

**UNIVERSIDAD NACIONAL DE CUYO**  
**FACULTAD DE INGENIERÍA**

---

**Proyecto Final de Cátedra**

**Carrera: Ingeniería en Mecatrónica**  
**Cátedra: Inteligencia Artificial I**

**Trabajo Final Inteligencia Artificial I - año 2021**  
**Visión Artificial**

Por: **Javier Alejandro Llano López**

Docente Titular: **Selva Rivera**  
Jefe de Trabajos Prácticos: **Juan Ignacio García**

**Mendoza, Argentina**

3 de julio de 2024

---

## Índice

<b>1. Resumen</b>	<b>2</b>
<b>2. Introducción</b>	<b>2</b>
<b>3. Especificación del agente</b>	<b>2</b>
<b>4. Diseño del agente</b>	<b>3</b>
<b>5. Código</b>	<b>4</b>
<b>6. Ejemplo de aplicación</b>	<b>15</b>
<b>7. Resultados</b>	<b>18</b>
<b>8. Conclusiones</b>	<b>19</b>
<b>Referencias bibliográficas</b>	<b>20</b>

---

## 1. Resumen

El objetivo de este proyecto fue desarrollar un clasificador de imágenes para distinguir entre diferentes tipos de objetos mecánicos: arandelas, tuercas, tornillos y clavos. Si el objeto detectado era un tornillo o clavo, se debía obtener su medida real.

Se utilizaron los algoritmos KNN (K-Nearest Neighbors) y KMeans sin utilizar librerías especializadas, implementándolos desde cero en Python. Las imágenes fueron preprocesadas con filtros para extraer los momentos de Hu, que proporcionan características robustas de las formas de los objetos.

Los resultados demostraron una eficacia razonable en la clasificación de los objetos. Al utilizar la librería OpenCV, se pudo calcular con mayor facilidad las medidas de los clavos y tornillos detectados. Los experimentos concluyeron que la combinación de técnicas de preprocesamiento y clasificación sin librerías adicionales permite un rendimiento adecuado, aunque la implementación de optimizaciones y el uso de herramientas especializadas podrían mejorar aún más la precisión y eficiencia del sistema.

## 2. Introducción

En este proyecto se utilizó visión artificial. Esta es una rama de la inteligencia artificial que se centra en permitir a las máquinas interpretar y comprender el mundo visual a partir de imágenes digitales. Utiliza técnicas de procesamiento de imágenes y aprendizaje automático para extraer información significativa y tomar decisiones basadas en esta información. La visión artificial tiene aplicaciones en diversas áreas, como la automatización industrial, la medicina, la robótica, la vigilancia y la conducción autónoma, entre otras.

El problema abordado en este proyecto consistió en desarrollar un sistema de clasificación de imágenes capaz de distinguir entre diferentes tipos de objetos mecánicos: arandelas, tuercas, tornillos y clavos. Este sistema debía procesar las imágenes para extraer características relevantes y luego utilizar algoritmos de clasificación para identificar correctamente cada objeto. Adicionalmente, si el objeto detectado era un clavo o un tornillo, el sistema debía medir la longitud real del objeto utilizando técnicas de visión por computadora.

El desafío residió en implementar los algoritmos de clasificación KNN (K-Nearest Neighbors) y KMeans desde cero, sin el uso de librerías especializadas, para demostrar una comprensión profunda de estos métodos y su aplicabilidad en tareas de visión artificial. Las imágenes se preprocesaron mediante filtros y se analizaron utilizando los momentos de Hu para extraer características robustas de las formas de los objetos, lo que facilitó su clasificación. El éxito del proyecto se evaluó en función de la precisión y eficiencia del sistema para identificar y medir correctamente los objetos en las imágenes.

## 3. Especificación del agente

El agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores.

Este programa utilizó el **agentes que aprenden**. Los agentes que aprenden son aquellos que pueden mejorar su rendimiento a lo largo del tiempo a través de la experiencia. Estos agentes utilizan algoritmos de aprendizaje automático para analizar datos y extraer patrones que les permitan tomar decisiones más precisas en el futuro. Pueden aprender a partir de ejemplos, retroalimentación o interacción directa con el entorno.

Al utilizar una base de datos con múltiples imágenes de los objetos a clasificar, se logró que los algoritmos Knn y KMeans logren identificar con éxito los objetos mediante decisiones informadas a partir de la información aprendida en el entrenamiento.

Por otra parte, no podría haber sido un agente reactivo simple dado que este tipo de agente no tiene conocimiento previo sobre la meta ni un agente reactivo basado en modelos dado que no se tiene internamente un modelo matemático que le informe sobre el objetivo a conseguir. Tampoco podría haber sido un agente basado en objetivos porque no hay planteado un objetivo a largo plazo. Por último, un agente basado en utilidad no hubiese aplicado al problema porque este agente evalúa la "bondad" de las acciones posibles. Los algoritmos utilizados clasifican inmediatamente sin una evaluación de utilidades a largo plazo.

### Tabla REAS

La tabla REAS (Reglas de Estado-Acción-Resultado) es una representación que muestra cómo un agente toma decisiones basadas en sus percepciones y el estado del entorno.

En la tabla 1 se puede observar como quedó planteado el agente y su interacción con el entorno.

Agente	Medidas de rendimiento	Entorno	Actuadores	Sensores
Clasificador	Precisión de clasificación, exactitud de medición de tornillos y clavos	Base de datos de imágenes, nuevas imágenes a clasificar	Pantalla para mostrar los resultados	Cámara

Cuadro 1: Tabla REAS para el Clasificador de Imágenes

### Propiedades del entorno

En cuanto a las propiedades del entorno, se pudo concluir que las nuevas imágenes a clasificar son parte del entorno de trabajo, este es observable dado que se tiene el sensor necesario para obtener información del entorno, es determinista porque la clasificación para una misma imagen no cambia respecto a variables del entorno. Además, se observó un entorno episódico dado que cada clasificación individual no tiene relación con las otras, es discreto porque los estados están claramente definidos y son finitos, por último, el agente es individual ya que no hay otro que forma parte en la clasificación.

Todo esto puede observarse en la tabla 2

Entorno de trabajo	Observable o No Observable	Determinista o Estocástico	Episódico o Secuencial	Estático o Dinámico	Discreto o Continuo	Agente o Multiagente
Caja con fondo negro, soporte de cámara y lámpara	Totalmente observable	Determinista	Episódico	Estático	Discreto	Agente individual

Cuadro 2: Propiedades del entorno

## 4. Diseño del agente

Para el diseño del agente se delimitó por etapas que consiste en las siguientes:

- **Adquisición de datos:** Recopilación de imágenes de arandelas, tuercas, tornillos y clavos para construir la base de datos de entrenamiento.
- **Preprocesamiento de imágenes:** Aplicación de filtros y técnicas de procesamiento de imágenes para resaltar las características propias.
- **Extracción de características:** Cálculo de momentos de Hu a partir de las imágenes.
- **Entrenamiento del modelo:** Implementación y entrenamiento de los algoritmos utilizando las características extraídas.
- **Clasificación:** Predicción de categoría de la nueva imagen a partir del modelo previamente entrenado.
- **Cálculo de medidas:** Implementación de técnicas de visión artificial para obtener la longitud real del clavo o tornillo clasificado.
- **Evaluación del rendimiento:** Validar la efectividad del agente clasificador y realizar ajustes si es necesario.

