



UNCUYO  
UNIVERSIDAD  
NACIONAL DE CUYO



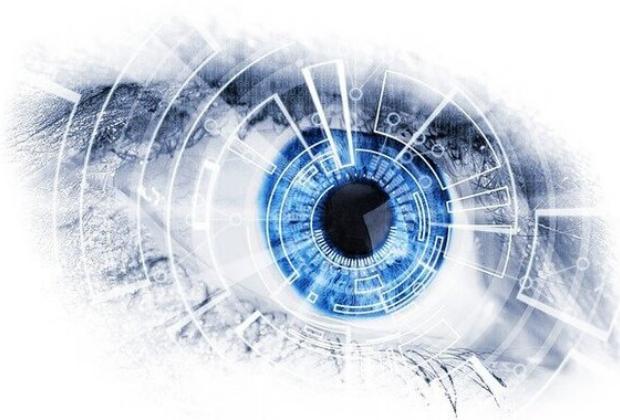
FACULTAD  
DE INGENIERÍA

# Trabajo final

## Inteligencia Artificial I

Año 2022

## Visión Artificial



**Alumno:** Luna Casabene, Juan Ignacio  
**Prof. Titular:** Dra. Ing. Selva S. Rivera



# Índice

<b>1.Resumen</b>	<b>3</b>
<b>2.Introducción</b>	<b>3</b>
2.1 Definición y objetivo de la visión artificial	3
2.2 Etapas elementales de la percepción	4
2.3 Problema a resolver	5
<b>3.Especificaciones del agente</b>	<b>6</b>
3.1 Tipo de agente	6
3.2 REAS	7
3.3 Propiedades del entorno de trabajo	8
3.3.1 Clasificación de cajas	8
3.3.2 Orden del apilamiento de cajas	9
3.3.3 Trayectoria de punto A al punto B	10
<b>4.Diseño del agente</b>	<b>11</b>
4.1 Descripción general y adquisición de la imagen	11
4.2 Preprocesamiento y segmentación de las imágenes	12
4.3 Extracción de características o propiedades	16
4.3.1 Aproximación polinomial	16
4.3.2 Momentos invariantes de Hu	17
4.4 Clasificación	18
4.4.1 Algoritmo de agrupamiento no supervisado KMEANS	18
4.4.2 Algoritmo de agrupamiento supervisado KNN	20
4.5 Interpretación	21
4.6 Planificación de reordenamiento codificada en STRIPS	22
4.7 Trayectoria de posición A a posición B	23
<b>5.Código</b>	<b>26</b>
5.1 Diagrama UML	27
5.2 Main	27
5.3 Consola	28
5.4 Imagen	31
5.5 Database	34
5.6 Kmeans	37
5.7 Knn	43
5.8 A-estrella	51
5.9 STRIPS	62
5.9.1 Domain	62
5.9.2 Problem	64
<b>6.Ejemplo de aplicación</b>	<b>65</b>
<b>7.Resultados</b>	<b>69</b>
<b>8.Conclusiones</b>	<b>69</b>
<b>9.Bibliografía y/o referencias</b>	<b>71</b>



## 1. Resumen

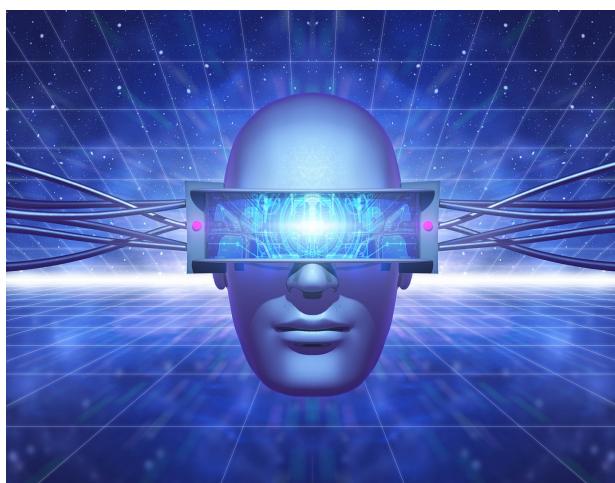
Para el trabajo final de Inteligencia Artificial I se nos plantea el desarrollo de software de un robot que debe identificar mediante visión artificial el contenido de cuatro cajas que se encuentran apiladas. Cada caja contiene piezas distintas, estas son: Arandelas, Tuercas, Clavos y Tornillos. Además debe ser capaz de reordenar el apilamiento de cajas inicial a otro pactado, así como transportar este grupo de cajas de un punto A a un punto B del aula. Los algoritmos utilizados para realizar el agrupamiento o clasificación de las imágenes son el algoritmo de agrupamiento no supervisado **KMEANS** y el algoritmo de agrupamiento supervisado **KNN**. Para planificar la reorganización de las cajas se utilizó lenguaje **STRIPS**. Finalmente también se utilizó el algoritmo de búsqueda **A\* (A estrella)** para trazar el camino del robot de A a B.

Los resultados fueron satisfactorios, se logró una buena detección de características de las imágenes y un buen agrupamiento con ambos algoritmos, lo que llevó a una clasificación exitosa en la mayoría de las pruebas que se realizaron. Los algoritmos de búsqueda y la planificación también funcionaron correctamente y cumplieron su cometido.

## 2. Introducción

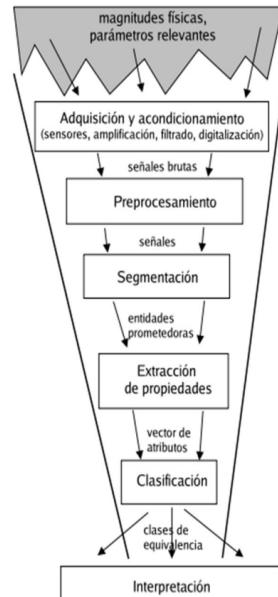
### 2.1 Definición y objetivo de la visión artificial

La visión artificial es una disciplina perteneciente a la Inteligencia Artificial que se encarga de observar y analizar imágenes reales con el objetivo de arrojar un tipo de información específica, cuyos datos son posteriormente asimilados y procesados por un conjunto de programas informáticos. Esta tecnología sigue un principio parecido al funcionamiento del cerebro y ojo humano, el cual detecta y procesa una imagen para comprenderla.





## 2.2 Etapas elementales de la percepción



La visión artificial funciona a través de 6 etapas elementales:

1. **Adquisición y Acomodamiento de la imagen:** Las magnitudes físicas (sonidos, imágenes, etc.) se transforman mediante sensores en señales eléctricas que una vez filtradas, amplificadas y digitalizadas pueden procesarse computacionalmente.
2. **Preprocesamiento:** Luego de adquirir la imagen, el elemento es sometido a una fase de preproceso que tiene como objetivo eliminar el ruido, afinar los bordes y límites que existen en la imagen, contrastar la imagen para resaltar los colores y que su visualización sea más precisa, pulir algunos aspectos generales de la imagen y, por último, someter la imagen a un proceso de transformación para llevarla a otro espacio representativo.
3. **Segmentación:** Sirve para encontrar dentro de la información los elementos individuales que parecen estar dotados de significado. Pueden clasificarse en:
  - **Temporales:** se aíslan fragmentos de la secuencia de datos recibidos a lo largo del tiempo (ej: micro fonemas en voz).
  - **Espaciales:** el conjunto de datos en un instante contiene varias subunidades relevantes (varios objetos en una escena)



4. **Extracción de características o propiedades:** Se calculan propiedades o atributos de cada entidad segmentada que deben ser, idealmente, discriminantes de las diferentes clases de interés e invariantes a todas sus posibles versiones.
5. **Clasificación:** En esta etapa se decide la categoría más probable (dentro de un conjunto preestablecido) al que pertenece cada observación sensorial elemental caracterizada por su vector de características. En este apartado se utilizan conocimientos propios de la inteligencia artificial, como las redes neuronales, los sistemas expertos, la lógica difusa o borrosa, o en nuestro caso particular, algoritmos de agrupamiento supervisados y no supervisados, como lo son el KNN y el KMEANS respectivamente. Básicamente se identifica y clasifica los objetos contenidos en la imagen. Se reconocen los elementos contenidos en una imagen, y se comprende la conexión existente entre cada unidad, otorgándole sentido a dicha composición visual.
6. **Interpretación:** Los conceptos obtenidos en la etapa anterior se organizan en una estructura espacio-temporal requerida por la aplicación. En este punto se decide qué hacer con la información obtenida.  
Se clasifica de 2 maneras: de forma **humana o manual**, cuyo significado es actuar con mecanismos propios en base a la información suministrada por la visión artificial, o de **forma automática**, dejando que un sistema obtenga todos los datos y que, dependiendo de la información procesada, tener un comportamiento en consecuencia.

Todas estas etapas aportan a una base de conocimiento, la cual utilizamos para nuestra interpretación.

## 2.3 Problema a resolver

Como se indicó anteriormente en el resumen, el problema que se nos presenta es el desarrollo de software de un robot que debe identificar mediante visión artificial el contenido de cuatro cajas que se encuentran apiladas. Cada una de estas cajas contiene piezas distintas, estas son: Arandelas, Tuercas, Clavos y Tornillos.

En un momento determinado el robot debe tomar imágenes de cada una de las cajas mediante su cámara integrada. Luego mediante un filtrado, amplificación y digitalización de las imágenes, enviar estas al software interno del robot, que nos permitirá realizarle el procesamiento computacional correspondiente.

Una vez las imágenes están cargadas en la memoria del robot, son sometidas a cierto preprocesamiento y segmentación, para la posterior extracción de características que nos permitirá lograr la clasificación que nos plantea el problema, así como el orden de apilamiento de las cajas.



Esto nos lleva al segundo punto de la consigna, se debe implementar el dominio y problema de planificación, codificado en lenguaje **STRIPS** (**Stanford Research Institute Problem Solver**), para lograr que el robot trace un plan para llevar el apilamiento de las cajas de un estado inicial, a un estado final pactado.

Finalmente el agente deberá ser capaz de llevar dichas cajas de un punto a otro del aula, para lo que tendrá que utilizar un algoritmo de búsqueda o pathfinding A\* (A estrella)

El trabajo que a continuación se expone es el resultado de una intensa investigación a lo largo de papers, documentación, foros, etc. Todo esto en conjunción con un arduo proceso iterativo en el que se probaron muchas implementaciones, algunas exitosas y otras no tanto, dan lugar a M.A.T.E.C.I.T.O ®, un software integro e intuitivo que nos permite cumplir con la meta propuesta en este trabajo.

## 3. Especificaciones del agente

### 3.1 Tipo de agente



Por definición un agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores. Así también, un agente racional es uno que en cada posible secuencia de percepciones, emprende aquella acción que supuestamente maximiza su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.

Analizando el robot con su respectivo software implementado, considero que este cumple con las características necesarias para ser considerado un **Agente Racional**. Partiendo de esta base considero que

el tipo de agente con el que tratamos es un **Agente basado en objetivos**.

Mi fundamento de esta respuesta se basa en que mantenemos la capacidad que tienen los agentes reactivos simples de analizar un momento particular de su entorno, y actúa en consecuencia de sus percepciones; tiene la capacidad de almacenar información o



mantener un estado interno, que le permite seguir el rastro de aspectos del mundo que no son evidentes según las percepciones actuales (como lo es la base de datos con la que contrasta las características extraídas de las imágenes) propio de los agentes reactivos basados en modelos; sin embargo la diferencia clave en la que radica mi consideración, es el hecho que nuestro agente no solo actúa en consecuencia de simples reglas de condición-acción y almacena datos anteriores, si no que siempre tiene un objetivo concreto en cada una de las etapas de su accionar, y actúa en consecuencia. En un primer momento su objetivo es poder identificar y clasificar las distintas cajas, así como su orden de apilamiento. Luego su objetivo es llevarlo de un orden inicial a otro pactado, por lo que establece una ruta de planificación que le permite lograr tal objetivo. Y finalmente tiene como objetivo el desplazarse de un punto a otro, para lo que se le pasa justamente la meta a alcanzar y el programa mediante algoritmo A\* logra encontrar el camino más corto y trazar el camino mediante backtracking.

### **3.2 REAS**

Agente	Medidas de rendimiento	Entorno	Actuadores	Sensores
Robot IA	<b>Maximizar:</b> -Calidad en la obtención de puntos de contorno -Calidad en los datos extraídos -Rapidez y eficiencia en el procesamiento de imágenes -Rapidez y eficiencia en la clasificación de las imágenes  <b>Minimizar:</b> -Ruidos propios de las imágenes -Clasificaciones erróneas -Costo del camino en los planes de apilamiento -Distancias de punto A a B -Tiempos de búsqueda de caminos posibles (tanto para la movilidad del punto A al punto B, como en la búsqueda de planes de apilamiento)	-Cajas -Aula -Mesas -Sillas	-Display de consola gráfica o interfaz (GUI). -Teclado, mouse, botones. -Gripper. -Iluminación	-Cámara. -Sensores ultrasonido para detección de distancias. -Sensores de presión en el gripper. -GPS



### 3.3 Propiedades del entorno de trabajo

Luego de un detenido análisis de los alcances del robot y su interacción con el medio que lo rodea, he decidido separar el análisis del entorno en 3 partes, ya que entiendo que dependiendo la tarea que tiene que hacer el robot, la interpretación del entorno es distinta.

#### 3.3.1 Clasificación de cajas

Entorno de trabajo: Cajas y Base de datos	
Parcialmente observable	Considero que el entorno del agente es parcialmente observable ya que, si bien podemos considerar que el agente puede ver la totalidad de las cajas apiladas a la hora de tomar las imágenes y analizarlas, debemos tener en cuenta como parte del entorno a la base de datos entregada al programa, ya que esta puede interpretarse como el "estado interno" del agente, donde este contrasta la información que ya conoce, con la información nueva y de esta manera mediante algoritmos de agrupamiento clasificar las imágenes. Considerando a la base de datos como parte del entorno, y dado que el sensor externo (la cámara) no puede captar a los elementos de esta, ya que o bien son otras piezas o bien la misma en un tiempo anterior, es que considero que el entorno es parcialmente observable.
Determinista	El entorno del agente es determinista debido a que el siguiente estado del medio está totalmente determinado por el estado actual y las acciones ejecutadas por el agente. El único grado de aleatoriedad que puede tener el agente es en el caso de que el algoritmo de agrupamiento que se utilice sea Kmeans, en ese caso dado que los centroides se determinan aleatoriamente, existe cierto grado de incertidumbre, pero esto no forma parte del entorno del agente en sí mismo, si no más bien del proceso interno de clasificación de las imágenes, por lo que considero que el entorno si es determinista.
Secuencial	El entorno del agente es secuencial, debido a que según el ángulo, iluminación, calidad de la cámara, es la imagen que tomará dicho sensor, y de esto dependerá la acción posterior
Dinámico	Por razones similares a las expuestas en el punto anterior, considero que el entorno en este caso es dinámico, debido a que a medida que transcurre el tiempo las condiciones del ambiente, como pueden ser la iluminación, el estado de los sensores, etc, pueden variar, por lo que el entorno es dinámico.



Discreto	Existe un número concreto de percepciones y acciones claramente definidos, en este caso las 4 imágenes que el agente toma serían las percepciones.
Agente individual	En nuestro entorno no interviene más de un agente, por ende consideramos simplemente un agente individual.

### 3.3.2 Orden del apilamiento de cajas

Entorno de trabajo: Cajas y su orden de apilamiento	
Totalmente observable	Considero que el entorno en este caso es totalmente observable, ya que el agente tiene a su disposición la cantidad de cajas, el orden de apilamiento inicial y una vez dado, el orden de apilamiento final. Los sensores (cámara, sensores de presión en gripper) proporcionan toda la información relevante.
Determinista	El entorno es determinista debido a que el estado siguiente se obtiene a partir del actual y de las acciones del agente. No hay componentes azarosos.
Secuencial	El entorno es secuencial, debido a que el plan que trazará el agente dependerá siempre del orden de apilamiento inicial y de los distintos órdenes que vayan surgiendo a medida que el agente lleva a cabo las acciones trazadas en la planificación.
Estático	El entorno es estático, ya que no sufre cambios mientras el agente está razonando. Los únicos cambios que sufre son causados por el mismo agente cuando está actuando.
Discreto	Las percepciones son simplemente las distintas posiciones que pueden tomar cada una de las 4 cajas, por lo que es un entorno discreto
Agente individual	En nuestro entorno no interviene más de un agente, por ende consideramos simplemente un agente individual.



### 3.3.3 Trayectoria de punto A al punto B

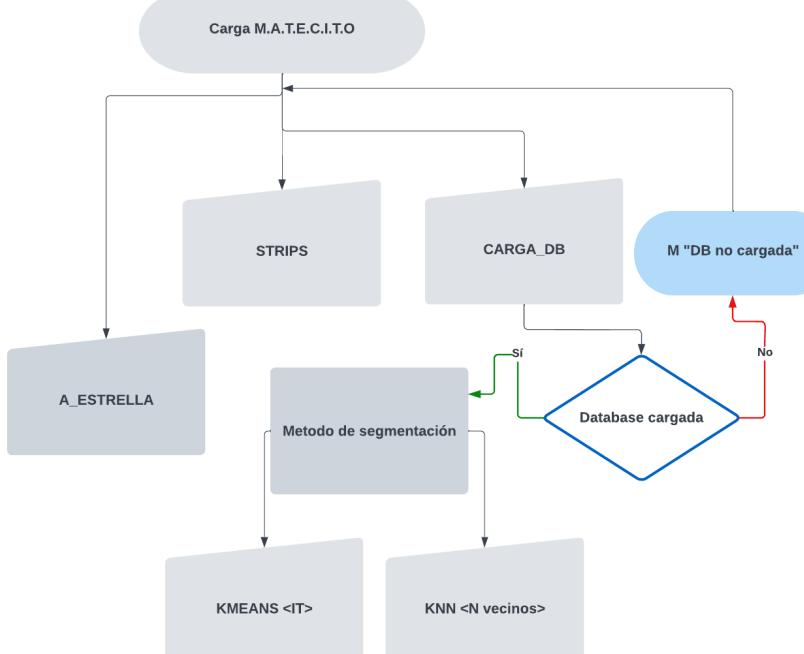
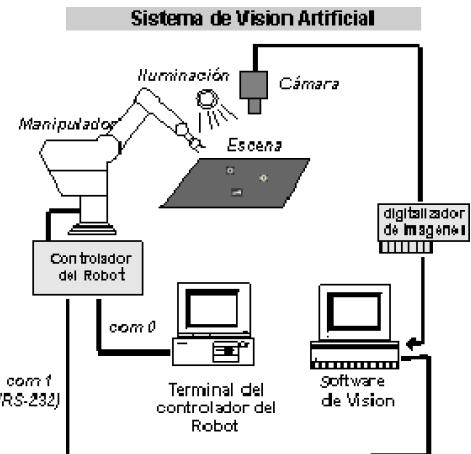
Entorno de trabajo: Aula, mesas y sillas	
Totalmente observable	El entorno en este caso es totalmente observable, dado que, como A estrella es un algoritmo de búsqueda informado, debemos calcular el valor $f$ (costo total) en cada casilla del openset hasta llegar al nodo objetivo, ya luego de eso calculamos el camino mediante backtracking, teniendo en cuenta los nodos padre (nodo el cual, producto de su expansión hizo que llegaramos al nodo actual) hasta llegar al nodo inicial. Para lograr hacer este cálculo de $f$ , que a su vez es igual a la suma de la heurística y el costo del camino, necesariamente requerimos de un entorno totalmente observable
Estocástico	El entorno es estocástico, ya que la distribución del laberinto que el robot tendrá que recorrer y los puntos A y B serán aleatorios al momento del examen, lo que me genera incertidumbre y el estado siguiente no dependerá solo de las acciones del agente si no de estas componentes azarosas.
Secuencial	El entorno es secuencial debido a que la posición de la casilla actual donde se encuentra el agente a lo largo de su trayectoria por el camino, dependerá de las casillas que el algoritmo eligió anteriormente para pasar.
Estático	El entorno es estático, dado que no variará a medida que el agente se está desplazando del punto A al punto B.
Discreto	La trayectoria del robot del punto A al punto B queda determinado por un número finito de nodos o casillas que el robot debe recorrer, por lo que el entorno es discreto.
Agente individual	En nuestro entorno no interviene más de un agente, por ende consideramos simplemente un agente individual.



## 4. Diseño del agente

### 4.1 Descripción general y adquisición de la imagen

El software M.A.T.E.C.I.T.O fue diseñado para ser cargado directamente a la memoria del Robot IA. Se conecta por puerto serie con dicho robot y por medio de una interfaz gráfica o consola (GUI) puede ser controlado por el usuario, manejando las tareas a realizar y pasándole los parámetros correspondientes.



