UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

Inteligencia Artificial I

Proyecto Final: Visión Artificial

Alumno: Juan Stella
Año: 2024
Carrera: Ingeniería Mecatrónica

Índice

1. Resumen del proyecto

El trabajo final plantea un problema donde se debe identificar el contenido mediante visión artificial, de cuatro cajas con distintos contenidos:

- Tornillos
- Clavos
- Arandelas
- Tuercas

Además, el robot debe luego reorganizar las cajas en un orden pedido en el momento de la evaluación. Por último, el robot debe transportar las cajas desde un punto a otro del aula, que también serán determinados en el momento de evaluación.

Los algoritmos utilizados son:

- Para reconocimiento de imágenes: KNN y KMeans
- Para la reorganización de las cajas: STRIPS
- Para el transporte de las cajas: A* (A estrella)

Como resultados, se consiguió una buena precisión a la hora de reconocer las imágenes, con un 90% de precisión o más (con los algoritmos KNN y KMeans). Además, el algoritmo de STRIPS reorganiza las cajas de manera correcta y por último el algoritmo A* encuentra el camino más corto para poder transportarlas.

2. Introducción

2.1. Visión artificial

La visión artificial es una disciplina de la inteligencia artificial y la informática que permite interpretar y comprender el contenido visual del mundo. Esta tecnología utiliza cámaras, algoritmos de procesamiento de imágenes y aprendizaje automático para captar, analizar y procesar imágenes y videos de forma similar a como lo hace el ojo humano.

El objetivo principal de la visión artificial es dar a las máquinas la capacidad de ver y comprender el entorno visual, permitiendo la automatización de tareas que requieren análisis visual. En el contexto de nuestro proyecto, la visión artificial se emplea para que el robot pueda identificar el contenido de las cajas apiladas, clasificarlas correctamente y ejecutar tareas de reorganización y transporte basadas en la información visual obtenida.

2.2. Problema a resolver

1. Adquisición de imágenes: El primer paso es tomar las imágenes, las cuales son tomadas en un entorno con buena iluminación y un fondo adecuado. Luego, se hace una sanitización de las imágenes, esto es por ejemplo recortar bordes que no pertenezcan al fondo u objeto en sí.
   Con éstas imágenes ya sanitizadas formamos nuestra base de datos, y podemos pasar a la siguiente etapa.

2. Preprocesamiento de imágenes: Este paso consiste en dejar a la imagen en condiciones de ser utilizada para luego poder extraer características. Algunos pasos son:
   a. Ajustar el tamaño de la imagen
   b. Pasar a escala de grises
   c. Contrastar
   d. Histogramas
   e. Eliminar ruido
   f. Binarización
   g. Detección de bordes

3. Extracción de características: Se calculan distintas características que nos ayuden a identificar la imagen analizada. Idealmente, estas características están diferenciadas de una clase a otra.

4. Clasificación: El siguiente paso es clasificar las imágenes mediante los algoritmos KNN y KMeans, donde se toma una imagen y se comparan estas características mediante distintos métodos, para luego clasificar a la imagen en una posible categoría. En este caso clasificamos las imágenes en tornillos, clavos, tuercas o arandelas.

5. Reordenamiento: Luego el problema nos dice que debemos reordenar las cajas en un orden determinado en el momento. Esto se realiza mediante el lenguaje de código STRIPS (Stanford Research Institute Problem Solver), que logra obtener una secuencia de acciones que debe realizar el robot para alcanzar el objetivo deseado.

6. Transporte: Por último, mediante el algoritmo A* debemos encontrar el camino más corto para transportar estas cajas a un destino específico, que en este caso corresponde a un espacio de trabajo que representa un aula de la universidad.

3. Especificación del agente

Definición de Agente: Un agente es una entidad autónoma que percibe su entorno a través de sensores y actúa sobre ese entorno utilizando actuadores para alcanzar objetivos específicos. Los agentes pueden ser físicos, como robots, o virtuales, como programas de software, y están diseñados para tomar decisiones y realizar acciones de manera autónoma para lograr sus metas.

3.1. Tipo de agente

En el caso de la aplicación de este proyecto, se considera que el agente es un agente basado en objetivos.
Esta consideración parte de que el agente no solo actúa de acuerdo a sus percepciones del entorno, si no que además tiene un objetivo o meta a la cuál llegar:

- En el caso de STRIPS, la meta es el orden de las cajas
- En el caso de A* tenemos la casilla o lugar de destino

En el caso de los algoritmos KNN y KMeans, podríamos decir que nuestro agente se estaría comportando mayormente como un agente basado en utilidad, ya que nos interesa saber qué tanta precisión tiene a la hora de identificar qué objeto está viendo en el momento

3.2. Tabla REAS

Agente	Rendimiento	Entorno	Actuadores	Sensores
Robot	Máxima calidad de toma y procesamiento de imágenes Maximizar precisión de clasificación Minimizar los tiempos de clasificación	Cajas Aula Mesa	Teclado, mouse Gripper Lámpara	Cámara Sensor de distancias Sensor de agarrado del objeto

	Minimizar el tiempo y cantidad de pasos para el reordenamiento Minimizar el tiempo y la distancia recorrida cuando se transporta			

3.3. Entorno

Describiremos cada entorno mencionado anteriormente por separado, ya que cada uno puede tener distintas características.

Entorno: Cajas (clasificación)	
Observabilidad	Parcialmente Observable
Episódico - Secuencial	Episódico
Estocástico - Determinista	Determinista
Discreto - Continuo	Continuo
Estático - Dinámico	Estático
Cantidad de agentes	Individual

Entorno: Mesas (ordenamiento de cajas)	
Observabilidad	Totalmente Observable
Episódico - Secuencial	Secuencial
Estocástico - Determinista	Determinista
Discreto - Continuo	Discreto
Estático - Dinámico	Estático
Cantidad de agentes	Individual

Entorno: Aula	
Observabilidad	Totalmente Observable
Episódico - Secuencial	Secuencial
Estocástico - Determinista	Determinista
Discreto - Continuo	Discreto
Estático - Dinámico	Estático
Cantidad de agentes	Individual

4. Diseño del proyecto

4.1. Obtención de imágenes y creación de base de datos

Primero debemos obtener las imágenes, las cuales se toman dentro de una caja con fondo verde y se iluminan con una lámpara externa. Se toman las imágenes con la cámara de un celular, y tomando el objeto en distintas posiciones, por ejemplo podemos introducir algún tipo de rotación, escala o traslación para tener una mayor variedad de fotos en la base de datos. En este caso en particular, se tomaron 5 fotos para cada clase.

Luego se les realizó un proceso de sanitización a las imágenes, donde se eliminaron ciertas impurezas (por ejemplo el borde de la caja en una foto defectuosa). Terminado este proceso, se cargaron todas las imágenes a la clase “BD”, correspondiente a la base de datos, mediante la especificación del directorio de cada imagen y también la clase en particular.

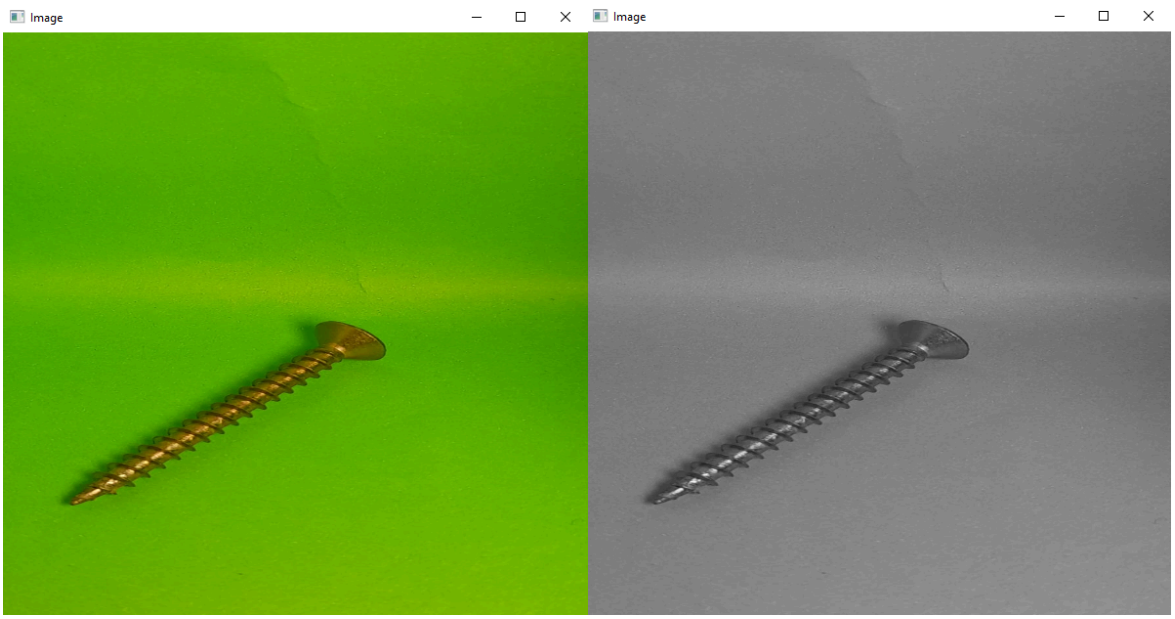
Aclaración: Las imágenes de la versión final fueron tomadas en el aula de la facultad, ya que fue necesaria la misma iluminación para el correcto funcionamiento.