### Planificación

Se propuso la siguiente planificación para terminar el proyecto con los objetivos cumplidos:

- **Semana 1-2:**

Adquisición de Datos.

Preprocesamiento de Imágenes.

Aprendizaje de los algoritmos Knn y Kmeans.

■ **Semana 3:**

- Cálculo de momentos de Hu para todas las imágenes.
- Desarrollo del Código Base para KNN y KMeans.
- Evaluación preliminar de la precisión del modelo.

■ **Semana 5-6:**

- Medición de Tornillos y Clavos.
- Análisis y ajustes finales en los modelos.
- Documentación del proyecto y redacción del informe final.

■ **Semana 7:**

- Revisión final del informe y corrección de posibles errores.

### Camino crítico

El camino crítico es la secuencia de tareas que determina el tiempo total del proyecto. Cualquier retraso en estas tareas afectará directamente la fecha de finalización del proyecto.

En este caso, se planteó que el camino crítico fuera:

1. Adquisición de Datos
2. Preprocesamiento de Imágenes
3. Extracción de Características
4. Entrenamiento del Modelo
5. Clasificación
6. Medición de Tornillos y Clavos
7. Evaluación del Rendimiento
8. Preparación del Informe

## 5. Código

A continuación se presenta el código del algoritmo KMeans:

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 import os
5 from medicion import *
6 from mpl_toolkits.mplot3d import Axes3D
7
8 class KMeansClustering:
9     def __init__(self, k=3):
10         self.k=k
11         self.centroides=None
12
13     @staticmethod
14     def distancia_euclidiana(punto_dato, centroides):
15         return np.sqrt(np.sum((centroides-punto_dato)**2, axis=1))
16     def fit(self, X, indices_seleccionados, iteraciones_max=200):
17         self.centroides = np.empty((self.k, 10))
18         #self.centroides=np.random.uniform(np.amin(X, axis=0), np.amax(X, axis=0))
19         self.centroides[0]=[2.19813262e-03, 8.31325680e-08, 1.54576883e-09, 3.84474836e-
20                         8.64757395e-18, 1.14845687e-13, 3.61535442e-18, 1.05852417e+
21                         8.77275960e-01, 1.00481841e+00]
22         self.centroides[1]=[ 4.61136120e-03, 1.43170724e-05,
5.92379493e-09, 2.86788554e-09,
```



```
23                                     -4.05912354e-19, -4.31901129e-12,
24                                     1.18137159e-17,  2.47867299e+00,
25                                     1.37026198e-01,  6.87215573e+00]
26             self.centroides[2]=[ 7.46113357e-03, 5.03785208e-05, 1.82905453e-09, 6.18684916e
27                                     2.06360068e-17, 4.34635459e-11, 2.70224947e-18, 7.95724466e
28                                     1.26312008e-01, 3.14797903e+01]
29             self.centroides[3]=[ 8.24795508e-04, 1.43647927e-10, 1.10900196e-12, 1.47910461e
30                                     1.88023351e-24, 1.67709379e-17, 2.30985853e-25, 1.00209644e
31                                     9.94098100e-01, 1.00619638e+00]
32             self.centroides = self.centroides[:, indices_seleccionados]
33             self.centroides = normalizar(self.centroides, datos_max, datos_min)
34             for _ in range(iteraciones_max):
35                 y=[]
36                 for punto_dato in X:
37                     distancias=KMeansClustering.distancia_euclidiana(punto_dato,
38 # ndice de la m nima componente (cluster al que pertenece
39                     numero_de_cluster=np.argmin(distancias)
40                     y.append(numero_de_cluster)

41             y=np.array(y) # Vector con lista de cluster correspondiente de cada punto_dato
42             indices_cluster=[]

43             for i in range(self.k):
44                 # Almacena como arrays los indices de los datos correspondiente a cada
45                 indices_cluster.append(np.argwhere(y==i))

46             centros_cluster=[]

47             for i, indices in enumerate(indices_cluster):
48                 if len(indices)==0: # No hay puntos asignados a ese cluster
49                     # Almacena el vector de caracter sticas correspondiente a cada cluster
50                     centros_cluster.append(self.centroides[i])
51                 else:
52                     # Almacena la media entre los puntos pertenecientes a indices
53                     centros_cluster.append(np.mean(X[indices], axis=0)[0])
54             # Calcula la mxima diferencia entre los dos vectores
55             if np.max(self.centroides-np.array(centros_cluster))<0.0001:
56                 print("Se lleg a la m nima diferencia")
57                 break
58             if _==iteraciones_max:
59                 print("Se lleg a la m xima iteraci n")
60                 self.centroides=np.array(centros_cluster)
61             else:
62                 self.centroides=np.array(centros_cluster)
63             return y

64 # Cargar im genes de base de datos
65 def cargar_imagenes(directorio):
66     imagenes = []
67     for archivo in os.listdir(directorio):
68         ruta = os.path.join(directorio, archivo)
69         thresh_, contorno=preprocesar_imagen(ruta, False, 1)
70         HuM=calcular_parametros_y_hu(thresh_, contorno)
71         if thresh_ is not None:
72             imagenes.append(HuM) # Agrega el vector de caracter sticas
73     return imagenes

74 def preprocesar_imagen(ruta, graficar, tipo):
75     img = cv2.imread(ruta)
```

```

82 # La convertimos a escala de grises.
83 imagen_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
84 # Filtro Gausiano.
85 imagen_gauss = cv2.GaussianBlur(imagen_gris, (5, 5), 0)
86 # Umbralizacion de la imagen.
87 if tipo==1 or tipo==2:           # Base de datos y objeto de referencia
88   _, imagen_binaria = cv2.threshold(imagen_gauss, 107, 255, cv2.THRESH_BINARY)
89 if tipo==3 or tipo==4:           # Objeto a clasificar y objeto a medir
90   _, imagen_binaria = cv2.threshold(imagen_gauss, 120, 255, cv2.THRESH_BINARY)

91 # Encontrar contornos
92 contours, _ = cv2.findContours(imagen_binaria, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
93 contour_max = max(contours, key=cv2.contourArea)

94 if graficar:
95   fig, axs = plt.subplots(1, 4, figsize=(10, 5))
96   fig.suptitle("Preprocesamiento de imágenes", fontsize=16)

97   axs[0].imshow(img)
98   axs[0].set_title('Imagen original')
99   axs[0].axis('off')

100  axs[1].imshow(imagen_gris)
101  axs[1].set_title('Imagen en grises')
102  axs[1].axis('off')

103  axs[2].imshow(imagen_gauss)
104  axs[2].set_title('Filtro Gausiano')
105  axs[2].axis('off')

106  axs[3].imshow(imagen_binaria, cmap='gray')
107  axs[3].set_title('Imagen binaria')
108  axs[3].axis('off')

109  plt.tight_layout()
110  plt.show()

111 if tipo == 1:
112   return imagen_binaria, contour_max
113 elif tipo == 2:
114   return img, imagen_binaria
115 elif tipo == 3:
116   return imagen_binaria, contour_max
117 elif tipo == 4:
118   return img, imagen_binaria