La forma en la que se controla dicha GUI es simple e intuitiva:

1. Se carga el software correspondiente a la memoria del Robot.
2. Se ingresa "HELP" para visualizar los comandos disponibles.
3. Se tipea el comando que dará lugar al algoritmo a utilizar.
4. En el caso de querer acceder a los algoritmos de agrupamiento con KMEANS y KNN es necesario primero cargar la base de datos de 20 imágenes, 5 de cada pieza.
5. Una vez cargada la base de datos, podemos acceder al método KMEANS pasandole como parametro el numero de iteraciones de cálculo de los centroides, o en el caso de KNN el número de los K vecinos con los que se pretende trabajar



6. Una vez ingresados los comandos, el programa accede y analiza las imágenes presentes en la carpeta “INPUT”, las procesa, las clasifica y las guarda en las rutas relativas “Output/Kmeans” o “Output/Knn” según el método que se elija.

## **4.2 Preprocesamiento y segmentación de las imágenes**

Para poder ser analizada y clasificada, una imagen debe pasar por una meticulosa serie de procesos que a continuación se expondrán, para finalmente entregarnos los resultados que buscamos. Para mostrar los resultados de cada procedimiento tomaremos como pieza de referencia un tornillo (las imágenes de cada pieza pasan por el mismo procedimiento y puede visualizarse al ejecutar el código).

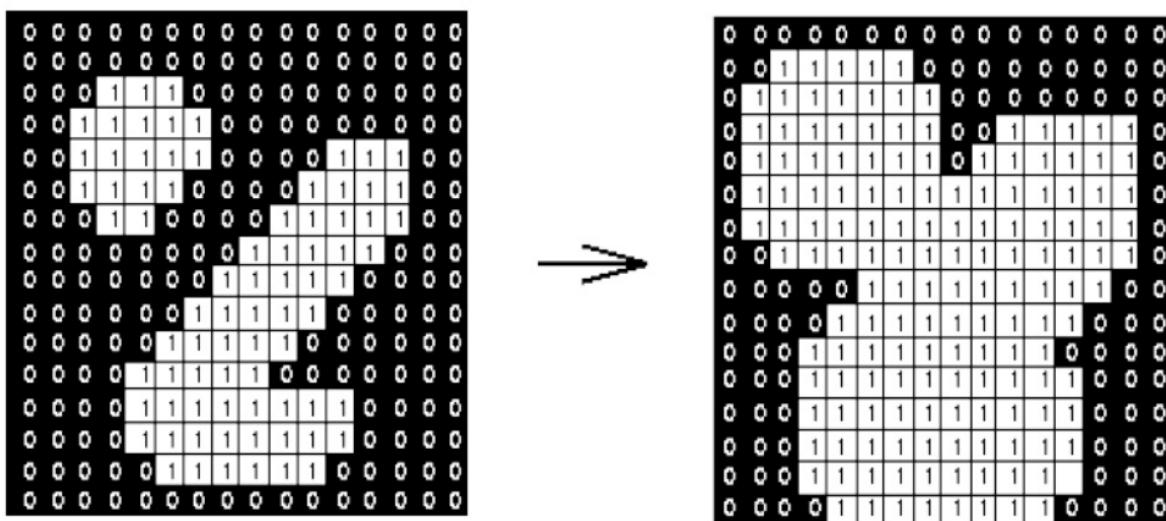
La primera etapa de nuestro preprocesamiento es **reajustar** la imagen a un tamaño fijo (para que cada imagen no se muestre en pantalla en su tamaño por defecto), en nuestro caso 512x512 pixeles.

Todo pixel de una imagen a color tiene por defecto “3 canales”, un canal R(red), un canal G(Green) y un canal B(Blue), lo que en conjunto se conoce como **RGB**. RGB es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios. Computacionalmente, es frecuente que cada color primario se codifique con un byte (8 bits), por lo que la intensidad de cada una de las componentes o canales se mide según una escala que va del 0 al 255. Para poder trabajar nuestra imagen, debemos trabajar con un solo canal cuya intensidad varía dentro de esos números, por ende pasamos nuestra imagen a **escala de grises**, logrando así tener pixeles de 1 solo canal, con intensidades que varian entre 0 y 255 donde 0 es el negro y el 255 el blanco; entre dichos valores tenemos los grises.





Luego, sometemos a las imágenes a métodos de **erosión y dilatación**. En el primer método los pixeles con intensidades más brillantes son convertidos a intensidades más oscuras, se utiliza para disminución de ruidos en la imagen. El segundo método hace lo contrario, y sirve justamente para ensanchar los bordes brillantes y acortar las partes más oscuras, con el objetivo de agrandar los bordes en caso de que la función de erosión haya borrado partes importantes de la imagen. Cabe destacar que esto se logra mediante un filtro espacial con un kernel, el cual no es más que una matriz identidad de dimensión 3x3.



Effect of dilation using a  $3 \times 3$  square structuring element

Activate Wi  
Settings

Para continuar con la implementación del preprocesamiento de las imágenes, se optó por utilizar **filtros frecuenciales**, ya que especialmente interesante su funcionamiento y la teoría detrás de los mismos.

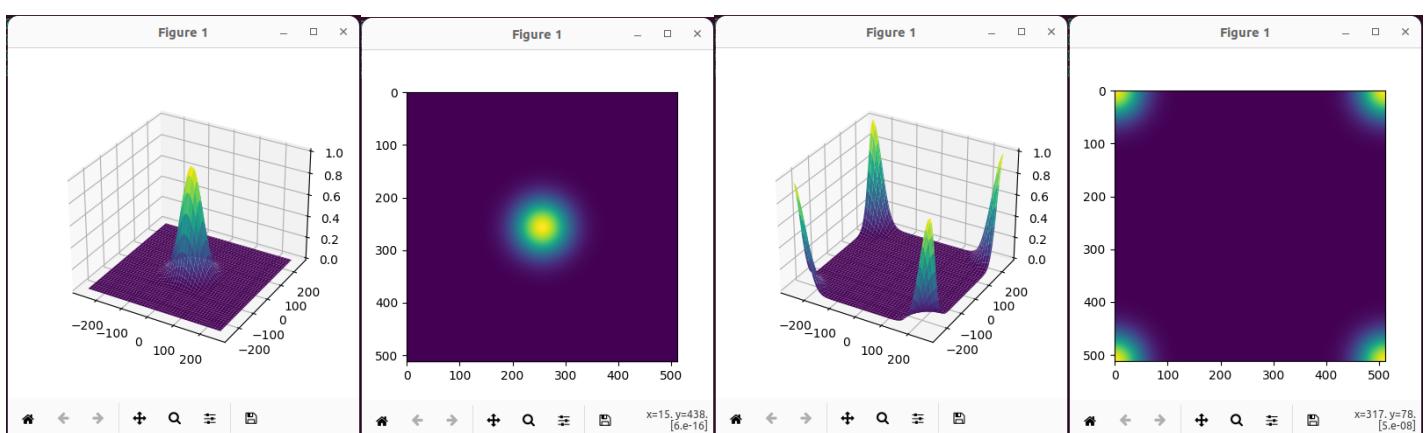
En el caso particular de mi implementación, opté por la utilización de un **filtro frecuencial pasa bajo Gaussiano** para la eliminación de ruidos en la imagen en una primera instancia.

$$G = e^{-\frac{1}{2} \left( \frac{R^2}{\sigma^2} \right)}$$

$$G = ke^{-\frac{1}{2} \left( \frac{x^2 + y^2}{\sigma_x^2 + \sigma_y^2} \right)}$$

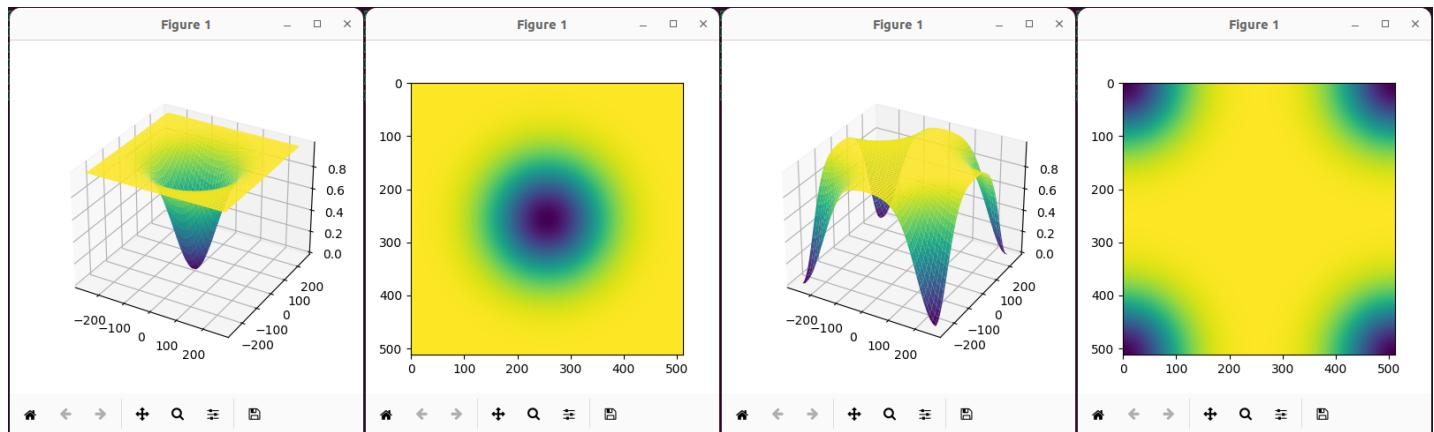
$$R = \sqrt{x^2 + y^2}$$

$$\sigma_x = \sigma_y = \sigma$$





Luego, como segunda instancia utilice un **filtro frecuencial pasa alto**, cuya función en el dominio de la frecuencia es la diferencia de 1 menos la función del filtro pasa bajo gaussiano anteriormente mostrada.



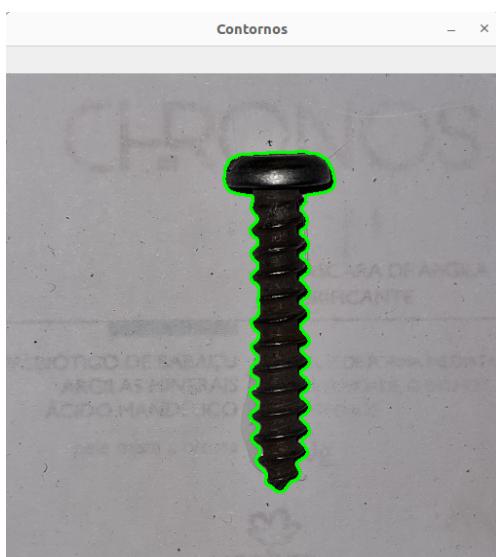
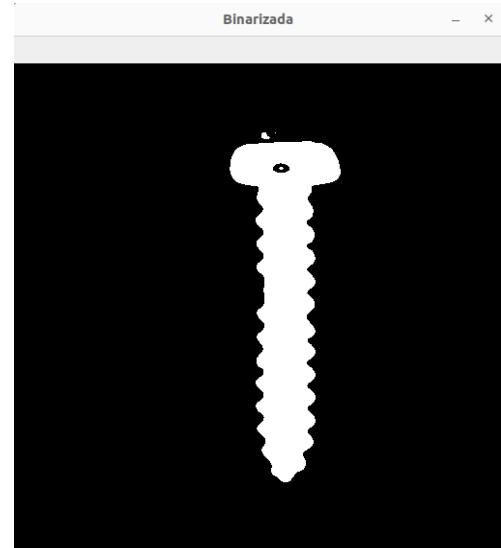
Luego mediante el producto de la función filtro en el dominio frecuencial (habiéndolo aplicado primero la función fftshift para reacomodar el origen del centro a las esquinas) con la transformada de Fourier en 2D de la imagen, lo cual es equivalente a realizar la convolución entre ambas funciones en el dominio espacial, y posteriormente aplicando la antitransformada de Fourier en 2D a dicho producto, obtenemos las imágenes filtradas.





El siguiente paso consiste en realizar una **Binarización** de la imagen. Esto consiste en pasar la imagen de una escala de grises, con pixeles de 1 solo canal de intensidad que varia entre 0 a 255, a pixeles cuyos únicos valores posibles de intensidad sean 0 (negro) o 1 (blanco).

Lo anteriormente nombrado se logró mediante un método conocido como "**Thresholding**" (o método del valor umbral), bajo ciertas configuraciones de implementación, tiene un funcionamiento muy simple. Básicamente se setea un **valor umbral**, el cual es un valor de intensidad entre 0 y 255, para luego analizar el valor de intensidad de cada pixel. Si el valor de intensidad del pixel está por encima del umbral este adquiere el valor 255 (o 1 en binario), si está por debajo adquiere el valor 0. De esta manera obtenemos una imagen donde solo hay blanco o negro, no hay grises, y donde se ve claramente una mejor diferenciación de fondo y bordes, permitiéndonos avanzar al último paso. Para elegir el valor umbral adecuado para una imagen nos serviremos del **histograma**. En un histograma se ve la frecuencia con que aparece cada valor de gris (cantidad de pixeles que tienen ese valor de intensidad). En el eje de coordenadas horizontal se tendrá los distintos valores de intensidad de 0 a 255, y en el eje "y" tenemos la frecuencia con que aparece cada tono de gris en la imagen.



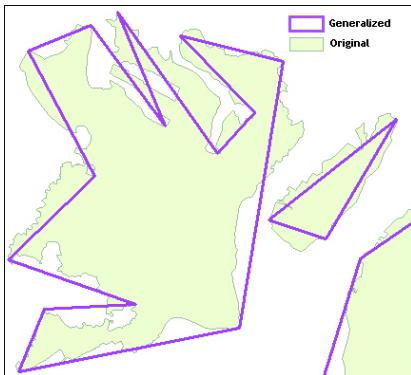
Finalmente debemos realizar la segmentación, que recordando lo anteriormente expresado, servía para encontrar dentro de la información los elementos individuales que parecen estar dotados de significado. Para ello utilizamos algunos métodos y módulos de programación, que a partir de una imagen binarizada me generan los **puntos de contorno**, los cuales luego de una interpolación se convierten en el **contorno** mismo de la imagen, permitiendo calcular el perímetro de la pieza, el área, los momentos invariantes de Hu, etc.



## 4.3 Extracción de características o propiedades

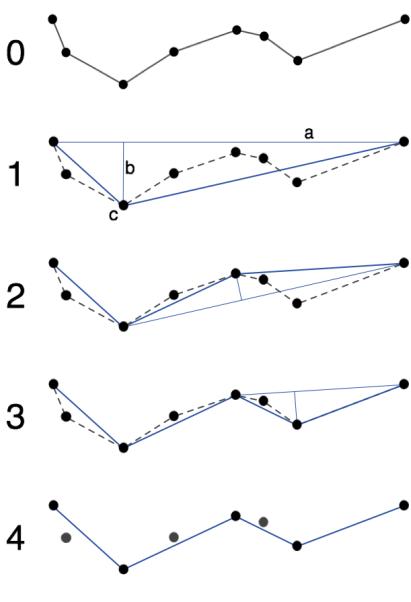
Para la extracción de características o propiedades se decidió trabajar con 3 parámetros: aproximación polinomial, 1er y 6to momento invariante de Hu.

### 4.3.1 Aproximación polinomial



La aproximación polinomial es un método que básicamente consiste en tomar el contorno de una imagen y quedarse solamente con algunos puntos del mismo, para luego trazar rectas que conectan dichos puntos y aproximan la figura. Para lograr dicho objetivo utilice una implementación basada en el algoritmo de **Ramer–Douglas–Peucker (RDP)**, algoritmo utilizado para reducir el número de puntos utilizados en la aproximación de una curva.

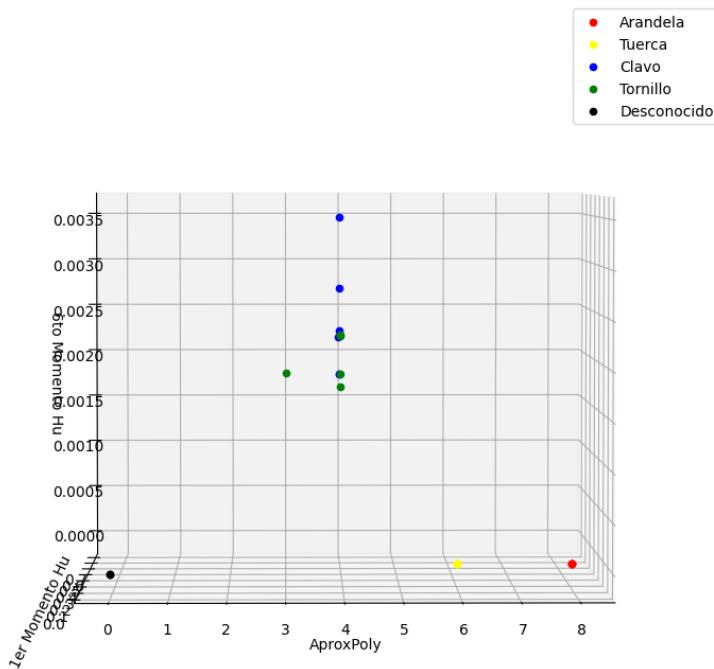
El algoritmo construye una aproximación de la curva inicial mediante un proceso recursivo. Se toma como solución inicial el segmento que une los dos puntos extremos de la curva. Entonces, se busca el punto más alejado de dicho segmento (peor punto). Además se establece un **valor  $\epsilon$  conocido como umbral de distancia**, el cual en nuestro caso será el diámetro de la figura original afectado por un cierto factor.



- Si el peor punto está más cerca del segmento que el umbral de distancia  $\epsilon$ , entonces se termina el proceso. Es seguro que el resto de puntos de la curva están a menor distancia que el umbral  $\epsilon$ , y por lo tanto todos los puntos de la curva (salvo los extremos) pueden ser descartados.

- Si el peor punto está más alejado que  $\epsilon$ , entonces ese punto debe permanecer en la simplificación. El algoritmo hace dos llamadas recursivas a sí mismo para calcular la aproximación de dos curvas de menor longitud. Una con los puntos entre el primer y el peor punto y otra con los puntos entre el peor punto y el punto final de la curva.

Cuando se completa la recursión la nueva curva puede ser generada a partir de los puntos que han permanecido tras haber aplicado el algoritmo.



Observamos que, con la aproximación polinomial logramos una buena diferenciación entre tornillos y clavos por un lado, y tuercas y arandelas por el otro.

A su vez y lo más importante (ya que veremos que la diferenciación anteriormente mencionada también se da con los momentos de Hu), es que con esta característica logramos una diferenciación importante entre las arandelas y las tuercas, que con los momentos de Hu, por una cuestión de escalas no seríamos capaces de apreciar.

#### 4.3.2 Momentos invariantes de Hu

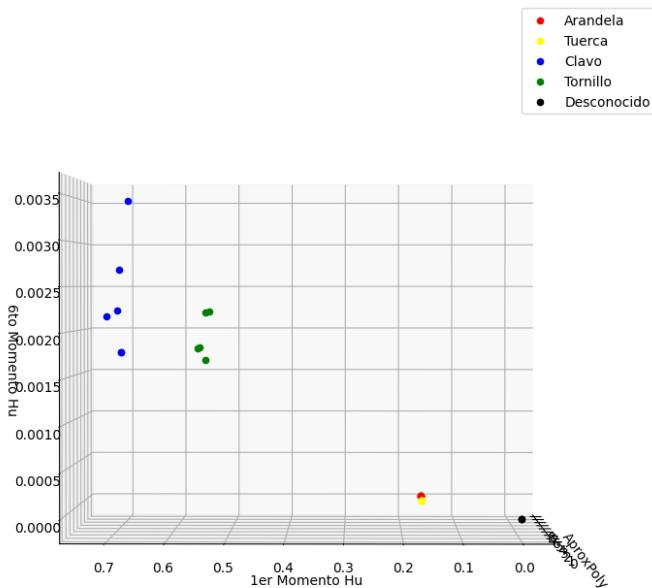
Un momento de imagen es cierto promedio ponderado particular de las intensidades de los píxeles de una imagen. El dominio de una imagen es discreto, y está dado por las coordenadas de sus píxeles. Tales coordenadas se expresan con números enteros. Interpretando las intensidades de los píxeles como  $I(x,y)$ , los momentos de una imagen se calculan de la siguiente manera:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

Si bien existen distintos momentos que podía utilizar como característica para extraer, como por ejemplo los momentos centrales, los cuales son invariantes a las traslación, o los momentos normales, los cuales son invariantes a la escala y la traslación; necesitaba un momento que fuera invariante a estos fenómenos, pero sobre todo a la rotación, teniendo en cuenta que las cajas apiladas pueden tener sus piezas rotadas a la hora de analizar las imágenes, es por eso que me decidí por utilizar los **Momentos invariantes de Hu**, los cuales son invariantes a la traslación, escala y rotación.



