



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO



FACULTAD DE  
INGENIERÍA

## **Trabajo Final Inteligencia Artificial I – año 2022:** Visión Artificial

### **Inteligencia Artificial I**

#### **Proyecto:**

Identificación mediante visión artificial el contenido de cuatro cajas que se encuentran apiladas

#### **Profesora:**

Prof. Titular Dra. Ing. Selva S. Rivera

#### **Alumno:**

Mario Fernando Bustillo López

Facultad de Ingeniería, Universidad Nacional de Cuyo  
Ciclo Lectivo 2022

## **RESUMEN**

Este proyecto de visión artificial consiste en la clasificación de piezas metálicas (tornillos, clavos, tuercas y arandelas) para la identificación del contenido de cajas que se encuentran apiladas. La solución al problema consta de tres etapas: Procesamiento de Imágenes, Pasos que debe realizar el robot para alcanzar otro orden de apilado y Transportar las cajas desde un punto A a un punto B. Se definió un agente racional que aprende, no omnisciente y que a partir de los algoritmos KNN y K-Means se entrene, aprenda a partir de sus percepciones y le permita ser autónomo gracias a la experiencia adquirida. Además el agente dispone del algoritmo del método A estrella ( $A^*$ ) para obtener la solución más corta y eficiente entre los puntos objetivos en el laberinto. Como objetivo se busca obtener un agente que en un entorno con condiciones controladas se desempeñe con un alto rendimiento. Se obtuvo un alto rendimiento en la identificación de piezas metálicas para ambos algoritmos, así como para la ejecución de los procesos involucrados en las otras etapas. La solución planteada bajo las condiciones establecidas, permitió obtener un alto rendimiento ante el problema planteado. Se utilizó el lenguaje de programación Python 3.0.

## **INTRODUCCIÓN**

La visión artificial es un campo de la inteligencia artificial (IA) que permite a los ordenadores y sistemas extraer información significativa a partir de imágenes digitales, videos y otras entradas visuales, y tomar medidas o realizar recomendaciones en función de esa información. Si la IA permite a los ordenadores pensar, la visión artificial les permite ver, observar y comprender.

La visión artificial entrena a las máquinas para realizar estas funciones, pero tiene que hacerlo en mucho menos tiempo con cámaras, datos y algoritmos en lugar de retinas, nervios ópticos y una corteza visual. Debido a que un sistema capacitado para inspeccionar productos o la manufactura de estos puede analizar miles de productos o procesos por minuto, que puede superar rápidamente las capacidades humanas, notando defectos o problemas imperceptibles.

El problema a resolver en este proyecto es la identificación mediante visión artificial del contenido de cuatro cajas que se encuentran apiladas. Una caja contiene tornillos, una segunda caja contiene tuercas, una tercera caja contiene clavos y la cuarta caja contiene arandelas; las cuales se encuentran apiladas en un orden aleatorio. Posteriormente al reconocer el orden de apilamiento de las cajas, mediante lenguaje STRIPS se debe encontrar los pasos que debe realizar el brazo robótico para alcanzar un nuevo orden de apilamiento de las cajas. Una vez apiladas las cajas en la nueva configuración, el robot deberá transportarlas desde un punto A a un punto B a través de un laberinto preestablecido, utilizando el algoritmo A\* para encontrar el camino más corto con Heurística de distancia de Manhattan.

El algoritmo que se plantea está conformado por varias etapas: Adquisición de imágenes (a través de una caja cerrada con fondo blanco, con iluminación frontal y comunicación por IP entre cámara y computadora); Transformación (para tener una imagen normalizada); Adaptación y preprocessamiento; Filtración (se utilizó filtro Gauss y Sobel) y segmentación; Extracción de rasgos (se utilizó Momentos de Hu, eje Menor y eje Mayor de la pieza) y reducción; Base de datos y rendimiento; Clasificación de objetos con KNN y K-Means; Reordenamiento del apilado de cajas aplicando una secuencia en lenguaje STRIPS; Transporte de cajas entre dos puntos en un laberinto establecido utilizando el método de A estrella (A\*) y Diseño de Interfaz de Usuario (GUI).

## ESPECIFICACIÓN DEL AGENTE

### Tipo de Agente

Es un agente que aprende. El aprendizaje permite que el agente opere en medios inicialmente desconocidos y que sea más competente que si sólo utilizase un conocimiento inicial. En este caso aprende debido a los métodos que utiliza KNN y KMeans. La base de datos le proporciona un “modelo” de su entorno, también los algoritmos en base a la “utilidad”.

También el agente es racional, esto implica: “En cada posible secuencia de percepciones, un agente racional deberá emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado”.

También es un agente no omnisciente debido a que no conoce el resultado de su acción y no actúa de acuerdo a él. La idea es que a partir de los algoritmos KNN y K-Means se entrene, aprenda a partir de sus percepciones y le permita ser autónomo gracias a la experiencia adquirida.

Basado en lo anterior, estamos frente a un agente inteligente, ya que percibe su entorno, procesa tales percepciones y responde o actúa en su entorno de manera racional, es decir, de manera correcta y tendiendo a maximizar un resultado esperado.

Para este proyecto el agente es una computadora donde están alojados los algoritmos KNN, K-Means y A\*. Los datos de entrada y salida son adquiridos por cámaras fotográficas, monitor, etc.

### **REAS**

El acrónimo REAS proviene de Rendimiento, Entorno, Actuadores y Sensores, todo ello forma parte de lo que se llama Entorno de trabajo.

Rendimiento	<ul style="list-style-type: none"> <li>- Confiabilidad de las predicciones con un margen de error pequeño.</li> <li>- Cantidad de predicciones correctas de las imágenes que se están evaluando.</li> <li>- Categorización de imagen correcta.</li> <li>- Porcentaje de piezas clasificadas de acuerdo a su categoría.</li> <li>- La menor cantidad de movimientos para obtener el nuevo orden de apilamiento.</li> <li>- La distancia más óptima para alcanzar la posición objetivo.</li> <li>- Obtener una ruta óptima ante un laberinto preestablecido.</li> </ul>
Entorno	<ul style="list-style-type: none"> <li>- Es el contexto plasmado en la imagen de entrada. Ampara factores como la iluminación, calidad de la imagen, fondo, etc.</li> <li>- Cajas con piezas de ferretería.</li> <li>- Laberinto preestablecido.</li> </ul>
Actuadores	<ul style="list-style-type: none"> <li>- Es el medio de notificación del usuario donde se muestra la</li> </ul>

	<p>predicción del algoritmo. En este caso es el monitor de la computadora, también podría ser una alarma u otro tipo de alarma.</p> <ul style="list-style-type: none"> <li>- Visualizar la categorización de una imagen.</li> <li>- Brazo Robótico con el cuál se manipulan las cajas.</li> </ul>
Sensores	<ul style="list-style-type: none"> <li>- Cámara fotográfica.</li> <li>- Extensión IP Webcam para obtener las imágenes de internet.</li> <li>- Matriz de píxeles de colores.</li> <li>- Módulos skimage, cv2, numpy, matplotlib, etc.</li> </ul>

*Tabla 1: Descripción del REAS del agente diseñado.*

## **Propiedades del Entorno**

Totalmente Observable	Los sensores previamente mencionados le proporcionan acceso al estado completo del medio en cada momento.
Determinista	El estado del medio está totalmente determinado por el estado actual y la acción ejecutada del medio. La imagen de entrada una vez tomada es inalterable en principio.
Episódico	La elección de la acción en cada episodio depende del episodio en sí mismo. Uno puede tomar las fotografías en cualquier orden de tornillos, clavos, arandelas o tuercas y el rendimiento del agente no se verá afectado.
Estático	El entorno no cambia cuando el agente está deliberando. Una vez tomada la fotografía, ese dato de entrada se mantiene inalterable.
Discreto	Las imágenes captadas por las cámaras son discretas, en sentido estricto, pero se tratan típicamente como representaciones continuas de localizaciones e intensidades variables.
Individual	El único ente racional es el agente previamente descrito.

*Tabla 2: Descripción de las propiedades del entorno donde el agente está diseñado.*

## DISEÑO DEL AGENTE

### Reconocimiento de objetos

El reconocimiento de objetos es la tarea de encontrar e identificar automáticamente objetos específicos en una imagen. Actualmente no existe un sistema artificial que “reconoce-todo”.

Un sistema de reconocimiento de objetos generalmente está constituido por las siguientes etapas básicas:

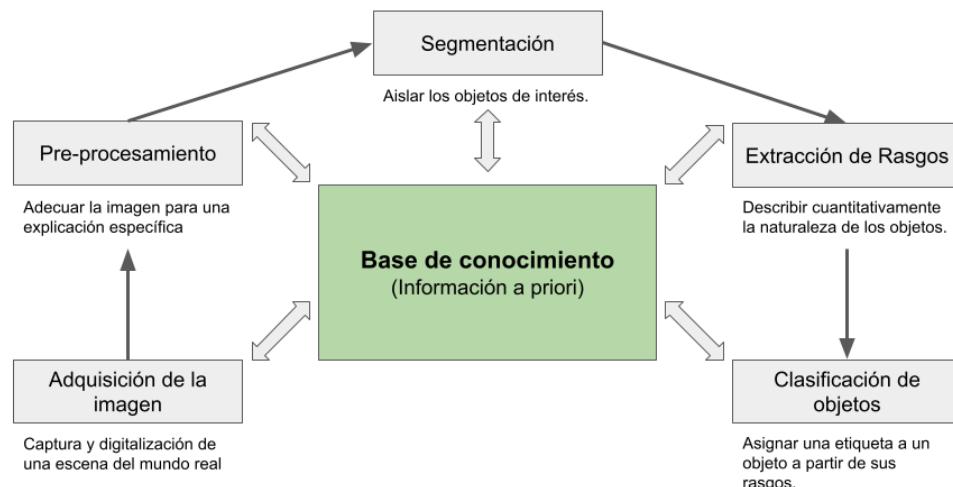


Ilustración 1: Esquema de modelo general de reconocimiento de objetos.

### Desarrollo del sistema reconocimiento de objetos

Problema: Reconocer automáticamente 4 objetos de ferretería distintos (tornillos, tuercas, arandelas y clavos) y determinar el orden en el qué están apiladas las cajas que las contienen.

Características notables de las imágenes: Ruido, objetos contrastados del fondo, iluminación heterogénea (zonas más brillantes que otras), diferentes posiciones y localizaciones.

Propuesta de solución: Basado en las etapas básicas de un sistema de reconocimiento de objetos se obtuvo lo siguiente:

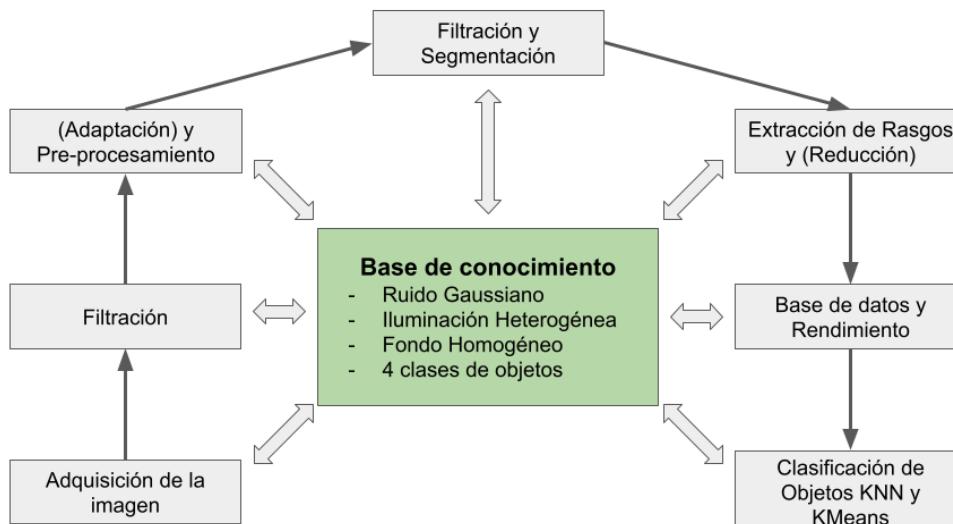


Ilustración 2: Propuesta de modelo de reconocimiento de objetos para este proyecto.

## **Breve descripción de objetos de estudio**

### **1) Tornillo**

Se pueden encontrar de muchas variedades de materiales, tipos y tamaños que existen para distintas aplicaciones. En él se distinguen tres partes básicas: cabeza, cuello y rosca.

Una primera clasificación puede ser: Tornillos tirañodos para madera, autorroscantes y autoperforantes para chapas metálicas y maderas duras, tornillos tirañodos para paredes y muros de edificios, tornillos de roscas cilíndricas y varillas roscadas de un metro de longitud.

Pueden ser de acero dulce, inoxidable, latón, cobre, bronce o aluminio, y pueden estar galvanizados, niquelados, bicromatados, etc.

Otra clasificación sería desde el punto de vista de utilización: Tornillos para usos generales, en miniatura, de alta resistencia, inviolables, de precisión, de titanio, etc.

### **2) Tuerca**

Es una pieza mecánica con un orificio central, el cual presenta una rosca, que se utiliza para acoplar a un tornillo, en forma fija o deslizante.

Las características básicas para identificar a una tuerca son: Número de caras, grosor, diámetro del tornillo que encaja en ella y el tipo de rosca.

Tipos de tuercas: Tuerca mariposa o de ala, hexagonal, con rebordes, ciegas, cuadradas, T, etc.

### **3) Arandelas**

Es un elemento de montaje con forma de disco delgado con un agujero usualmente en el centro (corona circular), siendo su uso más frecuente el sentar tuercas y cabezas de tornillos. Las arandelas normalmente son de metal o de plástico.

Tipos de arandelas: Plana, de presión, dentada, cónica, de cuero, etc.

### **4) Clavos**

Es un objeto delgado y alargado con punta filosa hecho de un metal duro (por lo general acero), utilizado para sujetar dos o más objetos.

Los clavos se clasifican de acuerdo con su uso, el diámetro, acabado y longitud. Podemos clasificarlos, según el tipo de cabeza, de punta o de cuello: clavo de cabeza ancha, clavo de escarpia, clavo para tornillos, clavo de acero, etc.

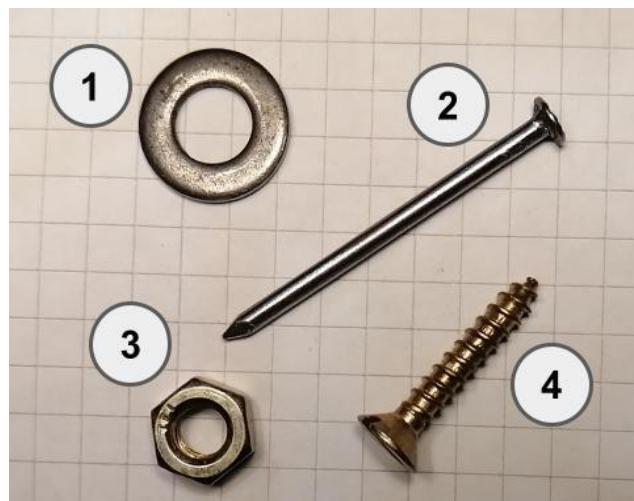


Ilustración 3: Piezas de ferretería utilizados en este proyecto: (1) Arandela de acero común, con  $d_1=16$  mm y  $d_2=8$  mm; (2) Clavo de acero común, con  $d=4$  mm y  $L=42$  mm; (3)Tuerca hexagonal, con  $d=5$  mm y hexagonal=10 mm; (4) Tornillo para madera autoperforante zincado  $d=4$  mm y  $L=23$  mm.

## **Adquisición de la imagen con cámara convencional**

La primera etapa, dentro de un proceso de visión computacional, es la etapa de adquisición de imágenes. En este primer paso, se trata de conseguir que la imagen sea lo más adecuada posible para que se pueda continuar con las siguientes etapas. Una correcta adquisición de la imagen supone un paso muy importante para que el proceso de reconocimiento tenga éxito. Dentro de esta etapa existen múltiples factores que intervienen directamente al proceso de captura de la imagen, formados fundamentalmente por: el sistema hardware de visión artificial (cámara, óptica, tarjeta de adquisición de imagen, ordenador y software) y el entorno y posicionamiento de los elementos (la iluminación, el fondo, posición correcta de la cámara, ruido eléctrico-óptico externo, etc.). Por lo tanto se analizaron los siguientes factores para reducir distorsiones de las fotografías de las piezas:

### **- Técnicas de iluminación**

Un entorno debidamente controlado es imprescindible para obtener unas condiciones de partida óptimas que aseguren una perfecta adquisición. Dentro del entorno aparecen como partes fundamentales en las expectativas tanto en calidad como de la imagen buscada: la iluminación, el fondo, la posición de la cámara, etc.

#### **1) La iluminación**

La iluminación de la escena tiene que realizarse de una forma correcta, dada la importancia que tiene. Existen fundamentalmente dos formas de iluminación: Iluminación frontal o trasera, y cada una de ellas tienen sus variantes.

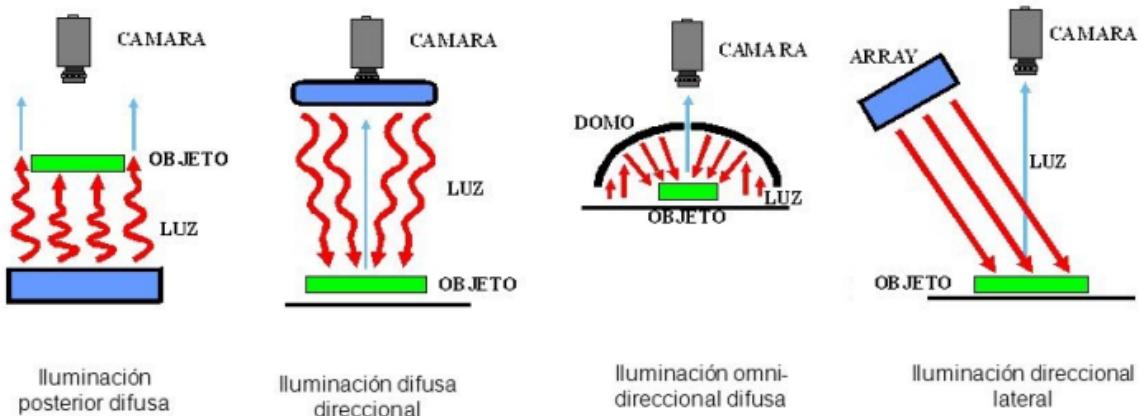


Ilustración 4: Imagen obtenida de: Presentación "Visión por computadora" por Gustavo Cerezo

La iluminación frontal permite distinguir claramente los detalles del mismo, pero esta iluminación puede presentar diferentes inconvenientes como: la creación de sombras y los reflejos. La mejor forma de evitar estos dos efectos, consiste en utilizar luz difusa ya sea con fuentes especiales que generan luz difusa (por ejemplo fibras ópticas que expanden la luz en todas direcciones), por el uso de lámparas circulares de luz que iluminen de forma homogénea o por el uso de luz indirecta. Por otro lado, como el sistema va a trabajar en un entorno industrial, es conveniente evitar las posibles interferencias externas (por ejemplo que las lámparas no estén totalmente fijas o que exista un juego entre la cámara y el soporte), también se usarán paredes que reduzcan lo más posible la entrada de luz ambiental.

## 2) Fondo

El fondo de la escena cumple un papel esencial cuando se trata de simplificar alguna de las etapas subsiguientes (como puede ser la segmentación). Éste debe ser lo más homogéneo posible y de un color que permita distinguirlo fácilmente de los objetos. Cualquier mancha o defecto que existan en el fondo, puede ocasionar errores en la etapa de reconocimiento.

Utilizando iluminación frontal, el fondo debe de ser lo más opaco posible evitando todo reflejo. El color negro opaco suele ser el más utilizado.

Para este proyecto, se empleó iluminación direccional frontal, que consiste en colocar la cámara apuntando al objeto e iluminarlo en la misma dirección de la cámara, con un bajo ángulo de incidencia. De este modo la cámara recibe la mayor parte de la luz reflejada por el objeto, para ello se seleccionó iluminación Led (un panel de iluminación recargable de 1200mAh con 60 diodos led, de intensidad lumínica y color de temperatura regulable, con una intensidad al 60%) con fondo blanco. Se descarto el uso de fondo negro porque no se obtuvieron buenos resultados con él.

## - Cámara

Para obtener una imagen correcta sin deformación, es necesario efectuar un perfecto calibrado. Para la calibración de la posición de la cámara del celular se utilizó la app “Nivel de Burbuja”, que utiliza el acelerómetro y giroscopio (en función del modelo de smartphone) para

ayudar a alinear/nivelar, suspender horizontal, vertical o en cualquier ángulo para ajustar cada objeto. Para este proyecto se consideró una inclinación en  $x=0,2^\circ$  y  $y=1,0^\circ$ .

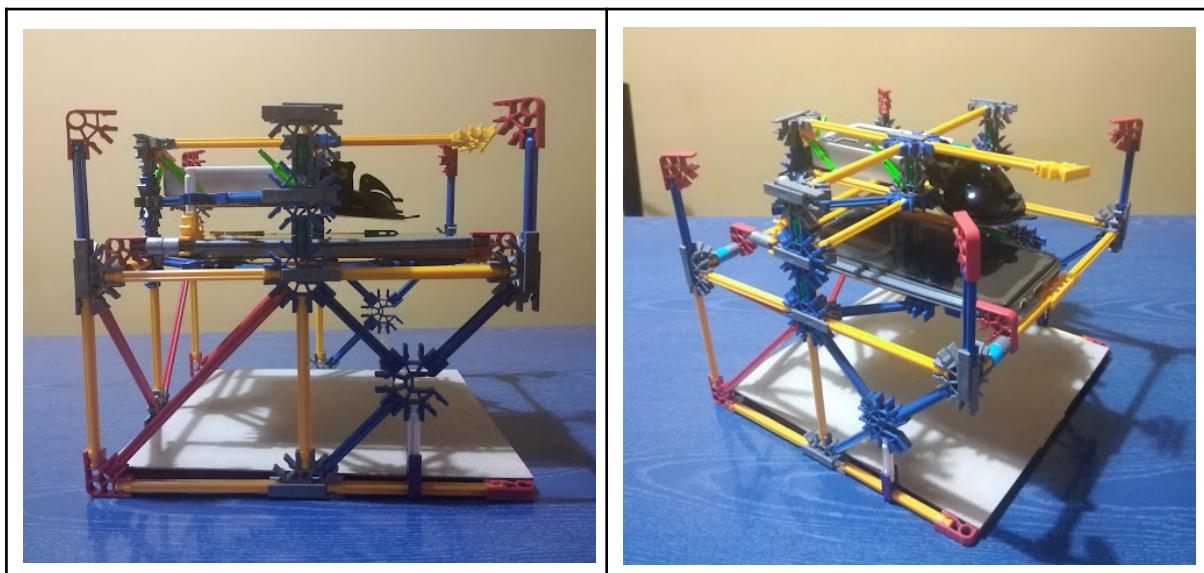


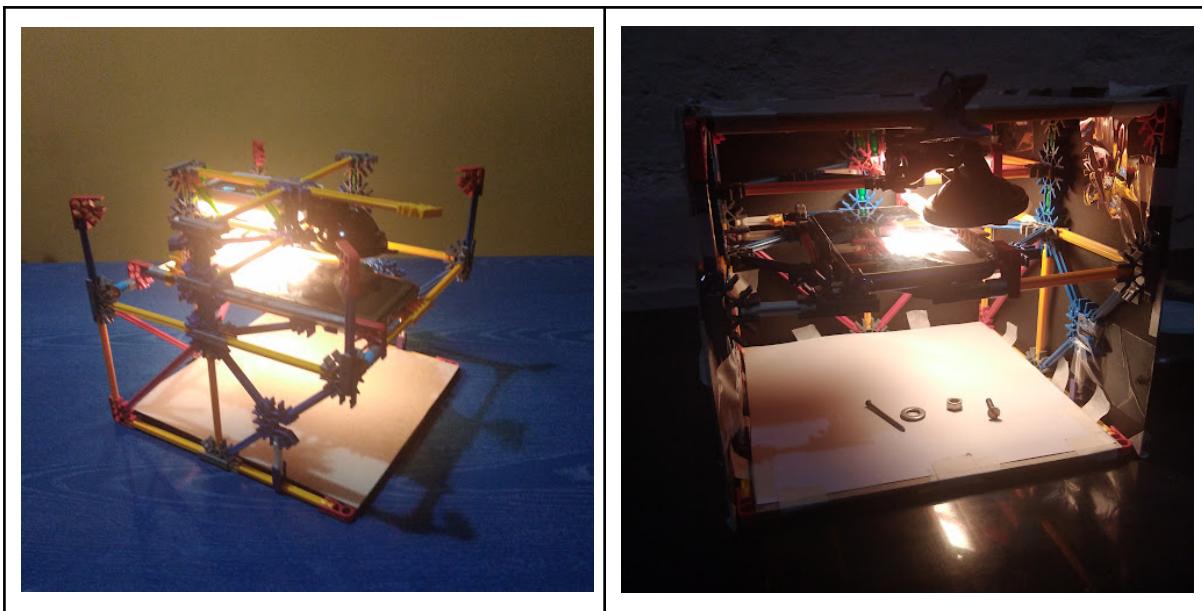
Ilustración 5: Calibración de la posición de la cámara del celular con la ayuda de la app Nivel de burbuja.

