

# Documentación del Sistema de Reconocimiento de Cartas

## Introducción

Este documento describe de forma estructurada el sistema completo de **detección, segmentación y reconocimiento de cartas** desarrollado para el examen de Inteligencia Artificial. Incluye la explicación del pipeline, los archivos implementados, su función dentro del sistema y la lógica aplicada para conseguir un reconocimiento robusto incluso con imágenes imperfectas.

El sistema utiliza técnicas clásicas de visión por computadora basadas en **OpenCV**, permitiendo la identificación automática del **valor (rank)** y **palo (suit)** de cada carta detectada.

## Arquitectura General del Proyecto

```
examen parcial/
├── data/
│   ├── raw/           # Fotos originales
│   ├── templates/     # Plantillas por palo y número
│   └── processed/     # Warps y corners procesados
├── results/
│   └── detecciones/   # Resultados finales con bounding boxes
└── src/
    ├── main.py
    ├── segmentacion.py
    ├── generate_templates.py
    ├── reconocimiento.py
    ├── utils.py
    └── config.py
```

La estructura está diseñada para separar claramente: - **Datos originales** (raw) - **Plantillas** utilizadas para reconocimiento - **Datos procesados** - **Resultados finales** - **Código fuente**

## Flujo Completo del Sistema (Pipeline)

El proceso consta de cuatro etapas principales:

### Detección y segmentación

Archivo: **segmentacion.py**

1. Conversión a escala de grises.
2. Binarización para extraer contornos.
3. Filtrado de contornos pequeños.
4. Corrección de perspectiva (**warping**) para obtener cada carta de forma vertical y uniforme.
5. Recorte de una esquina específica (**corner**) que contiene el número y el palo.

Esta esquina es clave: se utiliza para el reconocimiento basado en plantillas.

### Generación de plantillas (templates)

Archivo: **generate\_templates.py**

El sistema permite crear una librería de plantillas extrayendo la esquina de cartas bien capturadas. El usuario:

- Visualiza el corner detectado.
- Introduce manualmente el **rank** y el **palo**.
- El script guarda los templates en `data/templates/palo/rank_XXX.png`.

Estas plantillas se convertirán en la referencia para reconocer cartas desconocidas.

### Reconocimiento automático

Archivo: **reconocimiento.py**

1. Se cargan todas las plantillas por palo.
2. Para cada carta detectada, se binariza su corner.
3. Se compara el corner con cada template mediante **template matching**.
4. Se obtiene un score y se selecciona la mejor coincidencia.
5. Si el score supera un threshold, la carta se marca como **UNKNOWN**.

### Generación de resultados

Archivo: **main.py**

1. Recorre todas las imágenes en data/raw.
2. Detecta y segmenta cada carta.
3. Reconoce su rank y palo.
4. Dibuja un bounding box y etiqueta sobre la imagen original.
5. Guarda el resultado en results/detecciones.

## Archivos Principales y su Función

### **config.py**

Define rutas del sistema y thresholds para el reconocimiento.

### **segmentacion.py**

Encargado de detectar cartas, warpear y extraer esquinas normalizadas.

### **generate\_templates.py**

Script interactivo para construir la librería de plantillas.

### **reconocimiento.py**

Implementa el reconocimiento mediante corner matching con plantillas.

### **utils.py**

Funciones de soporte: dibujar bounding boxes, garantizar directorios, guardar imágenes.

### **main.py**

Orquesta todo el pipeline y genera los resultados finales.

## Consideraciones Técnicas

### Por qué usar **corner matching**

- Más robusto que comparar la carta completa.
- Menos dependiente de iluminación o variaciones de color.
- Más rápido computacionalmente.
- Solo requiere plantillas pequeñas.

### Condiciones necesarias para buen reconocimiento

No se requieren fotos perfectas, pero sí: - Cartas **visibles y no solapadas**. - Iluminación razonable sin reflejos extremos. - La esquina superior izquierda debe estar presente.

## Limitaciones conocidas

- Cartas rotadas más de  $\sim 45^\circ$  pueden fallar.
- Sombras muy fuertes afectan la binarización.
- Si el corner queda fuera del warp, la carta será **UNKNOWN**.

## Resultado Final

El sistema permite:

- Procesar cualquier cantidad de imágenes.
- Detectar todas las cartas presentes.
- Reconocer valor y palo mediante plantillas.
- Guardar resultados visuales con etiquetas claras.

El diseño modular facilita extender el proyecto con: - Nuevos métodos de reconocimiento.  
- Clasificadores ML más avanzados. - Nuevos tipos de barajas.

## Conclusión

El proyecto implementa un pipeline completo y funcional para el reconocimiento de cartas usando técnicas de visión por computadora. Incluye detección, segmentación, extracción de características (corners), generación de plantillas, identificación mediante comparación y exportación de resultados.

La arquitectura modular y el uso de warps y corners normalizados aseguran un rendimiento robusto y estable, adecuado tanto para el examen como para futuras ampliaciones del sistema.