UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DETECCIÓN Y RECONOCIMIENTO DE ELEMENTOS EN MESAS DE PÓKER ONLINE

Manuel Jorge Gómez Campo
Tutor: Álvaro García Martín
Ponente: Jose María Martínez Sánchez

Mayo 2018

DETECCIÓN Y RECONOCIMIENTO DE ELEMENTOS EN MESAS DE POKER ONLINE

AUTOR: Manuel Jorge Gómez Campo
TUTOR: Álvaro García Martín
PONENTE: Jose María Martínez Sánchez

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2018

Trabajo parcialmente financiado por el gobierno español bajo el proyecto TEC2014-53176-R (MobiNetVideo)

Resumen

A día de hoy, el área de la informática relacionada con la Visión Artificial (Computer Vision en inglés) está más de moda que nunca y está teniendo un gran impacto en áreas como la vigilancia, medicina y en otras no tan obvias como el ocio. Kinect o sistemas de realidad aumentada son un buen ejemplo del uso de esta tecnología para reconocer objetos en entornos reales.

En este trabajo se explora el caso concreto de detección y reconocimiento en interfaces de póker online, con el objetivo de obtener información de la partida de manera automática, partiendo de la misma información que tendría una persona jugando, las imágenes de la pantalla.

La elección del póker para este trabajo se basa en que es de los juegos de azar más extendidos y se utiliza en otras áreas de investigación como la Inteligencia Artificial.

En este documento se describe la implementación de este sistema, desarrollado en Python, haciendo uso principalmente de la librería de tratamiento de imágenes OpenCV, y diseñado para la variante de póker Texas hold'em, con el objetivo de funcionar en cualquier plataforma en tiempo real.

Con todo esto se pretende, por un lado, sacar a colación la duda de si realmente las plataformas de póker online (y otros juegos) pueden garantizar un juego justo, ya que un sistema como el implementado podría ser la base de un asistente o un bot difícilmente detectables. Y por otro lado es un intento de indagar en las capacidades del área de Computer Vision, en un escenario popular, y comprender mejor las buenas prácticas que se requieren en un proyecto de estas características, junto con los retos que lleva asociado.

Palabras clave (castellano)

Visión artificial, Poker, Texas hold'em, Python, OpenCV

Abstract

Nowadays the area of computer science related to Computer Vision is more fashionable than ever and is having a great impact in areas such as surveillance, medicine and others not as obvious as leisure. Kinect or augmented reality systems are a good example of the use of this technology to recognize objects in real environments.

This work explores the specific case of detection and recognition in online poker interfaces, with the aim of obtaining game information automatically, based on the same information a person would handle, the images on the screen.

The choice of poker for this work is based on the fact that it is one of the most widespread games of chance and is used in other areas of research such as Artificial Intelligence.

This document describes the implementation of this system, developed in Python, using mainly the image processing library OpenCV, and designed for the Texas hold'em poker variant, in order to work on any platform in real time.

With all this, this work seeks, on the one hand, to raise the question of whether online poker platforms (and other games) can guarantee a fair game, since a system like the one implemented could be the basis of an assistant or a bot hardly detectable. And on the other hand, it is an attempt to investigate the capabilities of Computer Vision, in a popular scenario, and to better understand the good practices that are required in a project of these characteristics, together with the associated challenges.

Keywords (inglés)

Computer Vision, Poker, Texas hold’em, Python, OpenCV

Agradecimientos

A mi tutor, Álvaro García, por darme la oportunidad de hacer este TFG y por el apoyo prestado.

A los profesores de la carrera, en especial a Fernando Maestre, cuyas clases estarán entre mis mejores recuerdos de la carrera.

Gracias Juan y David, el script que hicimos entre los tres (mentira, lo hizo casi todo Juan), que detectaba el color de un pixel en un juego muy chorra de minijuegos.com, y disparaba automáticamente, se quedó en mi subconsciente y fue una de las fuentes de inspiración para hacer este trabajo.

A mis amigos, y en especial a Jiawei, Jose, David, Jinle y Juan. Gracias de corazón por estar ahí, sin vosotros estos años no habrían sido lo mismo.

Gracias Laura, por aparecer en mi vida (ahora ya no tienes competencia).

A mis padres y a mi hermana, gracias por todo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

La motivación para realizar este trabajo de fin de grado vino de varios frentes.

Debo admitir que, a mi lado más tramposo le resultó bastante divertida la idea de usar Computer Vision (CV), para obtener información de una partida en curso. Y más teniendo en cuenta que no parecen existir proyectos parecidos, o al menos tan generales, ya que los bots de póker actuales basados en reconocimiento de imágenes suelen ser específicos de una plataforma y son sensibles a pequeños cambios en la interfaz.

Por contra, creo que no está de más concienciar un poco del riesgo al que se exponen los jugadores ajenos a estas trampas, en este caso en el póker, pero que también se aplica a otros juegos online.

Y, sobre todo, desde el punto de vista de proyecto de Visión Artificial, el póker con sus distintas plataformas ofrece un buen escenario para probar técnicas de detección y reconocimiento de objetos. Requiere un sistema capaz de abstraer bien la interfaz, robusto, pero a la vez flexible para funcionar en el mayor número de plataformas posible.

La unión de todo esto, hacen que sea un reto muy interesante.

1.2 Objetivos

El objetivo principal es crear un sistema capaz de detectar y reconocer elementos importantes y comunes a la mayor parte de mesas de póker online, en su variante Texas hold'em. Para ello, será necesario:

- Basarse en reconocimiento de imágenes. Estas podrían venir de capturas de pantalla de la interfaz del juego, vídeo, etc.
- Estudiar algoritmos de CV y la resolución de problemas de detección similares, entre otros. A la hora de la implementación, probar diferentes alternativas.
- Diseñar un sistema general, capaz de funcionar en la gran mayoría de plataformas.
- Desarrollar un código eficiente para poder ser ejecutado en tiempo real ya que los turnos de cada jugador suelen durar unos pocos segundos.

1.3 Organización de la memoria

En el primer capítulo, Estado del arte, se ofrece una visión del contexto general de este trabajo, pasando por las mecánicas básicas del Texas hold'em poker, y aproximaciones que se han realizado a este juego desde el área de la Visión Artificial.

Después, en Diseño se hace un repaso de las decisiones que se han tomado a más alto nivel, y de los principales retos que ha supuesto cada elemento, mientras que, en el siguiente capítulo, Desarrollo, se describen las soluciones implementadas en detalle.

