1. Hardware Utilizado

1.1 Descripción del Hardware

El sistema emplea un entorno controlado para maximizar la precisión:

- **Dispositivo de Captura:** Smartphone **Xiaomi Redmi Note 12** (transmisión de video en tiempo real).
- **Equipo de Procesamiento: ASUS TUF Gaming A15** (Linux, Python, GPU) para procesamiento de imágenes en tiempo real.
- **Elementos Críticos: Luz LED estable** para eliminar sombras y un **Tapete Verde** como fondo para una segmentación robusta.
- **Estabilización:** Un sostenedor de teléfono que mantiene la cámara fija en un ángulo casi perpendicular (80◦−90◦).

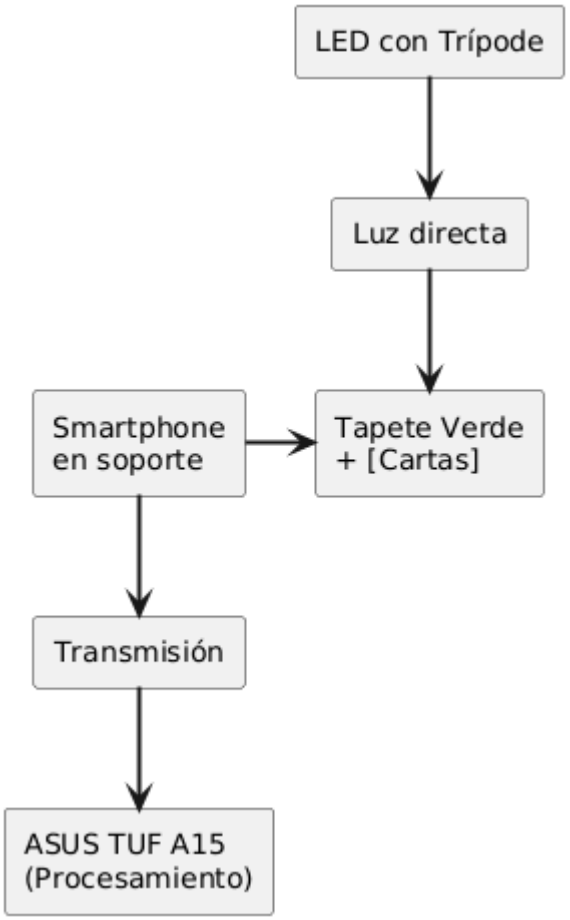
1.2 Justificación Técnica del Hardware

La selección de componentes se centró en el equilibrio entre **calidad, accesibilidad y rendimiento en tiempo real**:

- **Cámara (Smartphone):** Elegida por su **calidad de imagen** y **flexibilidad de posicionamiento**, superando webcams de menor calidad.
- **Computadora (ASUS TUF):** Cumple con los requisitos mínimos de CPU moderna, **GPU integrada** y **RAM** suficiente, asegurando un procesamiento sostenido de **15-30 FPS**.
- **Tapete Verde:** Esencial para la segmentación robusta. Su color facilita la separación en el espacio HSV: `lower_green = [35, 40, 40]`, `upper_green = [90, 255, 255]`. Esto aumenta la tasa de reconocimiento a > 90%.

1.3 Configuración del Entorno de Captura

Disposición espacial recomendada:



2. Software Utilizado

2.1 Sistema Operativo y Entorno de Desarrollo

- **SO: Ubuntu Linux** (20.04 LTS+) — Facilita el uso de Python y OpenCV.
- **IDE: Visual Studio Code** (VS Code).

2.2 Lenguaje de Programación y Versión

- **Lenguaje: Python 3.8 o superior** — Elegido por su ecosistema maduro para la visión por computadora y prototipado rápido.

2.3 Bibliotecas y Dependencias

- **OpenCV (4.8.0.74):** Núcleo del sistema. Se utiliza para captura, conversión de color, detección de contornos, transformación de perspectiva, *Template Matching* y análisis geométrico.
- **NumPy (1.24.3):** Fundamental para la computación científica y el **manejo eficiente** de arrays de imágenes.

2.4 Entorno Virtual (venv)

El uso de `venv` es una buena práctica para asegurar la **reproducibilidad** y el **aislamiento de dependencias**.

