3.7: Ecualización de la imagen Recortada 64
Figura 3.8: Gráfica del histograma de imagen ecualizada 64
Figura 3.9: Gráfica del histograma de la imagen recortada..... 65
Figura 3.10: Imagen Recortada nuevamente 65
Figura 3.11: Gráfica del histograma de la imagen recortada..... 66
Figura 3.12: Imagen Recortada y ecualizada 66
Figura 3.13: Detección de Bordes Canny..... 67
Figura 3.14: Detección de Bordes Sobel 67
Figura 3.15: Threshold a la detección de bordes Sobel..... 68
Figura 3.16: Sumatoria en el eje horizontal..... 68
Figura 3.17: Identificación de la faja transportadora..... 69
Figura 3.18: Imagen recortada desde la posición de la faja 69
Figura 3.19: Imagen recuperada 70
Figura 3.20: Suavizado de la imagen 70
Figura 3.21: La imagen es invertida para hallar el fondo..... 71

Figura 3.22: Imagen invertida y ecualizada 71
Figura 3.23: Imagen con Threshold..... 72
Figura 3.24: Imagen con Threshold adaptativo 72
Figura 3.25: Imagen Erosionado 73
Figura 3.26: Cierre de las aperturas de la imagen 73
Figura 3.27: Dilatación de fondo de la imagen 74
Figura 3.28: Recorte de la imagen Invertida 74
Figura 3.29: Erosión de la imagen de fondo..... 75
Figura 3.30: Recorte de la imagen no requerida..... 75
Figura 3.31: Erosión de la imagen recortada..... 75
Figura 3.32: Dilatación de la imagen recortada..... 76
Figura 3.33: Imagen contorneada 76
Figura 3.34: Mineral encontrado 78
Figura 3.35: Imagen de la Segunda Muestra 79
Figura 3.36: Histograma de la Imagen 80
Figura 3.37: Imagen reducida 8 veces del original..... 80
Figura 3.38: Imagen convertida a escala grises y reducida 8 veces 81
Figura 3.39: Histograma de la imagen a escala grises..... 81
Figura 3.40: Imagen ecualizada y reducida 8 veces 82
Figura 3.41: Histograma de la imagen ecualizada en escala grises..... 82
Figura 3.42: Histograma de la imagen recortada..... 83
Figura 3.43: Imagen Recortada 83
Figura 3.44: Histograma de la imagen recortada..... 84
Figura 3.45: Imagen recortada ecualizada..... 84
Figura 3.46: Detección de bordes método Canny 85
Figura 3.47: Detección de bordes Sobel..... 85
Figura 3.48: Umbralización de la detección de borde Sobel..... 86
Figura 3.49: Sumatoria en el eje horizontal..... 86
Figura 3.50: Imagen con identificación de la línea de faja..... 87
Figura 3.51: Imagen recortada parte inferior..... 87
Figura 3.52: Imagen recuperada 88
Figura 3.53: Imagen recuperada y suavizada 88
Figura 3.54: Imagen invertida para encontrar fondo 88
Figura 3.55: Imagen ecualizada suavizada 89

Figura 3.56: Imagen threshold 89
Figura 3.57: Imagen threshold adaptativo 90
Figura 3.58: Imagen erosionada 90
Figura 3.59: Cierre de las aperturas de la imagen 91
Figura 3.60: Dilatación del fondo de la imagen 91
Figura 3.61: Recortado de la imagen invertida 92
Figura 3.62: Fondo de la Imagen erosionada 92
Figura 3.63: Imagen restada 92
Figura 3.64: Erosión de la imagen 93
Figura 3.65: Imagen dilatada 93
Figura 3.66: Imagen contornos resaltados 94
Figura 3.67: Diagrama de Bloques del algoritmo 96
Figura 4.1: MINERAL16 tomada de la faja para el procesamiento 98
Figura4.2: Resultado obtenido mediante algoritmo 99
Figura 4.3: MINERAL17 para la segunda muestra 100
Figura 4.4: Resultado obtenido mediante algoritmo 101
Figura 4.5: MINERAL18 muestra a ejecutar con el algoritmo 102
Figura 4.6: Resultado obtenido por el algoritmo 102
Figura 4.7: MINERAL19 muestra para el procesamiento 103
Figura 4.8: Resultado obtenido por el algoritmo 104

ÍNDICE DE TABLAS

Tabla 2.1: Codificación de los colores 37

Tabla 4.1: Cuadro de comparación de resultados obtenidos 105

RESUMEN

El trabajo de investigación titulada, diseño de un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora en la unidad minera mallay, la investigación experimental de diseñó del algoritmo de procesamiento de imágenes para la minería, El objetivo de la investigación fue; Diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora y reducir la parada de producción, para el diseño del algoritmo se utilizó las librerías de Python: OpenCV, Numpy, Matplotlip mediante los cuales se realizó la manipulación de las imágenes con resultados requeridos, para esto se utilizaron las técnicas de; la conversión de imagen a escala grises, Erosión, Dilatación, detección de bordes Canny y Sobel, histograma, Threshold, Filtro Gaussiano, de esta manera el algoritmo de procesamiento de imágenes es desarrollado, posterior a este diseño se realizaron las pruebas en las muestras, verificándose el funcionamiento del algoritmo. La metodología aplicada para su desarrollo estuvo establecida a través del método descriptivo experimental, los que permitieron establecer conclusiones; el diseño de un algoritmo de procesamiento de imágenes con la herramienta Python OpenCV determina el volumen y peso del mineral que se transporta en la faja, obteniéndose el pesaje (algoritmo) del mineral en las 4 paradas del día en 38.28 segundos, en comparación del pesaje manual que demora 76 minutos aproximadamente obteniendo del mismo resultado, el pesaje en el algoritmo presenta una precisión del 99% y margen de error del 1%, que es aceptable de acuerdo a los objetivos de esta investigación, se recomienda OPenCV como una herramienta efectivo para este tipo de investigaciones.

Palabras Clave: Procesamiento de imágenes, faja transportadora, control automático, sistema de pesaje, OpenCV

ABSTRACT

The research work entitled, design of an image processing algorithm of the weighing system for automatic control of a conveyor belt in the Mallay mining unit, experimental research designed the image processing algorithm for mining, The objective of the investigation was; Design an image processing algorithm of the weighing system for the automatic control of a conveyor belt and reduce the production stop. Python libraries were used for the algorithm design: OpenCV, Numpy, Matplotlip by means of which the manipulation was carried out. of the images with required results, for this the techniques of; Gray-scale image conversion, Erosion, Dilation, Canny and Sobel edge detection, histogram, Threshold, Gaussian Filter, in this way the image processing algorithm is developed, after this design the tests were performed on the samples, checking the operation of the algorithm. The methodology applied for its development was established through the experimental descriptive method, which allowed conclusions to be established as follows; The design of an image processing algorithm with the specialized Python OpenCV tool determines the volume and weight of the ore that is transported in the belt, obtaining the weighing (algorithm) of the ore at the 4 stops of the day in 38.28 seconds, in comparison to the manual weighing that takes approximately 76 minutes obtaining the same result, the weighing in the algorithm has an accuracy of 99% and a margin of error of 1%, which is acceptable according to the objectives of this investigation, OPenCV is recommended as an effective tool For this type of research.

Keywords: Image processing, conveyor belt, automatic control, weighing system, OpenCV

CAPITULO I

1.1. INTRODUCCIÓN

El desarrollo de algoritmos para el procesamiento de imágenes está cobrando mayor impacto en esta era de la digitalización, especialmente en el campo de la inteligencia artificial para aplicaciones ligadas a la visión por computadora, esto se alimenta mediante la digitalización de los datos en los dispositivos para captar fotos o videos, como consecuencia estas imágenes obtenidas se convierten en el centro de la interpretación de la realidad. Así mismo en los procesos mineros y su automatización se están abriendo oportunidades para la aplicación de técnicas de análisis de minerales como la obtención del peso y el volumen, de este modo aumentar la productividad. La potencialidad de funcionamiento de este algoritmo es motivo del desarrollo de esta tesis para lo cual se tiene que ejecutar los comandos de procesamiento de imágenes secuencialmente hasta lograr diseñar el algoritmo que será capaz de hallar el área y volumen de los minerales que se transportan en la faja hacia el molino de este modo reducir los tiempos de parada mejorando la productividad; este proyecto se desarrollara de forma que sus propiedades y características se podrán a prueba.

En el capítulo I se presenta el planteamiento del problema del diseño del algoritmo de procesamiento de imágenes, la motivación del proyecto, y los objetivos de la presente investigación.

En el capítulo II se definen los antecedentes del proyecto, así como también los conceptos relacionados al proyecto, la descripción de las librerías y términos utilizados en el diseño del algoritmo.

En el capítulo III se describe los materiales y metodología de la investigación, población, muestra, el procedimiento de diseño del algoritmo, diagrama de bloques y diagrama de flujos sobre el funcionamiento del algoritmo.

En el capítulo IV se muestran los resultados del diseño del algoritmo, la obtención del área y volumen mediante la ejecución del algoritmo incluye también la sección de discusión sobre las diferencias y similitudes de este proyecto con los proyectos mencionados en antecedentes.

1.2. PLANETAMIENTO DEL PROBLEMA

Se tiene una balanza de pesaje marca Merrick en la faja transportadora ingresa a molino de 8x10. El problema es la variación de minerales ingresados al molino, en los ciclones, y electroválvulas por el incremento de mineral cuando el flujo de la densidad aumenta más de 2000 gr/cm3. El personal de operaciones verifica el peso mostrado por la balanza con set point de 25 toneladas por hora, la cual hace un contraste de peso de forma manual con el método de corte de la faja generando un tiempo de parada en la producción que usualmente se llega a 4 paradas diarias. Es por ello que se plantea Diseñar un algoritmo de procesamiento de imágenes en OpenCV que está orientado en la Visión por computador y tiene las herramientas necesarias para crear el algoritmo.

1.3. FORMULACIÓN DEL PROBLEMA

¿Cómo diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora y reducir la parada de producción en la unidad minera Mallay?

1.3.1. PROBLEMAS ESPECIFICOS

- ¿Cómo desarrollar un algoritmo de procesamiento de imágenes para la determinación de volumen en la faja transportadora?

- ¿Cómo obtener el volumen del material que ingresa a la faja transportadora en base a la equivalencia de un pixel en centímetros cuadrados y la densidad de la muestra?

1.4. HIPÓTESIS DE LA INVESTIGACION

1.4.1. HIPÓTESIS GENERAL

El diseño de un algoritmo de procesamiento de imágenes del sistema de pesaje permitirá el control automático de la faja transportadora y reducir la parada de producción en la unidad minera Mallay.

1.4.2. HIPÓTESIS ESPECÍFICAS

- Contar con un algoritmo de procesamiento de imágenes que determine el volumen de la faja transportadora.
- El volumen del material que ingresa en la faja transportadora se obtendrá en base a la equivalencia de un pixel en centímetros cuadrados y la densidad de la muestra.

1.5. OBJETIVOS DE LA INVESTIGACION

1.5.1. OBJETIVO GENERAL

Diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora y reducir la parada de producción en la unidad minera Mallay.

1.5.2. OBJETIVOS ESPECIFICOS

- Desarrollar un algoritmo de procesamiento de imágenes para determinar el volumen en la faja transportadora.
- Obtener el volumen del material que ingresa en la faja transportadora en base a la equivalencia de un pixel en centímetros cuadrados y la densidad de la muestra.

CAPITULO II

REVISIÓN DE LITERATURA

2.1. ANTECEDENTES DEL PROYECTO

Se encontraron investigaciones de referencia, estos antecedentes ayudarán a tener opciones al desarrollar el proyecto y estos serán mencionadas a continuación:

En el trabajo de (Abarca Cusimayta, 2018) Diseño de un modelo algorítmico basado en visión computacional para la detección y clasificación de retinopatía diabética en imágenes retinográficas digitales (Tesis de Pregrado) Pontificia Universidad Católica del Perú, Lima, Perú: Concluye que:

Se establecieron técnicas de preprocesamiento para poder mejorar la imagen y la información obtenida para enviarla al clasificador. Es así que, la detección de las regiones de interés se logró mediante la detección del fondo en la imagen de entrada para poder crear una máscara que permita extraer solo la sección de la retina (foreground). Adicionalmente, se realizó la extracción del disco óptico, pues en el análisis de exudados este elemento es considerado como ruido debido a que ambos elementos son considerados como brillosos. Cabe mencionar que algunas imágenes de la base de datos presentaban ruido alrededor del disco óptico lo cual originaba que se produjeran falsos positivos en la detección. Las imágenes resultantes de aplicar la extracción del foreground y del disco óptico son las que se utilizaron para la fase de detección de características clínicas.

El modelo de clasificación utilizado recibe el vector mencionado y lo procesa para determinar el grado de severidad de la retinopatía diabética (normal, leve, moderado, severo). Para la clasificación se utilizó el modelo SVM con el uso del kernel RBF. Se utilizaron un total de 600 imágenes (150 por categoría) se aplicó cross-validation para

evitar que el modelo caiga en el efecto del sobre-encaje (over fitting). El porcentaje de precisión alcanzado por el modelo fue de 95.08%. Este valor comparado con el trabajo de referencia (Mahendran & Dhanasekaran, 2015), quienes obtuvieron 97.89%, nos da un resultado aceptable debido a que este autor utilizó solo tres categorías de clasificación (normal, moderado y severo); mientras que en el presente proyecto se utilizaron cuatro. Por último, para temas de visualización y presentación del modelo se implementó una interfaz gráfica que permite, seleccionar una imagen y determinar el nivel de severidad de la retinopatía diabética.

En la siguiente investigación de (Viera Maza, 2017) Procesamiento de imágenes usando OpenCV aplicado en Raspberry Pi para la clasificación del cacao (Tesis de Pregrado) Universidad de Piura, Piura, Perú. Llego a las siguientes conclusiones:

El En el Perú no existen muchos sistemas de visión artificial para la evaluación de calidad y clasificación de productos agrícolas. Generalmente, la inspección se realiza de manera manual, siendo esto una limitante para el desarrollo de la agroindustria.

La visión artificial cumple un rol muy importante en el desarrollo de nuevas aplicaciones en la industria agroalimentaria, como la evaluación de calidad y selección de un producto respetando ciertos parámetros impuestos por el consumidor. Con las opciones baratas y rápidas tanto en software como hardware, se utilizarán más sistemas de visión artificial para la evaluación de la calidad de los alimentos, y así, aumentarán las posibilidades de comercialización.

En los últimos años, con el aumento del uso de técnicas de procesamiento de imágenes, que es el núcleo de la visión por computador, se ha logrado que la inspección de productos sea automatizada, objetiva, rápida e higiénica. Sobre la base de la visión por

ordenador, es factible aumentar el rendimiento de la producción, disminuir los costes de producción, y mejorar la confiabilidad de la calidad de los productos.

La iluminación juega un papel muy importante en los sistemas de visión artificial, se debe tener en cuenta la disposición e intensidad de la iluminación, pues influye en el procesamiento de la imagen. El sistema desarrollado se basa en la extracción de las características externas de los granos de cacao como el tamaño y color, entonces durante la adquisición se deben evitar en la imagen dos defectos: las sombras y los brillos. Las sombras aparecen cuando no se ubica adecuadamente la fuente de iluminación, o existe una escasez de fuentes de iluminación, y los brillos aparecen cuando la dirección de la iluminación no es la adecuada.

El sistema desarrollado implica una disminución del tiempo de selección con respecto a la manera artesanal, y una mayor precisión en la evaluación. Sería importante utilizar nuevas características específicas que complementen a las anteriormente utilizadas para determinar otras cualidades como el grado de fermentación durante la producción.

El resultado de esta tesis es un sistema para la clasificación de granos de cacao según su tamaño utilizando técnicas de procesamiento de imágenes con un porcentaje de acierto de 89% en un total de 300 imágenes evaluadas. Con respecto al tamaño de los granos se midió su largo, presentando en este caso un error máximo de 0.06 cm en un total de 300 fotos evaluadas. La diferencia entre el valor real y experimental se debe principalmente a que las medidas se realizaron con un pie de rey, como sabemos los granos tiene un volumen deforme entonces en ciertos casos había que hacer algunas aproximaciones.

En la investigación de (Lamego Castro, 2017) Desarrollo de un sistema inteligente de control de tráfico con software de código abierto en sistemas embebidos (Tesis de postgrado) CIATEO. Jalisco. México: Concluye que:

La combinación de software de código abierto y componentes de uso común permite la creación de prototipos funcionales en un corto tiempo, lo que beneficia directamente a los equipos de desarrollo, quienes pueden enfocar sus esfuerzos en el desarrollo y mejora de sus algoritmos de control y procesamiento de datos, aprovechando el uso de sistemas operativos y librerías existentes para el manejo de las capas de manejo de hardware y de comunicaciones, por ejemplo.

La organización del código de aplicación principal en bloques funcionales permite ofrecer diferentes grados de funcionalidad simplemente con la adición o supresión de sub-bloques. La publicación del código en repositorios públicos permite no sólo la reutilización de estos bloques en otros proyectos similares (disponibles en los repositorios de la comunidad de desarrolladores de software de código abierto) sino además la participación de dicha comunidad en la mejora y posible evolución de este proyecto.

En la investigación de (Bereciartua Pérez, 2016) Desarrollo de algoritmos de procesamiento de imagen avanzado para interpretación de imágenes médicas (tesis de Doctorado) Universidad de País Vasco (UPV/EHU) en su conclusión indica que:

El procesamiento y análisis de la imagen médica es un problema complejo que requiere de un conocimiento especializado. La imagen médica presenta muchos artefactos, ruidos y efectos de iluminación no homogénea en la imagen, causados por las tecnologías de adquisición, que hacen necesario el uso de técnicas avanzadas para la segmentación correcta y precisa de los tejidos presentes en las imágenes.

El descriptor compacto propuesto reduce notablemente el tiempo de cómputo necesario y permite la reducción de la dimensión del problema. En este caso, las 5 secuencias de entrada han sido transformadas en una única imagen que representa un mapa de distancias de cada píxel de la imagen con respecto a un modelo estadístico multivariable de hígado previamente generado. Si se aplica la misma metodología a cualquier otro caso, se puede transformar una entrada de N canales en una sola entrada. Esto es de gran importancia para reducir la dimensión del problema en caso de disponer de múltiples secuencias del mismo estudio, y facilita la aplicación de técnicas de modelos deformables o superficies activas sobre esta entrada única que contiene la información de todos los canales.

El siguiente trabajo de investigación de (Palomino Puma & Manrique Hernandez, 2015) Sistema de llenado Automático de Botellas con Control de nivel utilizando Procesamiento Digital de Imágenes (Tesis de Pregrado) Universidad Ricardo Palma, Lima. Concluye que:

Esta tesis propone el uso de procesamiento digital de imágenes para el control de clasificación; La banda transportadora construida para desplazar a las botellas no presenta problemas respecto al transporte porque la botella es de plástico y además está vacía, de este modo el motor que mueve la faja no presenta inconvenientes. Sin embargo, los problemas se presentaron en la regulación de la intensidad luminosa de los sensores que utiliza la faja para detectar a la botella y determinar el giro o la parada de la faja. Por esa razón se realizaron diferentes pruebas donde se pudo observar que la intensidad luminosa de los sensores debe tener un nivel bajo.

En la siguiente tesis de (Varela Jarquin & Sequeira Lanuza, 2016) Propuesta de diseño de un sistema de automatización de una cinta de trasportadora utilizada en la

industria minera (Tesis de Pregrado) Universidad Nacional De Ingeniería, Managua, Nicaragua. Concluye qué:

Al lograr estos cambios en esta instalación se da un valor muy importante a la seguridad de las personas y de los equipos, por lo que las zonas de trabajo son reguladas con velocidades máximas de funcionamiento predeterminadas y señalizadas mediante señales luminosas y acústicas.

El uso correcto de la instalación y la seguridad de la carga a transportar se aseguran mediante el sistema de detección de paso de carga, su principal defecto es la pérdida de exactitud en la detección en los arranques y las paradas de la instalación ya que no es capaz de detectar pérdidas de carga que se pudieran ocasionar en estos casos.

Como medida de mejora se podrían introducir una mayor cantidad de sensores intermedios que permitieran un seguimiento de la carga más exhaustivo u otras lógicas de supervisión de la carga mediante contadores.

2.2. MARCO TEÓRICO

2.2.1. LENGUAJE DE PROGRAMACIÓN

Un programa de computadora es una lista de declaraciones que una computadora debe realizar para completar una tarea, generalmente una tarea repetitiva que le tomaría mucho tiempo a un humano calcular. Un lenguaje de computadora describe la disposición o sintaxis de esas declaraciones. Existen varios lenguajes de computadora, cada uno adecuado para una o más tareas. Cada idioma tiene su propia sintaxis única y un conjunto de comandos, pero todos tienen construcciones que realizan aproximadamente los mismos tipos de acciones:

- Entrada
- Salida

- Ramificación (toma de decisiones basadas en datos)
- bucles

Un comando o palabra clave es una frase especial que utiliza el lenguaje para realizar una acción, ya sea para obtener información del usuario o mostrar texto en la pantalla. (Kelly, 2019)

Un Lenguaje de Programación es, un conjunto de instrucciones que son entendibles y ejecutables por un computador. No podemos esperar, por lo menos no por ahora, que el computador ejecute lo que nosotros concebimos como algoritmo (aunque sería lo óptimo) y por eso debemos incorporar al desarrollo técnico un paso o un conjunto de pasos más y son lo que involucran los Lenguajes de Programación. (Trejos Buritica, 2017)

Para que la computadora entienda nuestras instrucciones debe usarse un lenguaje específico conocido como código máquina, que la máquina lee fácilmente, pero que es excesivamente complicado para las personas. De hecho, solo consiste en cadenas extensas de números 0 y 1. Para facilitar el trabajo, los primeros operadores de computadoras decidieron crear un traductor para reemplazar los 0 y 1 por palabras o abstracción de palabras y letras provenientes del inglés; este se conoce como lenguaje ensamblador. Por ejemplo, para sumar se usa la letra A de la palabra inglesa add (sumar). El lenguaje ensamblador sigue la misma estructura del lenguaje máquina, pero las letras y palabras son más fáciles de recordar y entender que los números (Joyanes Aguilar, Echeverri Arias, Orrego Villa, & Arenas Arenas, 2016)

A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método más eficiente para programarlas. Entonces, se crearon los lenguajes de alto nivel, como lo fue BASIC en las versiones

introducidas en los microordenadores de la década de 1980. Mientras que una tarea tan sencilla como sumar dos números puede necesitar varias instrucciones en lenguaje ensamblador, en un lenguaje de alto nivel bastará una sola sentencia. (Depetris & Otros, 2018)

El lenguaje de programación permite especificar de manera precisa sobre qué datos debe operar un software específico, cómo deben ser almacenados o transmitidos dichos datos, y qué acciones debe dicho software tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción de un programa de forma colaborativa. (Trejos Buritica, 2017)

2.2.2. PYTHON

Python es un lenguaje de programación moderno que admite estilos de programación orientados a objetos, funcionales e imperativos. Es ideal para principiantes debido a su legibilidad y facilidad de uso. La ventaja de todo esto es que puede escribir programas en menos líneas de código que un programa C / C ++ o Java equivalente. (Kelly, 2019)

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio (en inglés: Benevolent Dictator for

Life, BDFL); sin embargo, el 12 de julio de 2018 declinó de dicha situación de honor sin dejar un sucesor o sucesora y con una declaración altisonante. (Troyano, Cruz, Gonzalez, Vallejos, & Toro, 2018)

En 1991, van Rossum publicó el código de la versión 0.9.0 en alt.sources.8 En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y los tipos modulares, como: str, list, dict, entre otros. Además en este lanzamiento inicial aparecía un sistema de módulos adoptado de Modula-3; van Rossum describe el módulo como "una de las mayores unidades de programación de Python".3 El modelo de excepciones en Python es parecido al de Modula-3, con la adición de una cláusula else.4 En el año 1994 se formó comp.lang.python, el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios de este lenguaje.

Python alcanzó la versión 1.0 en enero de 1994. Una característica de este lanzamiento fueron las herramientas de la programación funcional: lambda, reduce, filter y map. Van Rossum explicó que "hace 12 años, Python adquirió lambda, reduce(), filter() y map(), cortesía de un hacker informático de Lisp que las extrañaba y que envió parches". El donante fue Amrit Prem; no se hace ninguna mención específica de cualquier herencia de Lisp en las notas de lanzamiento. (Rios, Mora, & Sojos, 2016)

En el año 2000, el equipo principal de desarrolladores de Python se cambió a BeOpen.com para formar el equipo BeOpen PythonLabs. CNRI pidió que la versión 1.6 fuera pública, continuando su desarrollo hasta que el equipo de desarrollo abandonó CNRI; su programa de lanzamiento y el de la versión 2.0 tenían una significativa cantidad de traslapo. Python 2.0 fue el primer y único lanzamiento de BeOpen.com. Después que Python 2.0 fuera publicado por BeOpen.com, Guido van Rossum y los otros desarrolladores de PythonLabs se unieron en Digital Creations. (Díaz García, 2018)

El creador del lenguaje es un europeo llamado Guido Van Rossum. Hace ya más de una década que diseño Python, ayudado y motivado por su experiencia en la creación de otro lenguaje llamado ABC. El objetivo de Guido era cubrir la necesidad de un lenguaje orientado a objetos de sencillo uso que sirviese para tratar diversas tareas dentro de la programación que habitualmente se hacía en Unix usando C. (Honores Tapia & Camacho Osmolik, 2019)

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad. (Lorens Largo, Garcia Peñalgo, Molero Prieto, & Vendrel Vidal, 2017)

2.2.3. OPENCV

OpenCV es una Liberia de visión por computadora de código abierto disponible en http://opencv.org. La biblioteca está escrita en C y C ++ y se ejecuta en Linux, Windows, Mac OS X, iOS y Android. Las interfaces están disponibles para Python, Java, Ruby, Matlab y otros lenguajes de programación. OpenCV fue diseñado para la eficiencia computacional con un fuerte enfoque en aplicaciones en tiempo real: se realizaron optimizaciones en todos los niveles, desde algoritmos hasta instrucciones de múltiples núcleos y CPU. OpenCV usa automáticamente las instrucciones apropiadas de IPP en tiempo de ejecución. El módulo GPU también proporciona versiones aceleradas por CUDA de muchas rutinas (para GPU Nvidia) y optimizadas para OpenCL (para GPU genéricas). Uno de los objetivos de OpenCV es proporcionar una infraestructura de visión por computadora fácil de usar que ayude a las personas a crear aplicaciones de visión bastante sofisticadas rápidamente. La librería OpenCV contiene más de 500 funciones

que abarcan muchas áreas, incluida la inspección de productos de fábrica, imágenes médicas, seguridad, interfaz de usuario, calibración de la cámara, visión estéreo y robótica. Debido a que la visión por computadora y el aprendizaje automático a menudo van de la mano, OpenCV también contiene una librería de aprendizaje automático (MLL) completa y de propósito general. Esta sub-librería se centra en el reconocimiento y agrupamiento de patrones estadísticos. El MLL es muy útil para las tareas de visión que están en el centro de la misión de OpenCV, pero es lo suficientemente general como para ser utilizado para cualquier problema de aprendizaje automático. (Kaehler & Bradski, 2016)

2.2.4. ALGORITMO

En el contexto matemático, los algoritmos son una serie de normas o leyes específicas que hace posible la ejecución de actividades, cumpliendo una serie de pasos continuos que no le originen dudas a la persona que realice dicha actividad, un algoritmo es un conjunto de instrucciones paso a paso que manipulan la información para encontrar la solución a un problema. De hecho, algoritmo no es específico para las computadoras y tiene sus raíces derivadas de matemáticas. Casi todas nuestras actividades cotidianas que van desde multiplicar números a la programación de vuelos se basan en un conjunto definido de reglas se realizan de manera predefinida que constituyen el proceso algorítmico. (Olarte Gervacio, 2018)

El estudio de los algoritmos es una disciplina de las ciencias de la computación y en la mayoría de los casos, su estudio es completamente abstracto sin usar ningún tipo de lenguaje de programación ni cualquier otra implementación; por eso, en ese sentido, comparte las características de las disciplinas matemáticas. Así, el análisis de los algoritmos se centra en los principios básicos del algoritmo, no en los de la implementación particular. Una forma de plasmar (o algunas veces "codificar") un

algoritmo es escribirlo en pseudocódigo o utilizar un lenguaje muy simple tal como Léxico, cuyos códigos pueden estar en el idioma del programador. Algunos escritores restringen la definición de algoritmo a procedimientos que deben acabar en algún momento, mientras que otros consideran procedimientos que podrían ejecutarse eternamente sin pararse, suponiendo el caso en el que existiera algún dispositivo físico que fuera capaz de funcionar eternamente. En este último caso, la finalización con éxito del algoritmo no se podría definir como la terminación de éste con una salida satisfactoria, sino que el éxito estaría definido en función de las secuencias de salidas dadas durante un período de vida de la ejecución del algoritmo. (Contributors, 2019)

2.2.5. PROCESAMIENTO DE IMÁGENES

El Procesamiento de Imágenes (PI) es una subcategoría del tratamiento digital de señales. Es la ciencia de manipulación de imágenes usando computadores para realizar procedimientos específicos según las aplicaciones y requerimientos del usuario, tales como: filtrado, recorte, segmentación, compresión y reconocimiento. Es un área del conocimiento que tiene atención de investigadores para desarrollar y mejorar algoritmos para aplicaciones en: robótica, comunicaciones, sensores remotos, biomedicina, automatización industrial, sistemas de inspección, navegación, mediciones ópticas, entre otras. Las prácticas de laboratorio son actividades pedagógicas, y en el estudio del PI éstas se realizan mediante el uso de algún software especializado. Algunos softwares comerciales de procesamiento de imágenes como: Photoshop, CorelDraw, Ulead Photoimpact, entre muchos otros, realizan procedimientos internos que no permiten el entendimiento de algoritmos, la lógica o el método del proceso, presentando cajas negras que no admiten su manipulación. Después de su uso, los estudiantes se vuelven expertos en la herramienta mas no en el procesamiento de imágenes, que, por supuesto podría ser

adecuado para diseñadores digitales, pero no para ingenieros electrónicos, ni de sistemas. (Cavanzo Nisso, Pérez Pereira, & Villavisan Buitrago, 2017)