En el capítulo de Integración, pruebas y resultados se describe el proceso de testeo que se ha llevado a cabo, y se analiza el comportamiento del sistema en cuanto a eficiencia y precisión para varios casos. Y finalmente, en Conclusiones y trabajo futuro se hace una reflexión del trabajo realizado y posibles mejoras.

2 Estado del arte

En primer lugar, se explicará brevemente en que consiste la variante de póker Texas hold'em, para la cual está diseñado el sistema. Una vez comprendido, se hace un breve análisis de usos de técnicas de reconocimiento y detección en escenarios similares, en comparación con lo que se propone en este trabajo.

2.1 Texas Hold'em

Para entender como se ha resuelto el problema de abstraer la interfaz es necesario tener unas nociones de las mecánicas básicas de este juego.

El Texas Hold'em es un tipo de póker muy popular donde se reparten dos cartas boca abajo a cada jugador y se comparten cinco cartas entre todos. A las dos cartas privadas de cada jugador se las suele denominar "hole cards" y a las cinco cartas comunes, "community cards" o cartas comunitarias. En este documento se hará referencia a ellas muchas veces como cartas del jugador y cartas de la mesa, respectivamente.

Al principio, de las cartas comunitarias, solo hay tres boca arriba (el flop), y las otras dos, el turn y el river están boca abajo.

Todos los jugadores pueden hacer sus apuestas por turnos antes de que se muestre el flop, despés del flop, después del turn y después del river. Al final gana la mano el que tiene la mejor combinación de cartas.

2.2 Póker y Computer Vision

En varias ocasiones se ha utilizado procesamiento de imágenes aplicado al póker, pero en general suele ser en entornos reales.

En cuanto a procedimientos, no es raro ver el uso de template matching [1] o de redes neuronales [2] para el reconocimiento de cartas, o también técnicas de segmentación, como segmentación por color para detectar fichas [3].

En estos casos donde lo que se analiza es real, es importante tener en cuenta factores como el ángulo desde el que se toman las imágenes u otros, que en el caso de las interfaces planas no tiene mucho sentido, aunque obviamente comparten algunas similitudes. En el caso que nos atañe, se presentan otros tipos de problemas principalmente relacionados con lo que varía la interfaz en cada plataforma.

Nuestra aproximación al problema se basará principalmente en aplicar técnicas de preprocesamiento y de detección de contornos utilizando funciones de la librería OpenCV seguido de una fase de filtrado y procesamiento.

3 Diseño

3.1 Introducción

En la fase de diseño, se ha dado especial importancia a dos factores: la modularidad y la eficiencia del sistema.

Esta modularidad es importante ya que se pretende que la detección de los elementos sea parcialmente independiente, de manera que si hay que modificar la forma en que se detectan las cartas, por ejemplo, esto no debería afectar a la parte del código responsable de detectar jugadores. Por estas razones, y porque encaja bien con el problema, se ha decido adoptar el paradigma de programación orientada a objetos.

En cuanto a la eficiencia del código, es un aspecto muy a tener en cuenta ya que uno de los objetivos es que pueda ser ejecutado en tiempo real. Con esto en mente, el funcionamiento del programa se divide en dos fases o etapas.

Una primera fase inicial, de detección/localización de los elementos, donde se ha enfocado la mayor parte del trabajo, y después una fase de reconocimiento, la cual permitiría actualizar cada cierto tiempo el estado de los elementos ya localizados.

Una vez atendidos estos requisitos, pasamos a lo que es el problema de Vision Artificial en sí. El principal reto es determinar qué es lo que hace reconocible a cada elemento, independientemente de la plataforma. Las siguientes imágenes ilustran bien el problema:

Figura 3-1: Comparación visual de 6 escenarios diferentes.

Como se puede ver en la figura 3-1, todas las interfaces comparten varios elementos comunes (cartas, jugadores, mesa…) pero muy diferentes en apariencia.

3.2 Estructura

En este apartado se muestra la estructura del proyecto, primero, mediante el diagrama de clases de la figura 3-2, donde se reflejan las clases que lo componen, sus atributos y operaciones principales, y las relaciones entre los objetos.