Para mi implementación opte por utilizar el **1er y 6to momento de Hu**, debido a que luego de ciertas pruebas con distintos momentos, observé que estos se acomodaban mejor a mi escala para graficar.



Observamos que a partir del 1er momento de Hu logramos una buena diferenciación de lo que son tornillos y clavos por un lado, y arandelas y tuercas por otro. A su vez también logramos una diferenciación entre los clusters de clavos y tornillos.

Para el caso del 6to momento de Hu, este contribuye a la diferenciación entre los puntos que pertenecen al mismo cluster, evitando así el overfitting (en el caso de arandelas y tuercas no es apreciable debido a la escala)

## 4.4 Clasificación

### 4.4.1 Algoritmo de agrupamiento no supervisado KMEANS

K-means es un método de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos o “clusters” en el que cada observación pertenece al grupo cuyo valor medio es más cercano. El nombre K-means proviene del hecho de que este método representa cada uno de los “clusters” por la media (o media ponderada) de sus puntos, es decir, por su centroide. Como se trata de un algoritmo heurístico, no hay ninguna garantía de que convergen al óptimo global. El resultado puede depender de los grupos iniciales. Al ser un algoritmo no supervisado, la información presente en la base de datos no se encuentra previamente clasificada.

En nuestro caso trabajamos con un K constante e igual a 4, ya que son 4 los clusters que nosotros queremos agrupar, uno por cada caja. Luego generamos aleatoriamente los 4 centroides, que podrían ser parte de los puntos que considero para el agrupamiento o bien puntos al azar en la gráfica, sin embargo en el caso puntual de mi implementación, elegimos aleatoriamente como centroides a los puntos correspondientes a 1 imagen de cada grupo de piezas de la base de datos (20 imágenes en total, 5 imágenes de cada pieza), y adicionalmente pre clasificamos esos centroides con la etiqueta que le corresponda, que si bien no es intrínseco del algoritmo Kmeans ya que se supone no hay una preclasificación

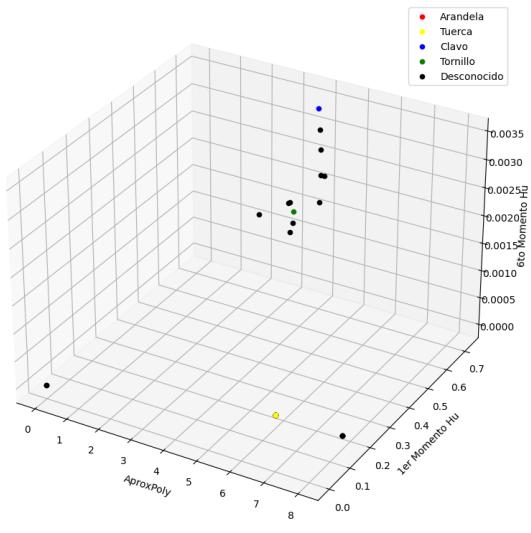


dado que es “no supervisado”, esto nos sirve simplemente para que, una vez realizada la agrupación por medio del algoritmo, se etique al cluster con el nombre de la pieza que le corresponde.

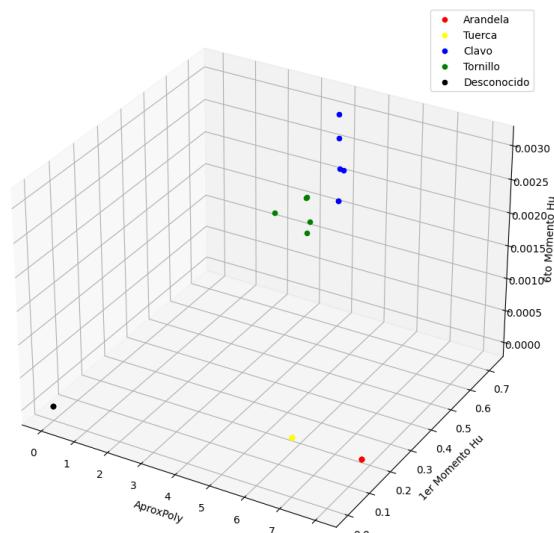
Podría parecer que esto corrompe la esencia misma del algoritmo, sin embargo esto no es así, porque a la hora de comenzar con el agrupamiento una vez calculados los centroides, el algoritmo no sabe a qué categoría corresponde cada punto, simplemente verifica la distancia euclídea de cada punto a cada uno de los 4 centroides y según cual haya sido la distancia mínima de cada uno de los puntos respecto a cada centroide, coloca al punto en el cluster que le corresponde, y recién ahí lo etica, según cual haya sido la etiqueta del centroide del cluster correspondiente.

Una vez que todos los puntos son colocados en cada cluster, se recalculan los K=4 centroides, utilizando como criterio de cálculo la media aritmética de los puntos que pertenecen al cluster. Este procedimiento se realiza tantas veces como el usuario requiera, pasando previamente como parámetro el número de iteraciones que se desea realizar.

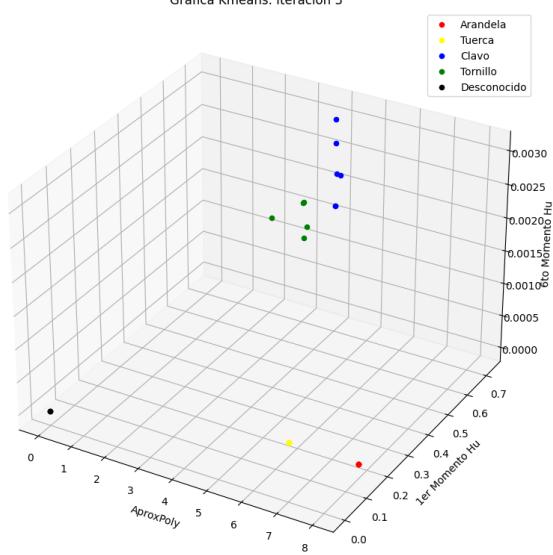
Grafica Kmeans. Iteracion 1



Grafica Kmeans. Iteracion 2



Grafica Kmeans. Iteracion 3

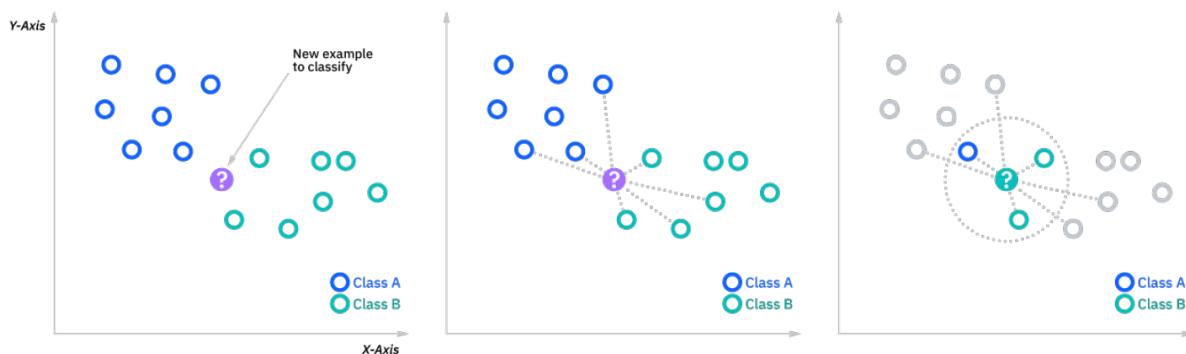


```
+ juani@juani-GS63VR-7RF:... Q - X
[[4, 0.5466837141754873, 0.0016643971024612079]
Iteracion 1
Centroides:
[[8, 0.15919184966750968, -3.341475015987438e-14], [6, 0.1597059724802038, -1.1282678306706147e-11], [4, 0.684968535605407, 0.003468870189038902], [4, 0.5658011821220297, 0.001988189966990194]]
Iteracion 2
Centroides:
[[8.0, 0.15916240934684786, -1.5139714591282156e-15], [6.0, 0.1597867001706275, 7.072321592985247e-15], [4.0, 0.697961863112395, 0.0017497174861831075], [4.0, 0.5603801423296036, 0.0017992592735711175]]
Iteracion 3
Centroides:
[[8.0, 0.15916240934684786, -1.5139714591282156e-15], [6.0, 0.1597867001706275, 7.072321592985247e-15], [4.0, 0.697961863112395, 0.0017497174861831075], [4.0, 0.5603801423296036, 0.0017992592735711175]]
Imagenes guardadas en la carpeta Output/Kmeans
Meteclar>>]
```



#### 4.4.2 Algoritmo de agrupamiento supervisado KNN

El algoritmo de k vecinos más cercanos, también conocido como KNN o k-NN, es un clasificador de aprendizaje supervisado no paramétrico, que utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual. Si bien se puede usar para problemas de regresión o clasificación, generalmente se usa como un algoritmo de clasificación, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro.



Para los problemas de clasificación, se asigna una etiqueta de clase sobre la base de un voto mayoritario, es decir, se utiliza la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinado

Se tienen 3 conjuntos de datos:

- El conjunto de entrenamiento que se utiliza para el aprendizaje del clasificador (en mi implementación sería la base de datos)
- El conjunto de validación
- El conjunto de prueba (datos no empleados durante el entrenamiento)

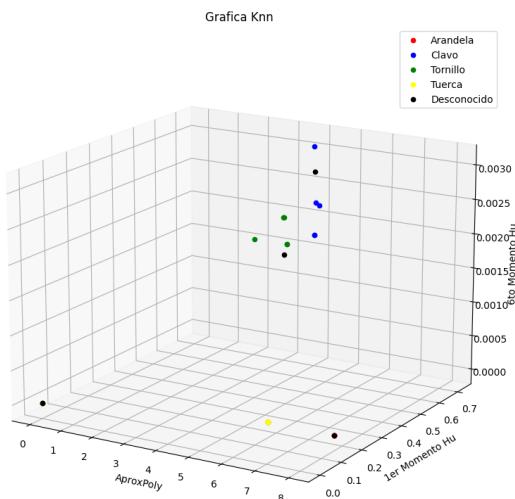
La mejor elección de k depende fundamentalmente de los datos. Generalmente, valores grandes de k reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas.

En mi implementación utilicé un esquema similar al del módulo Kmeans, sin embargo hay ciertas modificaciones, propias del evidente cambio entre ambos algoritmos.

Primeramente, los elementos de la base de datos, al ser un algoritmo supervisado, se encuentran clasificados de antemano.



Luego, se le pasa como parámetro los K vecinos a analizar, y los puntos de las 4 imágenes a clasificar.



De esta manera, para cada punto se calcula la distancia euclídea a todos los demás puntos, y solo nos quedamos con los K puntos que presenten la menor distancia. Para lograr esta implementación, utilizamos un esquema iterativo donde se analiza K veces cuál es la distancia más pequeña. En cada iteración, una vez seleccionado el punto con la distancia más pequeña, se actualiza a un valor grande (1000 en mi caso), para que de esta forma al realizar una nueva iteración, el programa pueda tomar como punto con la mínima distancia, al siguiente punto con la distancia mínima.

Finalmente se evalúa la etiqueta de los K vecinos más cercanos a cada punto y se analiza la frecuencia de los datos. La etiqueta que más veces se repita será la que utilicemos para etiquetar a mi pieza desconocida.

Podría pasar que el análisis no sea concluyente debido a que no predomina ninguna etiqueta frente a otra. En este caso lo que se hace es aumentar en 1 el valor de K y ejecutar nuevamente el algoritmo, entendiendo que, en algún momento, uno de los valores predominan frente a los demás. Esto el algoritmo lo hace automáticamente.

## 4.5 Interpretación

Anteriormente dijimos que la forma de interpretación de la información obtenida una vez clasificadas las imágenes podía clasificarse de 2 formas: de forma **humana o manual**, cuyo significado es actuar con mecanismos propios en base a la información suministrada por la visión artificial, o de **forma automática**, dejando que un sistema obtenga todos los datos y que, dependiendo de la información procesada, tener un comportamiento en consecuencia.

La forma de clasificación en nuestro caso sería **automática**, dado que es el robot el que toma las imágenes de el directorio “Output” presente en su memoria interna y pase a la tarea siguiente, que será el reordenamiento de la pila de cajas a un nuevo orden propuesto (realmente esto no será así debido a que al no haber robot somos nosotros los que interpretaremos el orden inicial y lo cargaremos al programa codificado en STRIPS, por lo que la forma sería humana o manual, pero suponiendo que el robot realmente existe, ese sería el caso)



## 4.6 Planificación de reordenamiento codificada en STRIPS

La “*AI Planning*” es un campo de la inteligencia artificial que explora el proceso de utilizar técnicas autónomas para resolver problemas de planificación y programa. Un problema de planificación es uno en el cual nosotros tenemos un estado inicial, y deseamos transformarlo en un estado objetivo deseado, mediante la aplicación de un conjunto de acciones. Ciertas acciones no pueden ocurrir antes que otras, es por eso que mediante los predicados lógicos, podemos expresar un cierto problema diciendo: “Si esto no ocurre, esto tampoco ocurrirá”

En Inteligencia artificial, **STRIPS** (**S**tanford **R**esearch **I**nstitute **P**roblem **S**olver) es un generador de planes automatizado. El mismo nombre fue utilizado más tarde para referirse al lenguaje formal de las entradas de este generador de planes. Una forma de implementar la planificación en STRIPS y poder programarlo es mediante lo que se conoce como PDDL.

**Planning Domain Definition Language (PDDL)** es una familia de lenguajes que nos permite definir un problema de planificación. Dado que la planificación ha evolucionado, también lo ha hecho el lenguaje usado para describir estos problemas, lo que ha llevado a que hayan varias versiones de PDDL y distintas plataformas que le dan soporte. En nuestro caso utilizaremos el sistema de planificación Fast Downward.

**Fast Downward** es un sistema de planificación clásico basado en la búsqueda heurística. Puede lidiar con problemas de planificación determinísticos generales codeados en el fragmento proposicional de PDDL 2.2.

Algo interesante que pude visualizar al descargar este sistema de planificación para compilar el programa directamente en mi computadora, es que utiliza para hallar los planes a trazar, algoritmos de búsqueda, como lo son el **A\*** o **Búsqueda Voraz primero el mejor**, con la posibilidad de utilizar distintas heurísticas, presentes dentro de la documentación.

En mi caso particular opte por una búsqueda **A estrella**, con **heurística “Blind” o “Ciega”**. La heurística ciega retorna un estimado heurístico basado en el logro de llegar a estados objetivos. El valor estimado puede ser representado en binario con un 0 para un estado objetivo o un 1 para un estado no objetivo.

Para estados no objetivos es probable que elija también la acción que proporcione el mínimo costo del camino para costos del camino no unitarios. Aspectos positivos de esta heurística son su **admisibilidad** (nunca sobre estima el costo de alcanzar el objetivo, el costo estimado siempre será menor o igual al costo mínimo de alcanzar la solución) y **consistencia** ( $h(n) \leq c(n, A, n') + h(n')$ ), de esta manera nos aseguramos que el plan que encuentre el algoritmo sea **óptimo**, es decir, el de menor coste.



Básicamente el problema se trata teniendo en cuenta que el apilamiento de las cajas es simplemente un sistema **LIFO (Last in, First Out)**, en donde tenemos funciones de Pop (Apilar) y Push (Desapilar).

Las acciones definidas en el dominio son **sacar**, **sacar-base**, **poner**, **poner-base**, las cuales están bien explicadas en los comentarios del código que se encuentra más adelante en el informe, pero en rasgos generales:

1. **Sacar (Pop):** Me permite desapilar la caja superior y colocarla a un lado. Esto será posible siempre y cuando la caja a desapilar no tenga otra caja arriba y no sea la caja base.
2. **Poner (Push):** Me permite apilar cualquiera de las cajas desapiladas y colocarla en la parte superior de la pila.
3. **Sacar-base:** Caso especial de “Sacar”, donde la condición que la distingue de esta primera es que la caja sea la caja de la base, y claramente la base donde esta apoyada no debe estar vacía, ya que eso implicaría que no hay caja.
4. **Poner-base:** Caso especial de “Poner”, donde la condición que la distingue de esta primera es que no haya ninguna caja apoyada en la base, y que no se apilara sobre ninguna otra caja, si no sobre la base.

Finalmente, en el problema simplemente se establecen las condiciones del orden inicial en el (:init) y en el (:goal) se establecen las condiciones del orden final al que se quiere llevar.

## 4.7 Trayectoria de posición A a posición B

Llegamos a la recta final del problema planteado para nuestro agente, el cual es desplazarse desde un punto A a un punto B del aula (ambos aleatorios) a través de un laberinto formado por mesas y sillas, cuya distribución inicial también será aleatoria.

Para ello utilizamos el algoritmo de búsqueda o pathfinding **A\* o A estrella**, con **heurística distancia de Manhattan o Taxi Cab**.

El algoritmo A\* se basa en la conjunción de 2 conceptos o métodos:

1. Primeramente trabaja con el **algoritmo de Dijkstra**, también llamado algoritmo de caminos mínimos, el cual es un caso particular del algoritmo de búsqueda Costo uniforme, con la diferencia que en Dijkstra no tenemos nodo objetivo. La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando

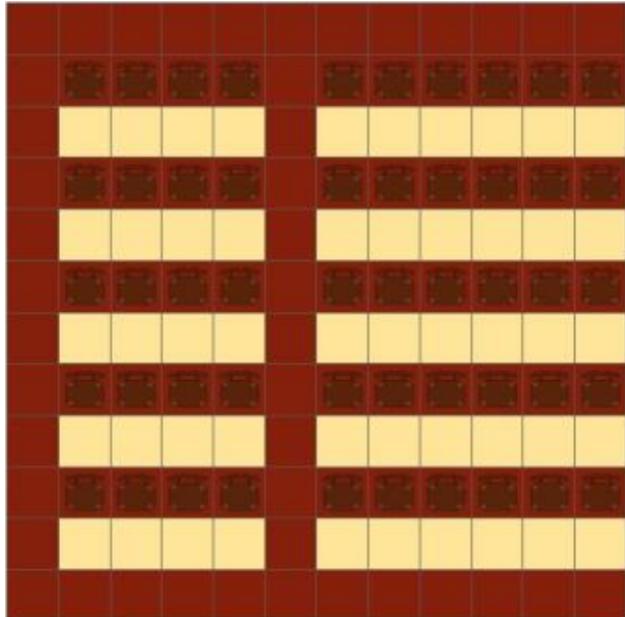


se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene.

2. El segundo concepto que debemos tener presente para entender el algoritmo A\* es el valor  $f(n)$  o costo total del camino, el cual en A\* sera:

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

donde  $g(n)$  es el costo del camino desde el nodo actual al nodo raíz y  $h(n)$  es la distancia (en este caso distancia Manhattan) del nodo actual al nodo objetivo.



El agente se desplaza en un entorno discreto compuesto por distintas casillas de posición. Los casillas rojas representan los espacios por los que el agente puede desplazarse, las casillas blancas representan mesas y las marrones sillas, lo cual a nosotros nos será indiferente en nuestra implementación debido a que a mesas y sillas lo interpretaremos de la misma forma, como obstáculos por donde el agente no puede desplazarse.

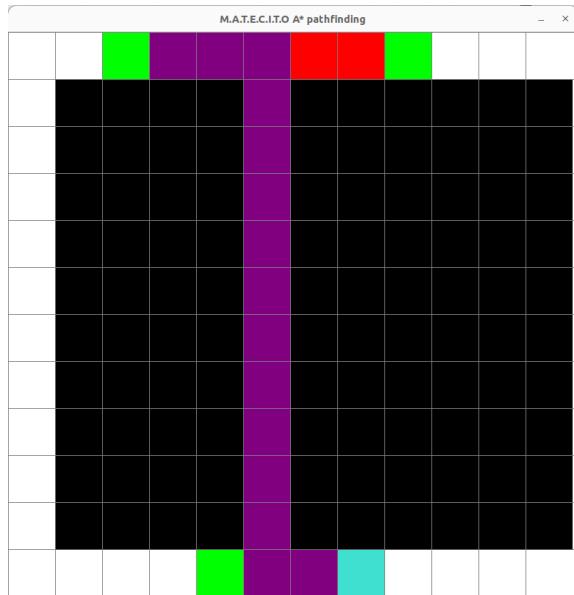
Se realizará una abstracción en la que se interpreta este entorno como una **matriz o grid** de  $12 \times 12$ , donde cada elemento de dicho grid será un objeto instanciado a partir de la clase "Nodo", el cual posee sus atributos y métodos característicos.