En la Universidad de los Llanos se ha planteado el uso de Python en el curso de PI, con el fin de lograr un aprendizaje integral del estudiante, al permitirle desarrollar habilidades de alta calidad en la producción de software, con aplicabilidad en la investigación y el diseño de proyectos de nivel avanzado. Python es un lenguaje de programación de alto nivel, interpretado y multipropósito, cuyo creador es Guido Van Rossum. En los últimos años su utilización ha aumentado y es uno de los lenguajes de programación más empleados para el desarrollo de software. Python puede ser utilizado en diversas plataformas y sistemas operativos, entre los que se puede destacar: Windows, Mac OS X y Linux. Pero, además, Python también puede funcionar en smartphones y sistemas embebidos. Para la enseñanza de PI, además de la herramienta de software es importante usar el Aprendizaje Basado en Proyectos (ABPr), que permite a los estudiantes la definición del propósito de la creación de un producto final, identificar su mercado, investigar, crear un plan de trabajo, diseñar y elaborar un producto. El proceso completo es auténtico, referido a la producción en forma real, utilizando las propias ideas de los estudiantes y completando las tareas en la práctica. Debido a esto, los estudiantes se enfocan en lograr soluciones a problemas no triviales, generando preguntas, consultando, discutiendo ideas, realizando predicciones, diseñando planes de trabajo y/o experimentos, recolectando y analizando información, estableciendo conclusiones, comunicando sus resultados, cuestionándose y creando o mejorando productos y procesos. El ABPr es una estrategia pedagógica favorable para la enseñanza de temáticas de ingeniería, que se complementa al involucrar proyectos industriales y de impacto en la comunidad universitaria y la región, para ofrecer una experiencia tan auténtica como sea posible. La tendencia también se dirige a realizar estos proyectos en forma

interdisciplinaria, con la colaboración de otras dependencias o departamentos de ingeniería. En este artículo se da a conocer la experiencia en la realización del curso de pregrado: Electiva de Profundización en Procesamiento de Imágenes de la Universidad de los Llanos y unos de los proyectos desarrollados en el curso; estos proyectos buscan brindar la solución a necesidades del entorno, además de permitir a los estudiantes la conceptualización de diferentes temáticas a través de la interactividad con los resultados obtenidos en las distintas etapas del procesamiento, enfocados en el entendimiento de los algoritmos y códigos utilizados. El software desarrollado en cada proyecto de curso está basado en QT4 y Python 2.7, permitiendo que los resultados se puedan aplicar no solo en el área de procesamiento de imágenes, sino también en otras áreas del conocimiento, especialmente en Ingeniería y Ciencias – Física. Algunas de las aplicaciones desarrolladas en el curso han sido fundamentales para los proyectos institucionales de la Universidad de los Llanos: Diseño e Implementación de un Laboratorio Virtual Remoto para prácticas de Mecánica – Cinemática en la Universidad de los Llanos, para el procedimiento de evaluación de las prácticas remotas mediante procesamiento digital de imágenes y Diseño e implementación de un sistema asistido por computador de la prueba de Ronchi en la Universidad de los Llanos – SAPRULL, en el que se optimiza el procedimiento de adquisición de imágenes e identificación de aberraciones en la fabricación de espejos. (Delgado Gutiérrez, Herrera Guillén, Medina Barragán, & Corredor Gómez, 2017)

2.2.6. FAJA TRANSPORTADORA

Una cinta transportadora o banda transportadora o transportador de banda o cintas francas o faja transportadora es un sistema de transporte continúo formado por una banda continua que se mueve entre dos tambores. Por lo general, la banda es arrastrada por la fricción de sus tambores, que a la vez este es accionado por su motor. Esta fricción es la resultante de la aplicación de una tensión a la banda transportadora, habitualmente

mediante un mecanismo tensor por husillo o tornillo tensor. El otro tambor suele girar libre, sin ningún tipo de accionamiento, y su función es servir de retorno a la banda. La banda es soportada por rodillos entre los dos tambores. Denominados rodillos de soporte. (Molleapaza Mamani, 2018)

Es un aparato para el transporte de objetos formado por dos poleas que mueven una cinta transportadora continua. Las poleas son movidas por motores, haciendo girar la cinta transportadora y así lograr transportar el material depositado en la misma. En el transporte de materiales, materias primas, minerales y diversos productos se han creado diversas formas; pero una de las más eficientes es el transporte por medio de bandas y rodillos transportadores. Ya que estos elementos son de una gran sencillez de funcionamiento, que una vez instalados en condiciones suelen dar pocos problemas mecánicos y de mantenimiento. Las bandas y rodillos transportadoras son elementos auxiliares de las instalaciones, cuya misión es la de recibir un producto de forma más o menos continua y regular para conducirlo a otro punto. (Varela Jarquin & Sequeira Lanuza, 2016)

2.3. VISIÓN POR COMPUTADORA

La visión por computadora o visión artificial es una parte de la inteligencia artificial, es un conjunto de teorías, técnicas y métodos que nos permiten simular el proceso de visión biológico de los humanos y la capacidad de extraer y analizar automáticamente información de las imágenes obtenidas. La visión artificial nos permite crear algoritmos y aplicaciones para interpretar el significado de una imagen, con esto podemos obtener información de un objeto espacial (3D) a partir de la adquisición y procesamiento de una o varias imágenes digitales (2D) de dicho objeto. La interpretación de ciertos tipos de imágenes, como escenas naturales o rasgos faciales, es un proceso

complejo para la máquina. Sin embargo, el análisis cuantitativo en las imágenes es relativamente sencillo para los sistemas de visión artificial. (Viera Maza, 2017)

Los procesos de la visión artificial pueden agruparse en 3 niveles según su complicación e implementación (Ingle & Proakis, 2016)

- Procesos de bajo nivel: se refieren a operaciones primitivas como es la captura o adquisición de imágenes y el pre-procesamiento de imágenes para reducir el ruido, mejorar el contraste y nitidez de la imagen. Una característica importante de estos procesos es que sus entradas y salidas o resultados son imágenes.
- Procesos de nivel medio: se refieren a operaciones como la segmentación y la descripción de los objetos individuales presentes en una escena para reducirlos a una forma adecuada para su tratamiento informático, reconocimiento y clasificación de estos. Un rasgo importante de estos procesos es que sus entradas son generalmente imágenes, pero sus resultados son características extraídas de las imágenes como bordes, contornos o la identidad de los objetos presentes en las imágenes.
- Procesos de nivel superior: se refieren a operaciones que reconocen un conjunto de objetos en la imagen y realizan funciones cognitivas que normalmente se asocian con la visión, conocido como el proceso de interpretación.

Aunque estas etapas son aparentemente secuenciales, esto no es necesario, y pueden aparecer interacciones entre los diferentes niveles incluyendo la retroalimentación de los niveles altos a los inferiores.

2.3.1. ELEMENTOS DEL SISTEMA DE VISIÓN POR COMPUTADORA

2.3.1.1 Sistema de Iluminación

La iluminación es un elemento de vital importancia en el desarrollo de sistemas de visión artificial para la obtención de resultados óptimos. Eligiendo la iluminación adecuada se puede obtener una mayor exactitud en las medidas, un sistema confiable y un menor tiempo de operación. Los principales objetivos respecto a la iluminación en los sistemas de visión artificial son: mantener constante la intensidad y dirección de la luz, y optimizar el contraste para diferenciar los objetos presentes del fondo. En los sistemas de visión artificial para procesos industriales o de laboratorio siempre se diseña un módulo de iluminación adecuado que resalte las características del producto que se piensa analizar. Si se cuenta con un sistema de iluminación adecuado, no es necesario corregir fallas de iluminación por medio de algoritmos. Si se captura una imagen en un entorno con iluminación arbitraria sin tomar en cuenta la información que se desea extraer, es muy probable que dicha imagen tenga un bajo contraste, reflexiones especulares o sombras. Por lo cual, una imagen obtenida con un sistema de iluminación adecuado implica un menor tiempo de procesamiento de ésta pues nos permite independizar las condiciones del entorno y resaltar los rasgos de interés en ella. (Viera Maza, 2017)

Que a su vez se puede clasificar los distintos tipos de iluminación según su intensidad, dirección y fuente de origen.

- **Dependiendo de la intensidad de la luz**: Cuando se varía la intensidad de la luz obtenemos distintos efectos resultantes en la imagen capturada, por ejemplo, una intensidad fuerte implica que aparezcan grandes contrastes entre las zonas iluminadas y las zonas sombreadas, y a la vez, debido a un

rango dinámico limitado, se pierden los detalles en la imagen tanto en las zonas iluminadas como sombreadas. Un efecto opuesto resulta cuando la intensidad es suave, se aprecian mejor los detalles de la imagen tanto en las zonas iluminadas como sombreadas, pero se pierden detalles en las texturas.

- **Dependiendo de la dirección de la iluminación:** *Posterior*; También llamada retroiluminación, consiste en colocar el objeto entre la fuente de luz y la cámara. Es el tipo de iluminación más adecuado para el reconocimiento y medida de objetos por medio de la detección de bordes, debido a que las imágenes resultantes tienen un alto contraste entre los objetos y el fondo, aunque se pierden algunos detalles de la escena. Un problema común es cuando existen objetos ubicados uno encima de otro, y se debe tener en cuenta que una de sus mayores desventajas es la forma de implementarla dentro de un sistema industrial automático. *Frontal*; consiste en iluminar el objeto frontalmente, es decir la luz incide frontalmente de forma directa sobre el objeto. Permite visualizar las características externas de los objetos como son forma, color o superficies, que permitan una mejor segmentación y reconocimiento de patrones. Es el tipo de iluminación más usado, pero en ocasiones no se obtiene un buen contraste entre el objeto y el fondo, debido a la aparición de sombras y reflejos. *Direccional*; consiste en proyectar una luz direccionada en algún sentido en el espacio para destacar determinadas características del objeto, la orientación del foco está hacia el objeto. Se genera sombras sobre el objeto, lo que aumenta el contraste entre partes tridimensionales y con ello obtener información tridimensional. Este tipo de iluminación se usa

principalmente para la localización y reconocimiento de objetos, inspección de superficies, seguimiento de cordones de soldadura, etc. *Estructurada*; consiste en proyectar patrones modulados de iluminación sobre el objeto y adquirir información sobre la superficie del objeto utilizando la luz reflejada. Generalmente las fuentes de luz son láseres y los usos principales de este tipo de iluminación son en reconstrucciones 3D de objetos y reconocimiento de formas.

- **Dependiendo de la fuente de iluminación:** *Lámpara incandescente;* fue la primera fuente de origen de luz por energía eléctrica y es la fuente más común de iluminación. *Fluorescente;* proporciona una luz brillante sin sombras, pero por su limitada variedad de formas, es limitada su aplicación en sistemas de visión artificial. *LED (diodo emisor de luz);* es una fuente de estado sólido que emite luz cuando la electricidad es aplicada a un semiconductor. *Fibra Óptica;* proporciona una gran intensidad de luz uniforme, fría y con ausencia de sombras, y consiste en dirigir la luz procedente de una bombilla halógena, *Láser;* es utilizado principalmente en la iluminación estructurada. Su principal desventaja es que no es eficiente en superficies que absorben luz.

2.3.1.2 Cámaras y Tarjeta de captura

Una cámara digital es un equipo de fotográfica que captura y almacena imágenes de manera digital, gracias a un dispositivo denominado sensor. Generalmente, las cámaras digitales utilizan dos tipos de sensores: sensor CCD (Charge Couple Device, en español “dispositivo de carga acoplada”) y sensor CMOS (Complementary Metal Oxide Semiconductor, en español “semiconductor complementario de óxido metálico”). La calidad de la resolución de una cámara

digital no depende únicamente del número y distribución de pixeles que me puede dar la cámara, sino también de otros factores como las características del sensor y características del lente.

En esta investigación se ha utilizado una cámara web, es un tipo de cámara digital para uso en red que se conecta a una laptop mediante puerto USB. Para su instalación solo se necesita conectar la cámara a la laptop, pero en algunas ocasiones se necesita el driver de instalación. Este tipo de cámara permite capturar imágenes de calidad promedio.

2.3.1.3 Módulo de procesamiento y software

El módulo de procesamiento puede ser una computadora o un sistema integrado, es el sistema que recibe, almacena las imágenes y las procesa a través de algoritmos adecuados para extraer la información necesaria y luego tomar decisiones según la aplicación del sistema de visión artificial. Los sistemas integrados son aquellos que incorporan el software y todo el hardware necesario en un mismo sistema, disponen de un procesador integrado con capacidad de tomar decisiones.

2.3.2. PIXEL

Cada imagen consiste en un conjunto de pixeles. Los pixeles son la unidad básica de análisis en una imagen. No hay un elemento de menor dimensión y más fino en una imagen que el pixel. Se suele relacionarlo con “color” o “intensidad”. Si pensamos en una imagen como una cuadrícula, cada cuadrado en la cuadrícula contiene un solo pixel. Por ejemplo, una imagen con una resolución de 500 x 300 significa que dicha imagen está conformada por una matriz de pixeles con 500 filas y 300 columnas. La mayoría de los pixeles se representan de dos maneras: escala de grises y color. En una imagen en escala

de grises, cada pixel tiene un valor entre 0 y 255, donde cero corresponde al color negro y 255 al color blanco. Los valores entre 0 y 255 representan variaciones de los tonos de gris, donde los valores cercanos a 0 son más oscuros y los cercanos a 255 son más claros. Los pixeles de color se representan normalmente en espacios de color; el más utilizado de ellos es el denominado RGB, compuesto por un valor para el componente rojo, uno para el verde y uno para el azul. Cada uno de estos tres colores está representado por un número entero en el rango 0 a 255. Dado que el valor de pixel solo necesita estar en el rango [0, 255] se usa normalmente un entero sin signo de 8 bits para representar cada intensidad de color. Como referencia, a continuación, se muestran en la tabla 2.1 ejemplos de colores y su vector RGB.

Tabla 2.1: Codificación de los colores

COLOR	**VECTOR RGB**
NEGRO	(0, 0, 0)
ROJO	(255, 0, 0)
AZUL	(0, 0, 255)
MARRON	(128, 0, 0)
AMARILLO	(255, 255, 0)
BLANCO	(255, 255, 255)
VERDE	(0, 255, 0)
FUCSIA	(255, 0, 255)
MORADO	(128, 0, 128)
AZUL MARINO	(0, 0, 128)

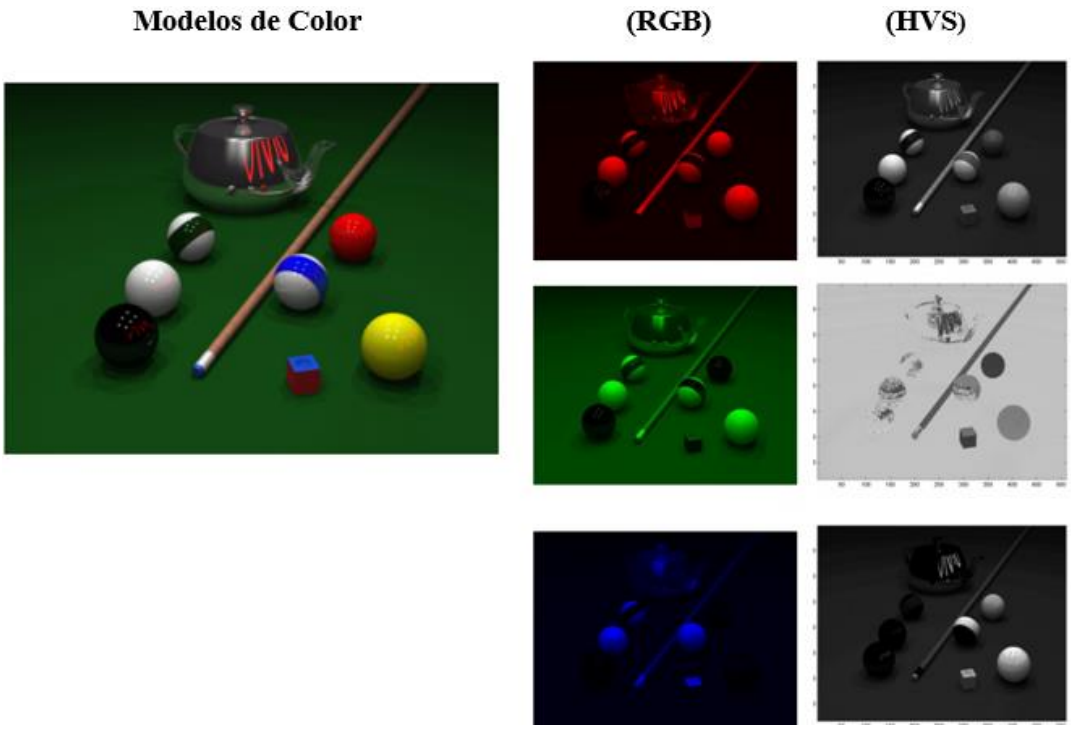
Elaboración Propia

Al formar un vector RGB (rojo, verde, azul), este vector representa al color; así, para construir un color blanco, un vector tendría cada uno de los colores rojo, verde y azul, quedando: (255, 255, 255) y luego, para un color negro, se tendría el vector (0, 0, 0). De esa manera cada pixel queda definido en toda la grilla de la imagen; esta grilla se convierte en un sistema coordenado, empezando siempre en la esquina superior izquierda, con la coordenada del punto (0, 0). Partiendo de este punto, los valores de los ejes x e y configuran las regiones y las diversas operaciones numéricas que se pueden aplicar a la imagen. (Hearn, 2014)

2.3.3. ESPACIOS DE COLOR

Entre los espacios de color más utilizados para el procesamiento de imágenes están el rojo, el verde y el azul (RGB). Según la Comisión Internacional de Iluminación, en su norma técnica CIE 1931-RGB, el color se conforma por tres dimensiones monocromáticas perceptibles al campo visual. Sin embargo, estos no son los únicos espacios de color; existen, entre otros, los espacios HSV (Hue-Saturation-Value) y los espacios L*a*b (es un espacio de color independiente del dispositivo que incluye todos los colores que pueden ser percibidos por los humanos.), más orientados a describir la percepción del color ante un cambio en alguna dimensión de la imagen. La elección de un espacio de color está en función del uso que se desea para la imagen, en caso de que fuera necesario hacer un análisis en relación con la superficie donde se utilizará. En este tipo de aplicaciones se emplea la combinación CMyK (Cyan, Magenta y Key) que tiene mucha utilidad para la comparación de patrones de impresión. Para el campo de procesamiento y análisis de pixeles en una imagen se utiliza en especial el espacio de color RGB. En la figura 2.1 se muestran diversas aplicaciones de espacio de color para una misma imagen. (Lamego Castro, 2017)

Figura 2.1: Espacios de Color



Elaboración Propia

Código de colores para el espacio de color utilizados en OpenCV son las siguientes:

- CV_RGB2GRAY. Conversión de RGB a escala de Grises
- CV_BGR2HSV. Conversión de BGR a escala HSV
- CV_RGB2HSV Conversión de escala RGB a escala HSV
- CV_HSV2BGR Conversión de HSV a escala BGR
- CV_HSV2RGB Conversión de HSV a escala RGB
- CV_BGR2HLS Conversión de BGR a escala HLS
- CV_RGB2HLS Conversión de RGB a escala HLS
- CV_HLS2BGR Conversión de HLS a escala BGR
- CV_HLS2RGB Conversión de HLS a escala RGB

los cuales estarán en el algoritmo, cumplen la función de conversión de colores.

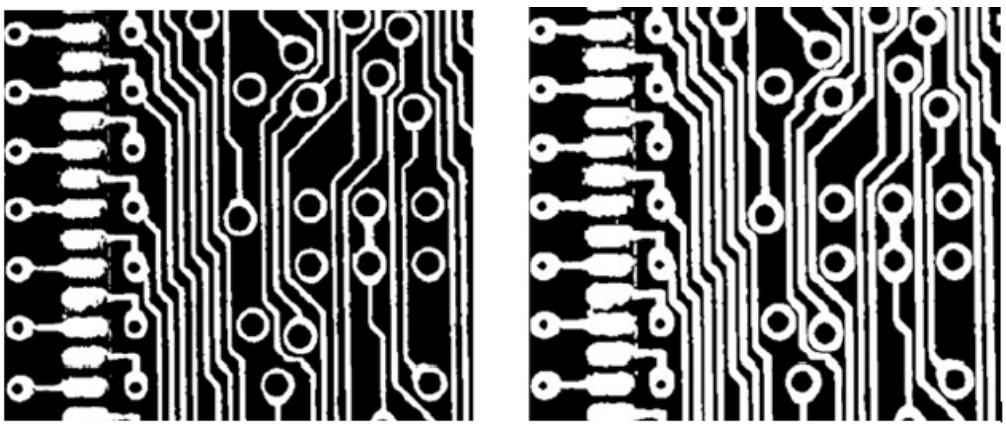
2.4. TRANSFORMACIONES MORFOLÓGICAS EN OPENCV

Las transformaciones morfológicas son algunas operaciones simples basadas en la forma de la imagen. Normalmente se realiza en imágenes binarias. Necesita dos entradas, una es nuestra imagen original, la segunda se llama elemento de estructuración o kernel que decide la naturaleza de la operación. Dos operadores morfológicos básicos son la dilatación y la erosión. (Rodríguez, Figueredo, & Chica, 2018)

2.4.1. DILATACION

La dilatación es una técnica para expandir el número de píxeles del objeto, generalmente en todas las direcciones simultáneamente, el siguiente código "dilated = cv2.dilate(closing4, kernel, iterations = 1)" devuelve como resultado pequeños agujeros que se llenan y llena espacios estrechos entre grupos más grandes de puntos de ajuste. También aumenta el tamaño de los objetos (es decir, el número de puntos en el conjunto). Tenga en cuenta que la dilatación normal en un contexto de imagen utiliza un elemento estructurante isotrópico. La erosión usando un elemento estructurante isotópico de 3 × 3 como se muestra en la figura 2.2.

Figura 2.2: Imagen Binario antes (izquierda) y después de la dilatación (derecha)

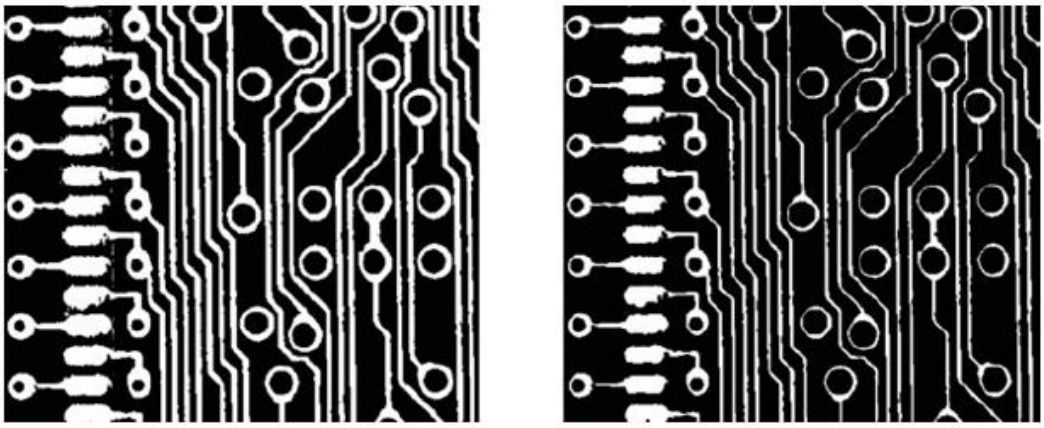


Fuente: (Howe, 2014, p. 61)

2.4.2. EROSION

La erosión es una técnica para reducir las formas de los objetos mediante la eliminación de píxeles del límite, el siguiente código “erode = cv2.erode(thresh1, kernel, iterations = 1)” da como resultado pequeños puntos de ruido y elimina cualquier característica estrecha. También reduce el tamaño de los objetos como se muestra en la figura 2.3 (es decir, el número de puntos en el conjunto). Similar a la dilatación, la erosión normal en un contexto de imagen utiliza un elemento estructurante isotrópico.

Figura 2.3: Binario antes (Izquierda) y después de Erosión (Derecha)



Fuente: (Howe, 2014, p. 63)

2.4.3. APERTURA Y CIERRE

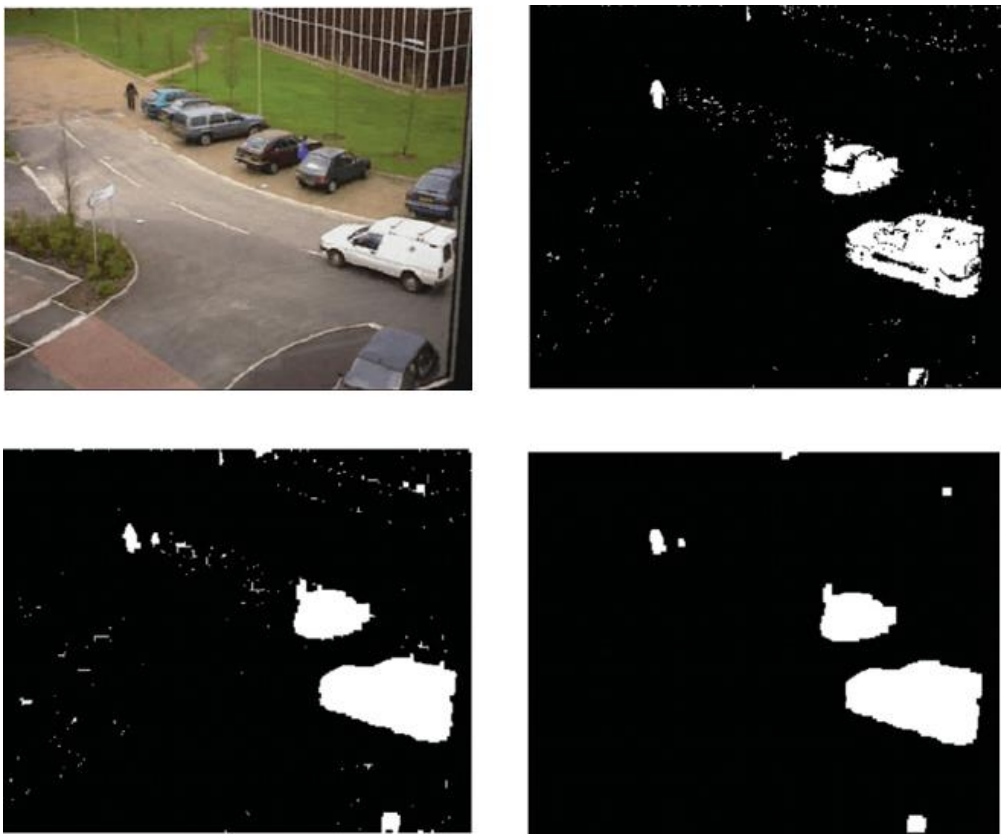
Apertura es solo otro nombre de erosión seguido de dilatación. Es útil para eliminar el ruido, como explicamos anteriormente. Y el cierre es al revés de Apertura, Dilatación seguida de Erosión. Es útil para cerrar pequeños orificios dentro de los objetos en primer plano o pequeños puntos negros en el objeto. (Reyes, 2017)

En OpenCV La apertura y el cierre están invocados por la siguiente función:

“opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)”

"closing = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)" y como resultado nos muestra la siguiente figura 2.4 donde la imagen binaria (arriba a la derecha) que se ha determinado restando la imagen actual (arriba a la izquierda) de una imagen de fondo, convirtiéndola en escala de grises y umbral. Esa imagen binaria se cierra para rellenar huecos y pequeños espacios (abajo a la izquierda), y luego se abre para eliminar regiones pequeñas y estrechas (abajo a la derecha).

Figura 2.4: Apertura y Cierre para obtener resultados requeridos



Fuente: (Howe, 2014, p. 64)

2.5. BORDES

La segmentación de una imagen es el proceso de dividir la imagen en piezas tal que diferentes objetos o partes de objetos son separados. Este es un paso de procesamiento esencial en la comprensión de la imagen (donde el objetivo es comprender el contenido de las imágenes). Hay dos formas principales de abordar la segmentación de imágenes:

- Procesamiento de bordes, donde identificamos las discontinuidades (los bordes) en las imágenes.
- Procesamiento de regiones, donde buscamos regiones homogéneas (o secciones) de las imágenes. La visión binaria es un ejemplo muy simple de procesamiento de regiones.