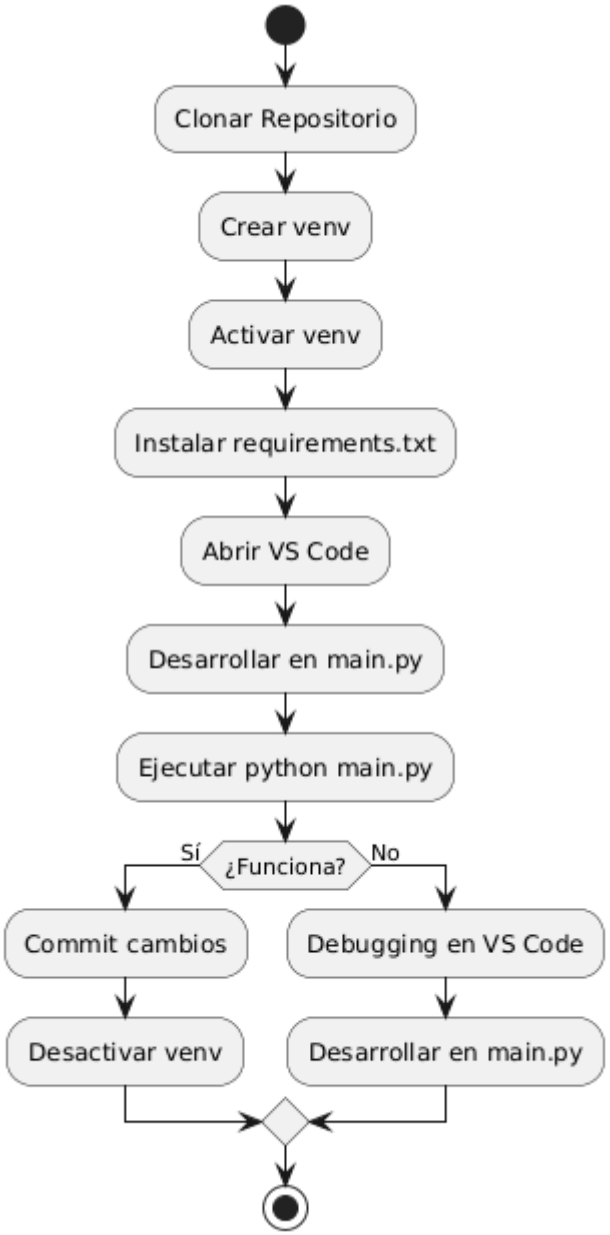
2.5 Justificación del Uso de Entorno Virtual (venv)

El **venv** **fue seleccionado** por ser **ligero, simple** y estar integrado en Python 3.3+, siendo suficiente para la gestión de librerías Python necesarias. Se descartaron alternativas más pesadas como Conda.

2.6 Estructura del Proyecto

```
examen_parcial/
├── venv/
├── templates/
├── captures/
├── main.py
├── utils.py
├── recognition_rank.py
├── recognition_suits.py
├── requirements.txt
├── explicacion.md
└── README.md
```

2.7 Workflow de Desarrollo



2.8 Comandos Esenciales

Se listan los comandos esenciales para la configuración (`python3 -m venv venv`, `source venv/bin/activate`, `pip install -r requirements.txt`) y ejecución (`python main.py`).

2.9 Resumen de Especificaciones Técnicas

Componente	Tecnología	Versión	Propósito
SO	Ubuntu Linux	20.04+	Sistema operativo base
IDE	Visual Studio Code	Latest	Desarrollo y debugging
Lenguaje	Python	3.8 - 3.11	Lenguaje de programación
Visión	OpenCV	4.8.0.74	Procesamiento de imágenes
Cálculo	NumPy	1.24.3	Operaciones numéricas

3. Hoja de Ruta del Desarrollo

El desarrollo se realizó en **7 fases** (~8 días total), siguiendo una metodología de **Prototipado Rápido Iterativo**.

3.2 Fase 1: Configuración Inicial del Entorno

- Se instaló el entorno de desarrollo y se **calibró el hardware** (luz, sostenedor, tapete) para minimizar sombras y asegurar el contraste.

3.3 Fase 2: Implementación del Sistema Base

- Se creó la arquitectura principal (`utils.py`, `main.py`). Se crearon y normalizaron los **templates de rango y palo**.

3.4 Fase 3: Primeras Pruebas y Problemas Detectados

- El *Template Matching* simple generó **alta confusión visual** (ej., 3, 5, 8). Se decidió migrar a **detectores especializados basados en geometría**.