En cuanto a la implementación, en un primer momento podemos decir que vamos a estar trabajando con 2 vectores o listas principales, conocidas como **openset** y **closedset**, los cuales almacenan nodos y su valor  $f(n)$ . En el openset encontraremos los nodos y sus valores que actualmente estoy analizando, mientras que en el closedset tendremos los nodos que ya analicé. Una vez que un nodo fue visitado, pasa del openset al closedset. Tendremos también una 3ra lista de importancia, la lista **came\_from**, en la cual cargaremos los "nodos padre", o nodos de los cuales provienen los sucesivos nodos hijos. Esto para luego poder ejecutar el backtracking desde el nodo final al nodo inicial y trazar el camino.

Dicho esto, el flujo de trabajo de la implementación del algoritmo es el siguiente:



1. Antes de iniciar el algoritmo se recorren todos los nodos y dentro de un vector “vecinos” o “neighbours” se colocan todos los nodos que no son obstáculos y por los que el agente puede desplazarse. De esta manera no cometemos el error de incluir alguno de estos nodos en el open set y considerarlo en nuestro procedimiento. Para ello el programa, utilizando la clase nodo, accede a un método que se posiciona en cada nodo y revisa los vecinos de arriba, abajo, izq y derecha. Si estos son de color negro, entonces representan un obstáculo, por ende no los coloca en la lista vecinos. Así para cada nodo.
2. Recorremos nuevamente cada nodo, esta vez solo los que no tienen obstáculos, y le asignamos valor  $g(n)=\text{infinito}$  y por ende  $f(n)=\text{infinito}$ , excepto al nodo inicial, el cual tendrá  $g(n)=0$
3. Ponemos el nodo inicial en el openset
4. El nodo inicial, por ser el nodo raíz tendrá un costo del camino  $g(n)=0$ , por lo que  $f(n)=h(n)$
5. Analizamos cada uno de los nodos vecinos de mi nodo actual (inicial), y entramos a cada uno de ellos de a uno por vez. Comparamos el valor de  $g(n')$  siendo  $n'$  el nodo vecino, con una variable temporal  $g\_temp=g(n)+1$ , que corresponde al costo del camino de avanzar una casilla. Si dicho  $g$  temporal es menor que  $g(n')$  (y lo será ya que la habíamos seteado como infinito al igual que todos los nodos menos el inicial) actualizamos a  $g(n')=g(n)+1$ , luego  $f(n')=g(n')+h(n')$ .
6. Hacemos el paso 5 para todos los nodos, iterando cada vez para colocar en openset los nodos frontera y en el closed set los nodos ya recorridos, hasta que finalmente llegamos al nodo objetivo. Cuando eso pasa la primera etapa del algoritmo (donde se aplica A\* propiamente dicho) se detiene y pasa a la fase 2, que es el backtracking, donde se accede a la lista `came_from` del nodo que llegó al nodo objetivo primero y se recorre sucesivamente los nodos padres hasta llegar al nodo raíz. Finalmente cuando esto pasa se traza la ruta desde el punto A al punto B por el camino anteriormente recorrido.



Paso	“frontier” (nodos y sus costos)	Ampliar*	“explored”: conjunto de nodos
1	{(A,0)}	A	Ø
2	{(D,3),(B,5)}	D	{A}
3	{(B,5),(E,5),(F,5)}	B	{A,D}
4	{(E,5),(F,5),(C,6)}	E	{A,D,B}
5	{(F,5),(C,6)}**	F	{A,D,B,E}
6	{(C,6),(G,8)}	C	{A,D,B,E,F}
7	{(G,8)}	G	{A,D,B,E,F,C}
8	Ø		



## 5.Código

La implementación se desarrolló en un 93,7% de su extensión en el lenguaje de programación “*Python*”, utilizando como módulo pilar de desarrollo el conocido como OpenCV. También se utilizaron módulos como numpy, matplotlib, pygame, etc. El 6,3% restante está codificado en PDDL.

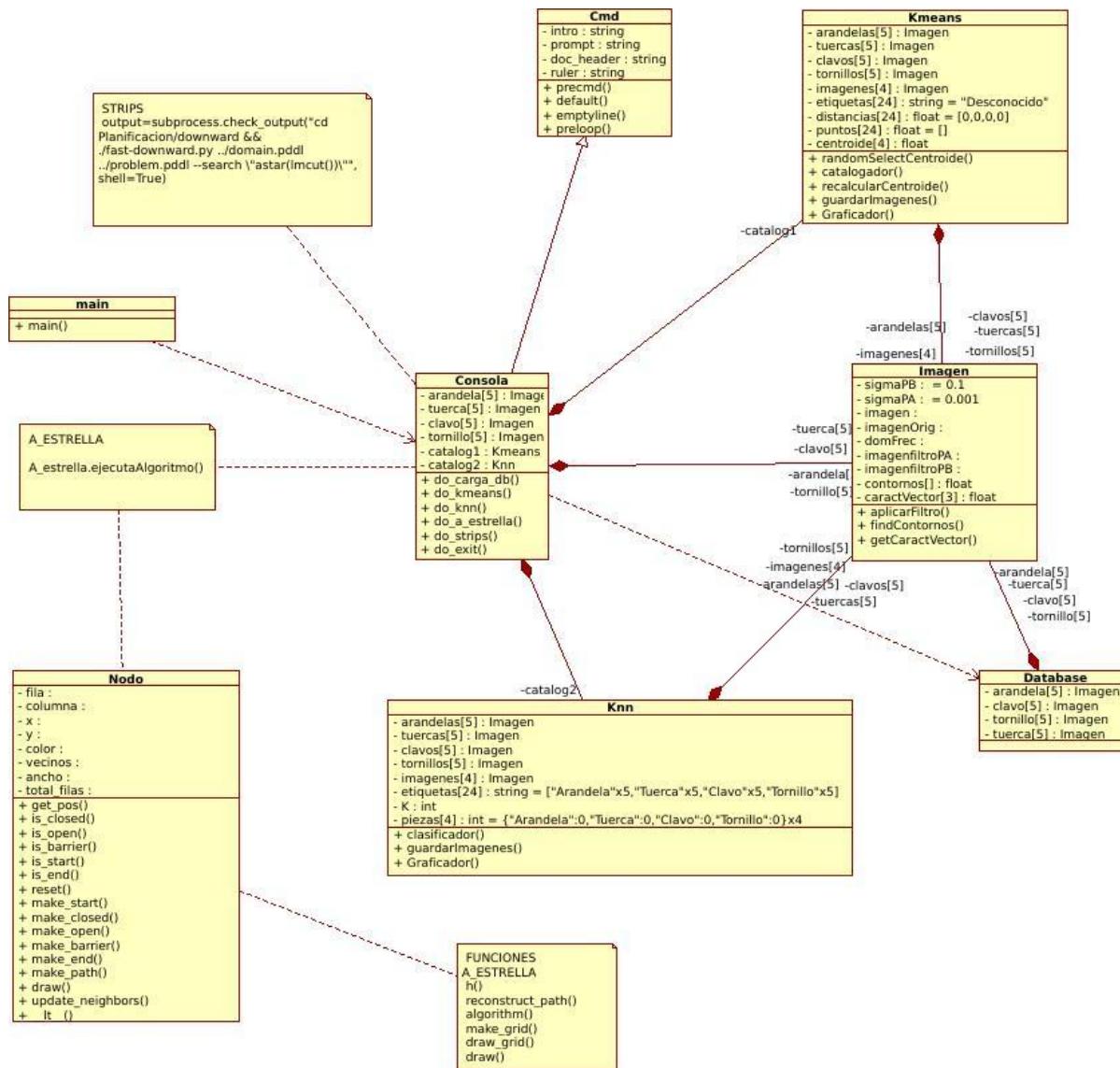
La implementación se plantea en su mayor parte en paradigma orientado a objetos, segmentado en 7 modulos de python y 2 modulos de PDDL (Domain y Problem).

El procedimiento que se llevó a cabo y las distintas versiones de código pueden ser seguidas a través del repositorio público que cree en Github:

<https://github.com/Juani1904/Matecito>



## 5.1 Diagrama UML



## 5.2 Main

```
#Importamos el modulo donde se encuentra la clase Database, que crea la base de datos
from database import Database
```

```
#Importamos el modulo donde se encuentra la clase consola
from consola import Consola
```

```
if __name__ == "__main__":
```

```
#Instanciamos el objeto consola, donde manejaremos todo
consola=Consola()
```

```
#Llamamos al cmdloop
```



consola.cmdloop()

## 5.3 Consola

```
#En este espacio vamos a programar una consola donde el usuario podra interactuar
#con el programa y aplicar los distintos metodos de segmentacion
import subprocess
from cmd import Cmd
import time
#Importamos el modulo donde hacemos la definicion de la clase Kmeans
from kmeans import Kmeans
#Importamos el modulo donde hacemos la definicion de la clase Kn
from knn import Kn
#Importamos la clase donde se encuentra la base de datos
from database import Database
#Importamos el modulo donde se encuentra la clase Nodo y las funciones para aplicar
algoritmo A estrella
import A_estrella
#Definimos la clase hija Consola, heredada de la clase padre Cmd
class Consola(Cmd):
    intro="Bienvenido a M.A.T.E.C.I.T.O ® by Juani. Tipee help o ? para listar los comandos
disponibles"
    prompt="Matecito>>"
    doc_header="Lista de comandos disponibles:"
    Cmd.ruler

#Definimos el constructor, para instanciar los objetos que utilizaremos en la consola
def __init__(self):
    Cmd.__init__(self)

#Definimos los distintos metodos a los cuales podremos accesar desde la consola

def do_carga_db(self,arg):
    'Carga la base de datos de imagenes al Robot.Tipee CARGA_DB'
    db=Database()
    self.arandela=db.arandela
    self.tuerca=db.tuerca
    self.clavo=db.clavo
    self.tornillo=db.tornillo

def do_kmeans(self,iteraciones):
    'Aplica el algortimo de segmentacion no supervisada Kmeans, para identificar las
imagenes de la carpeta Input. Tipee KMEANS <N de interacciones>'
```



#Agregamos una excepcion. Si no se carga la DB no se puede acceder al metodo try:

```
self.catalog1=Kmeans(self.arandela,self.tuerca,self.clavo,self.tornillo)
self.catalog1.Graficador(-1)
for i in range(int(iteraciones)):
    self.catalog1.catalogador()
    print("Iteracion "+str(i+1)+"")
    print("Centroides:")
    print(self.catalog1.centroide)
    if i==0:
        self.catalog1.Graficador(-1)
    self.catalog1.recalcularCentroide()
    self.catalog1.Graficador(i)
    self.catalog1.guardarImagenes()
except AttributeError:
    print ("No se ha cargado la base de datos. Intente con CARGA_DB")
except ValueError:
    print ("No se ha cargado la base de datos. Intente con CARGA_DB")

def do_knn(self,K):
    'Aplica el algortimo de clasificacion supervisada Knn, para identificar las imagenes de la carpeta Input. Tipee KNN <N de vecinos>'
    try:
        self.catalog2=Knn(self.arandela,self.tuerca,self.clavo,self.tornillo)
        self.catalog2.clasificador(int(K)) #Utilizamos el clasificador con un numero de vecinos igual a K
        self.catalog2.Graficador()
        self.catalog2.guardarImagenes()
    except AttributeError:
        print ("No se ha cargado la base de datos. Intente con CARGA_DB")
    except ValueError:
        print ("No se ha cargado la base de datos. Intente con CARGA_DB")

def do_a_estrella(self,arg):
    'Aplica el algortimo de busqueda A*. Tipee A_ESTRELLA'
    print("\n#####")
    print("Instrucciones:")
    print("1. Ingrese el punto de inicio con click izquierdo")
    print("2. Ingrese el punto de destino con click izquierdo")
    print("3. Ingrese obstaculos con click izquierdo")
    print("4. Si se equivoca, puede borrar con click derecho")
    print("5. Presione ESPACIO para comenzar busqueda")
    print("6. Presione C para restaurar ventana a default")
    print("#####\n")
    A_estrella.ejecutaAlgoritmo()
```



```
def do_strips(self,arg):
    'Ejecuta el plan en lenguaje STRIPS mediante FASTDOWNWARD.Tipee STRIPS'
    output = subprocess.check_output("cd Planificacion/downward && ./fast-downward.py
..domain.pddl ..problem.pddl --search \"astar(blind())\"", shell=True)
    listaoutput=output.decode("utf-8").split("\n")
    target1='poner'
    target2='sacar'
    print("El plan trazado es: ")
    contador=0
    print("\n")
    for elemento in listaoutput:
        if target1 in elemento:
            print(elemento)
            contador+=1

        elif target2 in elemento:
            print(elemento)
            contador+=1
    print("\n")
    print("El costo del camino es: ",end="")
    print(contador,end=" ")
    print("unidades")
    #Printear el costo del camino sin salto de linea
```

```
def do_exit(self,line):
    'Salir del programa (Apagar Robot)'
    return True
```

```
#Metemos un precmd para que las palabras escritas en mayuscula se conviertan en minuscula
def precmd(self,line):
    return line.lower()
#Mensaje por defecto cuando ponemos un comando incorrecto
def default(self):
    try:
        print("Comando no reconocido. Ingrese help <comando> para ver su sintaxis")
    except AttributeError:
        print("Comando no reconocido. Ingrese help <comando> para ver su sintaxis")
    except TypeError:
        print("Comando no reconocido. Ingrese help <comando> para ver su sintaxis")
#Mensaje por defecto cuando le damos enter sin escribir nada primero
def emptyline(self):
    pass
```



#Generamos un preloop de barra de carga, como si el software se estuviera cargando en la memoria del robot

```
def preloop(self):
    print("Iniciando...")
    time.sleep(1)
    for i in range(0,101):
        time.sleep(0.05)
        print("Cargando software a Robot IA...[%d%%]" % i, end="\r")
    print("Cargando software a Robot IA...[100%]")
    time.sleep(0.5)
```

## 5.4 Imagen

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import math as m
```

#Definimos la clase Imagen

class Imagen:

#La idea es tener un solo atributo imagen, y que ese atributo se vaya modificando segun lo que le vayamos

#haciendo a la imagen

sigmaPB=0.1

sigmaPA=0.001

#Creamos el constructor de la clase Imagen

```
def __init__(self,miImagen):
```

self.imagen=cv2.imread(miImagen)

self.imagenOrig=cv2.imread(miImagen)

self.domFrec=cv2.imread(miImagen)

self.imagenfiltroPA=cv2.imread(miImagen)

self.imagenfiltroPB=cv2.imread(miImagen)

#Una vez obtenida la imagen, la redimensionamos a 512x512px

self.imagen=cv2.resize(self.imagen,(512,512))

self.imagenOrig=cv2.resize(self.imagenOrig,(512,512))

#Pasamos la imagen a escala de grises

self.imagen=cv2.cvtColor(self.imagen,cv2.COLOR\_RGB2GRAY)

#Establecemos el tipo de dato de intensidad como float

self.imagen=np.float64(self.imagen)

#Aplicamos erosion y dilatacion

#La erosion sirve para expandir la parte negra de la imagen binarizada. Si hay ruidos no eliminados mediante el filtro PB



#se expande la parte negra de la imagen y se eliminan esos ruidos

#La dilatacion sirve para expandir los bordes de la imagen. Incrementa la parte blanca de la imagen binarizada

#Aplicamos la erosión y luego la dilatacion para ensanchar los bordes blancos, dado que la erosión los hizo mas chicos

#y corremos el riesgo de que alguna parte sea eliminada y no nos tome bien el contorno

```
kernel=np.ones((3,3),np.uint8)
```

```
self.imagen=cv2.erode(self.imagen,kernel,iterations=1)
```

```
self.imagen=cv2.dilate(self.imagen,kernel,iterations=1)
```

#Creamos un manejo de excepción para solucionar el error ZeroDivision

#Si la imagen no toma bien el contorno y por ende el perímetro, se ajusta la frec. de corte de los filtros

```
while(True):
```

```
    try:
```

#Aplicamos filtro pasa bajo para eliminar el ruido

```
    self.aplicarFiltro("PB")
```

```
    self.imagenfiltroPB=self.imagen
```

#Luego aplicamos el filtro para alto para resaltar los bordes

```
    self.aplicarFiltro("PA")
```

```
    self.imagenfiltroPA=self.imagen
```

#Aplicamos la binarización a la imagen

```
    #self.imagen=cv2.Canny(self.imagen,10,150)
```

```
    _, self.imagen = cv2.threshold(self.imagen, 0, 255, cv2.THRESH_BINARY |  
cv2.THRESH_TRIANGLE)
```

#Aplicamos el método de búsqueda de contornos

```
    self.findContornos()
```

#Creamos el vector de características

```
    self.getCaractVector()
```

```
    break
```

```
except ZeroDivisionError:
```

#Con esto disminuimos la frecuencia de corte del filtro PB y aumentamos la del filtro PA

```
    self.sigmaPB=0.01
```

```
    self.sigmaPA+=0.001
```

```
    continue
```

```
def aplicarFiltro(self,tipo):
```

#Primero definimos el R, el cual es el exponente de la función de filtro Gaussiano

```
F1=np.arange(-256,256,1)
```

```
F2=np.arange(-256,256,1)
```

```
[X,Y]=np.meshgrid(F1,F2)
```

```
R=np.sqrt(X**2+Y**2)
```

```
R=R/np.max(R)
```

#Aca es donde cambia según el tipo de filtro que elijamos



```
if tipo=="PB":  
    FiltroH = np.exp(-(R**2)/(2*sigmaPB**2))  
  
elif tipo=="PA":  
    FiltroH = 1-np.exp(-(R**2)/(2*sigmaPA**2))  
  
    #Modificamos el origen de coordenadas de la funcion. El cero pasa del centro a los  
extremos  
    FiltroHmod=np.fft.fftshift(FiltroH)  
    #Aplicamos transformada de Fourier 2D a la imagen  
    Fimg = np.fft.fft2(self.imagen)  
    self.domFrec=Fimg  
    #Generamos producto de filtro con imagen (CONVOLUCION)  
    FimgFiltro = Fimg*FiltroHmod  
    #Aplicamos la transformada inversa de Fourier 2D  
    imgFiltrada = np.fft.ifft2(FimgFiltro)  
    #Normalizamos la imagen  
    imagenFiltradaN = cv2.normalize(abs(imgFiltrada), None, alpha = 0, beta = 255,  
norm_type = cv2.NORM_MINMAX, dtype = cv2.CV_8U)  
    #Finalmente modificamos el atributo imagen  
    self.imagen=imagenFiltradaN  
  
def findContornos(self):  
    #Aca se va a implementar el algoritmo de deteccion de contornos  
  