2.5.1. DETECCIÓN DE BORDES

La detección de bordes se desarrolló y normalmente se presenta para imágenes de un solo canal (escala de grises). Los bordes son las ubicaciones donde el brillo cambia abruptamente y, por lo tanto, a menudo se consideran desde un punto de vista de derivados 2D. Debido a que realizamos nuestro procesamiento en un dominio discreto, generalmente representamos las ubicaciones donde el brillo cambia abruptamente como píxeles de borde (aunque el cambio de brillo a menudo será entre píxeles en lugar de en ellos). como muestra de detección de bordes Sobel se muestra en la Figura 2.5. Donde se muestra una imagen en escala de grises de una iglesia (izquierda), la magnitud del gradiente del borde Sobel (centro izquierda), una versión de la magnitud del gradiente (centro derecha) y las orientaciones Sobel correspondientes (derecha)

Figura 2.5: Detección de Bordes

Fuente: (Howe, 2014, p. 84)

2.5.2. DETECCIÓN DE BORDE CANNY

La detección de bordes es una técnica muy utilizada que nos permite aislar los objetos y separarlos del fondo. Una vez obtenido los bordes, lo único que nos faltaría es detectar los diferentes contornos para poder contar los objetos. Poder contar tornillos, tuercas, monedas o cualquier otro objeto es relativamente sencillo gracias a OpenCV. La Visión Artificial es un subcampo de las matemáticas que abarca muchas técnicas del tratamiento digital de imágenes (Gutiérrez, Salazar, Elías, & Lavalle, 2016)

En el área de procesamiento de imágenes, la detección de los bordes de una imagen es de suma importancia y utilidad, pues facilita muchas tareas, entre ellas, el reconocimiento de objetos, la segmentación de regiones, entre otras. Se han desarrollado variedad de algoritmos que ayudan a solucionar este inconveniente. El algoritmo de Canny es usado para detectar todos los bordes existentes en una imagen. Este algoritmo está considerado como uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución y basado en la primera derivada. Los puntos de contorno son como zonas de píxeles en las que existe un cambio brusco de nivel de gris (Kaehler & Bradski, 2016)

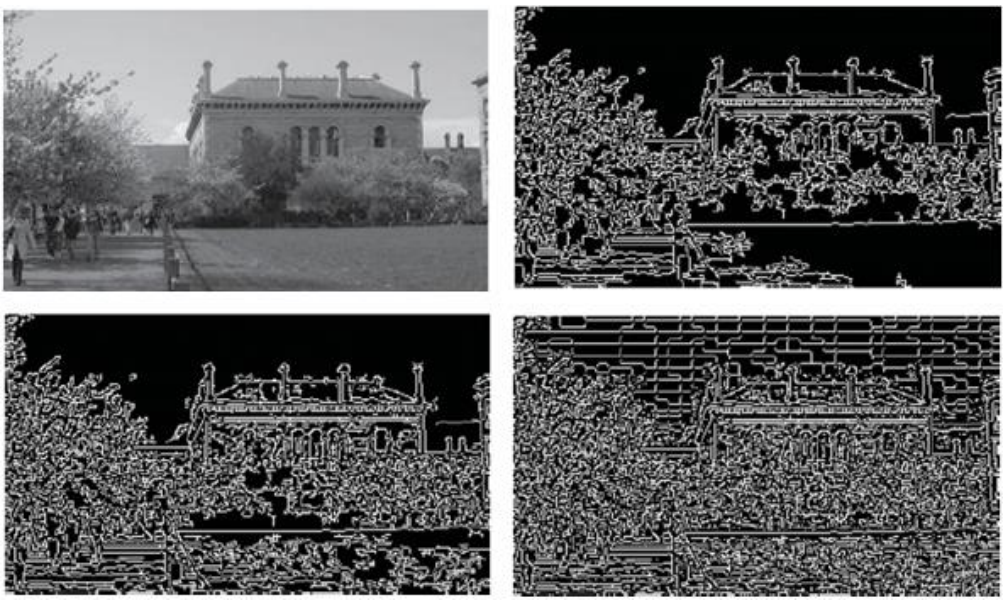
El detector de bordes Canny combina la detección de bordes de primera derivada y segunda derivada para calcular el gradiente y la orientación del borde. Fue diseñado para optimizar los siguientes tres criterios:

- Detección: no se deben perder los bordes.
- Localización: se debe minimizar la distancia entre los bordes reales y localizados.
- Una respuesta: minimiza las respuestas múltiples a una sola arista.

En OpenCV, se proporciona una función para calcular Canny, que requiere los umbrales alto y bajo. hay que tener en cuenta que esto solo puede procesar imágenes en escala de

grises como se muestra en la figura 2.6, Las tres imágenes de Canny usan el mismo filtro de suavizado pero varios umbrales de gradiente bajo y alto (7 y 236 - arriba a la derecha, 7 y 100 - abajo a la izquierda, 1 y 1 - abajo a la derecha). La imagen inferior derecha muestra todos los bordes posibles y los otros dos muestran algunos de los efectos de los umbrales.

Figura 2.6: Detección de bordes Canny



Fuente: (Howe, 2014, p. 97)

2.6. NUMPY

NumPy es una biblioteca Python que proporciona tipos de datos para almacenar de manera eficiente secuencias de valores numéricos y operar sobre ellos. NumPy almacena estos valores numéricos con un tamaño fijo y los almacena en regiones contiguas de memoria, lo que permite una comunicación sencilla con otros lenguajes de programación como C, C++ y Fortran (de hecho, varias funciones de la biblioteca están escritas en estos lenguajes para maximizar el rendimiento). Sobre estos datos, NumPy permite realizar operaciones matemáticas del álgebra lineal o algoritmos más avanzados como la transformada de Fourier. Desde el punto de vista de tipos de datos, NumPy ofrece ndarray,

un array ndimensional de elementos el mismo tipo. El acceso a estos elementos se realiza de manera natural mediante sus coordenadas en el espacio n-dimensional. NumPy ofrece distintos tipos de datos numéricos (Caballero Roldán, Martín Martín, & Riesco Rodríguez, 2018)

Entre los que destacan:

- int8, int16, int32 e int64: enteros con signo de 8, 16, 32 y 64 bits.
- uint8, uint16, uint32 e uint64: enteros sin signo de 8, 16, 32 y 64 bits.
- float16, float32 y float64: números en coma flotante de 16, 32 y 64 bits.
- complex64 y complex128: números complejos formados por dos números en coma flotante de 32 y 64 bits, respectivamente.

2.6.1. USANDO NUMPY

Antes de trabajar en las bibliotecas OpenCV, también se debe importar la biblioteca NumPy, incluidos los tipos de datos, funciones y sintaxis para trabajar con imágenes. El siguiente es el código para crear una matriz n-dimensional:

```
1 import numpy as np

2 newlist = [1,2,3]

3 type(newList)

4 newArray = np.array(newList)

5 type(newArray)
```

Describiendo cada línea de código se interpreta:

Línea 1: Importa la biblioteca NumPy.

Línea 2: Crea un nuevo objeto de lista en Python. Una lista está representada por llaves cuadradas, como en [y].

Línea 3: muestra el tipo de datos del objeto como lista.

Línea 4: utiliza la función de matriz () de NumPy para crear una nueva matriz utilizando el objeto de lista existente. Una matriz se representa entre paréntesis. Entonces, esta función crea un objeto llamado newArray que es una matriz inicializada con una sola fila y tres columnas, representada como ([1,2,3]).

Línea 5: muestra este nuevo objeto como numpy.ndarray.

La función ceros () o unos () en las bibliotecas NumPy se puede usar para crear una matriz n-dimensional. El siguiente código ayuda a crear una matriz 3 × 2 iniciada con ceros ():

- np.zeros(shape=(3,2))
- np.ones((2,4))

los resultados de los códigos ejecutados se muestran a continuación:

array ([[0.,0.], [0.,0.], [0.,0.]])

array ([[1.,1.,1.,1.], [1.,1.,1.,1.]])

Numpy es una herramienta fundamental que se emplea en el desarrollo de este proyecto y se detalla en el algoritmo del siguiente capítulo.

2.7. MATPLOTLIB

Matplotlib es una biblioteca de trazado 2D de Python que produce cifras de calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas. Matplotlib se puede usar en scripts de Python, los shells de Python e IPython, el cuaderno Jupyter, los servidores de aplicaciones web y cuatro kits de

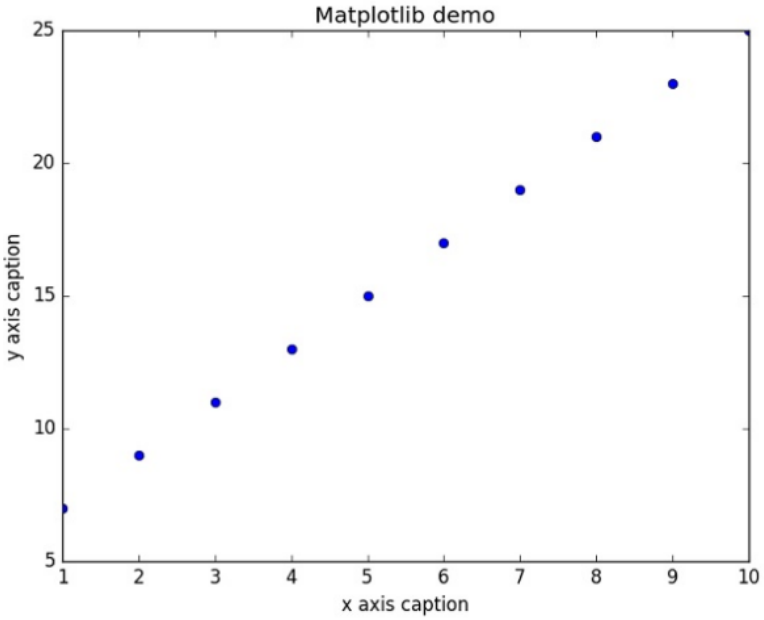
herramientas de interfaz gráfica de usuario. Matplotlib intenta hacer que las cosas fáciles sean fáciles y las cosas difíciles posibles. Puede generar gráficos, histogramas, espectros de potencia, gráficos de barras, gráficos de errores, gráficos de dispersión, etc., con solo unas pocas líneas de código. (Hunter, Dale, Firing, & Droettboom, 2019)

El siguiente código ejecuta y muestra resultados de la siguiente función **y = 2 * x + 5**

```
1: import numpy as np
2: from matplotlib import pyplot as plt
3: x = np.arange(1,11)
4: y = 2 * x + 5
5: plt.title("Matplotlib demo")
6: plt.xlabel("x axis caption")
7: plt.ylabel("y axis caption")
8: plt.plot(x,y,"ob")
9: plt.show()
```

La explicación de cada línea de código es la siguiente línea 1 importa la librería Numpy, La línea 2 Importa la librería Matplotlib, La línea 3 define el rango del gráfico, la línea 4 define la función Y, La línea 5 imprime el título del gráfico, La línea 6 muestra valores en la coordenada X, La línea 7 muestra el valor de la coordenada Y, la línea 8 muestra el grafico de la función representando por puntos. "**ob**" muestra los círculos que representan puntos en lugar de líneas. La línea 9 en si es el grafico generado por todo el código y se muestra en la figura 2.7.

Figura 2.7: Grafica generado por las líneas de código



Fuente: https://www.tutorialspoint.com/numpy/numpy_matplotlib.htm

2.8. UMBRAL O DELIMITADO

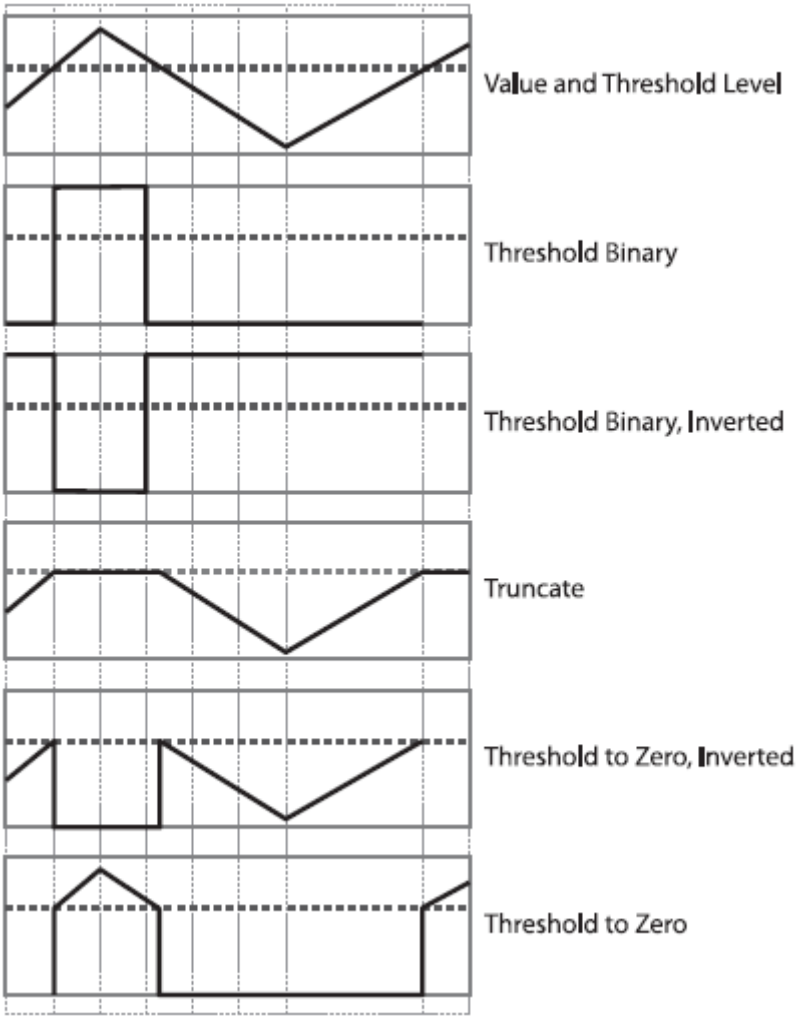
Esta función se usa para obtener una imagen binaria a partir de una imagen de escala de grises o también puede usarse para reducir “ruido”, es decir, filtrar valores demasiado pequeños o demasiado grandes de una matriz de datos de imagen. (Lamego Castro, 2017)

Con frecuencia, se realiza muchas capas de pasos de procesamiento y se quiere tomar una decisión final sobre los píxeles en una imagen o rechazar categóricamente esos píxeles por debajo o por encima de un valor mientras se conserva los otros. La función OpenCV cvThreshold () realiza estas tareas La idea básica es que se proporciona una matriz, junto con un umbral, y luego algo le sucede a cada elemento de la matriz dependiendo de si está por debajo o por encima del umbral (Kaehler & Bradski, 2016)

Se tiene varios tipos de Umbral como se muestra en la figura 2.8 ayudara a aclarar las implicaciones exactas de cada tipo de umbral, donde los resultados de variar el tipo

de umbral en cvThreshold (). La línea horizontal a través de cada gráfico representa un nivel de umbral particular aplicado al gráfico superior y su efecto para cada uno de los cinco tipos de operaciones de umbral.

Figura 2.8: Tipos de Umbral



Fuente: (Kaehler & Bradski, 2016, p. 136)

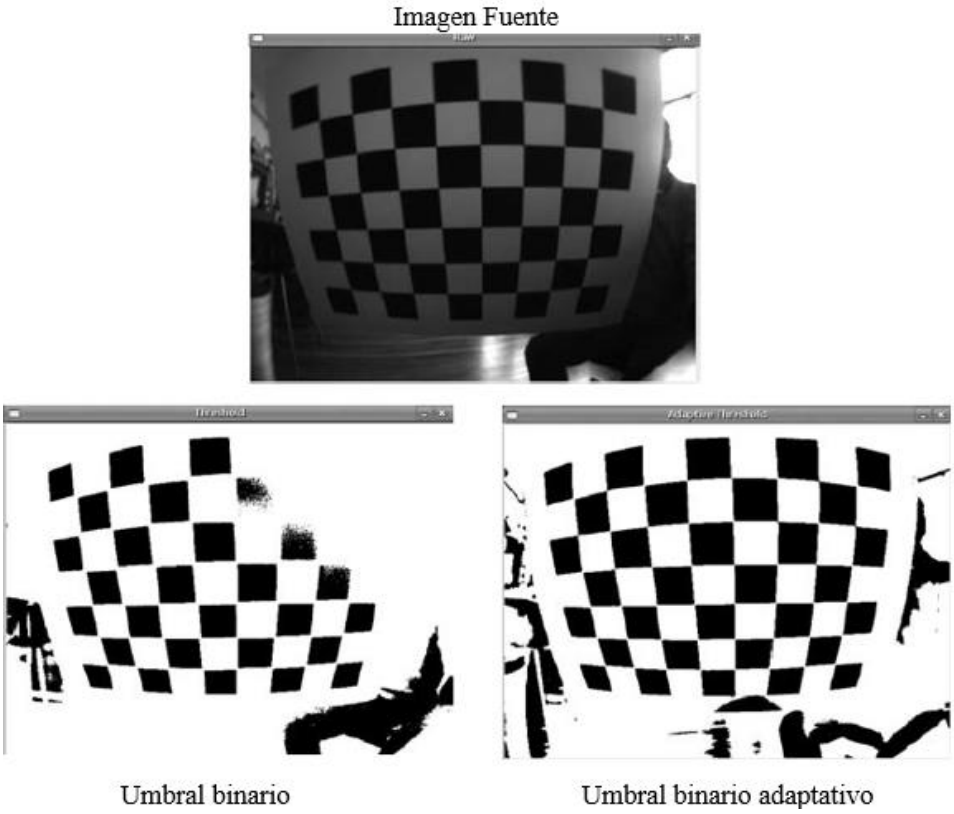
2.8.1. UMBRAL ADAPTATIVO

Existe una técnica de umbral modificada en la que el nivel de umbral es variable en sí mismo. En OpenCV, este método se implementa en cvAdaptiveThreshold () permite dos tipos de umbral adaptativo diferentes según la configuración de método adaptativo. La técnica de umbral adaptativo es útil cuando hay fuertes gradientes de iluminación o

reflectancia que necesita umbral en relación con el gradiente de intensidad general. Esta función maneja solo imágenes de un solo canal de 8 bits o de punto flotante, y requiere que las imágenes de origen y destino sean distintas (Kaehler & Bradski, 2016)

En la figura 2.9 se visualiza la comparativa de umbral binario versus umbral binario adaptativo: la imagen de entrada (arriba) se convirtió en una imagen binaria usando un umbral global (abajo a la izquierda) y un umbral adaptativo (abajo a la derecha)

Figura 2.9: Comparativa de Umbral Binario y Adaptativo



Fuente: (Kaehler & Bradski, 2016, p. 139)

2.9. HISTOGRAMA

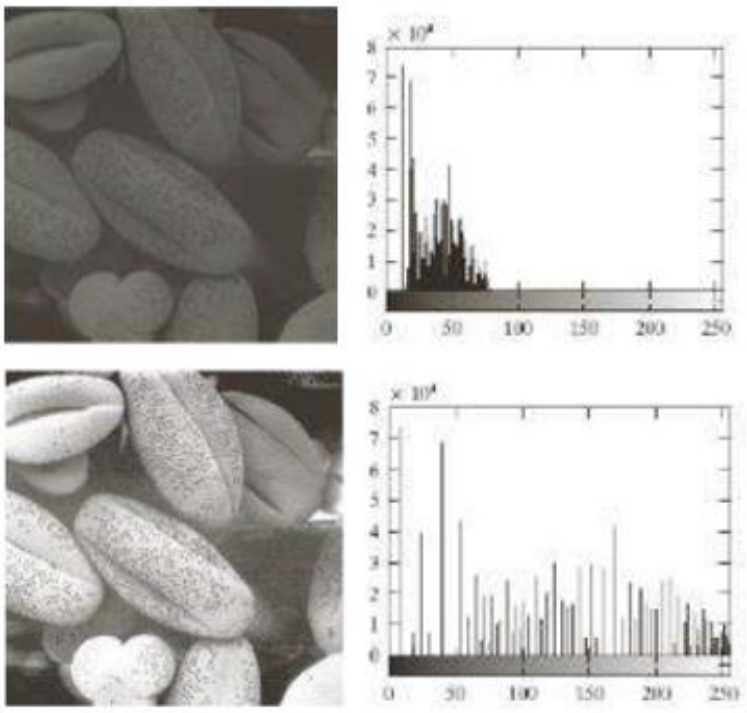
Un histograma representa la distribución de las intensidades de los pixeles (color o escala de grises) en una imagen. Se puede visualizar en un histograma el nivel de la intensidad (valor de pixel) del color. En un espacio de color RGB en los valores de pixel

estarán en el rango de 0 a 255. Al trazar el histograma, el eje X con 256 bins, entonces estamos contando efectivamente el número de veces que cada pixel produce este valor. En contraste, si se utilizan solo dos segmentos (espaciados equitativamente), entonces se estaría contando el número de veces que un pixel está en el rango [0, 128] o [128, 255]. Simplemente, al examinar el histograma de una imagen se obtiene una comprensión general con respecto al contraste, brillo y distribución de intensidad. (Taquía Gutiérrez, 2017)

La información que se obtiene del histograma de una imagen se utiliza para su descripción global. El histograma no proporciona información acerca del origen de los pixeles que lo conforman, es decir se pierde la información espacial de los pixeles en la imagen, por lo que resulta imposible reconstruir una imagen a partir de su histograma. Sin embargo, el histograma muestra características muy importantes de una imagen, como el contraste y el rango dinámico. El proceso de ecualización del histograma básicamente es la redistribución de los niveles de gris de la imagen por la reasignación de los valores de brillo de los píxeles, sin embargo, el ruido surge como un problema para la ecualización, debido a que este aumenta el contraste y distorsiona los distintos objetos que forman la imagen (Viera Maza, 2017)

La figura 2.10 muestra el histograma de una imagen normal a escala grises y una ecualizada. (Izquierda superior) Imagen de entrada, y (derecha superior) su histograma. (Izquierda inferior) Imagen con histograma ecualizado, y (Derecha inferior) su histograma, la mejora entre izquierda superior e izquierda inferior es evidente.

Figura 2.10: Ilustración de ecualización de histogramas



Fuente: (Viera Maza, 2017, p. 68)

Trabajando con una representación de histograma en OpenCV Hay otra representación para las imágenes, y ese es un histograma. Como se muestra los siguientes códigos.

```
1:  import numpy as np
2:  import cv2
3:  from matplotlib import pyplot as plt
4:  image = cv2.imread("./images/panda.jpg")
5:  #plot a histogram
6:  histogram_image = cv2.calaHist([Image], [0], done, [256], [0,256])
7:  #flaten the histogram
8:  plt.hist(histogram_image.ravel(), 256, [0,256])
```

```
9:  plt.show()

10:  #view color channels

11:  color = ['b','g','r']

12:  #seperate the colors and plot the histogram

13:  for I, col in enumaerate(color):

14:  hist = cv2.calcHist([image], [i], None, [256], [0,256])

15:  plt.plot(hist, color = col)

16:  plt.xlim([0.256])

17:  plt.show ()
```

Describiendo línea por línea el código ejecutado se tiene:

Línea 1: Importa la biblioteca NumPy. Esto es importante ya que el formato de matriz de la imagen está representado por el tipo de datos NumPy ndarray (matriz n-dimensional).

Línea 2: Importa la biblioteca OpenCV que da acceso a todas las funciones para operar en imágenes.

Línea 3: Importa matplotlib, que tiene bibliotecas para trazar un histograma.

Línea 4: Lee la imagen panda.jpg que se acaba de colocar en la carpeta de imágenes en la ruta predeterminada de Python. Esta línea lee la imagen y la almacena en una variable llamada imagen.

Línea 5: es un comentario, cada vez que se antepone un numeral denota comentario.

Línea 6: genera un histograma de la imagen que se carga.

Línea 7: es otro comentario

Línea 8: aplana el histograma.

Línea 9: muestra el histograma

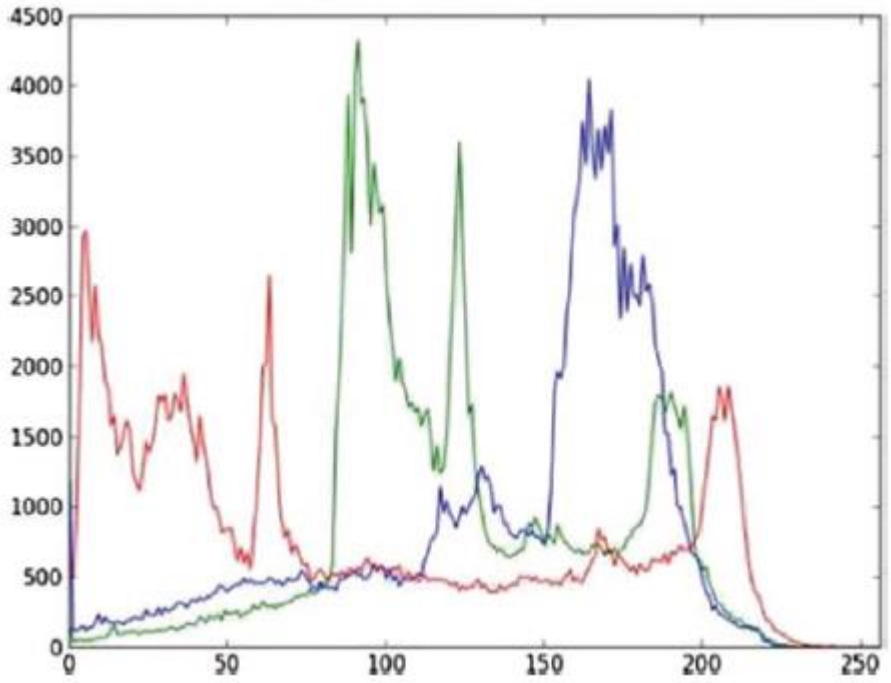
Línea 10: Comentario

Línea 11: colores que mostrara en el grafico

Línea 13-16: repite los valores en el histograma aplanado, separa los colores y traza los datos.

Línea 17: muestra el histograma. Como se aprecia en la figura 2.11

Figura 2.11: Histograma RGB



Fuente: (Gollapudi, 2019, p. 46)

CAPITULO III

MATERIALES Y METODOS

3.1. MATERIALES

3.1.1. HARDWARE

- Ordenador para programación (laptop).

  Modelo: HP 250

  Procesador: Intel(R) Core (TM) i5-2430 2.40GHz

  Memoria instalada (RAM): 4.00GB de RAM.

  Tipo de sistema: Sistema Operativo de 64 bits Windows 10

- Cámara Rapberry Pi 3 de 2592 x 1944 pixel

3.1.2. SOFTWARE.

- Sistema Operativo de 64 bits Windows 10.
- Microsoft Word 2013 profesional.
- OpenCV
- Python 3.7.3.
- Block de notas de Windows.

3.2. DISEÑO, NIVEL Y TIPO DE LA INVESTIGACION

3.2.1. DISEÑO DE LA INVESTIGACIÓN

El método planteado es Descriptivo - experimental, porque se trata de mostrar las características de una nueva investigación, brindando una mejor optimización de la tecnología, sus productos y materiales, este nivel de investigación se guía por la experimentación pues es la única metodología que permite establecer relaciones de causa efecto entre variables.

3.2.2. NIVEL DE LA INVESTIGACIÓN

El nivel de investigación experimental se refiere a la profundidad del conocimiento que se busca lograr con la investigación, por tanto, el nivel de la presente investigación es exploratoria, señalando que las investigaciones exploratorias buscan abrir nuevos caminos en el desarrollo del conocimiento humano. Y la presente investigación siendo un diseño de un sistema busca abrir un camino para un nuevo método siendo una alternativa el diseño de un algoritmo de procesamiento de imágenes para brindar soluciones al sistema de pesaje para el control automático de una faja transportadora en la unidad minera Mallay.

3.3. POBLACIÓN Y MUESTRA DE LA INVESTIGACIÓN

3.3.1. POBLACIÓN

Para el presente trabajo de investigación, la población está constituida por la cantidad de cuatro imágenes tomadas del mineral en la faja transportadora de la unidad minera Mallay

3.3.2. MUESTRA

Se define la muestra como parte que se estudia y es representativa de la población, es decir un segmento que tiene las características y propiedades de la población, por tanto, la muestra es, no probabilístico, representa los puntos de medición localizadas en el lugar de procesamiento de fotografías, material que ingresa al sistema de pesaje, control de mineral y faja transportadora de minerales en el área del terreno de la planta de procesamiento unidad minera Mallay.

3.4. UBICACIÓN Y DESCRIPCIÓN DE LA INVESTIGACIÓN

3.4.1. UBICACIÓN

La investigación, tanto pruebas y diseño se desarrollará en el laboratorio de Electrónica General, de la Escuela Profesional de Ingeniería Electrónica de la Universidad Nacional

del Altiplano. Las pruebas del diseño y su aplicación masiva se desarrollarán en la unidad minera Mallay. se encuentra ubicado el paraje de Mallay distrito de Oyón, provincia de Oyón, departamento de Lima a una elevación comprendida entre 4,090 y 4,470 msnm, en la vertiente occidental de la Cordillera de los Andes, en la cuenca del río Huaura al Suroeste de la ciudad del mismo nombre. Los campamentos actualmente están ubicados a 4,250 msnm; las cumbres sobrepasan los 5,000 msnm siendo la boca mina más baja el nivel 4090.

3.4.2. DESCRIPCIÓN DE LA INVESTIGACIÓN

En la presente investigación se pretende desarrollar o diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora en la unidad minera Mallay, esta investigación consta de cuatro etapas que se describen a continuación:

- 1ª etapa: en esta etapa trata esencialmente búsqueda de información, consulta bibliográfica y sitios web, sobre plataformas de desarrollo, elección de componentes y técnicas o métodos implementación para el diseño.
- 2ª etapa: comprende el inicio del desarrollo del algoritmo con el uso de Python OpenCV.
- 3º etapa: ya realizadas los procesos de obtención del algoritmo de procesamiento de imágenes se busca adaptar en planta para las pruebas de funcionamiento.
- 4ª etapa: comprende la obtención y el proceso de verificación, que consiste en analizar los resultados, para luego ver el funcionamiento y su posterior corrección.

3.5. TECNICAS E INSTRUMENTOS DE RECOLECCION DE DATOS

La recolección de datos se refiere a cómo y qué medios se usan para la obtención de la información que será de utilidad para la corroboración de nuestras hipótesis.

3.5.1. TECNICAS

- Revisión bibliográfica y de base de datos:

  Esta técnica permitirá recolectar y sistematizar la información requerida de; papers, foros, blogs, libros, tesis, video-tutoriales y más fuentes de información publicadas en Internet.