4.2. Preprocesamiento de las imágenes

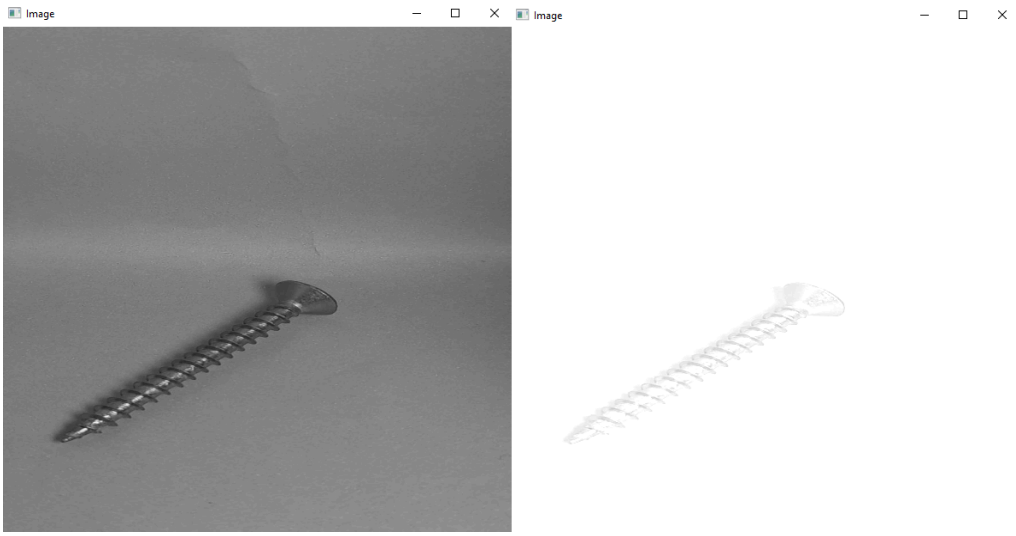
En este proceso primero debemos abrir una imagen desde su directorio donde está almacenada, para luego poder modificarla. Los pasos de su procesamiento son los siguientes:

a) Escalado: El escalado de la imagen se realiza para tener todas las imágenes en un tamaño determinado, ya que éstas representan un conjunto de píxeles cuyos valores de intensidad se almacenan en una matriz, es decir, hacemos que todas las imágenes tengan el mismo tamaño de la matriz que almacena sus valores.

b) Escala de grises: Toda imagen tiene 3 canales, R (correspondiente al rojo), G (correspondiente al verde) y B (correspondiente al azul). Nosotros debemos trabajar con un solo canal, por lo que se pasa a una escala de grises donde la intensidad de los pixeles varía entre 0 y 255, donde 0 es el negro y 255 es el blanco.

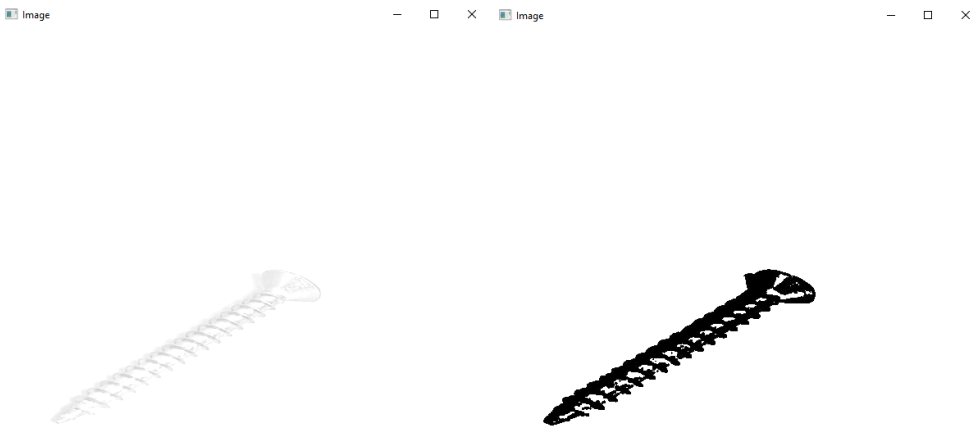


c) Contraste: Nos contrasta la imagen con la intención de resaltar mejor los detalles. En este caso la imagen se vuelve más brillante, ya que le estamos sumando un valor a su intensidad. Si bien esto pareciera empeorar la calidad de la imagen, luego obtuvimos una gran eliminación de ruido gracias a éste método.

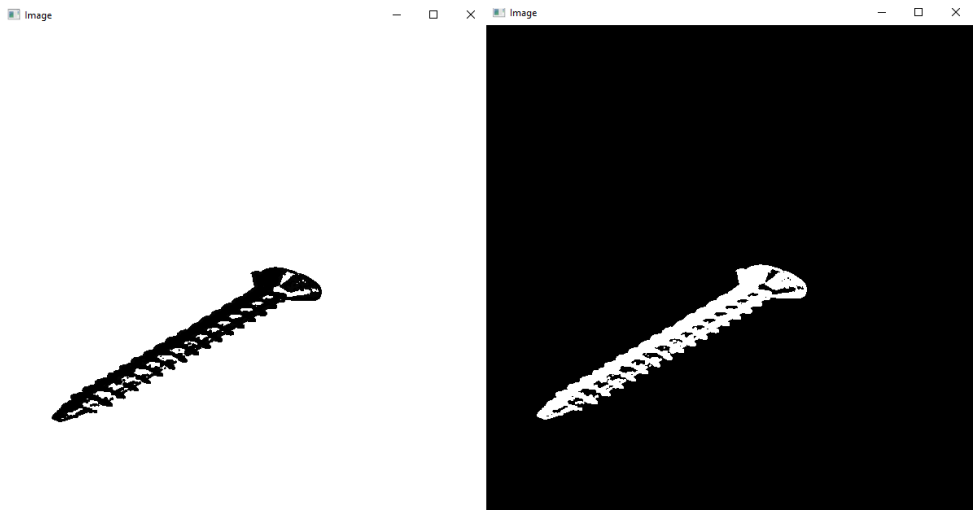


d) Eliminar ruido: La función para eliminar ruido suaviza esta imagen aplicando un filtro de desenfoque gaussiano, lo que ayuda a reducir el ruido y mejora la calidad general de la imagen. El tamaño del kernel es de 3x3 píxeles. Un kernel pequeño como este es útil para reducir el ruido mientras mantiene los detalles de la imagen.

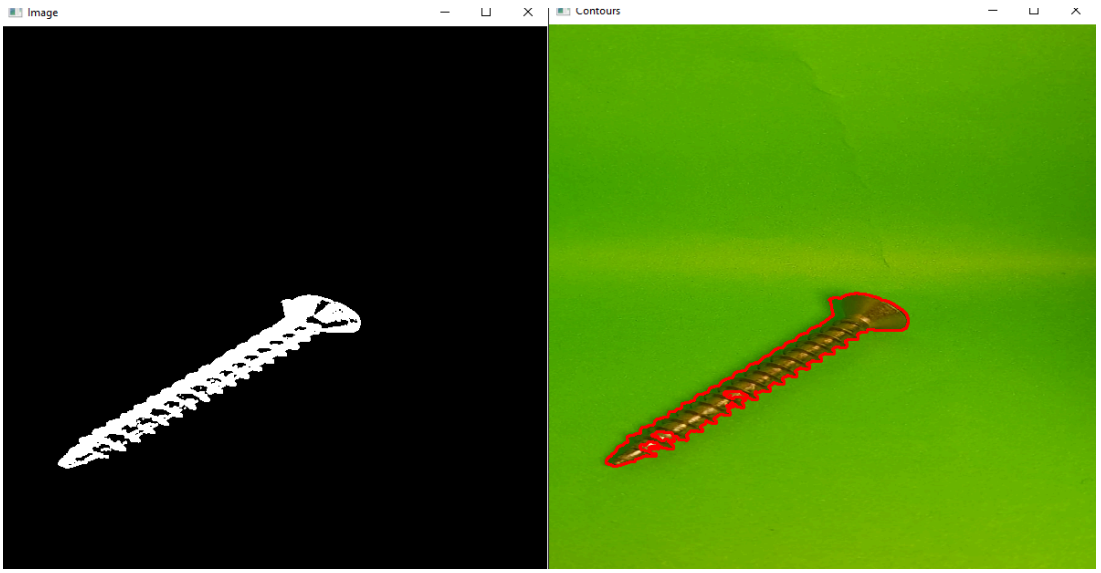
e) Histogramas: Esta función aplica la ecualización del histograma a una imagen en escala de grises. La ecualización del histograma es una técnica de procesamiento de imágenes que ajusta los contrastes de una imagen mediante el ajuste del histograma.
La ecualización del histograma redistribuye los valores de intensidad de manera que el histograma de la imagen resultante sea lo más uniforme posible. Esto se logra mapeando los valores de intensidad originales a nuevos valores de intensidad que están distribuidos más uniformemente.
Este proceso mejora el contraste de la imagen, especialmente en áreas donde las intensidades de los píxeles están muy concentradas en un rango estrecho.



f) Binarización: Se utiliza para resaltar una imagen en escala de grises donde se desea resaltar ciertos elementos, por ejemplo un tornillo, en blanco sobre un fondo negro. La función binarizar umbraliza la imagen usando el método de Otsu para determinar automáticamente el mejor umbral, y luego invierte los valores para resaltar los elementos de interés.
Luego, se invierten los colores de la imagen para poder tener la pieza en blanco y el fondo en negro.



g) Detectar contornos: Se utiliza para detectar contornos en una imagen binarizada. Un contorno es una curva que une todos los puntos continuos a lo largo de un límite, que tienen el mismo color o intensidad. En este caso se utilizan los contornos externos, es decir, se ignora cualquier contorno interno que se pueda obtener.



4.3. Extracción de características

Ahora que ya contamos con la imagen preprocesada, podemos extraer ciertas características de la misma, para luego poder hacer la respectiva clasificación.

Los pasos que se tomaron son los siguientes:

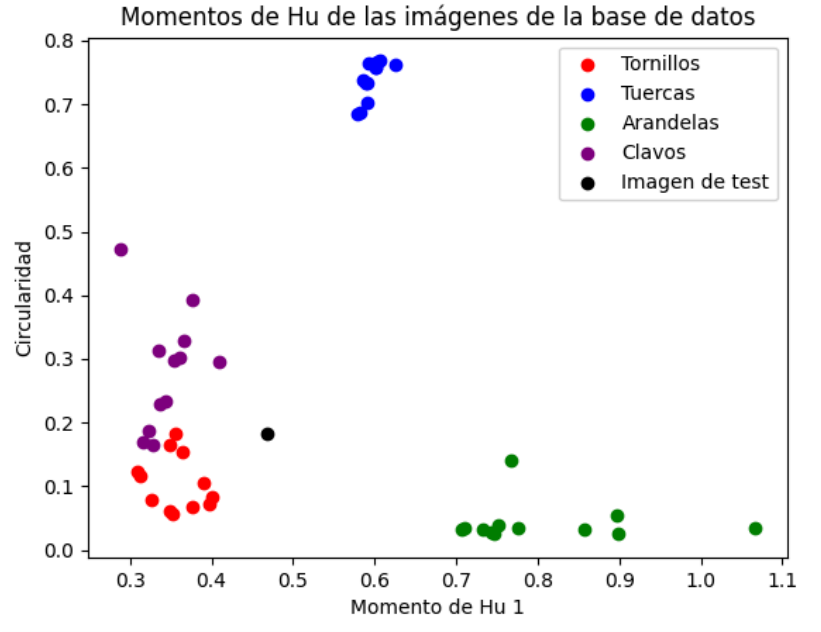
a) Circularidad: Luego de detectar los contornos de la imagen, proseguimos a calcular la circularidad, es decir, nos ayuda a distinguir los objetos cuyos contornos son más circulares. Como se muestra más adelante, en la práctica el resultado no es totalmente intuitivo, ya que los objetos que resultaron con menor circularidad fueron las arandelas.