119
120
121
122
123
124
125
126
127
128
129 def calcular_parametros_y_hu(imagen, contorno_maximo):
130
131 # Aproximacion de contorno del objeto a figura geometrica.
132 epsilon = 0.001 * cv2.arcLength(contorno_maximo, True)
133 aproximacion = cv2.approxPolyDP(contorno_maximo, epsilon, True)

134 momentos = cv2.moments(imagen)
135 momentosHu = cv2.HuMoments(momentos)

136 A = cv2.contourArea(aproximacion)
137 P = cv2.arcLength(aproximacion, True)
138 # Calcular relacion de aspecto
139 x, y, w, h = cv2.boundingRect(contorno_maximo)

```

```

142 relacion_aspecto = float(w) / h
143
144 # Calcular circularidad
145 if P == 0:
146     circularidad = 0
147 else:
148     circularidad = (4 * np.pi * A) / (P * P)
149
150 # Calcular elongacion
151 if len(contorno_maximo) >= 5: # fitEllipse requiere al menos 5 puntos
152     ellipse = cv2.fitEllipse(contorno_maximo)
153     (a, b) = ellipse[1] # Obtener los ejes de la elipse
154     if a>b:
155         elongation = a / b
156     else:
157         elongation = b / a
158 else:
159     print("No hay suficientes puntos para la elongacion")
160     elongation = None # No se puede calcular la elongacion si no hay suficientes puntos
161
162 # Crear el vector de caracteristicas
163 if elongation is not None:
164     vector_caracteristicas = np.append(momentosHu, [relacion_aspecto, circularidad, elongation])
165 else:
166     vector_caracteristicas = np.append(momentosHu, [relacion_aspecto, circularidad])
167
168 return vector_caracteristicas
169
170 def normalizar(datos, maximo, minimo):
171     # Obtener los valores minimos y maximos de cada caracteristica en X
172     return (datos-minimo)/(maximo-minimo)
173
174 # Directorio que contiene las imagenes de tuercas
175 directorio_tuercas = "Tuercas/"
176 tuercas = cargar_imagenes(directorio_tuercas)
177 # Directorio que contiene las imagenes de tornillos
178 directorio_tornillos = "Tornillos/"
179 tornillos = cargar_imagenes(directorio_tornillos)
180 # Directorio que contiene las imagenes de clavos
181 directorio_clavos = "Clavos/"
182 clavos = cargar_imagenes(directorio_clavos)
183 # Directorio que contiene las imagenes de arandelas
184 directorio_arandelas = "Arandelas/"
185 arandelas = cargar_imagenes(directorio_arandelas)
186
187 datos = []
188 datos = np.concatenate((tuercas, tornillos, clavos, arandelas), axis=0)
189 indices_seleccionados=[8, 1, 5, 9]
190 datos = datos[:, indices_seleccionados]
191 datos_max=np.max(datos, axis=0)
192 datos_min=np.min(datos, axis=0)
193 datos = normalizar(datos, datos_max, datos_min)
194 kmeans=KMeansClustering(k=4)
195
196 etiquetas=kmeans.fit(datos, indices_seleccionados)
197
198 # Cargar la imagen a clasificar
199 imagen_path="arandela.jpg"
200 thresh_, contorno= preprocesar_imagen(imagen_path, True, 3)
201 dato_analizar=calcular_parametros_y_hu(thresh_, contorno)

```

```

202 dato_analizar = dato_analizar[indices_seleccionados]
203 dato_analizar = normalizar(dato_analizar, datos_max, datos_min)
204 # Calcular la distancia de la imagen a los centroides
205 distancias = kmeans.distancia_euclidiana(dato_analizar, kmeans.centroides)
206 # Obtener la etiqueta del centroide más cercano (cluster)
207 etiqueta_predicha = np.argmin(distancias)
208 # Mapear la etiqueta predicha a la categoría correspondiente
209 categorias = ["Tuerca", "Tornillo", "Clavo", "Arandela"]
210 categoria_predicha = categorias[etiqueta_predicha]
211
212 print("La imagen pertenece a la categoría:", categoria_predicha)
213
214 # Medida de referencia
215 referencia_path="referencia.jpg"
216 referencia_original, referencia_filtrada = preprocesar_imagen(referencia_path, False, 2)
217 referencia_original = cv2.resize(referencia_original, (300, 250))
218 referencia_filtrada = cv2.resize(referencia_filtrada, (300, 250))
219 objeto_original, objeto_filtrado = preprocesar_imagen(imagen_path, False, 4)
220 objeto_original = cv2.resize(objeto_original, (300, 250))
221 objeto_filtrado = cv2.resize(objeto_filtrado, (300, 250))
222 # Calcular medida de clavo o tornillo
223 if (categoria_predicha==categorias[1] or categoria_predicha==categorias[2]):
224     height, width = medicion(referencia_original, referencia_filtrada, objeto_original,
225     if height>width:
226         medida = height
227     else:
228         medida = width
229     print("El", categoria_predicha, "mide:", medida, "cm.")
230
231 # Graficar
232 fig = plt.figure(figsize=(10, 8))
233 ax = fig.add_subplot(111, projection='3d')
234
235 colores = [ 'r', 'g', 'b', 'y']
236 for i in range(kmeans.k):
237     ax.scatter(datos[etiquetas == i, 0], datos[etiquetas == i, 1], datos[etiquetas == i,
238
239 ax.scatter(kmeans.centroides[:, 0], kmeans.centroides[:, 1], kmeans.centroides[:, 2], c=
240 ax.scatter(dato_analizar[0], dato_analizar[1], dato_analizar[2], c='purple', marker='o',
241
242 ax.set_xlabel('Circularidad')
243 ax.set_ylabel('2do Momento de Hu')
244 ax.set_zlabel('6to Momento de Hu')
245 ax.set_title('Gráfico 3D de los datos y centroides')
246 ax.legend()
247
248 plt.show()

```

El algoritmo Knn viene dado por:

```

1 import numpy as np
2 import cv2
3 import os
4 import matplotlib.pyplot as plt
5 from graficas import *
6 from medicion import *

7
8 # Definir una función para calcular la distancia Euclídea entre dos vectores
9 def distancia_euclidiana(x1, x2):
10     return np.sqrt(np.sum((x1 - x2) ** 2))

