

Trabajo Final Inteligencia Artificial I

# Visión Artificial

Año 2021

---



Ignacio Berridy, 11987

**Prof.: Dr. Ing. Selva Rivera**

**Universidad Nacional de Cuyo - Facultad de Ingeniería**

---

---

# Índice

<b>Índice</b>	<b>2</b>
<b>Resumen</b>	<b>3</b>
<b>Introducción</b>	<b>4</b>
<b>Especificación del agente</b>	<b>6</b>
Diseño del agente	7
<b>Código</b>	<b>10</b>
<b>Ejemplos de aplicación</b>	<b>22</b>
<b>Resultados</b>	<b>24</b>
<b>Conclusiones</b>	<b>29</b>
<b>Bibliografía</b>	<b>30</b>

---

## Resumen

Se ha desarrollado un sistema de clasificación de piezas metálicas, en concreto, arandelas, tuercas, tornillos y clavos, mediante visión artificial. Para esto se utilizó el lenguaje de programación python con el paradigma de programación orientada a objetos, principalmente la librería opencv para el tratamiento de imágenes, en concreto se aplicaron los filtros, gaussiano, canny y de binarización. Además se desarrollaron los algoritmos k means y knn para la clasificación de las imágenes.

Mediante distintos ensayos se ajustaron los parámetros para el tratamiento de las imágenes, así como también los datos más relevantes de cada imagen para ejecutar los algoritmos de clasificación.

Se concluyó que para este tipo de imágenes lo mejor es tomarlas con una misma proporción, en este caso 1:1, además de lograr un entorno libre de sombras para lo que se utiliza una caja cerrada con fondo blanco. A su vez se determinó mediante un análisis gráfico ,empírico y de investigación que los datos más relevantes, en este caso, eran el momento de hu2, la relación alto/ancho, y la redondez del objeto dada por  $\frac{4\pi \cdot \text{área} \cdot \text{número\_de\_esquinas}}{\text{perímetro}}$ .



---

## Introducción

Entendemos percepción como la manera en la que el cerebro de un organismo interpreta los estímulos sensoriales que recibe a través de los sentidos para formar una impresión consciente de la realidad física de su entorno. Para el caso de la visión es lo que percibimos a través de los ojos como ondas de luz, que gracias al procesamiento de estas ondas por nuestro cerebro, distinguimos las formas, tamaños y colores de lo que nos rodea, además mediante la experiencia previa, podemos diferenciar y clasificar de distintas maneras cada objeto que percibimos con la vista.

La visión artificial busca imitar esta percepción mediante el uso de computadoras y sensores como lo pueden ser cámaras, esto es, no solo capturar y renderizar las ondas de luz como imágenes, sino también poder extraer información de las mismas y clasificarlas, medirlas, etc. Dependiendo del caso se busca lograr un mayor acercamiento a lo que puede lograr el cerebro humano mediante el uso de la visión.

Podemos encontrar aplicaciones de la visión artificial en distintos campos, como por ejemplo la industria automotriz, en especial con las nuevas tecnologías de autos que se conducen de manera automática, requieren de la visión artificial, para percibir con la mayor precisión posible los objetos alrededor, como otros vehículos, personas, animales, señales. También es común su uso en medicina, por ejemplo para, que en conjunto con otros elementos como brazos robóticos, un médico especializado pueda realizar cirugías a distancia, siendo la visión artificial la encargada de procesar imágenes de alta calidad y detectar objetos.

Para este caso se buscaba clasificar y medir distintas imágenes de objetos metálicos, arandelas, tuercas, clavos y tornillos. Para ello se tomaron fotos de los elementos mencionados, que mediante ensayo empírico se determinó una manera óptima de tomar las fotos.

Una vez reunidas las imágenes se implementó un tratamiento a las mismas, que tenía como objetivo eliminar ruidos de las mismas y detectar solo los bordes de los objetos analizados, para extraer la información necesaria de los mismos.

Finalmente se realizaron dos algoritmos para la clasificación de las imágenes.

---

En primer lugar la clasificación se obtuvo mediante el algoritmo k means, este utiliza las coordenadas (altura/ancho, redondez, hu2) de cada imagen para calcular distancias, mediante pitágoras, a 4 centroides, que se inicializan tomando las coordenadas de 4 imágenes distintas del conjunto. Luego se las divide en 4 grupos, perteneciendo cada imagen al grupo de imágenes que el mismo centroide como más cercano. Una vez hecho esto se toma la media de cada coordenada de las imágenes de cada grupo para obtener 4 nuevos centroides y se rearranjan los grupos respecto a estos centroides. Esto se realiza repetidas veces, obteniendo al final 4 grupos bien definidos, que representan cada uno un objeto de los que se busca clasificar.

También se logró clasificar a los objetos con el uso del algoritmo Knn. Se hace uso de las mismas coordenadas para cada imagen, que en el caso de k means. Inicialmente se tienen 4 grupos de imágenes, conocidas, es decir, cada grupo tiene imágenes de un objeto diferente y que se sabe que objeto es. Luego al introducir una nueva imagen al conjunto se calcula la distancia, con pitágoras o euclidiana, a cada una de las otras imágenes y se determina a qué grupo pertenece teniendo en cuenta en qué grupo están la mayoría de las n imágenes más cercanas, siendo n un número impar.

---

## Especificación del agente

Un agente racional es aquel que actúa con intención de alcanzar el mejor resultado o el mejor posible, en el caso de que exista incertidumbre. En este caso el agente es el sistema de clasificación y medición de piezas metálicas, cuyo objetivo es el de determinar, de manera eficaz, si una pieza cualquiera se trata de un tornillo, un clavo, una arandela o una tuerca. Además de medir las dimensiones de la misma cuando se trata de un tornillo o un clavo.

Tabla Reas

AGENTE	MEDIDAS DE RENDIMIENTO	ENTORNO	ACTUADORES	SENSORES
Sistema de visión artificial para la clasificación y medición de piezas metálicas	Clasificación y medición correcta de las distintas piezas	Imagen capturada por el sensor	Comunicación mediante palabras escritas con el usuario, sobre las imágenes	Cámara

### Propiedades del entorno

- Totalmente observable

En este caso el agente es capaz de observar por completo la imagen con el objeto que debe clasificar y medir.

- Estocástico

La clasificación de cada imagen se realiza sin tomar en cuenta estados previos.

- Secuencial

El agente clasifica y mide las imágenes para posteriormente comunicarle el resultado al usuario de manera independiente para cada imagen, sin que esto dependa de las acciones previas.

- Estático

---

El entorno no cambia durante la acción del agente, si bien se aplican filtros durante el análisis de cada imagen, no ocurre que se agreguen objetos o cambie la imagen durante la clasificación y medición

- Discreto

Están previamente determinadas las acciones que realiza el agente y el tipo de elementos que el mismo puede analizar. No pudiendo por ejemplo clasificar un destornillador.

- Individual

Es un único agente que se encarga de clasificar y medir los objetos captados por el sensor (cámara)

## Diseño del agente

### Etapas del agente

- Detección y tratamiento de imágenes:

En primer lugar el agente detecta imágenes previamente obtenidas, base de datos de imágenes, en este caso tomadas por el desarrollador del sistema. Para luego realizar un tratamiento a las mismas.

- ➔ Redimensionamiento de la imagen: mediante el uso de la función resize de la librería cv2 se redimensiona cada imagen para tener todas con el mismo tamaño.
- ➔ Paso a escala de grises: mediante el uso de la función cvtColor de la librería cv2 se pasan las imágenes a escala de grises para trabajar con matrices de dos dimensiones y poder extraer mejor la información.
- ➔ Limpieza de ruido: mediante el uso del filtro GaussianBlur de la librería cv2, se hace un emborronamiento de la imagen eliminando ruidos como por ejemplo, puntos o pelos en el fondo de la imagen
- ➔ Detección de bordes y eliminación del relleno: Mediante el uso del filtro Canny de cv2 se detectan los bordes del objeto y solo permanecen los mismos para la extracción de los datos.