Para tomar las fotografías se utilizó la cámara del celular HUAWEI Y9 Prime 2019. La cámara trasera cuenta con triple lente, uno principal de 16MP, un gran angular de 8 MP y un lente adicional de 2 MP, cuyo lente principal tiene detección por fase PDAF y apertura focal de 1,8.

### **- Sistema de captación de imágenes**

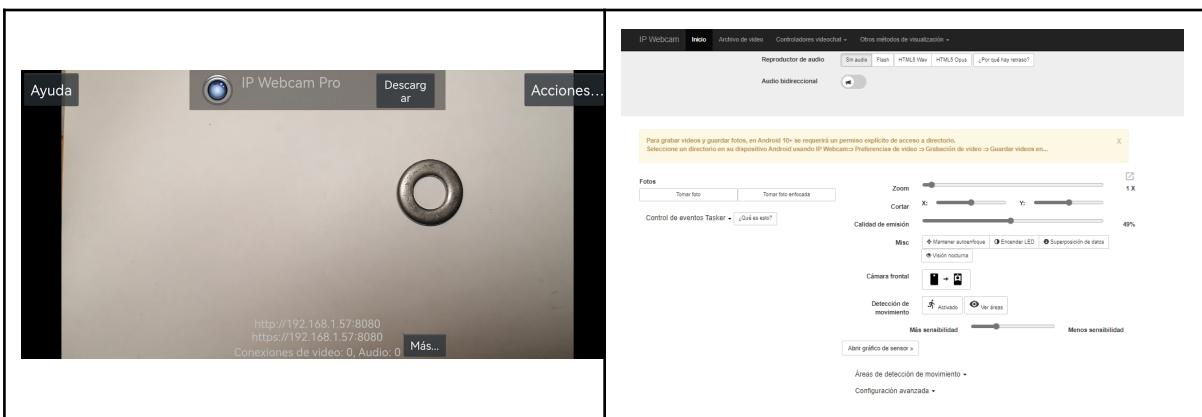
Para evitar las variaciones de las condiciones ambientales y de las distancias de medición, se decidió utilizar una atmósfera controlada mediante el diseño y armado de una caja hecha con material de piezas K'nex tapizada de negro en su interior, además cuenta con un panel de iluminación recargable de 1200mAh con 60 diodos led, de intensidad lumínica y color de temperatura regulable, con una intensidad al 60%. También cuenta con un soporte para el celular paralelo a la base, donde irá el objeto de ferretería a analizar.





*Ilustración 6: Sistema de captación de imágenes. Consiste en una estructura hecha de piezas K'nex forrada con cartulina negra excepto en la base que tiene cartulina blanca, la cual se cierra al momento de tomar las fotos, evitando así el ingreso de luz desde el exterior.*

Se utilizó la app “IP Webcam” que convierte el celular en una cámara en red con múltiples opciones de visualización. Funciona en cualquier plataforma con VLC player o navegador web. Emite a través de una red WiFi sin conexión a internet. Con ésto se hace que un celular se convierta en servidor, y mediante la dirección IPv4 se ingresa en el buscador de la computadora, se maneja todo desde ahí y se guarda directamente la foto en la computadora, agilizando el proceso de captación de imágenes.



*Ilustración 7: Captación de las imágenes con el uso de la app IP Webcam*

## **Transformación**

En esta etapa se cuenta la cantidad de objetos en la imagen, se recorta la imagen si es necesario y se hace una redimensión para tener un formato de imagen normalizado . Todo ésto para tener un estándar de imagen de entrada para la etapa de preprocesamiento.

### **- Quitar línea de la pared**

Al momento de tomar la imagen, debido a la posición de la cámara dentro de la caja toma una parte de la pared perpendicular a la base donde se encuentra el objeto, por eso se hace un recorte y se elimina la franja.

Para ello se eliminan 50 px de izquierda a derecha de la imagen, por lo que se pasa de tener una imagen de 1920x1080 px a una imagen de 1870x1080 px.



Ilustración 8: Remoción de franja de pared de la caja dentro de la imagen.

### **- Quitar fondo y convertirlo a uno totalmente blanco**

Uno de los problemas que se presenta en la caja donde toman las fotos es que se presenta una sombra que se aprecia muy bien cuando se aplica una segmentación de thresholding (separar un objeto de interés del fondo de una imagen), una etapa que se verá más adelante. Esa sombra envuelve a la pieza de ferretería y hace que se pierda entre la sombra. La solución propuesta es cargar la imagen y crear en paralelo una máscara color gris (específicamente tiene la tupla RGB:(100,100,100), estos datos se obtuvieron haciendo uso de la página web: [www.pinetoools.com](http://www.pinetoools.com)), con prueba y error se determinó los valores del rect que permite extraer la pieza de ferretería del fondo.

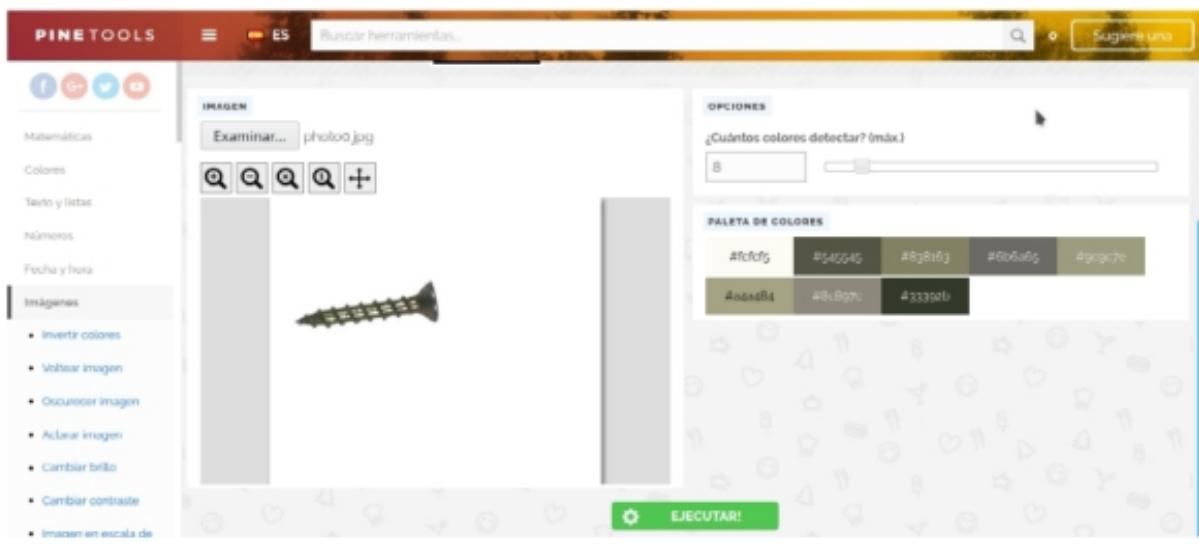


Ilustración 9: En la imagen se aprecia como la detección de colores con la página pinetoools.



Ilustración 10: (1) En la imagen izquierda se muestra la imagen sin la pared de la caja con fondo normal.  
(2) En la imagen derecha se aprecia la imagen con fondo totalmente blanco.

Una vez realizado eso, se toma la máscara cortada menos la imagen original y eso dio lugar a una imagen con fondo totalmente blanco. Prácticamente lo que hizo el programa fue convertir los píxeles en blanco menores a aquellos píxeles con valores menores a la tupla RGB (100,100,100). Así como convierte los píxeles del fondo también lo hace con la pieza de ferretería, por eso se pierde algo de información de la pieza por esta etapa.



Ilustración 11: Representación de la aplicación de filtros para obtener un fondo blanco.

### **- Redimensionamiento de las imágenes**

Las imágenes de entrada están en formato .jpg, las cuales presentan distintos tamaños. Es muy importante trabajar con las mismas dimensiones, porque los descriptores visuales se verán modificados por esta variación como se verá más adelante.

Luego de quitar la franja de pared de la imagen, se tiene una imagen de entrada de 1870x1080px que tiene una estructura de (1080,1870,3). Después de la primera dimensión, a la salida se tiene una imagen cuya estructura es de (400,500,3). Éste último representa una imagen de 500x400 px y las 3 capas correspondientes al RGB.

Este paso se hace con el fin de redimensionar la imagen a un tamaño más pequeño, con el fin de agilizar el proceso de corte de la sombra que se genera por las condiciones del sistema de captación de imágenes.

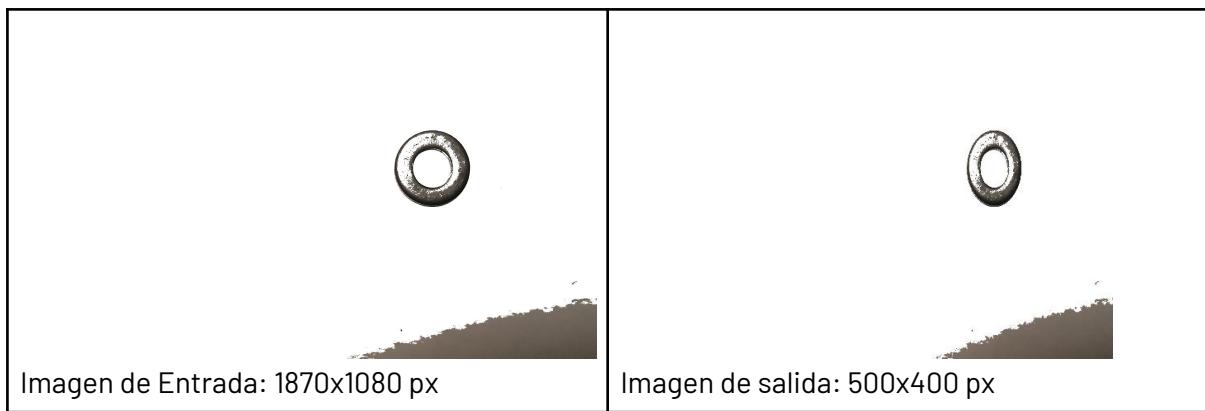


Ilustración 12: Redimensión de la imagen

### - Quitar sombra posterior a quitar fondo

Debido a la iluminación, se presenta un área que no tiene el mismo grado de iluminación que el resto del fondo y se presenta una sombra, por lo que al aplicar el enmascaramiento para quitar el fondo se detecta como una nueva superficie adicional a la de nuestra pieza. Por lo tanto, se debe recortar el área donde se presenta esta sombra para no agregar una mayor superficie a la pieza, para ello se hace un recorte y se elimina la franja. Para ello se eliminan 100px de abajo hacia arriba de la imagen, se pasa de tener una imagen de 500x400 px a una imagen de 500x300 px; el cuál será el formato a utilizar para este proyecto.

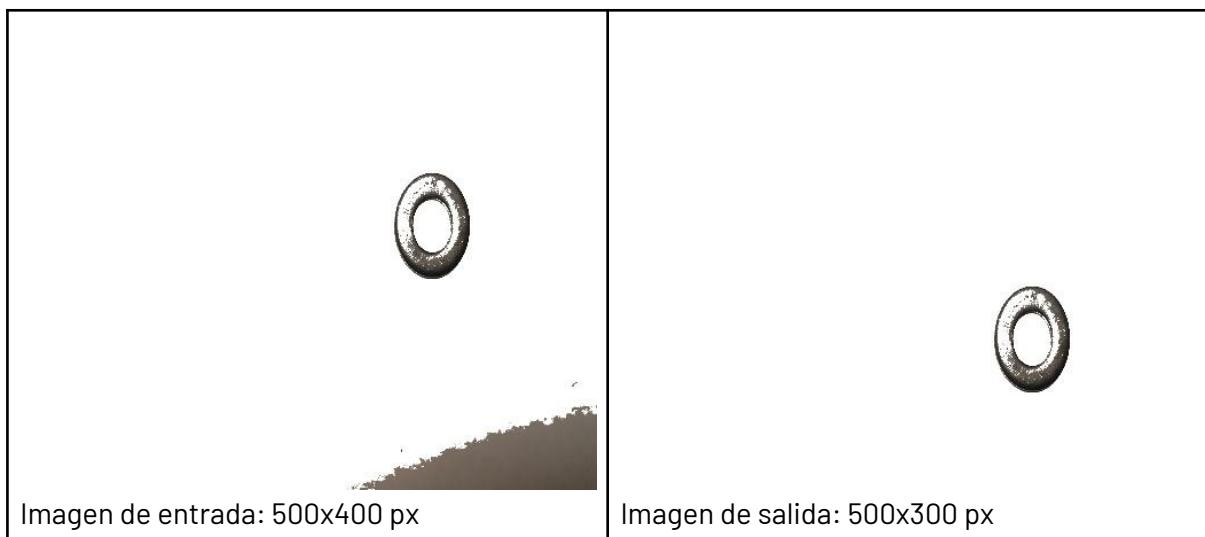


Ilustración 13: Recorte de sombra

### Adaptación

Esta etapa es opcional. En un principio fue planteada la idea de contar los objetos dentro de la imagen, luego fraccionar esa imagen para cada objeto encontrado y luego recortarla alrededor del objeto encontrado para su posterior análisis.

Una gran ventaja que permite realizar el crop de la imagen, es reducir el espacio de búsqueda al momento de evaluar los algoritmos KNN y K-Means más adelante. Ésto reduce los tiempos de ejecución significativamente. Pero la cuestión incide que al armar la base de datos con imágenes cortadas para el algoritmo planteado no se tuvo un gran rendimiento respecto a no

cortarlas, por ello se planteó su implementación aunque posteriormente al momento de armar la base de datos no se utilizó.

### **- Contar cantidad de elementos para un posterior recorte de los elementos dentro de la imagen**

Se aplican dos métodos distintos para realizar el conteo:

#### **1) Algoritmo de Canny para detectar contornos**

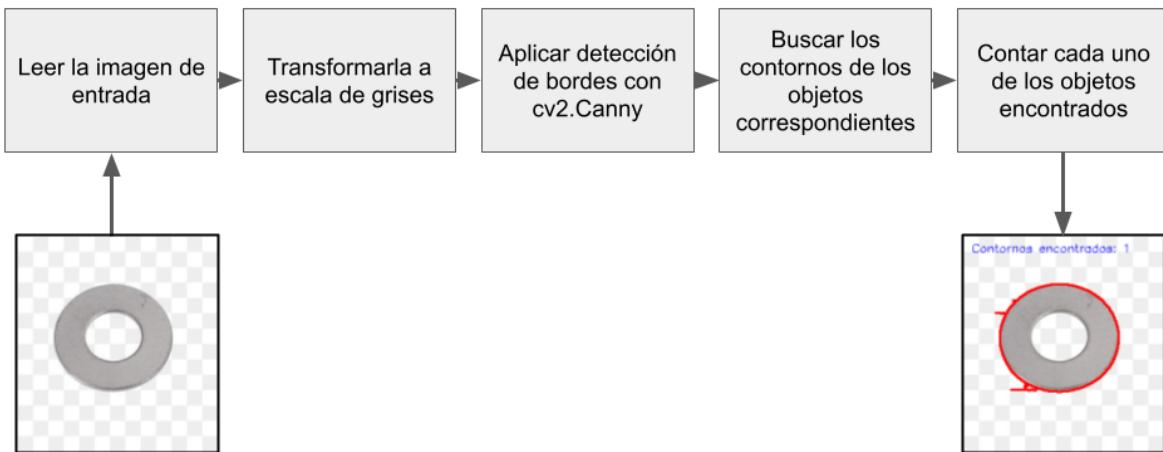


Ilustración 14: Proceso de detección de bordes.

El algoritmo de Canny es un operador que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes, cuyo propósito es descubrir el algoritmo óptimo de detección de bordes. Para que un detector de bordes pueda ser considerado óptimo debe cumplir los siguientes puntos:

- Buena detección: El algoritmo debe marcar el mayor número real en los bordes de la imagen como sea posible.
- Buena localización: Los bordes de marca deben estar lo más cerca posible del borde de la imagen real.
- Respuesta mínima: El borde de una imagen sólo debe ser marcado una vez, y siempre que sea posible, el ruido de la imagen no debe crear falsos bordes.

Cv2.Canny nos exige una imagen de entrada, primer umbral (`minVal`) y segundo umbral(`maxVal`). Para que los bordes se dibujen o no, se necesita de dos umbrales (`minVal` y `maxVal`). De este modo, todos los valores superiores se dibujaran. Los bordes que estén dentro del `maxVal` y `minVal` se dibujaran solo si se encuentran conectados a los que superan `maxVal`. Por el contrario, todos los bordes debajo del `minVal` no se dibujan. Se deben buscar los valores de umbral más adecuados para cada imagen aunque, ya que no importa que se detecten más bordes, sino que sean curvas cerradas.

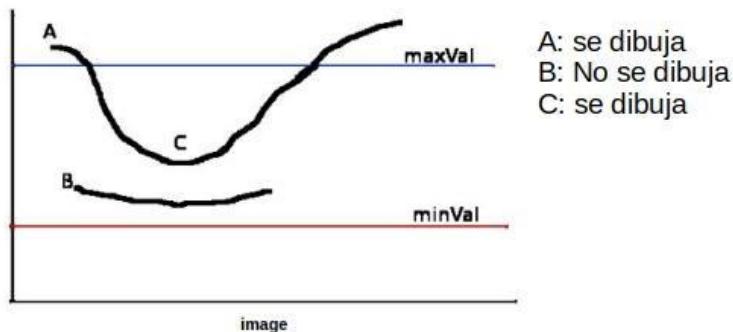


Ilustración 15: Imagen obtenida de la librería OpenCv sobre funcionamiento del algoritmo de Canny.

Una vez que se tiene claro el uso de estos dos umbrales se pueden hacer pruebas para analizar cómo se van dibujando los distintos bordes dada una imagen. Se presentaron una serie de resultados a distintos valores de umbrales: Para el caso de una arandela con umbrales ( $\text{minVal}=100$ ,  $\text{maxVal}=200$ ) la detectaba bien, pero ese valor para múltiples arandelas o tornillos contaba mucho más de lo que debería, debido a la complejidad de varios objetos en una imagen o la cantidad de filetes en el tornillo. Para éstos casos, se utilizaron un rango de umbrales más amplio ( $\text{minVal}=10$ ,  $\text{maxVal}=800$ ). En caso de que ese mismo rango de umbral tan grande se le aplique a una sola arandela nunca logrará encontrarla, por lo que remarca el contorno de la imagen.

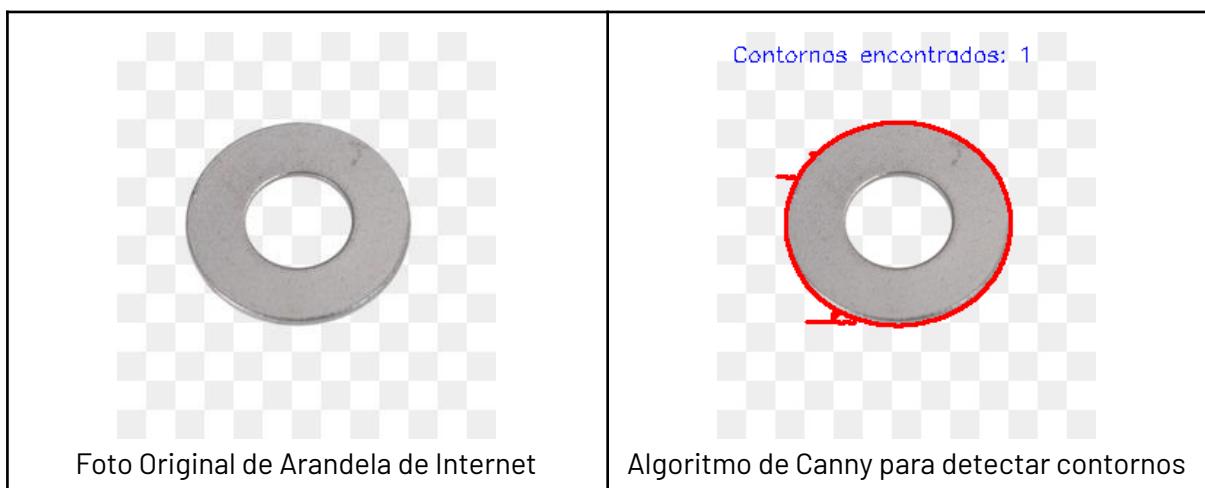


Ilustración 16: Resultados de detección de bordes para umbrales  $\text{minVal}=10$  y  $\text{maxVal}=140$ , se determinaron arbitrariamente para esta imagen.

La cantidad de contornos encontrados equivale a la cantidad de objetos dentro de la imagen. Es clave saber el número de objetos, porque el algoritmo de KNN y K-Means desarrollado, como se verá más adelante, sólo evalúa un objeto por imagen.

Conclusión: Si se prueba este algoritmo con imágenes de computadora (sin brillos, ni sombras, perfectamente planas, fondo homogéneo, etcétera) el contorno es continuo, y entra en vigencia que la cantidad de contorno equivale a la cantidad de objetos. Debido a que los bordes no son continuos, esa correspondencia no se cumple, pero se seguirá utilizando este programa, no como contador de objetos, sino que ahora con el mismo se puede detectar objetos, luego recortarlos y extraerlos de la imagen.

## 2) Uso de umbralización simple (Thresholding) para detectar contornos

No siempre con el algoritmo Canny se obtuvieron buenos resultados para las distintas piezas de ferretería, por eso también se utilizó este método: Umbralización simple (Thresholding), cuyo objetivo es separar un objeto de interés del fondo de una imagen.