11 # Definir la clase del clasificador Knn

```



```
13 class Knn:
14     def __init__(self, k=3):
15         self.k = k
16
17     def fit(self, X, y):
18         self.X_train = X
19         self.y_train = y
20
21     def predecir(self, X):
22         n_samples = X.shape[0]
23         predicciones = np.zeros(n_samples, dtype=int)
24
25         for i in range(n_samples):
26             # Calcular la distancia Euclíadiana entre la muestra actual y todas las muestras de entrenamiento
27             distancias = np.array([distancia_euclidiana(X[i], x_train) for x_train in self.X_train])
28
29             # Ordenar las distancias y obtener los índices de las k muestras más cercanas
30             indices = np.argsort(distancias)[:self.k]
31
32             # Obtener las etiquetas correspondientes de las k muestras más cercanas
33             k_etiquetas = [self.y_train[indice] for indice in indices]
34
35             # Elegir la etiqueta que aparece con más frecuencia entre las k muestras más cercanas
36             predicciones[i] = max(set(k_etiquetas), key=k_etiquetas.count)
37
38     return predicciones
39
40     def score(self, X, y):
41         predicciones = self.predecir(X)
42         exactitud = np.mean(predicciones == y)
43         return exactitud
44
45 # Crear matrices para las imágenes de entrenamiento y prueba procesadas
46 def cargar_imagenes_y_etiquetas(directorio, etiqueta):
47     imagenes = []
48     etiquetas = []
49     for archivo in os.listdir(directorio):
50         ruta = os.path.join(directorio, archivo)
51         thresh_, contorno=preprocesar_imagen(ruta, False, 1)
52         HuM=calcular_parametros_y_hu(thresh_, contorno)
53         if thresh_ is not None:
54             imagenes.append(HuM)
55             etiquetas.append(etiqueta)
56     return imagenes, etiquetas
57
58 def preprocesar_imagen(ruta, graficar, tipo):
59     img = cv2.imread(ruta)
60     # La convertimos a escala de grises.
61     imagen_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
62     # Filtro Gausiano.
63     imagen_gauss = cv2.GaussianBlur(imagen_gris, (5, 5), 0)
64     # Umbralización de la imagen.
65     if tipo==1 or tipo==2:          # Base de datos y objeto de referencia
66         _, imagen_binaria = cv2.threshold(imagen_gauss, 107, 255, cv2.THRESH_BINARY)
67     if tipo==3 or tipo==4:          # Objeto a clasificar y objeto a medir
68         _, imagen_binaria = cv2.threshold(imagen_gauss, 150, 255, cv2.THRESH_BINARY)
69
70     # Encontrar contornos
71     contours, _ = cv2.findContours(imagen_binaria, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
72     contour_max = max(contours, key=cv2.contourArea)
```

```

73
74  if graficar:
75      fig , axs = plt.subplots(1, 4, figsize=(10, 5))
76      fig . suptitle("Preprocesamiento de los genes", fontsize=16)
77
78      axs [0].imshow(img)
79      axs [0]. set _title('Imagen original')
80      axs [0]. axis('off')
81
82      axs [1].imshow(imagen_gris)
83      axs [1]. set _title('Imagen en grises')
84      axs [1]. axis('off')
85
86      axs [2].imshow(imagen_gauss)
87      axs [2]. set _title('Filtro Gaussiano')
88      axs [2]. axis('off')
89
90      axs [3].imshow(imagen_binaria , cmap='gray')
91      axs [3]. set _title('Imagen binaria')
92      axs [3]. axis('off')
93
94      plt .tight_layout()
95      plt .show()
96
97  if tipo == 1:
98      return imagen_binaria , contour_max
99  elif tipo == 2:
100     return img , imagen_binaria
101  elif tipo == 3:
102     return imagen_binaria , contour_max
103  elif tipo == 4:
104     return img , imagen_binaria
105
106 def calcular_parametros_y_hu(imagen , contorno_maximo):
107 # Aproximacion de contorno del objeto a figura geometrica .
108 epsilon = 0.001 * cv2.arcLength(contorno_maximo , True)
109 aproximacion = cv2.approxPolyDP(contorno_maximo , epsilon , True)
110
111 momentos = cv2.moments(imagen)
112 momentosHu = cv2.HuMoments(momentos)
113
114 A = cv2.contourArea(aproximacion)
115 P = cv2.arcLength(aproximacion , True)
116 # Calcular relacion de aspecto
117 x , y , w , h = cv2.boundingRect(contorno_maximo)
118 relacion_aspecto = float(w) / h
119
120 # Calcular circularidad
121 if P == 0:
122     circularidad = 0
123 else :
124     circularidad = (4 * np.pi * A) / (P * P)
125
126 # Calcular elongacion
127 if len(contorno_maximo) >= 5: # fitEllipse requiere al menos 5 puntos
128     ellipse = cv2.fitEllipse(contorno_maximo)
129     (a, b) = ellipse[1] # Obtener los ejes de la elipse
130     if a>b:
131         elongacion = a / b
132     else :
```



```
133     elongacion = b / a
134 else:
135     print("No hay suficientes puntos para la elongación")
136     elongacion = None # No se puede calcular la elongación si no hay suficientes p
137
138 # Crear el vector de características
139 if elongacion is not None:
140     vector_caracteristicas = np.append(momentosHu, [relacion_aspecto, circularidad,
141 else:
142     vector_caracteristicas = np.append(momentosHu, [relacion_aspecto, circularidad])
143
144 return vector_caracteristicas
145
146 def normalizar(datos, maximo, minimo):
147     # Obtener los valores máximos y mínimos de cada característica en X
148     return (datos - minimo) / (maximo - minimo)
149
150 # Directorio que contiene las imágenes de entrenamiento de tuercas
151 directorio_tuercas = "Tuercas_entrenamiento/"
152 tuercas_entrenamiento, etiquetas_tuercas_entrenamiento = cargar_imagenes_y_etiquetas(direct
153
154 # Directorio que contiene las imágenes de entrenamiento de tornillos
155 directorio_tornillos = "Tornillos_entrenamiento/"
156 tornillos_entrenamiento, etiquetas_tornillos_entrenamiento = cargar_imagenes_y_etiquetas(
157
158 # Directorio que contiene las imágenes de entrenamiento de clavos
159 directorio_clavos = "Clavos_entrenamiento/"
160 clavos_entrenamiento, etiquetas_clavos_entrenamiento = cargar_imagenes_y_etiquetas(direc
161
162 # Directorio que contiene las imágenes de entrenamiento de arandelas
163 directorio_arandelas = "Arandelas_entrenamiento/"
164 arandelas_entrenamiento, etiquetas_arandelas_entrenamiento = cargar_imagenes_y_etiquetas(
165
166 # Directorio que contiene las imágenes de validación de tuercas
167 directorio_tuercas = "Tuercas_validacion/"
168 tuercas_validacion, etiquetas_tuercas_validacion = cargar_imagenes_y_etiquetas(directori
169
170 # Directorio que contiene las imágenes de validación de tornillos
171 directorio_tornillos = "Tornillos_validacion/"
172 tornillos_validacion, etiquetas_tornillos_validacion = cargar_imagenes_y_etiquetas(direct
173
174 # Directorio que contiene las imágenes de validación de clavos
175 directorio_clavos = "Clavos_validacion/"
176 clavos_validacion, etiquetas_clavos_validacion = cargar_imagenes_y_etiquetas(directorio_
177
178 # Directorio que contiene las imágenes de validación de arandelas
179 directorio_arandelas = "Arandelas_validacion/"
180 arandelas_validacion, etiquetas_arandelas_validacion = cargar_imagenes_y_etiquetas(direct
181
182 # Directorio que contiene las imágenes de prueba de tuercas
183 directorio_tuercas = "Tuercas_prueba/"
184 tuercas_prueba, etiquetas_tuercas_prueba = cargar_imagenes_y_etiquetas(directorio_tuerca
185
186 # Directorio que contiene las imágenes de prueba de tornillos
187 directorio_tornillos = "Tornillos_prueba/"
188 tornillos_prueba, etiquetas_tornillos_prueba = cargar_imagenes_y_etiquetas(directorio_to
189
190 # Directorio que contiene las imágenes de prueba de clavos
191 directorio_clavos = "Clavos_prueba/"
192 clavos_prueba, etiquetas_clavos_prueba = cargar_imagenes_y_etiquetas(directorio_clavos,
```