- 
- Obtención de los datos de cada imagen:
    - Momentos de Hu: Haciendo uso nuevamente de cv2, obtenemos los momentos de Hu de cada imagen.
    - Alto/Ancho: Se dibuja un rectángulo alrededor de la imagen con la función minAreaRect de cv2 y se toma la relación del lado más largo sobre el más corto. Estos valores luego son usados para la medición.
    - Redondez: Mediante la fórmula  $\frac{4\pi \cdot \text{área} \cdot \text{número\_de\_esquinas}}{\text{perímetro}}$  y con la ayuda de cv2 para detectar el área, el perímetro y el número de esquinas de la figura de obtiene un valor para cada imagen.
  - 
  - Clasificación con k means
    - Inicialización de los centroides: Se eligen 4 imágenes diferentes de la base de datos y con el uso de 3 valores como coordenadas (altura/ancho, redondez, hu2), se establecen como centroides en el espacio
    - Inicialización de grupos o clusters: Se calcula la distancia euclíadiana de cada imagen con sus respectivas coordenadas, a los centroides y se arma un grupo por cada centroide, en cada grupo están las imágenes que tienen como centroide más cercano a dicho centroide.
    - Cálculo de los nuevos centroides: Con los grupos ya armados se calculan los nuevos centroides como la media de cada valor de coordenada teniendo en cuenta todas las imágenes del grupo
    - Armado de los nuevos grupos: Con los nuevos centroides se vuelven a armas los grupos de la misma manera que en el segundo paso.
    - Clasificación de una nueva imagen: en primer lugar realiza un filtrado de la imagen como se realizó con las otras y se extrae la información relevante de la misma, en este caso las coordenadas, luego se la añade las otras imágenes y se recalculan los centroides con este nuevo dato, finalmente esta queda con uno de los grupos.

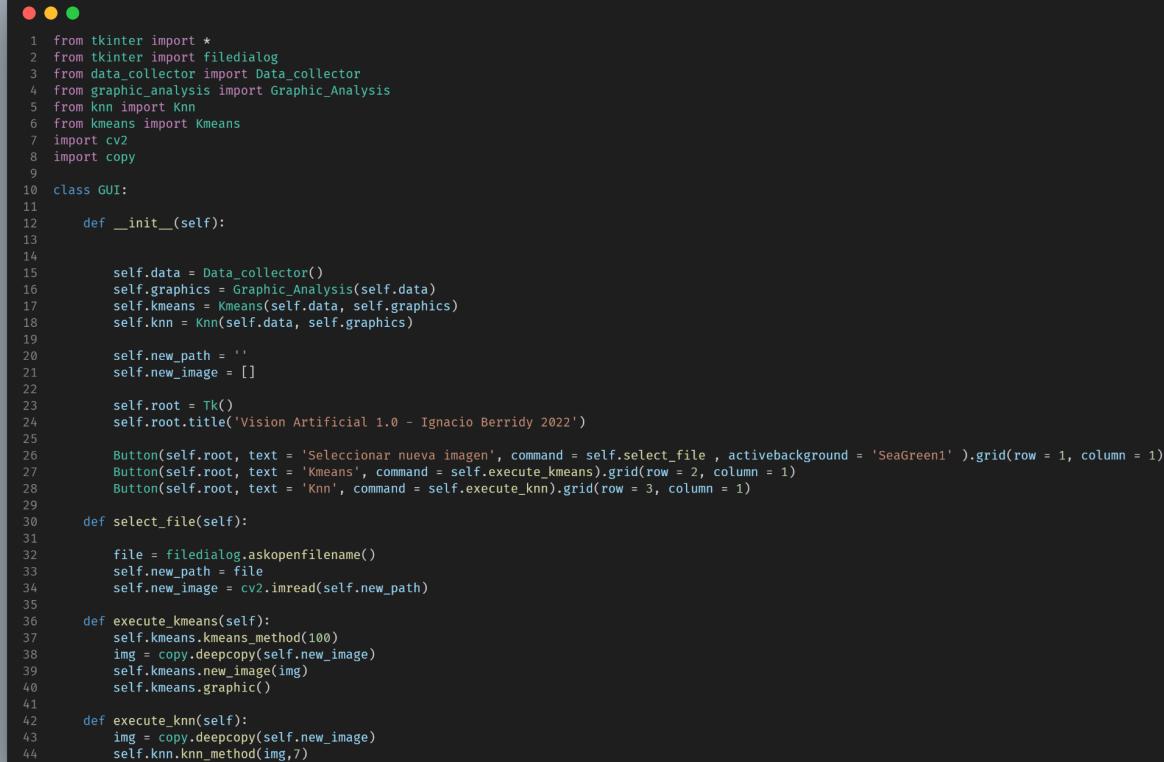
- 
- ➔ Actuador: Se muestra en pantalla la imagen recuadrada y con el nombre (tipo de pieza que corresponde a su grupo) y sus medidas en caso que corresponda.
  - Clasificación con Knn:
    - ➔ Se separan en grupos las imágenes, conociendo qué objeto es cada una. Se tienen en cuenta las coordenadas (altura/ancho, redondez, hu2).
    - ➔ Se introduce una nueva imagen desconocida, se le realiza el tratamiento que se realizó con las demás y se obtienen las coordenadas de la misma
    - ➔ Se calculan las distancias a cada una de las imágenes conocidas, y se agrega la misma al grupo al cual pertenezcan la mayoría de las n imágenes más cercanas.
    - ➔ Actuador: Se muestra en pantalla la imagen recuadrada y con el nombre (tipo de pieza que corresponde a su grupo) y sus medidas en caso que corresponda.

# Código

El código se divide en distintos archivos, haciendo uso de la programación orientada a objetos, correspondiendo cada archivo a una clase

GUI.py

Es la interfaz gráfica con las funciones para ejecutar cada algoritmo



```
 1  from tkinter import *
 2  from tkinter import filedialog
 3  from data_collector import Data_collector
 4  from graphic_analysis import Graphic_Analysis
 5  from Knn import Knn
 6  from kmeans import Kmeans
 7  import cv2
 8  import copy
 9
10 class GUI:
11
12     def __init__(self):
13
14         self.data = Data_collector()
15         self.graphics = Graphic_Analysis(self.data)
16         self.kmeans = Kmeans(self.data, self.graphics)
17         self.knn = Knn(self.data, self.graphics)
18
19         self.new_path = ''
20         self.new_image = []
21
22         self.root = Tk()
23         self.root.title('Vision Artificial 1.0 - Ignacio Berridy 2022')
24
25         Button(self.root, text = 'Seleccionar nueva imagen', command = self.select_file , activebackground = 'SeaGreen1' ).grid(row = 1, column = 1)
26         Button(self.root, text = 'Kmeans', command = self.execute_kmeans).grid(row = 2, column = 1)
27         Button(self.root, text = 'Knn', command = self.execute_knn).grid(row = 3, column = 1)
28
29     def select_file(self):
30
31         file = filedialog.askopenfilename()
32         self.new_path = file
33         self.new_image = cv2.imread(self.new_path)
34
35     def execute_kmeans(self):
36         self.kmeans.kmeans_method(100)
37         img = copy.deepcopy(self.new_image)
38         self.kmeans.new_image(img)
39         self.kmeans.graphic()
40
41     def execute_knn(self):
42         img = copy.deepcopy(self.new_image)
43         self.knn.knn_method(img,7)
```

data\_collector.py

Este archivo/clase, se encarga de cargar las imágenes, realizar su tratamiento y obtener la información de las mismas. (Por la longitud del archivo se pega por partes pero es único archivo)

```

● ● ●

1 import cv2
2 import os
3 import math
4 import numpy as np
5 import pandas as pd
6 from pathlib import Path
7 from list_functions import *
8
9 class Data_collector:
10     def __init__(self, image_path = str(Path.cwd())):
11
12
13
14     #Imagenes
15     self.screws = self.load_images(image_path + '\\\\Images\\\\screw')
16     self.nails = self.load_images(image_path + '\\\\Images\\\\nail')
17     self.nuts = self.load_images(image_path + '\\\\Images\\\\nut')
18     self.washers = self.load_images(image_path + '\\\\Images\\\\washer')
19
20     #Imagenes en escala de grises
21     self.gray_screws = self.gray_scale_list(self.screws)
22     self.gray_nails = self.gray_scale_list(self.nails)
23     self.gray_nuts = self.gray_scale_list(self.nuts)
24     self.gray_washers = self.gray_scale_list(self.washers)
25
26     #Imagenes con filtro gaussiano
27     self.gaussian_screws = self.gaussianblur_list(self.gray_screws)
28     self.gaussian_nails = self.gaussianblur_list(self.gray_nails)
29     self.gaussian_nuts = self.gaussianblur_list(self.gray_nuts)
30     self.gaussian_washers = self.gaussianblur_list(self.gray_washers)
31
32     #Imagenes con filtro canny
33     self.canny_screws = self.canny_list(self.gaussian_screws)
34     self.canny_nails = self.canny_list(self.gaussian_nails)
35     self.canny_nuts = self.canny_list(self.gaussian_nuts)
36     self.canny_washers = self.canny_list(self.gaussian_washers)
37     self.canny = [self.canny_screws, self.canny_nails, self.canny_nuts, self.canny_washers]
38
39
40     #Imagenes con filtro binario
41     self.binary_screws = self.binary_filter_list(self.gray_screws)
42     self.binary_nails = self.binary_filter_list(self.gray_nails)
43     self.binary_nuts = self.binary_filter_list(self.gray_nuts)
44     self.binary_washers = self.binary_filter_list(self.gray_washers)
45     self.binary = [self.binary_screws, self.binary_nails, self.binary_nuts, self.binary_washers]
46
47     #Imagenes solo con contorno
48
49     self.contour_screws = self.contour_images(self.canny_screws)
50     self.contour_nails = self.contour_images(self.canny_nails)
51     self.contour_nuts = self.contour_images(self.canny_nuts)
52     self.contour_washers = self.contour_images(self.canny_washers)
53     self.cont = [self.contour_screws, self.contour_nails, self.contour_nuts, self.contour_washers]
54
55     #Variables con informacion de las imagenes
56
57     self.canny_hu = []
58     self.canny_contour = []
59     self.canny_roundness = []
60     self.canny_slenderness = []
61     self.binary_hu = []
62     self.binary_contour = []
63     self.binary_roundness = []
64     self.binary_slenderness = []
65     self.fill_information()
66
67     #Data frame con la informacion de cada grupo de imagenes
68     self.binary_screw_df = pd.DataFrame()
69     self.canny_screw_df = pd.DataFrame()
70     self.binary_nail_df = pd.DataFrame()
71     self.canny_nail_df = pd.DataFrame()
72     self.binary_nut_df = pd.DataFrame()
73     self.canny_nut_df = pd.DataFrame()
74     self.binary_washer_df = pd.DataFrame()
75     self.canny_washer_df = pd.DataFrame()
76     self.organize_information()
77     self.canny_screw_df.to_csv('screws.csv')
78     self.canny_nail_df.to_csv('nails.csv')
79     self.canny_nut_df.to_csv('nuts.csv')
80     self.canny_washer_df.to_csv('washers.csv')
81     self.canny_df = [self.canny_screw_df, self.canny_nail_df, self.canny_nut_df, self.canny_washer_df]
82     self.binary_df = [self.binary_screw_df, self.binary_nail_df, self.binary_nut_df, self.binary_washer_df]
83