    #Buscamos los puntos de contorno  
  
    self.contornos,_=cv2.findContours(self.imagen,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)  
    #Con esta funcion sorted, mejoramos la toma de contorno de manera que se tome bien  
dicho contorno y se  
    #pueda calcular el perimetro y el area sin problemas  
    self.contornos = sorted(self.contornos,key=cv2.contourArea,reverse=True)[:1]  
    # Defectos convexos  
    self.hull = cv2.convexHull(self.contornos[0],returnPoints=True)  
  
def getCaractVector(self):  
    #Creamos el vector de caracteristicas  
    self.caractVector=[]  
    #Calculamos el area y el perimetro de la figura que se armo con la deteccion del  
contorno  
    areafig=cv2.contourArea(self.contornos[0])  
    perimetrofig=cv2.arcLength(self.contornos[0],True)
```



#Calculamos la circularidad

#Mientras mas cerca de uno la circularidad, mas se parece a un circulo perfecto

circularidad=(**4\*(m.pi)\*areafig**)/(perimetrofig)**\*\*2**

#self.caractVector.append(circularidad)

#Calculamos la elasticidad

#La elasticidad es lo mismo que el aspect ratio (width(ancho)/length(altura))

#Utilizamos boundingRect

#Si es 1 es un cuadrado perfecto, mayor a 1 estirado horizontal, menor a 1, estirado vertical

\_,\_,ancho,alto=cv2.boundingRect(self.contornos[**0**])

elasticidad=float(ancho)/alto

#self.caractVector.append(elasticidad)

#Realizamos una aprox polinomial que nos servira de parametro

#Lo que vamos a hacer es sumar al vect. de caracteristicas la cantidad de elementos de approxPoly

#Este numero de elementos corresponde a los "vertices de la figura"

#Esto es especialmente util para diferenciar la arandela de la tuerca

epsilon=**0.03**\*cv2.arcLength(self.contornos[**0**],True)

approxPoly=cv2.approxPolyDP(self.contornos[**0**],epsilon,True)

self.caractVector.append(len(approxPoly))

#Calculamos la solidez o "Solidity" de la figura

#Es la relacion entre el area de la figura y el area del casco convexo (Convex hull)

areaConvex=cv2.contourArea(self.hull)

solidez=areafig/areaConvex

#self.caractVector.append(solidez)

#Calculamos momento de Hu

momentosHu=cv2.HuMoments(cv2.moments(self.contornos[**0**])),flatten()

self.caractVector.append(momentosHu[**0**])

self.caractVector.append(momentosHu[**5**])

#self.caractVector.append(momentosHu[6])

#Fin de definición de clase

## 5.5 Database

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
```



```
import cv2
import math as m
```

```
#Importamos el modulo donde hacemos la definicion de clase Imagen
from imagen import Imagen
```

```
class Database():
```

```
#Definimos constructor
```

```
def __init__(self):
```

```
#-----BASE DE DATOS-----
```

```
print("ARANDELAS")
#ARANDELAS
```

```
cargaArandelas=["base_de_datos/arandela1.jpg","base_de_datos/arandela2.jpg","base_de_
datos/arandela3.jpg","base_de_datos/arandela4.jpg","base_de_datos/arandela5.jpg"]
```

```
arandelaimg=[]
self.arandela=[]
```

```
for nombre in cargaArandelas:
```

```
i=cargaArandelas.index(nombre)
```

```
arandelaimg.append(cv2.imread(nombre))
```

```
self.arandela.append(Imagen(nombre))
```

```
arandelaimg[i]=(cv2.resize(arandelaimg[i],(512,512)))
```

```
cv2.imshow("Original",self.arandela[i].imagenOrig)
```

```
cv2.imshow("FiltroPB",self.arandela[i].imagenfiltroPB)
```

```
cv2.imshow("FiltroPA",self.arandela[i].imagenfiltroPA)
```

```
cv2.imshow("Binarizada",self.arandela[i].imagen)
```

```
cv2.drawContours(arandelaimg[i],self.arandela[i].contornos,-1,(0,255,0),2)
```

```
cv2.imshow("Contornos",arandelaimg[i])
```

```
cv2.waitKey(0)
```

```
fig = plt.figure("Img Dom Frecuencial", figsize=(4, 4))
```

```
plt.imshow(np.log(np.abs(self.arandela[i].domFrec)), cmap='gray')
```

```
plt.show()
```

```
plt.close()
```

```
print(self.arandela[i].caractVector)
```

```
print("CLAVOS")
#CLAVOS
```



```
cargaClavos=["base_de_datos/clavo1.jpg","base_de_datos/clavo2.jpg","base_de_datos/clavo3.jpg","base_de_datos/clavo4.jpg","base_de_datos/clavo5.jpg"]
clavoimg=[]
self.clavo=[]
for nombre in cargaClavos:
    i=cargaClavos.index(nombre)
    clavoimg.append(cv2.imread(nombre))
    self.clavo.append(Imagen(nombre))
    clavoimg[i]=(cv2.resize(clavoimg[i],(512,512)))
    cv2.imshow("Original",self.clavo[i].imagenOrig)
    cv2.imshow("FiltroPB",self.clavo[i].imagenfiltroPB)
    cv2.imshow("FiltroPA",self.clavo[i].imagenfiltroPA)
    cv2.imshow("Binarizada",self.clavo[i].imagen)
    cv2.drawContours(clavoimg[i],self.clavo[i].contornos,-1,(0,255,0),2)
    cv2.imshow("Contornos",clavoimg[i])
    cv2.waitKey(0)
fig = plt.figure("Img Dom Frecuencial", figsize=(4, 4))
plt.imshow(np.log(np.abs(self.arandela[i].domFrec)), cmap='gray')
plt.show()
plt.close()
print(self.clavo[i].caractVector)

print("TORNILLOS")
#TORNILLOS
```

```
cargaTornillos=["base_de_datos/tornillo1.jpg","base_de_datos/tornillo2.jpg","base_de_datos/tornillo3.jpg","base_de_datos/tornillo4.jpg","base_de_datos/tornillo5.jpg"]
tornilloimg=[]
self.tornillo=[]
for nombre in cargaTornillos:
    i=cargaTornillos.index(nombre)
    tornilloimg.append(cv2.imread(nombre))
    self.tornillo.append(Imagen(nombre))
    tornilloimg[i]=(cv2.resize(tornilloimg[i],(512,512)))
    cv2.imshow("Original",self.tornillo[i].imagenOrig)
    cv2.imshow("FiltroPB",self.tornillo[i].imagenfiltroPB)
    cv2.imshow("FiltroPA",self.tornillo[i].imagenfiltroPA)
    cv2.imshow("Binarizada",self.tornillo[i].imagen)
    cv2.drawContours(tornilloimg[i],self.tornillo[i].contornos,-1,(0,255,0),2)
    cv2.imshow("Contornos",tornilloimg[i])
    cv2.waitKey(0)
fig = plt.figure("Img Dom Frecuencial", figsize=(4, 4))
plt.imshow(np.log(np.abs(self.arandela[i].domFrec)), cmap='gray')
plt.show()
```



```
plt.close()
print(self.tornillo[i].caractVector)

print("TUERCAS")
#TUERCAS

cargaTuercas=["base_de_datos/tuerca1.jpg","base_de_datos/tuerca2.jpg","base_de_datos/t
uerca3.jpg","base_de_datos/tuerca4.jpg","base_de_datos/tuerca5.jpg"]
tuercaimg=[]
self.tuerca=[]
for nombre in cargaTuercas:
    i=cargaTuercas.index(nombre)
    tuercaimg.append(cv2.imread(nombre))
    self.tuerca.append(Imagen(nombre))
    tuercaimg[i]=(cv2.resize(tuercaimg[i],(512,512)))
    cv2.imshow("Original",self.tuerca[i].imagenOrig)
    cv2.imshow("FiltroPB",self.tuerca[i].imagenfiltroPB)
    cv2.imshow("FiltroPA",self.tuerca[i].imagenfiltroPA)
    cv2.imshow("Binarizada",self.tuerca[i].imagen)
    cv2.drawContours(tuercaimg[i],self.tuerca[i].contornos,-1,(0,255,0),2)
    cv2.imshow("Contornos",tuercaimg[i])
    cv2.waitKey(0)
fig = plt.figure("Img Dom Frecuencial", figsize=(4, 4))
plt.imshow(np.log(np.abs(self.arandela[i].domFrec)), cmap='gray')
plt.show()
plt.close()
print(self.tuerca[i].caractVector)

#Finalmente para cerrar todas las imagenes abiertas
cv2.destroyAllWindows()
#-----FIN BASE DE DATOS-----
```

## 5.6 Kmeans

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
import cv2
import math as m
import os
from imagen import Imagen
from random import randint
from natsort import natsorted
```



**class Kmeans:**

```
#Definimos el constructor
def __init__(self,miarandela,mituerca,miclavos,mitornillo):
    #Incluimos los objetos creados en el main, para poder comunicarnos con ellos
    self.arandelas=miarandela
    self.tuercas=mituerca
    self.clavos=miclavos
    self.tornillos=mitornillo
    #Ahora instanciamos las imágenes desconocidas de la carpeta input
    #Inicializamos el vector de imágenes
    self.imagenes=[]
    archivos=natsorted(os.listdir("Input"))
    for filename in archivos:
        self.imagenes.append(Imagen("Input"+filename))
        ind=archivos.index(filename)
        cv2.imshow("Imagen "+str(filename),self.imagenes[ind].imagenOrig)
        cv2.imshow("FiltroPB "+str(filename),self.imagenes[ind].imagenfiltroPB)
        cv2.imshow("FiltroPA "+str(filename),self.imagenes[ind].imagenfiltroPA)
        cv2.imshow("Binarizada "+str(filename),self.imagenes[ind].imagen)

cv2.drawContours(self.imagenes[ind].imagenOrig,self.imagenes[ind].contornos,-1,(0,255,0),
2)
cv2.imshow("Contornos "+str(filename),self.imagenes[ind].imagenOrig)
cv2.waitKey(0)
fig = plt.figure("Img Dom Frecuencial "+str(filename), figsize=(4, 4))
plt.imshow(np.log(np.abs(self.imagenes[ind].domFrec)), cmap='gray')
plt.show()
plt.close()
cv2.destroyAllWindows()
print("Imagen "+str(ind+1))
print(self.imagenes[ind].caractVector)

#Definimos los centroides
self.randomSelectCentroide()
#Definimos el vector de etiquetas
self.etiquetas=[]
#Definimos el vector de distancias
self.distancias=[]
    #Llenamos el vector de etiquetas con 24 elementos que digan "desconocido", el de
distancias con 0
    #El vector distancia tendra como elemento una lista de 4 elementos porque esas son
las distancias de cada elemento a cada uno de los 4 centroides
```



```
for i in range(24):  
    self.etiquetas.append("Desconocido")  
    self.distancias.append([0,0,0,0])
```

#Armamos una lista con todos los puntos de todas las imágenes, en el mismo orden que antes

```
self.puntos=[]  
for i in range(5):  
    self.puntos.append(self.arandelas[i].caractVector)  
for i in range(5):  
    self.puntos.append(self.tuercas[i].caractVector)  
for i in range(5):  
    self.puntos.append(self.clavos[i].caractVector)  
for i in range(5):  
    self.puntos.append(self.tornillos[i].caractVector)  
for i in range(4):  
    self.puntos.append(self.imagenes[i].caractVector)
```

```
def randomSelectCentroide(self):
```

#Los parámetros no son más que vectores o listas de objetos Imagen

```
i1=randint(0,4)  
i2=randint(0,4)  
i3=randint(0,4)  
i4=randint(0,4)
```

```
self.centroide=[self.arandelas[i1].caractVector,self.tuercas[i2].caractVector,self.clavos[i3].ca  
ractVector,self.tornillos[i4].caractVector]
```

```
def catalogador(self):
```

#Calculamos la distancia euclídea de las imágenes a los centroides

#Primero para las arandelas

```
for i in range(5):  
    for j in range(4):
```

```
self.distancias[i][j]=m.sqrt((self.arandelas[i].caractVector[0]-self.centroide[j][0])**2+(self.ar  
anelas[i].caractVector[1]-self.centroide[j][1])**2+(self.arandelas[i].caractVector[2]-self.cen  
troide[j][2])**2)
```

#Ahora para las tuercas

```
for i in range(5):  
    for j in range(4):
```

```
self.distancias[i+5][j]=m.sqrt((self.tuercas[i].caractVector[0]-self.centroide[j][0])**2+(self.tuer  
cas[i].caractVector[1]-self.centroide[j][1])**2+(self.tuercas[i].caractVector[2]-self.cen  
troide[j][2])**2)
```



```
cas[i].caractVector[1]-self.centroide[j][1])**2+(self.tuercas[i].caractVector[2]-self.centroide[j][2])**2)
```

#Ahora para los clavos

```
for i in range(5):  
    for j in range(4):
```

```
self.distancias[i+10][j]=m.sqrt((self.clavos[i].caractVector[0]-self.centroide[j][0])**2+(self.clavos[i].caractVector[1]-self.centroide[j][1])**2+(self.clavos[i].caractVector[2]-self.centroide[j][2])**2)
```

#Ahora para los tornillos

```
for i in range(5):  
    for j in range(4):
```

```
self.distancias[i+15][j]=m.sqrt((self.tornillos[i].caractVector[0]-self.centroide[j][0])**2+(self.tornillos[i].caractVector[1]-self.centroide[j][1])**2+(self.tornillos[i].caractVector[2]-self.centroide[j][2])**2)
```

#Ahora para las imágenes desconocidas

```
for i in range(4):  
    for j in range(4):
```

```
self.distancias[i+20][j]=m.sqrt((self.imagenes[i].caractVector[0]-self.centroide[j][0])**2+(self.imagenes[i].caractVector[1]-self.centroide[j][1])**2+(self.imagenes[i].caractVector[2]-self.centroide[j][2])**2)
```

#Ahora asignamos las etiquetas

```
for distancias in self.distancias:  
    minimo=distancias.index(min(distancias))  
    if minimo==0:  
        self.etiquetas[self.distancias.index(distancias)]="Arandela"  
    elif minimo==1:  
        self.etiquetas[self.distancias.index(distancias)]="Tuerca"  
    elif minimo==2:  
        self.etiquetas[self.distancias.index(distancias)]="Clavo"  
    elif minimo==3:  
        self.etiquetas[self.distancias.index(distancias)]="Tornillo"
```

#Ahora definimos el método para recalcular el centroide mediante la media de los puntos de los clusters formados

```
def recalcularcentroide(self):
```

#Calculamos la media aritmética a lo largo de cada eje



```
ejeX=[]
ejeY=[]
ejeZ=[]
for coordenada in self.puntos:
    X,Y,Z=coordenada
    ejeX.append(X)
    ejeY.append(Y)
    ejeZ.append(Z)
```

*#Ahora vamos a calcular la media de los datos, segun su clasificacion*

```
ejeXArandela=[]
ejeYArandela=[]
ejeZArandela=[]
ejeXTuerca=[]
ejeYTuerca=[]
ejeZTuerca=[]
ejeXClavo=[]
ejeYClavo=[]
ejeZClavo=[]
ejeXTornillo=[]
ejeYTornillo=[]
ejeZTornillo=[]
```

```
for item in self.etiquetas:
    if item=="Arandela":
        ejeXArandela.append(ejeX[self.etiquetas.index(item)])
        ejeYArandela.append(ejeY[self.etiquetas.index(item)])
        ejeZArandela.append(ejeZ[self.etiquetas.index(item)])
    elif item=="Tuerca":
        ejeXTuerca.append(ejeX[self.etiquetas.index(item)])
        ejeYTuerca.append(ejeY[self.etiquetas.index(item)])
        ejeZTuerca.append(ejeZ[self.etiquetas.index(item)])
    elif item=="Clavo":
        ejeXClavo.append(ejeX[self.etiquetas.index(item)])
        ejeYClavo.append(ejeY[self.etiquetas.index(item)])
        ejeZClavo.append(ejeZ[self.etiquetas.index(item)])
    elif item=="Tornillo":
        ejeXTornillo.append(ejeX[self.etiquetas.index(item)])
        ejeYTornillo.append(ejeY[self.etiquetas.index(item)])
        ejeZTornillo.append(ejeZ[self.etiquetas.index(item)])
```

*#Finalmente, recalculamos el centroide*

```
self.centroide[0][0]=sum(ejeXArandela)/len(ejeXArandela)
self.centroide[0][1]=sum(ejeYArandela)/len(ejeYArandela)
self.centroide[0][2]=sum(ejeZArandela)/len(ejeZArandela)
```



```
self.centroide[1][0]=sum(ejeXTuerca)/len(ejeXTuerca)
self.centroide[1][1]=sum(ejeYTuerca)/len(ejeYTuerca)
self.centroide[1][2]=sum(ejeZTuerca)/len(ejeZTuerca)
self.centroide[2][0]=sum(ejeXClavo)/len(ejeXClavo)
self.centroide[2][1]=sum(ejeYClavo)/len(ejeYClavo)
self.centroide[2][2]=sum(ejeZClavo)/len(ejeZClavo)
self.centroide[3][0]=sum(ejeXTornillo)/len(ejeXTornillo)
self.centroide[3][1]=sum(ejeYTornillo)/len(ejeYTornillo)
self.centroide[3][2]=sum(ejeZTornillo)/len(ejeZTornillo)
```

#Definimos el metodo para guardar las 4 imagenes ahora catalogadas segun su etiqueta,  
en la carpeta Output, con su nombre particular

```
def guardarImagenes(self):
    #Primero borramos los archivos que pudieran haber en la carpeta output
    for archivo in os.listdir("Output/Kmeans/"):
        os.remove("Output/Kmeans/" + archivo)
    #Ahora guardamos las imagenes
    for imagen in self.imagenes:
        i=self.imagenes.index(imagen)
        if self.etiquetas[20+i]=="Arandela":
            cv2.imwrite("Output/Kmeans/" + str(i+1) + ".Arandela.jpg",imagen.imagenOrig)
        elif self.etiquetas[20+i]=="Tuerca":
            cv2.imwrite("Output/Kmeans/" + str(i+1) + ".Tuerca.jpg",imagen.imagenOrig)
        elif self.etiquetas[20+i]=="Clavo":
            cv2.imwrite("Output/Kmeans/" + str(i+1) + ".Clavo.jpg",imagen.imagenOrig)
        elif self.etiquetas[20+i]=="Tornillo":
            cv2.imwrite("Output/Kmeans/" + str(i+1) + ".Tornillo.jpg",imagen.imagenOrig)

    print("Imagenes guardadas en la carpeta Output/Kmeans")
def Graficador(self,iteracion):
    #En este metodo graficador lo que haremos sera ir graficando con distintos colores los
    #clusters
    #que se vayan formando y los centroides (y su recalculo)
    fig = plt.figure("Grafica Kmeans")
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlabel('AproxPoly')
    ax.set_ylabel('1er Momento Hu')
    ax.set_zlabel('6to Momento Hu')
    ax.set_title("Grafica Kmeans. Iteracion " + str(iteracion+1))
    #Colocamos una leyenda para clasificar las piezas por su color
    ax.scatter(0,0,0,c="red",marker="o",label="Arandela")
    ax.scatter(0,0,0,c="yellow",marker="o",label="Tuerca")
    ax.scatter(0,0,0,c="blue",marker="o",label="Clavo")
    ax.scatter(0,0,0,c="green",marker="o",label="Tornillo")
```



```
ax.scatter(0,0,0,c="black",marker="o",label="Desconocido")
ax.legend()
#Primero los centroides

ax.scatter(self.centroide[0][0],self.centroide[0][1],self.centroide[0][2],c="red",marker="o")

ax.scatter(self.centroide[1][0],self.centroide[1][1],self.centroide[1][2],c="yellow",marker="o")

ax.scatter(self.centroide[2][0],self.centroide[2][1],self.centroide[2][2],c="blue",marker="o")

ax.scatter(self.centroide[3][0],self.centroide[3][1],self.centroide[3][2],c="green",marker="o")
#Ahora los puntos, cuando esten sin clasificar iran en negro.
#Cuando el algoritmo los clasifique, adquiriran el color que les corresponde
for i in range(0,24):
    if self.puntos[i]!=self.centroide[0] and self.puntos[i]!=self.centroide[1] and
    self.puntos[i]!=self.centroide[2] and self.puntos[i]!=self.centroide[3]:
        if self.etiquetas[i]=="Arandela":
            ax.scatter(self.puntos[i][0],self.puntos[i][1],self.puntos[i][2],c="red",marker="o")
        elif self.etiquetas[i]=="Tuerca":
            ax.scatter(self.puntos[i][0],self.puntos[i][1],self.puntos[i][2],c="yellow",marker="o")
        elif self.etiquetas[i]=="Clavo":
            ax.scatter(self.puntos[i][0],self.puntos[i][1],self.puntos[i][2],c="blue",marker="o")
        elif self.etiquetas[i]=="Tornillo":
            ax.scatter(self.puntos[i][0],self.puntos[i][1],self.puntos[i][2],c="green",marker="o")
        elif self.etiquetas[i]=="Desconocido":
            ax.scatter(self.puntos[i][0],self.puntos[i][1],self.puntos[i][2],c="black",marker="o")
    else:
        continue
#Ahora los puntos, cuando esten clasificados iran en el color de su cluster

plt.show()
```

## 5.7 Knn

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
import cv2
import math as m
import os
```



```
from imagen import Imagen
from random import randint
from natsort import natsorted
#Ahora vamos a definir la clase Knn
```

```
class Knn:
```

```
#Definimos el constructor
def __init__(self,miarandela,mituerca,miclavo,mitornillo):
    #Incluimos los objetos creados en el main, para poder comunicarnos con ellos
    self.arandelas=miarandela
    self.tuercas=mituerca
    self.clavos=miclavo
    self.tornillos=mitornillo
    #Ahora instanciamos las imagenes desconocidas de la carpeta input
    #Inicializamos el vector de imagenes
    self.imagenes=[]
    archivos=natsorted(os.listdir("Input"))
    for filename in archivos:
        self.imagenes.append(Imagen("Input/"+filename))
        ind=archivos.index(filename)
        cv2.imshow("Imagen "+str(filename),self.imagenes[ind].imagenOrig)
        cv2.imshow("FiltroPB "+str(filename),self.imagenes[ind].imagenfiltroPB)
        cv2.imshow("FiltroPA "+str(filename),self.imagenes[ind].imagenfiltroPA)
        cv2.imshow("Binarizada "+str(filename),self.imagenes[ind].imagen)

    cv2.drawContours(self.imagenes[ind].imagenOrig,self.imagenes[ind].contornos,-1,(0,255,0),
2)
    cv2.imshow("Contornos "+str(filename),self.imagenes[ind].imagenOrig)
    cv2.waitKey(0)
    fig = plt.figure("Img Dom Frecuencial "+str(filename), figsize=(4, 4))
    plt.imshow(np.log(np.abs(self.imagenes[ind].domFrec)), cmap='gray')
    plt.show()
    plt.close()
    cv2.destroyAllWindows()
    print("Imagen "+str(ind+1))
    print(self.imagenes[ind].caractVector)

#Definimos el vector de etiquetas. Como este algoritmo es supervisado, el prellenado
#de los valores de etiqueta no sera "Desconocido", si no con el nombre de la pieza de
la base de datos
self.etiquetas=[]
```



```
for i in range(5):
    self.etiquetas.append("Arandela")
for i in range(5):
    self.etiquetas.append("Tuerca")
for i in range(5):
    self.etiquetas.append("Clavo")
for i in range(5):
    self.etiquetas.append("Tornillo")

def clasificador(self,K):
    #Definimos el numero de vecinos
    self.K=K
    #Definimos los vectores de distancias
    #Los definimos como variables locales y no como atributos porque solo los usaremos
    aca
    distimg1=[]
    distimg2=[]
    distimg3=[]
    distimg4=[]
    #Definimos los vectores respuesta donde se almacenara el resultado del contraste
    entre el vector de distancias
    #y el vector de etiquetas.
    respimg1=[]
    respimg2=[]
    respimg3=[]
    respimg4=[]
    #Ahora calculamos las distancias de cada imagen a cada una de las imagenes de la
    base de datos
    #Primero para la imagen 1
    for i in range(5):

        distimg1.append(m.sqrt((self.imagenes[0].caractVector[0]-self.arandelas[i].caractVector[0])**2+(self.imagenes[0].caractVector[1]-self.arandelas[i].caractVector[1])**2+(self.imagenes[0].caractVector[2]-self.arandelas[i].caractVector[2])**2))
        for i in range(5):

            distimg1.append(m.sqrt((self.imagenes[0].caractVector[0]-self.tuercas[i].caractVector[0])**2+(self.imagenes[0].caractVector[1]-self.tuercas[i].caractVector[1])**2+(self.imagenes[0].caractVector[2]-self.tuercas[i].caractVector[2])**2))
            for i in range(5):

                distimg1.append(m.sqrt((self.imagenes[0].caractVector[0]-self.clavos[i].caractVector[0])**2+(self.imagenes[0].caractVector[1]-self.clavos[i].caractVector[1])**2+(self.imagenes[0].caractVector[2]-self.clavos[i].caractVector[2])**2))
```