```
193 # Directorio que contiene las imágenes de prueba de arandelas
194 arandelas_prueba, etiquetas_arandelas_prueba = cargar_imagenes_y_etiquetas(directorio_aran-
195 
196 # Concatenar todas las imágenes y etiquetas
197 x_entrenamiento = []
198 y_entrenamiento = []
199 x_validacion = []
200 y_validacion = []
201 x_prueba = []
202 y_prueba = []
203 
204 
205 x_entrenamiento = np.concatenate((tuercas_entrenamiento, tornillos_entrenamiento, clavos_
206 y_entrenamiento = np.concatenate((etiquetas_tuercas_entrenamiento, etiquetas_tornillos_entrenam-
207 
208 x_validacion = np.concatenate((tuercas_validacion, tornillos_validacion, clavos_validacion))
209 y_validacion = np.concatenate((etiquetas_tuercas_validacion, etiquetas_tornillos_validacion))
210 
211 x_prueba = np.concatenate((tuercas_prueba, tornillos_prueba, clavos_prueba, arandelas_prue-
212 y_prueba = np.concatenate((etiquetas_tuercas_prueba, etiquetas_tornillos_prueba, etiquetas_aran-
213 
214 # Entrenar el modelo Knn con los momentos de Hu seleccionados
215 modelo_seleccionado = Knn(k=4)
216 modelo_seleccionado.fit(x_entrenamiento, y_entrenamiento)
217 
218 # Ajustar hiperparámetros usando el conjunto de validación
219 mejor_precision_seleccionada = 0
220 mejor_k_seleccionado = None
221 for k in range(1, 10):
222     modelo_seleccionado.k = k
223     precision_validacion = modelo_seleccionado.score(x_validacion, y_validacion)
224     if precision_validacion > mejor_precision_seleccionada:
225         mejor_precision_seleccionada = precision_validacion
226         mejor_k_seleccionado = k
227 print("Mejor valor de k encontrado:", mejor_k_seleccionado)
228 print("Precisión en el conjunto de validación:", mejor_precision_seleccionada)
229 
230 # Entrenar el modelo Knn con el mejor valor de k y los momentos de Hu seleccionados
231 modelo_seleccionado.k = mejor_k_seleccionado
232 x_entrenamiento_total = np.concatenate((x_entrenamiento, x_validacion))
233 y_entrenamiento_total = np.concatenate((y_entrenamiento, y_validacion))
234 indices_seleccionados=[8, 1, 5, 9]
235 x_entrenamiento_total = x_entrenamiento_total[:, indices_seleccionados]
236 X_max=np.max(x_entrenamiento_total, axis=0)
237 X_min=np.min(x_entrenamiento_total, axis=0)
238 x_entrenamiento_total = normalizar(x_entrenamiento_total, X_max, X_min)
239 x_prueba = x_prueba[:, indices_seleccionados]
240 x_prueba = normalizar(x_prueba, X_max, X_min)
241 modelo_seleccionado.fit(x_entrenamiento_total, y_entrenamiento_total)
242 
243 # Hacer predicciones en los datos de prueba
244 predicciones_seleccionadas = modelo_seleccionado.predecir(x_prueba)
245 
246 # Cargar la imagen a clasificar
247 imagen_path="arandela.jpg"
248 thresh_, contorno= preprocesar_imagen(imagen_path, True, 3)
249 dato_analizar=calcular_parametros_y_hu(thresh_,contorno)
250 dato_analizar = dato_analizar[indices_seleccionados]
251 dato_analizar = normalizar(dato_analizar, X_max, X_min)
```

```

253 # Realizar la predicci n usando el modelo KNN con los momentos de Hu seleccionados
254 etiqueta_predicha_seleccionada = modelo_seleccionado.predict(np.array([dato_analizar]))
255
256 # Definir un diccionario para mapear las etiquetas a los nombres de las categor as
257 categorias = {
258     0: "Tuerca",
259     1: "Tornillo",
260     2: "Clavo",
261     3: "Arandela"
262 }
263
264 # Imprimir la etiqueta predicha
265 print("La imgen es:", categorias[etiqueta_predicha_seleccionada])
266
267 # Medida de referencia
268 referencia_path="referencial.jpg"
269 referencia_original, referencia_filtrada = preprocesar_imagen(referencia_path, False, 2)
270 referencia_original = cv2.resize(referencia_original, (300, 250))
271 referencia_filtrada = cv2.resize(referencia_filtrada, (300, 250))
272 imagen_original, imagen_filtrada = preprocesar_imagen(imagen_path, False, 4)
273 # Para medir correctamente la imagen
274 referencia_original = cv2.resize(referencia_original, (300, 250))
275 referencia_filtrada = cv2.resize(referencia_filtrada, (300, 250))
276 imagen_original = cv2.resize(imagen_original, (300, 250))
277 imagen_filtrada = cv2.resize(imagen_filtrada, (300, 250))
278
279 # Calcular medida de clavo o tornillo
280 if (etiqueta_predicha_seleccionada==1 or etiqueta_predicha_seleccionada==2):
281     height, width = medicion(referencia_original, referencia_filtrada, imagen_original,
282     if height>width:
283         medida = height
284     else:
285         medida = width
286     print("El ", categorias[etiqueta_predicha_seleccionada], " mide: ", medida, " cm.")
287
288 # Evaluar el modelo con los momentos de Hu seleccionados
289 precision_seleccionada = modelo_seleccionado.score(x_prueba, y_prueba)
290 print("La precision del modelo con momentos de Hu seleccionados es:", precision_seleccionada)
291
292 # Calcular matriz de confusi n
293 matriz_confusion = np.zeros((4, 4), dtype=int)
294 for y_verdadero, y_predicho in zip(y_prueba, predicciones_seleccionadas):
295     matriz_confusion[y_verdadero, y_predicho] += 1
296
297 # Graficar matriz de confusi n
298 graficar_matriz_confusion(matriz_confusion)
299
300 # Graficar los datos de entrenamiento
301 plot_data(x_entrenamiento_total, y_entrenamiento_total, categorias, dato_analizar)

```

En cuanto al código para realizar las gráficas del algoritmo Knن, se muestra aquí:

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from mpl_toolkits.mplot3d import Axes3D
4
5 def plot_data(X, y, categorias, nuevo_punto=None):
6     fig = plt.figure(figsize=(8, 6))
7     ax = fig.add_subplot(111, projection='3d')
8
9     for clase, etiqueta in categorias.items():
10        ax.scatter(X[y == clase, 0], X[y == clase, 1], X[y == clase, 2], label=f'{etiqueta}')