3.5 Fase 4: Iteraciones de Mejora - Reconocimiento de Rangos

Se implementaron detectores especializados para los rangos más problemáticos:

- **Rango 8:** Detección de **dos agujeros internos** (`significant_components == 2`). **Mejora:** +45%.
- **Rango 6:** Detección de **un solo agujero cerrado**. **Mejora:** +53%.
- **Otros:** El resto de rangos se resuelven mediante *Template Matching* con votación múltiple.
- **Precisión final: 88%-95%** para los rangos críticos.

3.6 Fase 5: Iteraciones de Mejora - Reconocimiento de Palos

- **Problema:** Confusión entre palos negros (♠ vs ♣).
- **Solución: Sistema de Votación Múltiple** combinando color, *Template Matching* y métricas geométricas.
- **Métrica Clave: Solidez** (Pica > 0.82; Trébol < 0.78).
- **Precisión final: 87%-90%**.

3.7 Fase 6: Optimización y Sistema de Debug

- Se mejoró la eficiencia implementando una **caché de *templates*** y **extracción dinámica de ROIs**.
- Se creó un sistema de *debugging* condicional y un sistema de captura de *frames*.

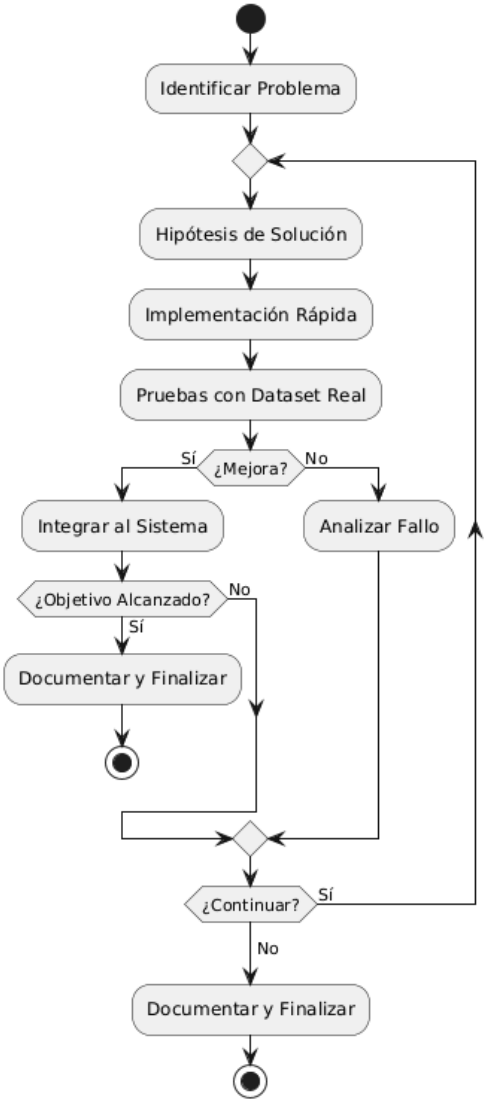
3.8 Fase 7: Pruebas Finales y Ajustes

- **Resultados Finales:** Precisión de Rangos **92.5%** y Palos **91.5%**. Rendimiento **22-28 FPS**.
- **Limitaciones:** La precisión baja ante sobreexposición, cartas dobladas o múltiples cartas superpuestas.