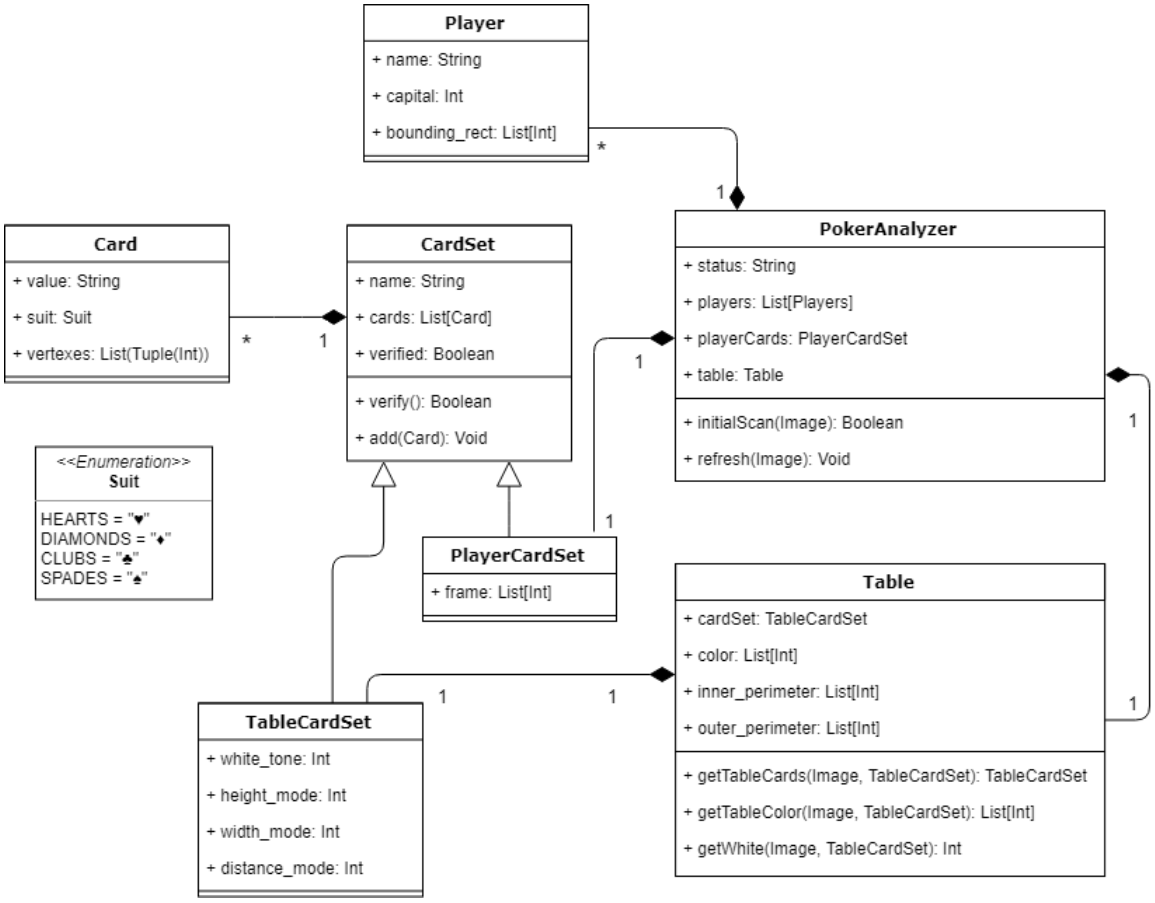

Figura 3-2: Diagrama de clases

En segundo lugar, una breve descripción de cada clase:

- Player: Representa a los jugadores, incluyendo al usuario que utiliza el sistema.
- Card: Representa una carta cualquiera. Almacena su valor y palo, y su posición, en forma de coordenadas del vértice superior izquierdo y vértice inferior derecho.
- Suit: Enumeración con los 4 palos (corazones, diamantes, tréboles y picas).
- CardSet: Clase abstracta que representa un conjunto de cartas. Es importante mencionar su campo verified, un booleano que está a True cuando se ha verificado la posición del conjunto de cartas.
- PlayerCardSet: Hereda de CardSet y representa las dos cartas del jugador.
- TableCardSet: Hereda de CardSet y representa las cartas comunes de la mesa.
- Table: Representa la mesa que suelen tener todas las interfaces, en cuyo centro suelen estar las cartas comunitarias y en cuyo perímetro suelen estar los jugadores.
- PokerAnalyzer: Es el núcleo del sistema, contiene de una forma u otra al resto de clases y es donde se va construyendo el contexto.

3.3 Detección

Uno de los requisitos más importantes del sistema es conseguir que funcione en el mayor número de plataformas posibles. Si bien es cierto que las interfaces de póker comparten muchas similitudes, también es cierto que hay multitud de detalles que complican tener un método de detección general, sobre todo, porque estos factores, como el color, la forma, elementos decorativos o la resolución de la imagen se combinan entre sí.

Por ello, un concepto clave a la hora de diseñar este sistema ha sido el de ir generando un contexto de la mesa poco a poco, de forma secuencial, en base a las primeras detecciones y a una serie de reglas que buscan abstraer los patrones más comunes en mesas de póker. Se trata de ir de lo más común y más fácil de detectar, a lo más complejo.

Dicho contexto se construye de la siguiente forma (diagrama del proceso en la figura 3-3):

1. En primer lugar, se realiza la detección de las cartas comunitarias, muy similares en todas las plataformas.

2. La detección de las cartas de la mesa nos aporta información del color, de las cartas del jugador suponiendo que sean iguales, pero también nos permite obtener el de la mesa. Esto facilita la tarea de detección de ambas, que por sí solas sería más complicado.

3. En este punto ya se tiene la posición y dimensiones de la mesa, facilitando la tarea de detectar a los jugadores ya que suelen estar dispuestos alrededor.

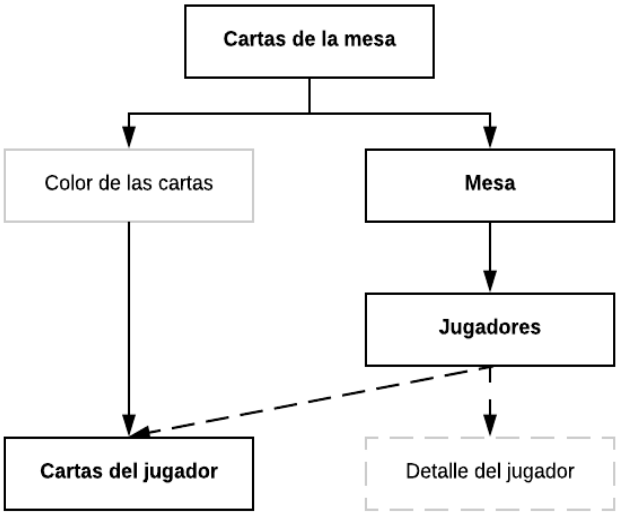

Figura 3-3: Construcción del contexto

3.3.1 Cartas de la mesa

En un principio, la detección de las cartas comunitarias puede parecer un problema sencillo. Básicamente consiste en buscar rectángulos que estén en un rango de color, área y ratio de altura/anchura. Sin embargo, una vez tenemos esos candidatos a cartas, aparecen algunos inconvenientes. El más inmediato es, ¿cómo diferenciar las cartas comunitarias de las del jugador y por ende descartar los falsos positivos?

Para ello la solución adoptada ha sido incluir un campo en el objeto TableCardSet que indica si la ubicación del objeto está verificada o no, y esta no se verifica hasta que los objetos detectados cumplan ciertos requisitos. En primer lugar, para evitar confundir las cartas de la mesa con las del jugador:

- Deben detectarse al menos 3. Recordamos que las cartas del jugador siempre son 2.

En segundo lugar, para evitar falsos positivos:

- Deben estar alineadas en el eje horizontal.
- Deben tener la misma altura y anchura.

Otra dificultad común en varias plataformas es el de la oclusión de las cartas, normalmente por fichas, aunque puede ser por otros elementos.

Figura 3-4: Ejemplo de oclusión en dos cartas

En la figura 3-4 se puede ver cómo las fichas tapan parte de las dos cartas de en medio, impidiendo la detección.

Para resolverlo, una vez se ha verificado la ubicación del set (como en el ejemplo de la figura), en siguientes iteraciones se completa la información relativa a la posición de las cartas que faltan mediante un algoritmo que se explica más en detalle en la sección de Desarrollo.

3.3.2 Mesa

Como tal, la posición de la mesa no le aporta nada al usuario. Pero sí que es interesante para construir el contexto que nos permite detectar a los jugadores, mucho más complicados de encontrar por si solos, debido a lo distintos que son en cada interfaz.