- Observación:

  El empleo de esta técnica fue importante para recabar información primaria, debido a que se pudo interactuar con la realidad a través de la observación como capturas a pantalla, proceso de diseño del algoritmo, apuntes de simulaciones y fotografías.

3.6. DESARROLLO DEL ALGORITMO

3.6.1. ADQUISICIÓN DE LAS IMÁGENES

Con el objetivo de desarrollar el algoritmo de visión Artificial, se tomó las imágenes con una cámara Raspberry Pi 3 el cual fue adaptado y cubierto con protectores para evitar algún daño, la obtención de las imágenes se realizó en plena producción como se puede apreciar en la figura 3.1, donde se obtuvieron como muestra 4 imágenes en formato jpeg, de dimensión 2592 × 1944, ya que no presentaron distorsión, este procedimiento se realizó para el experimento que se detalla a continuación.

Figura 3.1: Adaptación de la cámara a la faja transportadora

Elaboración Propia

3.6.2. IMAGEN PARA LA PRIMERA MUESTRA

Para el desarrollo del algoritmo se toma la imagen con nombre IMAGEN16 de la faja transportadora, posterior a esto se realiza el procesamiento de imagen con OpenCV de este modo filtrar y separar las estructuras metálicas de la faja para luego determinar el área y volumen de los minerales en la faja, mediante este proceso se desarrolló el algoritmo, para este caso se tiene la primera imagen a procesar como se muestra en la figura 3.2.

Figura 3.2: Imagen para la primera muestra

Elaboración Propia

3.6.2.1 Desarrollo del código de la primera Muestra

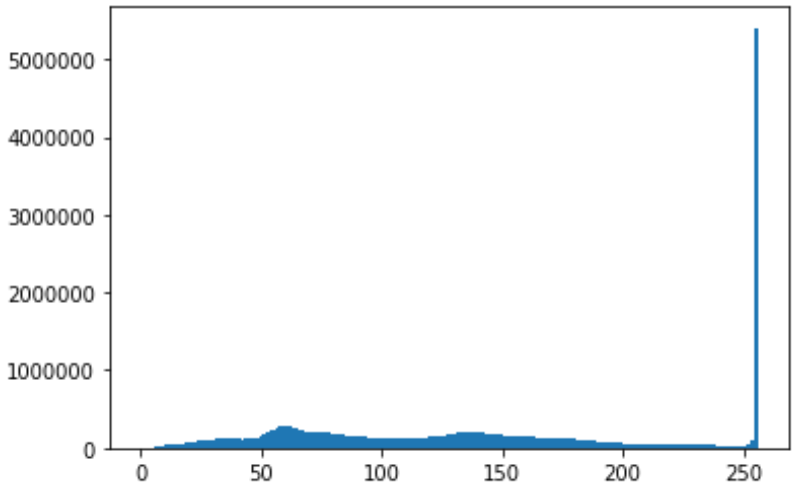
Una vez obtenido la imagen con la cámara, el siguiente paso es desarrollar el algoritmo de procesamiento, para ello se ejecutó las líneas de código que se encuentran en el apartado de ANEXOS.

3.6.2.2 Interpretación de las líneas de código desarrollado en la primera muestra, que se encuentran en el ANEXO 1

Las líneas de código se describen de la siguiente manera:

- Las líneas de código (1-5) del algoritmo importa las librerías Numpy, OpenCv, Matplotlib y tiempo respectivamente además de imprimir la versión de OpenCV.
- La línea de código 6 del algoritmo es para empezar a contar el tiempo que demorara todo el proceso.
- Las líneas de código (7-14) del algoritmo, definen las funciones para mostrar la imagen.
- La línea de código 15 del algoritmo, muestra la ubicación de la imagen que se está procesando
- Las líneas de código (16-19) del algoritmo, muestran la información de la imagen.
- Las líneas de código (20-21) del algoritmo grafica el histograma de la imagen, como se muestra en la figura 3.3.

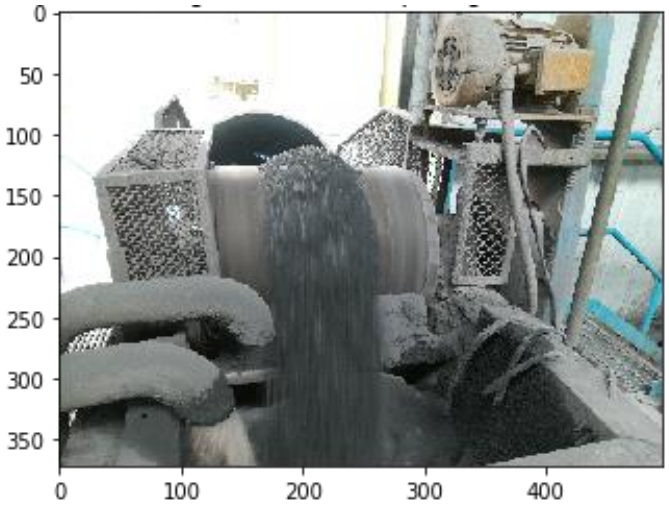
Figura 3.3: Gráfica del Histograma



Elaboración Propia

- La línea de código 22 del algoritmo, reduce la imagen 8 veces de la original y este se muestra en la figura 3.4.

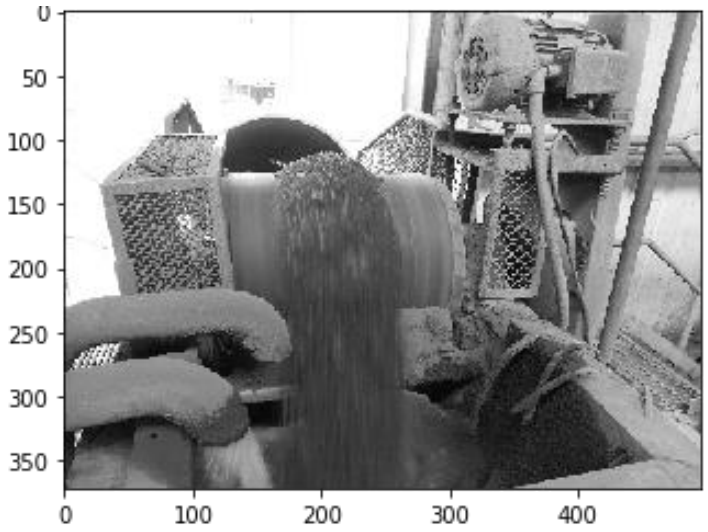
Figura 3.4: Reducción de la Imagen



Elaboración Propia

- Las líneas de código (23-27) del algoritmo, convierten la imagen de RGB a escala grises a la vez reduce 8 veces el tamaño de la imagen original, esta imagen reducida es la figura 3.5.

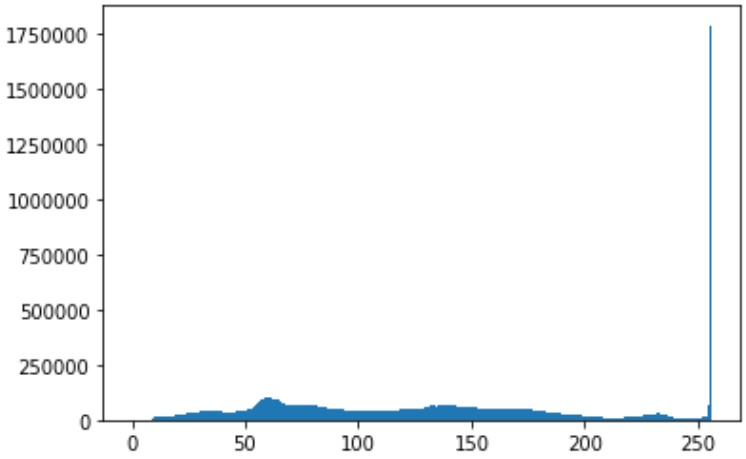
Figura 3.5: Conversión de la imagen RGB a Grises



Elaboración Propia

- Las líneas de código (28-29) del algoritmo, grafica el histograma de la imagen en escala de grises, se visualiza en la figura 3.6.

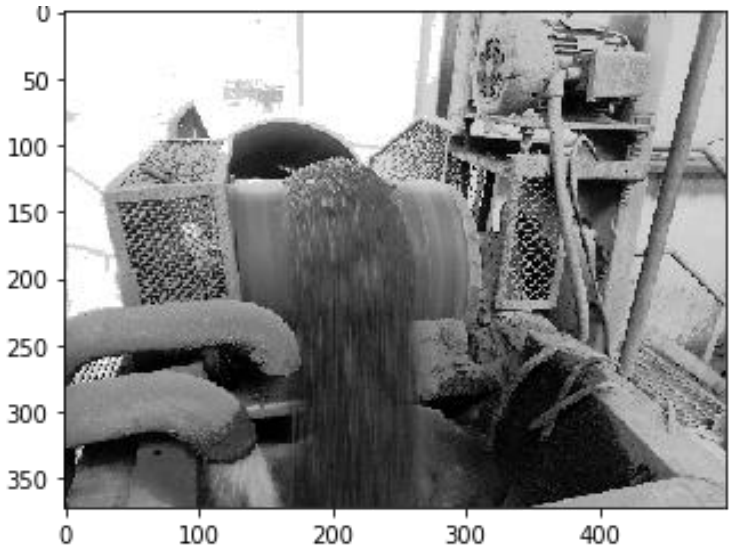
Figura 3.6: Gráfica del histograma de la imagen en grises



Elaboración Propia

- Las líneas de código (30-33) del algoritmo, ecualiza la imagen en escala grises y reduce 8 veces su tamaño original donde la figura 3.7 demuestra esa función.

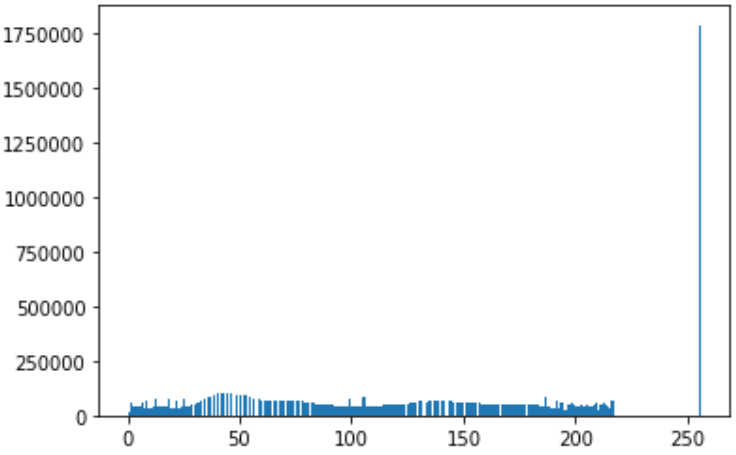
Figura 3.7: Ecualización de la imagen Recortada



Elaboración Propia

- Las líneas de código (34-35) del algoritmo, grafica el histograma de la imagen en grises ecualizada, figura 3.8.

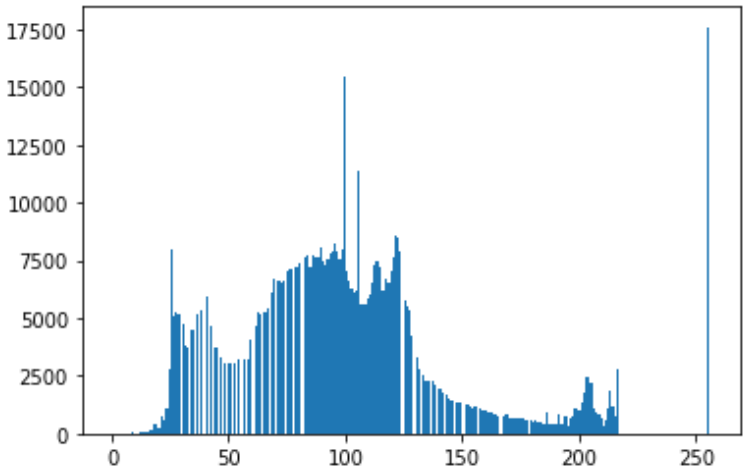
Figura 3.8: Gráfica del histograma de imagen ecualizada



Elaboración Propia

- Las líneas de código (36-39) del algoritmo, obtienen la información de la nueva imagen y tamaño correspondiente.
- Las líneas de código (40-42) del algoritmo, grafica el histograma de la imagen recortada, que representa la figura 3.9 con alto de 900 ancho de 700 respectivamente.

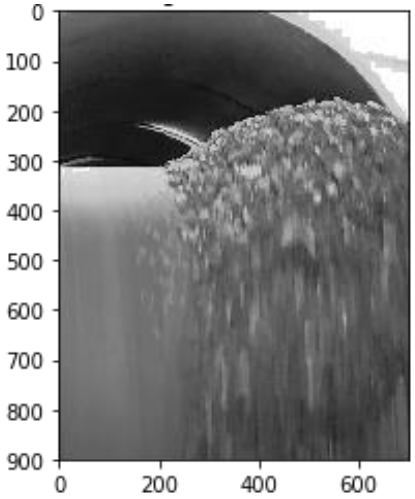
Figura 3.9: Gráfica del histograma de la imagen recortada



Elaboración Propia

- Las líneas de código (43-45) del algoritmo, muestra la imagen nuevamente recortada, como se aprecia en la figura 3.10.

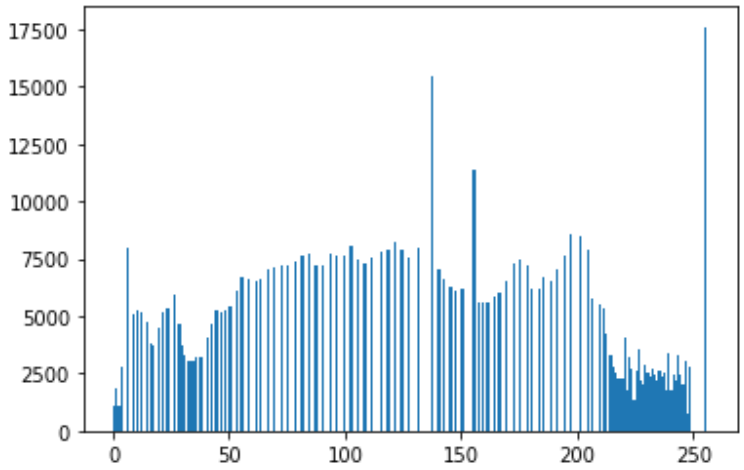
Figura 3.10: Imagen Recortada nuevamente



Elaboración Propia

- La línea de código (46-48) del algoritmo, grafica el histograma de esta imagen recortada, que representa una mejor distribución de los parámetros de la imagen, como muestra la figura 3.11.

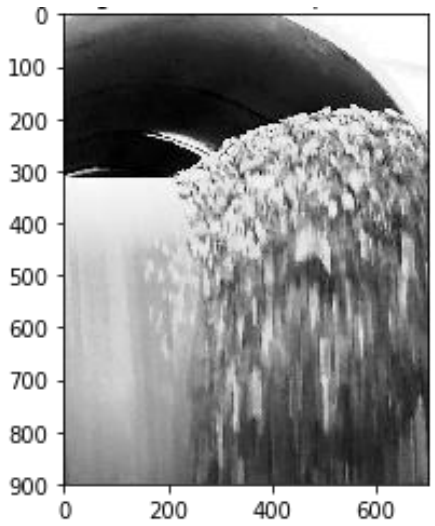
Figura 3.11: Gráfica del histograma de la imagen recortada



Elaboración Propia

- Las líneas de código (49-50) del algoritmo, muestra la imagen ecualizada, y se muestra en la figura 3.12.

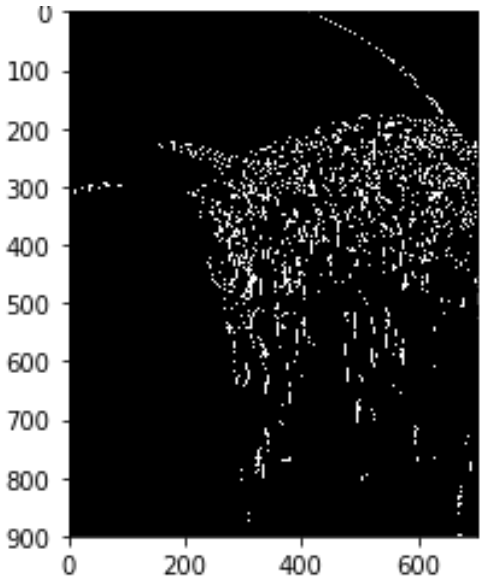
Figura 3.12: Imagen Recortada y ecualizada



Elaboración Propia

- Las líneas de código (51-55) del algoritmo, encuentra los bordes con el método Canny, esta grafica se muestra en la figura 3.13.

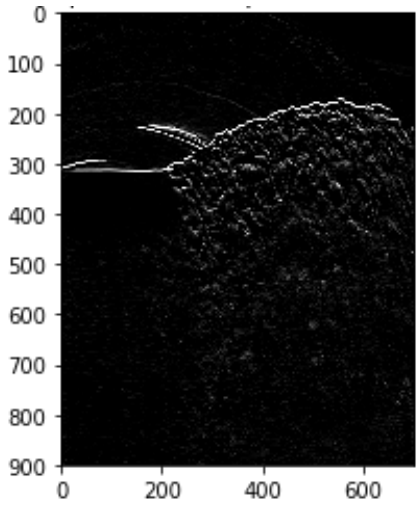
Figura 3.13: Detección de Bordes Canny



Elaboración Propia

- Las líneas de código (56-60) del algoritmo, aplica el segundo método de detección de bordes denominado Sobel y el resultado se muestra en la figura 3.14.

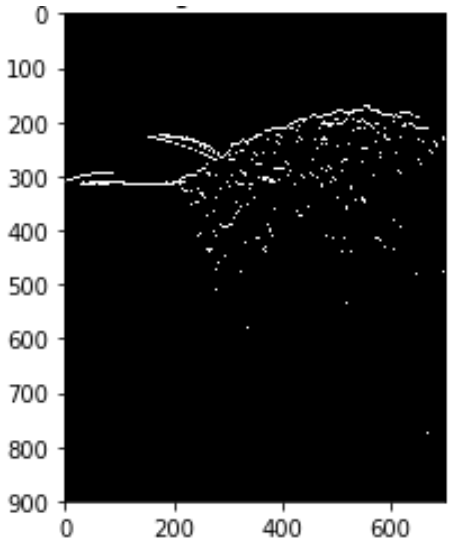
Figura 3.14: Detección de Bordes Sobel



Elaboración Propia

- Las líneas de código (61-66) del algoritmo, umbraliza la detección de bordes Sobel, este resultado se visualiza en la figura 3.15.

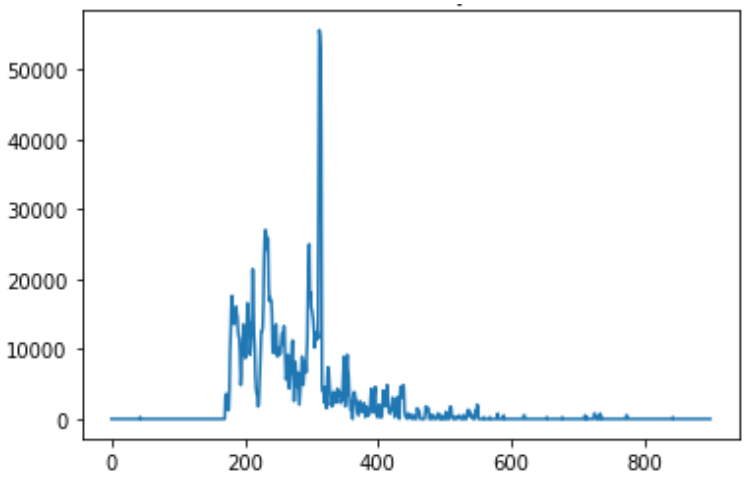
Figura 3.15: Threshold a la detección de bordes Sobel



Elaboración Propia

- Las líneas de código (65-69) del algoritmo, realiza la sumatoria de todos los puntos blancos en el eje horizontal, como muestra la figura 3.16.

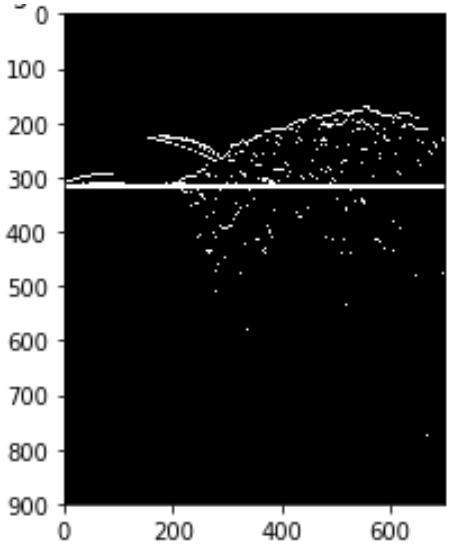
Figura 3.16: Sumatoria en el eje horizontal



Elaboración Propia

- Las líneas de código (71-74) del algoritmo, muestra la máxima position que es: 289 con 70125 pixeles.
- Las líneas de código (75-76) del algoritmo, encuentra la faja que es necesaria para calcular el mineral de la faja, esta detección se muestra en la figura 3.17.

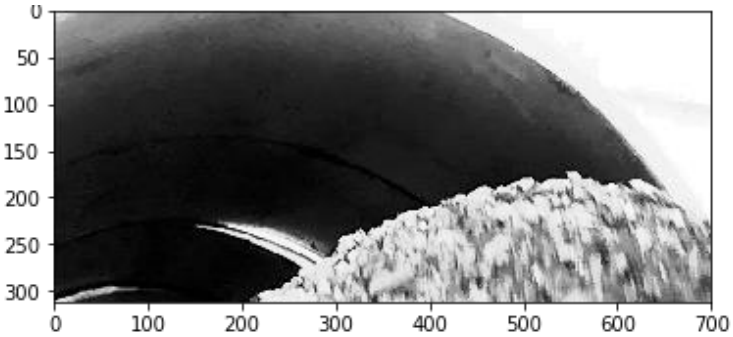
Figura 3.17: Identificación de la faja transportadora



Elaboración Propia

- Las líneas de código (77-83) del algoritmo, recorta la parte inferior de la imagen de esta forma descarta todo lo que está debajo de la línea detectada y se aprecia en la figura 3.18. también muestra la información de la imagen recortada, y obtiene la información de la imagen Tamaño: Alto: 289 Ancho: 700.

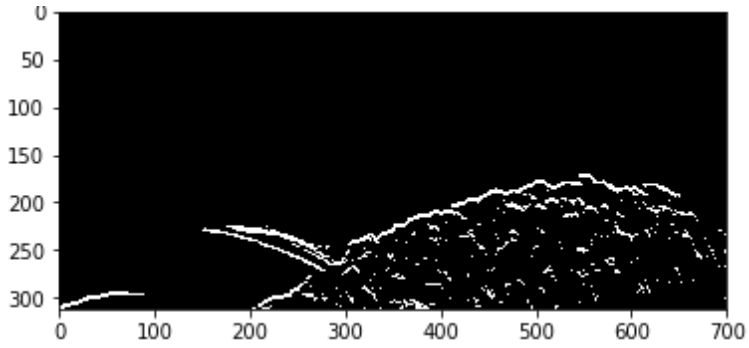
Figura 3.18: Imagen recortada desde la posición de la faja



Elaboración Propia

- Las líneas de código (84-88) del algoritmo, recupera la parte de la imagen buscada donde la figura 3.19 ayuda a verificar.

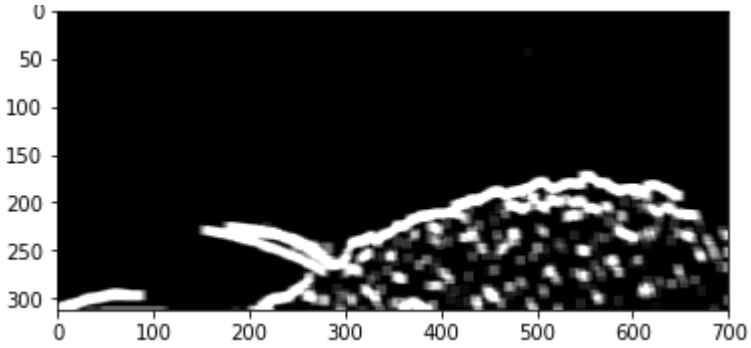
Figura 3.19: Imagen recuperada



Elaboración Propia

- La línea de código 89 del algoritmo, muestra la imagen recuperado suavizado y este es la figura 3.20.

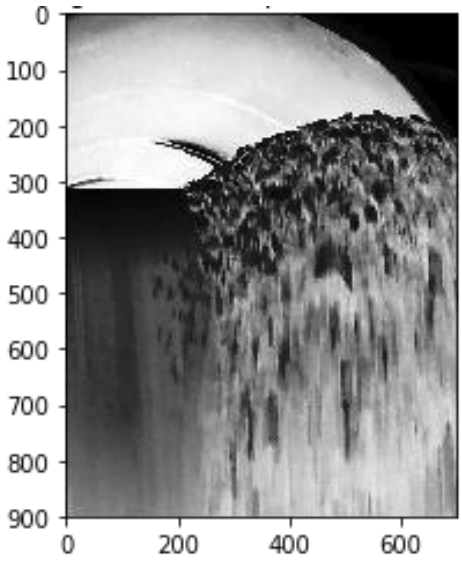
Figura 3.20: Suavizado de la imagen



Elaboración Propia

- Las líneas de código (90-93) del algoritmo, invierte a la imagen para halla el fondo, la figura 3.21 grafica este resultado.

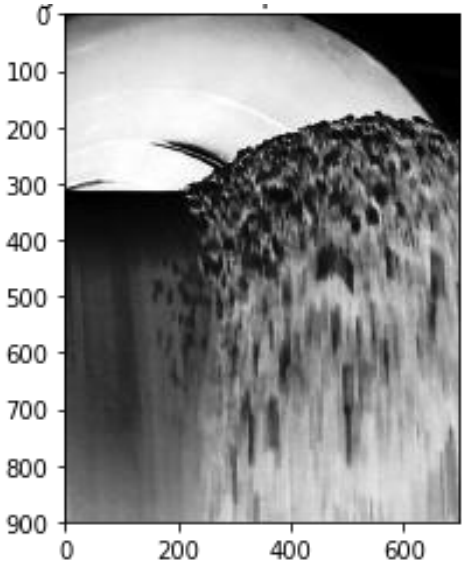
Figura 3.21: La imagen es invertida para hallar el fondo



Elaboración Propia

- Las líneas de código (94-98) del algoritmo, la imagen es invertida, ecualizada y suavizada el resultado la figura 3.22.

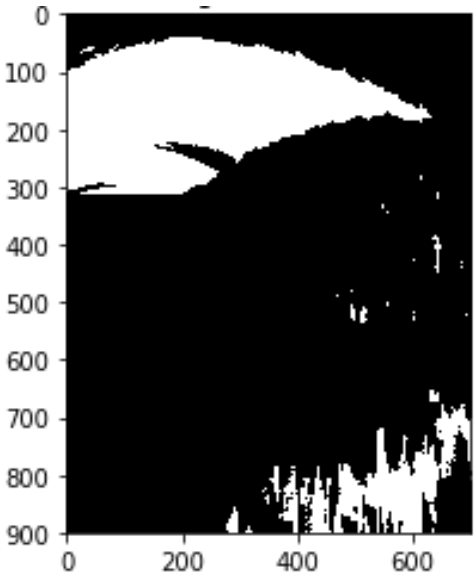
Figura 3.22: Imagen invertida y ecualizada



Elaboración Propia

- Las líneas de código (99-100) del algoritmo, aplica threshold a la imagen de fondo y se muestra en la figura 3.23.

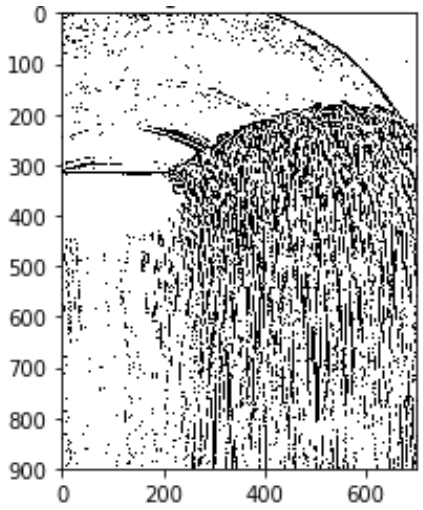
Figura 3.23: Imagen con Threshold



Elaboración Propia

- Las líneas de código (101-103) del algoritmo, aplica threshold adaptativo, que ayuda a identificar el fondo de la imagen, se muestra en la figura 3.24.

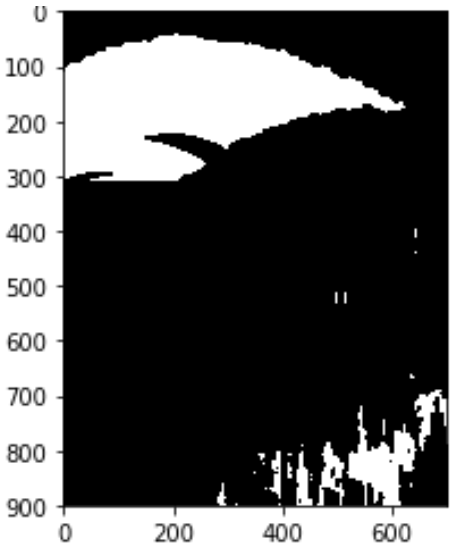
Figura 3.24: Imagen con Threshold adaptativo



Elaboración Propia

- Las líneas de código (104-106) del algoritmo, erosiona el fondo de la imagen con el objetivo de eliminar los ruidos presentes, como se muestra la figura 3.25.