**for i in range(5):**

```
distimg1.append(m.sqrt((self.imagenes[0].caractVector[0]-self.tornillos[i].caractVector[0])**2
+(self.imagenes[0].caractVector[1]-self.tornillos[i].caractVector[1])**2+(self.imagenes[0].caractVector[2]-self.tornillos[i].caractVector[2])**2))
```

*#Ahora para la imagen 2*

**for i in range(5):**

```
distimg2.append(m.sqrt((self.imagenes[1].caractVector[0]-self.arandelas[i].caractVector[0])*
*2+(self.imagenes[1].caractVector[1]-self.arandelas[i].caractVector[1])**2+(self.imagenes[1].caractVector[2]-self.arandelas[i].caractVector[2])**2))
```

**for i in range(5):**

```
distimg2.append(m.sqrt((self.imagenes[1].caractVector[0]-self.tuercas[i].caractVector[0])**2
+(self.imagenes[1].caractVector[1]-self.tuercas[i].caractVector[1])**2+(self.imagenes[1].caractVector[2]-self.tuercas[i].caractVector[2])**2))
```

**for i in range(5):**

```
distimg2.append(m.sqrt((self.imagenes[1].caractVector[0]-self.clavos[i].caractVector[0])**2+
(self.imagenes[1].caractVector[1]-self.clavos[i].caractVector[1])**2+(self.imagenes[1].caractVector[2]-self.clavos[i].caractVector[2])**2))
```

**for i in range(5):**

```
distimg2.append(m.sqrt((self.imagenes[1].caractVector[0]-self.tornillos[i].caractVector[0])**2
+(self.imagenes[1].caractVector[1]-self.tornillos[i].caractVector[1])**2+(self.imagenes[1].caractVector[2]-self.tornillos[i].caractVector[2])**2))
```

*#Ahora para la imagen 3*

**for i in range(5):**

```
distimg3.append(m.sqrt((self.imagenes[2].caractVector[0]-self.arandelas[i].caractVector[0])*
*2+(self.imagenes[2].caractVector[1]-self.arandelas[i].caractVector[1])**2+(self.imagenes[2].caractVector[2]-self.arandelas[i].caractVector[2])**2))
```

**for i in range(5):**

```
distimg3.append(m.sqrt((self.imagenes[2].caractVector[0]-self.tuercas[i].caractVector[0])**2
+(self.imagenes[2].caractVector[1]-self.tuercas[i].caractVector[1])**2+(self.imagenes[2].caractVector[2]-self.tuercas[i].caractVector[2])**2))
```

**for i in range(5):**

```
distimg3.append(m.sqrt((self.imagenes[2].caractVector[0]-self.clavos[i].caractVector[0])**2+
(self.imagenes[2].caractVector[1]-self.clavos[i].caractVector[1])**2+(self.imagenes[2].caractVector[2]-self.clavos[i].caractVector[2])**2))
```

**for i in range(5):**

```
distimg3.append(m.sqrt((self.imagenes[2].caractVector[0]-self.tornillos[i].caractVector[0])**2
```



```
+ (self.imagenes[2].caractVector[1]-self.tornillos[i].caractVector[1])**2+(self.imagenes[2].caractVector[2]-self.tornillos[i].caractVector[2])**2))
```

#Ahora para la imagen 4

```
for i in range(5):
```

```
distimg4.append(m.sqrt((self.imagenes[3].caractVector[0]-self.arandelas[i].caractVector[0])**2+(self.imagenes[3].caractVector[1]-self.arandelas[i].caractVector[1])**2+(self.imagenes[3].caractVector[2]-self.arandelas[i].caractVector[2])**2))
```

```
for i in range(5):
```

```
distimg4.append(m.sqrt((self.imagenes[3].caractVector[0]-self.tuercas[i].caractVector[0])**2+(self.imagenes[3].caractVector[1]-self.tuercas[i].caractVector[1])**2+(self.imagenes[3].caractVector[2]-self.tuercas[i].caractVector[2])**2))
```

```
for i in range(5):
```

```
distimg4.append(m.sqrt((self.imagenes[3].caractVector[0]-self.clavos[i].caractVector[0])**2+(self.imagenes[3].caractVector[1]-self.clavos[i].caractVector[1])**2+(self.imagenes[3].caractVector[2]-self.clavos[i].caractVector[2])**2))
```

```
for i in range(5):
```

```
distimg4.append(m.sqrt((self.imagenes[3].caractVector[0]-self.tornillos[i].caractVector[0])**2+(self.imagenes[3].caractVector[1]-self.tornillos[i].caractVector[1])**2+(self.imagenes[3].caractVector[2]-self.tornillos[i].caractVector[2])**2))
```

#Ahora vemos el index de las K minimas distancias y las comparamos con las #los elementos con ese mismo index en el vector de etiquetas

```
for contador in range(K):
```

#Para la imagen 1

```
for distancia in distimg1:
```

```
if distancia == min(distimg1):
```

```
index = distimg1.index(distancia)
```

```
respimg1.append(self.etiquetas[index])
```

#Para la imagen 2

```
for distancia in distimg2:
```

```
if distancia == min(distimg2):
```

```
index = distimg2.index(distancia)
```

```
respimg2.append(self.etiquetas[index])
```

#Para la imagen 3

```
for distancia in distimg3:
```

```
if distancia == min(distimg3):
```

```
index = distimg3.index(distancia)
```

```
respimg3.append(self.etiquetas[index])
```

#Para la imagen 4

```
for distancia in distimg4:
```



```
if distancia == min(distimg4):  
    index = distimg4.index(distancia)  
    respimg4.append(self.etiquetas[index])
```

#Ahora actualizamos en minimo de cada imagen por 1000 (numero grande), para,  
en el caso que

```
#K>1 podemos tomar los K-1 valores minimos restantes  
distimg1[distimg1.index(min(distimg1))] = 1000  
distimg2[distimg2.index(min(distimg2))] = 1000  
distimg3[distimg3.index(min(distimg3))] = 1000  
distimg4[distimg4.index(min(distimg4))] = 1000
```

#Para graficar las esferas creamos el siguiente vector

#Cuando salgamos de la funcion clasificador, los valores del vector corresponderan

#a los radios de las esferas, que seran las distancias al vecino mas lejano de los K  
considerados

```
#self.distEsferas=[min(distimg1),min(distimg2),min(distimg3),min(distimg4)]
```

#Ahora vemos cuantas veces aparece cada elemento en las respuestas

#Las cantidades las almacenamos en diccionarios cuyo nombre clave es el de cada  
pieza

#Y el valor es la cantidad de veces que aparece

```
piezas1 = {"Arandela":0,"Tuerca":0,"Clavo":0,"Tornillo":0}  
piezas2 = {"Arandela":0,"Tuerca":0,"Clavo":0,"Tornillo":0}  
piezas3 = {"Arandela":0,"Tuerca":0,"Clavo":0,"Tornillo":0}  
piezas4 = {"Arandela":0,"Tuerca":0,"Clavo":0,"Tornillo":0}
```

```
for pieza in respimg1:  
    if pieza == "Arandela":  
        piezas1["Arandela"] += 1  
    elif pieza == "Tuerca":  
        piezas1["Tuerca"] += 1  
    elif pieza == "Clavo":  
        piezas1["Clavo"] += 1  
    elif pieza == "Tornillo":  
        piezas1["Tornillo"] += 1
```

```
for pieza in respimg2:  
    if pieza == "Arandela":  
        piezas2["Arandela"] += 1  
    elif pieza == "Tuerca":  
        piezas2["Tuerca"] += 1  
    elif pieza == "Clavo":  
        piezas2["Clavo"] += 1  
    elif pieza == "Tornillo":  
        piezas2["Tornillo"] += 1
```

```
for pieza in respimg3:
```



```
if pieza == "Arandela":  
    piezas3["Arandela"] += 1  
elif pieza == "Tuerca":  
    piezas3["Tuerca"] += 1  
elif pieza == "Clavo":  
    piezas3["Clavo"] += 1  
elif pieza == "Tornillo":  
    piezas3["Tornillo"] += 1  
for pieza in respimg4:  
    if pieza == "Arandela":  
        piezas4["Arandela"] += 1  
    elif pieza == "Tuerca":  
        piezas4["Tuerca"] += 1  
    elif pieza == "Clavo":  
        piezas4["Clavo"] += 1  
    elif pieza == "Tornillo":  
        piezas4["Tornillo"] += 1  
  
self.piezas=[]  
self.piezas.append(piezas1)  
self.piezas.append(piezas2)  
self.piezas.append(piezas3)  
self.piezas.append(piezas4)  
  
def guardarImagenes(self):  
#Ahora vemos para cada imagen cual es el valor que mas se repite en el diccionario  
#y guardamos la imagen en la carpeta output con el nombre de la clave que mas se  
repite  
  
#Primero borramos los archivos que pudieran haber en la carpeta output  
for archivo in os.listdir("Output/Knn/"):  
    os.remove("Output/Knn/" + archivo)  
#Ahora guardamos las imágenes  
for imagen in self.imagenes:  
    i = self.imagenes.index(imagen)  
    if self.piezas[i]["Arandela"] > self.piezas[i]["Tuerca"] and self.piezas[i]["Arandela"] >  
    self.piezas[i]["Clavo"] and self.piezas[i]["Arandela"] > self.piezas[i]["Tornillo"]:  
        cv2.imwrite("Output/Knn/" + str(i + 1) + ".Arandela.jpg", imagen.imagenOrig)  
  
elif self.piezas[i]["Tuerca"] > self.piezas[i]["Arandela"] and self.piezas[i]["Tuerca"] >  
    self.piezas[i]["Clavo"] and self.piezas[i]["Tuerca"] > self.piezas[i]["Tornillo"]:  
        cv2.imwrite("Output/Knn/" + str(i + 1) + ".Tuerca.jpg", imagen.imagenOrig)
```



```
    elif self.piezas[i]["Clavo"] > self.piezas[i]["Arandela"] and self.piezas[i]["Clavo"] >
self.piezas[i]["Tuerca"] and self.piezas[i]["Clavo"] > self.piezas[i]["Tornillo"]:
    cv2.imwrite("Output/Knn/" + str(i+1) + ".Clavo.jpg", imagen.imagenOrig)

    elif self.piezas[i]["Tornillo"] > self.piezas[i]["Arandela"] and self.piezas[i]["Tornillo"] >
self.piezas[i]["Tuerca"] and self.piezas[i]["Tornillo"] > self.piezas[i]["Clavo"]:
    cv2.imwrite("Output/Knn/" + str(i+1) + ".Tornillo.jpg", imagen.imagenOrig)

else:
    print("Imagen " + str(i+1) + ": Clasificacion indefinida para este numero de vecinos")
    print("Intentando nuevamente con K=" + str(self.K+1))
    self.clasificador(self.K+1)
    self.guardarImagenes()
    #cv2.imwrite("Output/Knn/" + str(i+1) + ".Desconocido.jpg", imagen.imagenOrig)

print("Imagenes guardadas en la carpeta Output/Knn")

def Graficador(self):
    #En este metodo graficaremos los puntos de cada pieza con distintos colores
    #Tambien graficaremos los puntos que queremos conocer
    #Y finalmente una esfera desde el punto que queremos conocer, con radio igual la
    #distancia a los K vecinos
    #Para esto usaremos la libreria matplotlib
    #Primero graficaremos los puntos de cada pieza
    fig = plt.figure("Grafica Knn")
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlabel('AproxPoly')
    ax.set_ylabel('1er Momento Hu')
    ax.set_zlabel('6to Momento Hu')
    ax.set_title("Grafica Knn")
    for i in range(len(self.arandelas)):

        ax.scatter(self.arandelas[i].caractVector[0],self.arandelas[i].caractVector[1],self.arandelas[i].caractVector[2],c='red',marker='o')

    #Graficamos los clavos
    for i in range(len(self.clavos)):

        ax.scatter(self.clavos[i].caractVector[0],self.clavos[i].caractVector[1],self.clavos[i].caractVector[2],c='blue',marker='o')

    #Graficamos los tornillos
    for i in range(len(self.tornillos)):

        ax.scatter(self.tornillos[i].caractVector[0],self.tornillos[i].caractVector[1],self.tornillos[i].caractVector[2],c='green',marker='o')
```



#Graficamos las tuercas

```
for i in range(len(self.tuercas)):
```

```
    ax.scatter(self.tuercas[i].caractVector[0],self.tuercas[i].caractVector[1],self.tuercas[i].caractVector[2],c='yellow',marker='o')
```

#Graficamos las imágenes desconocidas

```
for i in range(len(self.imagenes)):
```

```
    ax.scatter(self.imagenes[i].caractVector[0],self.imagenes[i].caractVector[1],self.imagenes[i].caractVector[2],c='k',marker='o')
```

#Colocamos una leyenda para clasificar las piezas por su color

```
ax.scatter(0,0,0,c='red',marker='o',label='Arandela')
```

```
ax.scatter(0,0,0,c='blue',marker='o',label='Clavo')
```

```
ax.scatter(0,0,0,c='green',marker='o',label='Tornillo')
```

```
ax.scatter(0,0,0,c='yellow',marker='o',label='Tuerca')
```

```
ax.scatter(0,0,0,c='k',marker='o',label='Desconocido')
```

```
ax.legend()
```

```
plt.show()
```

## 5.8 A-estrella

```
def ejecutaAlgoritmo():
```

# En este espacio vamos a programar el algoritmo de path finding, A\* (o A estrella)

# Para desarrollar el GUI o la interfaz grafica donde el algoritmo se vera en accion utilizaremos el modulo pygame

```
import pygame
```

```
import math
```

```
from queue import PriorityQueue
```

#Basado en el codigo implementado por el usuario de

# youtube "Tech With Tim", el cual se encuentra en el siguiente link:

# <https://www.youtube.com/watch?v=JtiK0DOeI4A>

# Ancho de la ventana donde se mostrara el laberinto. Como sera de un tamaño NxN el alto sera igual al ancho

ANCHO = 800

# Creamos la ventana donde se mostrara el laberinto

VENTANA = pygame.display.set\_mode((ANCHO, ANCHO))

pygame.display.set\_caption("M.A.T.E.C.I.T.O A\* pathfinding")

# Definimos los distintos colores en escala RGB

ROJO = (255, 0, 0)



VERDE = (0, 255, 0)  
AZUL = (0, 0, 255)  
AMARILLO = (255, 255, 0)  
BLANCO = (255, 255, 255)  
NEGRO = (0, 0, 0)  
PURPURA = (128, 0, 128)  
NARANJA = (255, 165, 0)  
GRIS = (128, 128, 128)  
TURQUESA = (64, 224, 208)

# Vamos a manejar a los nodos del laberinto con paradigma orientado a objetos, de manera que cada nodo

# sera un objeto de la clase Nodo, el cual tiene atributos que nos permiten conocer # su posicion, su color, sus vecinos, su ancho y alto, etc

**class Nodo:**

# Definimos el constructor  
# Tener en cuenta que aca en los parametros:  
# fila: es la fila en la que se encuentra el nodo  
# columna: es la columna en la que se encuentra el nodo  
# ancho: es el ancho y alto DEL NODO, no de la ventana  
# total\_filas: es el total de filas que tiene el laberinto  
**def \_\_init\_\_(self, fila, columna, ancho, total\_filas):**  
    **self.fila** = fila  
    **self.columna** = columna  
    # Teniendo en cuenta que el eje X+ es vertical hacia abajo y el eje Y+ es horizontal hacia la derecha

# Las coordenadas x e y me indican el vertice del cuadradito que forma el nodo

**self.x** = fila \* ancho  
**self.y** = columna \* ancho  
**self.color** = BLANCO  
**self.vecinos** = []  
**self.ancho** = ancho  
**self.total\_filas** = total\_filas

# Declaramos los metodos

# El metodo get\_pos es un getter que me va a retornar la fila y columna donde se encuentra el nodo

**def get\_pos(self):**  
    **return self.fila, self.columna**



```
# METODOS IS
# La idea de los metodos is es verificar el estado de un nodo segun el color
que este tenga
    # El valor de retorno siempre sera un atributo == COLOR. Esto es porque se
llama a estos
        # metodos en los if, si se cumple se verifica la condicion en el if, si no no.
        # CLOSED verifica si el nodo esta en rojo, que significa que se encuentra en
el closedset (nodo ya analizado)
        def is_closed(self):
            return self.color == ROJO
        # OPEN verifica si el nodo esta en verde, que significa que se encuentra en
el openset (aun no verificada)

        def is_open(self):
            return self.color == VERDE
        # BARRIER verifica si el nodo esta en negro, que significa que es un
obstaculo (A modificar)

        def is_barrier(self):
            return self.color == NEGRO
        # START verifica si el nodo esta en naranja, que significa que es el punto
inicial

        def is_start(self):
            return self.color == NARANJA
        # END verifica si el nodo esta en turquesa, que significa que es el punto final

        def is_end(self):
            return self.color == TURQUESA

    # El metodo reset torna el nodo nuevamente en blanco
    def reset(self):
        self.color = BLANCO

# METODOS MAKE
# Establece el color de los nodos
# Establecemos el nodo de inicio en color naranja
def make_start(self):
    self.color = NARANJA
# Establecemos el nodo de closedset en color rojo

def make_closed(self):
    self.color = ROJO
# Establecemos el nodo de openset en color verde
```



```
def make_open(self):
    self.color = VERDE
    # Establecemos el nodo de obstaculo en color negro (A modificar)

def make_barrier(self):
    self.color = NEGRO
    # Establecemos el nodo final en color turquesa

def make_end(self):
    self.color = TURQUESA
    # Establecemos el nodo de camino en color purpura

def make_path(self):
    self.color = PURPURA

# Con el metodo draw dibujamos el nodo (cuadradito) en la ventana que ya creamos antes.
# Le pasamos la ventana donde queremos dibujarlo, las coordenadas del vertice del cubito, su ancho y largo(mismo que ancho)
def draw(self, ventana):
    pygame.draw.rect(ventana, self.color,
                     (self.x, self.y, self.ancho, self.ancho))

# Con este metodo lo que hacemos es checkear lo que rodea a nuestro nodo. Si es un openset,closedset,barrier(obstaculo),start o end
# Basicamente siempre que mis vecinos no sean obstaculos, los agrego dentro de mi lista de vecinos
# Esto lo hago para que el algoritmo identifique cuando llegue a una barrera y no siga indagando sobre la misma
# como posible vecino a ir, debido a que no puede
def update_neighbors(self, grid):
    self.vecinos = []
    # OBSTACULO ABAJO
    if self.fila < self.total_filas - 1 and not grid[self.fila + 1][self.columna].is_barrier():
        self.vecinos.append(grid[self.fila + 1][self.columna])

    # OBSTACULO ARRIBA
    if self.fila > 0 and not grid[self.fila - 1][self.columna].is_barrier():
        self.vecinos.append(grid[self.fila - 1][self.columna])

    # OBSTACULO DERECHA
    if self.columna < self.total_filas - 1 and not grid[self.fila][self.columna + 1].is_barrier():
        self.vecinos.append(grid[self.fila][self.columna + 1])
```



```
# OBSTACULO IZQUIERDA
if self.columna > 0 and not grid[self.fila][self.columna - 1].is_barrier():
    self.vecinos.append(grid[self.fila][self.columna - 1])

# Con este metodo lo que hago es comparar el nodo actual con el nodo
other y retornar un False

def __lt__(self, other):
    return False
# Fin de la clase Nodo

# Definimos la funcion heuristica con distancia de manhattan (tambien llamada taxi
cab distance)

def h(p1, p2):
    # P1 es un punto y P2 es otro punto
    # Con esta sintaxis obtenemos las coordenadas en x y en y de cada punto
    x1, y1 = p1
    x2, y2 = p2
    return abs(x1 - x2) + abs(y1 - y2)

#Funcion para reconstruir el camino (algoritmo de backtracking)
def reconstruct_path(came_from, current, draw):
    while current in came_from:
        current = came_from[current]
        current.make_path()
        draw()

def algorithm(draw, grid, start, end):
    count = 0
    #Creamos la "lista" de openset. Esta se crea instanciado el objeto open_set a
    #partir de la clase PriorityQueue
    open_set = PriorityQueue()
    # .put es un metodo de PriorityQueue que es equivalente al append para las
    #listas
    #Metemos al inicio en nuestro openset el nodo de start
    #Tambien le metemos el count, que es una variable que me va a permitir
    #saber luego si yo tengo 2 variables nodos iguales
    #cual fue creada primero, para descartar la otra
    open_set.put((0, count, start))
    #Nuestro open set esta formado por: OPENSET(Fscore,COUNT,NODO), asi
    #va a ser para todos los elementos del openset
```