¿Pero cómo detectar la mesa? Aunque no tanta, también en las mesas hay variedad, como se puede ver en la figura 3-5. En principio no se puede obtener el contorno debido a la oclusión de los jugadores, y también se desconoce el color.

Figura 3-5: Ejemplo de mesas

Solución: Sabiendo la posición de las cartas comunitarias, se puede extraer una muestra del color de la mesa y ahora sí, segmentar por un rango de color, obtener el contorno, y por lo tanto ubicarla.

3.3.3 Jugadores

Tanto a nivel de diseño como de implementación, la detección de jugadores fue todo un reto. Era un caso distinto a los anteriores al no poder basarnos en atributos como el ratio altura/anchura, anteriormente útil en el caso de las cartas, ni en la forma, ni en el color, porque varían demasiado en cada plataforma, como se puede ver en la figura 3-6:

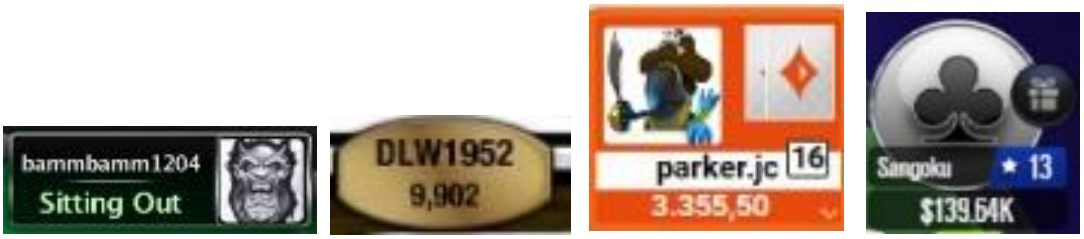

Figura 3-6: Ejemplo de jugadores

Un punto de partida prometedor era el texto y la posición. Normalmente en las interfaces, los jugadores suelen estar dispuestos cerca del perímetro de la mesa, y contienen algo de texto que suele ser el nombre del usuario, y/o el número de fichas que tiene.

Explorando esa idea, sería necesario un método para detectar texto, sin utilizar la librería de OCR (Optical Character Recognition), demasiado lenta. La implementación de este método se describe en el apartado de Desarrollo.

Para el perímetro, una vez más, el contexto sería útil ya que nos proporcionaría la posición y dimensiones de la mesa, pudiendo obtener de ahí una aproximación de la elipse, asumiendo

que la mesa suele tener esa forma. Es una gran asunción, pero se repite en la mayoría de interfaces.

Una vez se hubiesen localizado candidatos a texto, estos se filtrarían para obtener finalmente un grupo de textos con distancias al perímetro parecidas, y cercanas al perímetro.

En efecto, este ha acabado siendo el diseño final y el que mejores resultados ha dado, después de haber probado varias alternativas.

Inicialmente también se pretendía obtener el detalle de los jugadores, pero se acabó abandonando el objetivo principalmente por no contar con una librería de OCR suficientemente avanzada. En este caso era vital, ya que como se ha mencionado antes, un cuadro de texto puede hacer referencia tanto al nombre del jugador, como su capital, o su estado en la partida.

3.3.4 Cartas del jugador

La detección de las cartas del jugador es similar a lo visto con las cartas comunitarias, con algunas peculiaridades:

- En primer lugar, muchas veces están solapadas por lo que identificarlas por el ratio altura/anchura deja de ser tan fácil.
- Puede ser fácil que pasen desapercibidas al estar ubicadas en zonas con más ruido.
- Problema de falsos positivos: Podrían ser las cartas de otro jugador que se han desvelado momentáneamente.

Por los dos primeros motivos se llegó a la conclusión de que era necesario depender más del color.

Y, sobre todo, por la última razón, la detección de las “hole” cards depende claramente del contexto, en el sentido de que es necesario conocer la posición del jugador para detectar sus cartas. Para ello se solicita un input al usuario, en el que señala al sistema cuál es su jugador en la pantalla. Conociendo la posición del jugador, y el color exacto que tienen las cartas en esa plataforma, el sistema es capaz de detectarlas.

3.4 Reconocimiento

Una vez se han localizado todos los elementos, y se ha construido el contexto, el sistema pasa de preguntarse *dónde* están los elementos a averiguar *qué* contienen.

Llegado este punto, de hecho, ya no es necesario obtener la información secuencialmente y tiene la ventaja de que se podría paralelizar el proceso.

En esta fase de reconocimiento, el objetivo ha sido centrarse en actualizar el estado solamente de las cartas (tanto las de la mesa como las del jugador) y dejar como trabajo futuro el de los jugadores por las dificultades ya mencionadas a la hora de abordar el detalle.

3.4.1 Detalle de las cartas

Suponiendo que la detección se ha realizado con éxito, el algoritmo de reconocimiento podría recibir una imagen como la de la izquierda en la figura 3-7, si fuese una carta de la mesa. O la imagen del medio en el caso de reconocer las cartas del jugador, o incluso una imagen de una zona donde previamente se había detectado una carta y en ese momento no hay nada:

Figura 3-7: Ejemplos de carta simple, cartas solapadas y sin carta (de izquierda a derecha)

El primer paso consiste en comprobar si la imagen cumple la característica principal de las cartas, esto es, que suelen ser blancas. Consiste en un umbral de luminosidad media que, de no superar, supone detener el proceso y continuar con otra tarea. Esto ocurriría en el tercer caso de la figura 3-7.

Para reconocer una carta es necesario saber su valor y su palo, y en todos los casos siempre se encuentra uno encima del otro. Así pues, la base del algoritmo es buscar en la imagen pares de contornos de dimensiones mínimas similares, parcialmente alineados en el eje y.

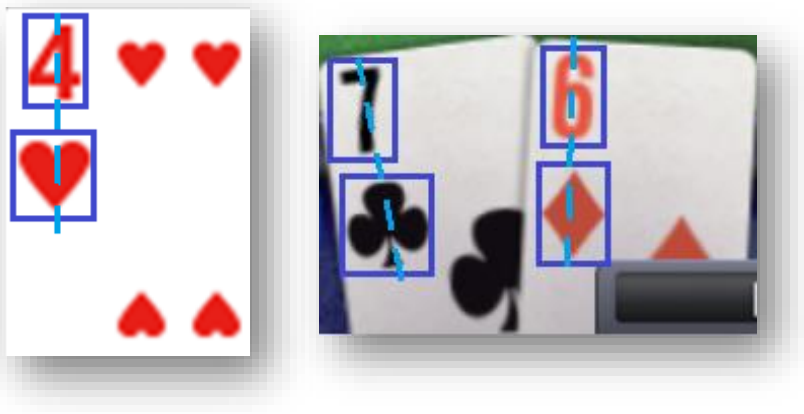

Figura 3-8: Ejemplo de reconocimiento de cartas: valor y palo (diseño)

De esta forma se tienen ubicadas la posición del valor y del palo, y se puede proceder a reconocer cada uno; el valor mediante una librería externa de OCR y el palo mediante un algoritmo implementado.