b) Momentos de HU: Los Momentos de Hu son un conjunto de siete invariantes de imagen que se utilizan para describir las formas de los objetos en las imágenes. Éstos son invariantes a la traslación, rotación y escala de los objetos. Estas propiedades hacen que los Momentos de Hu sean especialmente útiles para tareas de reconocimiento de patrones y análisis de formas en imágenes.

$$
\begin{aligned}
h_0 &= \eta_{20} + \eta_{02} \\
h_1 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
h_2 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
h_3 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
h_4 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
h_5 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})] \\
h_6 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_2
\end{aligned}
\tag{6}
$$

En este caso calculamos los 7 momentos de HU mediante una función de cv2 y luego le aplicamos logaritmo para que todos estén en la misma escala, ya que pueden tener rangos de valores muy distintos (por ejemplo, algunos pueden tener valores muy cercanos a 0, mientras que otros cercanos a 20).

c) Elección de características: Después de extensas pruebas, se determinó que el mejor uso (y que mejor se podía visualizar, ya que usando dos podemos visualizarlo en el plano) eran las características de circularidad y el momento de HU uno. Esto nos separa a las distintas clases, dando cierto margen para que se haga una correcta clasificación.

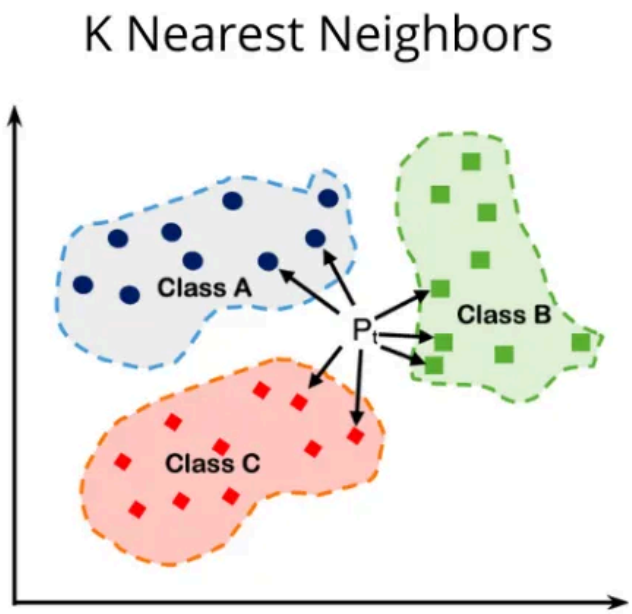


También se debe aclarar que se realizó un procesamiento manual de las características, es decir, se separaron las características de manera manual. Esto se hace con el objetivo de que haya menos margen de confusión entre clases y se pueda obtener una mayor precisión. Además, si bien el momento de HU cero no se utilizó para la comparación o cálculo de distancias, sí se utilizó para desplazar el momento de HU uno, a fin de hacer más notoria la diferencia entre clases.
Un ejemplo de este procesamiento manual es el alejamiento de los puntos correspondientes a las arandelas, en este caso si supera cierto valor del primer momento de HU, se lo multiplica para amplificar esa diferencia.

4.4. Clasificación de imágenes

4.4.1. KNN

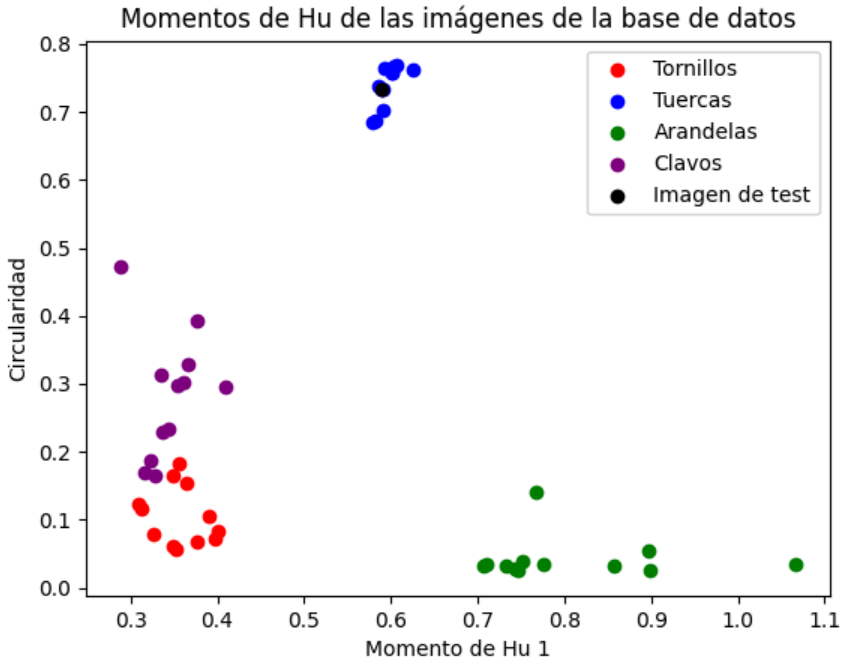
El algoritmo de clasificación KNN, es un algoritmo de aprendizaje supervisado. Esto quiere decir que le damos las etiquetas o los valores correctos a la hora de entrenar el modelo, por ejemplo cuando le pasamos los tornillos, se sabe que son tornillos.
El algoritmo toma los K vecinos más cercanos al datos que nosotros proporcionamos, es decir, compara los vectores de características de las imágenes de nuestra base de datos, con la imagen que se busca clasificar.



En este caso comparamos en las abscisas el primer momento de HU, y en las ordenadas la circularidad.
Particularmente, esta comparación se realizó mediante la siguiente ecuación:

$$D(A,B) = \sum_{i=0}^{6} \left| H_i^B - H_i^A \right|$$

La ecuación nos dice que la distancia entre las características de A y B, es la suma del valor absoluto de las diferencias de las características de la imagen A y de la imagen B. Por prueba y error, se pudo concluir que éste fue el método que mejores resultados consiguió.



Por ejemplo, en la imagen podemos ver que el punto negro representa la imagen que queremos clasificar. Si tomamos las distancias a cada clase, tenemos el siguiente resultado:

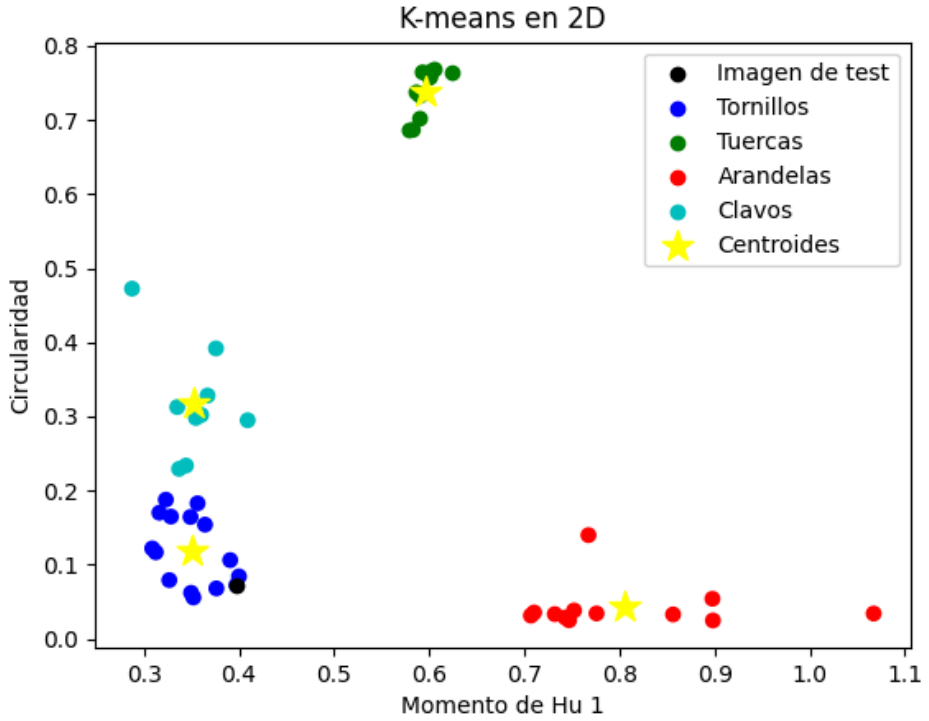
```
Clase: Tuercas, Distancia: 0.000000
Clase: Tuercas, Distancia: 0.002789
Clase: Tuercas, Distancia: 0.005765
Clase: Tuercas, Distancia: 0.033541
Clase: Tuercas, Distancia: 0.033721
[0, 0, 0, 5]

Clase predominante: Tuercas
```

Dónde podemos ver que la clase predominante es “Tuercas”, luego es esa la clase que el algoritmo elige como correcta.

4.4.2. KMeans

El algoritmo de clasificación Kmeans, es un algoritmo de aprendizaje no supervisado. Esto quiere decir que cuando pasamos los datos para entrenar el modelo, no proporcionamos información sobre la respuesta correcta, o la "etiqueta" de cada uno de esos datos.
Éste algoritmo funciona dividiendo los datos en "clústers", los cuáles son agrupaciones de los datos previstos por la base de datos. Luego, cuando introducimos una imagen a clasificar, se calcula la distancia a cada uno de los centroides de dichos clústers. Luego, se toma como valor correcto el clúster a cuyo centroide tenía menor distancia.

Si bien en la práctica el algoritmo de KMeans se utiliza con datos sin etiquetas, en nuestro caso en particular, sí tenemos las etiquetas. Por lo tanto, podemos elegir como el primer centroide elegido (que luego se actualiza) al primer dato ingresado de cada clase. Es decir, el primer tornillo ingresado será el centroide de aquellos datos, el primer clavo será el primer centroide de los clavos, y así sucesivamente.



Podemos ver en la imagen anterior que nuestra imagen de test está más cercana al centroide perteneciente a los tornillos, por lo que la clase elegida por KMeans como correcta dicha clase.