```

```

1 #Funciones de tratamiento de imagen
2
3 def load_images(self, dir):
4     """
5         Recibe un directorio y devuelve una lista con las imágenes
6         ...
7     paths = os.listdir(dir)
8     images = []
9
10    for i in paths:
11        path = dir+'\\"+i
12        images.append(cv2.resize(cv2.imread(path), (400, 400)))
13
14    return images
15
16 def gray_scale_list(self, img_list):
17     """
18         Recibe una lista de imágenes y las devuelve en escala de grises
19         ...
20     gray_images = []
21
22    for i in img_list:
23        gray_images.append(cv2.cvtColor(i, cv2.COLOR_BGR2GRAY))
24
25    return gray_images
26
27 def gaussianblur_list(self, img_list):
28     """
29         Recibe una lista de imágenes y devuelve otra lista con
30         las imágenes luego de aplicarles el filtro gaussiano
31         ...
32     filtered_images = []
33
34    for i in img_list:
35        filtered_images.append(cv2.GaussianBlur(i, (5, 5), 0))
36
37    return filtered_images
38
39 def canny_list(self, img_list):
40
41     """
42         Recibe una lista de imágenes y devuelve otra lista con
43         las imágenes luego de aplicarles el filtro Canny
44         ...
45     filtered_images = []
46    for i in img_list:
47        i = cv2.Canny(i, 50, 150)
48        i = cv2.dilate(i, None, iterations = 1)
49        i = cv2.erode(i, None, iterations = 1)
50        filtered_images.append(i)
51
52    return filtered_images
53
54 def binary_filter_list(self, img_list):
55
56     """
57         Recibe una lista de imágenes y las devuelve filtradas con un filtro
58         binario
59         ...
60     filtered_images = []
61
62    for i in img_list:
63        a, inv_binary = cv2.threshold(i, 100, 255, cv2.THRESH_BINARY_INV)
64        filtered_images.append(inv_binary)
65
66    return filtered_images
67
68 def contour_images(self, img_list):
69
70    images = []
71
72    for i in img_list:
73        contours, hierarchy = cv2.findContours(i, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
74        draw = np.zeros((i.shape[0], i.shape[1], 3), dtype=np.uint8)
75        i = cv2.drawContours(draw, contours, -1, (0, 255, 0), 5)
76        i = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)
77        #i = cv2.dilate(i, None, iterations = 1)
78        #i = cv2.erode(i, None, iterations = 1)
79        images.append(i)
80
81    return images
82
83

```

```

1 #Funciones de extraccion de datos de la imagen
2
3     def fill_information(self):
4
5         for i in self.cont:
6             self.canny_hu.append(self.hu_moments(i))
7             for i in self.canny:
8                 self.canny_contour.append(self.contour_list(i))
9                 for i in self.canny_contour:
10                    self.canny_roundness.append(self.roundness_list(i))
11                    self.canny_slenderness.append(self.slenderness_list(i))
12
13         for i in self.binary:
14             self.binary_hu.append(self.hu_moments(i))
15             self.binary_contour.append(self.contour_list(i))
16             for i in self.binary_contour:
17                 self.binary_roundness.append(self.roundness_list(i))
18                 self.binary_slenderness.append(self.slenderness_list(i))
19
20     def organize_information(self):
21
22         hu_names = ['hu1', 'hu2', 'hu3', 'hu4', 'hu5', 'hu6', 'hu7']
23
24         canny_screw_dict = {'roundness':self.canny_roundness[0],
25                             'slenderness':self.canny_slenderness[0]}
26         binary_screw_dict = {'roundness':self.binary_roundness[0],
27                             'slenderness':self.binary_slenderness[0]}
28         hu_counter = 0
29         for i in hu_names:
30             canny_screw_dict[i] = []
31             binary_screw_dict[i] = []
32             for j in self.canny_hu[0]:
33                 canny_screw_dict[i].append(j[hu_counter])
34             for l in self.binary_hu[0]:
35                 binary_screw_dict[i].append(l[hu_counter])
36             hu_counter = hu_counter + 1
37
38         canny_nail_dict = {'roundness':self.canny_roundness[1],
39                            'slenderness':self.canny_slenderness[1]}
40         binary_nail_dict = {'roundness':self.binary_roundness[1],
41                            'slenderness':self.binary_slenderness[1]}
42         hu_counter = 0
43         for i in hu_names:
44             canny_nail_dict[i] = []
45             binary_nail_dict[i] = []
46             for j in self.canny_hu[1]:
47                 canny_nail_dict[i].append(j[hu_counter])
48             for l in self.binary_hu[1]:
49                 binary_nail_dict[i].append(l[hu_counter])
50             hu_counter = hu_counter + 1
51
52
53         canny_nut_dict = {'roundness':self.canny_roundness[2],
54                           'slenderness':self.canny_slenderness[2]}
55         binary_nut_dict = {'roundness':self.binary_roundness[2],
56                           'slenderness':self.binary_slenderness[2]}
57         hu_counter = 0
58         for i in hu_names:
59             canny_nut_dict[i] = []
60             binary_nut_dict[i] = []
61             for j in self.canny_hu[2]:
62                 canny_nut_dict[i].append(j[hu_counter])
63             for l in self.binary_hu[2]:
64                 binary_nut_dict[i].append(l[hu_counter])
65             hu_counter = hu_counter + 1
66
67
68         canny_washer_dict = {'roundness':self.canny_roundness[3],
69                           'slenderness':self.canny_slenderness[3]}
70         binary_washer_dict = {'roundness':self.binary_roundness[3],
71                           'slenderness':self.binary_slenderness[3]}
72         hu_counter = 0
73         for i in hu_names:
74             canny_washer_dict[i] = []
75             binary_washer_dict[i] = []
76             for j in self.canny_hu[3]:
77                 canny_washer_dict[i].append(j[hu_counter])
78             for l in self.binary_hu[3]:
79                 binary_washer_dict[i].append(l[hu_counter])
80             hu_counter = hu_counter + 1
81
82
83         self.binary_screw_df = pd.DataFrame(binary_screw_dict)
84         self.canny_screw_df = pd.DataFrame(canny_screw_dict)
85
86         self.binary_nail_df = pd.DataFrame(binary_nail_dict)
87         self.canny_nail_df = pd.DataFrame(canny_nail_dict)
88
89         self.binary_nut_df = pd.DataFrame(binary_nut_dict)
90         self.canny_nut_df = pd.DataFrame(canny_nut_dict)
91
92         self.binary_washer_df = pd.DataFrame(binary_washer_dict)
93         self.canny_washer_df = pd.DataFrame(canny_washer_dict)
94
95