4 Desarrollo

4.1 Introducción

Se ha escogido Python 3.6 y la librería de tratamiento de imágenes OpenCV para el desarrollo del sistema por la popularidad de ambos, aunque se podía haber optado por C++ que también cuenta con dicha librería, o Matlab.

Al igual que en el capítulo anterior, este capítulo comienza describiendo la fase de detección, la parte de mayor peso del trabajo, y acaba con el reconocimiento aplicado a las cartas.

4.2 Detección

Prácticamente el proceso de detección de cada elemento se puede resumir en una etapa de preprocesamiento mediante transformaciones que permite OpenCV, una etapa de detección de contornos usando la función *findContours*, y normalmente, una etapa de filtrado de los contornos.

4.2.1 Cartas de la mesa

Preprocesamiento

Partiendo de una imagen a color, esta se pasa a escala de grises y a continuación se aplica una umbralización para destacar las cartas, asumiendo que están en un rango de luminosidad de entre 180 y 255, siendo 0 negro y 255 blanco absoluto. Como se muestra en la figura 4-1 esto permite eliminar ruido de la imagen.

Figura 4-1: Comparación visual de la imagen original (izquierda) y el resultado tras la umbralización (derecha).

Detección y filtrado de contornos

Se utiliza la función *findContours* para detectar contornos y se procede a filtrar solo los que nos interesan. En primer lugar, se extraen los contornos con 4 lados. Esto es posible gracias a la función *approxPolyDP* que nos permite aproximar un polígono a un contorno y devuelve el número de lados del polígono.

Para cada contorno de 4 lados se comprueba que su ratio altura/anchura y su área están en el rango habitual para una carta. En base a las pruebas realizadas en las distintas plataformas se ha establecido el ratio altura/anchura para una carta entre 1.3 y 1.7, y el del área entre 500 y 50 veces más pequeña que el área de la imagen completa.

Los contornos que pasan los filtros se consideran cartas y si se han detectado al menos 3 se construye el objeto TableCardSet donde se verifica que las cartas seleccionadas son realmente las de la mesa, teniendo en cuenta que están en el mismo eje y tienen la misma altura y anchura.

Si todos estos requisitos se cumplen, el objeto TableCardSet pasa a estar verificado, y se almacenan las posiciones de las cartas encontradas. A partir de este momento, en las siguientes iteraciones se llama a una función que está a caballo entre la detección y el reconocimiento, y que permite completar la detección de las cartas que faltan a la vez que realiza el reconocimiento.

Algoritmo Look

Se puede dar el caso de que se hayan verificado las cartas de la mesa habiendo detectado solo 3 o 4 en vez de las 5.

El algoritmo look consiste en "mirar", sin detección de contornos, carta por carta, partiendo de la posición de la primera carta verificada, y en el orden que se muestra en la figura 4-2. Es dentro de esta función donde se realiza una llamada a la función de reconocimiento de cartas. Si se reconoce una, se añade al grupo de cartas verificadas y se sigue buscando recursivamente en la misma dirección (empieza por la izquierda). Si no, busca en la dirección contraria a partir de la carta con la que se empezó hasta agotar todas las posibilidades.

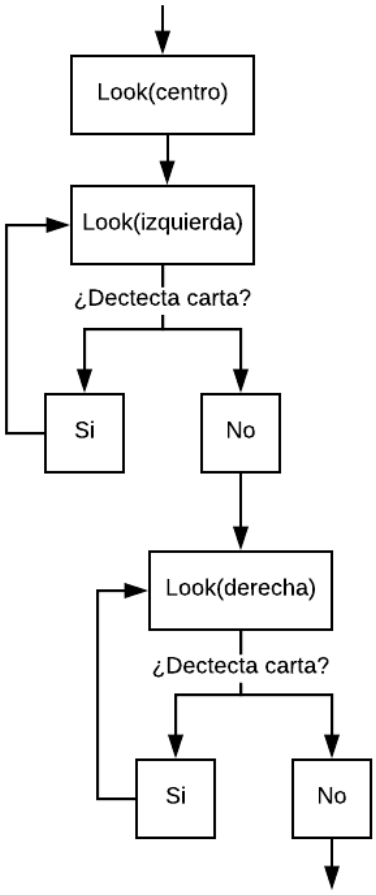

Figura 4-2: Diagrama de flujo del algoritmo de búsqueda y reconocimiento de cartas.

Restricciones en la detección:

- No hay soporte para cartas inclinadas. Igualmente, durante el desarrollo de este trabajo no se ha encontrado ninguna plataforma en la que se encuentren de esta manera, y el hecho de no soportarlas reduce la complejidad del problema (posible oclusión, necesidad de guardar más datos en memoria…) y el tiempo de procesamiento.
- No se tienen en cuenta casos excepcionales: Cartas no blancas, dimensiones atípicas, etc.

4.2.2 Mesa

El planteamiento inicial era realizar algún tipo de preprocesado sobre la imagen y buscar un contorno relativamente grande con un ratio parecido al de una mesa. El problema es que la oclusión por parte de jugadores y otros elementos impedía encontrar el contorno mediante la función *findContours*.

La solución fue apoyarse en un contexto recién construido, las cartas de la mesa. Sabiendo la posición de una de ellas, se puede obtener una muestra de la mesa, justo por encima de la carta y se obtiene la moda de cada canal. Se obtiene un rango con este color y se segmenta la imagen con una máscara en base a este rango. En la figura 4-3 se puede ver un ejemplo del proceso:

Figura 4-3: Ejemplo de extracción del rango de color de la mesa

Una vez se ha extraído la mesa a nivel de color es posible detectar el contorno, haciendo uso de findContours y obteniendo el que tiene mayor área, como se muestra en la figura 4-4:

Figura 4-4: Ejemplo de detección de la mesa

Restricciones en la detección:

- Debido al proceso de obtener una muestra de la mesa y suponer su color, solo es posible la detección si la mesa tiene un color medianamente uniforme, lo cual es lo habitual.