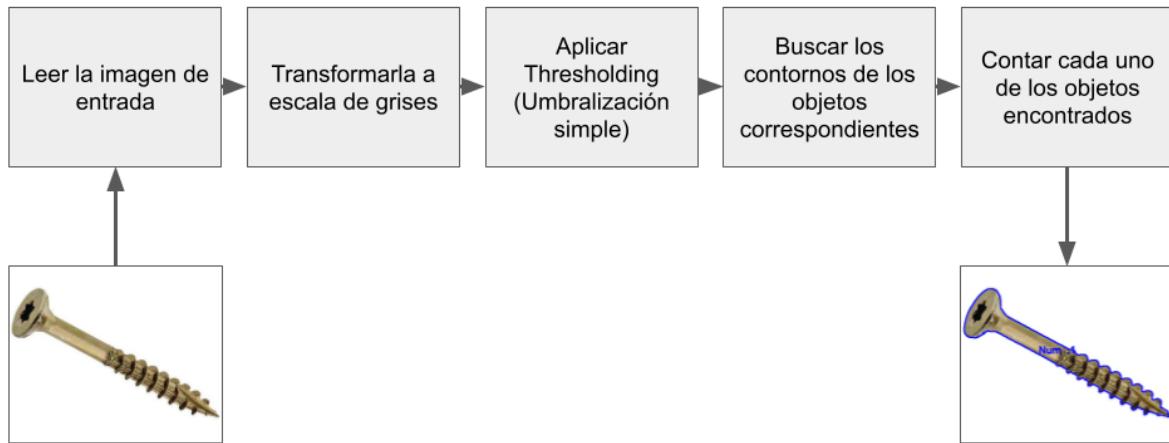


Ilustración 17: Proceso de detección de objetos con umbralización simple.

Debido a que el fondo de la imagen que se utiliza es blanco, se usa cv2.THRESH\_BINARY\_INV, para obtener el fondo negro, de este modo se obtiene un modo binario, que servirá para encontrar los contornos de la imagen. Se determinaron los momentos de la imagen con cv2.moments para luego encontrar los centroides de los mismos.

$$C_x = \frac{M_{10}}{M_{00}} \quad C_y = \frac{M_{01}}{M_{00}} \quad \text{donde } M \text{ denota los momentos.}$$

Esta forma de encontrar el contorno, es muy buena para el caso de los tornillos, ya que permite remarcar todos los filetes y demás características del tornillo, pero al utilizar la cantidad de contornos como parámetro para contar, no es muy preciso porque existen muchas discontinuidades entre filetes y cuenta de más. Para el caso de las arandelas, tiene problemas con el formato .jpg, cuenta los objetos y el fondo.

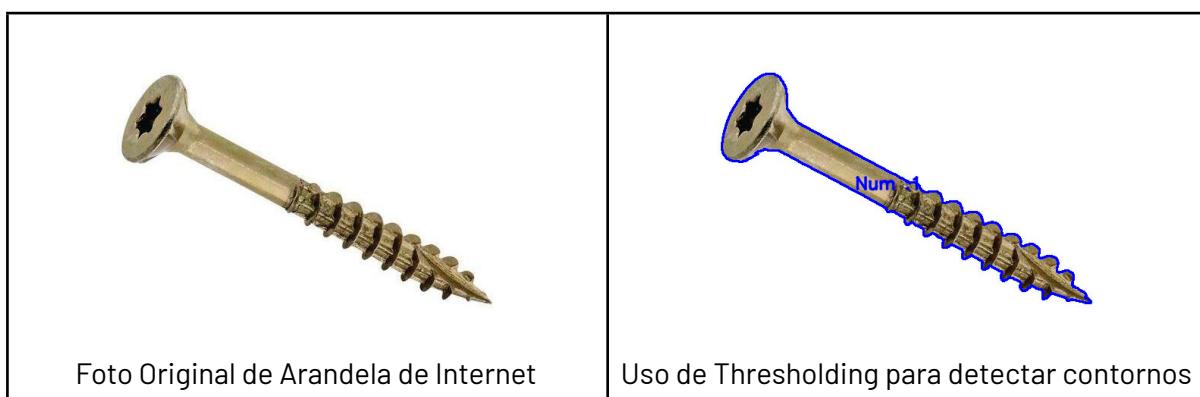


Ilustración 18: Resultados de detección de bordes para umbrales minVal=190 y maxVal=255, se determinaron arbitrariamente para esta imagen.

La conclusión de utilizar un contador de objetos a partir del número de contornos detectado en la imagen, es bastante complejo porque: Se necesitaría una imagen de entrada .jpg, con fondo blanco y uniforme, la cual fue tomada de forma perpendicular a la pieza de ferretería, y los parámetros de los programas de Adaptacion.py deberían ser definidos a priori, ya que varía para cada pieza analizada.

### 3) Recortar imagen a partir del Algoritmo de Canny

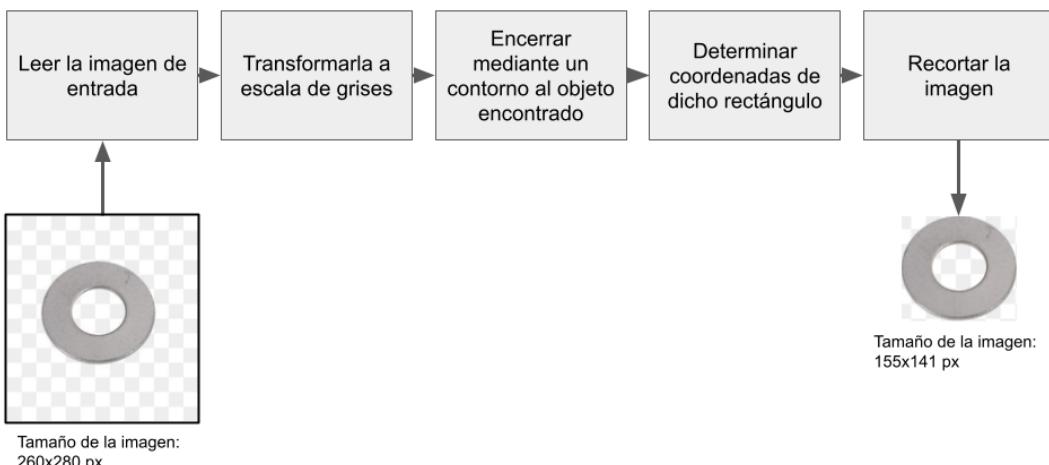


Ilustración 19: Se hace uso del algoritmo Canny para el recorte de la imagen

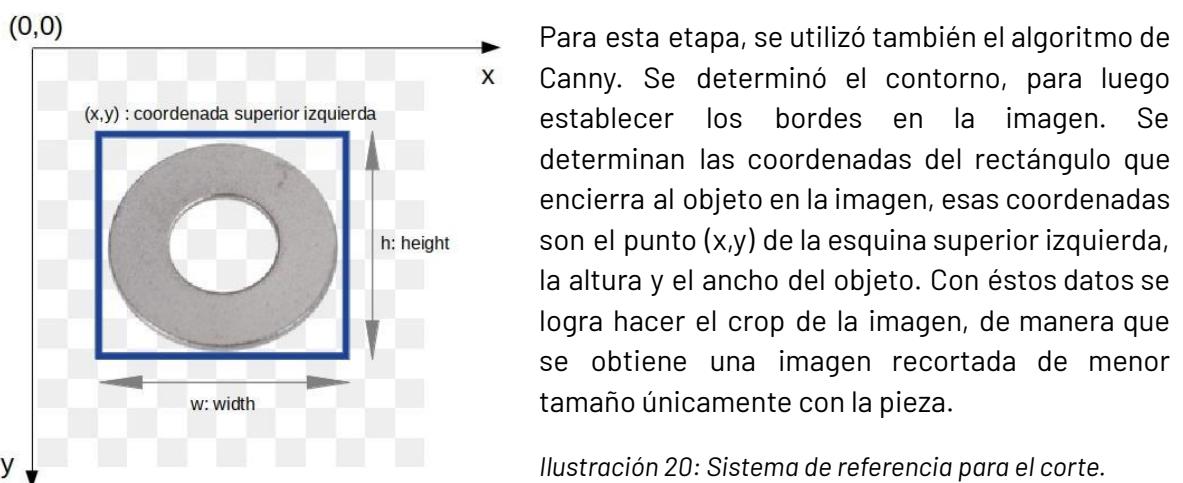


Ilustración 20: Sistema de referencia para el corte.



Ilustración 21: Resultados de detección de bordes para umbrales  $minVal=100$  y  $maxVal=200$ , se determinaron arbitrariamente para esta imagen.

Conclusión: La imagen de entrada debe tener fondo uniforme y homogéneo. Si se realiza el corte de la imagen nuevamente, se debe normalizar a 500x300px.

## **Preprocesamiento**

En el preprocesamiento se adecua la imagen para una aplicación específica, en este caso para extraer información de clavos, tornillos, arandelas o tuercas de la imagen normalizada para poder analizarla.

### **- Descomposición de la imagen en una tupla de valores R, G y B (Representación)**

Una forma de obtener información relevante para el estudio de la imagen es descomponerla en sus 3 capas RGB donde queda al final una matriz con los correspondientes colores rojo, verde y azul. Sería una tupla de tres enteros en la imagen a color o simplemente un entero para el caso de la imagen a escala de grises. Para el caso, se trabaja con imágenes de 8 bits, las cuales representan sus valores en un rango de 0 a 255 en números enteros, este rango también es conocido como Rango Dinámico, donde 255 representa el color blanco y 0 el color negro.

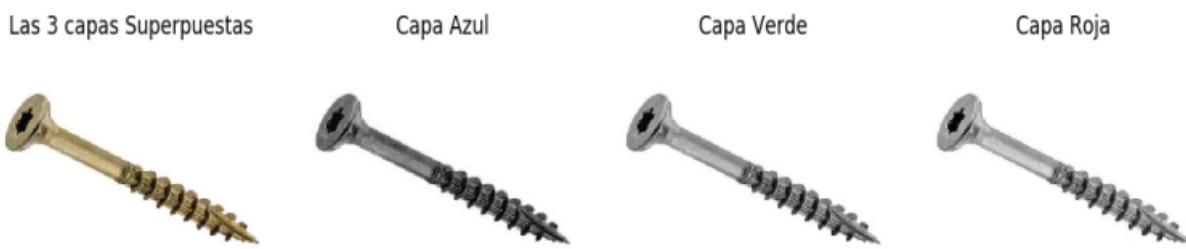


Ilustración 22: Descomposición de capas de una imagen. Se puede apreciar que las capas verde y roja tienen mayor intensidad mientras que la capa azul es más opaca y oscura.

### **- Conversión de RGB a escalas de grises utilizando librerías**

Esta conversión se realiza calculando un equivalente "E" formado a partir de los tres planos de la imagen a color. En su forma más sencilla se establece este equivalente como el promedio, es decir:

$$E(x, y) = \frac{R(x,y) + G(x,y) + B(x,y)}{3}$$

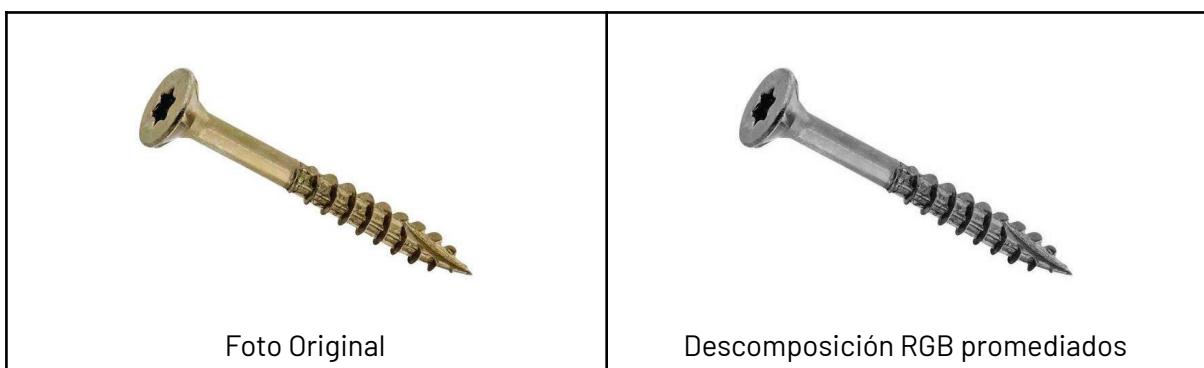


Ilustración 23: Conversión de imagen RGB a escala de grises utilizando pesos WG, WR y WB arbitrarios ingresados manualmente. Para este caso cada peso es  $\frac{1}{3}$ .

Como varía la iluminación que se tiene en las imágenes, presenta una subjetiva iluminación, propia del modelo RGB hace que utilizando el promedio de las imágenes con un valor grande en la componente roja y/o verde tengan una apariencia oscura. El efecto contrario sucede donde el contenido del plano azul es grande, mostrando en su versión a escala de grises una apariencia muy clara. Con el objetivo de solventar ésto se considera como una mejor aproximación calcular una combinación lineal de todos los planos, como ponerlo en función de sus pesos, definida como:

$$E(x, y) = WR * R(x, y) + WG * G(x, y) + WB * B(x, y)$$

Donde WR, WG y WB son los coeficientes que definen la transformación. Los cuales varían de acuerdo con el método considerado.

Por ejemplo, algunos de los valores para los coeficientes más utilizados son:

- WR=0.299 WG=0.589 WB=0.114 Método utilizado para señales a color en TV
- WR=0.2125 WG=0.7154 WB=0.072 Método ITU-BT.709 para codificación digital a color
- WR=  $\frac{1}{3}$  WG=  $\frac{1}{3}$  WB=  $\frac{1}{3}$  Se obtiene una imagen más oscura

Una consideración de importancia es la distorsión Gamma producida en las señales de TV que afecta de manera no lineal que se resuelve con los siguientes pesos: WR=0.309 WG=0.609 WB=0.082

Al hacer una escala de grises mediante la combinación lineal se obtienen imágenes más oscuras que utilizando el promedio. Para este proyecto, para evitar las variaciones en los resultados que se obtienen por la variación de los coeficientes, se utilizan los valores dados por el módulo Skimage.

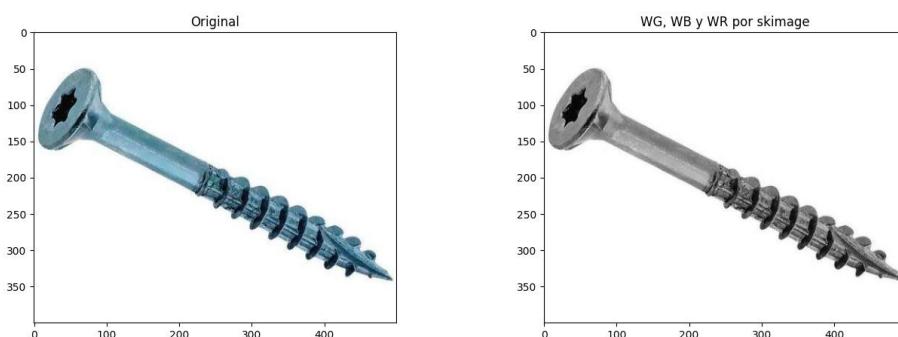


Ilustración 24: Los pesos WR, WG y WB los proporciona el módulo de skimage, lo que hace esa función es calcular la luminancia de una imagen RGB.

## Filtración

El objetivo de la filtración es suavizar la imagen, eliminar ruido, realzar la imagen y detectar bordes.

Los filtros digitales constituyen uno de los principales modos de operar en el procesamiento de imágenes digitales. Pueden usarse para distintos fines, pero en todos los casos, el resultado sobre cada píxel depende de los píxeles de su entorno. Una imagen se puede filtrar en el dominio del espacio (trabajando directamente sobre los píxeles de la imagen) o en el dominio de la frecuencia (donde las operaciones se llevan a cabo en la transformada de Fourier de la imagen).

### **- Filtros en el dominio del espacio**

Las operaciones espaciales de filtrado se definen en un entorno de vecindad del punto a transformar ( $x, y$ ).

Los filtros en el dominio del espacio pueden clasificarse en: Filtros lineales (filtros basados en máscaras de convolución, es decir, en combinación de máscaras lineales) y Filtros no lineales.

#### **1) Filtro Gaussiano (Filtro lineal)**

Este filtro se usa para desenfocar imágenes y eliminar ruido, esto con la finalidad de suavizar la imagen. Este tipo de filtrado suaviza los valores de píxeles desiguales de una imagen recortando los valores atípicos extremos.

Modelizando la función gaussiana:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-(r^2+c^2)}{2\sigma^2}\right)$$

Donde "r" es el número de filas desde el centro y "c" el número de columnas desde el centro. Al aplicarlo sobre un tornillo de nuestra base de datos, se puede modificar alguno de los parámetros (por ejemplo  $\sigma$ ) para ver cómo varían el filtrado de la imagen.

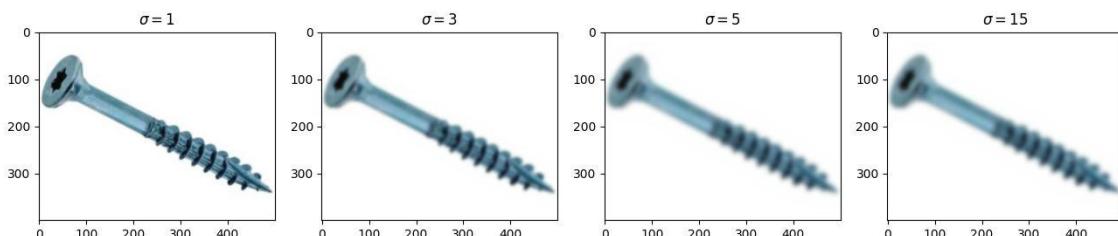


Ilustración 25: Puede observarse que si  $\sigma$  aumenta demasiado, la imagen comienza a hacerse borrosa.

#### **2) Filtro Sobel y Roberts**

El filtro Sobel detecta los bordes horizontales y verticales separadamente sobre una imagen en escala de grises. Las imágenes en color se convierten en RGB en niveles de grises. El resultado es una imagen transparente con líneas negras y algunos restos de color.

El filtro Roberts obtiene buena respuesta ante bordes diagonales, fácil y rápido de computar. Ofrece buenas prestaciones en cuanto a localización. El gran inconveniente de este operador es su extrema sensibilidad al ruido y tiene una respuesta débil a los verdaderos bordes, a menos que sean muy pronunciados, por tanto tiene pobres cualidades de detección. Para este propósito funciona mejor el operador Sobel.

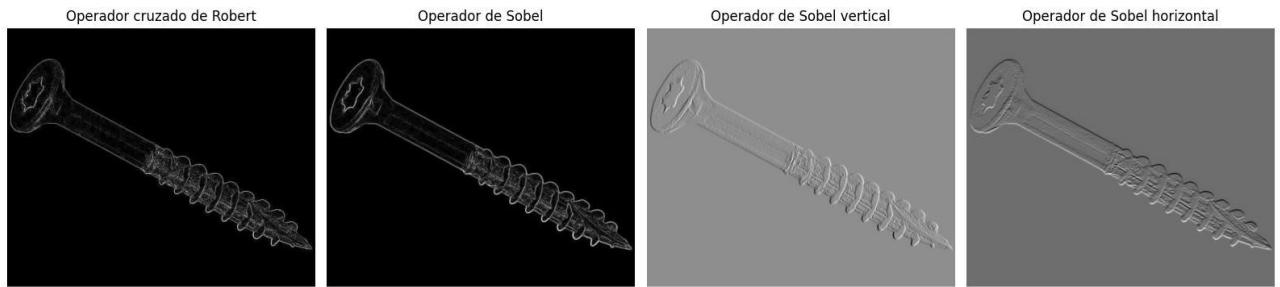


Ilustración 26: Variaciones del filtro Sobel y Robert.

Conclusión: El filtro de Sobel permite suavizar la imagen, de tal manera que se elimina un poco de ruido de la imagen si es que lo tiene, por lo consiguiente se puede desaparecer falsos bordes.

### 3) Filtro Gaussiano + Sobel

Durante el proceso de filtrado de imágenes, se pueden aplicar varios filtros a una misma imagen con el fin de obtener un mejor resultado. En este caso al aplicar el Filtro Gaussiano más el de Sobel, se obtiene un mejor resultado en comparación a aplicar únicamente uno de ellos sobre una imagen.

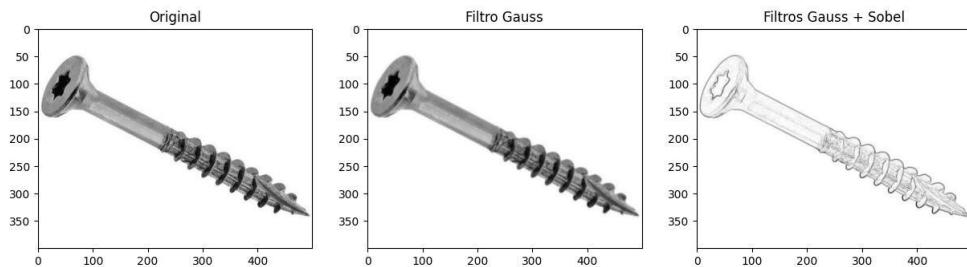


Ilustración 27: Se puede apreciar que se tiene la imagen en negativo de la imagen obtenida con el filtrado con Gauss y Sobel.

### 4) Filtro Perona Malik o Difusión Anisotrópica

Se usa para preservar los detalles de la forma original de los objetos y reducir el ruido eficientemente. Reduce el ruido en las regiones homogéneas y no en los bordes. Cada una de las imágenes resultantes de esta familia se dan como una convolución entre la imagen y el filtro gaussiano 2D isotrópico, donde el ancho del filtro aumenta con el parámetro.

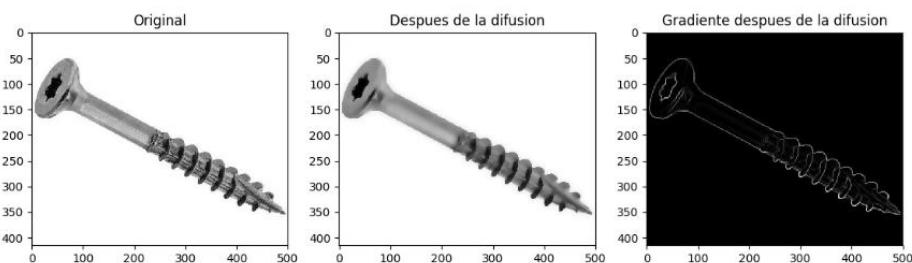


Ilustración 28: En (1) se tiene la imagen original, en (2) es la imagen después de la difusión y en (3) es el gradiente después de la difusión.

La finalidad de usar este filtro es eliminar el ruido de las imágenes digitales sin borrosidad en los bordes.

### 5) Filtro Laplace, Median, Frangi y Prewitt

El filtro Laplace detecta bordes en la imagen usando el método laplaciano, que produce bordes finos de un pixel de ancho.

El Laplaciano no se suele usar directamente en la práctica por ser muy sensible al ruido, por lo que suele ser sumado y restado (según la imagen utilizada) con la imagen original para realzar los contornos. Por eso se suele usar primero un filtro gaussiano para eliminar el ruido, lo que da lugar al filtro Laplaciano del gaussiano (LoG).

El filtro Median se suele usar para eliminar el ruido de la imagen. Consiste en visitar cada píxel de la imagen y se reemplaza por la mediana de los píxeles vecinos. La mediana se calcula ordenando los valores de los píxeles vecinos en orden y se selecciona el que queda en medio.

El filtro de Frangi se utiliza para detectar crestas continuas en una imagen (ej. arrugas, contornos, filetes del tornillo, etc) y calcular la fracción de la imagen que contiene estos elementos. Consiste en calcular los vectores hessianos para definir la similitud de una región de la imagen con el elemento buscado.