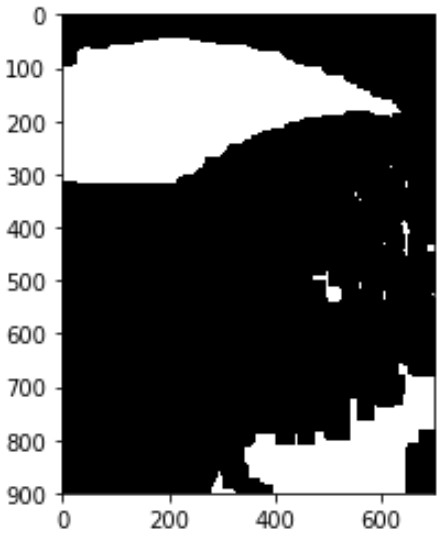
Figura 3.25: Imagen Erosionado



Elaboración Propia

- Las líneas de código (107-108) del algoritmo, apertura la imagen.
- Las líneas de código (109-113) del algoritmo, cierra las aperturas del fondo y se verifica en la figura 3.26.

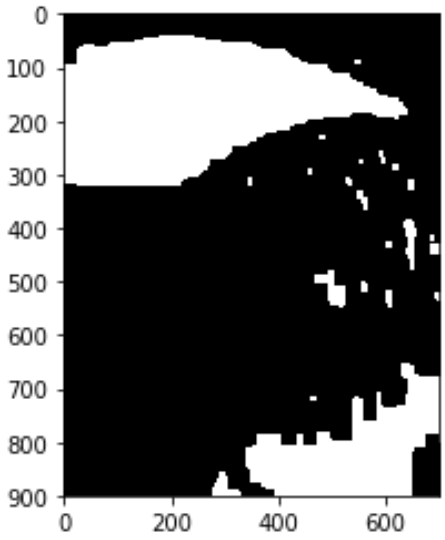
Figura 3.26: Cierre de las aperturas de la imagen



Elaboración Propia

- Las líneas de código (114-117) del algoritmo, dilata el fondo de la imagen, con el objetivo de desaparecer material no deseado y se muestra en la figura 3.27.

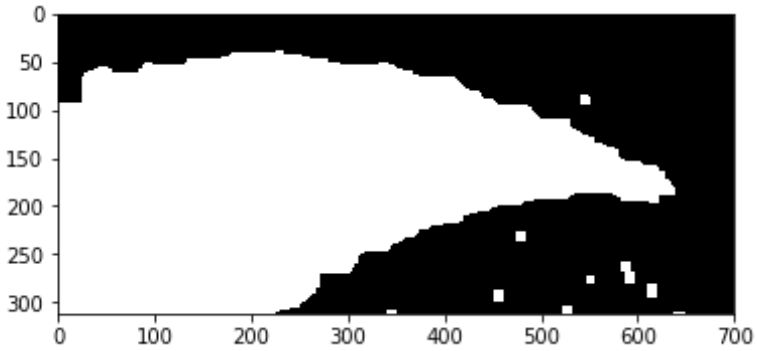
Figura 3.27: Dilatación de fondo de la imagen



Elaboración Propia

- Las líneas de código (118-119) del algoritmo, recorta la imagen invertida, como se muestra en la figura 3.28.

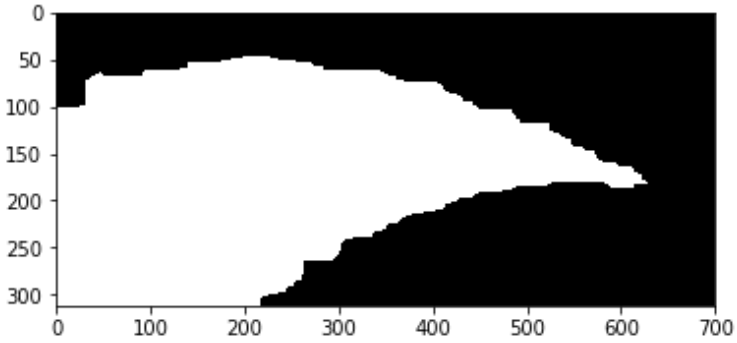
Figura 3.28: Recorte de la imagen Invertida



Elaboración Propia

- Las líneas de código (120-125) del algoritmo, erosiona el fondo de la imagen, figura 3.29.

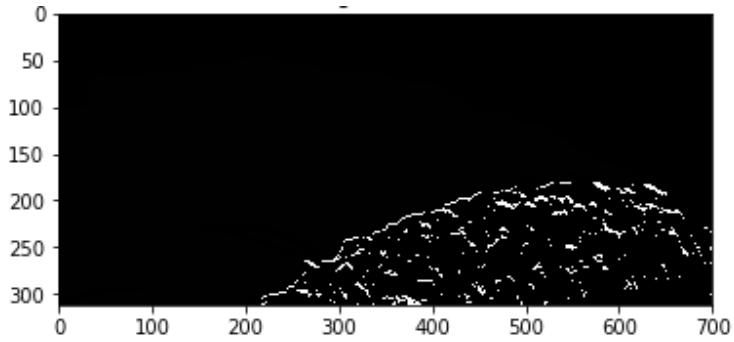
Figura 3.29: Erosión de la imagen de fondo



Elaboración Propia

- Las líneas de código (126-130) del algoritmo, resta imagen de fondo que no es requerido para el experimento, figura 3.30 muestra ese resultado.

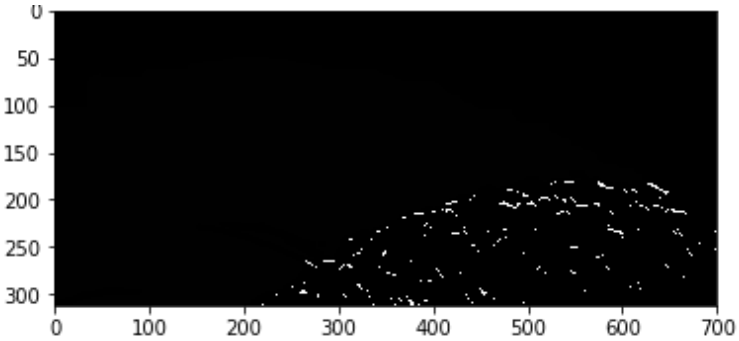
Figura 3.30: Recorte de la imagen no requerida



Elaboración Propia

- Las líneas de código (131-132) del algoritmo, erosiona la imagen restada, y se muestra en la figura 3.31.

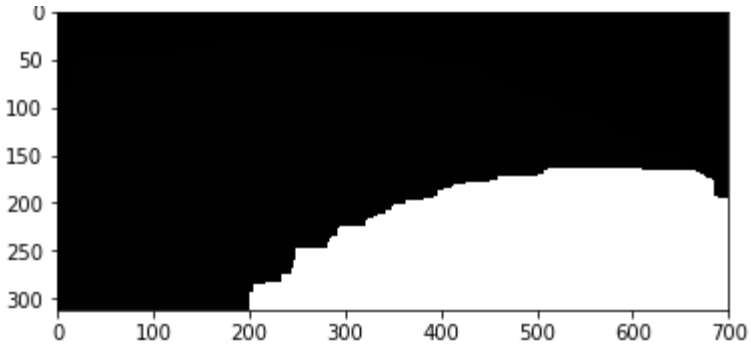
Figura 3.31: Erosión de la imagen recortada



Elaboración Propia

- Las líneas de código (133-135) del algoritmo, procede a dilatar la imagen erosionada, como se ve en la figura 3.32.

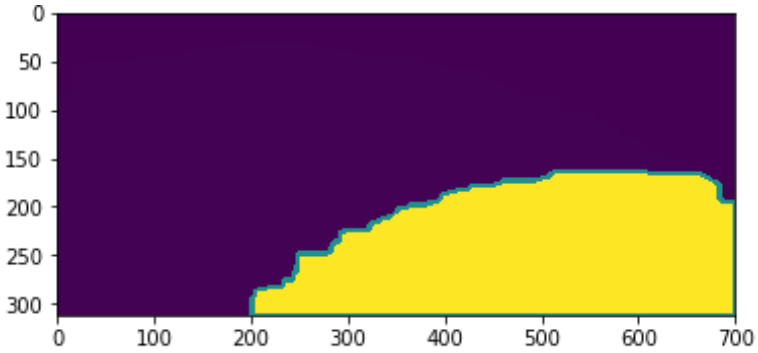
Figura 3.32: Dilatación de la imagen recortada



Elaboración Propia

- Las líneas de código (136-141) del algoritmo, convierte regiones para identificar la función de contorno, que es necesario para encontrar y buscar los contornos para luego graficar parte de la imagen que se requiere determinar además de resaltar los contornos, y se muestra en la figura 3.33.

Figura 3.33: Imagen contorneada



Elaboración Propia

- Las líneas de código (142-146) del algoritmo, define los parámetros para hallar el área de la imagen contorneada, la unidad de medida.
- Las líneas de código (147-148) del algoritmo, ajusta el porcentaje de la imagen contorneada en este caso es 3/2 entonces Área = área*3/2 pixeles cuadrados.

- La línea de código 149 es la determinación del factor de la equivalente de un pixel por centímetro como se detalla, multiplicando por un factor predefinido por el usuario en base a comparación de tamaño de imagen con medidas reales:

  10p = 1cm

  1p = 1/10 cm

  Factor = 1.0/10

- Las líneas de código (150-151) del algoritmo, imprime los vales del área en centímetros cuadrados y metros cuadrados. (1 cm2 = 0.0001 m2) y (**print** "Area es:",area*Factor*0.0001," metros cuadrados)
- Las líneas de código (152-153) del algoritmo, determina el volumen (volumen = area*Factor*0.0001) Tambien se define la profundidad igual a 1m, entonces resulta: "Volumen:", area*Factor*0.0001," metro cúbico" en cuanto a la densidad de los minerales se toma la densidad del plomo que es 11kg/m3 obteniendo un constante de densidad de 9:9.
- Línea de código 155 del algoritmo, muestra el valor del peso de mineral de la muestra en Kg.
- Línea de código 156 del algoritmo, indica el tiempo que termina en ejecutarse todo el algoritmo.
- Línea de código 157 del algoritmo, determina cuanto tiempo se demoró en obtener el algoritmo.
- Línea de código 158 del algoritmo, imprime el tiempo que demoro en obtener el algoritmo en segundos.

Como resultado para esta primera muestra se obtiene los valores que se muestran a continuación:

Área es: 57188.0 pixeles cuadrados

Ajuste por imagen incompleta: 85782.0 pixeles cuadrados

Área es: 8578.2 cm cuadrados

Área es: 0.85782 metros cuadrados

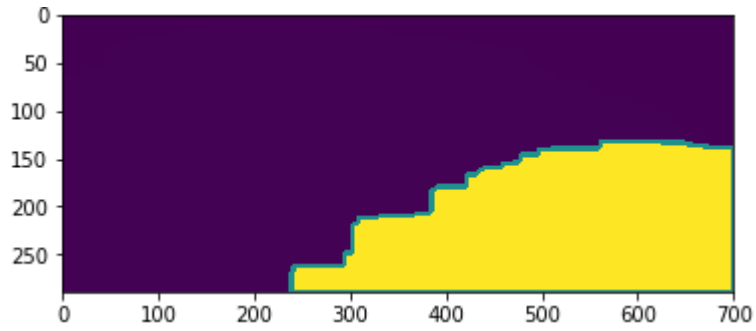
Volumen es: 0.85782 metros cúbico

Peso es: 8.492418 Kg

El tiempo de ejecución fue: 9.88880395889 segundos.

Como resultado de estas líneas de código del algoritmo, se ha obtenido el volumen requerido como se muestra en la Figura 3.34 donde se aprecia la imagen procesada de esta manera quedando solamente el mineral, de la misma manera se puede determinar el área y volumen de las demás muestras.

Figura 3.34: Mineral encontrado



Elaboración Propia

3.6.3. IMAGEN PARA LA SEGUNDA MUESTRA

Para el desarrollo del algoritmo se toma la siguiente muestra de nombre MINERAL17 de la faja transportadora, posterior a esto se realiza el procesamiento de imagen con OpenCV de este modo filtrar y separar las estructuras metálicas de la faja para luego determinar el área y volumen de los minerales en la faja, mediante este proceso se desarrolló el

algoritmo, para este caso se tiene la primera imagen a procesar como se muestra en la figura 3.35.

Figura 3.35: Imagen de la Segunda Muestra

Elaboración Propia

3.6.3.1 Desarrollo del código de la segunda Muestra

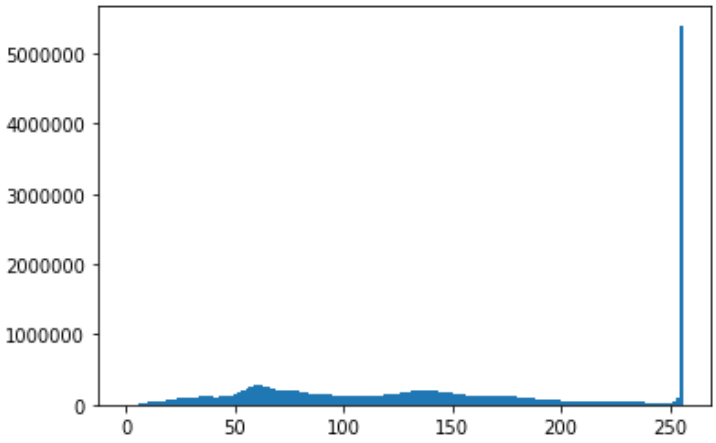
Una vez obtenido la segunda muestra con la cámara, el siguiente paso es desarrollar el algoritmo de procesamiento, para ello se ejecutó las líneas de código que se encuentran en el apartado de ANEXOS.

3.6.3.2 Interpretación de las líneas de código desarrollado en la primera muestra, que se encuentran en el ANEXO 2

- Las líneas de código (1-5) del algoritmo, importa las librerías Numpy, OpenCv, Matplotlib y tiempo respectivamente además de imprimir la versión de OpenCV.
- La línea de código 6 del algoritmo, es para empezar a contar el tiempo que demorara todo el proceso.
- Las líneas de código (7-14) del algoritmo, definen las funciones para mostrar la imagen.

- La línea de código 15 del algoritmo, muestra la ubicación de la imagen que se está procesando.
- Las líneas de código (16-19) del algoritmo, muestran la información de la imagen.
- Las líneas de código (20-21) del algoritmo, grafica el histograma de la imagen, como se muestra en la figura 3.36

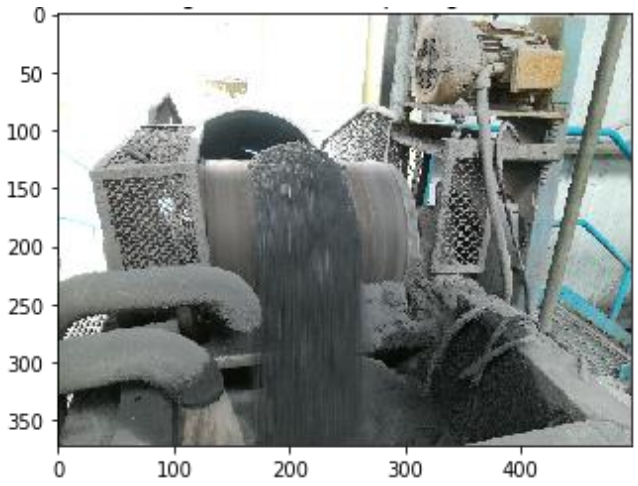
Figura 3.36: Histograma de la Imagen



Elaboración Propia

- La línea de código 22 del algoritmo, reduce la imagen 8 veces de la original y este se muestra en la figura 3.37

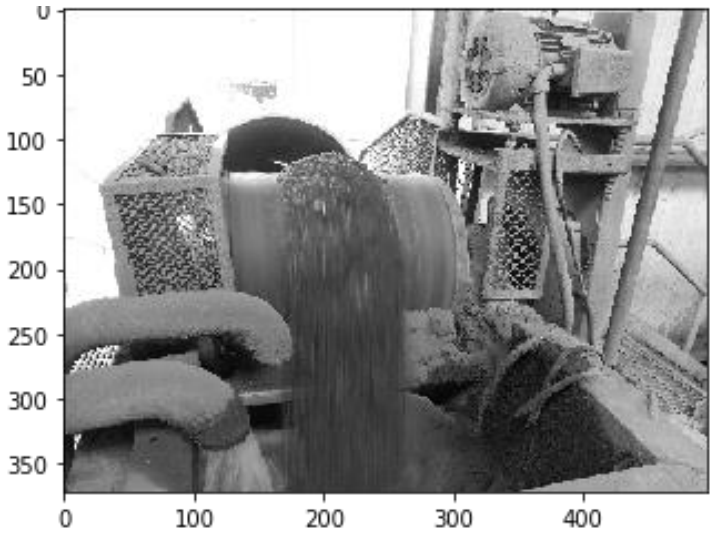
Figura 3.37: Imagen reducida 8 veces del original



Elaboración Propia

- Las líneas de código (23-27) del algoritmo, convierten la imagen de RGB a escala grises a la vez reduce 8 veces la imagen original para un mejor desempeño del programa, esta imagen reducida es la figura 3.38

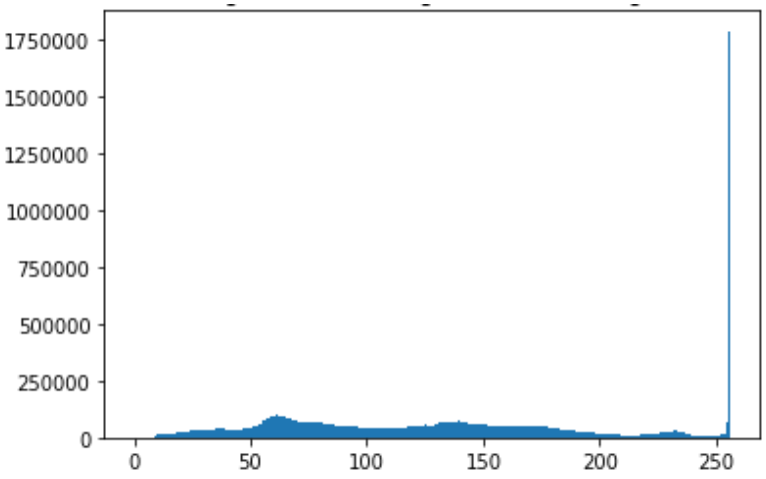
Figura 3.38: Imagen convertida a escala grises y reducida 8 veces



Elaboración Propia

- Las líneas de código (28-29) del algoritmo, grafica el histograma de la imagen en escala de grises, se visualiza en la figura 3.39.

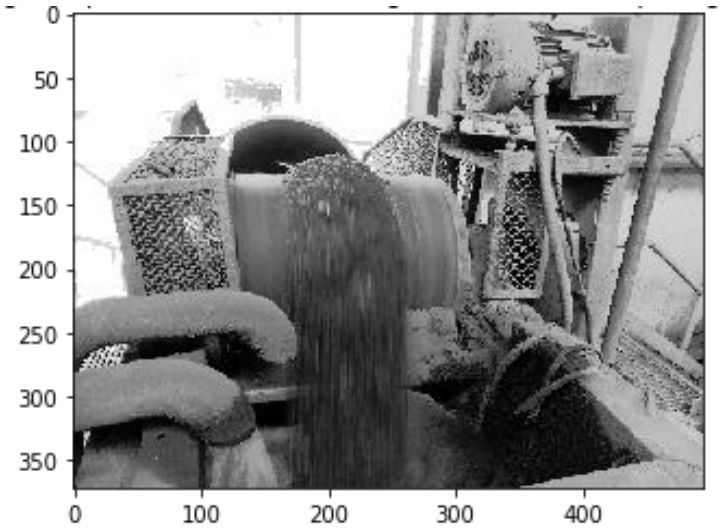
Figura 3.39: Histograma de la imagen a escala grises



Elaboración Propia

- Las líneas de código (30-33) del algoritmo, ecualiza la imagen en escala grises y reduce 8 veces su tamaño original donde la figura 3.40 demuestra esa función.

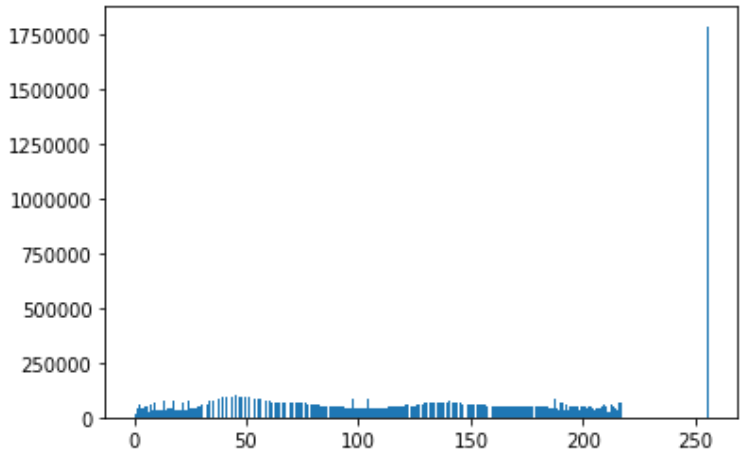
Figura 3.40: Imagen ecualizada y reducida 8 veces



Elaboración Propia

- Las líneas de código (34-35) del algoritmo, grafica el histograma de la imagen en grises ecualizada, figura 3.41

Figura 3.41: Histograma de la imagen ecualizada en escala grises

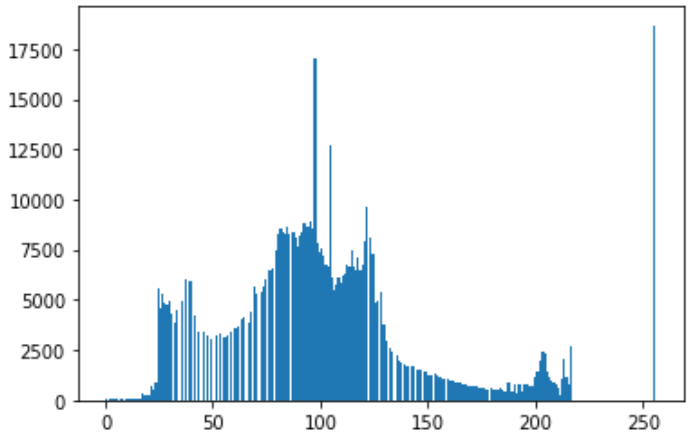


Elaboración Propia

- Las líneas de código (36-39) del algoritmo, obtienen la información de la nueva imagen y tamaño correspondiente.

- Las líneas de código (40-42) del algoritmo, grafica el histograma de la imagen recortada, que representa la figura 3.42.

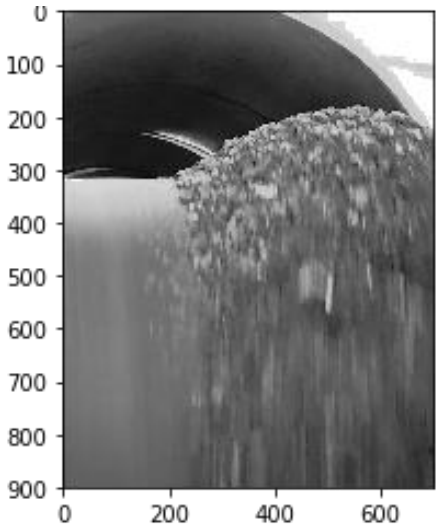
Figura 3.42: Histograma de la imagen recortada



Elaboración Propia

- Las líneas de código (43-45) del algoritmo, muestra la imagen nuevamente recortada, como se aprecia en la figura 3.43.

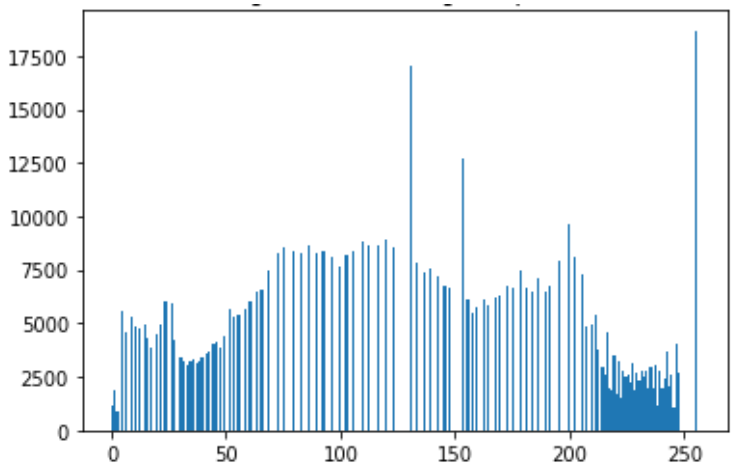
Figura 3.43: Imagen Recortada



Elaboración Propia

- La línea de código (46-48) del algoritmo, grafica el histograma de esta imagen recortada, que representa una mejor distribución de los parámetros de la imagen, como muestra la figura 3.44

Figura 3.44: Histograma de la imagen recortada



Elaboración Propia

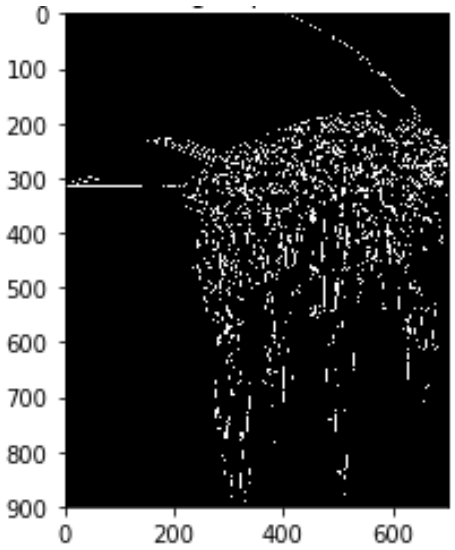
- Las líneas de código (49-50) del algoritmo, muestra la imagen ecualizada, y se muestra en la figura 3.45.

Figura 3.45: Imagen recortada ecualizada

Elaboración Propia

- Las líneas de código (51-55) del algoritmo, encuentra los bordes con el método Canny, esta grafica se muestra en la figura 3.46.

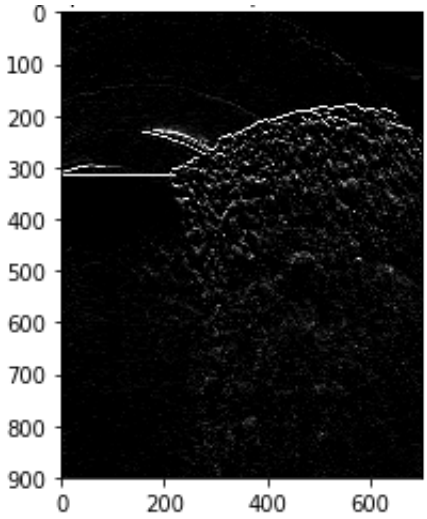
Figura 3.46: Detección de bordes método Canny



Elaboración Propia

- Las líneas de código (56-60) del algoritmo, aplica el segundo método de detección de bordes denominado Sobel y el resultado se muestra en la figura 3.47

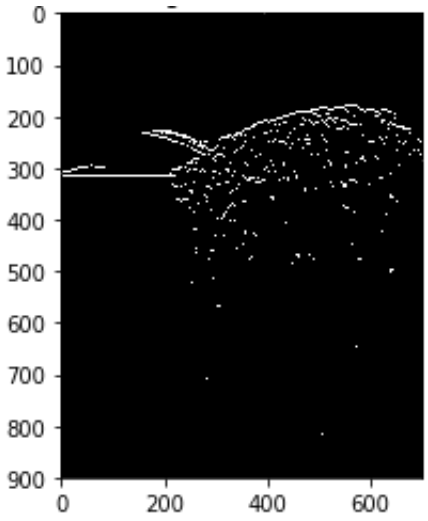
Figura 3.47: Detección de bordes Sobel



Elaboración Propia

- Las líneas de código (61-66) del algoritmo, umbraliza la detección de bordes Sobel, este resultado se visualiza en la figura 3.48.

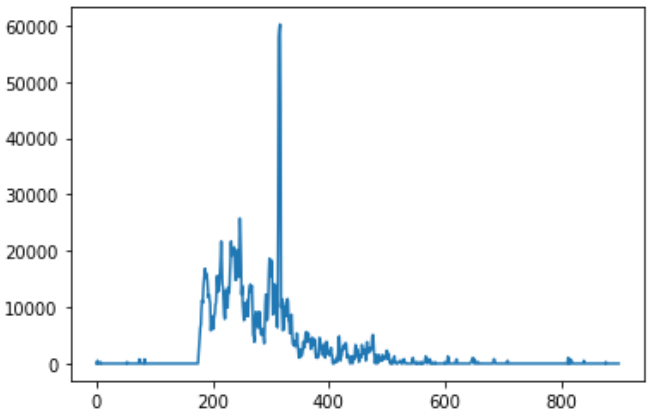
Figura 3.48: Umbralización de la detección de borde Sobel



Elaboración Propia

- Las líneas de código (67-70) del algoritmo, realiza la sumatoria de todos los puntos blancos en el eje horizontal, como muestra la figura 3.49

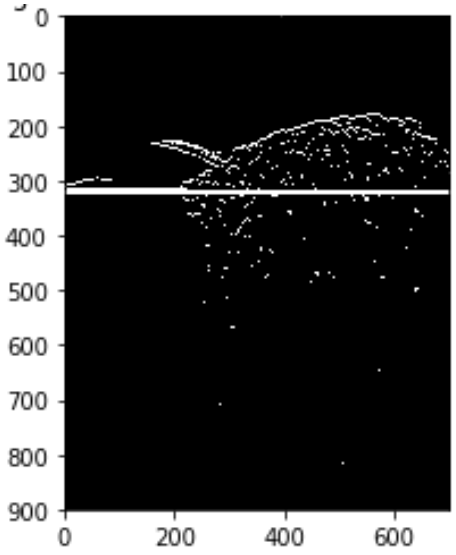
Figura 3.49: Sumatoria en el eje horizontal



Elaboración Propia

- Las líneas de código (71-74) del algoritmo, muestra la máxima position que es: 289 con 70125 pixeles.
- Las líneas de código (75-76) del algoritmo, encuentra la faja que es necesaria para calcular el mineral de la faja, esta detección se muestra en la figura 3.50.