4.5. Planificación: STRIPS

El problema de planificación involucra un estado inicial y un estado objetivo deseado, con la intención de transformarlo aplicando un conjunto de acciones ordenadas. Algunas acciones tienen ciertas restricciones, las que se imponen utilizando predicados lógicos que definen la secuencia de eventos necesaria para lograr el objetivo.

STRIPS (Stanford Research Institute Problem Solver) es un generador de planes automatizado que ha sido fundamental para el desarrollo de la planificación. La implementación de la planificación en STRIPS puede hacerse utilizando el Planning Domain Definition Language (PDDL), una familia de lenguajes que permite definir problemas de planificación de manera estructurada y formal.

Para nuestro proyecto, hemos utilizado el sistema de planificación Fast Downward. Fast Downward es un sistema de planificación clásico que se basa en la búsqueda heurística, capaz de manejar problemas de planificación determinísticos generales. Este sistema emplea algoritmos de búsqueda como A* (A estrella) o Greedy Best-First Search (Búsqueda Voraz Primero el Mejor).

Acciones Definidas: El problema de planificación en nuestro proyecto se aborda considerando el apilamiento de las cajas como un sistema LIFO (Last In, First Out), utilizando funciones de Pop (Desapilar) y Push (Apilar). Las acciones definidas en el dominio son:

1. **Sacar (Pop):** Permite desapilar la caja superior y colocarla a un lado, siempre que no haya otra caja encima y no sea la caja base.
2. **Poner (Push):** Permite apilar cualquiera de las cajas desapiladas en la parte superior de la pila.
3. **Sacar-base:** Variante de "Sacar" específica para la caja de la base, que se ejecuta si la base no está vacía.
4. **Poner-base:** Variante de "Poner" específica para colocar una caja directamente en la base, sin apilarla sobre otra caja.

Definición del Problema: En el problema de planificación, las condiciones del orden inicial se establecen en la sección (init:) y las condiciones del orden final deseado se definen en la sección (goal:). Esto asegura que el sistema de planificación pueda transformar el estado inicial al estado objetivo de manera eficiente y efectiva, cumpliendo con los requisitos del proyecto.

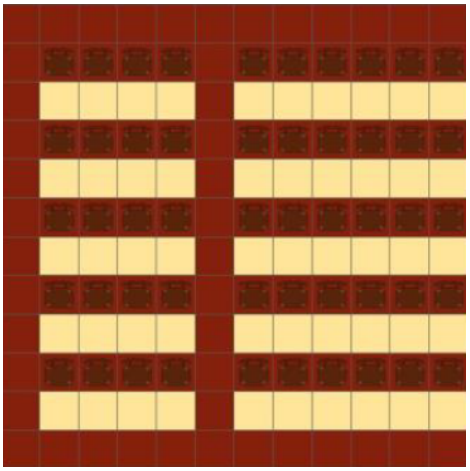
4.6. Transporte de las cajas

Para encontrar el camino más corto para transportar las cajas desde la posición inicial A a la posición final B, se utilizó el algoritmo de búsqueda A*.
Este algoritmo utiliza distintos costos para determinar el camino más corto:

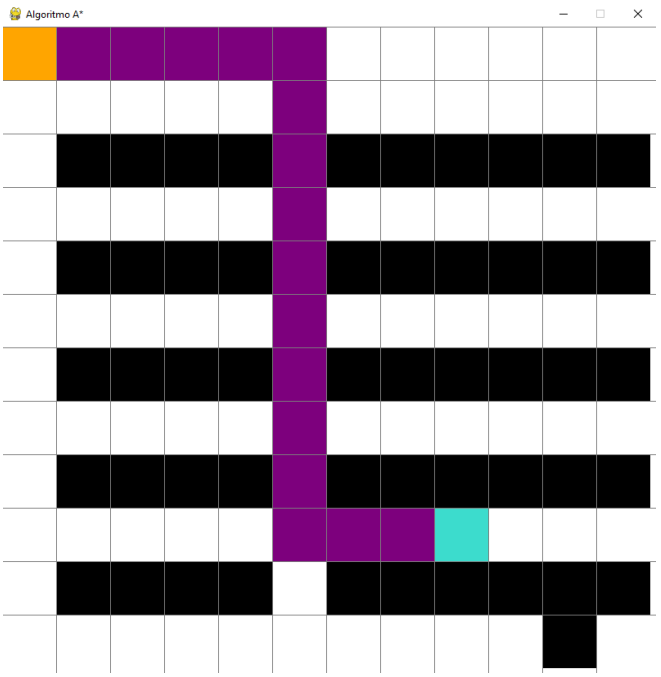
- G(n): Representa el costo acumulado de llegar desde el nodo inicial hasta el nodo actual (n). Este costo incluye la distancia recorrida o el costo acumulado de las acciones ejecutadas para alcanzar (n).
- H(n): Es una estimación del costo para llegar desde el nodo (n)hasta el nodo objetivo. Esta función heurística es crucial en A* ya que guía la búsqueda hacia la dirección correcta. En este caso particular se utiliza la distancia de Manhattan.
- F(n): Se define como la suma H(n) + G(n)

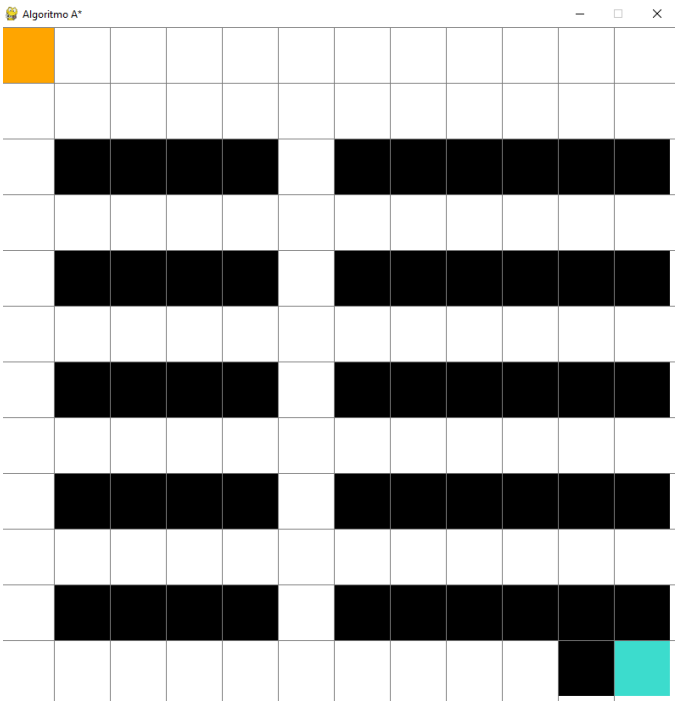
El algoritmo empieza en el punto inicial A, luego se exploran los vecinos y se elige el que tenga menor costo, es decir, el que tenga menos valor de costo F. Los vecinos que todavía no son explorados están almacenados en una lista abierta (open set) y se les asigna el estado de “open”, mientras que los vecinos que ya se evaluaron y se descartan como posibilidad, se eliminan de esta lista y se les asigna el estado de “closed”. Los vecinos son las posiciones que se encuentran arriba, abajo, a la izquierda o derecha de la posición actual, es decir, se asume que no nos podemos mover en diagonal.

El entorno en el que se trabaja representa un aula con distintas filas de bancos:



Los puntos inicial (naranja) y final (celeste) se eligen con el mouse, y se encuentra el camino más corto cuando se presiona la tecla “espacio” del teclado. Además, se agregó un bloque extra para examinar la situación en la que no haya camino posible.





No hay camino posible

5. Código del proyecto

Repositorio del código: **https://github.com/JuanStella/Reconocimiento_de_imágenes**

5.1. Leer y preprocesar imagen

```
import math
import cv2
import numpy as np
class imagen:

    def __init__(self, directorio=None):
        self.img = cv2.imread(directorio)
        self.conts = self.img.copy()
        if self.img is None:
            raise ValueError(f"Error al cargar la imagen desde la ruta: {directorio}")

        # Pasar a escala de grises la imagen obtenida
        self.img = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
        self.conts = cv2.cvtColor(self.conts, cv2.COLOR_BGR2GRAY)

        # Redimensionar la imagen a 600x600
        self.img = cv2.resize(self.img, (600, 600))
        self.conts = cv2.resize(self.conts, (600, 600))
        self.es_tornillo = False
        self.es_clavo = False

        self.momentos_hu = [0] * 7
        self.preprocesar()
        return None

    def preprocesar(self):
        self.img = contraste(self.img)
        self.img = eliminar_ruido(self.img)
        self.img = histogramas(self.img)
        self.img = binarizar(self.img)
        #Detección de contornos
        cnts,_ = cv2.findContours(self.img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        self.conts = cv2.drawContours(self.conts, cnts, -1, (0, 0, 255), 2)
```

```
        '''cv2.imshow("imagen",self.img)
        cv2.imshow("contornos",self.conts)
        cv2.waitKey(0)
        cv2.destroyAllWindows()'''

        HU = calc_momentos_HU(self.img)

        for c in cnts:
            if len(c) > 0:  # Asegurarse de que el contorno no esté vacío
                hu1, circularity = analizar_objeto(c)

                if hu1 < - 0.225:
                    HU[1] += 5
                if HU[1] > 13:
                    HU[1] -=3

                if HU[1] < 6 and circularity > 0.6:
                    circularity -= 0.4

                if HU[1] < 4.6 and circularity > 0.2:
                    circularity += 0.6

                if HU[1] < 6.5 and circularity > 0.6:
                    circularity -= 0.35

                if circularity < 0.115:
                    HU[1] = HU[1] + 0.3
                elif circularity > 0.6 and HU[1] > 6.35:
                    HU[1] = HU[1] + 2
            else:
                print("Contorno inválido")

        self.momentos_hu[0] = HU[1]/15
        self.momentos_hu[5] = circularity
        return self.img

def histogramas(img):
    img2 = cv2.equalizeHist(img)
    return img2

def contraste(img):
    img2 = cv2.convertScaleAbs(img, alpha=1, beta= 170)
    return img2
```

```
def eliminar_ruido(img):
    img = cv2.GaussianBlur(img, (3,3), 0)
    return img

def binarizar(img):
    _, img2 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    # Invertir la imagen binarizada para resaltar el tornillo en blanco y el fondo en negro
    img2 = cv2.bitwise_not(img2)
    return img2


def calc_momentos_HU(im):
    # Calcular Momentos
    moments = cv2.moments(im)
    # Calcular Hu Moments
    huMoments = cv2.HuMoments(moments)
    # Convertir los valores de huMoments a escalares antes de realizar las operaciones matemáticas
    huMoments = [moment[0] for moment in huMoments]
    for i in range(0, 7):
        huMoments[i] = -1 * math.copysign(1.0, huMoments[i]) * math.log10(abs(huMoments[i]))
    return huMoments


def analizar_objeto(contorno):
    if contorno is None or len(contorno) == 0:
        return 0, 0, 0  # Retorna valores por defecto si el contorno es inválido

    # Calcular Momentos de Hu
    moments = cv2.moments(contorno)
    huMoments = cv2.HuMoments(moments)

    # Convertir a escala logarítmica
    for i in range(7):
        if huMoments[i] != 0:
                                    huMoments[i] = -1 * math.copysign(1.0, huMoments[i]) *
math.log10(abs(huMoments[i]))

    hu1 = huMoments[0][0]  # Primer momento de Hu

    area = cv2.contourArea(contorno)
    perimeter = cv2.arcLength(contorno, True)

    circularity = 4 * math.pi * area / (perimeter * perimeter) if perimeter > 0 else 0

    return hu1, circularity
```