3.9 Lecciones Aprendidas

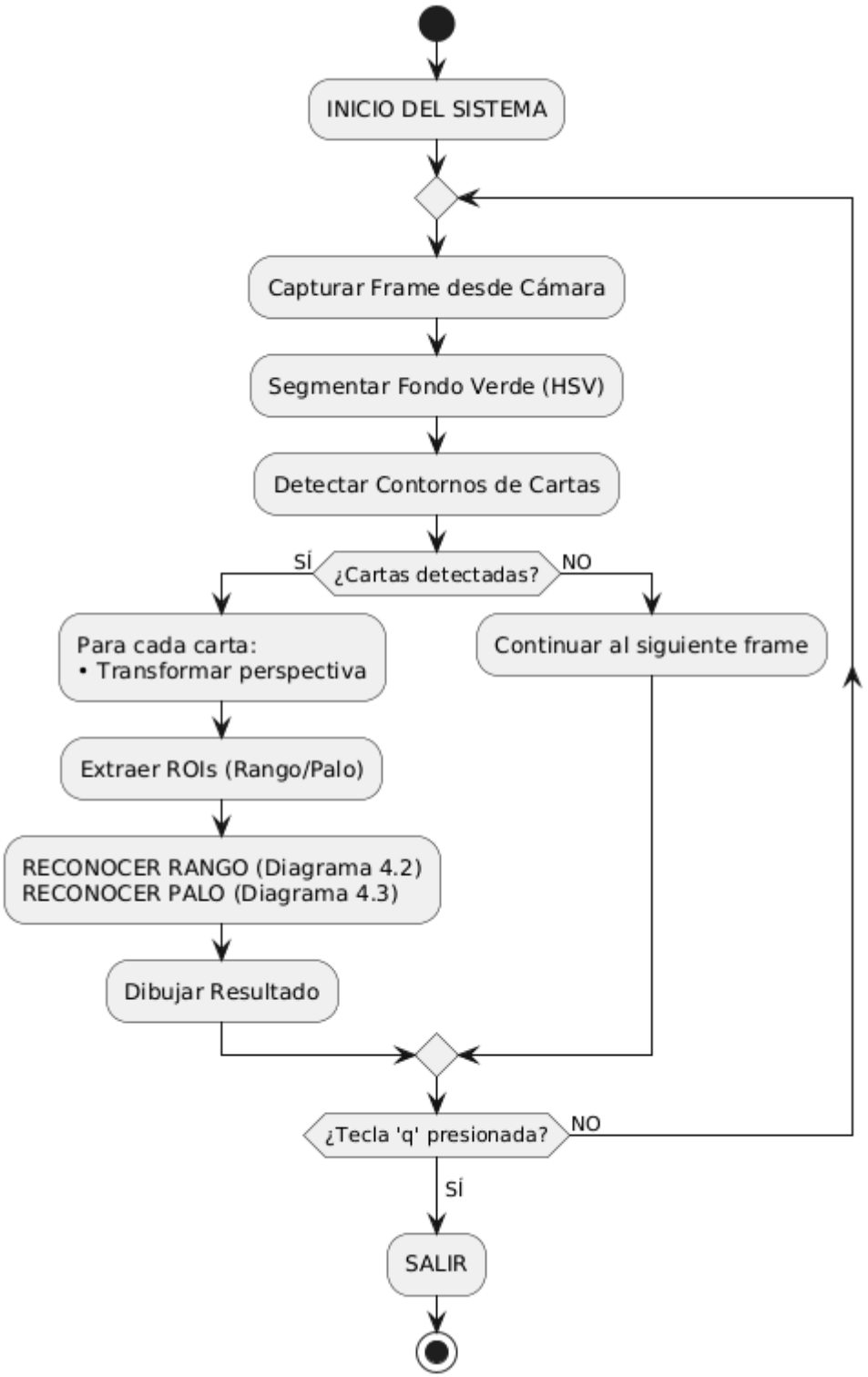
- El ***Template Matching* no es suficiente**; el enfoque híbrido es esencial.
- **HSV es superior a RGB** para la segmentación de fondo.