#Came\_from es un diccionario que nos sirve para asignarle a un nodo, cual es su nodo padre, o "de que nodo proviene"

#Esto sirve para luego aplicar el backtracking

came\_from = {}

#Como vimos antes, en un principio el nodo de start tiene su valor g en 0

#Todos los demás nodos tendrán sus valores g en infinito (y por ende f)

#Aca asignamos a g y a f la transformación de string a float de la palabra "inf"

#Si bien no es infinito, nos sirve como referencia y sirve a nuestro propósito

#Asignamos el valor g infinito a todos los nodos

g\_score = {nodo: float("inf") for fila in grid for nodo in fila}

#Modificamos el valor g del nodo start a 0

g\_score[start] = 0

#Asignamos el valor f infinito a todos los nodos

f\_score = {nodo: float("inf") for fila in grid for nodo in fila}

#Modificamos el valor f del nodo start y lo igualamos a la heurística (ya que  $f = g + h$  y  $g=0 \rightarrow f=h$ )

f\_score[start] = h(start.get\_pos(), end.get\_pos())

#El diccionario open\_set\_hash va guardando el nodo actual. Lógicamente empezamos con el primer nodo

open\_set\_hash = {start}

#Este while basicamente hace que nuestro algoritmo comience a actuar

#Si nuestro openset está vacío, significaría que consideramos todos los nodos vecinos y no hay camino

#posible para seguir. Esto podría pasarnos tanto al inicio (Si estuvieramos rodeados de obstáculos en nuestro nodo start)

#o podría pasarnos en medio de la búsqueda, si los obstáculos no permitieran llegar de A a B de ningún modo

#En ese caso el algoritmo llegaría hasta donde puede y se pararía, teniendo que empezar de nuevo

while not open\_set.empty():

#Revisamos los eventos que vayan llegando. A diferencia de cuando revisábamos esto antes

#Antes lo hacíamos cuando el algoritmo aún no iniciaba, ahora estamos analizando los eventos

#MIENTRAS que el algoritmo se desarrolle

for event in pygame.event.get():

#Si el evento es de salir, salimos del programa

if event.type == pygame.QUIT:

pygame.quit()

#Extraemos el 3er elemento de nuestro openset, osea, el objeto nodo



#Basicamente current sera el nodo actual (es un objeto nodo)  
current = open\_set.get()[2]  
#Removemos el actual del open\_set\_hash para darle paso al siguiente nodo a analizar  
open\_set\_hash.remove(current)

#Si el siguiente nodo es el nodo objetivo, paramos la busqueda y aplicamos el algoritmo de backtraking  
#dado por la funcion reconstruct\_path()  
**if** current == end:  
    reconstruct\_path(came\_from, end, draw)  
    end.make\_end()  
    **return** True

#Si no es el nodo objetivo, lo consideramos simplemente como un nodo que tengo que analizar

#Recordemos que con la funcion update\_neighbors le asignabamos a una lista atributo "vecinos" los nodos

#que rodean a mi nodo actual y que son recorribles(es decir, no son un obstaculo)

#Por ende aca analizamos los vecinos del nodo actual(current) que SI son recorribles

**for** neighbor **in** current.vecinos:  
    #Creamos una variable g\_score temporal que sea igual al costo del camino actual + 1

#Lo que seria equivalente a decir el costo del camino mas pequeño al vecino (hay mil formas de llegar al nodo vecino

# pero dar un solo paso es el que tiene costo de camino mas pequeño)

temp\_g\_score = g\_score[current] + 1

#Establecemos una condicion que dice que, si este costo g temporal es menor que el costo g de mis vecinos (mejor)

#entonces actualizamos el valor del costo g de mi vecino a el valor del g temporal

#Logicamente, como en un inicio seteamos todos los valores g de todos los nodos distintos del nodo inicial en infinito,

#esta condicion se cumplira con cada nodo nuevo, no recorrido

**if** temp\_g\_score < g\_score[neighbor]:  
    #Actualizamos el nodo padre de dicho vecino, el cual sera nuestro nodo actual  
    came\_from[neighbor] = current  
    #Actualizamos el g\_score del vecino  
    g\_score[neighbor] = temp\_g\_score



```
#Actualizamos el f_score del vecino
f_score[neighbor] = temp_g_score + h(neighbor.get_pos(), end.get_pos())
#Finalmente, si el programa compilo bien, deberiamos entrar en esta condicion
# ya que previamente sacamos el current del open_set_hash y no metimos nunca al vecino
#Aca entendemos el por que usamos open_set_hash y no solamente open_set, debido a que el objeto PriorityQueue
#no tiene un metodo para ver si un elemento esta dentro de el o no, simplemente podemos meter o sacar un elemento, no chequear si esta
if neighbor not in open_set_hash:
    #Aumentamos el contador porque metimos algo dentro del open_set
    count += 1
    #Metemos el valor f del vecino, el nuevo count y
    open_set.put((f_score[neighbor], count,
    #Añadimos al openset hash el nodo vecino para
    open_set_hash.add(neighbor)
    neighbor.make_open())

#Llamamos a la funcion lambda draw, que pasamos como parametro en la definicion de esta funcion algorithm
#La llamamos para poder setear el nodo que acabamos de analizar en rojo (closedset)
draw()
#Despues de todo esto, si current no es el nodo de start (porque tiene otro color), le digo
#al programa que setee el nodo actual (que acabo de analizar) en rojo, de esta manera pasa a ser del closedset (ya analizado)
if current != start:
    current.make_closed()

#Esto es para decir, aun no encontramos el camino
return False

# Funcion para construir la cuadrícula con los nodos en la ventana que creamos anteriormente
# Solo le pedimos rows(filas) y no colum(columnas) porque tendremos el mismo numero de ambas
# Lo que hacemos es basicamente crear una matriz o una lista de listas
```



```
def make_grid(rows, width):
    grid = []
    gap = width // rows # Ancho de cada nodo en el grid
    for i in range(rows):
        grid.append([ ])
        for j in range(rows):
            nodo = Nodo(i, j, gap, rows)
            grid[i].append(nodo)

    return grid

# La anterior funcion nos setea los cuadrados, pero no la separacion entre ellos,
para
# eso creamos la funcion draw_grid

# Funcion para dibujar las lineas de cuadricula (lineas entre nodos)
def draw_grid(win, rows, width):
    gap = width // rows
    for i in range(rows):
        pygame.draw.line(win, GRIS, (0, i * gap), (width, i * gap))
    for j in range(rows):
        pygame.draw.line(win, GRIS, (j * gap, 0), (j * gap, width))

# Funcion "main" de dibujo, es la que dibuja todo
def draw(win, grid, rows, width):
    # Cada vez que actualizamos transformamos todo a blanco y luego lo
    cambiamos mas abajo
    # Es una forma de borrar lo que estaba
    win.fill(BLANCO)

    for row in grid:
        for nodo in row:
            # Aca utilizamos el METODO DE LA CLASE NODO .draw
            nodo.draw(win)

    draw_grid(win, rows, width)
    pygame.display.update()

# Para que el programa identifique donde posiciono el mouse
def get_clicked_pos(pos, rows, width):
    gap = width // rows
    y, x = pos
```



row = y // gap  
col = x // gap

**return** row, col

# Iniciamos la definicion del main, donde todo lo que declaramos antes sera utilizado y aplicado junto

```
def main(win, ancho):  
    FILAS = 12 # Numero de filas (y de columnas)  
    grid = make_grid(FILAS, ancho) # Crea el grid(matriz de nodos)  
  
    start = None # Posicion inicial  
    end = None # Posicion final  
  
    run = True # Variable para hacer seguimiento a ver si el algoritmo esta activo  
o no  
    while run:  
        draw(win, grid, FILAS, ancho)  
        # Aca lo que vamos a hacer es recorrer todos los eventos que  
suceden en pygame  
        # Si alguno de ellos corresponde al atributo pygame.ARTRIBUTO,  
asignamos una accion  
        for event in pygame.event.get():  
            # Si el tipo de evento es un pygame.QUIT dejamos de runear  
el programa  
            if event.type == pygame.QUIT:  
                run = False  
  
            # Si el evento es un click izquierdo.Pinta de uno de estos  
colores los nodos  
            if pygame.mouse.get_pressed()[0]: # Si hacemos click  
izquierdo  
                posicion = pygame.mouse.get_pos()  
                fila, columna = get_clicked_pos(posicion, FILAS,  
ancho)  
                nodo = grid[fila][columna]  
  
                # Si no hay nodo inicial, lo seteamos  
                # Establecemos una segunda condicion de que no  
tiene que ser el nodo end, para no sobreescribirlos  
                if not start and nodo != end:  
                    start = nodo  
                    start.make_start()
```



```
# Si no hay nodo final, lo seteamos
# Lo mismo que arriba pero con start
elif not end and nodo != start:
    end = nodo
    end.make_end()

# Si no es ninguno de los anteriores, lo seteamos como
un obstaculo
# Lo mismo que arriba pero con end y start
elif nodo != end and nodo != start:
    nodo.make_barrier()

# Si el evento es un click derecho.Borra los nodos colocados,
o mas bien los reestablece en el color neutro
elif pygame.mouse.get_pressed()[2]: # Si hacemos click
derecho
    posicion = pygame.mouse.get_pos()
    fila, columna = get_clicked_pos(posicion, FILAS,
ancho)
    nodo = grid[fila][columna]
    nodo.reset()
    if nodo == start:
        start = None
    elif nodo == end:
        end = None

# Con esto decimos que si el evento es presionar la barra
espaciadora, se identifiquen los vecinos
# alrededor de los nodos, para saber cuales nodos son
efectivamente por los cuales podemos pasar
# y finalmente ejecutamos el algoritmo A*
# Si la tecla que tocamos es "C" reseteamos todo a blanco
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_SPACE and start and end:
        for fila in grid:
            for nodo in fila:
                nodo.update_neighbors(grid)
        #Para llamar al algoritmo Aestrella, utilizamos
una funcion lambda, que simplemente es una funcion de una sola linea
        #Esta funcion como vemos llama a la funcion
draw
        #Los otros parametros son grid, start y end
        #El usar una f. lambda lo hacemos para poder
meter una llamada a funcion como parametro de otra funcion
```



#Caso contrario no podríamos hacerlo

```
algorithm(lambda: draw(win, grid, FILAS,
ancho),
          grid, start, end)

        if event.key == pygame.K_c:
            start = None
            end = None
            grid = make_grid(FILAS, ancho)

    pygame.quit()

main(VENTANA, ANCHO)
```

## 5.9 STRIPS

### 5.9.1 Domain

```
;Primero vamos a definir el dominio
(define (domain orden-cajas2)
  (:requirements :negative-preconditions)
  ;Definimos los predicados con los que vamos a trabajar
  (:predicates
    (caja ?x) (sobre ?x ?y) (mesa ?x) (nadaEncima ?x) (base ?x)
    (basepos ?x) (clear ?x)
  )
  ;Definimos la funcion sacar, la cual me sirve para desapilar (pop)
  ;de mi pila o stack
  (:action sacar
    ;Los parametros con los que trabajaremos seran simplemente 2
    cajas
    :parameters (?c1 ?c2)
    ;Las precondiciones son: Que c1 y c2 sean cajas que c1 este
    ;encima de c2 y que no haya nada encima de c1 (caja superior)
    ;Ademas c1 no debe ser la caja base
    :precondition (
      and (caja ?c1) (caja ?c2) (sobre ?c1 ?c2)
          (nadaEncima ?c1) (not(base ?c1)))
    )
    ;Si esto se cumple el efecto sera que c1 no este mas sobre c2,
    ;que se encuentre desapilado y que no haya nada encima de c2
    :effect (
```



```
        and (not(sobre ?c1 ?c2)) (mesa ?c1) (nadaEncima ?c2)
    )
)

;Definimos la funcion sacar-base, la cual es un caso especial de la
funcion sacar para el caso donde la caja a retirar es la caja base
(:action sacar-base
    ;Los parametros con los que trabajaremos es 1 caja y la base,
pero ahora la base como parametro de posicion
    :parameters (?c ?b)
        ;Las precondiciones son: que c sea una caja, que b sea la
posicion base, que c este en la base y que la base no este vacia
    :precondition (
        and (caja ?c) (basepos ?b) (base ?c) (not(clear ?b))
        (nadaEncima ?c)
    )

    ;Cumpliendose esto, c pasaria a esta desapilada, por ende no se
encontraria en la base, y ademas la posicion base quedaria vacia
    :effect (
        and (mesa ?c) (not(base ?c)) (clear ?b)
    )
)

;Definimos la funcion poner, que me sirve para desapilar (push) una
caja de mi pila o stack
(:action poner
    ;Vamos a trabajar nuevamente con 2 cajas
    :parameters (?c1 ?c2)
        ;Esta vez las precondiciones son que c1 y c2 sean cajas, que c1
que sencuentre desapilado y que no haya ninguna caja encima de c2
        ;Si esto se cumple entonces el robot colocara la caja c1 sobre
c2, c1 dejara de estar desapilado y c2 comenzara a tener una caja
encima
    :precondition (
        and (caja ?c1) (caja ?c2) (mesa ?c1) (nadaEncima ?c2)
    )

    :effect (
        and (sobre ?c1 ?c2) (not(mesa ?c1)) (not(nadaEncima ?c2))
    )
)
```



;Definimos la funcion poner-base, caso especial de poner, donde en este caso lo que se coloca es la caja en la posicion base

```
(:action poner-base
```

;Los parametros con los que trabajamos son una caja y la posicion base

```
:parameters (?c ?b)
```

;Las precondiciones en este caso son que c sea una caja, que la posicion base sea b, que c se encuentre desapilado (sobre la mesa), no tenga

```
;nada encima y la posicion base este despejada.
```

```
:precondition (
```

```
    and (caja ?c) (basepos ?b) (mesa ?c) (nadaEncima ?c) (clear
```

```
?b)
```

```
)
```

;Si todo esto se cumple el efecto sera que c dejara de estar desapilada, se coloca como caja base y la posicion base deja de estar vacia

```
:effect (
```

```
    and (not(mesa ?c)) (base ?c) (not(clear ?b))
```

```
)
```

```
)
```

```
)
```

## 5.9.2 Problem

;Definimos el problema, modifiable segun el orden inicial que se de en el examen final y el orden final que se pida

;En este caso el orden inicial es, de arriba hacia abajo Tuercas->Clavos->Arandelas->Tornillo

;Queremos llegar a Arandelas->Tuercas->Tornillos->Clavos

```
(define (problem orden-cajas-problema) (:domain orden-cajas2)
```

```
    (:objects arandelas tuercas clavos tornillos base1)
```

```
    (:init
```

```
        (caja arandelas) (caja tuercas) (caja clavos) (caja tornillos)
```

```
(basepos base1)
```

```
        (sobre tuercas clavos) (sobre clavos arandelas) (sobre arandelas
```

```
tornillos)
```

```
        (base tornillos) (nadaEncima tuercas) (not(clear base1))
```

```
)
```



```
(:goal (
    and
        (sobre arandelas tuercas) (sobre tuercas tornillos) (sobre
    tornillos clavos) (base clavos)
    )
)
```

## 6. Ejemplo de aplicación

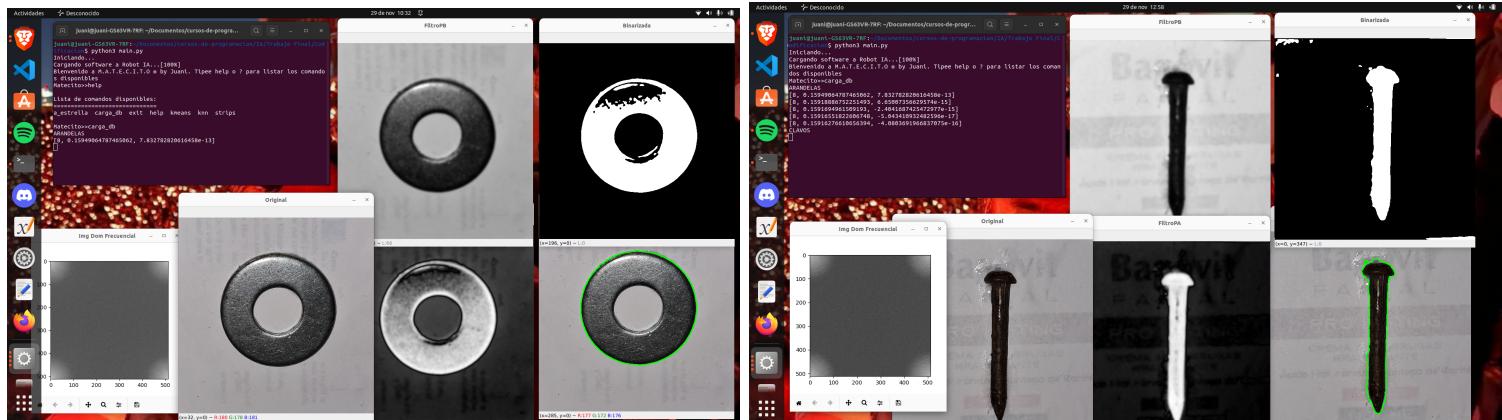
Dado que a lo largo de todo el informe se presentaron las imágenes de los distintos resultados, producto de aplicar los filtros, obtener los contornos, las gráficas correspondientes a cada método, el laberinto que tiene que cruzar el robot, etc, en esta sección se presentan simplemente algunas capturas de pantalla de la consola y la ejecución del programa en sí