```

```

11     if nuevo_punto is not None:
12         ax.scatter(nuevo_punto[0], nuevo_punto[1], nuevo_punto[2], color='r', s=100, lab
13
14
15         ax.set_xlabel('Circularidad')
16         ax.set_ylabel('2do Momento de Hu')
17         ax.set_zlabel('6to Momento de Hu')
18         ax.set_title('Gráfica de dispersión de los datos en 3D')
19         ax.legend()
20         plt.show()
21
22 def graficar_matriz_confusion(matriz_confusion):
23     plt.imshow(matriz_confusion, cmap='Blues', interpolation='nearest')
24     plt.colorbar()
25     plt.xticks(np.arange(4), ["Tuercas", "Tornillos", "Clavos", "Arandelas"], rotation=45)
26     plt.yticks(np.arange(4), ["Tuercas", "Tornillos", "Clavos", "Arandelas"])
27     plt.xlabel("Etiqueta Predicha")
28     plt.ylabel("Etiqueta Verdadera")
29     plt.title("Matriz de Confusión")
30     for i in range(4):
31         for j in range(4):
32             plt.text(j, i, str(matriz_confusion[i, j]), horizontalalignment="center", color="w")
33     plt.show()

```

El código para realizar la medición de tornillos y clavos es el siguiente:

```

1 from scipy.spatial.distance import euclidean
2 from imutils import perspective
3 import numpy as np
4 import imutils
5 import cv2
6 import matplotlib.pyplot as plt
7
8 def mostrar_imagenes(referencia, objeto):
9     fig, axs = plt.subplots(1, 2, figsize=(10, 5))
10    fig.suptitle("Imagenes medidas", fontsize=16)
11    axs[0].imshow(referencia)
12    axs[0].set_title('Imagen de referencia')
13    axs[0].axis('off')
14    axs[1].imshow(objeto)
15    axs[1].set_title('Objeto a medir')
16    axs[1].axis('off')
17
18    plt.tight_layout()
19
20
21    plt.show()
22
23 # Procesar imagen y encontrar contornos
24 def obtener_contornos(imagen):
25     contorno = cv2.findContours(imagen.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
26     contorno = imutils.grab_contours(contorno)
27     contorno = [x for x in contorno if cv2.contourArea(x) > 100]
28     return contorno
29
30 # Procesar la imagen del objeto de referencia
31 def medicion(referencia_original, referencia_filtrada, objeto_original, objeto_filtrado):
32     ref_contornos = obtener_contornos(referencia_filtrada)
33     # Se asume que el objeto de referencia tiene el contorno más grande de la imagen
34     ref_objeto = max(ref_contornos, key=cv2.contourArea)
35     ref_caja = cv2.minAreaRect(ref_objeto)
36     ref_caja = cv2.boxPoints(ref_caja)

```

```

37 ref_caja = np.array(ref_caja, dtype="int")
38 ref_caja = perspective.order_points(ref_caja)
39 (tl, tr, br, bl) = ref_caja
40 dist_en_cm = 3.5
41 dist_en_pixel = euclidean(tl, tr)
42 pixel_por_cm = dist_en_pixel / dist_en_cm
43 # Contorno m s grande
44 cv2.drawContours(referencia_original, [ref_caja.astype("int")], -1, (0, 0, 255), 2)
45 punto_medio_horizontal = (tl[0] + int(abs(tr[0] - tl[0]) / 2), tl[1] + int(abs(tr[1] - tl[1]) / 2))
46 punto_medio_vertical = (tr[0] + int(abs(br[0] - tl[0]) / 2), tr[1] + int(abs(br[1] - tl[1]) / 2))
47 ancho = euclidean(tl, tr) / pixel_por_cm
48 alto = euclidean(tr, br) / pixel_por_cm
49 cv2.putText(referencia_original, "{:.1f}cm".format(ancho),
50             (int(punto_medio_horizontal[0] - 15), int(punto_medio_horizontal[1] - 10)),
51             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
52 cv2.putText(referencia_original, "{:.1f}cm".format(alto),
53             (int(punto_medio_vertical[0] + 10), int(punto_medio_vertical[1])),
54             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)

55 # Procesar imagen de objeto a medir
56 objetos_contornos = obtener_contornos(objeto_filtrado)
57 largest_cnt = max(objectos_contornos, key=cv2.contourArea)
58 # Contorno m s grande
59 caja = cv2.minAreaRect(largest_cnt)
60 caja = cv2.boxPoints/caja)
61 caja = np.array(caja, dtype="int")
62 caja = perspective.order_points(caja)
63 (tl, tr, br, bl) = caja
64 cv2.drawContours(objeto_original, [caja.astype("int")], -1, (0, 0, 255), 2)
65 punto_medio_horizontal = (tl[0] + int(abs(tr[0] - tl[0]) / 2), tl[1] + int(abs(tr[1] - tl[1]) / 2))
66 punto_medio_vertical = (tr[0] + int(abs(br[0] - tl[0]) / 2), tr[1] + int(abs(br[1] - tl[1]) / 2))
67 ancho = euclidean(tl, tr) / pixel_por_cm
68 alto = euclidean(tr, br) / pixel_por_cm
69 cv2.putText(objeto_original, "{:.1f}cm".format(ancho),
70             (int(punto_medio_horizontal[0] - 15), int(punto_medio_horizontal[1] - 10)),
71             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
72 cv2.putText(objeto_original, "{:.1f}cm".format(alto),
73             (int(punto_medio_vertical[0] + 10), int(punto_medio_vertical[1])),
74             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)

75 mostrar_imagenes(referencia_original, objeto_original)

76
77 return alto, ancho

```

## 6. Ejemplo de aplicación

En este proyecto se utilizaron los algoritmos Knn y KMeans:

### Knn

KNN (K-Nearest Neighbors) es un algoritmo de aprendizaje supervisado utilizado principalmente para problemas de clasificación y regresión.

KNN clasifica un punto de datos nuevo basándose en la mayoría de los "vecinos" más cercanos en el espacio de características. Para ello, utiliza una métrica de distancia (generalmente la distancia euclídea) para identificar los K puntos de datos más cercanos en el conjunto de entrenamiento. El punto de datos nuevo se clasifica en la categoría más común entre sus K vecinos más cercanos.

### KMeans

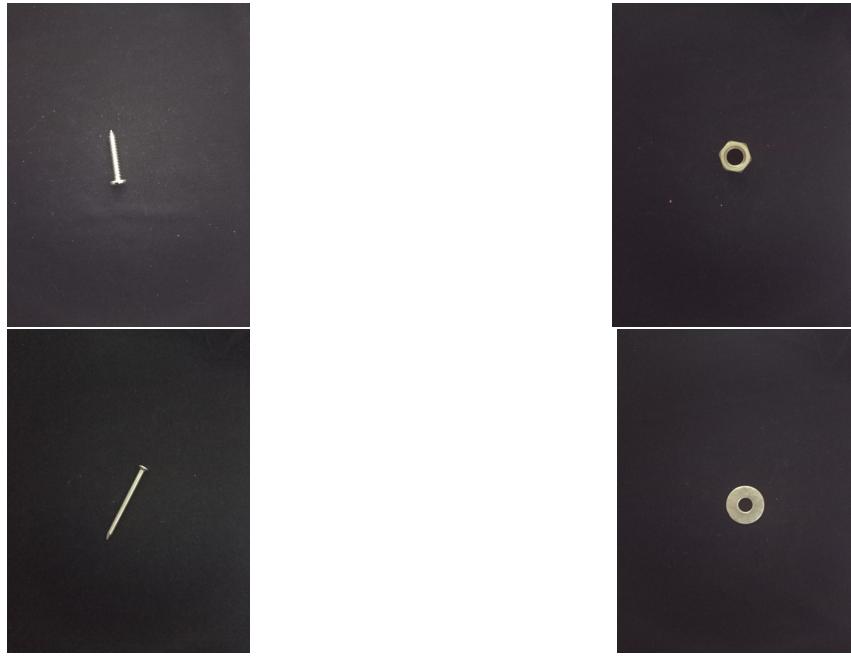
KMeans es un algoritmo de aprendizaje no supervisado que se utiliza para agrupar puntos de datos en K clústeres, donde K es un número predeterminado.