```

```

1  def contour_list(self, img_list):
2
3      ...
4      Recibe una lista de imagenes y devuelve una matriz con los
5      contornos de cada imagen
6      ...
7
8      contours_list = []
9
10     for i in img_list:
11         contours = []
12         hierarchy = []
13         contours,hierarchy = cv2.findContours(i, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
14         contours_list.append(contours)
15
16     return contours_list
17
18 def hu_moments(self, img_list):
19
20     ...
21     Recibe una lista de imagenes y devuelve una lista con sus momentos de hu escalados
22     ...
23     Hu = []
24     for i in img_list:
25         hu_list = []
26         moment = cv2.moments(i)
27         hu_moments = (cv2.HuMoments(moment))
28         for j in range(len(hu_moments)):
29             hu_list.append(float(-1 * math.copysign(1.0, hu_moments[j]) * math.log10(abs(hu_moments[j]))))
30         Hu.append(hu_list)
31
32     return Hu
33
34 def roundness_list(self, con_list):
35
36     ...
37     Recibe una matriz con contornos de una lista de imagen y devuelve una lista
38     con ru redondez
39     ...
40     roundness = []
41     p = math.pi
42
43     for i in con_list:
44         r = 0
45         area = 0
46         arc_length = 0
47
48         for j in i:
49             area = area + cv2.contourArea(j)
50             arc_length = arc_length + cv2.arcLength(j, True)
51             coef = 0.03 * cv2.arcLength(j, True)
52             n = cv2.approxPolyDP(j, coef, True)
53
54             try:
55                 r = (4*p*area)/(arc_length**2)
56                 #round(r, 2)*
57             except:
58                 roundness.append('Error')
59             roundness.append(round(len(n)*r, 5))
60
61     return roundness
62
63 def slenderness_list(self, con_list):
64
65     ...
66     Recibe una matriz con los contornos de una lista de imagenes
67     y devuelve un valor de esbeltez para cada una de las mismas
68     ...
69     slenderness = []
70     for i in con_list:
71         height = 0
72         width = 0
73         for j in i:
74             (x, y), (current_width, current_height), angle = cv2.minAreaRect(j)
75             if current_height < current_width:
76                 current_width, current_height = current_height, current_width
77             if current_width > width:
78                 width = current_width
79             if current_height > height:
80                 height = current_height
81
82             try:
83                 slenderness.append(round(height/width, 2))
84             except:
85                 slenderness.append('Error')
86
87     return slenderness

```

```

1  def single_img(self, img):
2
3      img = cv2.resize(img, (400, 400))
4      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
5
6      img_canny = cv2.GaussianBlur(img, (5, 5), 0)
7      img_canny = cv2.Canny(img_canny, 50, 150)
8      img_canny = cv2.dilate(img_canny, None, iterations = 1)
9      img_canny = cv2.erode(img_canny, None, iterations = 1)
10
11     contours, hierarchy = cv2.findContours(img_canny, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
12     draw = np.zeros((img_canny.shape[0], img_canny.shape[1], 3), dtype=np.uint8)
13     img_contour = cv2.drawContours(draw, contours, -1, (0, 255, 0), 5)
14     img_contour = cv2.cvtColor(img_contour, cv2.COLOR_BGR2GRAY)
15
16     moments = cv2.moments(img_contour)
17     hu = cv2.HuMoments(moments)
18     HU = []
19     for j in range(len(hu)):
20         HU.append(float(-1* math.copysign(1.0, hu[j])* math.log10(abs(hu[j]))))
21
22
23     p = math.pi
24     r = 0
25     area = 0
26     arc_length = 0
27
28     height = 0
29     width = 0
30
31     for i in contours:
32         area = area + cv2.contourArea(i)
33         arc_length = arc_length + cv2.arcLength(i, True)
34         coef = 0.03 * cv2.arcLength(i, True)
35         n = cv2.approxPolyDP(i, coef, True)
36         r = (4*p*area)/(arc_length**2)
37
38         rect = cv2.minAreaRect(i)
39         (x, y), (current_width, current_height), angle = rect
40         if current_height<current_width:
41             current_width, current_height = current_height, current_width
42         if current_width>width:
43             width = current_width
44         if current_height>height:
45             height = current_height
46     try:
47         slenderness = (round(height/width, 2))
48     except:
49         pass
50
51     roundness = (round(len(n)*r, 5))
52     return(roundness, slenderness, HU, height, width, rect)
53

```

## graphic\_analysis.py

Se encarga de renderizar cuando sea necesario los gráficos o las imágenes

```

● ● ●
1 from cProfile import label
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.mplot3d import axes3d
6
7
8 class Graphic_Analysis:
9
10    def __init__(self, data):
11        self.data = data
12
13    def hu_plot(self, n, filter_df):
14        hu_names = ['hu1', 'hu2', 'hu3', 'hu4', 'hu5', 'hu6', 'hu7']
15        tools = ['Screws', 'Nails', 'Nuts', 'Washers']
16
17        if n == 0:
18            counter = 0
19            for i in filter_df:
20                fig , ax = plt.subplots()
21                plt.title('Hu moments: ' + tools[counter])
22                plt.xlabel('Image')
23
24                plt.ylabel('hu values')
25                counter = counter +1
26                for j in hu_names:
27                    ax.plot(i[j], marker = 'o', label = j)
28
29                plt.legend()
30
31        if n != 0:
32            counter = 0
33            fig , ax = plt.subplots()
34            for i in self.data.canny_df:
35                plt.title(hu_names[n-1])
36                plt.xlabel('Image')
37                plt.ylabel('hu values')
38                ax.plot(i[hu_names[n-1]], marker = 'o', label = tools[counter])
39                plt.legend()
40                counter = counter +1
41
42            plt.show()
43            cv2.waitKey(0)
44            cv2.destroyAllWindows()
45
46    def roundness_plot(self, filter_df):
47
48        tools = ['Screws', 'Nails', 'Nuts', 'Washers']
49        counter = 0
50        fig, ax = plt.subplots()
51        plt.title('Roundness')
52        plt.xlabel('Image')
53        plt.ylabel('Roundness values')
54        for i in filter_df:
55            ax.plot(i['roundness'], marker = 'o', label = tools[counter])
56            counter = counter +1
57        plt.legend()
58        plt.show()
59        cv2.waitKey(0)
60        cv2.destroyAllWindows()
61
62    def slenderness_plot(self, filter_df):
63
64        tools = ['Screws', 'Nails', 'Nuts', 'Washers']
65        counter = 0
66        fig, ax = plt.subplots()
67        plt.title('height/width')
68        plt.xlabel('Image')
69        plt.ylabel('height/width values')
70        for i in filter_df:
71            ax.plot(i['slenderness'], marker = 'o', label = tools[counter])
72            counter = counter +1
73        plt.legend()
74        plt.show()
75        cv2.waitKey(0)
76        cv2.destroyAllWindows()
77

```

```

1  def img_visualization(self, rectangle, img, l, tool):
2
3      (x, y) = rectangle[0]
4      (al, an) = rectangle[1]
5      rect = cv2.boxPoints(rectangle)
6      rect = np.int0(rect)
7      cv2.polylines(img, [rect], True, (0, 255, 0), 2)
8      cv2.putText(img, tool, (int(x)+10, int(y)-70), cv2.LINE_AA, 0.8, (255, 0, 0), 2)
9      if l == True:
10          cv2.putText(img, 'Alto: %s' % round(al, 2), (int(x)+10, int(y)-15), cv2.LINE_AA, 0.8, (255, 0, 0), 2)
11          cv2.putText(img, 'Ancho: %s' % round(an, 2), (int(x)+10, int(y)+15), cv2.LINE_AA, 0.8, (255, 0, 0), 2)
12      cv2.imshow('Imagen', img)
13      cv2.waitKey(0)
14      cv2.destroyAllWindows()
15
16  def k_means_graphic(self, cluster, cen):
17
18      def extract_values(cluster):
19
20          x = []
21          y = []
22          z = []
23
24          for i in cluster:
25
26              x.append(i[0])
27              y.append(i[1])
28              z.append(i[2])
29
30          return((x, y, z))
31
32
33      fig = plt.figure()
34      (x1, y1, z1) = extract_values(cluster[0])
35      (x2, y2, z2) = extract_values(cluster[1])
36      (x3, y3, z3) = extract_values(cluster[2])
37      (x4, y4, z4) = extract_values(cluster[3])
38
39      ax1 = fig.add_subplot(111, projection='3d')
40      ax1.scatter(x1, y1, z1, c = 'b', label = 'Tornillos')
41      ax1.scatter(cen[0][0], cen[0][1], cen[0][2], c = 'b', marker = 'c')
42      ax1.scatter(x2, y2, z2, c = 'y', label = 'Clavos')
43      ax1.scatter(cen[1][0], cen[1][1], cen[1][2], c = 'y', marker = 'c')
44      ax1.scatter(x3, y3, z3, c = 'g', label = 'Tuercas')
45      ax1.scatter(cen[2][0], cen[2][1], cen[2][2], c = 'g', marker = 'c')
46      ax1.scatter(x4, y4, z4, c = 'r', label = 'Arandelas')
47      ax1.scatter(cen[3][0], cen[3][1], cen[3][2], c = 'r', marker = 'c')
48
49      ax1.set_xlabel('hu2')
50      ax1.set_ylabel('slenderness')
51      ax1.set_zlabel('roundness')
52
53      plt.legend()
54      plt.show()
55
56      cv2.waitKey(0)
57      cv2.destroyAllWindows()