5.2. KNN

```
import BD
import leer_img
import matplotlib.pyplot as plt
class Knn :

    def __init__(self, k):
        self.k = k
        self.bd = BD.BD()
        self.bd.cargar_imagenes()
        self.bd.guardar_momentos_en_archivo('momentos_hu.txt')

    #Suma de las diferencias absolutas de las características de las imágenes
    def distancia_puntos(self, momentos_hu):

        distancias = []

        for tornillo in self.bd.tornillos:
            dist = abs(tornillo[0] - momentos_hu[0]) + abs(tornillo[5] - momentos_hu[5])
            distancias.append((dist, "Tornillos"))

        for tuerca in self.bd.tuercas:
            dist = abs(tuerca[0] - momentos_hu[0]) + abs(tuerca[5] - momentos_hu[5])
            distancias.append((dist, "Tuercas"))

        for arandela in self.bd.arandelas:
            dist = abs(arandela[0] - momentos_hu[0]) + abs(arandela[5] - momentos_hu[5])
            distancias.append((dist, "Arandelas"))

        for clavo in self.bd.clavos:
            dist = abs(clavo[0] - momentos_hu[0]) + abs(clavo[5] - momentos_hu[5])
            distancias.append((dist, "Clavos"))


        distancias.sort(key=lambda x: x[0])  # Ordenar por distancia ascendente

        return distancias
```

```
  def graficar_2D(self, direc_prueba):
    for i in range(12):
        plt.scatter(self.bd.tornillos[i][0], self.bd.tornillos[i][5], color='red', label='Tornillos' if i == 0 else "")
        plt.scatter(self.bd.tuercas[i][0], self.bd.tuercas[i][5], color='blue', label='Tuercas' if i == 0 else "")
        plt.scatter(self.bd.arandelas[i][0], self.bd.arandelas[i][5], color='green', label='Arandelas' if i == 0 else "")
        plt.scatter(self.bd.clavos[i][0], self.bd.clavos[i][5], color='purple', label='Clavos' if i == 0 else "")

    img1 = leer_img.imagen(direc_prueba)
      plt.scatter(img1.momentos_hu[0], img1.momentos_hu[5], color='black', label='Imagen de test')

    plt.xlabel('Momento de Hu 1')
    plt.ylabel('Circularidad')
    plt.title('Momentos de Hu de las imágenes de la base de datos')

    plt.legend()
    plt.show()
    return None



  def distancia_a_cada_imagen(self):
    img = leer_img.imagen(direc_prueba)
    distancias = self.distancia_puntos(img.momentos_hu)

    with open('momentos_hu.txt', 'a') as f:
      f.write("\n\nImagen test\n")
      for i, momento in enumerate(img.momentos_hu, 1):
        f.write(f"Momento de HU {i}: {momento}\n")

    # Crear una lista de tuplas (distancia, clase)
    lista_distancias_clases = [(dist, clase) for dist, clase in distancias]

    # Ordenar la lista de tuplas por la distancia
    lista_distancias_clases.sort()

    # Obtener las primeras 5 distancias
    primeras_5 = lista_distancias_clases[:5]
```

```
    # Contar las ocurrencias de cada clase en las primeras 5 distancias
    conteo_clases = {"Tornillos": 0, "Tuercas": 0, "Arandelas": 0, "Clavos": 0}
    cont = 0
    for dist, clase in primeras_5:

        conteo_clases[clase] += 1


    # Determinar la clase predominante
    clase_predominante = max(conteo_clases, key=conteo_clases.get)
    valores = list(conteo_clases.values())
    print("Primeras 5 distancias ordenadas por clase:")
    for dist, clase in primeras_5:
        cont +=1
        print(f"Clase: {clase}, Distancia: {dist:.6f}")
        if cont == 1:
            clasee = clase

    valores.sort()
    print(valores)
    if valores[3] == valores[2]:
        clase_predominante = clasee

    print(f"\nClase predominante: {clase_predominante}")


    return clase_predominante


direc_prueba = 'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\Test\\tu5.jpeg'


def main():
    knn = Knn(5)
    knn.distancia_a_cada_imagen()
    knn.graficar_2D(direc_prueba)

if __name__ == '__main__':
    main()
```

5.3. KMeans

```
# Kmeans.py
import BD as bd
import numpy as np
import matplotlib.pyplot as plt
import leer_img

class Kmeans:
  def __init__(self, k=4):
    self.k = k
    self.centroides = None
    self.bd = bd.BD()
    self.bd.cargar_imágenes()
    self.bd.guardar_momentos_en_archivo('momentos_hu.txt')
      self.momentos_Hu_np = np.array(self.bd.momentos_Hu)  # Convertir momentos_Hu a una matriz numpy

  def colocar(self, X, etiquetas, it=100):
    # Inicializar los centroides eligiendo un punto de cada clase
    clases = np.unique(etiquetas)
    if len(clases) < self.k:
      raise ValueError(f'Número de clases ({len(clases)}) es menor que k ({self.k}).')

    self.centroides = []
    self.cluster_names = []  # Lista para almacenar los nombres de los clusters

    for clase in clases[:self.k]:
      indices = np.where(etiquetas == clase)[0]
      if len(indices) > 0:
        self.centroides.append(X[indices[0]])  # Elige el primer punto de cada clase
        self.cluster_names.append(self.bd.etiquetas[clase])  # Asignar el nombre del cluster
    self.centroides = np.array(self.centroides)

    for _ in range(it):
      y = []

      for data_point in X:
        distancias = Kmeans.distancia_euclediana(data_point, self.centroides)
```

```
            numero_cluster = np.argmin(distancias)


        y.append(numero_cluster)
        y = np.array(y)

        indice_de_cluster = []
        for i in range(self.k):
            indice_de_cluster.append(np.argwhere(y == i))  # Obtener los indices de los clusters

        centro_clusters = []


        for i, indice in enumerate(indice_de_cluster):
            if len(indice) == 0:
                centro_clusters.append(self.centroides[i])
            else:
                centro_clusters.append(np.mean(X[indice], axis=0)[0])

        if np.max(self.centroides - np.array(centro_clusters)) < 1e-3:
            break

        else:
            self.centroides = np.array(centro_clusters)
    return y

@staticmethod
def distancia_eucledіana(datos, centroides):
    distancias = np.sqrt(np.sum((centroides - datos) ** 2, axis=1))
    return distancias

def predecir(self, nuevo_dato):
    distancias = Kmeans.distancia_eucledіana(nuevo_dato, self.centroides)
    return np.argmin(distancias)


def main():
    km = Kmeans(4)

    # Generar etiquetas para los datos
    etiquetas = np.array([0] * len(km.bd.tornillos) +
                         [1] * len(km.bd.tuercas) +
                         [2] * len(km.bd.arandelas) +
                         [3] * len(km.bd.clavos))
```

```
    etiquetas_clusters = km.colocar(km.momentos_Hu_np, etiquetas)

    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Asignar colores a cada clúster
    colores = ['b', 'g', 'r', 'c']  # Definir colores para cada cluster (modifícalos según tus necesidades)
    colores_clusters = [colores[etiqueta] for etiqueta in etiquetas_clusters]

    scatter= ax.scatter(km.momentos_Hu_np[:, 0], km.momentos_Hu_np[:, 5], c=colores_clusters)

    ax.set_xlabel('Momento de Hu 1')
    ax.set_ylabel('Circularidad')
    ax.set_title('K-means en 2D')

    # Predecir el cluster al que pertenece el nuevo dato
direc_prueba=
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imágenes\\imágenes\\Test\\tortest1.jpeg'
    img1 = leer_img.imagen(direc_prueba)
        ax.scatter(img1.momentos_hu[0], img1.momentos_hu[5], color='black', label='Imagen de
test')

    nuevo_dato = img1.momentos_hu
    cluster_asignado = km.predecir(nuevo_dato)
    print(f'El nuevo dato pertenece al clúster: {km.cluster_names[cluster_asignado]}')

    # Agregar leyenda para los clusters
    etiquetas_unicas = np.unique(etiquetas_clusters)
    for i, etiqueta in enumerate(etiquetas_unicas):
        ax.scatter([], [], c=colores[i], label=f'{km.cluster_names[etiqueta]}')

    # Graficar los centroides finales
        ax.scatter(km.centroides[:, 0], km.centroides[:, 5], color='yellow', marker='*', s=200,
label='Centroides')

    ax.legend()
    plt.show()
    return None

if __name__ == '__main__':
    main()
```