El filtro Prewitt hace uso del operador Prewitt particularmente utilizado para la detección de bordes. Se basa en hacer girar la imagen con un filtro pequeño, separable y de valores enteros en direcciones horizontales y verticales y, por lo tanto, es relativamente económico en términos de cálculos como operadores Sobel. Por otro lado, la aproximación de gradiente que produce es relativamente cruda, en particular para variaciones de alta frecuencia en la imagen. Se involucran a los vecinos de filas / columnas adyacentes para proporcionar mayor inmunidad al ruido y permite detectar los bordes verticales y horizontales respectivamente.

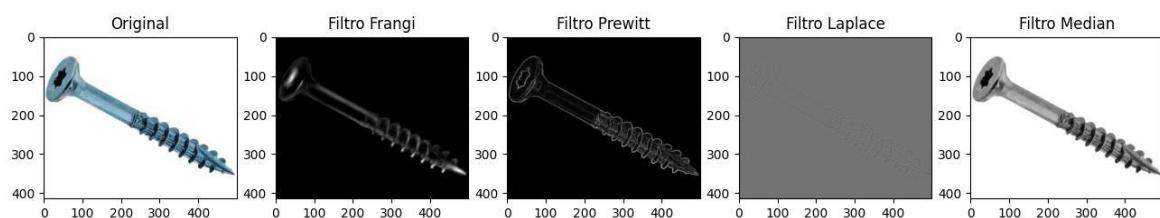


Ilustración 29: Otros filtros. Se utilizó `plt.rcParams['image.cmap'] = 'gray'` por eso las imágenes obtenidas como resultado de cada filtrado es en escala de grises, sino hubiera sido así se puede apreciar una mayor variación para cada caso.

## Segmentación

El objetivo es aislar los objetos de interés dejando una imagen más limpia de valores binarios, a la que ya se le podrá extraer un determinado tipo de características.

La segmentación puede ser supervisada o no supervisada. Por ejemplo se puede utilizar el script de LabelImage que al ejecutarlo despliega una pantalla para ir segmentando el objeto de

interés en la imagen por imagen, en este caso sería supervisado. Para este proyecto se optó por una segmentación no supervisada, ya que se trabaja con base de datos y luego la clasificación es sin intervención humana.

### **- Thresholding**

Es una técnica para dividir una imagen en dos (o más) clases de píxeles, que generalmente se denominan "primer plano" y "fondo", dependiendo del threshold.

Si  $a(x, y) \geq th$  entonces  $b(x, y) = 0$

caso contrario  $b(x, y) = 255$

Suponiendo "a" la imagen original y "b" la imagen resultante o imagen segmentada, donde 255 representa el color blanco.

#### **1) Thresholding Supervisado**

Manualmente se modifican los valores de "th" para ver las variaciones en la imagen segmentada.

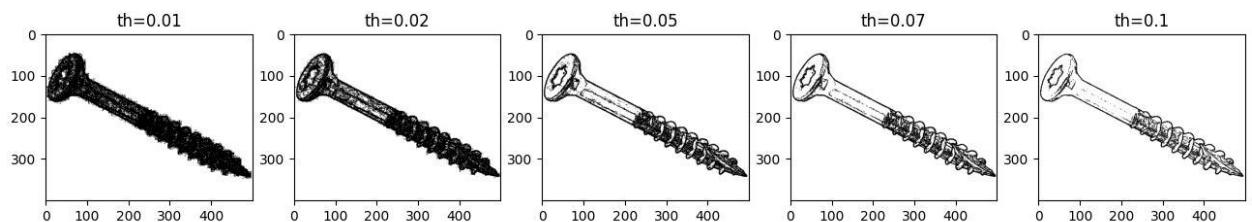


Ilustración 30: Diferentes valores de th.

#### **2) Thresholding No Supervisado**

Se utilizó el módulo de skimage para observar los distintos métodos no supervisados.

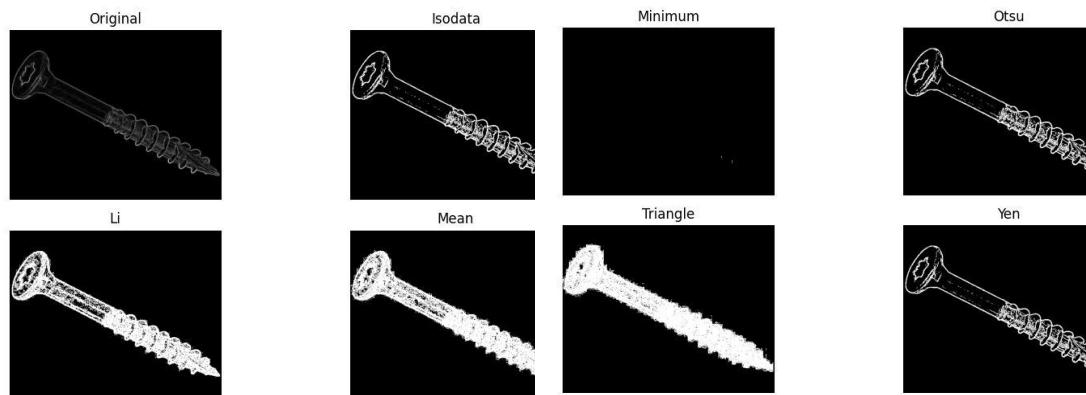


Ilustración 31: Se muestran los distintos métodos de thresholding que permite el módulo skimage.

El método global de segmentación por valor umbral es muy sensible a las variaciones en la luminosidad de la imagen. Una solución alternativa es homogeneizar la luminosidad durante el preprocesamiento de la imagen, por ejemplo realizando una corrección de sombras o por medio de una imagen de referencia que nos permita homogeneizar la luminosidad. Eso es lo que se hizo en la etapa de transformación porque se presentaba

este problema, por eso se decidió enmascarar la imagen y dejar un fondo totalmente blanco.

Además del problema de la luminosidad, pueden aparecer otros problemas que se pueden reducir tratando la imagen antes de segmentar. A menudo se utilizan técnicas de reducción de la borrosidad o de incremento de la nitidez de los bordes. Los métodos del valor umbral siempre utilizan información unidimensional de la imagen (normalmente un valor de intensidad o un valor de gris). No se tienen en cuenta otras informaciones, como por ejemplo los diferentes colores.

## Extracción de Rasgos

El objetivo es describir cuantitativamente la naturaleza de los objetos. El objetivo es tratar de escoger los parámetros representativos de cada uno de los objetos con el fin de dar la mejor descripción de cada uno de ellos, simplificar la imagen extrayendo información útil y desechar información extraña.

### **- Histograma de gradientes orientados (HOG)**

Es un tipo de descriptor de características, que mediante un proceso se obtienen ciertos valores de una misma imagen que puede describir a la misma. El tornillo, clavo, tuerca o arandela quedaría definida por esos valores. La idea esencial es que la apariencia y forma de un objeto local dentro de una imagen pueda ser descrita por la intensidad de la distribución de la dirección de los gradientes.

El objetivo del HOG es ser un descriptor de características que pueda generalizar la pieza de tal manera que para la misma pieza, ante distintas condiciones, se pueda obtener un mismo valor constante

Se realizó un análisis para 3 tornillos distintos.

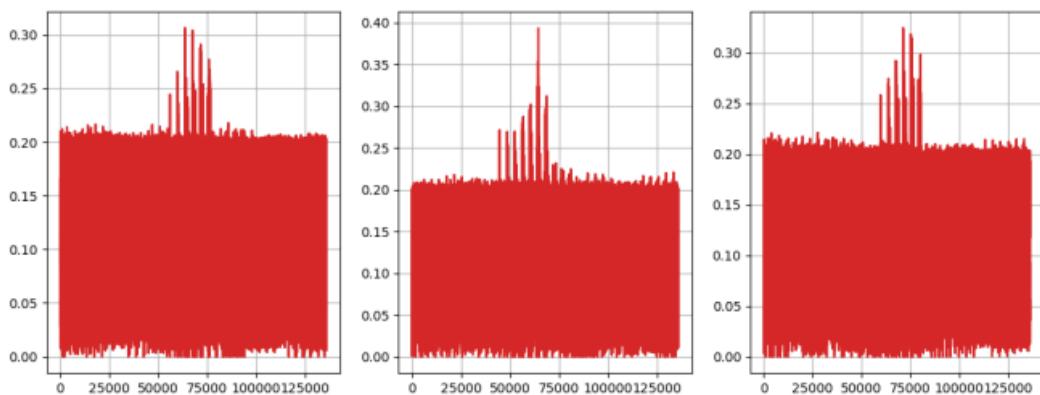


Ilustración 32: HOG para tres tornillos “iguales” ante las mismas condiciones.

El HOG permite comparar elementos “iguales” al comparar para cada uno de ellos los gradientes de luminosidad bajo las mismas condiciones. Sin embargo, en el gráfico anterior se observa qué hay múltiples variaciones por más que sea el mismo tipo de pieza en iguales condiciones. De manera que se realizó un análisis para un tornillo, una tuerca, una arandela y un clavo, con el

fin de determinar algún patrón para cada una de las piezas. En ese mismo orden se presenta en el gráfico siguiente.

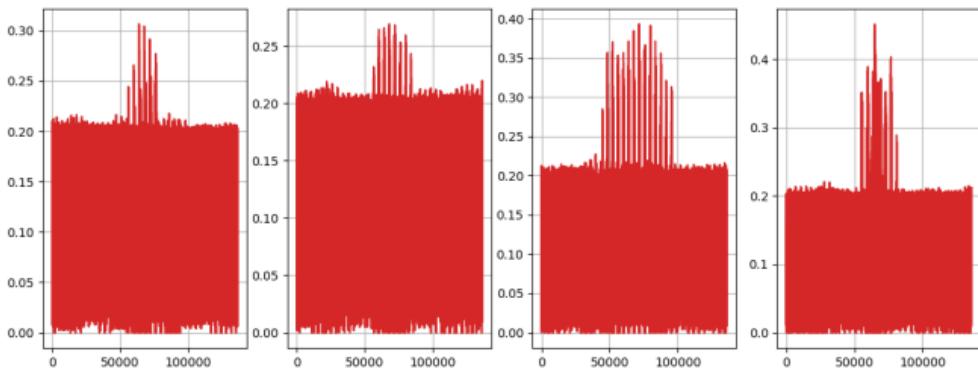


Ilustración 33: HOG para las piezas de ferretería de estudio.

Conclusión: Se utiliza mucho con el propósito de detección de bordes. Sin embargo en esta implementación tiene un alto costo computacional y temporal en comparación a los otros extractores de características. No se consideró para la implementación en el código final.

### **- Momentos de Hu**

La forma característica de un objeto se puede cuantificar mediante sus momentos de Hu, los cuales describen la distribución de los píxeles sobre el plano. Son un promedio ponderado de las intensidades de píxeles de la imagen. Los momentos deben ser invariantes a las transformaciones geométricas que puede sufrir los objetos y discriminantes para objetos con formas distintas.

Los momentos de Hu son siete descriptores invariantes que cuantifican la forma de un objeto. Los primeros 6 momentos de Hu son invariantes a la transformación, rotación y escala; mientras que el 7mo cambia para el reflejo de la imagen. Muchas veces por fines prácticos se toman los primeros 2 momentos.

$$\text{Rasgo 1: } \theta_1 = \eta_{20} + \eta_{02}$$

$$\text{Rasgo 2: } \theta_2 = (\eta_{20} - \eta_{02})^2 + 4 * \eta_{11}^2$$

donde los momento centrales normalizados de segundo orden se calculan como:

$$\eta_{20} = \frac{m_{20} - \frac{m_{10}^2}{m_{00}}}{m_{00}^2}, \quad \eta_{02} = \frac{m_{02} - \frac{m_{01}^2}{m_{00}}}{m_{00}^2}, \quad \eta_{11} = \frac{m_{11} - \frac{m_{10} * m_{11}}{m_{00}}}{m_{00}^2}$$

y los momentos geométricos (éstos son la base de los momentos de Hu) de orden  $p+q$  se calculan como:

$$m_{pq} = \sum_x \sum_y x^p * y^q * I(x, y)$$

Existe un trabajo de J. Flusser que plantea una teoría general sobre conjuntos completos e independientes de invariantes rotacionales derivados de momentos. Demostró que el conjunto invariantes de Hu no es ni completo ni independiente: por un lado el 3er momento es redundante, pues es dependiente de otros, y por otro falta un invariante, propuesto por Flusser

y denominado "octavo invariante de Hu". Por lo qué se suele solo considerar los momentos de Hu 1,2 y 4.

$$\text{Rasgo 3: } \theta_3 = (\eta_{30} - 3 * \eta_{12})^2 + (3 * \eta_{21} - \eta_{03})^2$$

$$\text{Rasgo 4: } \theta_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

Para este proyecto se consideró utilizar el 1er momento de Hu porque es análogo, al momento de inercia alrededor del centroide, donde las intensidades de los píxeles son análogas a densidad física. Sin embargo se hicieron pruebas de rendimiento y de agrupación de objetos con el 2do y 4to momento de Hu, sin embargo se observó una dispersión de los valores en comparación a usar el 1er momento de Hu. Para este proyecto se usa una librería que calcula directamente los momentos de Hu.

Conclusión: Todas las configuraciones consideradas acá, se evaluarán con el programa p8\_rendimiento.py, en función del mejor porcentaje de predicciones definirá qué configuración se usará.

### **- Histograma de color**

En las imágenes digitales, un histograma de color representa el número de píxeles que tienen colores en cada una de las listas fijas de rangos de colores, que se extienden sobre el espacio de color de la imagen, es decir, el conjunto de todos los posibles colores.

A continuación se presenta una imagen filtrada en escala de grises, donde se encuentra los picos es aproximadamente donde se encuentran los contornos. Este histograma muestra la distribución de las intensidades de una imagen en escala de grises.

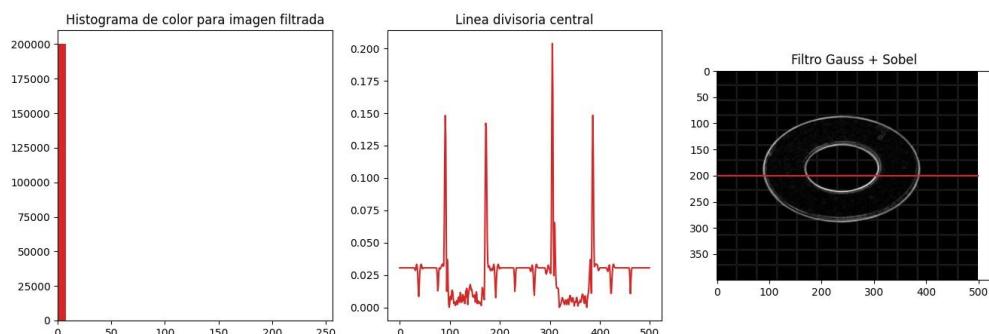


Ilustración 34: Se muestra un histograma de color con abcisas bins y ordenadas número de píxeles.

Conclusión: Se muestra una extracción de características por color, pero debido a que usamos piezas de acero con colores similares, no se consideró para la implementación en el código final.

### **- Eje Mayor y Eje Menor**

En los objetos de estudio se observa que poseen distintas longitudes, anchura, entre otras particularidades características de cada pieza. Este tipo de propiedades se pueden determinar mediante las propiedades de regiones, las cuales otorgan información sobre los objetos de una imagen, tales área, perímetro, solidez, diámetro equivalente, entre otras.

Para el caso, algunas de las propiedades características de las piezas son la longitud y la anchura, las cuales se pueden determinar por el eje menor y eje mayor del elipse centrado en el rectángulo que encierra al objeto en la imagen de la pieza.

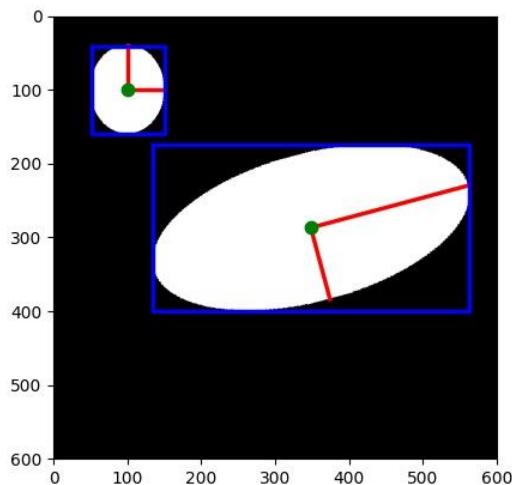


Ilustración 35: Ejemplificación del eje menor y eje mayor de las propiedades de figura de una imagen.

El cálculo de estos parámetros se pueden ver alterados de acuerdo a la orientación de la pieza en la imagen, por lo que se normaliza la rotación respecto al plano en la que se debe poner las piezas para la captación de imágenes para evitar falsas predicciones.

Conclusión: Se consideró en la implementación el eje menor y eje mayor de cada pieza, ya que se observa una variación entre cada una de las piezas a analizar.

### **- Excentricidad**

La excentricidad es la relación entre anchura y longitud del objeto. Para el caso, como se tienen piezas de distintas longitudes y formas, se tiene un aspecto sobresaliente de cada uno que puede servir para la identificación de cada una de las piezas.



Ilustración 36: Cálculo de excentricidad a una de las piezas de estudio.

La excentricidad será cero para un círculo y tenderá a la unidad a medida que el objeto se alarga. Su valor es discriminante para formas diferentes e invariante a transformaciones geométricas.

La fórmula para calcular la excentricidad es:

$$\varepsilon = \frac{(\mu_{20} - \mu_{02})^2 + 4 * \mu_{11}^2}{(\mu_{20} + \mu_{02})^2}$$

donde  $\mu$  son los momentos centrales. Para facilitar los cálculos se utilizará una librería que calcula este valor.

Por lo tanto las características a analizar, en términos teóricos (aún falta determinar si son variables aleatorias entre piezas “iguales”), serán: Momento de Hu 1, Excentricidad, Eje Menor y Eje Mayor.

## **Reducción de Dimensionalidad**

Es el proceso de reducción del número de variables aleatorias que se trate, y se puede dividir en selección de la función y extracción de la función.

Un inconveniente que se presentó al momento de realizar el proyecto fue que debido a las variaciones en las dimensiones de la imagen (magnitud de la dimensión de los vectores de características) afectando los gráficos anteriores.

Se puede utilizar estadísticos como la media aritmética y desviación estándar para los distintos vectores que describen su magnitud y dispersión, esto con la finalidad de determinar la frecuencia de aparición de cada medida. Un buen resultado se representa por una secuencia de puntos de una misma pieza que estén alineados de manera vertical.

Se procedió a analizar las propiedades del Histograma de Gradientes Orientados, Momentos de Hu 1, Eje Menor, Eje Mayor y Excentricidad.

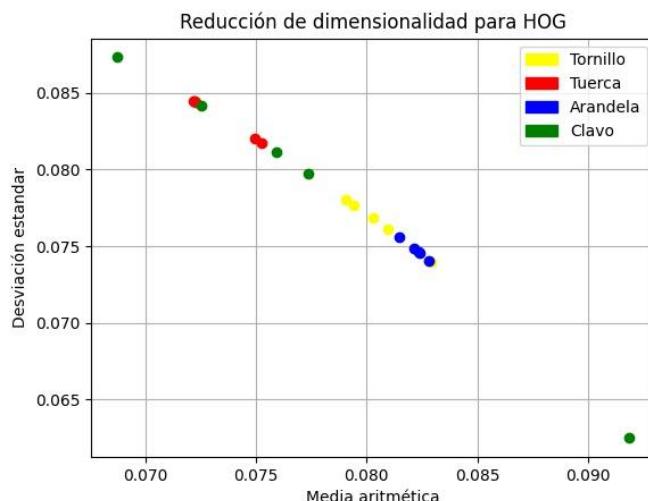


Ilustración 37: Reducción de dimensionalidad para el Histograma de Gradientes Orientados, para este gráfico se contó con una base de datos de 80 fotos. Cada pieza de ferretería está formada por 20 fotos cada uno.

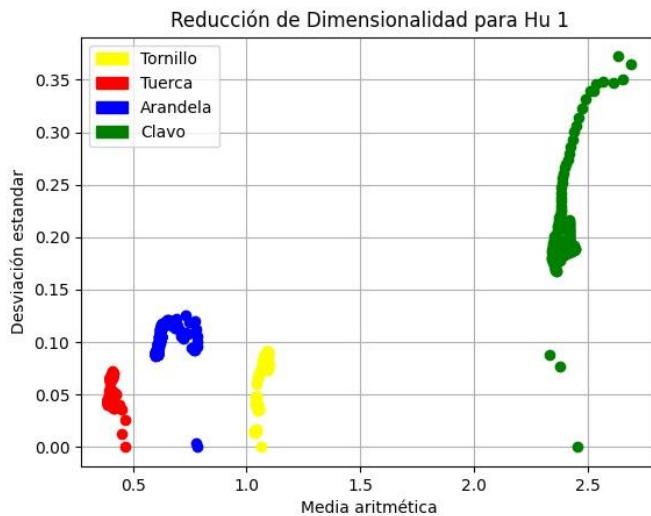


Ilustración 38: Reducción de dimensionalidad para el Momentos de Hu, para este gráfico se contó con una base de datos de 804 fotos. Cada pieza de ferretería está formada por 201 fotos cada uno

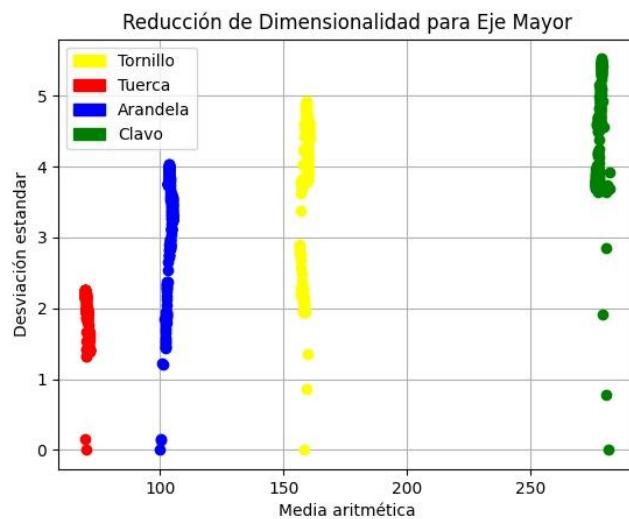


Ilustración 39: Reducción de dimensionalidad para el Eje Mayor, para este gráfico se contó con una base de datos de 804 fotos. Cada pieza de ferretería está formada por 201 fotos cada uno

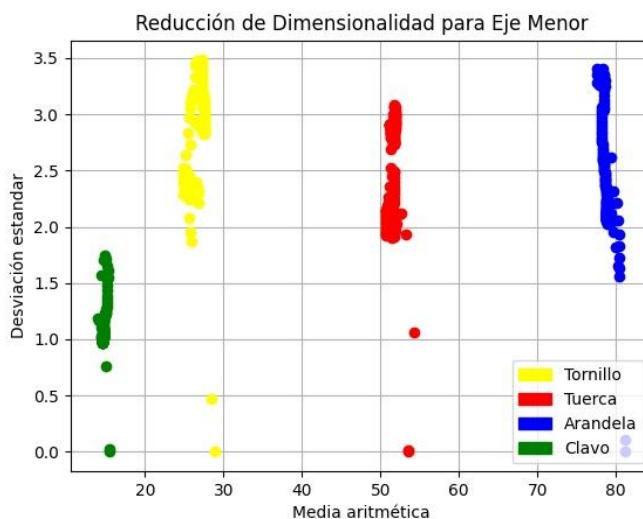
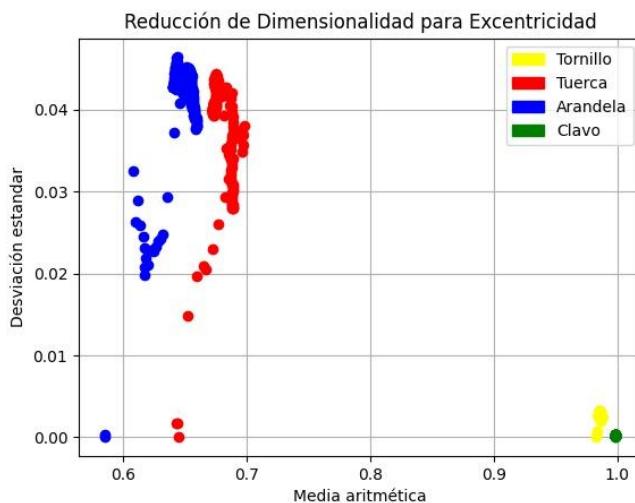


Ilustración 40: Reducción de dimensionalidad para el Eje Menor, para este gráfico se contó con una base de datos de 804 fotos. Cada pieza de ferretería está formada por 201 fotos cada uno



*Ilustración 41: Reducción de dimensionalidad para la Excentricidad, para este gráfico se contó con una base de datos de 804 fotos. Cada pieza de ferretería está formada por 201 fotos cada uno*

Conclusión: Mediante la reducción de dimensionalidad se logró una mejor forma de visualización del eje Menor, eje Mayor y momentos de Hu 1, pero no para el Histograma de Gradientes Orientados (HOG) y Excentricidad en este caso.