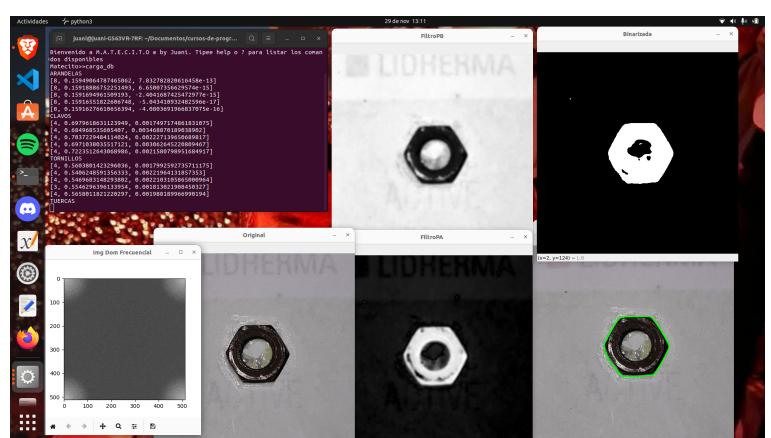
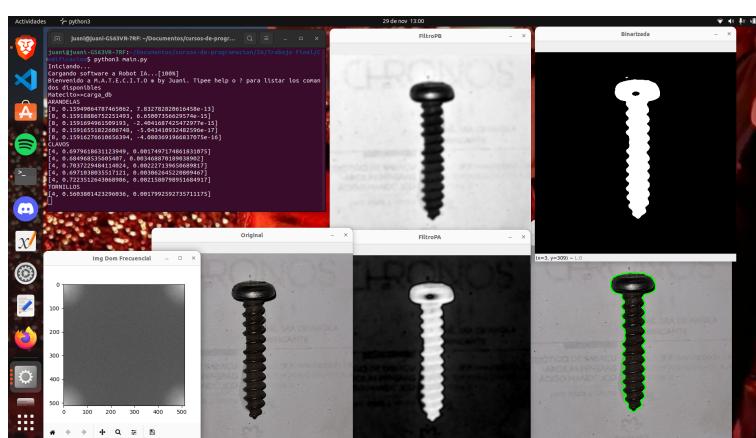
### Inicio de consola y lista de comandos

```
juani@juani-GS63VR-7RF: ~/Documentos/cursos-de-programacion/IA/Trabajo Final/Codificación$ python3 main.py
Iniciando...
Cargando software a Robot IA...[100%]
Bienvenido a M.A.T.E.C.I.T.O o by Juani. Tipee help o ? para listar los comandos disponibles
Matecito>>[]

juani@juani-GS63VR-7RF: ~/Documentos/cursos-de-programacion/IA/Trabajo Final/Codificación$ python3 main.py
Iniciando...
Cargando software a Robot IA...[100%]
Bienvenido a M.A.T.E.C.I.T.O o by Juani. Tipee help o ? para listar los comandos disponibles
Matecito>>help
Lista de comandos disponibles:
=====
a_estrella carga_db exit help kmeans knn strips
Matecito>>[]
```

### Carga de base de datos





## Kmeans y Knn

```
juani@juani-GS63VR-7RF: ~/Documentos/cursos-de-prog... [2] + 0:0 python3
[6, 0.1597059724802038, -1.1282678306706147e-11]
Matecito>>>kmeans 3
Imagen 1
[6, 0.15986703979482275, -1.221595165510722e-12]
Imagen 2
[4, 0.7006032864465326, 0.0026894150803553484]

Imagen 3
[8, 0.15915949551880512, -1.124577244723128e-17]
Imagen 4
[4, 0.5466837141754873, 0.0016643971024612079]
Iteracion 1
Centrodes:
[[8, 0.15949064787465062, 7.832782820616458e-13], [6, 0.1602454561363026, 4.490390115578456e-11], [4, 0.7037229484114024, 0.00227139650689817], [3, 0.5546296396133954, 0.001813021908450327]]
Iteracion 2
Centrodes:
[[8, 0.15949064787465062, 7.832782820616458e-13], [6, 0.1597867001706275, 7.072321592985247e-15], [4, 0.697961863112395, 0.0017497174861831077], [3, 0.5546296396133954, 0.001813021908450327]]
Iteracion 3
Centrodes:
[[8, 0.15949064787465062, 7.832782820616458e-13], [6, 0.1597867001706275, 7.072321592985247e-15], [4, 0.697961863112395, 0.0017497174861831077], [3, 0.5546296396133954, 0.001813021908450327]]
Imagenes guardadas en la carpeta Output/Kmeans
Matecito>>
```

```
juani@juani-GS63VR-7RF: ~/Documentos/cursos-de-prog... [2] + 0:0 python3
396133954, 0.001813021908450327]]
Iteracion 2
Centrodes:
[[8.0, 0.15949064787465062, 7.832782820616458e-13], [6.0, 0.1597867001706275, 7.072321592985247e-15], [4.0, 0.697961863112395, 0.0017497174861831077], [3.0, 0.5546296396133954, 0.001813021908450327]]
Iteracion 3
Centrodes:
[[8.0, 0.15949064787465062, 7.832782820616458e-13], [6.0, 0.1597867001706275, 7.072321592985247e-15], [4.0, 0.697961863112395, 0.0017497174861831077], [3.0, 0.5546296396133954, 0.001813021908450327]]
Imagenes guardadas en la carpeta Output/Kmeans
Matecito>>kn
Imagen 1
[6, 0.15986703979482275, -1.221595165510722e-12]
Imagen 2
[4, 0.7006032864465326, 0.0026894150803553484]
Imagen 3
[8, 0.15915949551880512, -1.124577244723128e-17]
Imagen 4
[4, 0.5466837141754873, 0.0016643971024612079]
Imagenes guardadas en la carpeta Output/Knn
Matecito>>
```

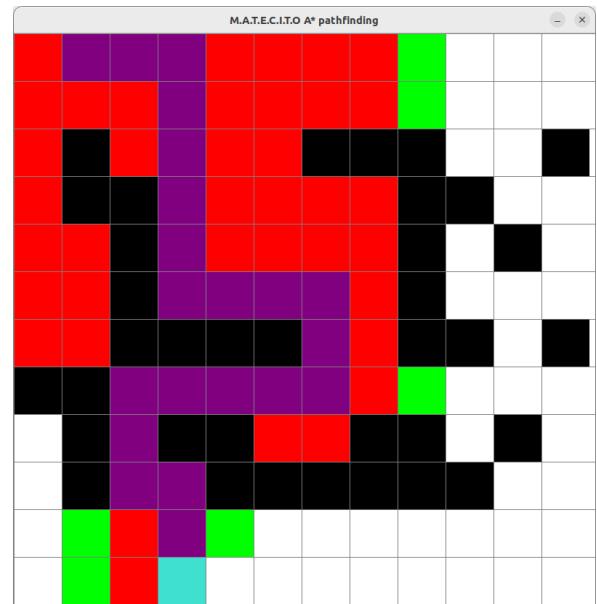
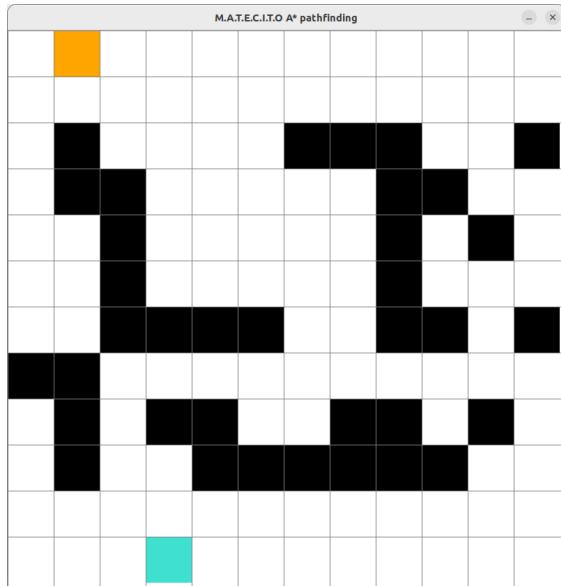
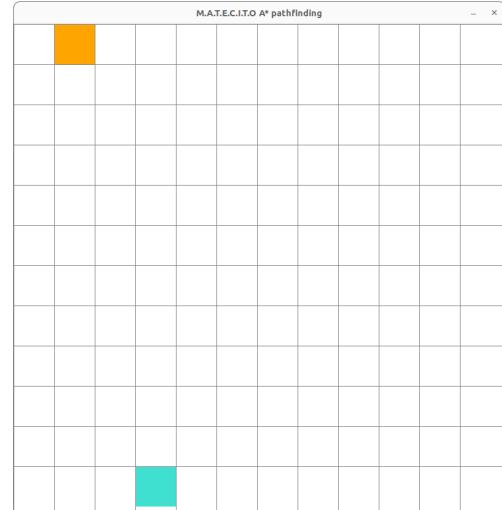


## A estrella

```
juani@juani-GS63VR-7RF: ~/Documentos/cursos-de-program... [+] - x
[6, 0.15986703979482275, -1.221595165510722e-12]
Imagen 2
[4, 0.7006032864465326, 0.0026894150803553484]
Imagen 3
[8, 0.15915949551880512, -1.124577244723128e-17]
Imagen 4
[4, 0.5466837141754873, 0.0016643971024612079]
Imagenes guardadas en la carpeta Output/Knn
Matecito>>a_estrella

#####
Instrucciones:
1. Ingrese el punto de inicio con click izquierdo
2. Ingrese el punto de destino con click izquierdo
3. Ingrese obstaculos con click izquierdo
4. Si se equivoca, puede borrar con click derecho
5. Presione ESPACIO para comenzar busqueda
6. Prestone C para restaurar ventana a default
#####

pygame 2.1.2 (SDL 2.0.16, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
□
```





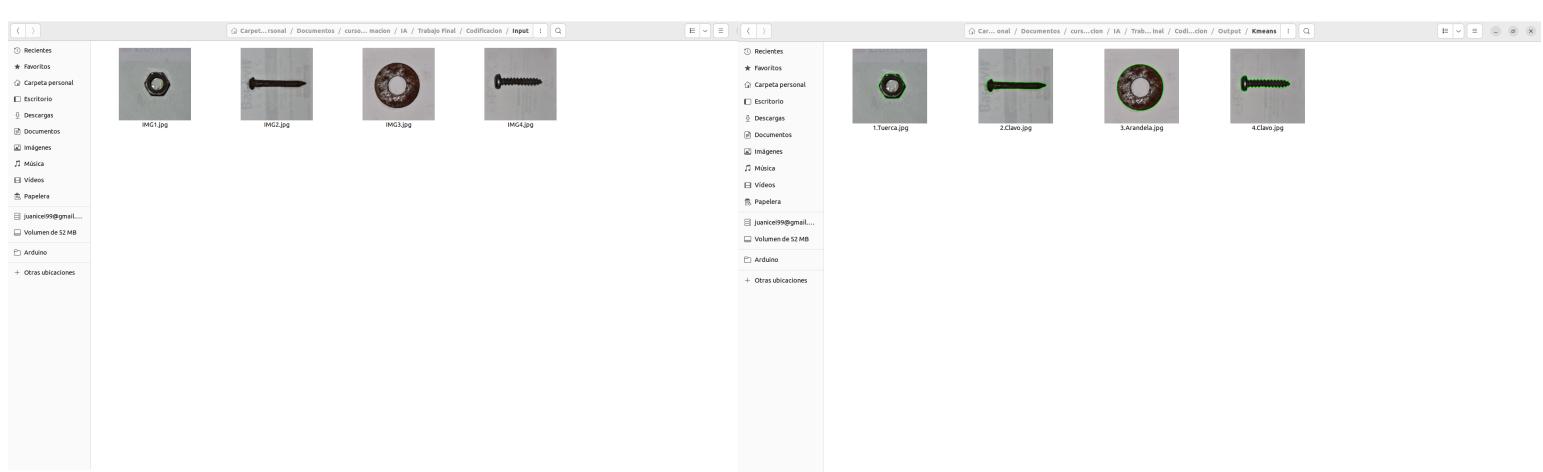
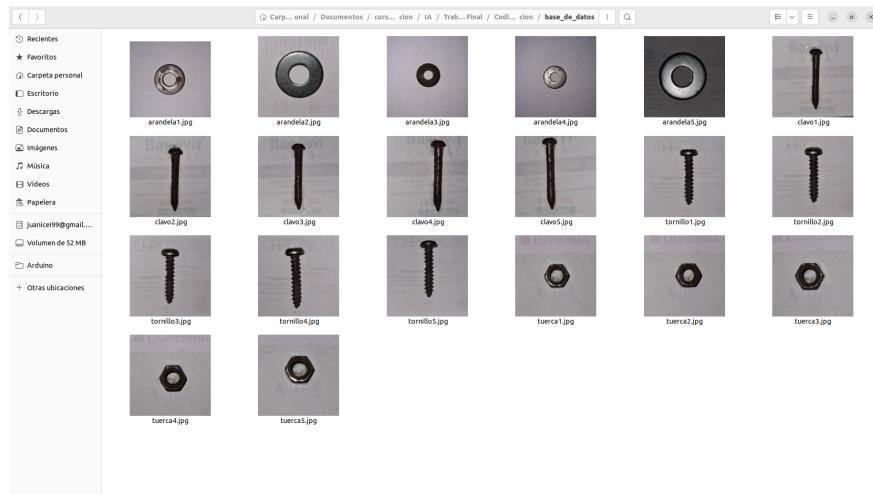
## STRIPS

```
juani@juani-GS63VR-7RF: ~/Documentos/cursos-de-programacion/IA/Trabajo_Final/codificacion$ python3 main.py
Iniciando...
Cargando software a Robot IA...[100%]
Bienvenido a M.A.T.E.C.I.T.O ® por Juani. Tipee help o ? para listar los comandos disponibles
Matecito>>strips
El plan trazado es:

sacar tuercas clavos (1)
sacar clavos arandelas (1)
sacar arandelas tornillos (1)
sacar-base tornillos base1 (1)
poner-base clavos base1 (1)
poner tornillos clavos (1)
poner tuercas tornillos (1)
poner arandelas tuercas (1)

El costo del camino es: 8 unidades
Matecito>>
```

## Base de datos, Input, Output





## 7.Resultados

Las especificaciones técnicas, los datos utilizados y los datos de prueba fueron cuidadosamente analizados en las secciones 4.2 y 4.3 del presente informe.

Luego del análisis de una muestra de 10 resultados podemos concluir que, en condiciones satisfactorias (buena iluminación, cámara sin desperfectos, buenas imágenes en la base de datos,etc ):

- La clasificación mediante el método KMEANS, teniendo en cuenta la aleatoriedad en la selección de sus centroides, es acertada un 100% de las veces
- La clasificación mediante el método KNN, es acertada un 100% de las veces

En cuanto a la planificación y la trayectoria, no se estudió una cantidad fija de veces, pero en ninguno de los intentos que se realizaron durante las implementaciones dieron error alguno, por ende podemos decir que acierta el 100% de las veces.

Después de observar las estadísticas puedo decir que el software cargado al robot es muy preciso, dando lugar al overfitting en algunos casos, sin embargo sigue cumpliendo su objetivo.

## 8.Conclusiones

Para concluir con este informe me gustaría mencionar que se logró la implementación de un programa compacto, de facil utilización, con buenos resultados independientemente del algoritmo que se utilice.

Cabe aclarar que la concepción del mismo no fue un procedimiento simple, hubo una gran investigación de por medio y se cometieron muchos errores, los cuales fueron solventados con distintas estrategias o de plano probando algún método distinto al pensado inicialmente.

Algunas de las dificultades con las que me encontré a la hora de trabajar con este proyecto:

- **Lenguaje de programación:** elegí Python como lenguaje principal en esta implementación debido a que personalmente me parece muy potente y versátil, con una gama muy amplia de módulos para utilizar. También el hecho de ser un lenguaje débilmente tipado lo hace un candidato óptimo para el desarrollo de software de Machine Learning e Inteligencia Artificial, ya que es mucho más moldeable. Sin



embargo más allá de la adquisición de la imagen y la facilidad de programación que provee openCV (módulo de visión artificial usado) es innegable que esta implementación tiene detrás un fuerte trabajo matemático, para el cual indudablemente Matlab es el candidato número 1 en cuanto a herramientas que facilitan el cálculo. Es por esto que considero que la elección del lenguaje no fue una tarea fácil.

- **Infinidad de técnicas y módulos para lograr la misma tarea:** como mencione anteriormente, la ventaja de la utilización de Python es la infinidad de módulos que posee y nos permite realizar todo lo que nos propongamos, sin embargo en su mayor virtud reside su mayor desventaja, es tanta la cantidad que a veces es difícil definir la calidad de los mismos, y cual es óptimo para nuestro caso particular.  
Un ejemplo particular de mi implementación se dio a la hora de realizar la binarización de la imagen, donde empecé utilizando un método de OpenCV conocido como Canny, que si bien daba resultados muy prometedores, era tan preciso que identificaba cada detalle de la imagen, como si fuera un boceto, por lo que a la hora de identificar los bordes no lograba adquirir contornos cerrados, si no que el programa identificaba contornos abiertos a lo largo de toda la figura, lo que causaba errores de “ZeroDivision” a la hora de calcular algunos parámetros que dependen del perímetro de la figura, en la extracción de características.
- **Condiciones del entorno:** a veces uno se concentra tanto en los detalles internos de la implementación que se olvida de un detalle fundamental que lo cambia todo, las condiciones en las que nos llegará el dato desde el exterior, para poder trabajarla y entregar el resultado.  
En un principio opté por utilizar como características de mis imágenes todas características geométricas, como lo son la circularidad, convexidad, aspect ratio (elasticidad), solidez, etc, con las cuales había logrado una segmentación muy buena. Sin embargo, hasta que la Dra. Rivera no me lo comentó, no tuve en cuenta el hecho de que alguna de las figuras podría estar, por ejemplo, rotada.  
Dado que las característica que tome en un principio no eran invariantes a dicha rotación, especialmente el aspect ratio, la segmentación se vuelve inutil. Es por esto que debí recurrir a mantener solamente la aproximación polinomial como característica geométrica, y trabajar con los momentos invariantes de Hu, para solventar esta dificultad.
- **Adaptación de los algoritmos:** si bien la esencia del algoritmo que usamos en la implementación y su forma de trabajo es la misma, a veces se hace necesario generar algunas modificaciones o técnicas que nos permiten adaptarlo a lo que nosotros necesitamos, como fue el caso de la referenciación de los clusters por medio de los centroides en Kmeans, explicado con anterioridad en el apartado 5.6.

En términos generales el software es eficiente, se logran buenos resultados en un tiempo relativamente corto. Como principales ventajas destacaría su intuitividad y compactación, que me permiten un uso simple del programa, sin requerir de herramientas externas,



además de un amplia gama de opciones que nos permiten controlar varios aspectos del robot.

Como principales desventajas destacaría su falta de adaptación a otros modelos, ya que fue programado para este fin específico, con un K fijo para el método Kmeans y un número de nodos fijo para el laberinto planteado para el recorrido del robot, sin embargo son aspectos fácilmente modificables y podría ser adaptado sin problema con los conocimientos y la dedicación correspondiente.

Otra desventaja es su tamaño. Debido a su amplia cantidad de funciones, es un programa poco portable, lo que dificulta su carga en placas de desarrollo o microcontroladores (como podría ser una Raspberry Pi pico por ejemplo). Sin embargo con una memoria de 1GB para el Robot podremos cargarle el programa sin problema.

Una implementación que sin duda implementaría a futuro sería el colocar una cámara que capte video, y de esta manera identificar los objetos en tiempo real. También me gustaría implementar la función de reconocimiento de rostros de los estudiantes, utilizando como base de datos las fotos de su legajo, para el día de mañana gestionar los préstamos del centro de estudiantes con inteligencia artificial.

## 9.Bibliografía y/o referencias

- <https://www.youtube.com/watch?v=R82EcsCgnfq>
- <https://www.youtube.com/watch?v=YyRlvbLC99U>
- <https://programarfacil.com/blog/vision-artificial/detector-de-bordes-canny-opencv/#:~:text=Para%20detectar%20los%20bordes%20con,Gaussiano%20para%20eliminar%20el%20ruido.>
- <https://opencv.org/>
- [https://github.com/DavidReveloLuna/ProcesamientoDatos/blob/master/Scripts/8\\_FiltroFFTVideo.ipynb](https://github.com/DavidReveloLuna/ProcesamientoDatos/blob/master/Scripts/8_FiltroFFTVideo.ipynb)
- <https://docs.python.org/3/library/cmd.html>
- <https://www.youtube.com/watch?v=1gszEk8rUS4>
- [https://pygame.readthedocs.io/en/latest/1\\_intro/intro.html](https://pygame.readthedocs.io/en/latest/1_intro/intro.html)
- [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)
- <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2>
- <https://www.youtube.com/watch?v=JtiK0DOel4A>
- <https://www.ceupe.com/blog/vision-artificial.html>
- <https://www.javatpoint.com/opencv-erosion-and-dilation#:~:text=Erosion%20and%20Dilation%20are%20morphological.or%20structure%20of%20an%20object.>
- <https://stackoverflow.com/questions/3503879/assign-output-of-os-system-to-a-variable-and-prevent-it-from-being-displayed-on>
- <https://poiritem.wordpress.com/2009/11/16/6-4-2-agentes-inteligentes-y-la-naturaleza-de-su-entorno/>
- <https://es.wikipedia.org/wiki/K-medias>
- [https://es.wikipedia.org/wiki/Momentos\\_de\\_imagen](https://es.wikipedia.org/wiki/Momentos_de_imagen)



<https://planning.wiki/>

<https://es.wikipedia.org/wiki/STRIPS>

<https://www.fast-downward.org/>

<https://www.jair.org/index.php/jair/article/view/10457>

<https://cogsys.uni-bamberg.de/theses/reissner/MAReissner.pdf>

[https://hmn.wiki/es/Consistent\\_heuristic](https://hmn.wiki/es/Consistent_heuristic)