KMeans determina las posiciones aleatorias de los clústeres, calcula la distancia mínima de cada punto de datos

a cada clúster y los asigna al clúster correspondiente. Se actualizan las posiciones de los clústers y se recalcula la distancia mínima de cada punto de datos, de manera iterativa, hasta cierto número de iteraciones o hasta que las posiciones de los clústers hayan convergido a cierto valor.

Para la aplicación se utilizó una caja con fondo negro para que las imágenes contrasten y una tapa para evitar la luz de la sala y la luz de ambiente, no se utilizó el flash del celular porque ingresaba demasiado ruido a las imágenes. Además se utilizó una caja de celular para que las fotos se tomen siempre a la misma distancia.

A continuación se muestran ejemplos de las imágenes a clasificar.



Estas imágenes se preprocesaron para obtener las características más representativas que las distingan de las otras. En la figura 1 se muestran los filtros utilizados.

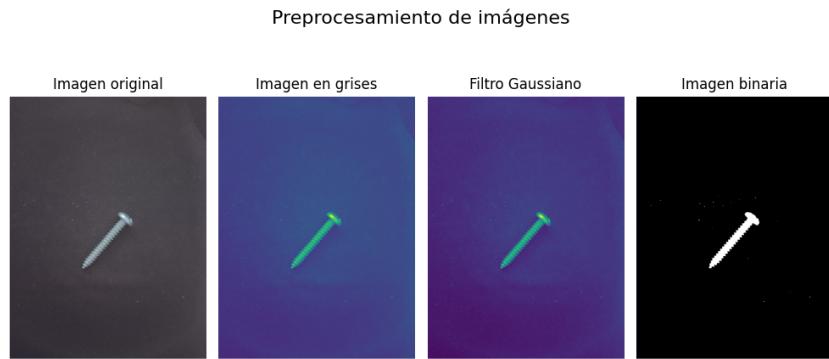


Figura 1: Filtros aplicados en una imagen de un tornillo

Una vez filtradas las imágenes, se obtuvieron los momentos de Hu que son momentos invariantes en una imagen y no varían respecto a la traslación, la rotación o el escalado de imagen.

También se obtuvieron la circularidad (cuán circular es el objeto en cuestión), la elongación (mide lo alargado del objeto) y la relación de aspecto (relación entre el ancho y alto de la imagen).

Luego, modificando la ruta de la imagen a clasificar en los códigos, se obtienen los resultados de los algoritmos por terminal.

Se llegó a la mínima diferencia  
La imagen pertenece a la categoría: Tuerca

Mejor valor de k encontrado: 1  
Precisión en el conjunto de validación: 1.0  
La imagen es: Tornillo  
El Tornillo mide: 3.9864215280443025 cm.  
La precisión del modelo con momentos de Hu seleccionados es: 1.0

Figura 2: Terminal de KMeans

Figura 3: Terminal de Knn

Las gráficas de los datos de dispersión para una tuerca (KMeans) y un tornillo (Knn) se muestran en las siguientes figuras:

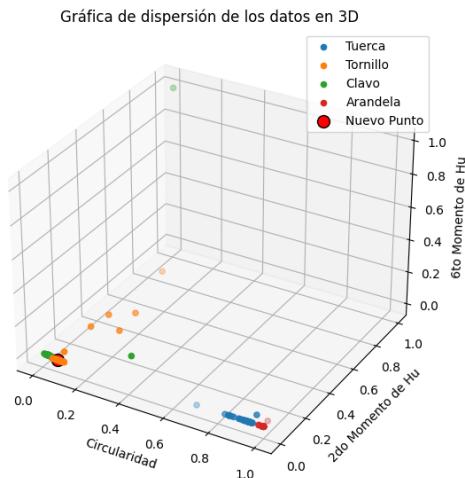


Figura 4: Dispersion 3D Knn

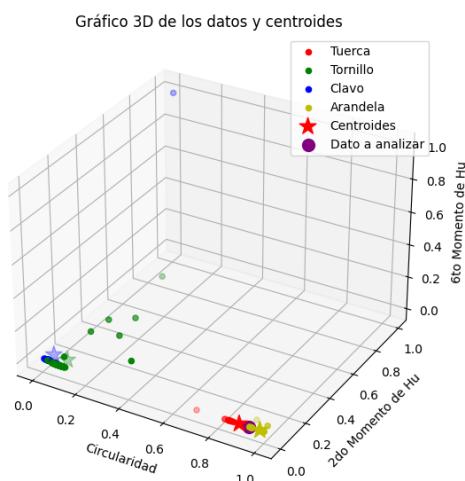


Figura 5: Dispersion 3D KMeans

En la figura 6 se muestra la medición de un tornillo.

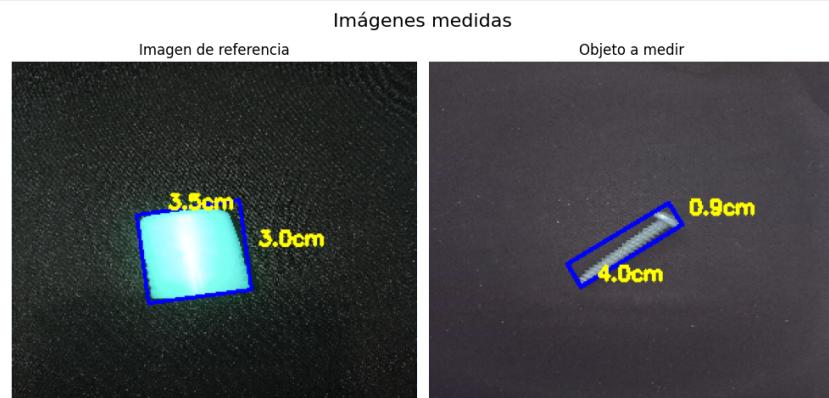


Figura 6: Medida de tornillo

Las mediciones se lograron utilizando la librería OpenCV y una imagen de referencia cuya longitud real de 3,5 centímetros ya se conocía previamente.

Por medio de esta librería se obtuvieron los contornos de los objetos y, mediante la relación de píxeles por centímetro, se obtuvieron las medidas reales.

Se puede apreciar que los objetos se clasificaron satisfactoriamente en los dos algoritmos.

## 7. Resultados

Para analizar el rendimiento de los algoritmos se pusieron a prueba con 20 imágenes, 5 de cada objeto.