3.10 Metodología de Desarrollo

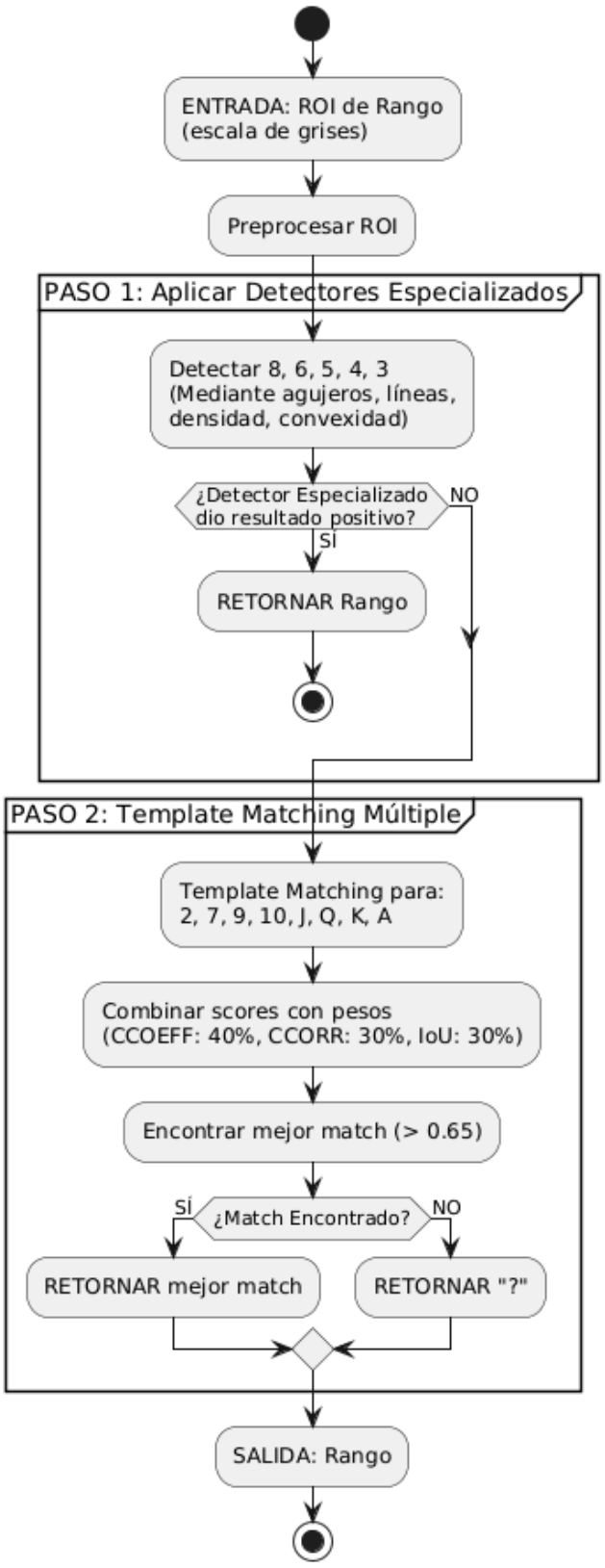


4. Diagramas de Decisión para Clasificación de Cartas

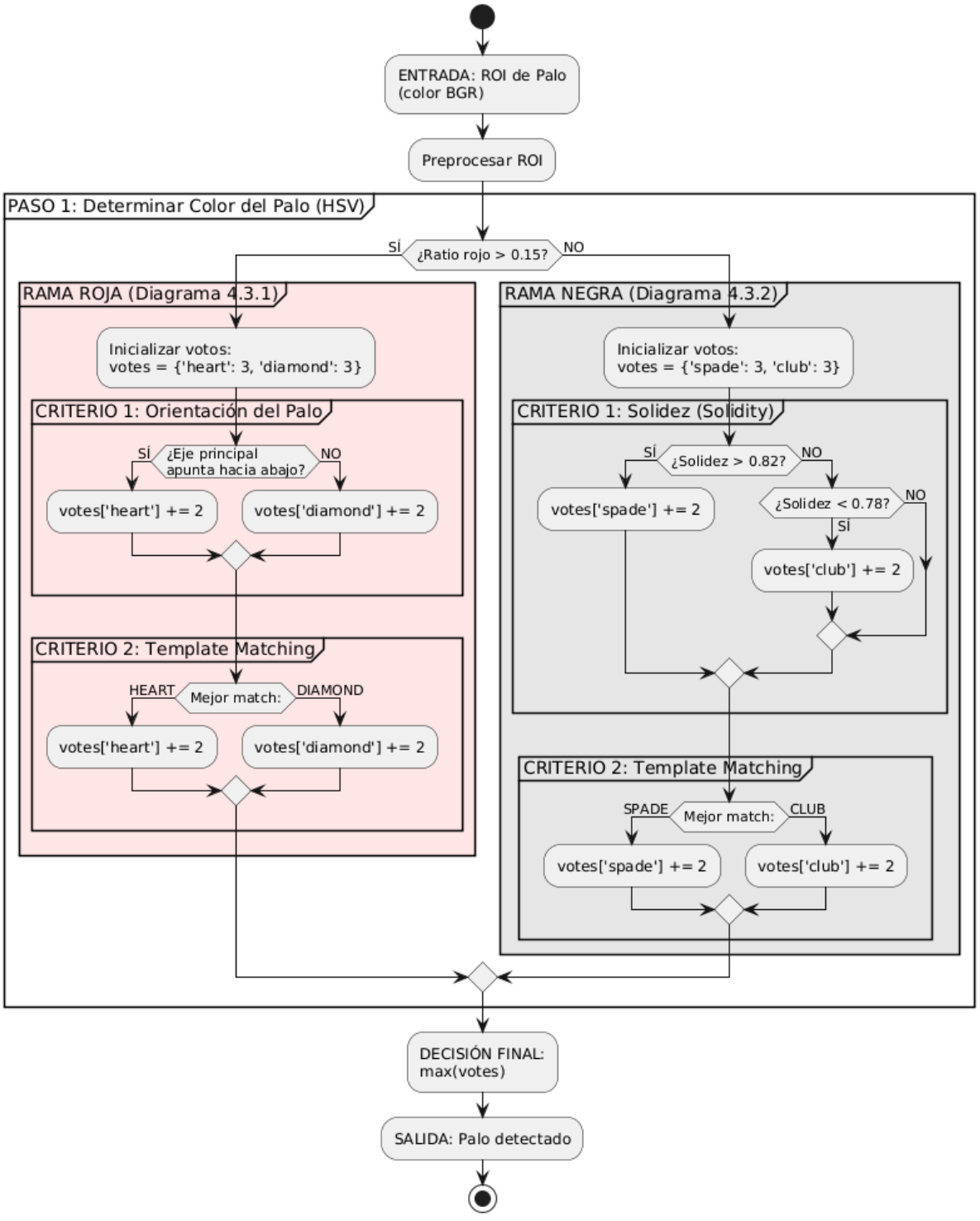
4.1 Diagrama de Flujo General del Sistema