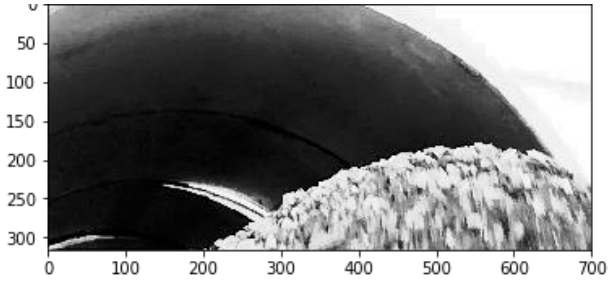
Figura 3.50: Imagen con identificación de la línea de faja



Elaboración Propia

- Las líneas de código (77-83) del algoritmo, recorta la parte inferior de la imagen de esta forma descarta todo lo que está debajo de la línea detectada y se aprecia en la figura 3.51 también muestra la información de la imagen recortada, y obtiene la información de la imagen Tamaño: Alto: 289 Ancho: 700.

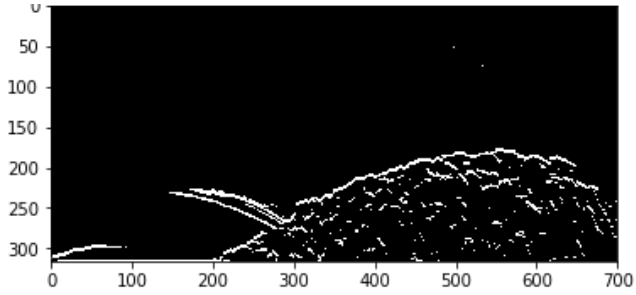
Figura 3.51: Imagen recortada parte inferior



Elaboración Propia

- Las líneas de código (84-88) del algoritmo, recupera la parte de la imagen buscada donde la figura 3.52 ayuda a verificar.

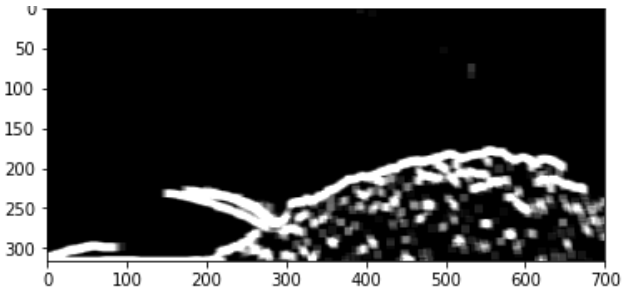
Figura 3.52: Imagen recuperada



Elaboración Propia

- La línea de código 89 del algoritmo, muestra la imagen recuperado, suavizado y su representación gráfica es la figura 3.53

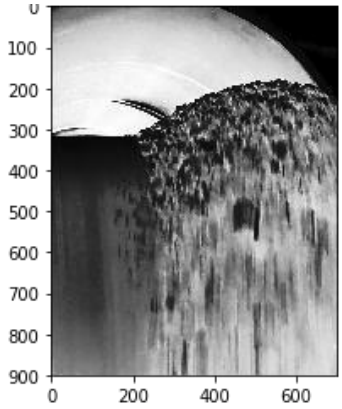
Figura 3.53: Imagen recuperada y suavizada



Elaboración Propia

- Las líneas de código (90-93) del algoritmo, invierte a la imagen para halla el fondo, la figura 3.54 grafica este resultado.

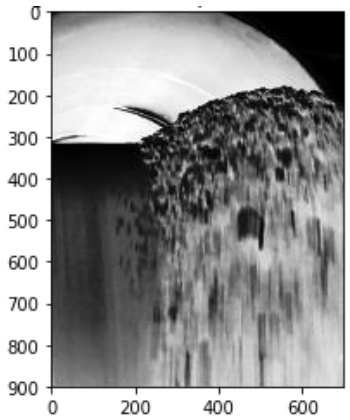
Figura 3.54: Imagen invertida para encontrar fondo



Elaboración Propia

- Las líneas de código (94-98) del algoritmo, la imagen invertida, ecualizada y suavizada el resultado la figura 3.55

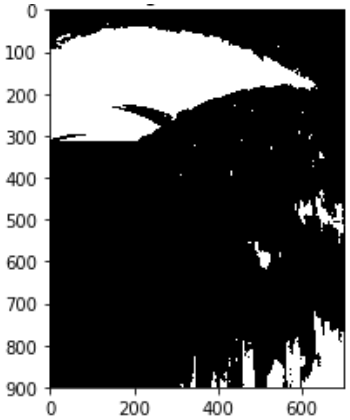
Figura 3.55: Imagen ecualizada suavizada



Elaboración Propia

- Las líneas de código (99-100) del algoritmo, aplica threshold a la imagen de fondo y se muestra en la figura 3.56

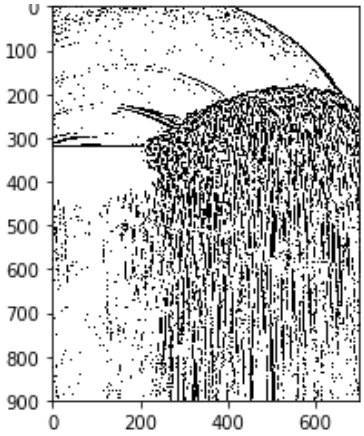
Figura 3.56: Imagen threshold



Elaboración Propia

- Las líneas de código (101-103) del algoritmo, aplica threshold adaptativo, que ayuda a identificar el fondo de la imagen, se muestra en la figura 3.57

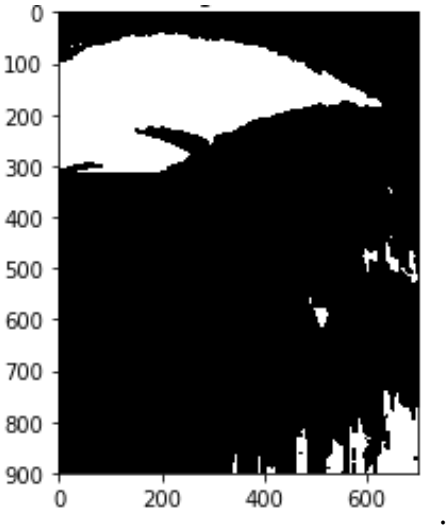
Figura 3.57: Imagen threshold adaptativo



Elaboración Propia

- Las líneas de código (104-106) del algoritmo, erosiona el fondo de la imagen con el objetivo de eliminar los ruidos presentes, como se muestra la figura 3.58

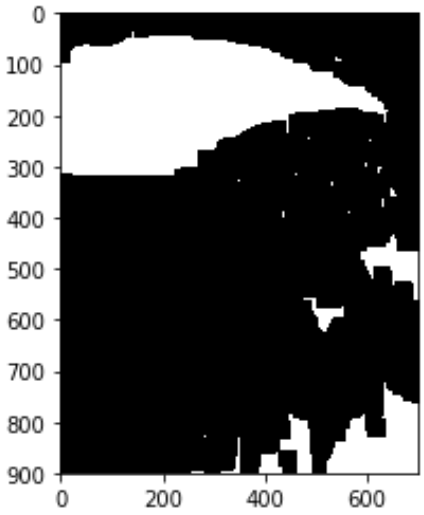
Figura 3.58: Imagen erosionada



Elaboración Propia

- Las líneas de código (107-108) del algoritmo, apertura la imagen.
- Las líneas de código (109-113) del algoritmo, cierra las aperturas del fondo y se verifica en la figura 3.59

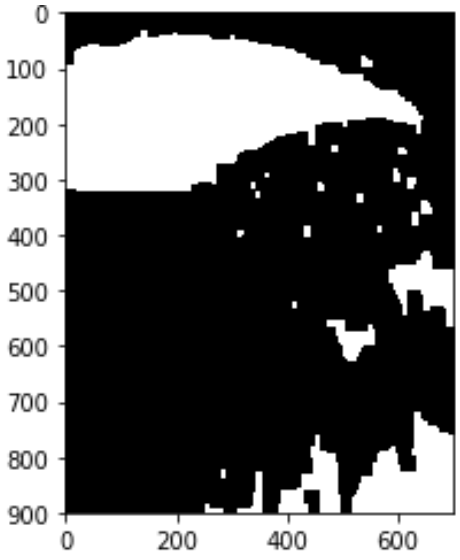
Figura 3.59: Cierre de las aperturas de la imagen



Elaboración Propia

- Las líneas de código (114-117) del algoritmo, dilata el fondo de la imagen, con el objetivo de desaparecer material no deseado y se muestra en la figura 3.60

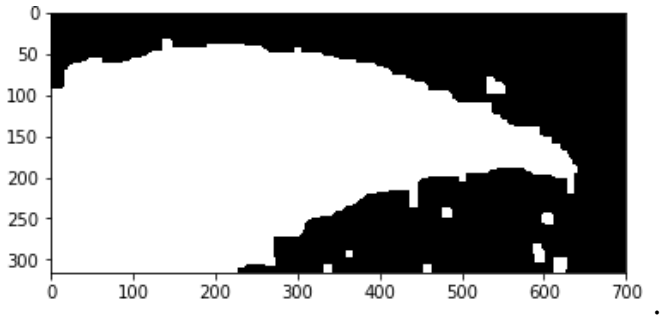
Figura 3.60: Dilatación del fondo de la imagen



Elaboración Propia

- Las líneas de código (118-119) del algoritmo, recorta la imagen invertida, se muestra en la figura 3.61

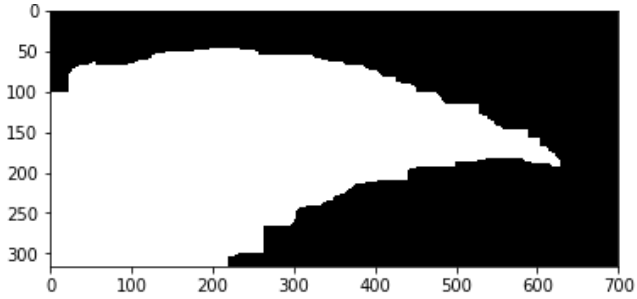
Figura 3.61: Recortado de la imagen invertida



Elaboración Propia

- Las líneas de código (120-125) del algoritmo, erosiona el fondo de la imagen, figura 3.62

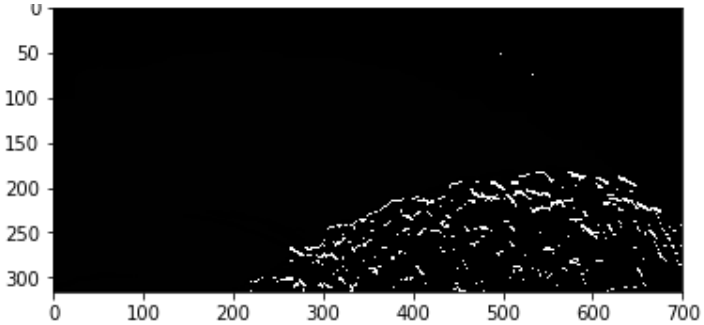
Figura 3.62: Fondo de la Imagen erosionada



Elaboración Propia

- Las líneas de código (126-130) del algoritmo, resta imagen de fondo que no es requerido para el experimento, figura 3.63 muestra ese resultado.

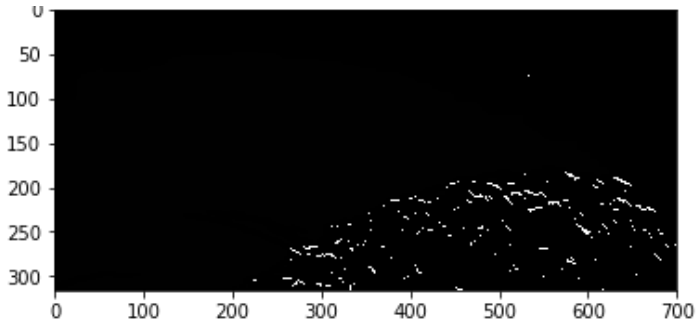
Figura 3.63: Imagen restada



Elaboración Propia

- Las líneas de código (131-132) del algoritmo, erosiona la imagen restada, y se muestra en la figura 3.64

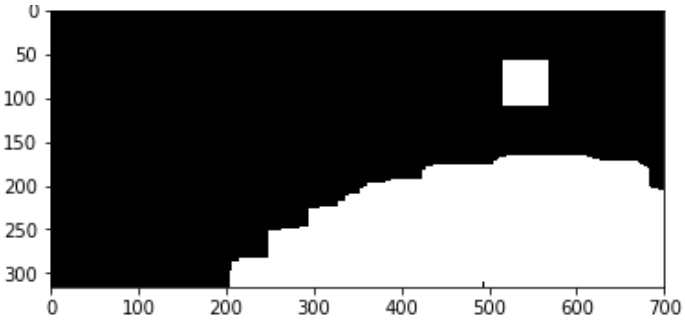
Figura 3.64: Erosión de la imagen



Elaboración Propia

- Las líneas de código (133-135) del algoritmo, procede a dilatar la imagen erosionada, como se ve en la figura 3.65

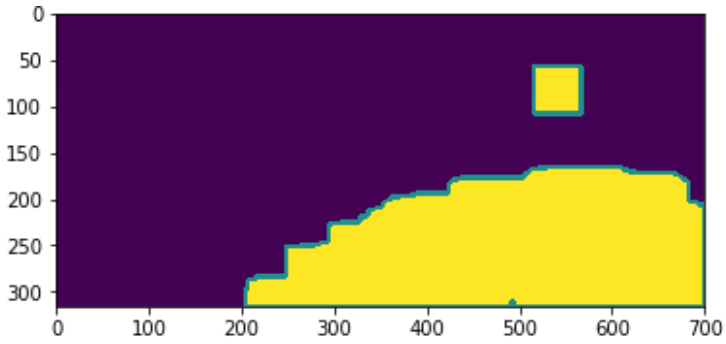
Figura 3.65: Imagen dilatada



Elaboración Propia

- Las líneas de código (136-141) del algoritmo, convierte regiones para identificar la función de contorno, que es necesario para encontrar y buscar los contornos para luego graficar parte de la imagen que se requiere determinar además de resaltar los contornos, y se muestra en la figura 3.66

Figura 3.66: Imagen contornos resaltados



Elaboración Propia

- Las líneas de código (142-146) del algoritmo, define los parámetros para hallar el área de la imagen contorneada, la unidad de medida.
- Las líneas de código (147-148) del algoritmo, ajusta el porcentaje de la imagen contorneada en este caso es 3/2 entonces Área = área*3/2 pixeles cuadrados.
- La línea de código 149 del algoritmo, es la determinación del factor de la equivalente de un pixel por centímetro como se detalla, multiplicando por un factor predefinido por el usuario en base a comparación de tamaño de imagen con medidas reales:

  10p = 1cm

  1p = 1/10 cm

  Factor = 1.0/10
- Las líneas de código (150-151) del algoritmo, imprime los vales del área en centímetros cuadrados y metros cuadrados. (1 cm2 = 0.0001 m2) y (print "Área es:",área*Factor*0.0001," metros cuadrados)
- Las líneas de código (152-153) del algoritmo, determina el volumen (volumen = área*Factor*0.0001) También se define la profundidad igual a 1m, entonces resulta: "Volumen:", área*Factor*0.0001," metro cúbico" en cuanto a la densidad

de los minerales se toma la densidad del plomo que es 11kg/m3 obteniendo un constante de densidad de 9:9.

- Línea de código 155 del algoritmo, muestra el valor del peso de mineral de la muestra en Kg.
- Línea de código 156 del algoritmo, indica el tiempo que termina en ejecutarse todo el algoritmo.
- Línea de código 157 del algoritmo, determina cuanto tiempo se demoró en obtener el algoritmo.
- Línea de código 158 del algoritmo, devuelve el tiempo que demoro en obtener el algoritmo en segundos.

Como resultado para esta primera muestra se obtiene los valores que se muestran a continuación

Área es: 57223.5 pixeles cuadrados

Ajuste por imagen incompleta: 85835.25 pixeles cuadrados

Área es: 8583.525 cm cuadrados

Área es: 0.8583525 metros cuadrados

Volumen es: 0.8583525 metro cúbico

Peso es: 8.49768975 Kg

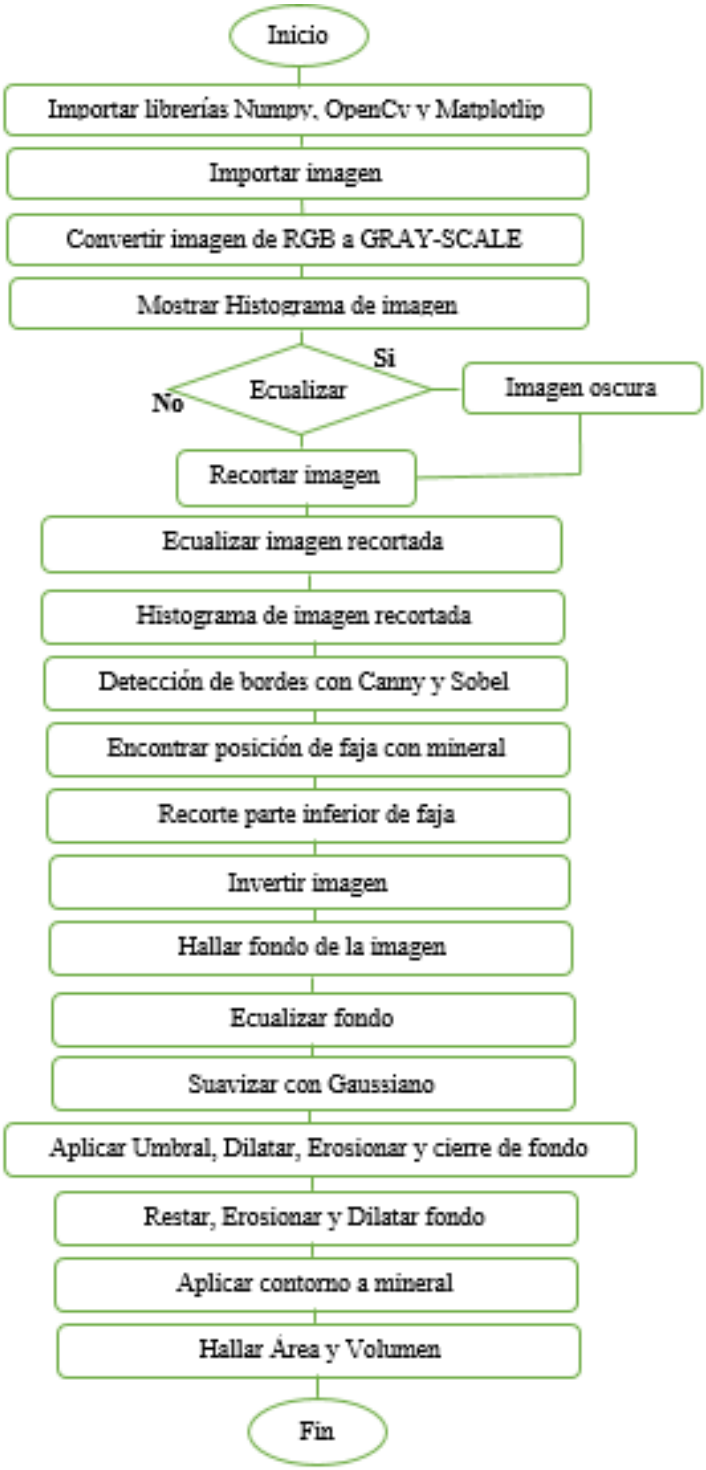
El tiempo de ejecución fue: 9.92130112648 segundos.

El procedimiento es el mismo para todas las muestras la diferencia está en el tiempo que demora el procesamiento de cada imagen eso dependerá de la iluminación y la potencia del equipo que ejecuta el programa. Los algoritmos de las demás muestras se encuentran en el apartado de ANEXOS

3.7. DIAGRAMA DE BLOQUES DEL ALGORITMO

La secuencia del algoritmo se muestra en la figura 3.67 el diagrama de bloques del procesamiento que realiza para determinar la área y volumen del mineral en la faja.

Figura 3.67: Diagrama de Bloques del algoritmo



Elaboración Propia

3.7.1. DESCRIPCION DEL DIAGRAMA DE BLOQUES

El algoritmo comienza con la importación de las librerías NUMPY, OPENCV, y Matplotlib que son las herramientas fundamentales en visión por computadora, seguidamente se importa la imagen a procesar, continuación se convierte la imagen RGB en escala grises de este modo se verifica con el histograma la distribución de los colores a escala grises, será necesario ecualizar si la distribución de colores no esté adecuado y si no, se procede con el siguiente paso el cual es reducir los pixeles de la imagen el cual ayuda a procesar mucho mejor la imagen, seguidamente se verifica de nuevo la distribución de colores de la imagen recortada, el siguiente paso es la detección de los bordes con los métodos canny y Sobel, una vez detectado los bordes también se detecta la posición de la faja, una vez detectado se recorta la parte inferior de la faja, el siguiente paso es la inversión de la imagen con esto se logra detectar el fondo de la imagen que contiene las estructuras metálicas que se necesitan filtrar, de ese modo se repite los mismos pasos de ecualización, suavización hasta lograr eliminar el fondo una vez aplicado el umbral , dilatación, erosión y cierre de fondo , el siguiente paso es restar el fondo de la imagen no invertida, otra vez aplica erosión y dilatación, quedando solamente el mineral filtrado seguidamente se contornea los minerales para posteriormente determinara el área y volumen del mineral en la faja , que es lo que se busca con este algoritmo, terminando el proceso en segundos, que es la característica que ofrece OPENCV.

CAPITULO IV

RESULTADOS Y DISCUSIÓN

4.1. RESULTADOS OBTENIDOS EN EL EXPERIMENTO

Los resultados de medición en este experimento fueron próximos a la medición manual ya que se empleó procedimientos avanzados de procesamiento de imágenes y se manipulo la imagen hasta lograr separar por completo los minerales del resto de la infraestructura sólida.

4.1.1. MUESTRA 1

En la figura 4.1 se muestra la imagen de nombre MINERAL16, la faja transportadora con los minerales segundos antes de la parada y medición manual los cuales hasta ese instante están en plena producción de ese modo se compararon las mediciones manuales y con el algoritmo.

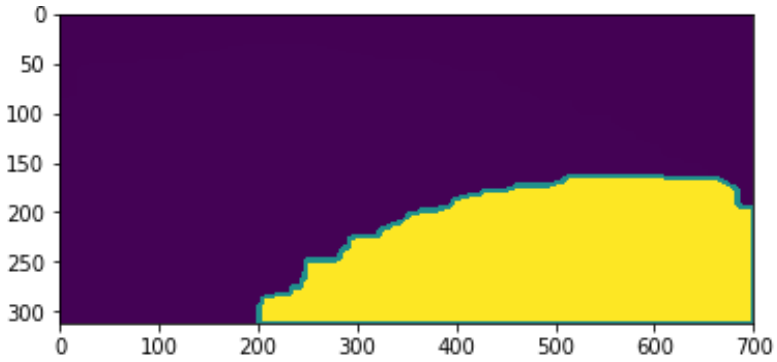
Figura 4.1: MINERAL16 tomada de la faja para el procesamiento

Elaboración Propia

Medición obtenida con el algoritmo de procesamiento de imágenes donde el área resultante es de 57188 Pixeles cuadrados que convirtiendo esta unidad a centímetros

cuadrados resulta igual a 8578.2 centímetros cuadrados , de la misma manera esa unidad se convirtió en metros cuadrados resultando igual a 0.85782 metros cuadrados, y el volumen resulta de hallarse el área por la profundidad asignada y en este caso es de un metro , resultando el volumen de esta manera 0.85782 metros cúbicos, con estos valores se obtuvo el peso aplicando la densidad de los minerales en este caso se toma el valor de 9.9 kilogramos por metro cubico, resultando de este modo el peso igual a Volumen por densidad, obteniéndose el peso de 8.492418 kg como se muestra en la figura 4.2

Figura4.2: Resultado obtenido mediante algoritmo



Elaboración Propia

Área es: 57188.0 pixeles cuadrados

Ajuste por imagen incompleta: 85782.0 pixeles cuadrados

Área es: 8578.2 cm cuadrados

Área es: 0.85782 metros cuadrados

Volumen es: 0.85782 metro cúbico

Peso es: 8.492418 Kg

El tiempo de ejecución con graficas fue: 9.88880395889 segundos.

Tiempo de ejecución sin graficas fue de 0.524757146835 segundos.

4.1.2. MUESTRA 2

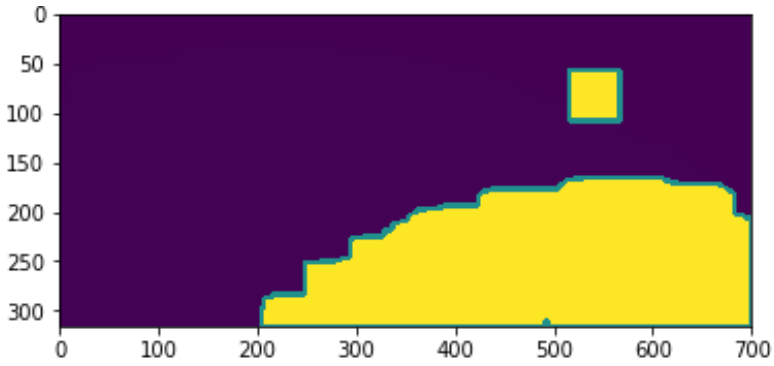
De la misma manera en la Figura 4.3 la imagen de nombre MINERAL17, fue tomada antes de la segunda parada de este modo se compararon la medición manual con la medición mediante algoritmo.

Figura 4.3: MINERAL17 para la segunda muestra

Elaboración Propia

Medición obtenida con el algoritmo de procesamiento de imágenes figura 4.4 donde se ejecutó la equivalencia de pixeles con el centímetro de esta forma que 10pixeles equivalen a un centímetro en consecuencia un pixel equivale a la décima parte de un centímetro, calculando el área se obtuvo un valor de 57223.5 pixeles cuadrados, el área convirtiendo en centímetros cuadrados es de 8583.525 centímetros cuadrados ,que a su vez convirtiendo a metros cuadrados es 0.8583525 metros cuadrados en consecuencia el volumen resultante es de 0.8583525 metros cúbicos, y para medir el peso se ejecutó la multiplicación del volumen con la densidad del mineral resultando el peso en 8.49768975 kilo gramos

Figura 4.4: Resultado obtenido mediante algoritmo



Elaboración Propia

Área es: 57223.5 pixeles cuadrados

Ajuste por imagen incompleta: 85835.25 pixeles cuadrados

Área es: 8583.525 cm cuadrados

Área es: 0.8583525 metros cuadrados

Volumen es: 0.8583525 metro cúbico

Peso es: 8.49768975 Kg

El tiempo de ejecución con graficas fue: 9.92130112648 segundos

Tiempo de ejecución sin graficas fue de 0.48321890831 segundos

4.1.3. MUESTRA 3

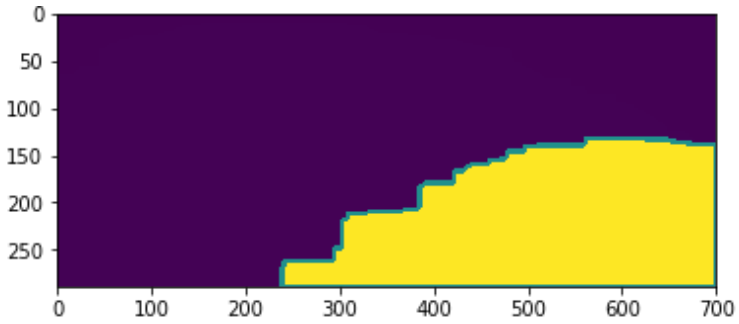
Siguiendo la misma secuencia se obtiene la figura 4.5 minutos antes de la tercera parada para nuestro caso es la imagen de nombre MINERAL18.

Figura 4.5: MINERAL18 muestra a ejecutar con el algoritmo

Elaboración Propia

La figura 4.6 muestra la Medición Obtenida con el algoritmo siguiendo los mismos procedimientos el área en pixeles 52701.5 pixeles cuadrados, convirtiendo en centímetros cuadrados resulta 7905.225 centímetros cuadrados que a su vez convertido en metros es 0.7905225 metros cuadrados, seguidamente el volumen se obtuvo con la multiplicación del área por la profundidad asignada que es de un metro, resultando el volumen de la muestra tres en 0.7905225 metros cúbicos, finalmente hallando el peso del mineral que resulta de la multiplicación del área por densidad y como resultado se tiene el peso de 7.82617275 kilogramos.

Figura 4.6: Resultado obtenido por el algoritmo



Elaboración Propia

Área es: 52701.5 pixeles cuadrados

Ajuste por imagen incompleta: 79052.25 pixeles cuadrados

Área es: 7905.225 cm cuadrados

Área es: 0.7905225 metros cuadrados

Volumen es: 0.7905225 metro cúbico

Peso es: 7.82617275 Kg

El tiempo de ejecución con graficas fue: 8.81595897675 segundos.

Tiempo de ejecución sin graficas fue de 0.530152082443 segundos

4.1.4. MUESTRA 4

Al igual que las anteriores muestras la figura 4.7 de nombre MINERAL19, se obtuvo segundos antes de la cuarta parada y comparando los resultados manuales y mediante algoritmo.