```

---

## data\_analysis.py

Se utilizó para analizar la información de las imágenes y determinar qué elementos utilizar

```
● ● ●

1 from data_collector import Data_collector
2 from graphic_analysis import Graphic_Analysis
3
4 data = Data_collector()
5 graphics = Graphic_Analysis(data)
6
7 for i in data.canny_df:
8     i.to_csv('data.csv')
9
10 graphics.slenderness_plot(graphics.data.canny_df)
11 graphics.roundness_plot(graphics.data.canny_df)
12 for i in range(1,8):
13     graphics.hu_plot(i, graphics.data.canny_df)
```

## K means

Se encarga de ejecutar el algoritmo de de clasificación k means

```

1 import cv2
2
3 class Kmeans():
4
5     def __init__(self, data, graphics):
6
7         self.data = data
8         self.graphics = graphics
9
10        self.screws = []
11        self.nails = []
12        self.nuts = []
13        self.washers = []
14        self.images = []
15
16        self.first_centers()
17
18        self.cen_1 = (self.screws[5][0], self.screws[5][1], self.screws[5][2])
19        self.cen_2 = (self.nails[0][0], self.nails[0][1], self.nails[0][2])
20        self.cen_3 = (self.nuts[0][0], self.nuts[0][1], self.nuts[0][2])
21        self.cen_4 = (self.washers[0][0], self.washers[0][1], self.washers[0][2])
22
23        self.cen = (self.cen_1, self.cen_2, self.cen_3, self.cen_4)
24
25        self.cluster = []
26
27    def first_centers(self):
28
29        for i in self.data.canny_df[0].index:
30            self.images.append((self.data.canny_df[0]['hu2'][i], self.data.canny_df[0]['slenderness'][i], self.data.canny_df[0]['roundness'][i]))
31            self.screws.append((self.data.canny_df[0]['hu2'][i], self.data.canny_df[0]['slenderness'][i], self.data.canny_df[0]['roundness'][i]))
32
33        for i in self.data.canny_df[1].index:
34            self.images.append((self.data.canny_df[1]['hu2'][i], self.data.canny_df[1]['slenderness'][i], self.data.canny_df[1]['roundness'][i]))
35            self.nails.append((self.data.canny_df[1]['hu2'][i], self.data.canny_df[1]['slenderness'][i], self.data.canny_df[1]['roundness'][i]))
36
37        for i in self.data.canny_df[2].index:
38            self.images.append((self.data.canny_df[2]['hu2'][i], self.data.canny_df[2]['slenderness'][i], self.data.canny_df[2]['roundness'][i]))
39            self.nuts.append((self.data.canny_df[2]['hu2'][i], self.data.canny_df[2]['slenderness'][i], self.data.canny_df[2]['roundness'][i]))
40
41        for i in self.data.canny_df[3].index:
42            self.images.append((self.data.canny_df[3]['hu2'][i], self.data.canny_df[3]['slenderness'][i], self.data.canny_df[3]['roundness'][i]))
43            self.washers.append((self.data.canny_df[3]['hu2'][i], self.data.canny_df[3]['slenderness'][i], self.data.canny_df[3]['roundness'][i]))
44
45    def distance(self, images, cen):
46
47        cluster_1 = []
48        cluster_2 = []
49        cluster_3 = []
50        cluster_4 = []
51
52        for i in images:
53
54            distance_1 = (((i[0]-cen[0][0])**2) + ((i[1]-cen[0][1])**2) + ((i[2]-cen[0][2])**2))** (1/2)
55            distance_2 = (((i[0]-cen[1][0])**2) + ((i[1]-cen[1][1])**2) + ((i[2]-cen[1][2])**2))** (1/2)
56            distance_3 = (((i[0]-cen[2][0])**2) + ((i[1]-cen[2][1])**2) + ((i[2]-cen[2][2])**2))** (1/2)
57            distance_4 = (((i[0]-cen[3][0])**2) + ((i[1]-cen[3][1])**2) + ((i[2]-cen[3][2])**2))** (1/2)
58
59            if distance_1 < distance_2 and distance_1 < distance_3 and distance_1 < distance_4:
60                cluster_1.append(i[:])
61            elif distance_2 < distance_1 and distance_2 < distance_3 and distance_2 < distance_4:
62                cluster_2.append(i[:])
63            elif distance_3 < distance_1 and distance_3 < distance_2 and distance_3 < distance_4:
64                cluster_3.append(i[:])
65            elif distance_4 < distance_1 and distance_4 < distance_2 and distance_4 < distance_3:
66                cluster_4.append(i[:])
67
68
69        return(cluster_1, cluster_2, cluster_3, cluster_4)
70
71

```

```

1 def recalculate_centroid(self, cluster_list ,cent):
2
3     cen = []
4     cont = 0
5     for i in cluster_list:
6         n = len(i)
7         x = 0
8         y = 0
9         z = 0
10        for j in i:
11            x = x + j[0]
12            y = y + j[1]
13            z = z + j[2]
14        try:
15            cen.append((x/n, y/n, z/n))
16        except:
17            cen.append(cent[cont])
18        cont = cont+1
19
20    return cen
21
22 def kmeans_method(self, n):
23
24     it = 0
25
26     while it<n:
27
28         it = it + 1
29         self.cluster = self.distance(self.images, self.cen)
30         self.cen = self.recalculate_centroid(self.cluster, self.cen)
31
32 def graphic(self):
33
34     self.graphics.k_means_graphic(self.cluster, self.cen)
35
36 def new_image(self, img):
37
38     r, s, h, al, an, rect= self.data.single_img(img)
39     img_filtered = (h[1], s, r)
40     img = cv2.resize(img, (400, 400))
41     self.images.append(img_filtered)
42     self.kmeans_method(100)
43
44     if img_filtered in self.cluster[0]:
45         #print('ES UN TORNILLO y su medida es: alto = %s ancho = %s'%(al, an))
46         self.graphics.img_visualization(rect, img, True, 'TORNILLO')
47     elif img_filtered in self.cluster[1]:
48         #print('ES UN CLAVO y su medida es: alto = %s ancho = %s'%(al, an))
49         self.graphics.img_visualization(rect, img, True, 'CLAVO')
50     elif img_filtered in self.cluster[2]:
51         #print('ES UNA TUERCA')
52         self.graphics.img_visualization(rect, img, False, 'TUERCA')
53     elif img_filtered in self.cluster[3]:
54         #print('ES UNA ARANDELA')
55         self.graphics.img_visualization(rect, img, False, 'ARANDELA')
56

```

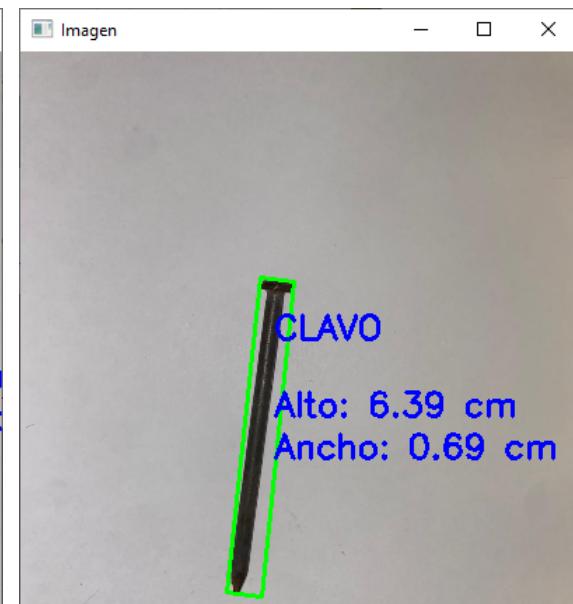
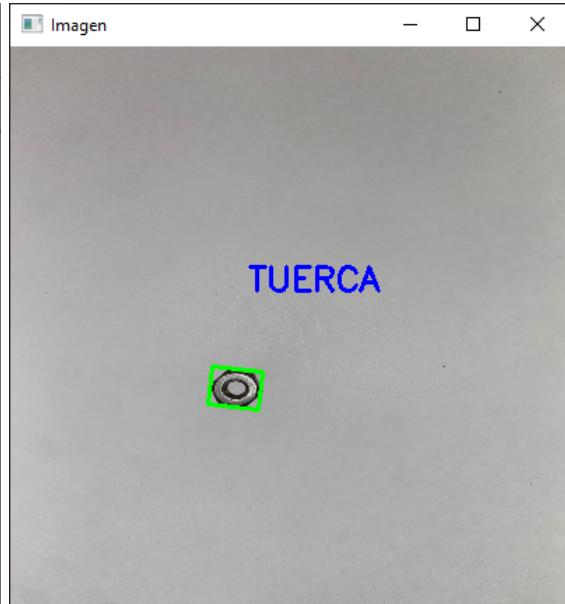
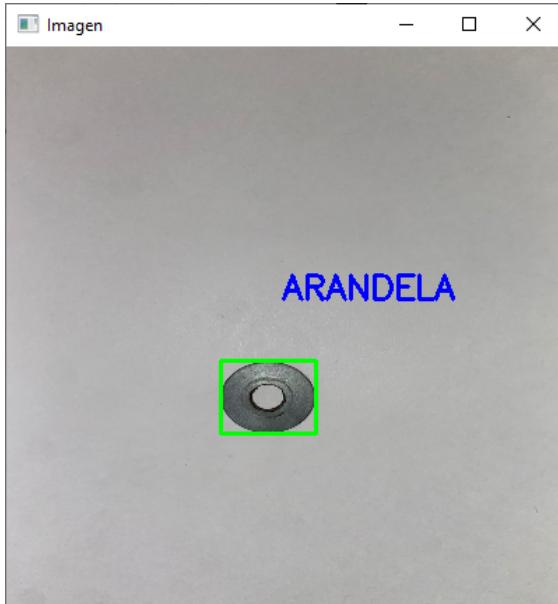
## Knn