4.2 Diagrama de Decisión: Reconocimiento de Rangos (Números/Letras)



4.3 Diagrama de Decisión: Reconocimiento de Palos



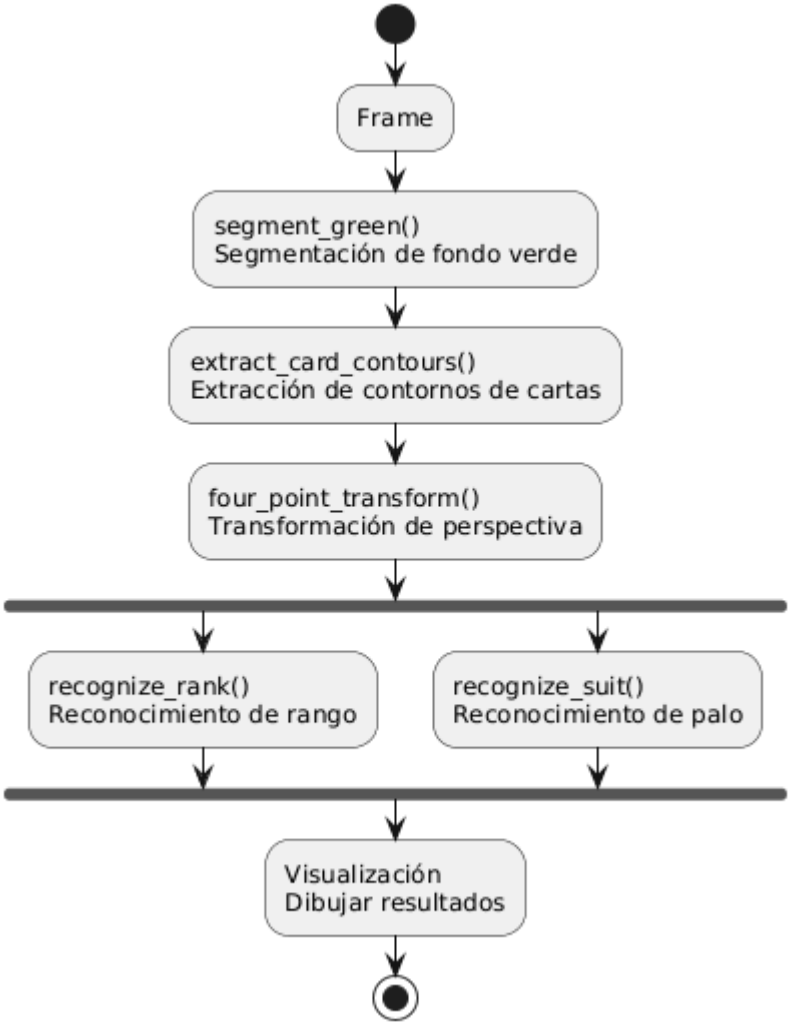
4.4 Tabla de Decisión Consolidada para Palos