Se logró la clasificación correcta de todas las imágenes por los dos algoritmos, una buena elección de parámetros para los filtros de preprocesamiento fue fundamental para lograr esto.

Se notó que en el algoritmo Knn el conjunto de prueba, intrínseco del algoritmo, dió una matriz de confusión ideal con todos los elementos en la diagonal principal (Figura 7). El algoritmo utilizó como mejor número de vecinos el 1, lo que denota que el sistema es sensible a los ruidos por lo que los datos tienen que ser homogéneos y bien etiquetados.

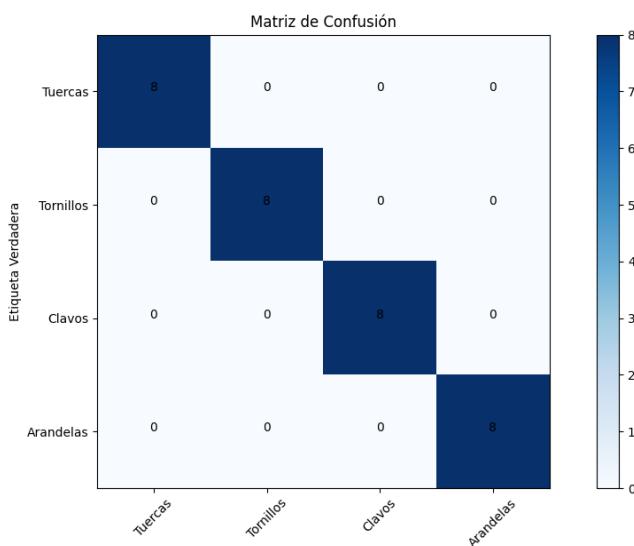


Figura 7: Matriz de confusión

Se modificó el algoritmo KMeans para que no posicione los centroides de manera aleatoria, sino que se les den posiciones iniciales fijas obtenidas por objetos de cada clase.

En cuanto a las mediciones de los clavos y tornillos, se redimensionaron las imágenes de referencia y del objeto a medir dado que experimentalmente se obtuvieron mejores medidas de esta manera.

---

Estas medidas tuvieron entre 6 y 8 milímetros de error, es decir, un error porcentual de entre un 14 % y un 30 % de la longitud real. Se concluye que el resultado fue aceptable para la aplicación.

Para mejorar este error se propuso sacar nuevamente la foto del objeto de referencia y aplicar un nuevo valor en el filtro binario, tanto de la imagen de referencia como del objeto a medir. Estas modificaciones lograron reducir el error a tan solo 1 a 2 milímetros, esto es un error porcentual de entre un 2.9 % y un 6.3 % de la longitud real.

## 8. Conclusiones

Para concluir, en este problema en particular, se optó por el algoritmo KNN en lugar de KMeans.

Esta decisión se basó en la superioridad de KNN en términos de rendimiento y precisión. Además, como se conocen los objetos a clasificar, se pueden determinar sus posiciones y ajustar el escenario para favorecer la clasificación. Por estas razones, se consideró que KNN es más útil que KMeans para este caso específico.

Como posibles mejoras al proyecto podría ser implementar una interfaz gráfica para navegar por el programa de clasificación, de esta manera la utilización se vuelve más apta para operarios. También, la medición de los objetos podría hacerse en tiempo real y no mediante una imagen. Como última mejora, se podrían fortalecer los algoritmos para poder clasificar correctamente los objetos en diferentes escenarios.

---

## Referencias bibliográficas

- [1] AssemblyAI. *How to implement KNN from scratch with Python*. URL: <https://www.youtube.com/watch?v=rTEtEy5o3X0&list=LL&index=8>.
- [2] Material de cátedra. IA I. 2024. URL: <https://aulaabierta.ingenieria.uncuyo.edu.ar/course/view.php?id=95&section=0#tabs-tree-start>.
- [3] Datacamp. *Clasificación K vecinos más próximos (KNN) con scikit-learn*. URL: <https://www.datacamp.com/es/tutorial/k-nearest-neighbor-classification-scikit-learn>.
- [4] DataSmarts. *Cómo Calcular Momentos de Hu en OpenCV*. URL: [https://www.youtube.com/watch?v=\\_Gp7elsyKA4](https://www.youtube.com/watch?v=_Gp7elsyKA4).
- [5] Documentación OpenCV. *Contornos*. URL: [https://docs.opencv.org/3.4.3/d9/d8b/tutorial\\_%20py\\_contours\\_hierarchy.html](https://docs.opencv.org/3.4.3/d9/d8b/tutorial_%20py_contours_hierarchy.html).
- [6] Documentación OpenCV. *Umbralización*. URL: <https://omes-va.com/simple-thresholding/>.
- [7] Gede Geyge Andika Lesmana, I Nyoman Piarsa, I Made Suwija Putra. “Identification of Baby’s Feet Using Principal Component Analysis (PCA) Method Character Extraction with K-Nearest Neighbor (KNN) Classification in Matlab Application”. En: *JURNAL ILMIAH MERPATI* (2021).
- [8] IBM. *¿Qué es Knn?* URL: <https://www.ibm.com/mx-es/topics/knn#main-content>.
- [9] Ifan Prihandi. “KNN on Iris Data with Python Programming”. En: *International Journal of Multidisciplinary Research and Publications* (2019).
- [10] Medium. *K-Means Clustering for Image Classification*. URL: [https://medium.com/@joel\\_34096/k-means-clustering-for-image-classification-a648f28bdc47](https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47).
- [11] Medium. *Understanding K-means Clustering and Image Compression*. URL: <https://medium.com/@dpatel32/understanding-k-means-clustering-and-image-compression-bdd26a1da267#:~:text=K-means%20clustering%20is%20a%20popular%20unsupervised%20learning%20algorithm,in%20an%20image%20while%20preserving%20its%20visual%20quality..>
- [12] NeuralNine. *K-Means Clustering From Scratch in Python (Mathematical)*. URL: <https://www.youtube.com/watch?v=5w5iUbTlpMQ&list=LL&index=9&t=1465s>.
- [13] NeuralNine. *K-Nearest Neighbors Classification From Scratch in Python (Mathematical)*. URL: [https://www.youtube.com/watch?v=xtaom\\_\\_-drE&list=LL&index=11&pp=gAQBiAQB8AUB](https://www.youtube.com/watch?v=xtaom__-drE&list=LL&index=11&pp=gAQBiAQB8AUB).
- [14] No Learning. *70. Implement the KNN algorithm from scratch*. URL: <https://www.youtube.com/watch?v=EzOM0y0d5JQ&list=LL&index=11&t=30s>.
- [15] Pyresearch. *Measure size of objects with image — Computer Vision — Opencv with Python — Pyresearch*. URL: <https://www.youtube.com/watch?v=4L5jF1Dkfxw>.
- [16] Stackoverflow. *How to get the real life size of an object from an image, when not knowing the distance between object and the camera?* URL: <https://stackoverflow.com/questions/9940140/how-to-get-the-real-life-size-of-an-object-from-an-image-when-not-knowing-the-d>.
- [17] Terrones Digital. *Python - OpenCV - Procesamiento de Imágenes*. URL: <https://www.youtube.com/watch?v=Bd1BTn6eNHQ>.