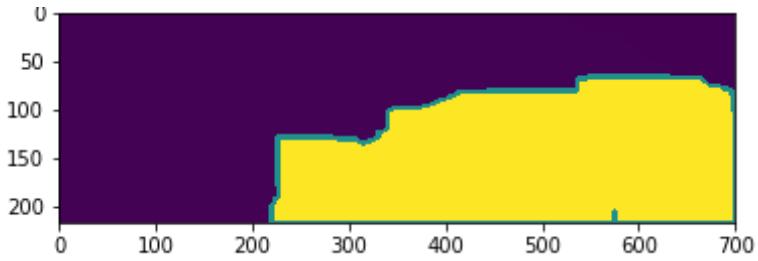
Figura 4.7: MINERAL19 muestra para el procesamiento

Elaboración Propia

La figura 4.8 muestra el resultado de la medición obtenida por el algoritmo. Siguiendo el mismo algoritmo primero se obtuvo el área en pixeles que es de 59803.0 pixeles cuadrados, convirtiendo a centímetros resulta un área de 8970.45 centímetros cuadrados,

a su vez convirtiendo a metros se tiene un área de 0.8970045 metros cuadrados de esta forma el siguiente paso es hallar el volumen que es igual a área por profundidad asignado, resultando como 0.897045 metros cúbicos, y finalmente el peso de la muestra es volumen por densidad del mineral resultando 8.8807455 kilogramos

Figura 4.8: Resultado obtenido por el algoritmo



Elaboración Propia

Área es: 59803.0 pixeles cuadrados

Ajuste por imagen incompleta: 89704.5 pixeles cuadrados

Área es: 8970.45 cm cuadrados

Área es: 0.897045 metros cuadrados

Volumen es: 0.897045 metro cúbico

Peso es: 8.8807455 Kg

El tiempo de ejecución con graficas fue: 9.67265081406 segundos

Tiempo de ejecución sin graficas fue de 0.615705013275 segundos

Después de estas mediciones realizadas se hicieron las comparativas en las mediciones y los resultados se muestran en la tabla: 4.1.

Tabla 4.1: Cuadro de comparación de resultados obtenidos

IMAGEN PROCESADA	VOLUMEN (ALGORITMO) (m3)	PESAJE (ALGORITMO) (kg)	%	TIEMPO DE EJECUCION		PESAJE MANUAL (kg)	%	TIEMPO DE PESAJE EN MINUTOS	MARGEN DE ERROR (PM-PA) (kg)	%
				ALGORITMO SIN GRAFICOS EN SEGUNDOS	ALGORITMO CON GRAFICOS EN					
MINERAL 16	0.85782	8.49242	99.79	0.58	9.88	8.51	100.00	19	0.0176	0.21
MINERAL 17	0.85835	8.49769	99.50	0.49	9.92	8.54	100.00	19	0.0423	0.50
MINERAL 18	0.79052	7.82617	99.57	0.54	8.81	7.86	100.00	18	0.0338	0.43
MINERAL 19	0.89705	8.88075	99.78	0.61	9.67	8.90	100.00	20	0.0193	0.22
				2.22	38.28			76		

Elaboración Propia

Interpretación de la tabla 4.1, de los resultados obtenidos del experimento, la muestra 1 de la imagen procesada MINERAL16, el volumen (algoritmo) en metros cúbicos es 0.85782 el mismo que se realiza la conversión a peso (algoritmo) en 8.492418 kilogramos que representa al 99.79%, el tiempo de verificación del algoritmo sin gráficos es de 0.58 segundos y teniendo en cuenta el algoritmo con gráficos se ejecutó en 9.88 segundos, y con el pesaje realizado manualmente se obtiene 8.51 kilogramos que representa al 100% obtenido en 19 minutos, la relación entre peso (algoritmo) y peso manual el margen de error es de 0.017582 kilogramos que representa a 0.21 %. En la muestra 2 de la imagen procesada MINERAL17, el volumen (algoritmo) en metros cúbicos es 0.8583525 el mismo que se realiza la conversión a peso (algoritmo) es de 8.49768975 kilogramos que representa al 99.50%, el tiempo de ejecución del algoritmo sin gráficos es de 0.49 segundos y teniendo en cuenta el algoritmo con gráficos se ejecutó en 9.92 segundos, y el pesaje obtenido manualmente es de 8.54 kilogramos que representa al 100% realizado en 19 minutos, la relación entre peso (algoritmo) y peso manual el margen de error es de 0.04231025 kilogramos que representa a 0.50 %. y en la siguiente muestra 3 de la imagen

procesada MINERAL18, el volumen (algoritmo) en metros cúbicos es 0.7905225 el mismo que se realiza la conversión a peso (algoritmo) en 7.82617275 kilogramos y representa al 99.57%, el tiempo de verificación del algoritmo sin gráficos es 0.54 segundos y teniendo en cuenta el algoritmo con gráficos se ejecutó en 8.81 segundos, y con el pesaje realizado manualmente se obtiene 7.86 kilogramos que representa al 100% realizado en 18 minutos, la relación entre peso (algoritmo) y peso manual el margen de error es de 0.03382725 kilogramos que representa a 0.43 %. y con respecto a la muestra 4 de la imagen procesada MINERAL19, el volumen (algoritmo) en metros cúbicos es 0.897045 el mismo que se realiza la conversión a peso (algoritmo) es 8.8807455 kilogramos que representa al 99.78% , el tiempo de verificación del algoritmo sin gráficos en 0.61 segundos y teniendo en cuenta el algoritmo con gráficos se realizó en 9.67 segundos, y con el pesaje realizado manualmente se obtiene 8.90 kilogramos que representa al 100% determinado en 20 minutos, la relación entre peso (algoritmo) y peso manual el margen de error es de 0.0192545 kilogramos que representa a 0.22 % respectivamente.

4.2. DISCUSIÓN

- Para el diseño del algoritmo el autor Abarca Cusimayta en su tesis Diseño de un modelo algorítmico basado en visión computacional para la detección y clasificación de retinopatía diabética en imágenes retinográficas digitales, concluye que Se establecieron técnicas de preprocesamiento para poder mejorar la imagen y la información obtenida para enviarla al clasificador. Es así que, la detección de las regiones de interés se logró mediante la detección del fondo en la imagen de entrada para poder crear una máscara que permita extraer solo la sección de la retina (foreground). En este caso hay relación de concordancia con la investigación experimental realizada ya que los algoritmos diseñados en este

proyecto mediante técnicas de procesamiento determinan el peso del mineral mediante la conversión de pixeles en centímetros luego en metros de esta manera se obtuvo el área y el volumen que al final encontramos el peso del mineral que se trasportaba por la faja.

- En el procesamiento de imágenes usando OPENCV el autor Viera Maza en su tesis Procesamiento de imágenes usando OpenCV aplicado en Raspberry Pi para la clasificación del cacao concluye que en el Perú no existen muchos sistemas de visión artificial para la evaluación de calidad y clasificación de productos agrícolas. Generalmente, la inspección se realiza de manera manual, siendo esto una limitante para el desarrollo de la agroindustria. Efectivamente en el sector minero también es necesario por eso en la entidad donde se realiza el experimento necesita de personal para poder realizar las actividades manualmente y mientras con el diseño del algoritmo ahorramos tiempo y personal. es por eso que la visión artificial y el procesamiento de imágenes logran un gran avance en la producción.
- El desarrollo de los algoritmos según el autor Bereciartua Pérez en su tesis Desarrollo de algoritmos de procesamiento de imagen avanzado para interpretación de imágenes médicas concluye que El procesamiento y análisis de la imagen médica es un problema complejo que requiere de un conocimiento especializado. La imagen médica presenta muchos artefactos, ruidos y efectos de iluminación no homogénea en la imagen, causados por las tecnologías de adquisición, que hacen necesario el uso de técnicas avanzadas para la segmentación correcta y precisa de los tejidos presentes en las imágenes, por lo tanto, de acuerdo a nuestra investigación es de mucha importancia desarrollar algoritmos con OPenCV es la mejor herramienta para muchos campos de la industria.

CONCLUSIONES

Se ha realizado el diseño de un algoritmo de procesamiento de imágenes con la herramienta especializada Python OpenCV que determina el área, volumen y peso del mineral que se transporta en la faja, a través de imágenes obtenidas con la cámara Raspberry Pi, mediante sus herramientas de procesamiento, segmentación, detección de bordes, caso del algoritmo en el experimento tres demostró eficiencia en todo el proceso de obtención de resultados esperados, el tiempo total de verificación del peso (algoritmo) sin gráficos de la muestra 1 al 4 es de 2.22 segundos en un día y con gráficos es de 38.28 segundos y en referencia a pesaje manual de la muestra del 1 al 4 el tiempo acumulado de pesaje fue de 76 minutos en un día, reduciendo el tiempo de pesaje a menos de 1 minuto lo que conlleva a una eficiencia en la empresa, de esa forma se reduce la parada de producción en la unidad minera Mallay

Según los resultados obtenidos del experimento, el diseño de un algoritmo de procesamiento de imágenes en Python OpenCV, en efecto determina el área, volumen y peso del mineral que se transporta en la faja a través de imágenes obtenidas en la cámara Rasberry Pi, de esa manera se determinó el volumen (algoritmo) de la muestra 1 la imagen procesada MINERAL16 es 0.85782, de la muestra 2 la imagen procesada del MINERAL17 el volumen (algoritmo) determinado es 0.8583525, también considerando la muestra 3 del MINERAL18 el volumen (algoritmo) determinado es 0.7905225, y de la muestra 4 MINEARL19 el volumen (algoritmo) determinado es 0.897045 así sucesivamente se determina en cada muestra las variaciones en la faja transportadora de la unidad minera Mallay.

En el siguiente resultado, con la herramienta especializada de OpenCV se logró obtener el volumen (algoritmo) de 0.85782 m3 de la muestra 1 de la imagen procesada

MINERAL16 que resalta más dentro de las muestras, el mismo que fue convertido en peso (algoritmo) igual a 8.492418 kg que representa al 99.79% con un margen de error del 0.017582 kg que representa al 0.21%, y el pesaje manual es de 8.51 kg que representa el 100%, con respecto de la muestra 2 la imagen procesada MINERAL17 el volumen (algoritmo) es 0.8583525 m3 convertido al peso (algoritmo) es 8.49768975 que representa al 99.50% con un margen de error de 0.04231025 que representa al 0.50% por lo tanto el margen de error de las muestras 1, 2, 3 y 4 son razonables porque no superan ni llegan al 1%, existe variación del peso manual de las muestras 1, 2, 3 y 4 por que la faja transporta poli metales (variedad de metales). Se logro obtener el volumen con un 99% de precisión lo cual es aceptable de acuerdo a los objetivos de esta investigación.

RECOMENDACIONES

PRIMERO: Diseñar algoritmos en OPENCV para que puedan ser aplicadas e implementadas en todas las industrias y de este modo aumentar la productividad a costos adecuados.

SEGUNDO: Aplicar el diseño del algoritmo en todos los campos, ya que al desarrollarlos de manera efectiva se reduce inmensamente los costos de personal, costos económicos entre otros.

TERCERO: Ejecutar añadiendo más cámaras para sincronizar, obtener las imágenes y encontrar resultados adecuados con margen de error mínima.

REFERENCIAS BIBLIOGRÁFICAS

Abarca Cusimayta, D. R. (2018). *Diseño de un modelo algorítmico basado en visión computacional para la detección y clasificación de retinopatía diabética en imágenes retinográficas digitales.* Lima.

Bereciartua Pérez, A. (2016). *Desarrollo de algoritmos de procesamiento de imagen avanzado para interpretación de imágenes médicas.* Bilbao.

Caballero Roldán, R., Martín Martín, E., & Riesco Rodríguez, A. (2018). *BIG DATA CON PYTHON Recolección, almacenamiento y proceso.* Madrid: Alfaomega.

Cavanzo Nisso, G. A., Pérez Pereira, M. R., & Villavisan Buitrago, F. (2017). Medición de eficiencia de algoritmos de visión artificial implementados en raspberry pi y ordenador personal mediante python. *Ingenium*, 105-119.

Contributors, E. (15 de July de 2019). *Algoritmo*. Obtenido de EcuRed: https://www.ecured.cu/index.php?title=Algoritmo&oldid=3458281

Delgado Gutiérrez, M. J., Herrera Guillén, D. F., Medina Barragán, L. M., & Corredor Gómez, J. P. (2017). Implementación de un sistema de procesamiento de imágenes integrado con Raspberry PI 2B para reconocimiento y recolección de fresas maduras. *Revista Politécnica*, 75-85.

Depetris, B., & Otros. (2018). *Diseño y aplicacion de estrategias para la enseñanza inicial de la programacion.*

Díaz García, J. (2018). Aplicaciones del lenguaje de programación Python en la docencia de ingeniería de estructuras. *Educación Editora*, 653-657.

Gollapudi, S. (2019). *Learn Computer Vision Using OpenCV.* Hyderabad, Telangana, India: APRESS.

Gutiérrez, J. S., Salazar, A. M., Elías, R. P., & Lavalle, M. M. (2016). *SÚPER RESOLUCIÓN Y MEJORA DEL ALGORITMO CANNY PARA LA DETECCIÓN DE BORDES EN IMÁGENES MÉDICAS.* exico: Tecnológico Nacional de México.

Hearn, B. C. (2014). *Computer Graphics with Open GL.* Harlow: Pearson.

Honores Tapia, J. A., & Camacho Osmolik, A. V. (2019). *Desarrollo de aplicación web para gestionar un campeonato de fútbol usando metodología snail y lenguaje de programación Python.* Ecuador.

Howe, K. D. (2014). *A PRACTICAL INTRODUCTION TO COMPUTER VISION WITH OPENCV.* Trinity College Dublin, Ireland: Wiley.

Hunter, J., Dale, D., Firing, E., & Droettboom, M. (26 de August de 2019). *The Matplotlib development team*. Obtenido de Matplotlib: https://matplotlib.org/index.html

Ingle, V. K., & Proakis, J. C. (2016). *Digital Signal Processing Using MATLAB: A Problem Solving Companion.* CAlifornia.

Joyanes Aguilar, L., Echeverri Arias, J. A., Orrego Villa, G. A., & Arenas Arenas, Ó. H. (2016). *Programación teoría y aplicaciones.* Medellin: U de Medellin.

Kaehler, A., & Bradski, G. (2016). *Learning OpenCV 3 Computer Vision in C++ with The OpenCV Library.* O'REILLY.

Kelly, S. (2019). *Pyhon, Pygam, and Raspberry Pi Game Dvelopment.* Niagafara Falls, Ontaro, Canada: APRESS.

Lamego Castro, J. A. (2017). *Desarrollo de un sistema inteligente de control de tráfico con software de código abierto en sistemas embebidos.* Jalisco.

Lorens Largo, F., Garcia Peñalgo, F. J., Molero Prieto, X., & Vendrel Vidal, E. (2017). La enseñanza de la informática, la programación y el pensamiento computacional en los estudios preuniversitarios. *Education in the Knowledge Society*, 7-11.

Molleapaza Mamani, J. W. (2018). *Estudio metodologico del tramo 1 de una faja transportadora para trasladar trozos de mineral de cobre con una capacidad de 944 TPH usando contrapeso de bloques de concreto evaluada con normas del Sistema Ingles.* Peru: UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA.

Olarte Gervacio, L. (29 de Enero de 2018). *Teoría de la programación*. Obtenido de Conogasi.org: http://conogasi.org/articulos/algoritmo/

Palomino Puma, Y., & Manrique Hernandez, J. A. (2015). *SISTEMA DE LLENADO AUTOMÁTICO DE BOTELLAS CON CONTROL DE NIVEL UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES.* LIMA.

Reyes, E. P. (2017). *Aplicacion para el Recuento y Caracterizacion.* Universidad de las Palmas de Gran Canaria.

Rios, J. R., Mora, M. N., & Sojos, E. L. (2016). Evaluación de los Frameworks en el Desarrollo de Aplicaciones Web con Python. Revista Latinoamericana de Ingenieria de Software. 201-207.

Rodríguez, Á. A., Figueredo, J. A., & Chica, J. A. (2018). *Sistema de reconocimiento de maduración del tomate, mediante el procesamiento de imágenes.* Medellin : Editorial IAI.

Taquía Gutiérrez, J. A. (2017). El procesamiento de imágenes y su potencial aplicación en empresas con estrategia digital. *INTERFASES*, 18 - 20.

Trejos Buritica, O. I. (2017). *PROGRAMACION IMPERATIVA CON LENGUAJE C.* Mexico: ECOE EDICIONES.

Troyano, J. A., Cruz, F., Gonzalez, M., Vallejos, C. J., & Toro, M. (2018). *Introducción a la Programación con Python, Computación.* Jenui.

Varela Jarquin, E. R., & Sequeira Lanuza, M. J. (2016). *Propuesta de diseño de un sistema de automatización de una cinta de trasportadora utilizada en la industria minera.* Managua.

Viera Maza, G. I. (2017). *Procesamiento de imágenes usando OpenCV aplicado en Raspberry Pi para la clasificación del cacao.* Piura.

ANEXOS

ANEXO 1

CODIGO FUENTE DE LA PRIMERA MUESTRA

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
1: from matplotlib import pyplot as plt
2: from time import time #importamos la funcion time para medir el tiempo de ejecucion
3: import cv2
4: import numpy as np
5: print "OpenCV version:",cv2.__version__
6: tiempo_inicial = time()
#=====================================================
======
# Definir funciones para mostrar imagenes
#=====================================================
======
7: def mostrar (img,mensaje):
8:   plt.imshow(img, cmap='gray', vmin=0, vmax=255)
9:   plt.title(mensaje)
10: plt.show()
11: def mostrar_color (img,mensaje):
12:  plt.imshow(img)
13:  plt.title(mensaje)
14:  plt.show()
#=====================================================
======
15: img = cv2.imread('./img/MINERAL16.jpg')
#=====================================================
======
16: print "Obteniendo informacion de la imagen:"
17: height, width = img.shape[:2]
18: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
19: hist = cv2.calcHist([img],[0],None,[256],[0,256])
#=====================================================
======
20: plt.title("Histograma de la imagen")
21: plt.hist(img.ravel(),256,[0,256]); plt.show()
22: mostrar (img[::8, ::8],'Imagen reducida x8 para graficar :')
23: print "Convertir imagen a escala de grises para equalizar el histograma"
24: gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
25: plt.imshow(gray[::8, ::8], cmap='gray', vmin=0, vmax=255)
26: plt.title('Imagen en escala de grises reducida x8 para graficar :')
27: plt.show()
```

```
28: plt.title("Histograma de la imagen en escala de grises")
29: plt.hist(gray.ravel(),256,[0,256]); plt.show()
30: img_gray_eq = cv2.equalizeHist(gray)
31: plt.imshow(img_gray_eq[::8, ::8], cmap='gray', vmin=0, vmax=255)
32: plt.title('Imagen equalizada en escala de grises reducida x8 para graficar :')
33: plt.show()
34: plt.title("Histograma de la imagen equalizada")
35: plt.hist(img_gray_eq.ravel(),256,[0,256]); plt.show()
#=================================================
==============
# recorte de la imagen
36: img_crop = img_gray_eq[700:1600,1100:1800]
37: print "Obteniendo informacion de la nueva imagen:"
38: height, width = img_crop.shape[:2]
39: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
40: plt.title("Histograma de la imagen recortada")
41: plt.hist(img_crop.ravel(),256,[0,256]); plt.show()
42: plt.imshow(img_crop, cmap='gray', vmin=0, vmax=255)
43: plt.title('Imagen recortada :')
44: plt.show()
45: img_crop_eq = cv2.equalizeHist(img_crop)
46: plt.title("Histograma de la imagen equalizada")
47: plt.hist(img_crop_eq.ravel(),256,[0,256]); plt.show()
48: plt.imshow(img_crop_eq, cmap='gray', vmin=0, vmax=255)
49: plt.title('Imagen recortada equalizada :')
50: plt.show()
#=================================================
=====
# Eliminar ruido
51: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
#=================================================
=====
# hallar bordes
# 1 opcion canny
52: bordes_canny = cv2.Canny(img_crop_eq_blur,50,250)
53: plt.imshow(bordes_canny, cmap='gray', vmin=0, vmax=255)
54: plt.title('Bordes de imagen por funcion Canny :')
55: plt.show()
#=================================================
=====
# 2 opcion sobel
56: SCALE = 1
57: DELTA = 0
58: DDEPTH = cv2.CV_16S  ## to avoid overflow
59:  sobelx  =  cv2.Sobel(img_crop_eq_blur,  DDEPTH,  0,  1,  ksize=3,  scale=SCALE,
delta=DELTA)
60: plt.imshow(sobelx, cmap='gray', vmin=0, vmax=255)
61: plt.title("Sobel, para hallar la posicion de la faja :")
62: plt.show()
63: ret,thresh = cv2.threshold(sobelx,100,255,0)
```

```
64: plt.imshow(thresh, cmap='gray', vmin=0, vmax=255)
65: plt.title('Imagen Threshold :')
66: plt.show()
67: sum = np.array
68: sum = np.sum(thresh, axis=1)
69: plt.title("Sumatoria en el eje horizontal")
70: plt.plot(sum)
71: plt.show()

72: max_position = sum.argmax()
73: min_position = sum.argmin()
74: print "maxima position es:",max_position, " con ", sum[max_position], " pixels"
75: thresh[max_position+1:max_position+10:] = 255
76: mostrar (thresh,'Imagen con identificacion de linea de faja :')
#Nuevo recorte para tomar solamente la imagen encima de la faja
77: img = img_crop_eq[0:max_position,:]
78: mostrar (img, "Imagen recortada 2")
79: print "Obteniendo informacion de la imagen:"
80: height, width = img.shape[:2]
81: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
82: hist = cv2.calcHist([img],[0],None,[256],[0,256])
83: mostrar (img,'Imagen descartando la parte inferior de la faja')
84: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
85: img = thresh[0:max_position,:]
86: kernel = np.ones((10,10),np.float32)/25
#Filtra la imagen utilizando el kernel anterior
87: dst = cv2.filter2D(img,-1,kernel)
88: mostrar (img,'Imagen recuperada')
89: mostrar (dst,'Imagen recuperada filtro medio suavizado')

#==================================================
=
#++++++++++++++++++++++++++++++++++++++++++++++++++
+
# nueva imagen para hallar el fondo
#++++++++++++++++++++++++++++++++++++++++++++++++++
+
90: img_fondo = cv2.bitwise_not(img_crop_eq)
91: plt.imshow(img_fondo, cmap='gray', vmin=0, vmax=255)
92: plt.title('Imagen invertida para hallar fondo')
93: plt.show()
94: img_fondo_eq = cv2.equalizeHist(img_fondo)
95: mostrar (img_fondo_eq,'Imagen invertida para hallar fondo equalizada')
96: img_fondo_blur_tmp = cv2.GaussianBlur(img_fondo_eq,(3,3),0)
97: img_fondo_blur = cv2.GaussianBlur(img_fondo_blur_tmp,(5,5),0)
98: mostrar (img_fondo_blur,'imagen invertida equalizada suavizada')
99: ret,thresh1 = cv2.threshold(img_fondo_blur,200,255,cv2.THRESH_BINARY)
100: mostrar (thresh1,'imagen threshold')
101: img2 = cv2.convertScaleAbs(img_fondo_blur)
```

```
102: thresh2 = cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
103: mostrar (thresh2,'imagen threshold 2')
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
104: kernel = np.ones((3,3),np.uint8)
105: erode = cv2.erode(thresh1,kernel,iterations = 2)
106: mostrar (erode,'imagen erode')

107: kernel = np.ones((20,20),np.uint8)
108: opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
109: kernel = np.ones((30,30),np.uint8)
110: closing = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)
111: closing2 = cv2.morphologyEx(closing, cv2.MORPH_CLOSE, kernel)
112: closing3 = cv2.morphologyEx(closing2, cv2.MORPH_CLOSE, kernel)
113: closing4 = cv2.morphologyEx(closing3, cv2.MORPH_CLOSE, kernel)
114: kernel = np.ones((3,3),np.uint8)
115: dilated = cv2.dilate(closing4, kernel, iterations = 4)
116: mostrar (closing4,'fondo cerrado')
117: mostrar (dilated,'fondo dilatado')
118: img_inv2 = dilated[0:max_position,:]
119: mostrar (img_inv2,'recortando ')
120: kernel = np.ones((3,3),np.uint8)
121: eroded = cv2.erode(img_inv2,kernel,iterations = 7)
122: mostrar (eroded,'erosionando fondo ')
123: print eroded.dtype
124: img2 = cv2.convertScaleAbs(img)
125: print img2.dtype
#=================================================================
#Erosion/dilatacion
126: kernel = np.ones((2,2),np.uint8)
127: ret,thresha = cv2.threshold(img2,200,255,cv2.THRESH_BINARY)
128: ret,threshb = cv2.threshold(eroded,200,255,cv2.THRESH_BINARY)
129: imga = thresha - threshb
130: mostrar (imga,"Imagen restada" )
131: erosionada = cv2.erode(imga,kernel,iterations = 1)
132: mostrar (erosionada,'Imagen erosionada')
#kernel = np.ones((2,2),np.uint8)
#abierta = cv2.morphologyEx(erosionada, cv2.MORPH_OPEN, kernel)
#mostrar (abierta,'Imagen abierta')
133: kernel = np.ones((4,4),np.uint8)
134: dilatada = cv2.dilate(erosionada,kernel,iterations = 17)
135: mostrar (dilatada,'Imagen dilatada')
# convert for region identification function contours
136: dilatada2 = cv2.convertScaleAbs(dilatada)
#necesario para encontrar contornos
137: ret,dilatada3 = cv2.threshold(dilatada2,200,255,cv2.THRESH_BINARY)
```