4.2.3 Jugadores

Tras pasar por varios diseños, se llegó a la conclusión de que la mejor solución era buscar texto (el correspondiente al nombre o al capital del jugador, por ejemplo) en el perímetro de la mesa, ya que es de las pocas cosas que tienen en común las distintas plataformas, en cuanto a los jugadores.

Esto a su vez suponía varios problemas, ¿cómo detectar texto? ¿cómo calcular su distancia al perímetro?

Afortunadamente, el texto suele tener un gran contraste con el fondo para poder ser legible y justo es esa intensidad de contraste en una vecindad lo que indica el gradiente morfológico. Haciendo uso de una función de OpenCV que permite esta transformación, *morphologyEx* y el parámetro *MORPH_GRADIENT*, obtenemos una imagen como la de la figura 4-5.

Figura 4-5: Comparación visual de la imagen original (izquierda) y el resultado tras aplicar el gradiente morfológico (derecha).

Como se puede ver, el texto destaca más después de aplicar el gradiente, pero sin embargo, sigue sin estar aislado del todo por lo que es necesario aplicar un par de transformaciones morfológicas más para simplificar la imagen, concretamente una apertura y un cierre. Para definirlas se ha tenido que asumir un tamaño mínimo de lo que se considera texto, y se ha tenido en cuenta, a la hora de crear el kernel utilizado en las transformaciones, que el texto

crece en el eje horizontal y no en el vertical. A continuación, se muestra un ejemplo del resultado en la figura 4-6:

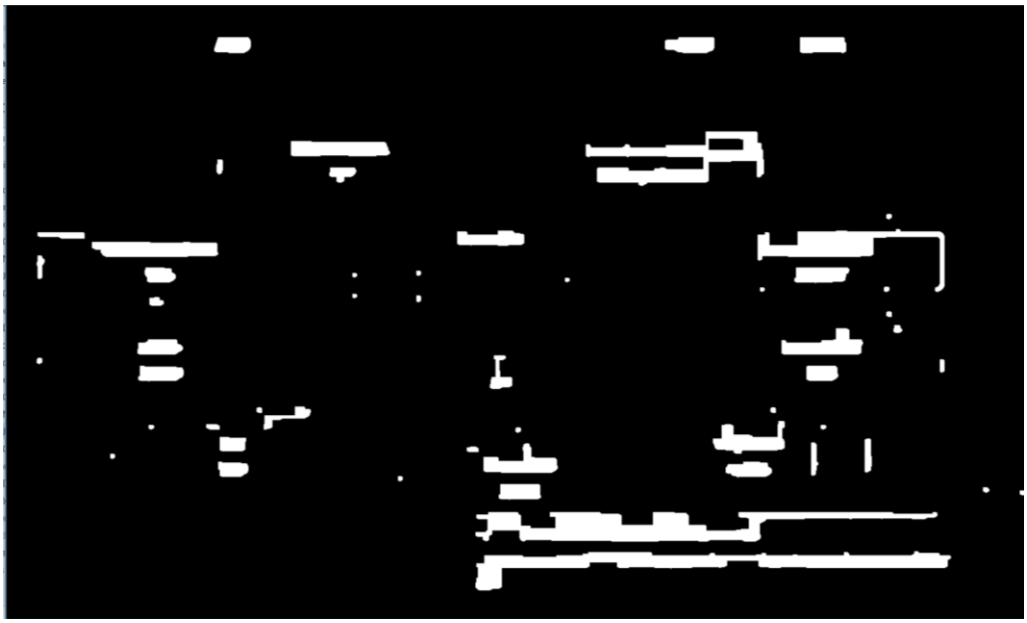

Figura 4-6: Ejemplo de bloques de texto tras simplificación morfológica

A pesar de no haberse eliminado del todo el ruido, llegado este punto ya se pueden buscar contornos y como se muestra en las figuras 4-7 y 4-8 se reconoce el texto relativamente bien.

Figura 4-7: Ejemplo 1 de detección de texto

Figura 4-8: Ejemplo 2 de detección de texto

El paso siguiente es filtrar los bloques de forma que sólo nos quedamos con los que están en el contorno de la mesa, que suele ser similar a una elipse.

A partir de la información de la mesa obtenida en el paso anterior, deducimos unos centros y un radio para las circunferencias que forman la elipse. Cabe mencionar que este enfoque tiene sus limitaciones, ya que el contorno supuesto no tiene por qué ser exactamente fiel al contorno real, aunque en muchos casos coincide.

Para simplificar el problema y obtener los candidatos a jugadores, agrupamos los contornos en base a la distancia euclídea al centro de la elipse más cercano y nos quedamos con el grupo de contornos con una distancia media al centro más similar al radio de la elipse. Por último, se hace una unión de los contornos más cercanos ya que suelen ser parte del mismo elemento. Se puede ver un ejemplo del perímetro aproximado con circunferencias y los contornos resultantes en las figuras 4-9 y 4-10.

Figura 4-9: Ejemplo 1 de detección de jugadores

Figura 4-10: Ejemplo 2 de detección de jugadores

4.2.4 Cartas del jugador

Para resolver los problemas particulares que presentan las cartas de jugador y que son distintos a los de las cartas de la mesa, se decidió diferenciar entre tableCardSet y playerCardSet. Mientras que uno almacena la posición de cada carta en el eje horizontal, el otro guarda un marco, en el que se encuentran las dos cartas.

Es preferible así porque normalmente dichas cartas están solapadas, incluso ligeramente torcidas a veces, y siempre son dos, así que se pueden tratar como un solo objeto.

La detección comienza, una vez se conoce el color de las cartas, y al igual que en casos anteriores, lo primero es una etapa de preprocesamiento. Se pasa la imagen a escala de grises y se umbraliza en base al blanco de las cartas de la mesa. También se aplica un filtro gaussiano y una dilatación para eliminar ruido.