5.4. Base de datos

```
from leer_img import imagen
class BD:
  def __init__(self):
    self.tornillos = []
    self.tuercas = []
    self.arandelas = []
    self.clavos = []
    self.momentos_Hu = []
    self.etiquetas = ['Tornillos', 'Tuercas', 'Arandelas', 'Clavos']

    self.direc_tornillos = [
 'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\1.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\2.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\3.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\4.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\5.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\6.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\7.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\8.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\9.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\10.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\11.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tornillos\\12.jpeg',
    ]

    self.direc_tuercas = [
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer1.jpeg',

'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer2.jpeg',
```

```
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer3.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer4.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer5.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer6.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer7.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer8.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer9.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer10.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer11.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\tuercas\\tuer12.jpeg',
    ]

    self.direc_arandelas = [
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a1.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a2.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a3.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a4.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a5.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a6.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a7.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a8.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a9.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a10.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a11.jpeg',
'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\arandelas\\a12.jpeg',
```

```
    ]


    self.direc_clavos = [
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c1.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c2.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c3.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c4.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c5.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c6.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c7.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c8.jpeg',
        'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c9.jpeg',
       'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c10.jpeg',
       'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c11.jpeg',
       'D:\\Facu\\IA1\\Proyecto\\Reconocimiento_de_imagenes\\Imagenes\\clavos\\c12.jpeg',
    ]

def cargar_imagenes(self):
    for i in range(12):
        try:
            img = imagen(self.direc_tornillos[i])
            self.tornillos.append(img.momentos_hu)
        except ValueError as e:
            print(e)

    for i in range(12):
        try:
            img = imagen(self.direc_tuercas[i])
            self.tuercas.append(img.momentos_hu)
        except ValueError as e:
            print(e)

    for i in range(12):
        try:
            img = imagen(self.direc_arandelas[i])
            self.arandelas.append(img.momentos_hu)
        except ValueError as e:
            print(e)

    for i in range(12):
        try:
            img = imagen(self.direc_clavos[i])
            self.clavos.append(img.momentos_hu)
        except ValueError as e:
            print(e)
```

```
        self.momentos_Hu = self.tornillos + self.tuercas + self.arandelas + self.clavos
        return None


    def guardar_momentos_en_archivo(self, archivo):
        with open(archivo, 'w') as f:
            f.write("Momentos de Hu de las imagenes de los tornillos:\n\n")
            for i, momentos in enumerate(self.tornillos):
                f.write(f"Imagen {i+1}:\n")
                for j, momento in enumerate(momentos):
                    f.write(f"  Hu[{j+1}]: {momento:.6f}\n")
                f.write("\n")

            f.write("Momentos de Hu de las imagenes de las tuercas:\n\n")
            for i, momentos in enumerate(self.tuercas):
                f.write(f"Imagen {i+1}:\n")
                for j, momento in enumerate(momentos):
                    f.write(f"  Hu[{j+1}]: {momento:.6f}\n")
                f.write("\n")

            f.write("Momentos de Hu de las imagenes de las arandelas:\n\n")
            for i, momentos in enumerate(self.arandelas):
                f.write(f"Imagen {i+1}:\n")
                for j, momento in enumerate(momentos):
                    f.write(f"  Hu[{j+1}]: {momento:.6f}\n")
                f.write("\n")

            f.write("Momentos de Hu de las imagenes de los clavos:\n\n")
            for i, momentos in enumerate(self.clavos):
                f.write(f"Imagen {i+1}:\n")
                for j, momento in enumerate(momentos):
                    f.write(f"  Hu[{j+1}]: {momento:.6f}\n")
                f.write("\n")
        return None
```

5.5. A*

```
import sys
import pygame
from queue import PriorityQueue

WIDTH = 800
```

```
WIN = pygame.display.set_mode((WIDTH, WIDTH))
pygame.display.set_caption("Algoritmo A*")

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
PURPLE = (128, 0, 128)
ORANGE = (255, 165 ,0)
GREY = (128, 128, 128)
TURQUOISE = (64, 224, 208)

class Spot:
    def __init__(self, row, col, width, total_rows):
        self.row = row
        self.col = col
        self.x = row * width
        self.y = col * width
        self.color = WHITE
        self.neighbors = []
        self.width = width
        self.total_rows = total_rows

    def get_pos(self):
        return self.row, self.col

    def is_barrier(self):
        return self.color == BLACK

    def is_start(self):
        return self.color == ORANGE

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

    def make_start(self):
        self.color = ORANGE

    def make_barrier(self):
        self.color = BLACK

    def make_end(self):
        self.color = TURQUOISE

    def make_path(self):
```

```
            self.color = PURPLE

        def draw(self, win):
            pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.width))

        def update_neighbors(self, grid):
            self.neighbors = []
            if self.row < self.total_rows - 1 and not grid[self.row + 1][self.col].is_barrier():
#DOWN
                self.neighbors.append(grid[self.row + 1][self.col])

            if self.row > 0 and not grid[self.row - 1][self.col].is_barrier(): # UP
                self.neighbors.append(grid[self.row - 1][self.col])

            if self.col < self.total_rows - 1 and not grid[self.row][self.col + 1].is_barrier():
#RIGHT
                self.neighbors.append(grid[self.row][self.col + 1])

            if self.col > 0 and not grid[self.row][self.col - 1].is_barrier(): # LEFT
                self.neighbors.append(grid[self.row][self.col - 1])


def h(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return abs(x1 - x2) + abs(y1 - y2)

def reconstruct_path(came_from, current, draw):
    while current in came_from:
        current = came_from[current]
        current.make_path()
        draw()

# Definir la función para mostrar la ventana de alerta
def mostrar_alerta():
  pygame.init()
  pantalla = pygame.display.set_mode((400, 200))
  pygame.display.set_caption("Alerta")
  fuente = pygame.font.SysFont(None, 48)
  texto = fuente.render("No hay camino posible", True, (255, 0, 0))
  rect_texto = texto.get_rect(center=(200, 100))
  corriendo = True
```

```
    while corriendo:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                corriendo = False

        pantalla.fill((255, 255, 255))
        pantalla.blit(texto, rect_texto)
        pygame.display.flip()

    pygame.quit()
    sys.exit()


def algorithm(draw, grid, start, end):
    count = 0
    open_set = PriorityQueue()
    open_set.put((0, count, start))
    came_from = {}
    g_score = {spot: float("inf") for row in grid for spot in row}
    g_score[start] = 0
    f_score = {spot: float("inf") for row in grid for spot in row}
    f_score[start] = h(start.get_pos(), end.get_pos())

    open_set_hash = {start}

    while not open_set.empty():
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()

        current = open_set.get()[2]
        open_set_hash.remove(current)


        if current == end:
            reconstruct_path(came_from, end, draw)
            end.make_end()
            start.make_start()
            return True

        for neighbor in current.neighbors:
            temp_g_score = g_score[current] + 1
```

```
            if temp_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = temp_g_score
                f_score[neighbor] = temp_g_score + h(neighbor.get_pos(), end.get_pos())
                if neighbor not in open_set_hash:
                    count += 1
                    open_set.put((f_score[neighbor], count, neighbor))
                    open_set_hash.add(neighbor)

        draw()

        if current != start:
            pass

    mostrar_alerta()
    return False


def make_grid(rows, width):
    grid = []
    gap = width // rows
    for i in range(rows):
        grid.append([])
        for j in range(rows):
            spot = Spot(i, j, gap, rows)
            grid[i].append(spot)
    return grid


def draw_grid(win, rows, width):
    gap = width // rows
    for i in range(rows):
        pygame.draw.line(win, GREY, (0, i * gap), (width, i * gap))
        for j in range(rows):
            pygame.draw.line(win, GREY, (j * gap, 0), (j * gap, width))
    return None


def draw(win, grid, rows, width):
    win.fill(WHITE)
    for row in grid:
```

```
                for spot in row:
                        spot.draw(win)


        draw_grid(win, rows, width)
        pygame.display.update()
        return None

def get_clicked_pos(pos, rows, width):
    gap = width // rows
    y, x = pos
    row = y // gap
    col = x // gap
    return row, col

def main(win, width):
        ROWS = 12
        grid = make_grid(ROWS, width)
        run = True
        start = None
        end = None

        while run:
                draw(win, grid, ROWS, width)
                for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                                run = False

                        grid[10][11].make_barrier()


                        for i in range(12):
                                for j in range(12):
                                        if i % 2 == 0 and j != 0 and j != 5 and i !=0:  # Ajuste en
la generación del laberinto
                                                grid[j][i].make_barrier()

                        if pygame.mouse.get_pressed()[0]: # LEFT
                                pos = pygame.mouse.get_pos()
                                row, col = get_clicked_pos(pos, ROWS, width)
                                spot = grid[row][col]
                                if not start and spot != end and not spot.is_barrier():
```

```
                    start = spot
                    start.make_start()

                elif not end and spot != start and not spot.is_barrier():
                    end = spot
                    end.make_end()

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE and start and end:
                    for row in grid:
                        for spot in row:
                            spot.update_neighbors(grid)
                    algorithm(lambda: draw(win, grid, ROWS, width), grid,
start, end)

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

    pygame.quit()

if __name__ == "__main__":
    main(WIN, WIDTH)
```

5.6. STRIPS

```
#(Domain)
(define (domain reorganizar-cajas)
 (:requirements :negative-preconditions :strips)
 (:predicates
  (caja ?x)
  (sobre ?x ?y)
  (mesa ?x)
  (libre ?x)
  (base ?x)
  (posicion ?x)
  (despejado ?x)
```

```
(:action mover
 :parameters (?caja1 ?caja2)
 :precondition (and (caja ?caja1)
                    (caja ?caja2)
                    (sobre ?caja1 ?caja2)
                    (libre ?caja1)
                    (not (base ?caja1)))
 :effect (and (not (sobre ?caja1 ?caja2))
              (mesa ?caja1)
              (libre ?caja2)))

(:action quitar-ultimo
 :parameters (?caja ?base)
 :precondition (and (caja ?caja)
                    (posicion ?base)
                    (base ?caja)
                    (not (despejado ?base))
                    (libre ?caja))
 :effect (and (mesa ?caja)
              (not (base ?caja))
              (despejado ?base)))

(:action juntar
 :parameters (?caja1 ?caja2)
 :precondition (and
                (caja ?caja1)
                (caja ?caja2)
                (mesa ?caja1)

                (libre ?caja2))
 :effect (and (sobre ?caja1 ?caja2)
              (not (mesa ?caja1))
              (not (libre ?caja2))))

(:action poner-ultimo
 :parameters (?caja ?base)
 :precondition (and
                (caja ?caja)
                (posicion ?base)
                (mesa ?caja)
```

```
            (libre ?caja)
            (despejado ?base))


   :effect (and (not (mesa ?caja))
            (base ?caja)
            (not (despejado ?base))))
)
```