Se encarga de ejecutar el algoritmo de clasificación knn

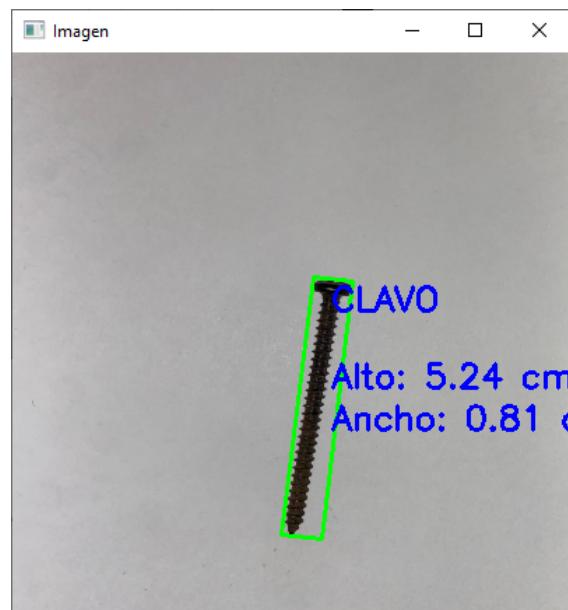
```
1 import cv2
2
3 class Knn:
4
5     def __init__(self, data, graphic):
6
7         self.data = data
8         self.graphic = graphic
9         self.screws = []
10        self.nails = []
11        self.nuts = []
12        self.washers = []
13        self.data_split()
14
15    def data_split(self):
16
17        for i in self.data.canny_df[0].index:
18            self.screws.append((self.data.canny_df[0]['hu2'][i], self.data.canny_df[0]['slenderness'][i], self.data.canny_df[0]['roundness'][i]))
19
20        for i in self.data.canny_df[1].index:
21            self.nails.append((self.data.canny_df[1]['hu2'][i], self.data.canny_df[1]['slenderness'][i], self.data.canny_df[1]['roundness'][i]))
22
23        for i in self.data.canny_df[2].index:
24            self.nuts.append((self.data.canny_df[2]['hu2'][i], self.data.canny_df[2]['slenderness'][i], self.data.canny_df[2]['roundness'][i]))
25
26        for i in self.data.canny_df[3].index:
27            self.washers.append((self.data.canny_df[3]['hu2'][i], self.data.canny_df[3]['slenderness'][i], self.data.canny_df[3]['roundness'][i]))
28
29    def distance(self, img):
30
31        distance = []
32        for i in self.screws:
33            distance.append((((img[0]-i[0])**2) + ((img[1]-i[1])**2) + ((img[2]-i[2])**2))**(.5), 'screw')
34        for i in self.nails:
35            distance.append((((img[0]-i[0])**2) + ((img[1]-i[1])**2) + ((img[2]-i[2])**2))**(.5), 'nail')
36        for i in self.nuts:
37            distance.append((((img[0]-i[0])**2) + ((img[1]-i[1])**2) + ((img[2]-i[2])**2))**(.5), 'nuts')
38        for i in self.washers:
39            distance.append((((img[0]-i[0])**2) + ((img[1]-i[1])**2) + ((img[2]-i[2])**2))**(.5), 'washer')
40
41        return(distance)
42
43
44    def knn_method(self, img, n):
45
46        r, s, h, al, an , rect = self.data.single_img(img)
47        img_filtered = (h[1], s, r)
48        distances = self.distance(img_filtered)
49        distances.sort()
50        neighbours = distances[0:n]
51        img = cv2.resize(img, (400, 400))
52        s = 0
53        na = 0
54        nu = 0
55        w = 0
56
57        for i in neighbours:
58            if i[1] == 'screw':
59                s = s + 1
60            elif i[1] == 'nail':
61                na = na + 1
62            elif i[1] == 'nut':
63                nu = nu + 1
64            elif i[1] == 'washer':
65                w = w + 1
66
67        if s >= na and s >= nu and s >= w:
68            print('ES UN TORNILLO y su medida es: alto = %s ancho = %s'%(al, an))
69            self.graphic.img_visualization(rect, img, True, 'TORNILLO')
70        if na >= s and na >= nu and na >= w:
71            print('ES UN CLAVO y su medida es: alto = %s ancho = %s'%(al, an))
72            self.graphic.img_visualization(rect, img, True, 'CLAVO')
73        if nu >= s and nu >= na and nu >= w:
74            print('ES UNA TUERCA')
75            self.graphic.img_visualization(rect, img, False, 'TUERCA')
76        if w >= s and w >= na and w >= nu:
77            print('ES UNA ARANDELA')
78            self.graphic.img_visualization(rect, img, False, 'ARANDELA')
```

## Ejemplos de aplicación

A continuación se muestran resultados la aplicación del sistema a distintos elementos

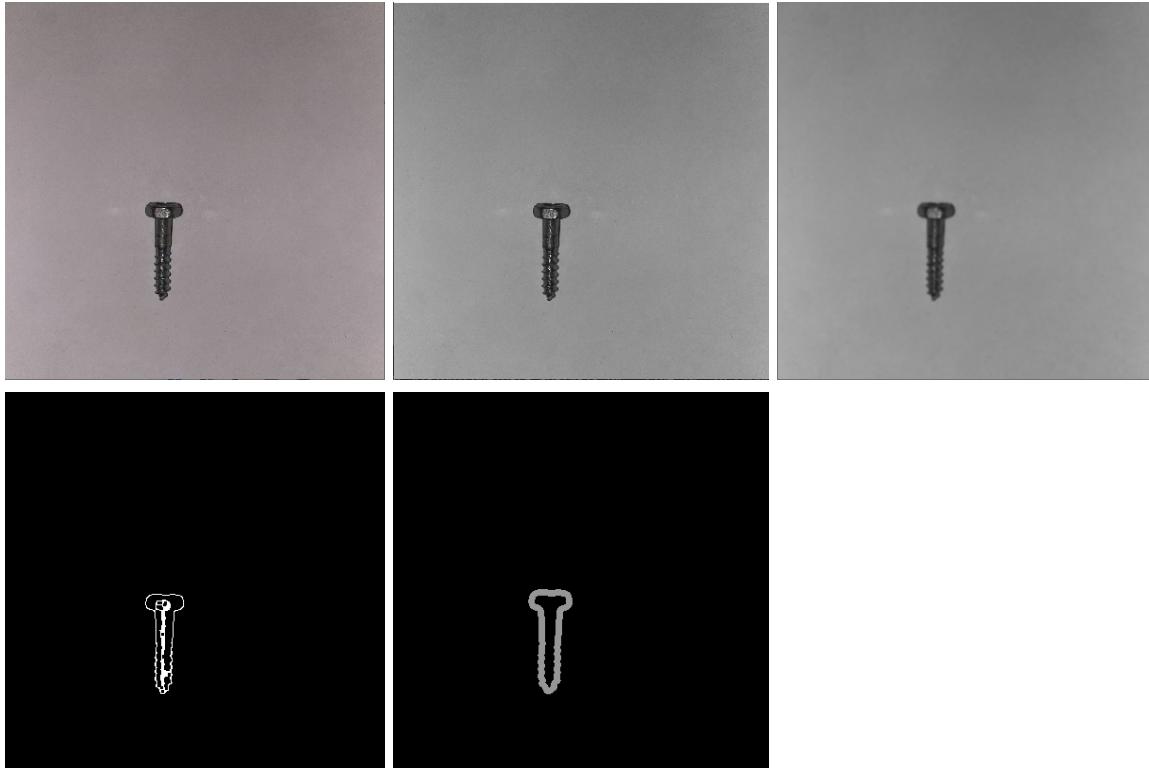


Cabe destacar que la eficiencia de ambos algoritmos demostró ser similar en la mayoría de los casos, sin embargo para algunos casos como por ejemplo el del tornillo antes mostrado Knn arrojó un resultado incorrecto:



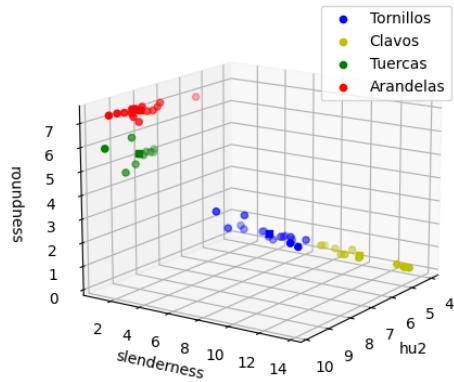
## Resultados

Filtrado de las imágenes



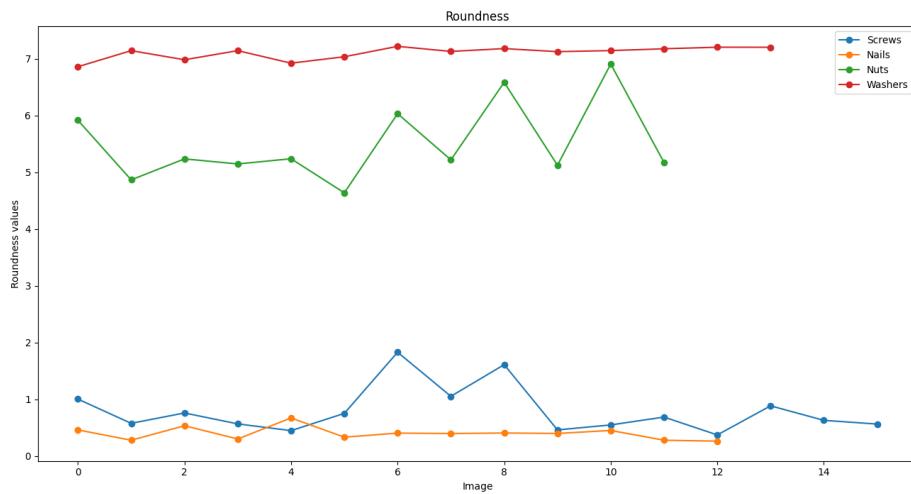
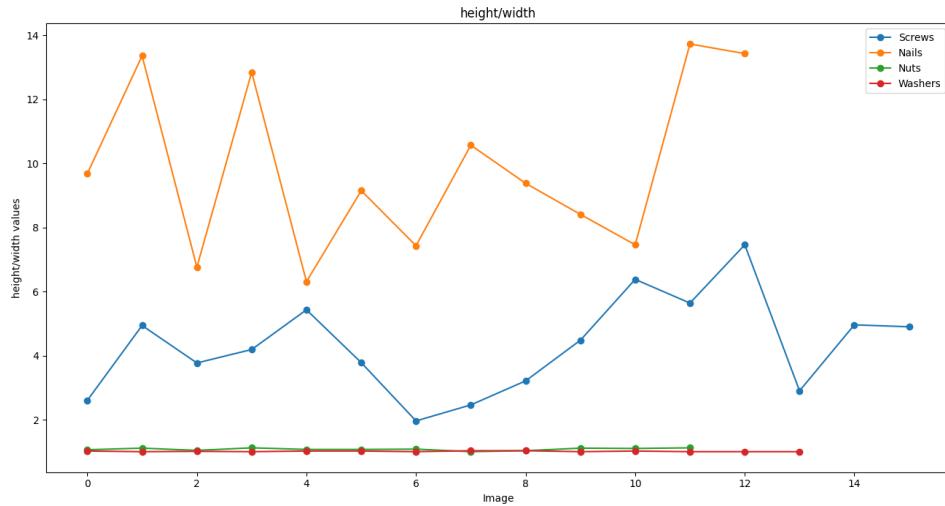
Podemos observar el tratamiento realizado a las imágenes, en este caso aplicado a uno de los tornillos.

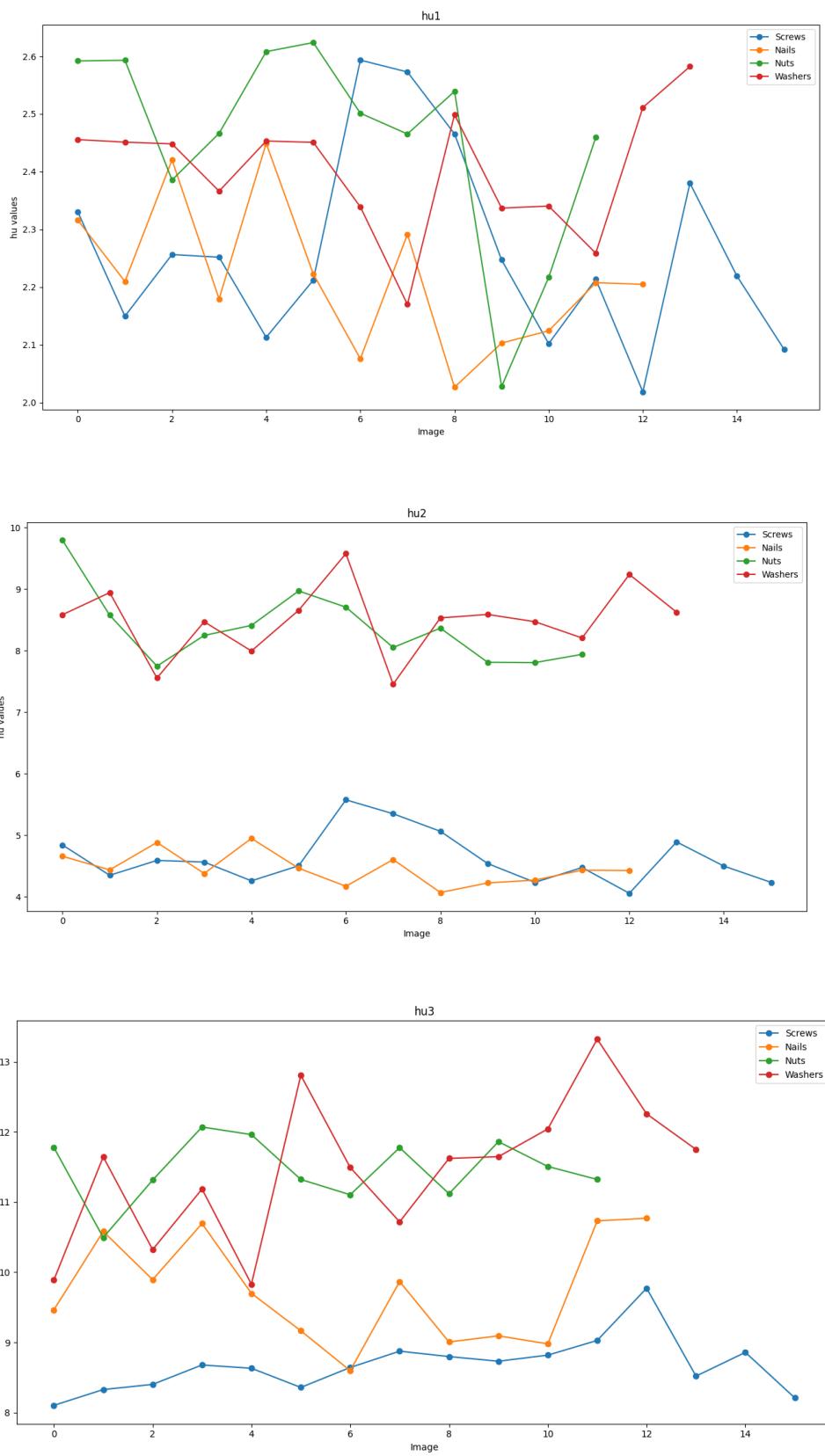
Separación del algoritmo k means : Luego de realizar el algoritmo de k means se obtuvieron los grupos mostrado en la imagen