De manera que los rasgos característicos a extraer de cada una de las imágenes será eje Menor, eje Mayor y momentos de Hu 1.

## **Planificación con lenguaje STRIPS**

Planificar es encontrar una secuencia de acciones que alcanzan un determinado objetivo si se ejecutan desde un determinado estado inicial.

Para poder representar un plan, que es una secuencia de acciones que consiguen el objetivo, se emplea el lenguaje STRIPS para representar este tipo de planificación.

### **- Elementos que Intervienen**

Para nuestra situación, tendremos los siguientes elementos qué intervienen para llegar al objetivo:

- Una superficie plana, delimitada por 4 sectores (mesa A, mesa B, mesa C, mesa D).
- Una serie de 4 cajas identificados qué contienen las piezas de ferretería (Clavo, Arandela, Tornillo, Tuerca). Cada una puede estar sobre la mesa o apilada sobre otra caja.
- Un brazo robotizado, que puede sujetar una caja a la vez.

Para determinar el orden de apilamiento de las cajas, se extrae una pieza de cada caja en representación de toda la muestra. Posterior a ello se ingresan al sistema de captación de imagen para tomar fotografías que posteriormente se ingresan al algoritmo que posee el agente.

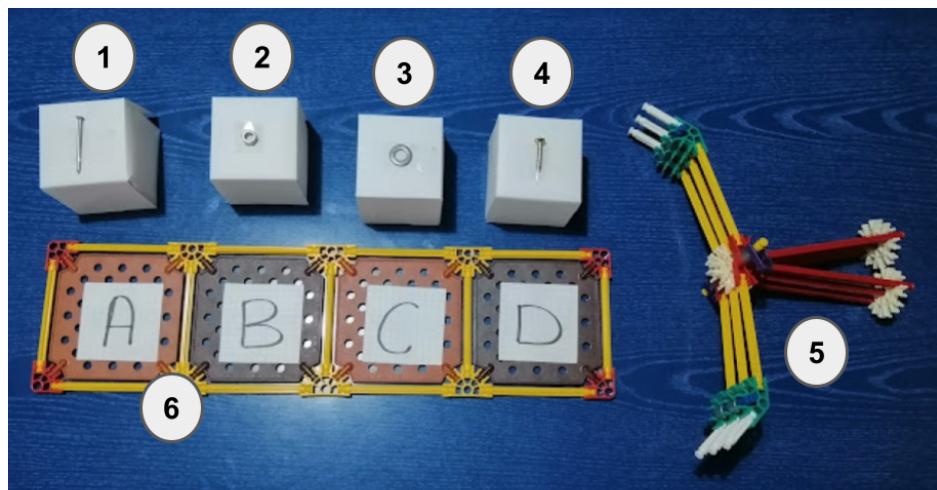


Ilustración 42: Representación de los elementos que intervienen.

(1) Caja con Clavos (2) Caja con Tuercas (3) Caja con Arandelas (4) Caja con Tornillos (5) Representación del Brazo robotizado (6) Superficie plana delimitada por 4 sectores.

### **- Representación de Estados**

Es una descripción de la situación del “mundo” en términos de condiciones lógicas, simples y sin dependencias funcionales. Inicialmente tendremos los 4 bloques apilados en un orden aleatorio, donde para cada uno podremos tener los siguientes estados:

- Sobre (x,y)= El bloque x está sobre el bloque y.
- Sobre (x, mesa A)= El bloque x está sobre la mesa en la posición A
- Libre (x)= El bloque x está despejado.
- Libre (Brazo)= El brazo no agarra ningún bloque.
- Caja (x)= La existencia de una caja con piezas x.

Una vez conocido el estado inicial de cada una de las componentes, podremos empezar a realizar las acciones necesarias para llegar al estado objetivo.

### **- Estado Objetivo**

Un estado satisface un objetivo si es posible sustituir las variables del objetivo por objetos del mundo de manera que sus literales están incluidos en la descripción del estado, donde un estado es estado final si satisface el objetivo requerido. Para el caso, el estado objetivo es alcanzar un nuevo orden de apilamiento distinto al inicial.

### **- Operadores**

Los operadores son los que modifican el estado del “mundo”. Para intentar solucionar el problema del marco, sólo se especifica lo que cambia por la acción del operador. Para cada operador hay precondiciones, efectos y reglas para qué se puedan ejecutar. Las acciones planteadas para llegar al estado objetivo son: Apilar, Desapilar, Sujetar y Soltar. A continuación se detalla cada uno de los operadores:

**Apilar(x,y)** { #Apilar una caja sobre otra  
 PRECOND: Libre(y), Sujetar(x), Caja(x), Caja(y), ( $x \neq y$ )  
 EFECTO: Libre(Brazo), Sobre(x,y), Libre(x),  $\sim$ Libre(y)  
 }

**Desapilar(x,y)** { #Quitar una caja que estaba sobre otra

PRECOND: Sobre(x,y), Caja(x), Caja(y), Libre(x), Libre(Brazo), (x ≠ y)

EFFECTO: Sujetar(x), Libre(y), ~Sobre(x,y), ~Libre(Brazo)

}

**Sujetar(x)** { #Sujetar una caja

PRECOND: Libre(x), Libre(Brazo), Caja(x)

EFFECTO: Sujetar(x), ~Libre(Brazo)

}

**Soltar(x,mesa)** { #Soltar una caja en la Mesa

PRECOND: Sujetar(x), ~Libre(Brazo), Caja(x)

EFFECTO: Sobre(x,mesa), ~Sujetar(x), Libre(x), Libre(Brazo)

}

## Búsqueda A Estrella (A\*)

El Algoritmo A estrella (A\*) es una de las estrategias más conocidas, se clasifica dentro de los algoritmos de búsqueda con información y es mayormente utilizado para búsquedas sobre grafos. El algoritmo encuentra siempre y el camino de menor costo entre un nodo origen y uno objetivo siempre y cuando exista.

Lo que realiza el algoritmo es construir distintas rutas desde un punto inicial hasta encontrar alguna que llegue hasta el nodo final. De este modo solo construye aquellas rutas que son candidatas a formar una solución.

Para poder determinar qué rutas son las que tienen mayor probabilidad de llegar al nodo meta, se llega a una fórmula que es considerada lo más importante en este algoritmo de búsqueda heurística que es la siguiente:

$$f(n) = g(n) + h(n)$$

En donde:  $f(n)$ = es el costo total del nodo.

$g(n)$ = es la distancia entre el nodo actual y el nodo inicial

$h(n)$ = es la función heurística. Representa el costo estimado del mejor camino.

El modo de realizar el cálculo de la distancia necesaria para llegar a la meta depende del tipo de movimientos permitidos. En nuestro caso, solo podemos movernos de manera vertical y horizontal. Por otro lado, en nuestro caso aplicaremos como heurística la distancia Manhattan, qué es la distancia en línea recta hasta el objetivo, para el caso no se aplica la raíz cuadrada durante este cálculo.

La complejidad del desarrollo del algoritmo A\* está relacionado directamente a que tan buena sea la heurística que se proporciona, también sabemos que existen heurísticas muy bien definidas y otras no tanto, por ende esto es lo principal para determinar la complejidad que puede llegar a ser exponencial si este factor es de dudosa calidad. En cuanto al espacio requerido para ejecutarse se trata de un espacio exponencial respecto al tamaño del problema, ya que en un problema con muchos nodos esta se vuelve una tarea ardua al tener que considerar todos los sucesores con sus respectivos costos.

Inicialmente se tiene un mapa preestablecido donde se aplicará el algoritmo A\*, de manera que se conoce el espacio y limitaciones en dimensiones, con el fin de que siempre se pueda devolver una solución válida para una ubicación dentro del laberinto.

## **Interfaz Gráfica de Usuario (GUI)**

El desarrollo de este proyecto está destinado para un usuario con poco conocimiento en programación, de manera que se diseñó una interfaz de simple uso e intuitiva para qué sea fácil de usar por cualquier persona. Para ello se utilizó Gradio, una API que permite el diseño de interfaces web amigables para cualquier persona, de manera que el usuario solo ingrese los inputs indicados para cada etapa del proceso, sin intervenir en la cmd y/o código. Se diseñó una GUI para cada una de las 3 etapas consideradas en la solución al problema.

Además se puede generar automáticamente un enlace público que se puede compartir con demás usuarios, de tal forma que le permitiría interactuar con el modelo desde otra computadora de forma remota. Por otro lado se tiene la posibilidad de alojar el proyecto de manera permanentemente en Hugging Face, comunidad de Inteligencia Artificial de Código abierto en Machine Learning.

Del mismo modo, para facilitar el uso de cada GUI se ha añadido la descripción del desafío a resolver con cada uno de los algoritmos, así como ejemplos precargados y demás información necesaria para el usuario.

**Input:** Pieza representativa de cada una de las cajas, según el orden en que están apiladas.

**Output:** Identificación del contenido de cada una de las cajas y orden de apilamiento ascendente.

**Orden de apilamiento:** Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1,Mesa)

**Foto - Caja 4**  
 Coloque la imagen aquí  
 - o -  
 Haga click para cargar

**Foto - Caja 3**  
 Coloque la imagen aquí  
 - o -  
 Haga click para cargar

**Foto - Caja 2**  
 Coloque la imagen aquí  
 - o -  
 Haga click para cargar

**Foto - Caja 1**  
 Coloque la imagen aquí  
 - o -  
 Haga click para cargar

Resultado del Análisis - Caja 4

Resultado del Análisis - Caja 3

Resultado del Análisis - Caja 2

Resultado del Análisis - Caja 1

Orden ascendente de Apilamiento

[Avisar](#)

[Limpiar](#)

[Enviar](#)

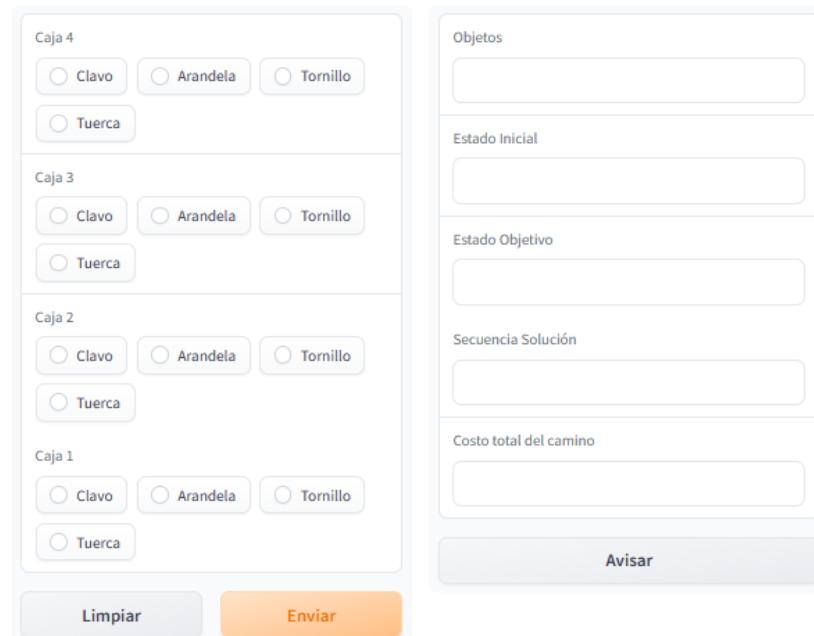
Ilustración 43: GUI para el Procesamiento de Imágenes

## Apilamiento

**Input:** Orden de Apilamiento Objetivo.

**Output:** Propiedades del Problema, Secuencia en Lenguaje STRIPS , Cantidad de Movimientos

**Orden de apilamiento:** Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)



The screenshot shows a user interface for planning a stacking problem. On the left, there are four sections labeled 'Caja 4', 'Caja 3', 'Caja 2', and 'Caja 1'. Each section contains three radio buttons for 'Clavo', 'Arandela', and 'Tornillo', and one for 'Tuerca'. To the right, there is a vertical stack of five fields: 'Objetos' (empty), 'Estado Inicial' (empty), 'Estado Objetivo' (empty), 'Secuencia Solución' (empty), and 'Costo total del camino' (empty). At the bottom are two buttons: 'Limpiar' (Grey) and 'Enviar' (Orange).

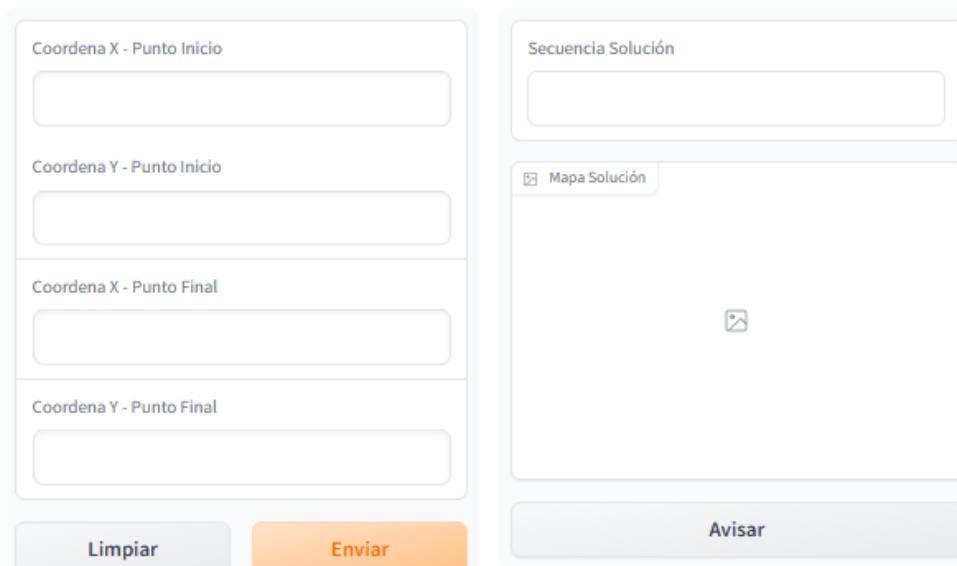
Ilustración 44: GUI para planificación de apilamiento de cajas con lenguaje STRIPS

## Resolución de Laberinto

**Input:** Ingresar coordenadas de Inicio y Fin.

**Output:** Path , Mapa con la solución

**Características:** El Laberinto tiene dimensiones: 11x11, siendo la esquina superior izquierda la coordenada (0,0)



The screenshot shows a user interface for solving a labyrinth. On the left, there are four input fields: 'Coordenada X - Punto Inicio' (empty), 'Coordenada Y - Punto Inicio' (empty), 'Coordenada X - Punto Final' (empty), and 'Coordenada Y - Punto Final' (empty). To the right, there is a large field labeled 'Secuencia Solución' (empty) and a checkbox 'Mapa Solución' (unchecked). Below these are two buttons: 'Limpiar' (Grey) and 'Enviar' (Orange). At the bottom right is a button 'Avisar' (Grey).

Ilustración 45: GUI para Transportar las cajas desde un punto A a un punto B

## **CONTEXTUALIZACIÓN**

### **Base de Datos**

Se debe obtener la etiqueta o “label” para cada pieza de ferretería. Para clasificarlo de una mejor manera se crearon las carpetas “Arandelas”, “Clavos”, “Tornillos” y “Tuercas”, que contienen fotografías de cada uno de los elementos que se van a categorizar.

### **- Overfitting y underfitting**

Son las principales causas al obtener malos resultados en los análisis. Cuando se entrena los modelos se intenta “hacerlos encajar” (en inglés fit), los datos de entrada entre ellos y con la salida. Se puede traducir overfitting como sobreajuste y underfitting como subajuste, ambos hacen referencia al fallo de nuestro modelo al generalizar y/o encajar el conocimiento que se pretende que adquieran.

- Un modelo con underfitting es aquel en donde los errores tanto de entrenamiento como de validación son similares y relativamente altos.
- Un modelo con overfitting es aquel en donde se obtiene un error de entrenamiento relativamente bajo y uno de validación relativamente alto.

Se debe encontrar un punto medio en el aprendizaje de nuestro modelo en el que no estemos incurriendo en underfitting y tampoco en overfitting. Para reconocer este problema se debe subdividir el conjunto de datos de entrada para entrenamiento en dos: uno para entrenamiento y otro para la Test que el modelo no conocerá de antemano. Esta división se suele hacer del 80% para entrenar y 20% para Test. El conjunto de imágenes Test deberá tener muestras diversas en lo posible y una cantidad de muestras suficiente para poder comprobar los resultados una vez entrenado el modelo.

Si el modelo entrenado con el conjunto de entrenamiento tiene un 90% de aciertos y con el conjunto de test tiene un porcentaje muy bajo, esto señala claramente un problema de overfitting. Si en el conjunto de Test sólo se acierta un tipo de clase (por ejemplo tornillos) o el único resultado que se obtiene es siempre el mismo valor será que se produjo un problema de underfitting.

Se tomaron 908 imágenes en total. Se destinó un 89% de imágenes para el entrenamiento y el 11% para el test. Cada pieza de ferretería cuenta con 26 imágenes para el Test y 201 imágenes para el entrenamiento, siendo un total de 104 imágenes para el Test y 804 imágenes para el entrenamiento

### **KNN**

K-Nearest Neighbors (KNN), también llamado el vecino más cercano, puede ser utilizado para clasificación y problemas de regresión predictiva, en la industria es más utilizado para problemas de clasificación, donde la mayoría de los conjuntos de datos del mundo real no siguen supuestos teóricos matemáticos. Trata de buscar los K puntos más cercanos a un punto concreto para poder inferir su valor. Este algoritmo pertenece al conjunto de técnicas del aprendizaje automático supervisado, es decir, un algoritmo que recibe un conjunto de datos que están etiquetados con los valores de salida correspondientes sobre los que puede entrenarse y definir un modelo de predicción.

Implementación: Facilidad para interpretar las salidas (predicciones), bajo tiempo de cálculo y el poder de predicción.

### **- Pasos del algoritmo KNN**

- 1) Medir la distancia euclídea entre el patrón que se desea reconocer y todas las muestras del conjunto de entrenamiento.
- 2) Identificar los K patrones de entrenamiento más cercanos al patrón que se desea reconocer (i.e., aquellos que obtuvieron menor distancia euclídea) y obtener la etiqueta de clase de cada vecino.
- 3) El patrón que se desea reconocer se asigna a la clase que obtuvo el mayor número de concurrencias o votos.

### **- ¿Cómo se decide el número de vecinos de KNN?**

El número de vecinos (K) en KNN, es un hiper-parámetro que debe elegir en el momento de la construcción del modelo. Se puede pensar en K como una variable de control para el modelo de predicción.

En el caso de una pequeña cantidad de vecinos, el ruido tendrá una mayor influencia en el resultado, y una gran cantidad de vecinos lo hará computacionalmente costoso. También una pequeña cantidad de vecinos son más flexibles, tendrán un sesgo bajo pero una varianza alta y un gran número de vecinos tendrá un límite de decisión más uniforme, lo que significa una varianza más baja pero un sesgo más alto. Osea KNN funciona mejor con un menor número de características que un gran número de características.

Se recomienda un número impar si el número de clases es par. También se puede verificar generando el modelo en diferentes valores de K y verificar su rendimiento.

### **- Maldición de la Dimensionalidad**

El aumento de la dimensión también conduce al problema del sobreajuste. Para evitar el sobreajuste, los datos necesarios deberán crecer exponencialmente a medida que aumente el número de dimensiones y se ha corroborado que en grandes dimensiones, la distancia euclídea ya no es útil. Este problema de dimensión superior se conoce como la Maldición de la Dimensionalidad. Esto ocurría cuando se utilizaban todos los 7 momentos de Hu en el algoritmo con la base de datos relativamente pequeña que se tenía.

### **- Ventajas y desventajas**

La fase de entrenamiento de la clasificación de vecino K más cercano es mucho más rápida en comparación con otros algoritmos de clasificación. No hay necesidad de entrenar un modelo para la generalización, es por eso que KNN se conoce como el "Algoritmo de aprendizaje simple y basado en instancias". El valor de salida para el objeto se calcula por el promedio del valor de K vecinos más cercanos.

La fase de prueba de la clasificación de vecino K más cercano es más lenta y costosa en términos de tiempo y memoria. Requiere una gran memoria para almacenar todo el conjunto de datos de entrenamiento para la predicción. KNN requiere escalado de datos porque usa la distancia euclídea entre dos puntos de datos para encontrar vecinos más cercanos. La distancia euclídea es sensible a las magnitudes. Las características con altas magnitudes

pesarán más que las características con bajas magnitudes. KNN tampoco es adecuado para datos de grandes dimensiones.

### - ¿Cómo mejorar el KNN?

Para obtener mejores resultados, se recomienda encarecidamente normalizar los datos en la misma escala, es lo que se hace en la etapa de Transformación. Generalmente, el rango de normalización está considerado entre 0 y 1. Este algoritmo no es adecuado para los datos de grandes dimensiones. En tales casos, la dimensión debe reducirse para mejorar el rendimiento, como se aprecia en los gráficos de la etapa de reducción. Otra posible solución es utilizar una muestra representativa de todo el grupo, esto con la finalidad de reducir la cantidad de cálculos y por consiguiente reducir la cantidad de memoria utilizada.

## **KMeans**

Es un algoritmo de agrupación que divide las observaciones en K agrupaciones. La agrupación es un tipo de aprendizaje automático no supervisado que tiene como objetivo encontrar subgrupos homogéneos de manera que los objetos en el mismo grupo (agrupaciones) sean más similares entre sí que los demás. Para conseguirlo, el algoritmo busca un número fijo ( $k$ ) de clústers en el dataset.

Esta técnica no supervisada se basa en identificar grupos en los datos de tal manera que todos los datos del grupo (clúster) son datos con características similares mientras que los datos de los otros grupos son diferentes.

El algoritmo intenta minimizar la distancia entre las observaciones que pertenecen a un clúster y su centroide. Es decir, el objetivo del K-Means es minimizar la suma de las distancias entre los puntos y el centroide al que pertenecen.

### - Pasos del algoritmo KMeans

- 1) Inicialización: Se elige la localización de los centroides K grupos aleatoriamente.
- 2) Asignación: Se asigna a cada dato el centroide más cercano.
- 3) Actualización: Se actualiza la posición del centroide a la media aritmética de las posiciones de los datos asignados al grupo.
- 4) Se repiten los paso 2) y 3) hasta que ya no haya cambios.