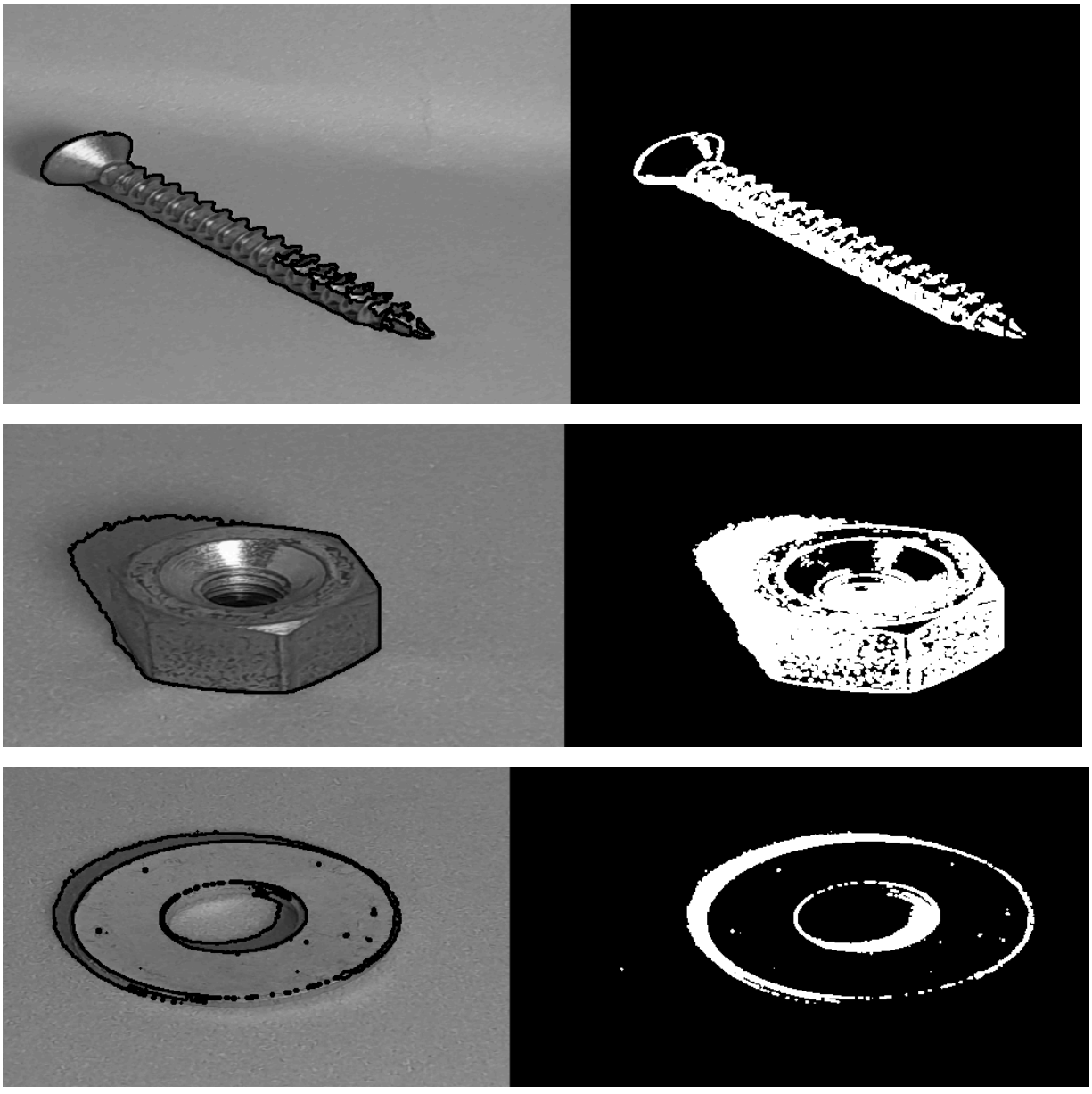
#(Problem)

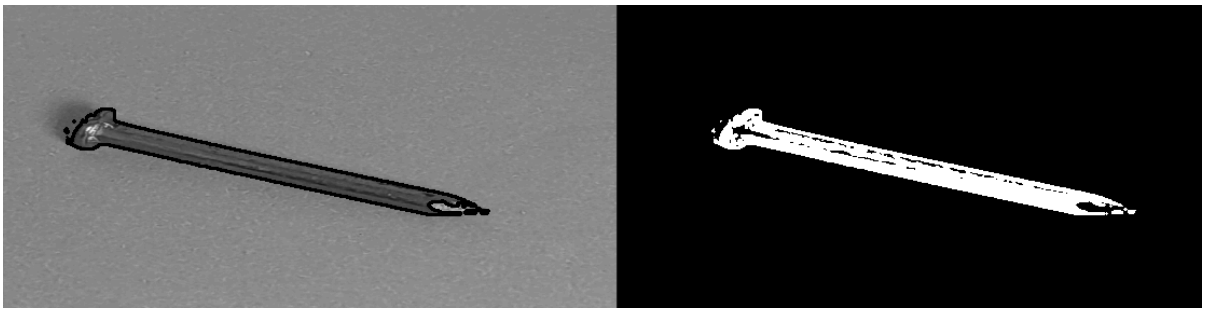
```
(define (problem reordenar-cajas)
 (:domain reorganizar-cajas)
 (:objects arandelas tuercas clavos tornillos piso)

 (:init
  (caja arandelas)
  (caja tuercas)
  (caja clavos)
  (caja tornillos)
  (posicion piso)
  (sobre tuercas tornillos)
  (sobre tornillos clavos)
  (sobre clavos arandelas)
  (base arandelas)
  (libre tuercas)
  (not (despejado piso)))

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

6. Resultados

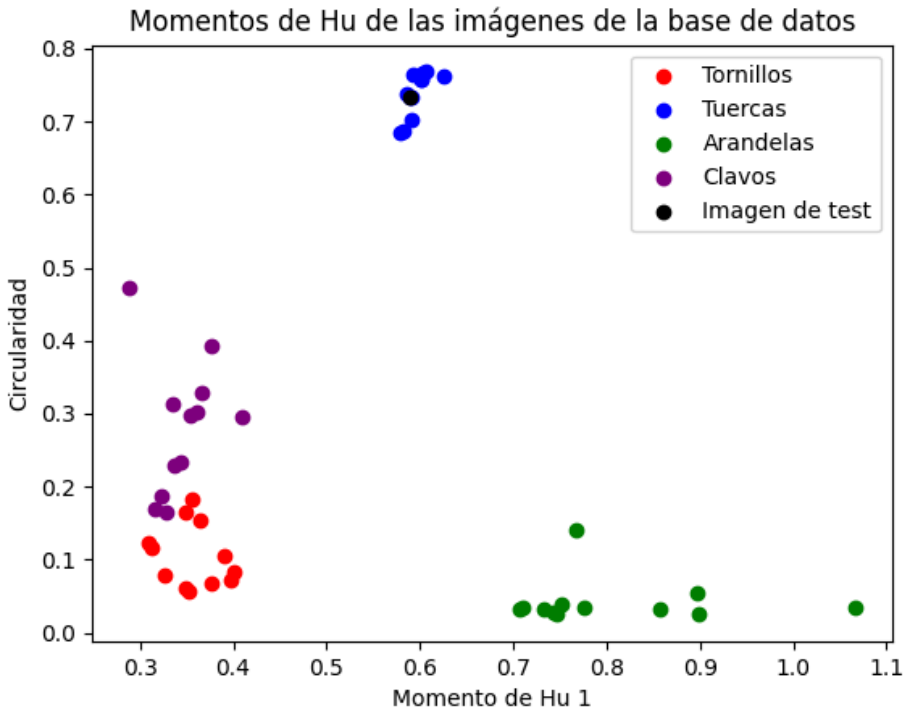
6.1. Muestra de funcionamiento

6.1.1. Leer y preprocesar imagen

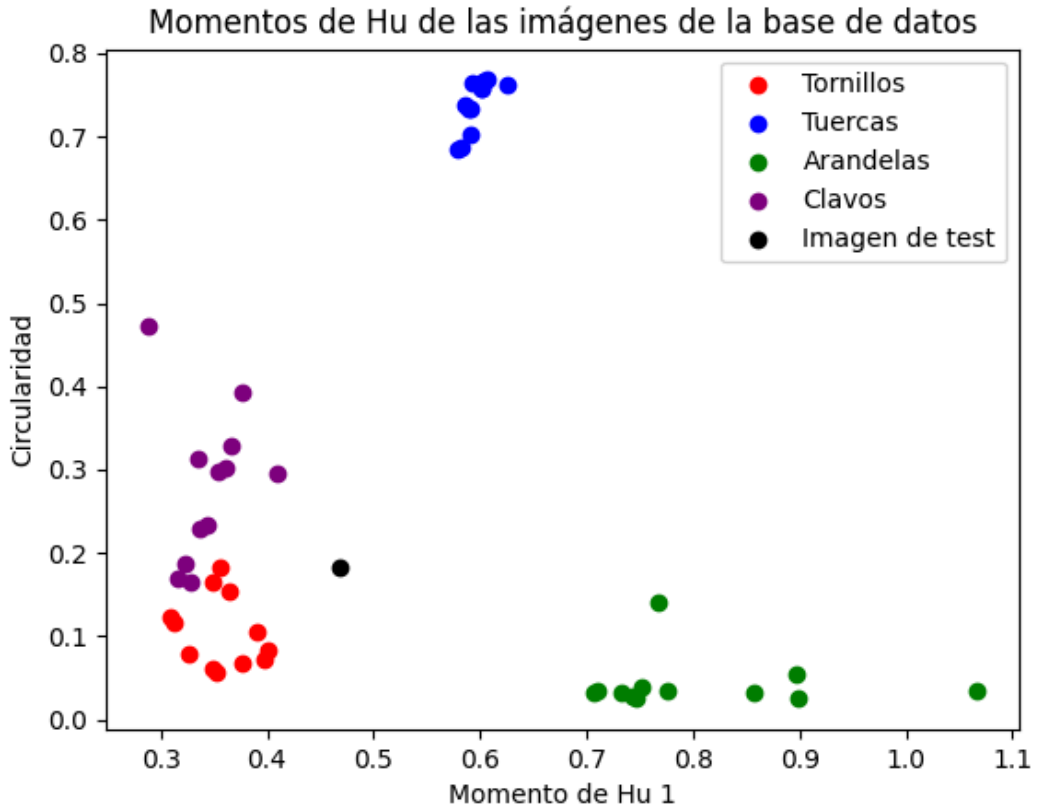




6.1.2. KNN

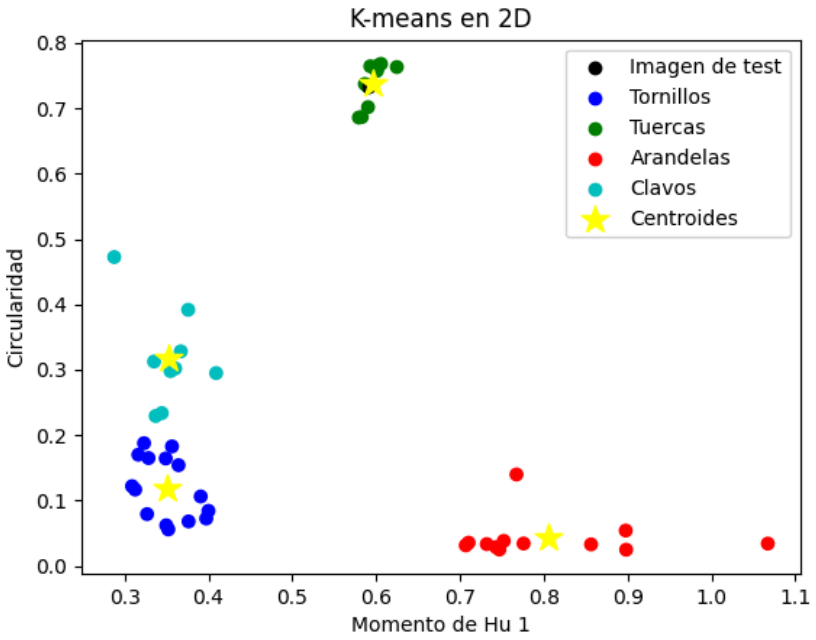


Clase: Tuercas, Distancia: **0.000000**
Clase: Tuercas, Distancia: **0.002789**
Clase: Tuercas, Distancia: **0.005765**
Clase: Tuercas, Distancia: **0.033541**
Clase: Tuercas, Distancia: **0.033721**
[0, 0, 0, 5]
Clase predominante: Tuercas

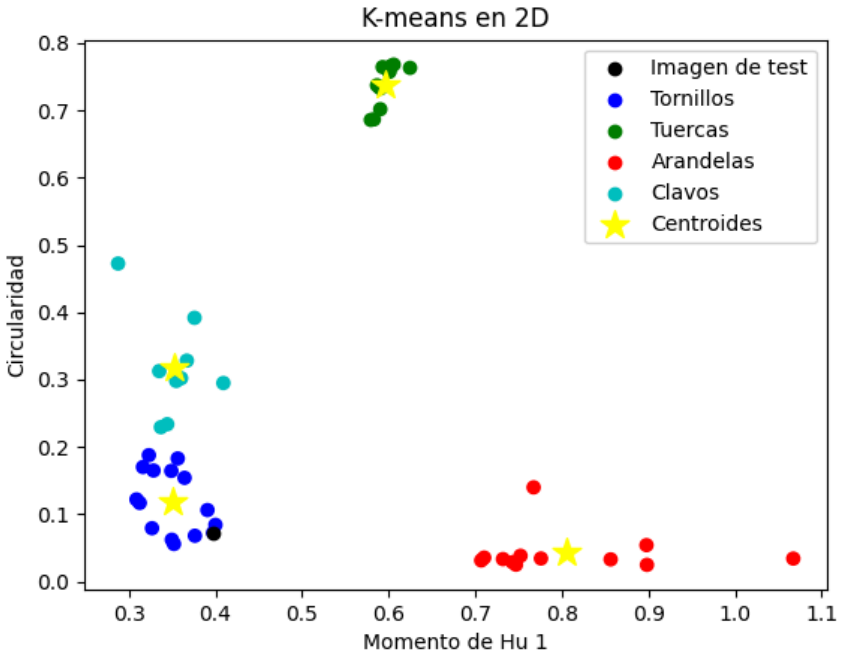


Clase: Tornillos, Distancia: **0.112535**
Clase: Tornillos, Distancia: **0.132224**
Clase: Tornillos, Distancia: **0.137149**
Clase: Clavos, Distancia: **0.150761**
Clase: Tornillos, Distancia: **0.153626**
[0, 0, 1, 4]
Clase predominante: Tornillos

6.1.3. KMeans

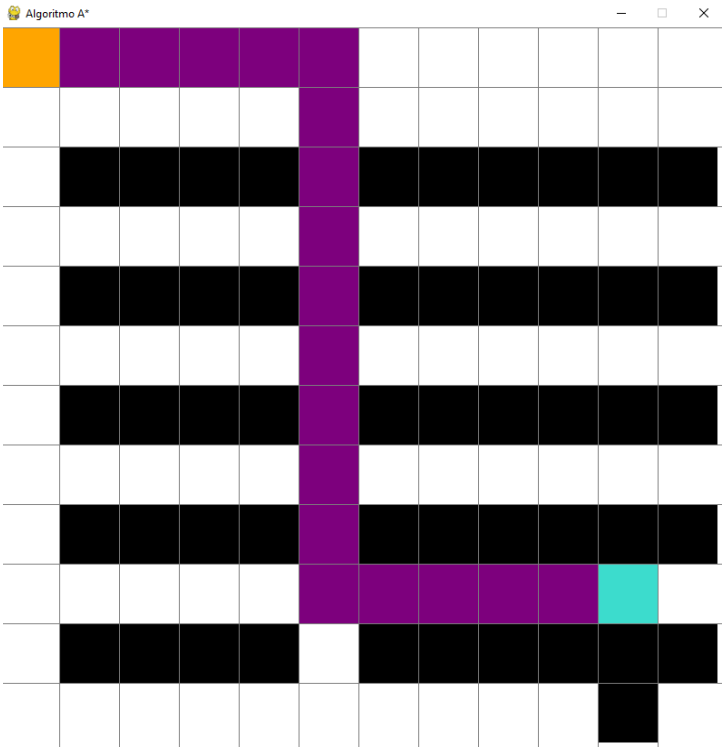


El nuevo dato pertenece al clúster: Tuercas



El nuevo dato pertenece al clúster: Tornillos

6.1.4. A*



6.1.5. STRIPS

Plan