Se buscan contornos con un ratio parecido al de una carta, o al de dos solapadas.
Para filtrar y obtener solo las del jugador, se pide al usuario que señale donde se encuentra el jugador, y se obtienen las dos cartas más cercanas a ese punto. A continuación, se muestra en la figura 4-11 un ejemplo de detección exitosa:

Figura 4-11: Ejemplo de detección de cartas del jugador

4.3 Reconocimiento

Una vez completada la detección, el sistema pasa a la fase de reconocimiento a partir de la cual se actualiza el estado de los elementos indefinidamente. En esta etapa y en el caso concreto del póker es de vital importancia contar con una librería OCR para reconocer texto, por ejemplo, a la hora de obtener el nombre de los jugadores, capital, información de un

chat, etc. En este proyecto se ha utilizado la herramienta pytesseract, un wrapper para Python del motor OCR de Google, Tesseract. Durante el desarrollo se ha comprobado que en casos simples funciona bien, pero en escenarios donde el texto puede variar más sus resultados son algo irregulares o requiere de una fase de preprocesamiento que excede el alcance de este trabajo.

Por este motivo se decidió abordar simplemente el reconocimiento de cartas y dejar el de jugadores como trabajo futuro.

4.3.1 Detalle de las cartas

Al igual que en la detección de cartas se umbraliza la imagen, en este caso para resaltar los elementos que conforman la carta, principalmente el valor y el palo.

Para descartar rápidamente las imágenes que no son cartas se hace una media del valor de los pixeles, y si esta es inferior a 100 no se prosigue. Si supera el umbral se considera que se trata de una o dos cartas y se procede a buscar contornos mediante *findContours*. Concretamente se buscan pares de contornos parcialmente alineados en el eje y, con unas alturas y anchuras similares, y con una distancia máxima entre ellos.

Como se muestra en la figura 4-12, este método se puede utilizar tanto en cartas sueltas como en un conjunto de cartas solapadas (siendo los rectángulos de trazo fino el valor, y los de mayor grosor el palo).

Figura 4-12: Ejemplo de reconocimiento de cartas: valor y palo (desarrollo)

Una vez se tienen los pares de contornos valor-palo (se asume que en todas las cartas el de arriba es el valor y el de abajo es el palo) se procede a reconocer cada uno con un método distinto.

Reconocimiento del valor

Se aplica una umbralización más un filtrado gaussiano para limpiar la imagen y se utiliza la función *image_to_string* de pytesseract con un parámetro que le indica que la imagen contiene sólo un carácter. Esto excluye al 10 y es un aspecto a mejorar, pero se gana en precisión a la hora de reconocer el resto de valores.

Reconocimiento del palo

De nuevo, en cuanto a preprocesamiento se aplica una umbralización más un filtrado gaussiano. Para clasificar el palo, el algoritmo tiene en cuenta dos aspectos: el color (que puede ser rojo o negro), y la forma. Para la forma, se ha vuelto a utilizar la función *approxPolyDP* que permitía aproximar un polígono a un contorno y saber el número de lados que tiene, especificando previamente la precisión con la que se quiere aproximar.

Las reglas del algoritmo para clasificar se muestran en el diagrama de la figura 4-13:

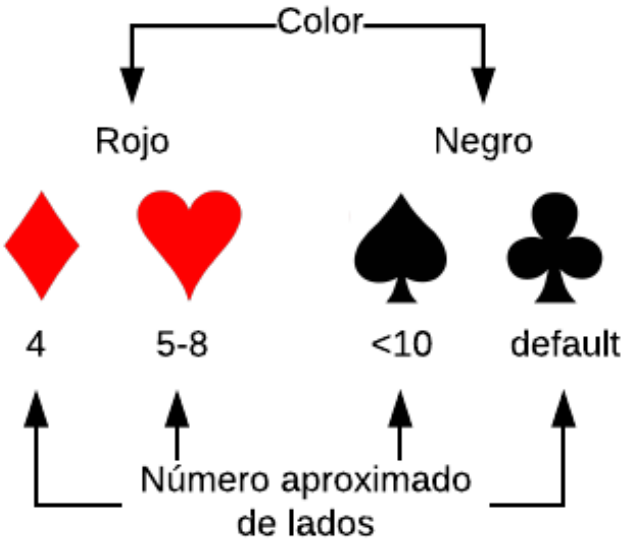

Figura 4-13: Flujo de reconocimiento del palo

5 Integración, pruebas y resultados

En el capítulo de Diseño se hacía referencia a lo importante que ha sido tener en cuenta la modularidad del código. Esta ha permitido afrontar cada problema de manera prácticamente individual, en la medida de lo posible ya que se depende del contexto, pero claramente ha facilitado realizar modificaciones, probar cada elemento por separado, y finalmente juntar todo.

A la hora de probar el funcionamiento del sistema, se ha hecho de dos formas distintas: prueba con una imagen (1 frame) y pruebas en tiempo real en una partida en juego (varios frames).

Principalmente se han hecho pruebas del primer tipo, por la siguiente razón: La detección solo requiere de unos pocos frames para resolverse, pero tienen que ser frames que cumplan ciertas condiciones, como que haya 3 cartas comunitarias para que empiece a construirse el contexto. Proporcionar directamente al sistema una imagen que reúna estos requisitos agiliza las pruebas y también te evita el proceso de entrar a una partida, seleccionar la ventana que se quiere analizar, etc. Por supuesto, también dan una muy buena idea del funcionamiento de la fase de reconocimiento.

Además, se han hecho con solo una imagen por plataforma, ya que el escenario apenas cambia durante las partidas, en lo que respecta a la detección.

También se han hecho pruebas en tiempo real, pero en menor medida. La ventaja de estas pruebas ha sido que permite ver cómo se comporta el sistema en la fase de reconocimiento, actualizando constantemente cada pocos segundos.

5.1 Pruebas

Características del equipo utilizado para las pruebas:

- Procesador: Intel(R) Core(TM) i5-8250U
- Memoria RAM: 8 GB

A continuación, se muestran los resultados de ejecutar la batería de pruebas utilizando una sola imagen por plataforma, en 8 plataformas distintas. En rojo se muestran los elementos clasificados como cartas, tanto las de la mesa como las del jugador, en verde la mesa y en azul los jugadores. Además, se muestra el output tras la fase de reconocimiento, e información sobre la precisión y el tiempo que ha tardado en ejecutar cada fase.