Se recomienda normalizar los datos, ya que ayuda al clustering porque los grupos se forman a partir de la distancia euclíadiana cuadrada. Esto se realiza en la etapa de la Transformación. También se recomienda evitar caer en la maldición de la dimensionalidad.

### - Desventajas de K-Means

- Se debe elegir el valor de  $k$  y es muy posible que se cometa un error, o que sea imposible escoger una  $k$  óptima.
- Es sensible a outliers. Los casos extremos hacen que el clúster se vea afectado. Aunque esto puede ser algo positivo a la hora de detectar anomalías.
- Es un algoritmo que sufre de la maldición de la dimensionalidad.
- K-Means es muy pesado computacionalmente hablando.

## **RESULTADOS**

### **Evaluación del Rendimiento**

Se midió el rendimiento de cada uno de los algoritmos planteados para el problema.

#### **- Algoritmo de Clasificación de Imágenes**

Se realizó el test de rendimiento con 26 imágenes de cada pieza de ferretería, dando un total de 104 imágenes para el Test de rendimiento.

```
Resultados:  
    Se analizaron 104 piezas  
    Se acertaron: 97 piezas con KNN con K=1  
    Se acertaron: 104 piezas con KNN con K=2  
    Se acertaron: 104 piezas con KNN con K=3  
    Se acertaron: 104 piezas con KNN con K=4  
    Se acertaron: 104 piezas con KNN con K=5  
    Se acertaron: 104 piezas con KNN con K=6  
    Se acertaron: 104 piezas con KNN con K=7  
    Se acertaron: 104 piezas con KNN con K=8  
    Se acertaron: 104 piezas con KNN con K=9  
    Se acertaron: 104 piezas con KMEANS  
  
Porcentajes:  
    Porcentaje de certeza KNN con K=1: 93.26923076923077%  
    Porcentaje de certeza KNN con K=2: 100.0%  
    Porcentaje de certeza KNN con K=3: 100.0%  
    Porcentaje de certeza KNN con K=4: 100.0%  
    Porcentaje de certeza KNN con K=5: 100.0%  
    Porcentaje de certeza KNN con K=6: 100.0%  
    Porcentaje de certeza KNN con K=7: 100.0%  
    Porcentaje de certeza KNN con K=8: 100.0%  
    Porcentaje de certeza KNN con K=9: 100.0%  
    Porcentaje de certeza KMEANS: 100.0%
```

Ilustración 46: Captura de pantalla del programa p8\_rendimiento.py

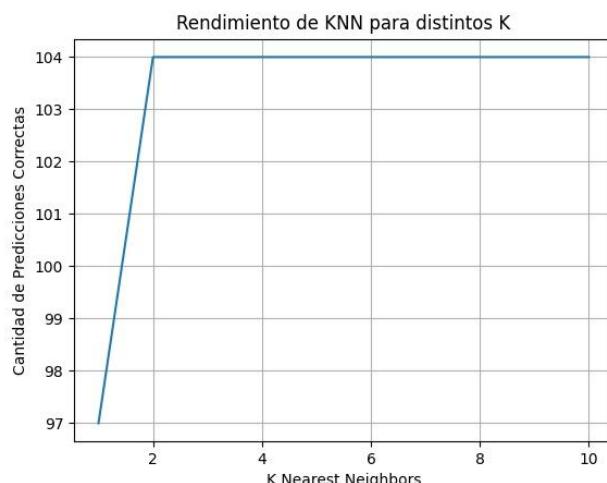


Ilustración 47: Gráfica de predicciones correctas para distintos valores de K para el algoritmo KNN

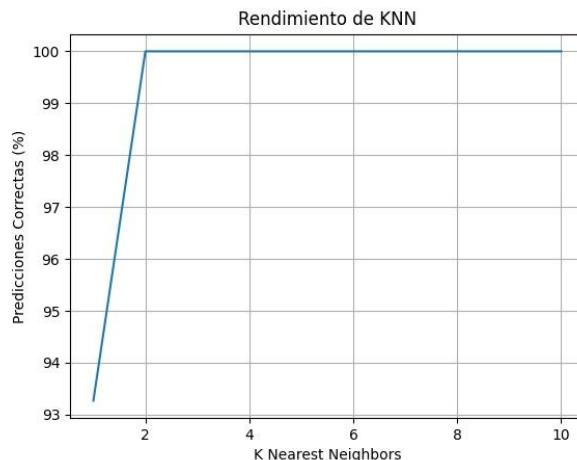


Ilustración 48: Gráfica de porcentaje de certeza para distintos valores de K para el algoritmo KNN

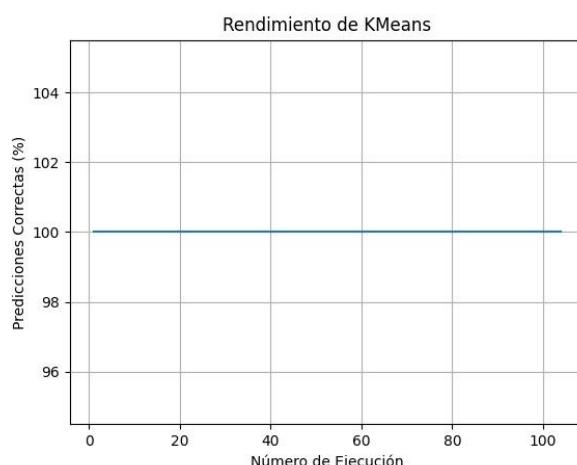


Ilustración 49: Gráfica de porcentaje de certeza para distintas iteraciones del algoritmo KMeans

### 1) KNN

- Se analizó el algoritmo KNN con K Nearest Neighbors entre K=1 y K=10. Se ejecutó el programa variando para la misma imagen el valor de K, obteniendo así una categorización para cada valor K.
- Con todas las consideraciones anteriores, se logró un rendimiento: 96% para K=1 y se observa qué se obtiene una certeza de la categorización del 100% a partir de un valor K=2

### 2) KMeans

- Con todas las consideraciones se logró un rendimiento del 100%, esto se debe al buen agrupamiento de las piezas en la base de datos

#### - Algoritmo de Apilamiento de Cajas

Para medir el rendimiento del algoritmo se hizo la comparación con el juego de Torres de Hanoi con 4 objetos y 3 soportes (Este se obtiene de la página web en <http://lcas.lincoln.ac.uk/fast-downward/>). Se compara la cantidad de movimientos necesarios para llegar al mismo estado objetivo.

```

1 (define (domain hanoi)
2   (:requirements :strips)
3   (:predicates (clear ?x) (on ?x ?y) (smaller ?x ?y))
4   (:action move
5     :parameters (?disc ?from ?to)
6     :precondition (and
7       (on ?disc ?from)
8       (clear ?disc) (clear ?to))
9     :effect (and (clear ?from) (on ?disc ?to)
10      (not (on ?disc ?from))
11      (not (clear ?to))))
12 )

```

Ilustración 50: Reglas en el entorno de las Torres de Hanoi con 4 objetos y 3 soportes

Apilamiento Inicial (Ascendente)	Apilamiento Objetivo	Cantidad de Movimientos	
		Torres de Hanoi	Algoritmo de Apilamiento
		7 Movimientos	6 Movimientos
		9 Movimientos	0 Movimientos

		4 Movimientos	4 Movimientos
		7 movimientos	6 Movimientos
		6 Movimientos	6 Movimientos
		8 Movimientos	6 Movimientos

Tabla 3: Comparación de cantidad de movimientos para distintos algoritmo solución

- Se observa que para obtener el mismo de apilamiento objetivo a partir del mismo estado inicial, el algoritmo desarrollado para el apilamiento de cajas es más eficiente respecto a una menor cantidad de movimientos necesarios para alcanzar el estado objetivo en comparación a la dinámica de resolución de las Torres de Hanoi

### **- Algoritmo de A Estrella (A\*)**

En los requerimientos para desarrollar la solución al problema se menciona implícitamente que se debe usar el algoritmo A Estrella (A\*), sin embargo este debe ser eficiente ante un mapa predefinido, de manera que las pruebas de rendimiento se hicieron con distintas variaciones del laberinto y distintas solicitudes de ruta solución en estos mapas, esto con el fin de determinar la adaptación del algoritmo ante distintos laberintos.

Mapa	Coordenadas	Ruta solución
 Dimensions: 11x11	Punto A: (0,0) Punto B: (5,5)	$[(0, 0), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (5, 6), (5, 5)]$
	Punto A: (2,3) Punto B: (5,9)	$[(2, 3), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (4, 7), (5, 7), (5, 8), (5, 9)]$
	Punto A: (0,0) Punto B: (11,11)	$[(0, 0), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7), (8, 7), (9, 7), (10, 7), (11, 7), (11, 8), (11, 9), (11, 10), (11, 11)]$
	Punto A: (10,10) Punto B: (1,1)	$[(10, 10), (9, 10), (9, 9), (9, 8), (9, 7), (8, 7), (7, 7), (6, 7), (5, 7), (5, 6), (5, 5), (5, 4), (5, 3), (5, 2), (5, 1), (5, 0), (4, 0), (3, 0), (2, 0), (1, 0), (1, 1)]$
 Dimensions: 9x9	Punto A: (1,1) Punto B: (7,4)	No se obtuvo una solución
	Punto A: (2,4) Punto B: (7,9)	$[(2, 4), (2, 5), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (6, 7), (6, 8), (7, 8), (7, 9)]$
	Punto A: (8,1) Punto B: (5,3)	$[(8, 1), (7, 1), (6, 1), (6, 2), (5, 2), (5, 3)]$
	Punto A: (0,0) Punto B: (9,9)	$[(0, 0), (1, 0), (1, 1), (2, 1), (2, 2), (3, 2), (3, 3), (4, 3), (5, 3), (5, 4), (5, 5), (6, 5), (6, 6), (7, 6), (7, 7), (8, 7), (8, 8), (9, 8), (9, 9)]$

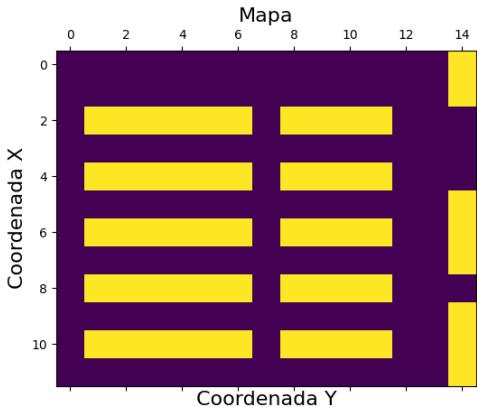
 Dimensiones: 14x11	Punto A: (1,2) Punto B: (5,9)	$[(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (5, 8), (5, 9)]$
	Punto A: (2,3) Punto B: (7,7)	$[(2, 3), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7)]$
	Punto A: (12,11) Punto B: (5,6)	$[(12, 11), (11, 11), (11, 10), (11, 9), (11, 8), (11, 7), (10, 7), (9, 7), (8, 7), (7, 7), (6, 7), (5, 7), (5, 6)]$
	Punto A: (14,11) Punto B: (7,9)	No se obtuvo una solución

Tabla 4: Comparación de cantidad de movimientos para distintos algoritmo solución

- Para cada mapa se probaron 4 rutas que se debían trazar entre las coordenadas del punto inicial y las coordenadas del punto objetivo. Se considera caso exitoso si el programa devuelve una ruta entre ambos puntos
- Se observa que en los laberintos con dimensiones cuadras se pudo obtener todas las rutas, mientras que los laberintos con dimensiones rectangulares no pudo dar una ruta para algunos casos.

## Análisis de Resultados

Luego de todo el proceso de análisis y prueba, se seleccionó que las siguientes características:

### - Algoritmo de Clasificación de Imágenes

- La imagen debe ser recortada 50 píxeles respecto al borde izquierdo, para eliminar la franja de la pared. Luego realizar un enmascaramiento para que tenga un fondo totalmente blanco. Posteriormente redimensionada a 500x400 píxeles y se eliminan 100px de abajo hacia arriba de la imagen, para eliminar la sombra que se detecta durante el enmascaramiento. Se pasa de tener una imagen de 500x400 px a una imagen de 500x300 px; el cuál será el formato a utilizar para este proyecto.
- La imagen no debe ser recortada en la etapa de adaptación, o sea se debe tomar directamente la imagen de la fase de transformación para la base de datos.
- Las características de la pieza a analizar son el momento de Hu 1, el eje menor y eje mayor de la pieza.
- Se hará un filtrado con Gauss+Sobel.
- No se utilizará thresholding, por eso no se utiliza la etapa de adaptación.
- Si bien se obtuvieron buenos resultados para un valor bajo del hiperparámetro K, se evaluará para K=1 y K=3. Se eligió esos valores porque se aprecia que para esos valores

se tiene un alto rendimiento y como se tiene un número par de piezas de ferretería se eligió K=3 en vez de K=2.

- A valores bajos de K, hay sobreajuste de datos / alta varianza. Por lo tanto, el error de prueba es alto y el error de tren es bajo. En K = 1 en los datos del tren, el error siempre es cero, porque el vecino más cercano a ese punto es ese punto en sí. Por lo tanto, aunque el error de entrenamiento es bajo, el error de prueba es alto con valores de K más bajos. A esto se le llama sobreajuste. A medida que aumentamos el valor de K, se reduce el error de prueba.
- Se devuelve un archivo de texto con los elementos que se identificaron con registro horario y fecha, que posteriormente se utiliza en el algoritmo de apilamiento de cajas.
- Las predicciones que fallan suelen ser porque el algoritmo no distingue un tornillo o un clavo de una arandela. Ésto suele pasar porque:
  - a. Si al momento de tomar la imagen el tornillo o clavo no se encuentra próximo al ángulo de normalización, una rotación brusca altera la determinación de propiedades (algunas varían de acuerdo a su rotación), por lo que los valores de la pieza no coinciden con los de su categorización. De manera que para evitar este error se debe de tratar de poner las piezas lo más horizontal posible a la cámara.
  - b. Si durante la ejecución del programa gui\_clasificacion.py se interrumpe y luego se continua con la misma ejecución, en las gráficas de agrupación de características aparecerán datos anómalos que alteran la predicción de resultados. Para evitar este tipo de pronósticos, si se interrumpe la ejecución del programa gui\_clasificacion.py, se debe comenzar de nuevo el análisis completo de las piezas.

### **- Algoritmo de Apilamiento de Cajas**

- Se consideró qué en juego hay 4 cajas con piezas distintas para el análisis de la solución.
- Al tener 4 objetos que están en juego durante el proceso de apilamiento de cajas, se consideraron 4 regiones delimitadas sobre la mesa en las cuales se podrá interactuar con las cajas.
- Todas las cajas se encuentran inicialmente apiladas una sobre otra y posicionadas en la región de la “mesa A”.
- Se debe distinguir que si al movilizar una caja, esta se encuentra sobre la mesa o sobre otra caja, ya que hay operadores designados para realizar cada una de estas acciones.
- Durante la implementación se consideraron 3 posibles situaciones respecto al apilamiento inicial con el apilamiento objetivo:
  - a. El orden de apilamiento inicial es igual al orden de apilamiento objetivo, por lo que no hay ningún movimiento por hacer

- b. El orden de apilamiento inicial es igual al orden de apilamiento objetivo invertido, por lo que se debe invertir únicamente el orden de apilamiento, reduciendo así el coste del camino.
- c. El orden de apilamiento inicial no tiene relación directa con el orden de apilamiento objetivo, por lo que se debe hacer todo la secuencia preestablecida para alcanzar el orden de apilamiento objetivo.
- En comparación al Juego de Torres de Hanoi (disponible en <http://lcas.lincoln.ac.uk/fast-downward/>), el algoritmo de Torres de Hanoi provoca que el coste del camino solución sea mayor al algoritmo de apilamiento para ciertas situaciones. Sin embargo con ambos métodos se llega al resultado esperado.

#### **- Algoritmo de A Estrella (A\*)**

- Para establecer un laberinto para utilizarlo en el programa a\_estrella.py o gui\_movimiento.py, este se debe cargar desde el programa laberinto.py como una lista de una lista, donde los número 1 representan espacios donde no se puede avanzar y los número 0 espacios por donde sí se puede avanzar.
- El tiempo máximo promedio para calcular el path solución es de 90 segundos, de manera que si supera ese tiempo, significa qué no hay una solución para los parámetros establecidos. Algunas razones para qué suceda esto:
  - a. Se observa un mejor rendimiento cuando las dimensiones del laberinto son un cuadrado en comparación a dimensiones rectangulares.
  - b. Puede que uno de los factores para que se devuelva una ruta o no, sin importar las dimensiones del laberinto, es la cantidad de obstáculos qué posee el mismo, ya que implica un mayor cálculo en los nodos del grafo.
- Dado los resultados en las pruebas de rendimiento, para garantizar obtener una solución, el laberinto debe tener proporciones cuadradas y pocos obstáculos.
- El tiempo de cómputo es una gran desventaja de A\*. Debido a que guarda en la memoria todos los nodos generados, por lo general A\* se queda sin memoria mucho antes de agotar el tiempo.
- No se puede aplicar el algoritmo diseñado para mapas que presentan discontinuidades, es decir, un mapa que tiene al menos una fila o columna de mayor o menor dimensión que el resto. En esta situación el programa directamente marca un error durante el cálculo del grafo. De manera que uno de los requisitos para aplicar este programa es que el laberinto debe ser continuo y uniforme en sus bordes.

## CÓDIGO

### **Etapa 1: Procesamiento de Imágenes**

1) Primero se hizo el programa en forma separada para evaluar cada una de las funciones a implementar en el proyecto. Ésto se puede observar en los programas “pX\_etapa.py”, donde X es el número de la etapa de procesamiento de imágenes.

2) Luego se pone en evidencia cada etapa del proyecto por separado con su respectivo programa en Python. Haga clic en cada hipervínculo para mostrar la ubicación del programa y poder verlo.

- Transformación: Se muestra quitar línea de fondo, redimensión de imagen, quitar fondo y quitar sombra posterior a quitar fondo → convertir el fondo a uno totalmente blanco. ([Ver código](#)).
- Adaptación: Se muestra el algoritmo de Canny para detectar bordes, uso de thresholding para detectar bordes y recortar imagen a partir del Algoritmo de Canny. ([Ver código](#)).
- Preprocesamiento: Se muestra la descomposición de una imagen en la tupla de valores R, G y B, conversión de RGB a escala de grises utilizando librerías y conversión de RGB a escalas de grises utilizando pesos WG, WR y WB arbitrarios ingresados manualmente. ([Ver código](#)).
- Filtración: Se muestra la implementación del filtro Sobel y Roberts, filtro Gaussiano+Sobel, Perona Malik, filtro Laplace, Median, Frangi y Prewitt. ([Ver código](#)).
- Segmentación: Se muestra thresholding supervisado y thresholding no supervisado. ([Ver código](#)).
- Procesamiento: Registra todas las transformaciones que sufre una imagen previo a la extracción de rasgos, esto para fines ilustrativos pero no interviene en la categorización de piezas. ([Ver código](#)).
- Extracción: Se muestra la representación de Histograma de color para una imagen filtrada y comparación de características Hu, HOG, Excentricidad, Eje menor y Eje mayor para las distintas piezas de ferretería. ([Ver código](#)).
- Reducción: Se muestra la reducción de dimensionalidad para Hu, HOG, Excentricidad, Eje menor y Eje mayor. ([Ver código](#)).
- Rendimiento: Se determina el rendimiento para KNN y KMeans para distintos métodos de extracción de características. Contiene gran parte de los programas previamente mencionados. ([Ver código](#)).
- Clasificación: Este es el programa principal del proyecto, con este se evalúa la clasificación para cada imagen. Contiene gran parte de los programas previamente mencionados. ([Ver código](#)).
- GUI\_Clasificación: Este programa lanza el servidor local que permite usar la interfaz de usuario que ejecuta el programa principal de clasificación de piezas (p7\_clasificacion.py) y devuelve la identificación del contenido de cada una de las cajas, así como el orden de apilamiento ascendente de las cajas. ([Ver código](#)).

## **Etapa 2: Pasos que debe realizar el robot para alcanzar otro orden de apilado**

Para el diseño del algoritmo se tomó como inspiración la resolución al juego de las Torres de Hanoi. Posterior a determinar el orden de apilamiento de las cajas en la etapa 1, se pide por consola o por la GUI el nuevo orden de apilamiento.

- Apilamiento: Este programa devuelve la secuencia de operaciones qué se deben hacer para obtener el nuevo orden de apilamiento. Esta secuencia es en lenguaje STRIPS. Devuelve las propiedades del Problema, Secuencia en Lenguaje STRIPS , Cantidad de Movimientos. ([Ver código](#)).
- Datos: Es un archivo de texto que contiene el orden de apilamiento que se obtuvo en la etapa de procesamiento de imágenes.
- GUI\_Apilamiento: Este programa lanza el servidor local que permite usar la interfaz de usuario que ejecuta el programa Apilamiento.py y devuelve las propiedades del problema, secuencia en lenguaje STRIPS y cantidad de Movimientos. ([Ver código](#)).

## **Etapa 3: Transportar las cajas desde un punto A a un punto B**

Se hicieron dos códigos, de manera qué uno contenga los parámetros que se deben ajustar para cada situación según el mapa donde se debe aplicar el algoritmo y otro que contenga el método de resolución A\*. Estos se encuentran dentro de la carpeta Movimiento.

- Laberinto: Contiene el mapa preestablecido donde se debe aplicar el algoritmo como un conjunto de listas. Es el único parámetro que variará en cada contexto de aplicación. ([Ver código](#)).
- A estrella: Extrae del código Laberinto.py el mapa para armar los nodos y aplicar el método de resolución A\*. Se ingresa por consola las coordenadas del punto A (punto de partida) y punto B (punto objetivo). ([Ver código](#)).
- Mapa: Es una imagen qué muestra de manera gráfica la disposición del espacio del mapa para el contexto de aplicación.
- Mapa Solución: Es una imagen qué muestra la solución de llegar del punto A al punto B de manera gráfica.
- GUI\_Movimiento: Este programa lanza el servidor local que permite usar la interfaz de usuario que ejecuta el programa A\_Estrella.py y devuelve la ruta junto con el mapa con la solución. ([Ver código](#)).

Para cada una de las etapas, se desarrolló la implementación de manera que se pueda ejecutar por consola o por la Interfaz Gráfica de Usuario, de manera que se pueda ejecutar el programa solución sin lanzar el servidor local de la GUI.

El proyecto está alojado en un repositorio en el sitio de GitHub, con el fin de facilitar las versiones de cada uno de los programas desarrollados. ([Ver Repositorio](#))

## EJEMPLO DE APLICACIÓN

A continuación se describe un hipotético caso que el programa desarrollado debe poder resolver.