```
(mover tuercas tornillos)
(mover tornillos clavos)
(mover clavos arandelas)
(quitar-ultimo arandelas piso)
(poner-ultimo clavos piso)
(juntar tuercas clavos)
(juntar arandelas tornillos)
(juntar tornillos tuercas)
; cost = 8 (unit cost)
```

6.1.6. Archivo externo de Base de datos

Momentos de Hu de las imágenes de los tornillos**:**

Imagen **1:**
Hu**[1]: 0.376274**
Hu**[2]: 0.000000**
Hu**[3]: 0.000000**
Hu**[4]: 0.000000**
Hu**[5]: 0.000000**
Hu**[6]: 0.068086**
Hu**[7]: 0.000000**

Momentos de Hu de las imágenes de las tuercas:

Imagen 1:
Hu[1]: 0.593058
Hu[2]: 0.000000
Hu[3]: 0.000000
Hu[4]: 0.000000
Hu[5]: 0.000000
Hu[6]: 0.764167
Hu[7]: 0.000000

6.1.7. Precisión y otros resultados

a) Procesado de imágenes:
Para el procesado de imágenes podemos encontrar que, si la iluminación es correcta y no hay errores en la toma de las mismas, se pueden distinguir correctamente las distintas partes y detalles de cada una de las piezas. Además como se aclaró anteriormente, la toma de imágenes se realizó en un aula de la facultad.

b) KNN:
   i) Tornillos : Para los tornillos se encontró una precisión del 100%
   ii) Tuercas: Para las tuercas encontramos una precisión del 100%
   iii) Arandelas: Para las arandelas encontramos una precisión del 100%
   iv) Clavos: Para los clavos encontramos una precisión del 100%

c) KMeans:
   i) Tornillos : Para los tornillos se encontró una precisión del 100%
   ii) Tuercas: Para las tuercas encontramos una precisión del 100%
   iii) Arandelas: Para las arandelas encontramos una precisión del 100%
   iv) Clavos: Para los clavos encontramos una precisión del 100%

d) A*:
   El algoritmo encuentra correctamente el camino más corto a la posición deseada. Además, en el caso de no haber un camino posible, se muestra un mensaje en una ventana emergente que nos indica dicho error.

e) STRIPS:
   El algoritmo de STRIPS, devuelve correctamente la secuencia de acciones a realizar por el robot para encontrar el orden necesario. Además, lo resuelve en la menor cantidad de pasos o acciones posibles.

7. Conclusiones

Del análisis hecho luego de las pruebas de cada uno de los algoritmos, puedo deducir que es posible hacer las siguientes mejoras:

- Separar mejor los datos de tornillos y clavos
- Agrandar la base de datos
- Posibilidad de hacerlo en tiempo real
- Posibilidad de ejecutar todo en secuencia

Dichas mejoras se refieren a la clasificación de las imágenes, debido a que los algoritmos de búsqueda y planificación funcionan de acuerdo a lo deseado.

Igualmente, con las técnicas empleadas se obtuvo una muy buena precisión, incluso con una base de datos relativamente pequeña y utilizando un vector de características con solo dos componentes. Por lo tanto, tiene la ventaja de ser una implementación mayormente simple al problema.

Respecto a los algoritmos de clasificación, es importante también aclarar que KMeans normalmente no sería implementado con la "mejora" con la que se implementó en este caso, por lo tanto, sería correcto usar KNN para problemas de este tipo.

Como observaciones extra puedo agregar en hacer énfasis en la calidad y entorno en la que se captan las imágenes, ya que esto puede ser un factor muy grande en el éxito a la hora de clasificar las mismas. Es decir, es recomendable realizar un trabajo extensivo a la hora de la toma y procesamiento de las imágenes, para captar correctamente las características.

Por último, recomiendo también ampliamente el uso del lenguaje Python para aplicaciones del estilo, ya que simplifica mucho la programación orientada a objetos y permite una prolijidad mayor en el código e implementación. Además, tenemos librerías muy útiles y rápidas para su uso, como OpenCv, Matplotlib, Numpy, etc.

8. Referencias

Momentos de HU:
https://learnopencv.com/shape-matching-using-hu-moments-c-python/

Procesamiento de imágenes:
TRATAMIENTO DE IMAGENES CON cv2 EN PYTHON

KNN:
👉🔴 KNN ALGORITMO en Español (K VECINOS MÁS CERCANOS) de CLASIFICACIÓN S…

Detección de figuras geométricas:
https://omes-va.com/detectando-figuras-geometricas-con-opencv-python/

KMeans:
K-Means Clustering From Scratch in Python (Mathematical)

STRIPS:
https://lcas.lincoln.ac.uk/fast-downward/