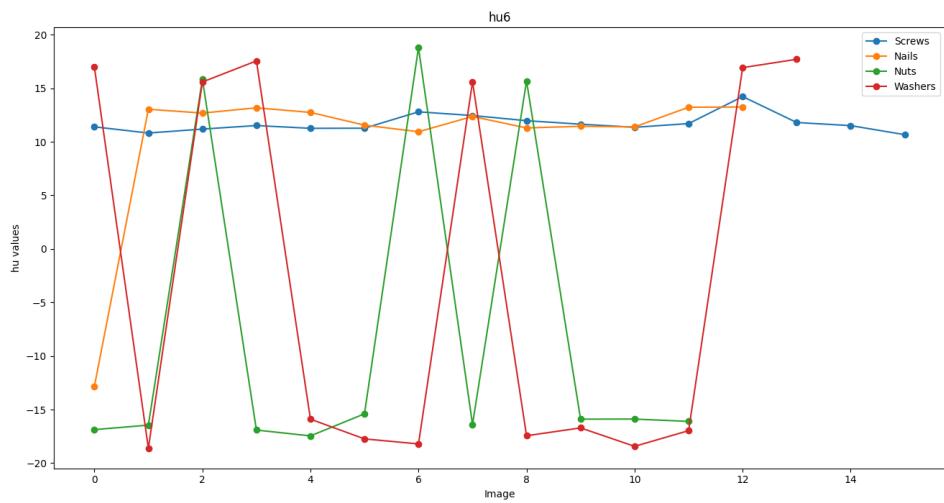
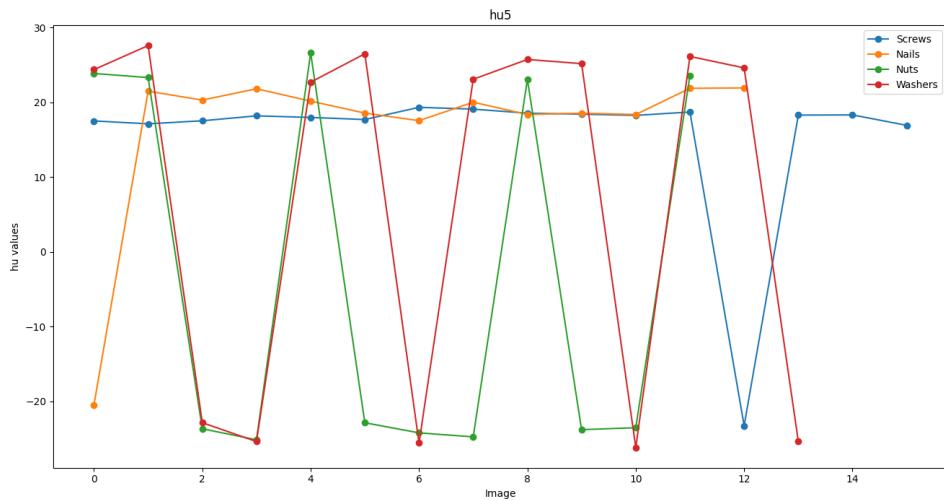
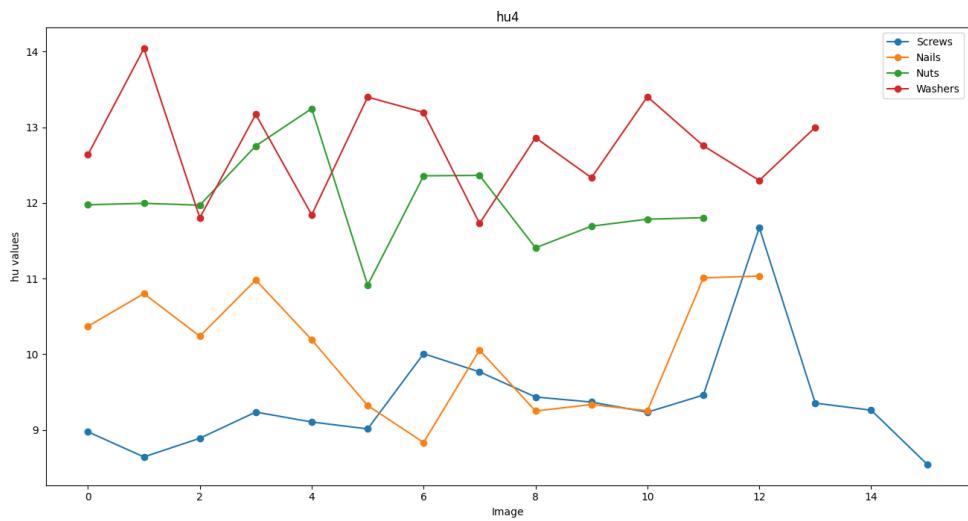


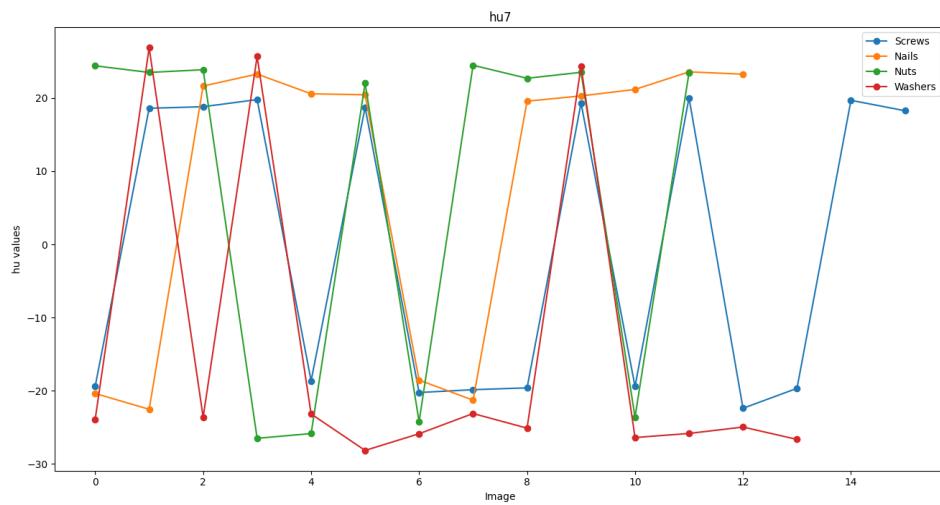
## Determinación de las características más relevantes de las imágenes

Para determinar la información más relevante se realizó un análisis gráfico de distintas características.









Luego de este análisis y como se puede observar en las gráficas se concluyó que los datos en donde más segmentados estaban cada tipo de elemento eran en altura/ancho, redondez, momento de hu 2 y momento de hu 4. A efectos prácticos se decidió utilizar solo uno de los momentos de hu, para poder visualizar, el conjunto de dicho momento con las otras relaciones como un punto en el espacio, que representaría cada imagen.

---

## Conclusiones

Luego de haberse realizado este trabajo, se puede decir que a la hora de realizar sistemas inteligentes con visión artificial, hay que tener en cuenta ciertos aspectos fundamentales.

En primer lugar las condiciones de uso, en particular las condiciones de luz del ambiente en donde se desea realizar los análisis y aplicar el sistema, como también la calidad de los sensores a utilizar. Si las condiciones de luz no son óptimas se debe mejorar la calidad de los sensores e intensificar el tratamiento de las imágenes para obtener mejores resultados.

En segundo lugar, el objetivo que se persigue a la hora de implementar un sistema de visión artificial, para este caso resultó útil utilizar imágenes en escala de grises y analizar solo sus bordes, pero puede ocurrir que necesitemos analizar los colores u otras características de las imágenes.

Respecto de los algoritmos de clasificación, en general demostraron ser muy eficientes, con el tratamiento correcto de las imágenes, y la selección adecuada de la información de las mismas, a pesar de utilizar pocas imágenes en la base de datos, en la mayoría de los casos clasifican de manera adecuada las piezas.

En cuanto a la medición de objetos, es importante conocer la relación de tamaño con la que captamos la imagen con nuestros sensores, en este caso la relación pixel/cm o cm/pixel.

Finalmente, queda destacar la importancia de desarrollar estos sistemas que nos permiten no sólo automatizar tareas como puede ser separar piezas metálicas, sino también detectar objetos en imágenes y realizar análisis que serían muy difíciles y requerirían de mucho entrenamiento para realizarlo solo con nuestra vista.

---

## Bibliografía

[https://aulaabierta.ingenieria.uncuyo.edu.ar/pluginfile.php/20301/mod\\_resource/content/8/Clase%201-%20Agentes%20Inteligentes.pdf](https://aulaabierta.ingenieria.uncuyo.edu.ar/pluginfile.php/20301/mod_resource/content/8/Clase%201-%20Agentes%20Inteligentes.pdf)

<https://www.redalyc.org/journal/5722/572262176018/html/#:~:text=Medicina%3A%20En%20el%20campo%20de,al%20lugar%20de%20la%20cirug%C3%ADa.>

<https://www.inesem.es/revistadigital/informatica-y-tics/opencv/>

[https://es.wikipedia.org/wiki/Visi%C3%B3n\\_artificial](https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial)

[https://www.tutorialspoint.com/python/python\\_gui\\_programming.htm](https://www.tutorialspoint.com/python/python_gui_programming.htm)

<https://www.youtube.com/watch?v=VgI6sNK06d0&t=1168s>

[https://docs.opencv.org/4.x/d6/d00/tutorial\\_py\\_root.html](https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html)