### Descripción de la tarea

Etapa #1	Etapa #2	Etapa #3
<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Clavos</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Arandelas</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Tornillos</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Tuercas</div> <div style="background-color: #8B4513; width: 100px; height: 10px;"></div> </div> <p>Orden de Apilamiento Inicial</p>	<div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Tornillos</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Clavos</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Tuercas</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">Arandelas</div> <div style="background-color: #8B4513; width: 100px; height: 10px;"></div> </div> <p>Orden de Apilamiento Objetivo</p>	<p>Mapa</p> <p>Coordenada X</p> <p>Coordenada Y</p> <p>Se deben movilizar de la posición (3,5) a la posición (7,9)</p>

Tabla 5: Requerimientos de la tarea a cumplir

### Desarrollo de Etapa 1: Procesamiento de imágenes de las piezas representativas de cada caja

Primero se establece la comunicación por IP entre cámara y computadora para tomar las fotografías. Luego se procede a extraer una pieza de cada una de las cajas y se realiza la toma de imágenes de cada una de ellas para realizar su procesamiento. Luego, manteniendo la nomenclatura para cada caja, se ingresan las fotos al programa `gui_clasificacion.py` para comenzar la identificación de las piezas. El programa `gui_clasificacion.py` es la interfaz gráfica de usuario del algoritmo `p7_clasificacion.py`

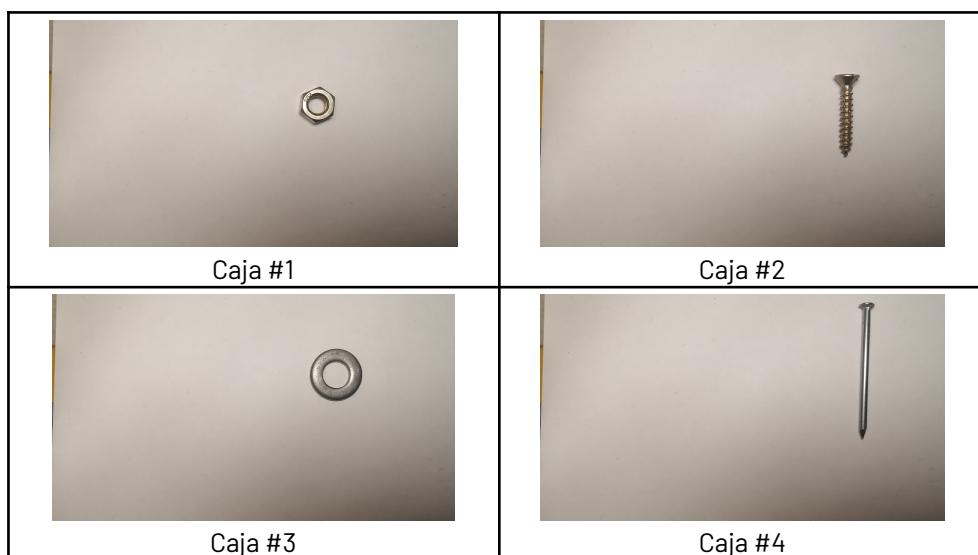


Ilustración 51: Toma de Fotos del contenido de cada caja

Orden de apilamiento: Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)

## Clasificación

**Input:** Pieza representativa de cada una de las cajas, según el orden en que están apiladas.

**Output:** Identificación del contenido de cada una de las cajas y orden de apilamiento ascendente.

**Orden de apilamiento:** Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)

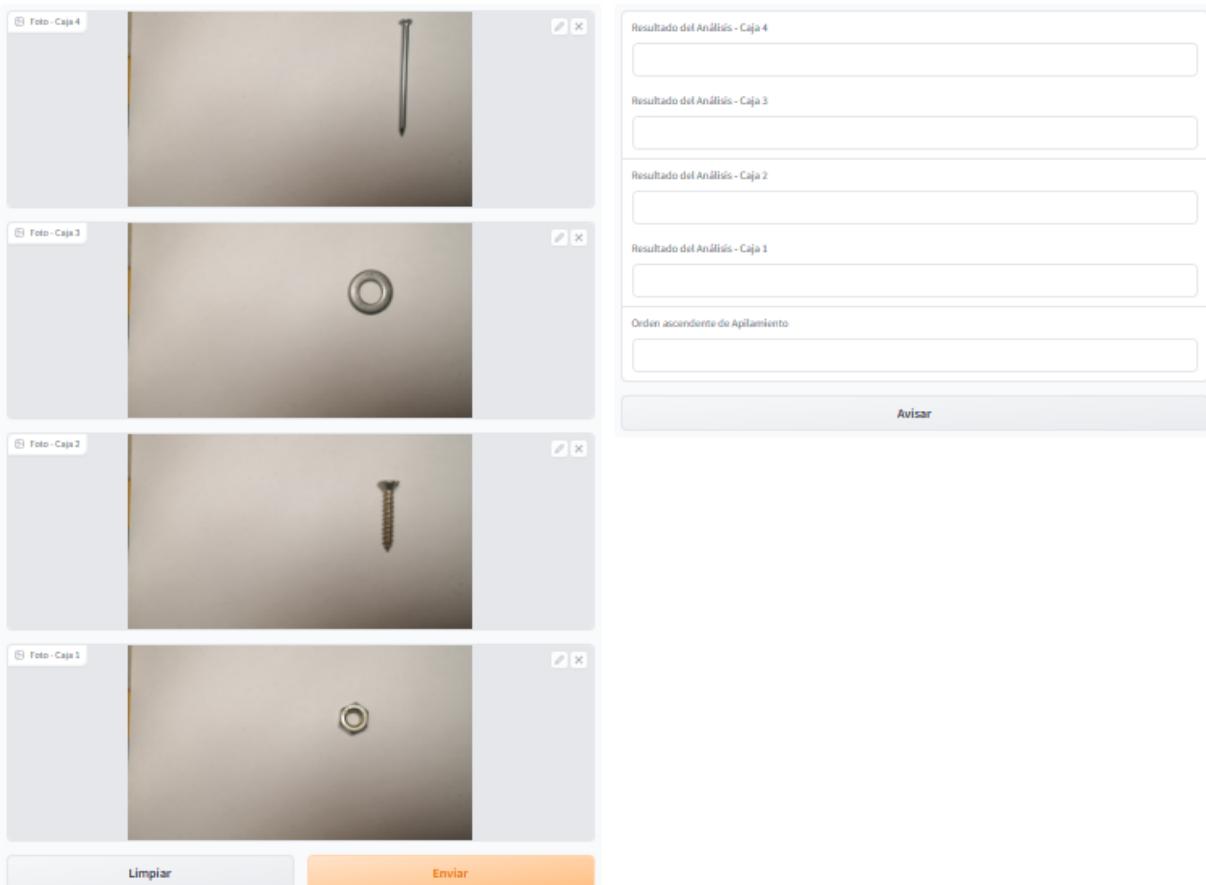


Ilustración 52: Carga de inputs en el programa gui\_clasificacion.py

Orden de apilamiento: Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)

## Análisis de Datos

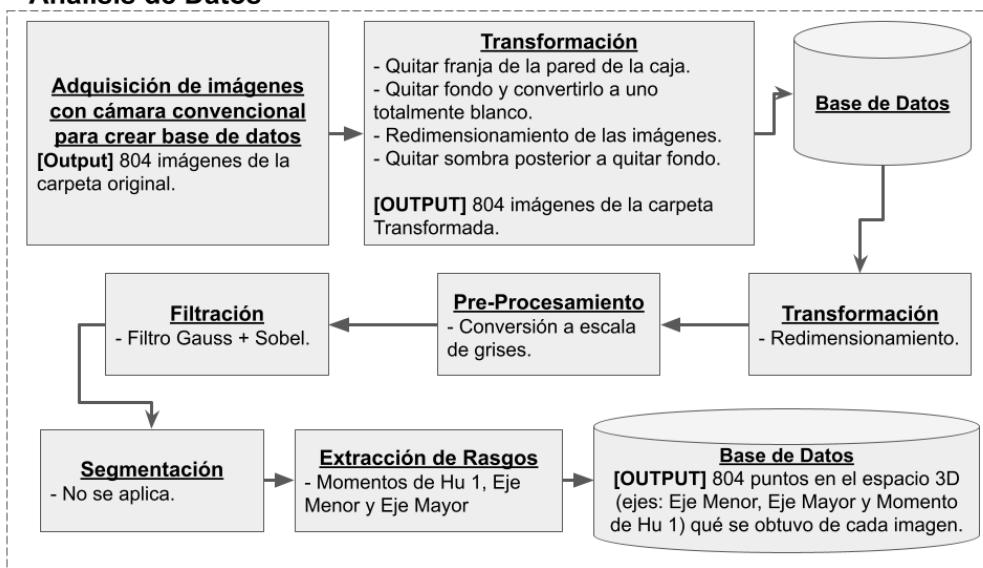


Ilustración 53: Diagrama de flujo de la parte de "Análisis de datos" del programa p7\_clasificación.py

### Procesamiento de Imágenes

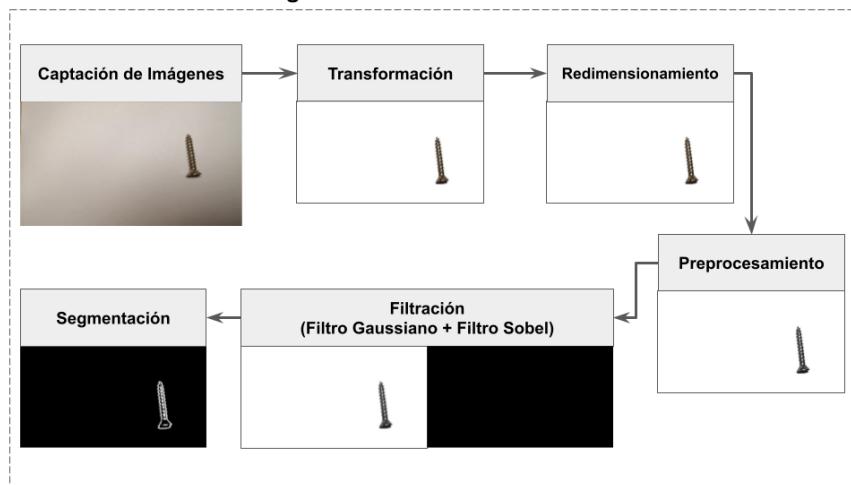


Ilustración 54: Diagrama de flujo del resultado de procesamiento de cada una de las imágenes de entrada, previo a la extracción de características.

### Análisis completo de carpeta Train

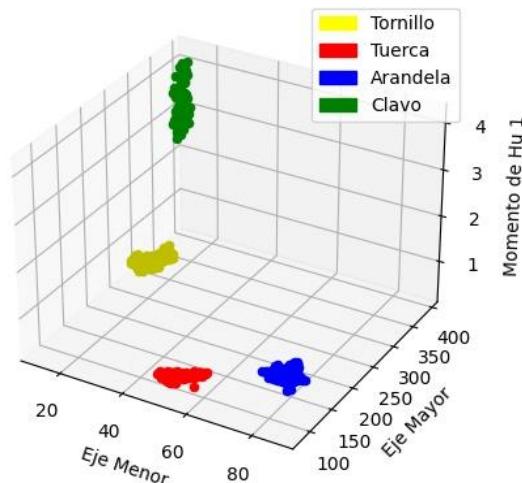


Ilustración 55: Base de datos formada por 804 imágenes de la carpeta Transformada.

### KNN

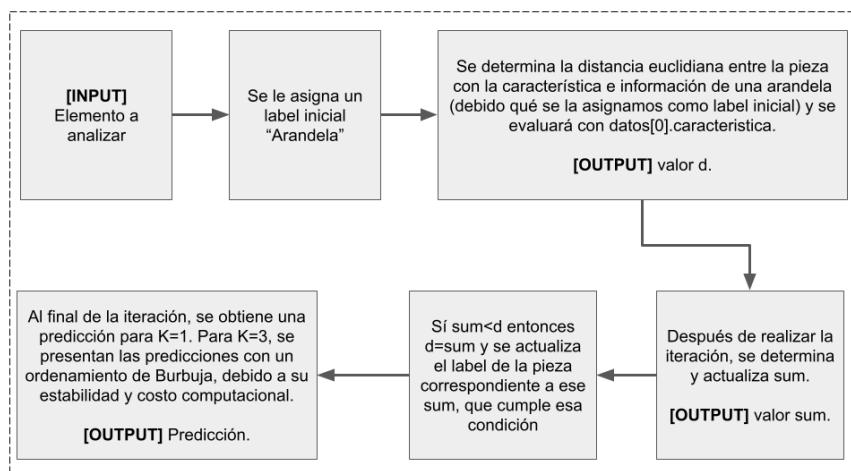


Ilustración 56: Diagrama de flujo de la parte de "KNN" del programa p7\_clasificacion.py

## KMeans

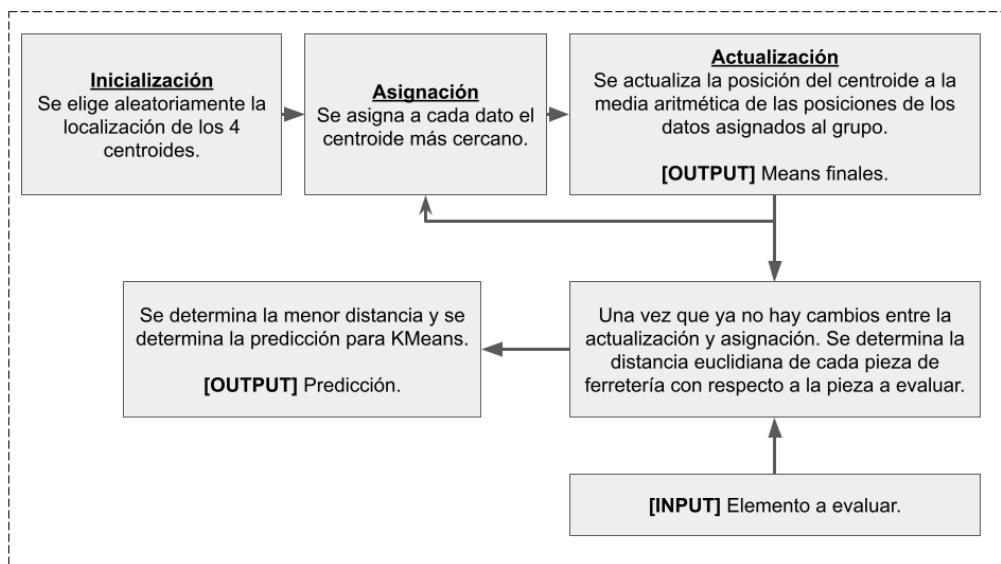


Ilustración 57: Diagrama de flujo de la parte de "KMeans" del programa p7\_clasificación.py

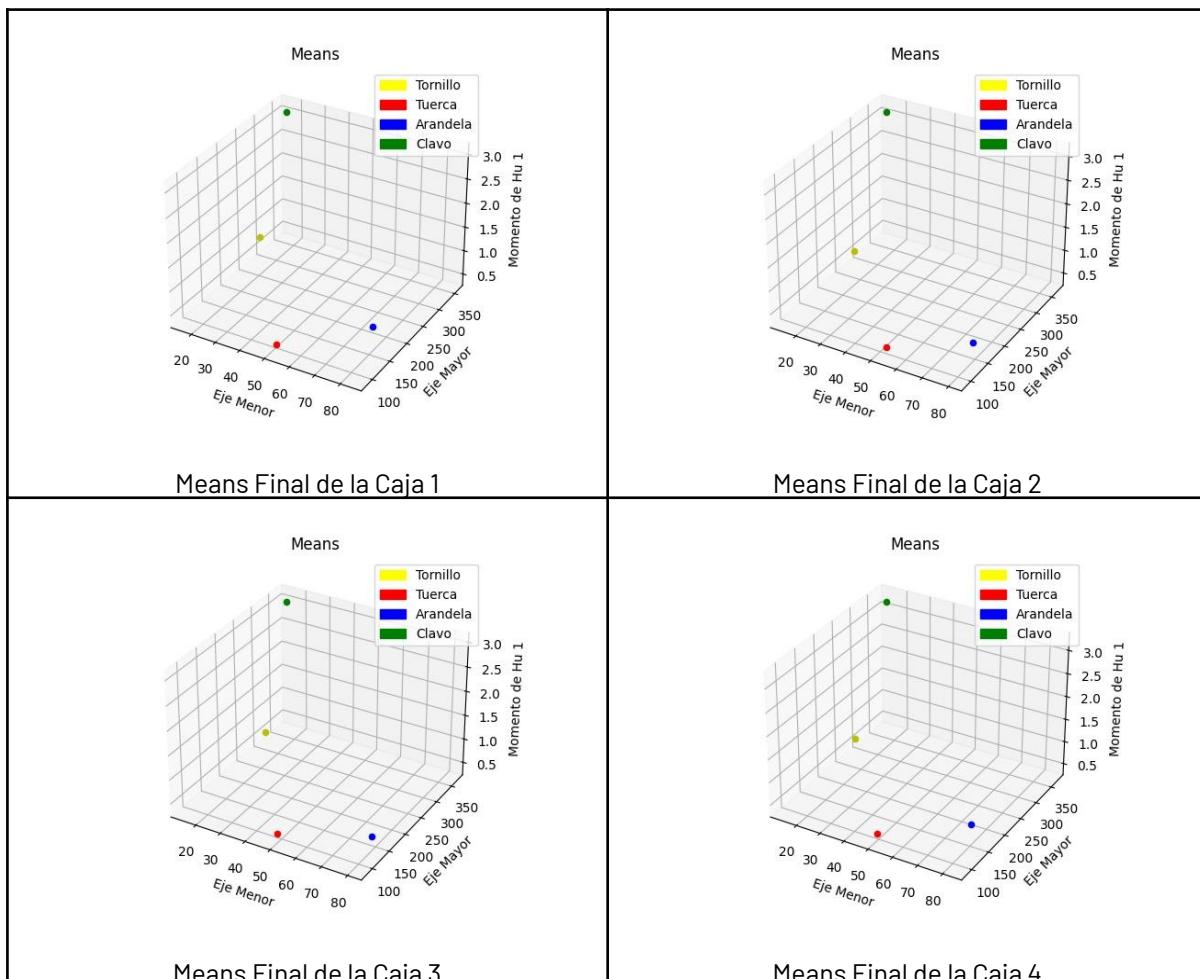


Ilustración 58: Means aleatorios iniciales obtenido a partir de las 804 imágenes

Means Finales

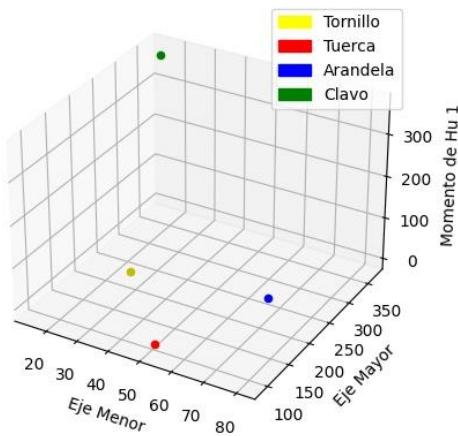


Ilustración 59: Means finales obtenido a partir de las 804 imágenes

### Clasificación

**Input:** Pieza representativa de cada una de las cajas, según el orden en que están apiladas.  
**Output:** Identificación del contenido de cada una de las cajas y orden de apilamiento ascendente.  
**Orden de apilamiento:** Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)

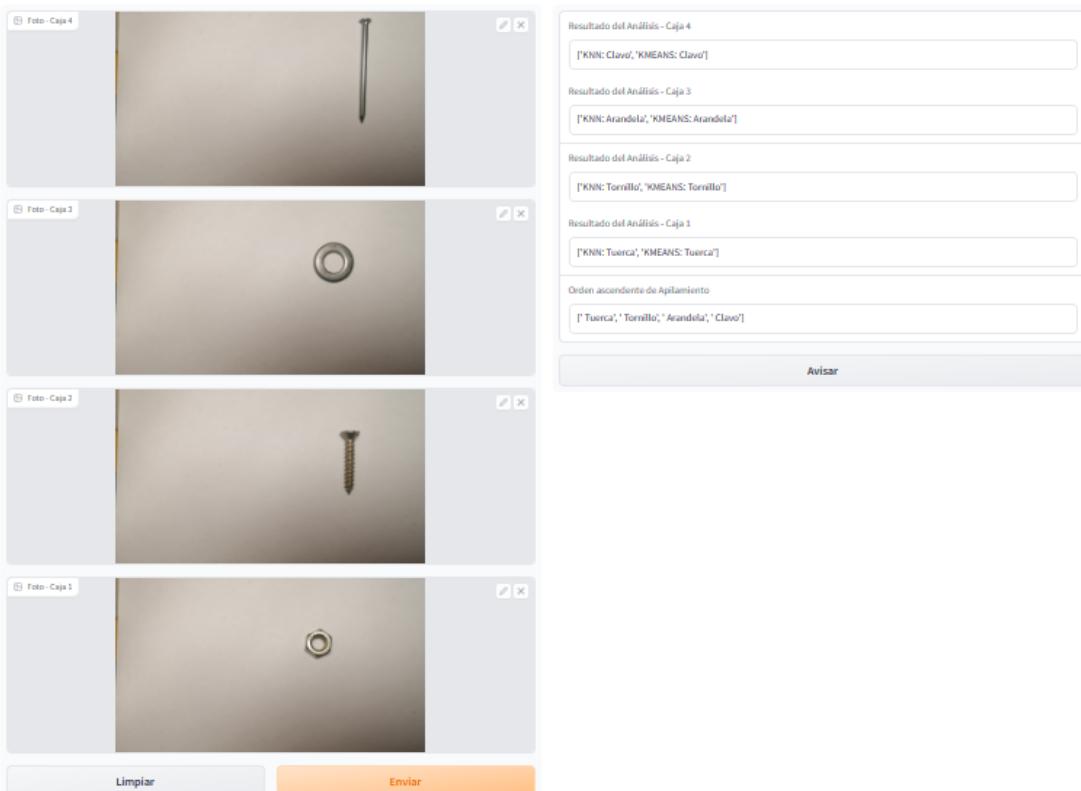


Ilustración 60: Resultado del análisis de imágenes del contenido de las cajas

El programa `gui_clasificación.py` devolvió la identificación del contenido de cada caja y el orden de apilamiento ascendente de las cajas, con ello se concluye la etapa 1.

## Desarrollo de Etapa 2: Pasos que debe realizar el robot para alcanzar el nuevo orden de apilamiento de cajas

Posterior a ello, se ejecuta el programa `gui_apilamiento.py` en el cuál se debe ingresar el nuevo orden de apilamiento objetivo de las cajas planteado en la etapa 2.