Palo	Color	Solidez	Defectos	Componentes	Orientación	Votos Base
♠ Pica	Negro (+3)	>0.82 (+2)	≤1 (+1)	≤2 (+1)	Arriba (+1)	**8-10**
♣ Trébol	Negro (+3)	<0.78 (+2)	≥3 (+1)	>2 (+1)	N/A	**7-9**
♥ Corazón	Rojo (+3)	0.75-0.85	2-3	1	Abajo (+2)	**7-9**
♦ Diamante	Rojo (+3)	0.80-0.90	4	1	N/A (+2)	**7-9**

5. Secuencialización de Operaciones sobre Imágenes

5.1 Pipeline General

El flujo de procesamiento de una imagen sigue la siguiente secuencia continua:



5.2 Funciones Principales de Reconocimiento

`recognize_rank(warp)` - Reconocimiento de Números/Letras

- **Objetivo:** Identifica el rango de la carta (A, 2-10, J, Q, K) analizando la Región de Interés (**ROI**) de la esquina superior izquierda.
- **Proceso Clave:** Combina **Preprocesamiento** adaptativo (CLAHE + Otsu), una **Detección especial** para el "10" (buscando 2 componentes separados con `cv2.connectedComponentsWithStats()`), y **Template Matching con votación** (combinando scores de TM_CCOEFF_NORMED, TM_CCORR_NORMED y IoU).
- **Funciones Clave:** `cv2.createCLAHE()`, `cv2.threshold()`, `cv2.connectedComponentsWithStats()`, `cv2.matchTemplate()`.

`recognize_suit(warp)` - **Reconocimiento de Palos**

- **Proceso Clave:** Utiliza un **Sistema de Votación** donde la **Geometría** tiene el **90% del peso**:
  - **Detección de Color:** Clasifica primero como "red" o "black" analizando píxeles en **HSV** (Rangos rojos: [0−10◦] y [170−180◦] con `ratio > 0.15`).
  - **Análisis Geométrico:** Usa **Solidez** (Pica: >0.82; Trébol: <0.82) y **Defectos de Convexidad** para distinguir palos negros. Para palos rojos, utiliza orientación y vértices.
  - **Respaldo: Template Matching** con solo el 10% del peso.
- **Funciones Clave:** `cv2.cvtColor()` (BGR→HSV), `cv2.inRange()`, `cv2.convexHull()`, `cv2.convexityDefects()`.

5.3 Operaciones de Preprocesamiento (en `utils.py`)

Función	Propósito	Operaciones Clave	Salida
segment_green(frame)	Aislar las cartas del fondo verde.	Conversión **BGR → HSV**; aplicación de cv2.inRange() con rango del verde: $[35, 40, 40]$ a $[90, 255, 255]$; Operaciones morfológicas (MORPH_OPEN + MORPH_CLOSE).	Máscara binaria de cartas.
extract_card_contours(mask)	Detectar y validar formas rectangulares.	cv2.findContours() (RETR_EXTERNAL); Filtra por área mínima ($>2000 \text{ px}^2$); Aproxima a 4 vértices con cv2.approxPolyDP ($\text{epsilon} = 0.02 \times \text{perimeter}$).	Lista de contornos válidos.
four_point_transform(image, pts)	Corregir la perspectiva inclinada.	Ordena 4 puntos; Calcula matriz de transformación (cv2.getPerspectiveTransform()); Aplica corrección (cv2.warpPerspective()).	Carta normalizada ($300 \times 450 \text{ px}$).

5.4 Resumen de Parámetros Críticos

Función	Parámetro	Valor	Justificación
segment_green	Rango Hue	$[35-90^\circ]$	Captura verdes sin amarillos/azules.
	Kernel morfológico	5×5	Balance ruido vs detalles.
extract_card_contours	Área mínima	2000 px^2	Filtra ruido, conserva cartas.
recognize_rank	CLAHE clipLimit	3.0	Evita sobre-amplificación de ruido.
	Threshold match	0.50	Balance precisión/recall.
recognize_suit	Threshold rojo	$\text{ratio} > 0.15$	15% píxeles rojos = corazón/diamante.
	Solidez picas	> 0.82	Crítico para distinguir de tréboles.
	Solidez tréboles	< 0.82	Tienen espacios entre círculos.
	Peso geometría	90%	Más confiable que *Template Matching*.