```
138: contours, hierarchy = cv2.findContours(dilatada3,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
139: print len(contours)
140: cv2.drawContours(dilatada2,contours,-1,(128,0,0),3)
141: mostrar_color (dilatada2,'Resaltando contornos')
142: area = 0
143: for cnt in contours:
144:   if cv2.contourArea(cnt) > area:
145:     area = cv2.contourArea(cnt)
146: print "Area es:",area," pixeles cuadrados"
147: print "Ajuste por imagen imcompleta:",area*3/2," pixeles cuadrados"
148: area = area * 3/2
# conversion a cm2, multiplicando por un factor predefinido por el usuario en
# base a comparacion de tamano de imagen con medidas reales
#10p = 1cm
#1p = 1/10 cm
149: Factor = 1.0/10
150: print "Area es:",area*Factor," cm cuadrados"
#1 cm2 = 0.0001 m2
151: print "Area es:",area*Factor*0.0001," metros cuadrados"
#profundidad es 1m
152: volumen = area*Factor*0.0001
153: print "Volumen es:",area*Factor*0.0001," metro cúbico"
# densidad del plomo 11kg/m3
154: DENSIDAD = 9.9
155: print "Peso es:",volumen*DENSIDAD," Kg"
156: tiempo_final = time()
157: tiempo_ejecucion = tiempo_final - tiempo_inicial
158: print 'El tiempo de ejecucion fue:',tiempo_ejecucion #En segundos
```

ANEXO 2

CODIGO FUENTE DE LA SEGUNDA MUESTRA

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
1: from matplotlib import pyplot as plt
2: from time import time #importamos la funcion time para medir el tiempo de ejecucion
3: import cv2
4: import numpy as np
5: print "OpenCV version:",cv2.__version__
6: tiempo_inicial = time()
#=================================================
======
# Definir funciones para mostrar imagenes
```

```
#=====================================================
======
7: def mostrar (img,mensaje):
8:   plt.imshow(img, cmap='gray', vmin=0, vmax=255)
9:   plt.title(mensaje)
10: plt.show()
11: def mostrar_color (img,mensaje):
12:  plt.imshow(img)
13:  plt.title(mensaje)
14:  plt.show()
#=====================================================
======
15: img = cv2.imread('./img/MINERAL17.jpg')
#=====================================================
======
16: print "Obteniendo informacion de la imagen:"
17: height, width = img.shape[:2]
18: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
19: hist = cv2.calcHist([img],[0],None,[256],[0,256])
#=====================================================
======
20: plt.title("Histograma de la imagen")
21: plt.hist(img.ravel(),256,[0,256]); plt.show()
22: mostrar (img[::8, ::8],'Imagen reducida x8 para graficar :')
23: print "Convertir imagen a escala de grises para equalizar el histograma"
24: gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
25: plt.imshow(gray[::8, ::8], cmap='gray', vmin=0, vmax=255)
26: plt.title('Imagen en escala de grises reducida x8 para graficar :')
27: plt.show()
28: plt.title("Histograma de la imagen en escala de grises")
29: plt.hist(gray.ravel(),256,[0,256]); plt.show()
30: img_gray_eq = cv2.equalizeHist(gray)
31: plt.imshow(img_gray_eq[::8, ::8], cmap='gray', vmin=0, vmax=255)
32: plt.title('Imagen equalizada en escala de grises reducida x8 para graficar :')
33: plt.show()
34: plt.title("Histograma de la imagen equalizada")
35: plt.hist(img_gray_eq.ravel(),256,[0,256]); plt.show()
#=====================================================
==============
# recorte de la imagen
36: img_crop = img_gray_eq[700:1600,1100:1800]
37: print "Obteniendo informacion de la nueva imagen:"
38: height, width = img_crop.shape[:2]
39: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
40: plt.title("Histograma de la imagen recortada")
41: plt.hist(img_crop.ravel(),256,[0,256]); plt.show()
42: plt.imshow(img_crop, cmap='gray', vmin=0, vmax=255)
43: plt.title('Imagen recortada :')
44: plt.show()
45: img_crop_eq = cv2.equalizeHist(img_crop)
```

```
46: plt.title("Histograma de la imagen equalizada")
47: plt.hist(img_crop_eq.ravel(),256,[0,256]); plt.show()
48: plt.imshow(img_crop_eq, cmap='gray', vmin=0, vmax=255)
49: plt.title('Imagen recortada equalizada :')
50: plt.show()
#=================================================================
=====
# Eliminar ruido
51: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
#=================================================================
=====
# hallar bordes
# 1 opcion canny
52: bordes_canny = cv2.Canny(img_crop_eq_blur,50,250)
53: plt.imshow(bordes_canny, cmap='gray', vmin=0, vmax=255)
54: plt.title('Bordes de imagen por funcion Canny :')
55: plt.show()
#=================================================================
=====
# 2 opcion sobel
56: SCALE = 1
57: DELTA = 0
58: DDEPTH = cv2.CV_16S  ## to avoid overflow
59: sobelx = cv2.Sobel(img_crop_eq_blur, DDEPTH, 0, 1, ksize=3, scale=SCALE,
delta=DELTA)
60: plt.imshow(sobelx, cmap='gray', vmin=0, vmax=255)
61: plt.title("Sobel, para hallar la posicion de la faja :")
62: plt.show()
63: ret,thresh = cv2.threshold(sobelx,100,255,0)
64: plt.imshow(thresh, cmap='gray', vmin=0, vmax=255)
65: plt.title('Imagen Threshold :')
66: plt.show()
67: sum = np.array
68: sum = np.sum(thresh, axis=1)
69: plt.title("Sumatoria en el eje horizontal")
70: plt.plot(sum)
71: plt.show()

72: max_position = sum.argmax()
73: min_position = sum.argmin()
74: print "maxima position es:",max_position, " con ", sum[max_position], " pixels"
75: thresh[max_position+1:max_position+10:] = 255
76: mostrar (thresh,'Imagen con identificacion de linea de faja :')
#Nuevo recorte para tomar solamente la imagen encima de la faja
77: img = img_crop_eq[0:max_position,:]
78: mostrar (img, "Imagen recortada 2")
79: print "Obteniendo informacion de la imagen:"
80: height, width = img.shape[:2]
81: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
82: hist = cv2.calcHist([img],[0],None,[256],[0,256])
```

```
83: mostrar (img,'Imagen descartando la parte inferior de la faja')
84: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
85: img = thresh[0:max_position,:]
86: kernel = np.ones((10,10),np.float32)/25
#Filtra la imagen utilizando el kernel anterior
87: dst = cv2.filter2D(img,-1,kernel)
88: mostrar (img,'Imagen recuperada')
89: mostrar (dst,'Imagen recuperada filtro medio suavizado')

#=====================================================
=
#+++++++++++++++++++++++++++++++++++++++++++++++++++++
+
# nueva imagen para hallar el fondo
#+++++++++++++++++++++++++++++++++++++++++++++++++++++
+
90: img_fondo = cv2.bitwise_not(img_crop_eq)
91: plt.imshow(img_fondo, cmap='gray', vmin=0, vmax=255)
92: plt.title('Imagen invertida para hallar fondo')
93: plt.show()
94: img_fondo_eq = cv2.equalizeHist(img_fondo)
95: mostrar (img_fondo_eq,'Imagen invertida para hallar fondo equalizada')
96: img_fondo_blur_tmp = cv2.GaussianBlur(img_fondo_eq,(3,3),0)
97: img_fondo_blur = cv2.GaussianBlur(img_fondo_blur_tmp,(5,5),0)
98: mostrar (img_fondo_blur,'imagen invertida equalizada suavizada')
99: ret,thresh1 = cv2.threshold(img_fondo_blur,200,255,cv2.THRESH_BINARY)
100: mostrar (thresh1,'imagen threshold')
101: img2 = cv2.convertScaleAbs(img_fondo_blur)
102:                                    thresh2                                    =
cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THR
ESH_BINARY,11,2)
103: mostrar (thresh2,'imagen threshold 2')
#+++++++++++++++++++++++++++++++++++++++++++++++++++++
+
104: kernel = np.ones((3,3),np.uint8)
105: erode = cv2.erode(thresh1,kernel,iterations = 2)
106: mostrar (erode,'imagen erode')

107: kernel = np.ones((20,20),np.uint8)
108: opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
109: kernel = np.ones((30,30),np.uint8)
110: closing = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)
111: closing2 = cv2.morphologyEx(closing, cv2.MORPH_CLOSE, kernel)
112: closing3 = cv2.morphologyEx(closing2, cv2.MORPH_CLOSE, kernel)
113: closing4 = cv2.morphologyEx(closing3, cv2.MORPH_CLOSE, kernel)
114: kernel = np.ones((3,3),np.uint8)
115: dilated = cv2.dilate(closing4, kernel, iterations = 4)
116: mostrar (closing4,'fondo cerrado')
117: mostrar (dilated,'fondo dilatado')
118: img_inv2 = dilated[0:max_position,:]
```

```
119: mostrar (img_inv2,'recortando ')
120: kernel = np.ones((3,3),np.uint8)
121: eroded = cv2.erode(img_inv2,kernel,iterations = 7)
122: mostrar (eroded,'erosionando fondo ')
123: print eroded.dtype
124: img2 = cv2.convertScaleAbs(img)
125: print img2.dtype
#=================================================
=====
#Erosion/dilatacion
126: kernel = np.ones((2,2),np.uint8)
127: ret,thresha = cv2.threshold(img2,200,255,cv2.THRESH_BINARY)
128: ret,threshb = cv2.threshold(eroded,200,255,cv2.THRESH_BINARY)
129: imga = thresha - threshb
130: mostrar (imga,"Imagen restada" )
131: erosionada = cv2.erode(imga,kernel,iterations = 1)
132: mostrar (erosionada,'Imagen erosionada')
#kernel = np.ones((2,2),np.uint8)
#abierta = cv2.morphologyEx(erosionada, cv2.MORPH_OPEN, kernel)
#mostrar (abierta,'Imagen abierta')
133: kernel = np.ones((4,4),np.uint8)
134: dilatada = cv2.dilate(erosionada,kernel,iterations = 17)
135: mostrar (dilatada,'Imagen dilatada')
# convert for region identification function contours
136: dilatada2 = cv2.convertScaleAbs(dilatada)
#necesario para encontrar contornos
137: ret,dilatada3 = cv2.threshold(dilatada2,200,255,cv2.THRESH_BINARY)
138:                contours,                hierarchy                =
cv2.findContours(dilatada3,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
139: print len(contours)
140: cv2.drawContours(dilatada2,contours,-1,(128,0,0),3)
141: mostrar_color (dilatada2,'Resaltando contornos')
142: area = 0
143: for cnt in contours:
144:   if cv2.contourArea(cnt) > area:
145:     area = cv2.contourArea(cnt)
146: print "Area es:",area," pixeles cuadrados"
147: print "Ajuste por imagen imcompleta:",area*3/2," pixeles cuadrados"
148: area = area * 3/2
# conversion a cm2, multiplicando por un factor predefinido por el usuario en
# base a comparacion de tamano de imagen con medidas reales
#10p = 1cm
#1p = 1/10 cm
149: Factor = 1.0/10
150: print "Area es:",area*Factor," cm cuadrados"
#1 cm2 = 0.0001 m2
151: print "Area es:",area*Factor*0.0001," metros cuadrados"
#profundidad es 1m
152: volumen = area*Factor*0.0001
153: print "Volumen es:",area*Factor*0.0001," metro cúbico"
```

```
# densidad del plomo 11kg/m3
154: DENSIDAD = 9.9
155: print "Peso es:",volumen*DENSIDAD," Kg"
156: tiempo_final = time()
157: tiempo_ejecucion = tiempo_final - tiempo_inicial
158: print 'El tiempo de ejecucion fue:',tiempo_ejecucion #En segundos
```

ANEXO 3

CODIGO FUENTE DE LA TERCERA MUESTRA

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
1: from matplotlib import pyplot as plt
2: from time import time #importamos la funcion time para medir el tiempo de ejecucion
3: import cv2
4: import numpy as np
5: print "OpenCV version:",cv2.__version__
6: tiempo_inicial = time()
#=====================================================
======
# Definir funciones para mostrar imagenes
#=====================================================
======
7: def mostrar (img,mensaje):
8:   plt.imshow(img, cmap='gray', vmin=0, vmax=255)
9:   plt.title(mensaje)
10: plt.show()
11: def mostrar_color (img,mensaje):
12:  plt.imshow(img)
13:  plt.title(mensaje)
14:  plt.show()
#=====================================================
======
15: img = cv2.imread('./img/MINERAL18.jpg')
#=====================================================
======
16: print "Obteniendo informacion de la imagen:"
17: height, width = img.shape[:2]
18: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
19: hist = cv2.calcHist([img],[0],None,[256],[0,256])
#=====================================================
======
20: plt.title("Histograma de la imagen")
21: plt.hist(img.ravel(),256,[0,256]); plt.show()
```

```
22: mostrar (img[::8, ::8],'Imagen reducida x8 para graficar :')
23: print "Convertir imagen a escala de grises para equalizar el histograma"
24: gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
25: plt.imshow(gray[::8, ::8], cmap='gray', vmin=0, vmax=255)
26: plt.title('Imagen en escala de grises reducida x8 para graficar :')
27: plt.show()
28: plt.title("Histograma de la imagen en escala de grises")
29: plt.hist(gray.ravel(),256,[0,256]); plt.show()
30: img_gray_eq = cv2.equalizeHist(gray)
31: plt.imshow(img_gray_eq[::8, ::8], cmap='gray', vmin=0, vmax=255)
32: plt.title('Imagen equalizada en escala de grises reducida x8 para graficar :')
33: plt.show()
34: plt.title("Histograma de la imagen equalizada")
35: plt.hist(img_gray_eq.ravel(),256,[0,256]); plt.show()
#================================================
==============
# recorte de la imagen
36: img_crop = img_gray_eq[700:1600,1100:1800]
37: print "Obteniendo informacion de la nueva imagen:"
38: height, width = img_crop.shape[:2]
39: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
40: plt.title("Histograma de la imagen recortada")
41: plt.hist(img_crop.ravel(),256,[0,256]); plt.show()
42: plt.imshow(img_crop, cmap='gray', vmin=0, vmax=255)
43: plt.title('Imagen recortada :')
44: plt.show()
45: img_crop_eq = cv2.equalizeHist(img_crop)
46: plt.title("Histograma de la imagen equalizada")
47: plt.hist(img_crop_eq.ravel(),256,[0,256]); plt.show()
48: plt.imshow(img_crop_eq, cmap='gray', vmin=0, vmax=255)
49: plt.title('Imagen recortada equalizada :')
50: plt.show()
#================================================
=====
# Eliminar ruido
51: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
#================================================
=====
# hallar bordes
# 1 opcion canny
52: bordes_canny = cv2.Canny(img_crop_eq_blur,50,250)
53: plt.imshow(bordes_canny, cmap='gray', vmin=0, vmax=255)
54: plt.title('Bordes de imagen por funcion Canny :')
55: plt.show()
#================================================
=====
# 2 opcion sobel
56: SCALE = 1
57: DELTA = 0
58: DDEPTH = cv2.CV_16S  ## to avoid overflow
```

```
59: sobelx = cv2.Sobel(img_crop_eq_blur, DDEPTH, 0, 1, ksize=3, scale=SCALE, delta=DELTA)
60: plt.imshow(sobelx, cmap='gray', vmin=0, vmax=255)
61: plt.title("Sobel, para hallar la posicion de la faja :")
62: plt.show()
63: ret,thresh = cv2.threshold(sobelx,100,255,0)
64: plt.imshow(thresh, cmap='gray', vmin=0, vmax=255)
65: plt.title('Imagen Threshold :')
66: plt.show()
67: sum = np.array
68: sum = np.sum(thresh, axis=1)
69: plt.title("Sumatoria en el eje horizontal")
70: plt.plot(sum)
71: plt.show()

72: max_position = sum.argmax()
73: min_position = sum.argmin()
74: print "maxima position es:",max_position, " con ", sum[max_position], " pixels"
75: thresh[max_position+1:max_position+10:] = 255
76: mostrar (thresh,'Imagen con identificacion de linea de faja :')
#Nuevo recorte para tomar solamente la imagen encima de la faja
77: img = img_crop_eq[0:max_position,:]
78: mostrar (img, "Imagen recortada 2")
79: print "Obteniendo informacion de la imagen:"
80: height, width = img.shape[:2]
81: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
82: hist = cv2.calcHist([img],[0],None,[256],[0,256])
83: mostrar (img,'Imagen descartando la parte inferior de la faja')
84: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
85: img = thresh[0:max_position,:]
86: kernel = np.ones((10,10),np.float32)/25
#Filtra la imagen utilizando el kernel anterior
87: dst = cv2.filter2D(img,-1,kernel)
88: mostrar (img,'Imagen recuperada')
89: mostrar (dst,'Imagen recuperada filtro medio suavizado')

#=====================================================
=
#++++++++++++++++++++++++++++++++++++++++++++++++++++++
+
# nueva imagen para hallar el fondo
#++++++++++++++++++++++++++++++++++++++++++++++++++++++
+
90: img_fondo = cv2.bitwise_not(img_crop_eq)
91: plt.imshow(img_fondo, cmap='gray', vmin=0, vmax=255)
92: plt.title('Imagen invertida para hallar fondo')
93: plt.show()
94: img_fondo_eq = cv2.equalizeHist(img_fondo)
95: mostrar (img_fondo_eq,'Imagen invertida para hallar fondo equalizada')
96: img_fondo_blur_tmp = cv2.GaussianBlur(img_fondo_eq,(3,3),0)
```

```
97: img_fondo_blur = cv2.GaussianBlur(img_fondo_blur_tmp,(5,5),0)
98: mostrar (img_fondo_blur,'imagen invertida equalizada suavizada')
99: ret,thresh1 = cv2.threshold(img_fondo_blur,200,255,cv2.THRESH_BINARY)
100: mostrar (thresh1,'imagen threshold')
101: img2 = cv2.convertScaleAbs(img_fondo_blur)
102: thresh2 = cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
103: mostrar (thresh2,'imagen threshold 2')
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
104: kernel = np.ones((3,3),np.uint8)
105: erode = cv2.erode(thresh1,kernel,iterations = 2)
106: mostrar (erode,'imagen erode')

107: kernel = np.ones((20,20),np.uint8)
108: opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
109: kernel = np.ones((30,30),np.uint8)
110: closing = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)
111: closing2 = cv2.morphologyEx(closing, cv2.MORPH_CLOSE, kernel)
112: closing3 = cv2.morphologyEx(closing2, cv2.MORPH_CLOSE, kernel)
113: closing4 = cv2.morphologyEx(closing3, cv2.MORPH_CLOSE, kernel)
114: kernel = np.ones((3,3),np.uint8)
115: dilated = cv2.dilate(closing4, kernel, iterations = 4)
116: mostrar (closing4,'fondo cerrado')
117: mostrar (dilated,'fondo dilatado')
118: img_inv2 = dilated[0:max_position,:]
119: mostrar (img_inv2,'recortando ')
120: kernel = np.ones((3,3),np.uint8)
121: eroded = cv2.erode(img_inv2,kernel,iterations = 7)
122: mostrar (eroded,'erosionando fondo ')
123: print eroded.dtype
124: img2 = cv2.convertScaleAbs(img)
125: print img2.dtype
#=================================================
#Erosion/dilatacion
126: kernel = np.ones((2,2),np.uint8)
127: ret,thresha = cv2.threshold(img2,200,255,cv2.THRESH_BINARY)
128: ret,threshb = cv2.threshold(eroded,200,255,cv2.THRESH_BINARY)
129: imga = thresha - threshb
130: mostrar (imga,"Imagen restada" )
131: erosionada = cv2.erode(imga,kernel,iterations = 1)
132: mostrar (erosionada,'Imagen erosionada')
#kernel = np.ones((2,2),np.uint8)
#abierta = cv2.morphologyEx(erosionada, cv2.MORPH_OPEN, kernel)
#mostrar (abierta,'Imagen abierta')
133: kernel = np.ones((4,4),np.uint8)
134: dilatada = cv2.dilate(erosionada,kernel,iterations = 17)
135: mostrar (dilatada,'Imagen dilatada')
```

```
# convert for region identification function contours
136: dilatada2 = cv2.convertScaleAbs(dilatada)
#necesario para encontrar contornos
137: ret,dilatada3 = cv2.threshold(dilatada2,200,255,cv2.THRESH_BINARY)
138: contours, hierarchy = cv2.findContours(dilatada3,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
139: print len(contours)
140: cv2.drawContours(dilatada2,contours,-1,(128,0,0),3)
141: mostrar_color (dilatada2,'Resaltando contornos')
142: area = 0
143: for cnt in contours:
144:   if cv2.contourArea(cnt) > area:
145:     area = cv2.contourArea(cnt)
146: print "Area es:",area," pixeles cuadrados"
147: print "Ajuste por imagen imcompleta:",area*3/2," pixeles cuadrados"
148: area = area * 3/2
# conversión a cm2, multiplicando por un factor predefinido por el usuario en
# base a comparación de tamaño de imagen con medidas reales
#10p = 1cm
#1p = 1/10 cm
149: Factor = 1.0/10
150: print "Area es:",area*Factor," cm cuadrados"
#1 cm2 = 0.0001 m2
151: print "Area es:",area*Factor*0.0001," metros cuadrados"
#profundidad es 1m
152: volumen = area*Factor*0.0001
153: print "Volumen es:",area*Factor*0.0001," metro cúbico"
# densidad del plomo 11kg/m3
154: DENSIDAD = 9.9
155: print "Peso es:",volumen*DENSIDAD," Kg"
156: tiempo_final = time()
157: tiempo_ejecucion = tiempo_final - tiempo_inicial
158: print 'El tiempo de ejecucion fue:',tiempo_ejecucion #En segundos
```

ANEXO 4

CODIGO FUENTE DE LA CUARTA MUESTRA

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
1: from matplotlib import pyplot as plt
2: from time import time #importamos la función time para medir el tiempo de ejecución
3: import cv2
4: import numpy as np
5: print "OpenCV version:",cv2.__version__
```

```
6: tiempo_inicial = time()
#=====================================================
======
# Definir funciones para mostrar imagenes
#=====================================================
======
7: def mostrar (img,mensaje):
8:   plt.imshow(img, cmap='gray', vmin=0, vmax=255)
9:   plt.title(mensaje)
10: plt.show()
11: def mostrar_color (img,mensaje):
12:  plt.imshow(img)
13:  plt.title(mensaje)
14:  plt.show()
#=====================================================
======
15: img = cv2.imread('./img/MINERAL19.jpg')
#=====================================================
======
16: print "Obteniendo informacion de la imagen:"
17: height, width = img.shape[:2]
18: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
19: hist = cv2.calcHist([img],[0],None,[256],[0,256])
#=====================================================
======
20: plt.title("Histograma de la imagen")
21: plt.hist(img.ravel(),256,[0,256]); plt.show()
22: mostrar (img[::8, ::8],'Imagen reducida x8 para graficar :')
23: print "Convertir imagen a escala de grises para equalizar el histograma"
24: gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
25: plt.imshow(gray[::8, ::8], cmap='gray', vmin=0, vmax=255)
26: plt.title('Imagen en escala de grises reducida x8 para graficar :')
27: plt.show()
28: plt.title("Histograma de la imagen en escala de grises")
29: plt.hist(gray.ravel(),256,[0,256]); plt.show()
30: img_gray_eq = cv2.equalizeHist(gray)
31: plt.imshow(img_gray_eq[::8, ::8], cmap='gray', vmin=0, vmax=255)
32: plt.title('Imagen equalizada en escala de grises reducida x8 para graficar :')
33: plt.show()
34: plt.title("Histograma de la imagen equalizada")
35: plt.hist(img_gray_eq.ravel(),256,[0,256]); plt.show()
#=====================================================
==============
# recorte de la imagen
36: img_crop = img_gray_eq[700:1600,1100:1800]
37: print "Obteniendo informacion de la nueva imagen:"
38: height, width = img_crop.shape[:2]
39: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
40: plt.title("Histograma de la imagen recortada")
41: plt.hist(img_crop.ravel(),256,[0,256]); plt.show()
```

```
42: plt.imshow(img_crop, cmap='gray', vmin=0, vmax=255)
43: plt.title('Imagen recortada :')
44: plt.show()
45: img_crop_eq = cv2.equalizeHist(img_crop)
46: plt.title("Histograma de la imagen equalizada")
47: plt.hist(img_crop_eq.ravel(),256,[0,256]); plt.show()
48: plt.imshow(img_crop_eq, cmap='gray', vmin=0, vmax=255)
49: plt.title('Imagen recortada equalizada :')
50: plt.show()
#=====================================================
=====
# Eliminar ruido
51: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
#=====================================================
=====
# hallar bordes
# 1 opcion canny
52: bordes_canny = cv2.Canny(img_crop_eq_blur,50,250)
53: plt.imshow(bordes_canny, cmap='gray', vmin=0, vmax=255)
54: plt.title('Bordes de imagen por funcion Canny :')
55: plt.show()
#=====================================================
=====
# 2 opcion sobel
56: SCALE = 1
57: DELTA = 0
58: DDEPTH = cv2.CV_16S  ## to avoid overflow
59: sobelx = cv2.Sobel(img_crop_eq_blur, DDEPTH, 0, 1, ksize=3, scale=SCALE,
delta=DELTA)
60: plt.imshow(sobelx, cmap='gray', vmin=0, vmax=255)
61: plt.title("Sobel, para hallar la posicion de la faja :")
62: plt.show()
63: ret,thresh = cv2.threshold(sobelx,100,255,0)
64: plt.imshow(thresh, cmap='gray', vmin=0, vmax=255)
65: plt.title('Imagen Threshold :')
66: plt.show()
67: sum = np.array
68: sum = np.sum(thresh, axis=1)
69: plt.title("Sumatoria en el eje horizontal")
70: plt.plot(sum)
71: plt.show()

72: max_position = sum.argmax()
73: min_position = sum.argmin()
74: print "maxima position es:",max_position, " con ", sum[max_position], " pixels"
75: thresh[max_position+1:max_position+10:] = 255
76: mostrar (thresh,'Imagen con identificacion de linea de faja :')
#Nuevo recorte para tomar solamente la imagen encima de la faja
77: img = img_crop_eq[0:max_position,:]
78: mostrar (img, "Imagen recortada 2")
```

```
79: print "Obteniendo informacion de la imagen:"
80: height, width = img.shape[:2]
81: print 'Imagen, Tamaño:  Alto:' ,str(height)," Ancho: ",str(width)
82: hist = cv2.calcHist([img],[0],None,[256],[0,256])
83: mostrar (img,'Imagen descartando la parte inferior de la faja')
84: img_crop_eq_blur = cv2.GaussianBlur(img_crop_eq,(3,3),0)
85: img = thresh[0:max_position,:]
86: kernel = np.ones((10,10),np.float32)/25
#Filtra la imagen utilizando el kernel anterior
87: dst = cv2.filter2D(img,-1,kernel)
88: mostrar (img,'Imagen recuperada')
89: mostrar (dst,'Imagen recuperada filtro medio suavizado')

#=================================================
=
#+++++++++++++++++++++++++++++++++++++++++++++++++
+
# nueva imagen para hallar el fondo
#+++++++++++++++++++++++++++++++++++++++++++++++++
+
90: img_fondo = cv2.bitwise_not(img_crop_eq)
91: plt.imshow(img_fondo, cmap='gray', vmin=0, vmax=255)
92: plt.title('Imagen invertida para hallar fondo')
93: plt.show()
94: img_fondo_eq = cv2.equalizeHist(img_fondo)
95: mostrar (img_fondo_eq,'Imagen invertida para hallar fondo equalizada')
96: img_fondo_blur_tmp = cv2.GaussianBlur(img_fondo_eq,(3,3),0)
97: img_fondo_blur = cv2.GaussianBlur(img_fondo_blur_tmp,(5,5),0)
98: mostrar (img_fondo_blur,'imagen invertida equalizada suavizada')
99: ret,thresh1 = cv2.threshold(img_fondo_blur,200,255,cv2.THRESH_BINARY)
100: mostrar (thresh1,'imagen threshold')
101: img2 = cv2.convertScaleAbs(img_fondo_blur)
102:                                    thresh2                                    =
cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THR
ESH_BINARY,11,2)
103: mostrar (thresh2,'imagen threshold 2')
#+++++++++++++++++++++++++++++++++++++++++++++++++
+
104: kernel = np.ones((3,3),np.uint8)
105: erode = cv2.erode(thresh1,kernel,iterations = 2)
106: mostrar (erode,'imagen erode')

107: kernel = np.ones((20,20),np.uint8)
108: opening = cv2.morphologyEx(thresh1, cv2.MORPH_OPEN, kernel)
109: kernel = np.ones((30,30),np.uint8)
110: closing = cv2.morphologyEx(thresh1, cv2.MORPH_CLOSE, kernel)
111: closing2 = cv2.morphologyEx(closing, cv2.MORPH_CLOSE, kernel)
112: closing3 = cv2.morphologyEx(closing2, cv2.MORPH_CLOSE, kernel)
113: closing4 = cv2.morphologyEx(closing3, cv2.MORPH_CLOSE, kernel)
114: kernel = np.ones((3,3),np.uint8)
```

```
115: dilated = cv2.dilate(closing4, kernel, iterations = 4)
116: mostrar (closing4,'fondo cerrado')
117: mostrar (dilated,'fondo dilatado')
118: img_inv2 = dilated[0:max_position,:]
119: mostrar (img_inv2,'recortando ')
120: kernel = np.ones((3,3),np.uint8)
121: eroded = cv2.erode(img_inv2,kernel,iterations = 7)
122: mostrar (eroded,'erosionando fondo ')
123: print eroded.dtype
124: img2 = cv2.convertScaleAbs(img)
125: print img2.dtype
#=================================================
=====
#Erosion/dilatacion
126: kernel = np.ones((2,2),np.uint8)
127: ret,thresha = cv2.threshold(img2,200,255,cv2.THRESH_BINARY)
128: ret,threshb = cv2.threshold(eroded,200,255,cv2.THRESH_BINARY)
129: imga = thresha - threshb
130: mostrar (imga,"Imagen restada" )
131: erosionada = cv2.erode(imga,kernel,iterations = 1)
132: mostrar (erosionada,'Imagen erosionada')
#kernel = np.ones((2,2),np.uint8)
#abierta = cv2.morphologyEx(erosionada, cv2.MORPH_OPEN, kernel)
#mostrar (abierta,'Imagen abierta')
133: kernel = np.ones((4,4),np.uint8)
134: dilatada = cv2.dilate(erosionada,kernel,iterations = 17)
135: mostrar (dilatada,'Imagen dilatada')
# convert for region identification function contours
136: dilatada2 = cv2.convertScaleAbs(dilatada)
#necesario para encontrar contornos
137: ret,dilatada3 = cv2.threshold(dilatada2,200,255,cv2.THRESH_BINARY)
138:                contours,              hierarchy              =
cv2.findContours(dilatada3,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
139: print len(contours)
140: cv2.drawContours(dilatada2,contours,-1,(128,0,0),3)
141: mostrar_color (dilatada2,'Resaltando contornos')
142: area = 0
143: for cnt in contours:
144:   if cv2.contourArea(cnt) > area:
145:     area = cv2.contourArea(cnt)
146: print "Area es:",area," pixeles cuadrados"
147: print "Ajuste por imagen imcompleta:",area*3/2," pixeles cuadrados"
148: area = area * 3/2
# conversion a cm2, multiplicando por un factor predefinido por el usuario en
# base a comparacion de tamano de imagen con medidas reales
#10p = 1cm
#1p = 1/10 cm
149: Factor = 1.0/10
150: print "Area es:",area*Factor," cm cuadrados"
#1 cm2 = 0.0001 m2
```

```
print "Area es:",area*Factor*0.0001," metros cuadrados"
#profundidad es 1m
volumen = area*Factor*0.0001
print "Volumen es:",area*Factor*0.0001," metro cúbico"
# densidad del plomo 11kg/m3
DENSIDAD = 9.9
print "Peso es:",volumen*DENSIDAD," Kg"
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print 'El tiempo de ejecucion fue:',tiempo_ejecucion #En segundos
```

ANEXO 5

MATRIZ DE CONSISTENCIA

FORMULACION DEL PROBLEMA	OBJETIVOS	HIPOTESIS	VARIABLES	INDICADORES
¿Cómo diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora y reducir la parada de producción en la unidad minera Mallay?	Diseñar un algoritmo de procesamiento de imágenes del sistema de pesaje para el control automático de una faja transportadora y reducir la parada de producción en la unidad minera Mallay	El diseño de un algoritmo de procesamiento de imágenes del sistema de pesaje permitirá el control automático de la faja transportadora y reducir la parada de producción en la unidad minera Mallay.	**V.I** Diseño de algoritmo **V.D** reducción tiempo de parada de producción	• Algoritmo • Imágenes de sistema de pesaje • tiempo producción
¿Cómo desarrollar un algoritmo de procesamiento de imágenes para la determinación del volumen en la faja transportadora?	Desarrollar un algoritmo de procesamiento de imágenes para determinar el volumen en la faja transportadora.	Contar con un algoritmo de procesamiento de imágenes que determine el volumen de la faja transportadora.	**V.I** Desarrollo de algoritmo **V.D** procesamiento de imágenes	
¿Cómo obtener el volumen del material que ingresa a la faja transportadora en base a la equivalencia de un pixel en centímetros cuadrados y la densidad de la muestra?	Obtener el volumen del material que ingresa en la faja transportadora en base a la equivalencia de un pixel en centímetros cuadrados y la densidad de la muestra.	El volumen del material que ingresa en la faja transportadora se obtendrá en base a la equivalencia de un pixel en centímetros cuadrados y la densidad de la muestra.	**V.I** volumen del material **V.D** equivalencia de un pixel en centímetros cuadrados	