Input: Orden de Apilamiento Objetivo.  
Output: Propiedades del Problema, Secuencia en Lenguaje STRIPS , Cantidad de Movimientos  
Orden de apilamiento: Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)

Desafío: Al reconocer el orden de apilamiento de las cajas, mediante lenguaje STRIPS se debe encontrar los pasos que debe realizar el brazo robótico para alcanzar un nuevo orden de apilado de cajas.  
Apilamiento Inicial: Sobre(Tuerca,mesa A), Sobre( Tornillo,Tuerca), Sobre( Arandela,Tornillo), Sobre( Clavo, Arandela),  
Mario Bustillo 2023 [🔗](#) | [Github](#) | [Linkedin](#) 😊

Ilustración 61: Carga de inputs en el programa `gui_apilamiento.py`

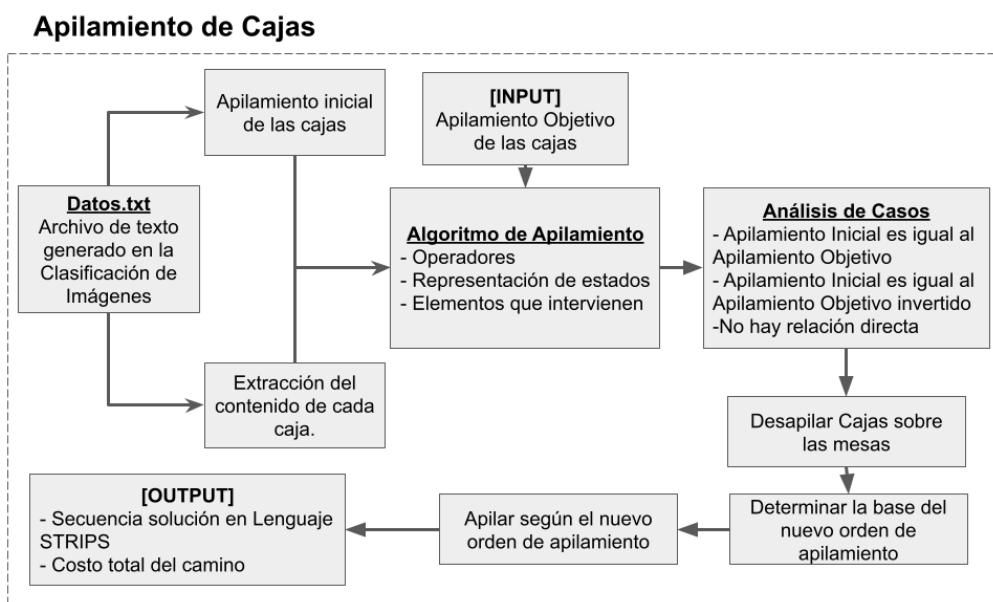


Ilustración 62: Diagrama de flujo para calcular los pasos que debe realizar el robot para alcanzar otro orden de apilado del programa `apilamiento.py`

### Apilamiento

**Input:** Orden de Apilamiento Objetivo.  
**Output:** Propiedades del Problema, Secuencia en Lenguaje STRIPS , Cantidad de Movimientos  
**Orden de apilamiento:** Sobre(Caja 4,Caja 3), Sobre(Caja 3,Caja 2), Sobre(Caja 2,Caja 1), Sobre(Caja 1, Mesa)

Caja 4	<input type="radio"/> Tuerca	<input checked="" type="radio"/> Tornillo	<input type="radio"/> Arandela	<input type="radio"/> Clavo
Caja 3	<input type="radio"/> Tuerca	<input type="radio"/> Tornillo	<input type="radio"/> Arandela	<input checked="" type="radio"/> Clavo
Caja 2	<input checked="" type="radio"/> Tuerca	<input type="radio"/> Tornillo	<input type="radio"/> Arandela	<input type="radio"/> Clavo
Caja 1	<input type="radio"/> Tuerca	<input type="radio"/> Tornillo	<input checked="" type="radio"/> Arandela	<input type="radio"/> Clavo

**Objetos**  
 Caja(Tuerca), Caja(Tornillo), Caja(Arandela), Caja(Clavo), mesa A, mesa B, mesa C, mesa D, Brazo

**Estado Inicial**  
 Sobre(Tuerca,mesa A), Sobre(Tornillo, Tuerca), Sobre(Arandela, Tornillo), Sobre(Clavo, Arandela), Libre(Tornillo), Libre(Brazo)

**Estado Objetivo**  
 Sobre(Arandela,mesa C), Sobre(Tuerca, Arandela), Sobre(Clavo, Tuerca), Sobre(Tornillo, Clavo),

**Secuencia Solución**  
 Desapilar(Clavo, Arandela)  
 Soltar(Clavo, mesa B)  
 Desapilar(Arandela, Tornillo)  
 Soltar(Arandela, mesa C)  
 Desapilar(Tornillo, Tuerca)  
 Soltar(Tornillo, mesa D)  
 Sujetar(Tuerca)  
 Apilar(Tuerca, Arandela)  
 Sujetar(Clavo)  
 Apilar(Clavo, Tuerca)  
 Sujetar(Tornillo)  
 Apilar(Tornillo, Clavo)

**Costo total del camino**  
 6 Movimientos

Ilustración 63: Resultado del procedimiento mediante lenguaje STRIPS

El programa `gui_clasificación.py` devolvió la serie de pasos que debe ejecutar el usuario/operario/brazo robótico para alcanzar el nuevo orden de apilamiento de las cajas, además devuelve las propiedades del Problema y la cantidad de movimientos que le tomará al usuario/operario/brazo robótico llegar al apilamiento objetivo para las condiciones del problema dado.

## **Desarrollo de Etapa 3: Transportar las cajas desde un punto A a un punto B a través del Laberinto**

El siguiente paso es comenzar con la etapa 3, para ello se debe ejecutar el programa `gui_movimiento.py` en el cuál se debe ingresar las coordenadas del punto actual y las coordenadas del punto objetivo a donde se desean trasladar las cajas.

### Resolución de Laberinto

**Input:** Ingresar coordenadas de Inicio y Fin.  
**Output:** Path , Mapa con la solución  
**Características:** El Laberinto tiene dimensiones: 11x11, siendo la esquina superior izquierda la coordenada (0,0)

Coordenada X - Punto Inicio  
 3

Coordenada Y - Punto Inicio  
 5

Coordenada X - Punto Final  
 7

Coordenada Y - Punto Final  
 9

**Secuencia Solución**

**Mapa Solución**


**Avisar**

Ilustración 64: Carga de inputs en el programa `gui_movimiento.py`

## Algoritmo A Estrella (A\*)

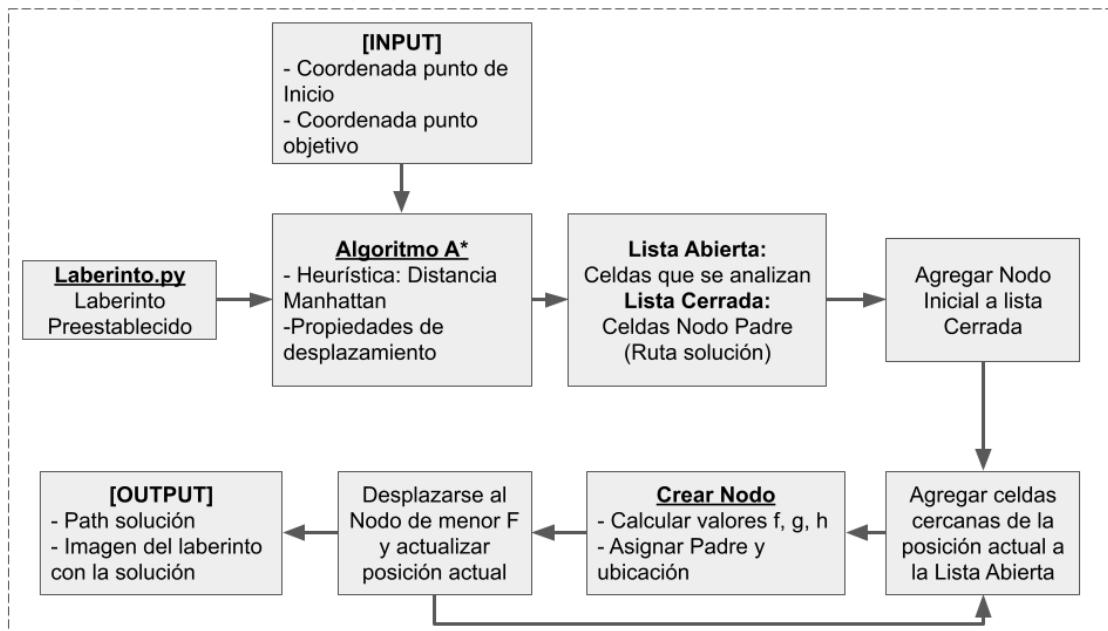


Ilustración 65: Diagrama de flujo para calcular los movimientos que debe realizar el robot para transportar las cajas desde un punto A a un punto B del programa a\_estrella.py

### Resolución de Laberinto

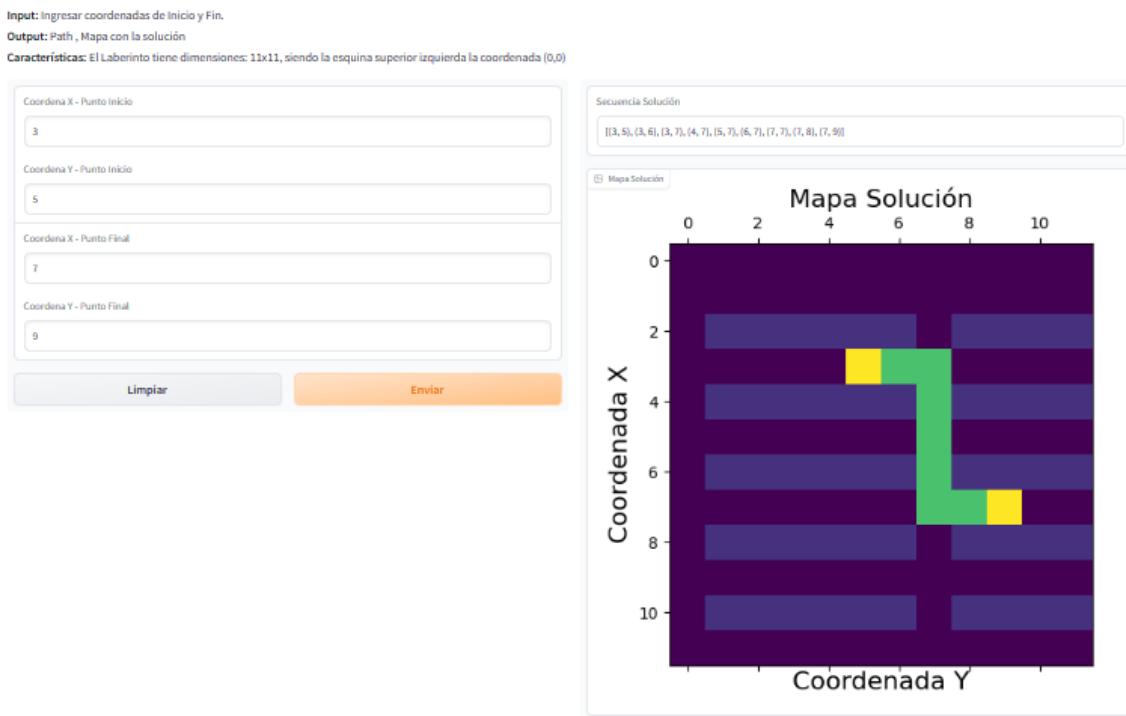


Ilustración 66: Captura de pantalla del programa gui\_movimiento.py

El programa `gui_movimiento.py` devuelve la ruta que debe seguir el usuario/operario para transportar las cajas desde un punto inicial a un punto objetivo con el coste de camino, esta ruta se devuelve como ubicaciones coordenadas(x,y) y como versión gráfica.

## **CONCLUSIONES**

### **Conclusiones Generales**

- Debido al sistema operativo Kubuntu utilizado existieron muchos conflictos con las librerías utilizadas, por eso las imágenes se tuvieron que procesar en partes. Por un lado la eliminación de la franja de pared de la caja y conversión del fondo a uno totalmente blanco, y por otro lado el correspondiente análisis.
- En total se tomaron 928 imágenes para el análisis de este proyecto. El 87% se destinó al entrenamiento, el 11% se destinó al Test y el otro 2% se utilizó para la evaluación de los programas durante cada etapa.
- El diseño de interfaz de usuario (GUI) si bien se planteó para generar una mejor experiencia y facilidad para el usuario, al momento de desarrollar la implementación presentó varios desafíos respecto a la muestra de datos y resultados, ya qué para la programación de esta es una combinación de lenguajes de desarrollo Web y python 3.0.
- Con el uso de una caja cerrada con iluminación frontal y fondo blanco (a diferencia del fondo negro, con éste se ponía en evidencia las sombras que se producían debido a la disposición de los focos en la caja), se obtuvo un alto rendimiento superior al 90% tanto para KNN y KMeans. Por otro lado, debido a la disposición dentro de la cámara en la caja se tenía que realizar transformaciones a la imagen de entrada.
- El armado de una caja cerrada con iluminación controlada permitió tener un entorno controlado y estático, de manera que los factores externos no afectarán la toma de fotografías de las piezas para realizar su análisis, de manera qué no depende de factores externos en la predicción de resultados.
- Dentro de la caja, se delimitó un área donde se puede poner la pieza ya que durante la etapa de transformación como se hace un recorte de la imagen, si alguna parte de la pieza estaba fuera de esta área, se perdería información. Así qué se debe tener la precaución de no superar estos márgenes.
- Dentro de la caja existe una región en la cual no se producen sombras ni brillos, debido a la iluminación frontal, pero el problema es que si la base de datos se tomará solo en esta región los momentos de Hu entre tuercas y arandelas son muy similares disminuyendo el rendimiento por falsos positivos. En base a eso, se logra determinar una región que ampara todas estas consideraciones que es donde tomamos cada imagen para la base de datos.
- Con el algoritmo KNN se obtuvieron mejores resultados, tanto para K=1 y K=3, pero presenta un alto tiempo de ejecución. Recordar que KNN compara el vector de características de la imagen a clasificar con cada vector de características de cada una de las imágenes de la base de datos Train. Mientras que KMeans es más rápido en ejecución debido que calcula la distancia a los 3 means principales.
- El tiempo promedio para la categorización del contenido de las cajas es aproximadamente entre 90 a 120 segundos.

- En la etapa de apilamiento de cajas, se observó una reducción de cantidad de movimientos necesarios para alcanzar el estado objetivo respecto a la similitud con el juego Torres de Hanoi con 4 piezas y 3 soportes. De manera que el desarrollo del algoritmo para resolver el problema demuestra una mejor eficiencia respecto a usar la plataforma propuesta para resolver el problema.
- Uno de los problemas al momento de representar el path solución durante la tercera etapa fue la orientación de los ejes, ya que la enumeración de los mismos esta invertida, de manera que para solucionar esto se tuvo que reorientar todos los parámetros de coordenadas, se pasó del estilo (x,y) a coordenada del tipo (y,x).
- Al momento de encontrar el camino de menor costo, para ciertas celdas se tuvo el mismo valor de costo final, de manera qué para escoger la siguiente casilla se definió un orden de prioridades para estas circunstancias. En orden de mayor prioridad, este se estableció como Arriba, Abajo, Izquierda, Derecha; sin embargo se puede modificar de acuerdo a circunstancias del problema.

## **Ventajas y Desventajas**

- Al comprobar el sistema de captación de imágenes ante distintos niveles de luz natural y obtener resultados certeros en la categorización, el agente que se ha planteado es en su mayoría independiente de factores naturales externos .
- Si bien se tuvieron qué normalizar ciertos aspectos, como la posición y giro de los tornillos y clavos en el plano para tomar las fotos, la posición inicial de las cajas apiladas y las dimensiones de cuadradas del laberinto con pocos obstáculos; esto permitió asegurar que los programas devolverán una respuesta y esta tendrá un gran porcentaje de certeza, evitando así resultados erróneos y falso positivos en las predicciones.
- De no cumplirse la normalización de los aspectos anteriores, no se puede asegurar tener un output por parte del programa o garantizar la certeza de la predicción de la misma.
- Al implementar el lenguaje STRIPS en el algoritmo de Apilamiento de cajas, este indica que acciones hacer pero no indica cuánto dura una acción ni cuando ocurre y en algunos dominios es importante saber cuando empiezan y terminan las acciones.

## **Posibles mejoras a futuro**

- Si bien se normalizó la posición y el ángulo de rotación respecto al plano de los tornillos y clavos para la etapa de toma de imágenes, se puede agrandar el Database con más imágenes con las piezas en otra posición y ángulos de rotación, de manera que el análisis de imágenes sea independiente de la orientación de la pieza al plano. Pero consecuentemente puede que se deba replantear las propiedades a analizar de los objetos para obtener una mejor agrupación de las piezas.
- Si bien está establecido qué únicamente hay en juego 4 cajas, cada una con un contenido distinto, es posible la expansión del algoritmo de clasificación y apilamiento a una mayor cantidad de elementos en juego.

- Para obtener un mejor rendimiento, respecto a una menor cantidad de movimientos necesarios para alcanzar un estado objetivo, se podría armar el grafo completo o incluir más casos dentro del programa de apilamiento, ya que se puede presentar la situación de que 2 de las 4 cajas no se tengan que movilizar de la posición en la que se encuentran.
- Para encontrar el camino más óptimo para llegar de un punto A a un punto B, se podría optimizar el programa combinando el algoritmo A estrella con otro algoritmo de búsqueda informada.

## **Aplicaciones**

- Podría pensarse en un sistema de reconocimiento automático de éstos objetos, los cuales van por una cinta transportadora, para su clasificación en caja o embolsados. Si en la línea de producción aparece un elemento que, captado por una cámara y una vez digitalizada la imagen, se parece (dentro de un margen de confianza en el parecido) a un tornillo, tuerca, arandela o clavo, se dirigirá a la caja correspondiente. Si no se tirará en una caja de elementos desconocidos. De acuerdo con la velocidad de la cinta transportadora se puede elegir entre el algoritmo KNN y KMeans, siendo éste último más rápido en tiempo de ejecución.
- También se podría plantear una cinta transportadora transparente por la que circulen tornillos, tuercas, arandelas y clavos, donde se necesite desarrollar un sistema que cuente cuántas unidades de cada tipo hay en cada momento en un intervalo de la cinta. Se supone que la iluminación se realiza a contraluz con lo que se obtienen las siluetas. Acá se puede utilizar el criterio el número de agujeros presentes en la pieza y la desviación típica de las distancias del perímetro al centro de la pieza.
- En base al mismo planteamiento de categorización de piezas de ferretería, se podría desarrollar un sistema de recomendación, de manera que en lugar de categorizar una imagen devuelva objetos similares en base a las propiedades de la imagen.

## **Recomendaciones para la implementación de la aplicación**

Para recomendar un algoritmo para esta aplicación de visión artificial, en especial para la identificación de piezas de ferretería, dependerá de varios factores:

- Tiempo de Ejecución
- Memoria utilizada
- Distribución de Datos
- Condiciones de trabajo

En base al tiempo de ejecución y memoria utilizada, se prefiere utilizar el algoritmo KMeans, ya que para este algoritmo, una vez ubicados los centroides, solo se requiere realizar 4 cálculos (uno por cada categoría) por pieza, por lo que el tiempo de ejecución y la memoria utilizada son pequeños. En comparación con el algoritmo KNN, este algoritmo requiere realizar cálculos para cada elemento del Data Base, por lo que al tener un Data Base grande, el tiempo de ejecución y la memoria utilizada serán mayores en comparación a lo utilizado por KMeans.

Por otro lado, al considerar la distribución de los datos(qué tan agrupados están) y condiciones de trabajo, se prefiere utilizar el algoritmo KNN, ya que si se tiene un buen agrupamiento de datos, las predicciones no se ven afectado por outliers, los cuales pueden estar dado por las condiciones de trabajo, de manera que estos no intervendrán demasiado en la predicción de resultados. Respecto al algoritmo K-Means, vimos qué este es sensible a outliers, de manera que si las condiciones de trabajo se alteran, puede provocar obtener outliers que alterarán la predicción.

De manera que en el contexto planteado, ya que nos interesa tener una predicción con un alto nivel de certeza, aunque requiera un mayor tiempo de ejecución, se prefiere utilizar el algoritmo KNN.

## **BIBLIOGRAFÍA**

### **Libros**

- Inteligencia Artificial. Un Enfoque Moderno por Stuart Russell - Peter Norvig.
- Técnicas y algoritmos básicos de visión Artificial por Grupo de Investigación EDMANS
- Presentación "Reconocimiento de objetos en fotografías" por Cinvestav Tamaulipas.
- Procesado digital de señales - 2: Fundamentos para comunicaciones y control por Eduard Bertran Albertí.

### **Web - Documentación**

- <https://gradio.app/>
- <https://scikit-image.org/>
- <https://www.datacamp.com>
- <https://docs.opencv.org>
- <https://docs.gimp.org>
- <https://www.aprendemachinelearning.com>
- <https://barcelonageeks.com/mahotas-excentricidad-de-la-imagen/>
- <https://programarfacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/>
- <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>
- <https://pythoneyes.wordpress.com/2017/05/22/conversion-de-imagenes-rgb-a-escala-de-grises/>

### **Web - Contenido**

- [https://nanopdf.com/download/planificaci-n\\_pdf](https://nanopdf.com/download/planificaci-n_pdf)
- <https://docplayer.es/91429912-Tema-planificacion-en-inteligencia-artificial-agentes-planificadores.html>
- <https://www.ibm.com/ar-es/topics/computer-vision>
- [https://www.cs.umss.edu.bo/doc/material/mat\\_gral\\_139/Busqueda%20A%20estrella-2.pdf](https://www.cs.umss.edu.bo/doc/material/mat_gral_139/Busqueda%20A%20estrella-2.pdf)
- [http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas\\_alumnos/A-Star/A-Star\(2005-II-A\).pdf](http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/A-Star/A-Star(2005-II-A).pdf)

- <https://www.cs.us.es/cursos/iiia-2006/temas/tema-09-iiia06.pdf>
- <https://www.cs.us.es/cursos/ia1-2003/temas/tema-08.pdf>
- <https://www.boletin.upiita.ipn.mx/index.php/ciencia/669-cyt-numero-55/1293-extraccion-de-bordes-operadores-sobel-prewitt-y-roberts>
- [https://es.wikipedia.org/wiki/Difusi%C3%B3n\\_anisotr%C3%B3pica](https://es.wikipedia.org/wiki/Difusi%C3%B3n_anisotr%C3%B3pica)
- <https://programmerclick.com/article/78881681773/>
- <https://la.mathworks.com/help/images/ref/regionprops.html>
- <https://la.mathworks.com/help/images/pixel-values-and-image-statistics.html>
- [https://es.wikipedia.org/wiki/Algoritmo\\_de\\_Canny](https://es.wikipedia.org/wiki/Algoritmo_de_Canny)
- <https://www.codificandobits.com/blog/underfitting-y-overfitting/>
- [https://es.wikipedia.org/wiki/Momentos\\_de\\_imagen](https://es.wikipedia.org/wiki/Momentos_de_imagen)
- <https://www.themachinelearners.com/algoritmo-knn/>
- <https://www.themachinelearners.com/k-means/>

## ÍNDICE

<b>RESUMEN</b>	<b>2</b>
<b>INTRODUCCIÓN</b>	<b>3</b>
<b>ESPECIFICACIÓN DEL AGENTE</b>	<b>4</b>
Tipo de Agente	4
REAS	4
Propiedades del Entorno	5
<b>DISEÑO DEL AGENTE</b>	<b>6</b>
Reconocimiento de objetos	6
Desarrollo del sistema reconocimiento de objetos	6
Breve descripción de objetos de estudio	7
Adquisición de la imagen con cámara convencional	8
Transformación	11
Adaptación	14
Preprocesamiento	19
Filtración	20
Segmentación	23
Extracción de Rasgos	25
Reducción de Dimensionalidad	29
Planificación con lenguaje STRIPS	31
Búsqueda A Estrella (A*)	33
<b>CONTEXTUALIZACIÓN</b>	<b>36</b>
Base de Datos	36
KNN	36
KMeans	38
<b>RESULTADOS</b>	<b>39</b>
Evaluación del Rendimiento	39
Análisis de Resultados	44
<b>CÓDIGO</b>	<b>47</b>
Etapa 1: Procesamiento de Imágenes	47
Etapa 2: Pasos que debe realizar el robot para alcanzar otro orden de apilado	48
Etapa 3: Transportar las cajas desde un punto A a un punto B	48
<b>EJEMPLO DE APLICACIÓN</b>	<b>49</b>
Descripción de la tarea	49
Desarrollo de Etapa 1: Procesamiento de imágenes de las piezas representativas de cada caja	49
Desarrollo de Etapa 2: Pasos que debe realizar el robot para alcanzar el nuevo orden de apilamiento de cajas	54
Desarrollo de Etapa 3: Transportar las cajas desde un punto A a un punto B a través del Laberinto	55
	63

<b>CONCLUSIONES</b>	<b>57</b>
Conclusiones Generales	57
Ventajas y Desventajas	58
Posibles mejoras a futuro	58
Aplicaciones	59
Recomendaciones para la aplicación	59
<b>BIBLIOGRAFÍA</b>	<b>61</b>
<b>ÍNDICE</b>	<b>63</b>

El proyecto está alojado en un repositorio en el sitio de GitHub, con el fin de facilitar las versiones de cada uno de los programas desarrollados. ([Ver Repositorio](#))