Figura 5-1: Prueba 1

Detección:

- Tiempo: 0.02693 s
- Precisión:
  - Cartas comunitarias: Detectadas: 4/4 Falsos positivos: 0
  - Jugadores: Detectados: 9/9 Falsos positivos: 0
  - Mesa: Detectada
  - Cartas del jugador: No procede

Reconocimiento:

- Tiempo: 0.62433 s
- Precisión:
  - Cartas comunitarias: Valor: 3/4 Palo: 4/4
  - Cartas del jugador: No procede


```
Table Cards: [J♠, ♥, J♦, J♥]
```

Figura 5-2: Prueba 2

Detección:

- Tiempo: 0.06985 s
- Precisión:
  - Cartas comunitarias: Detectadas: 4/4 Falsos positivos: 0
  - Jugadores: Detectados: 9/9 Falsos positivos: 1
  - Mesa: Detectada
  - Cartas del jugador: No detectadas

Reconocimiento:

- Tiempo: 0.61446 s
- Precisión:
  - Cartas comunitarias: Valor: 3/4 Palo: 4/4
  - Cartas del jugador: No procede


```
Table Cards: [4♥, A♠, 0♠, 6♦, J♥]
```

Figura 5-3: Prueba 3

Detección:

- Tiempo: 0.02293 s
- Precisión:
  - Cartas comunitarias: Detectadas: 5/5 Falsos positivos: 0
  - Jugadores: Detectados: 4/4 Falsos positivos: 8
  - Mesa: Detectada
  - Cartas del jugador: No procede

Reconocimiento:

- Tiempo: 0.93342 s
- Precisión:
  - Cartas comunitarias: Valor: 4/5 Palo: 5/5
  - Cartas del jugador: No procede

Figura 5-4: Prueba 4

Detección:

- Tiempo: 0.02789 s
- Precisión:
  - Cartas comunitarias: Detectadas: 5/5 Falsos positivos: 0
  - Jugadores: Detectados: 0/6 Falsos positivos: 1
  - Mesa: Detectada
  - Cartas del jugador: No procede

Reconocimiento:

- Tiempo: 1.38897 s
- Precisión:
  - Cartas comunitarias: Valor: 3/5 Palo: 5/5
  - Cartas del jugador: No procede


```
Table Cards: [3♠, 2♥, 5♠, 9♣, 0♠]
Player Cards: [6♥, Q♦]
```

Figura 5-5: Prueba 5

Detección:

- Tiempo: 0.02694 s
- Precisión:
  - Cartas comunitarias: Detectadas: 5/5 Falsos positivos: 0
  - Jugadores: Detectados: 4/6 Falsos positivos: 7
  - Mesa: Detectada
  - Cartas del jugador: Detectadas

Reconocimiento:

- Tiempo: 0.77223 s
- Precisión:
  - Cartas comunitarias: Valor: 4/5 Palo: 4/5
  - Cartas del jugador: Valor: 2/2 Palo: 2/2

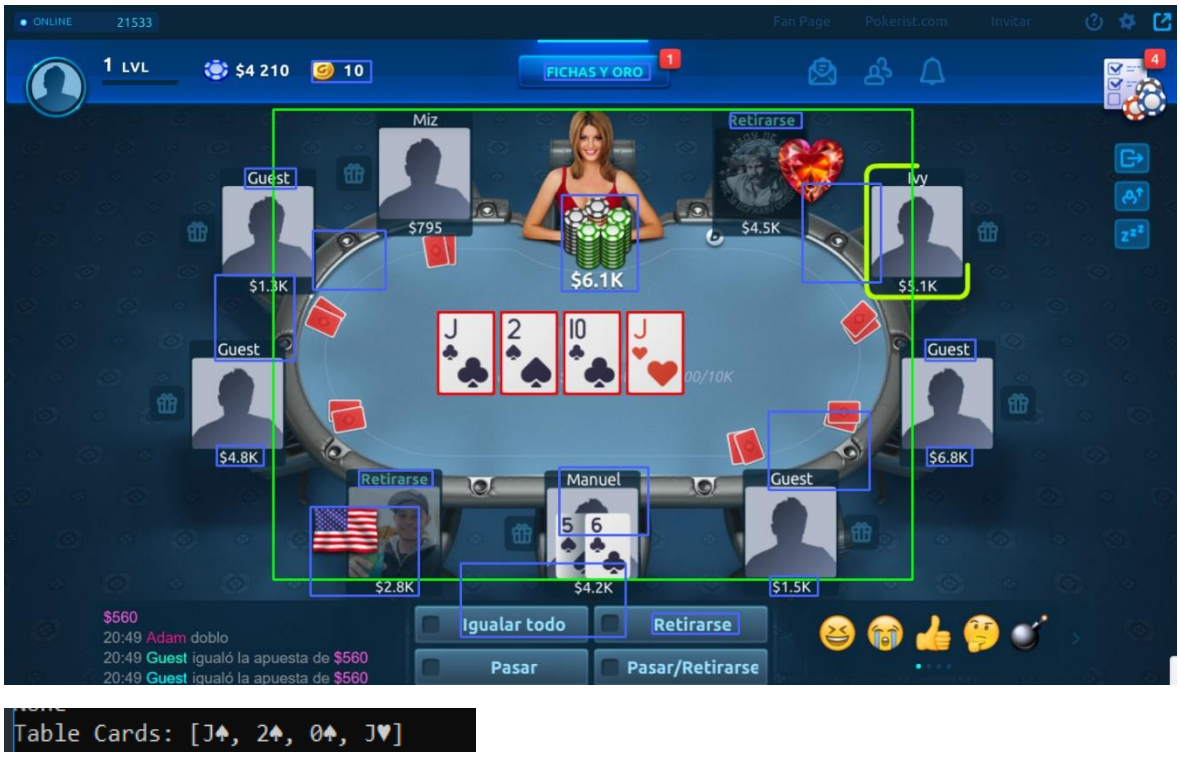

Figura 5-6: Prueba 6

Detección:

- Tiempo: 0.04588 s
- Precisión:
  - Cartas comunitarias: Detectadas: 4/4 Falsos positivos: 0
  - Jugadores: Detectados: 0/9 Falsos positivos: 16
  - Mesa: Detectada (resultado mejorable)
  - Cartas del jugador: No detectadas

Reconocimiento:

- Tiempo: 0.61795 s
- Precisión:
  - Cartas comunitarias: Valor: 4/4 Palo: 2/4
  - Cartas del jugador: No procede


```
Table Cards: [Q♥, J♠, 8♦, 7♠, Q♠]
Player Cards: [7♦, 4♠]
```

Figura 5-7: Prueba 7

Detección:

- Tiempo: 0.03092 s
- Precisión:
  - Cartas comunitarias: Detectadas: 5/5 Falsos positivos: 0
  - Jugadores: Detectados: 0/5 Falsos positivos: 1
  - Mesa: Detectada
  - Cartas del jugador: Detectadas

Reconocimiento:

- Tiempo: 0.76309 s
- Precisión:
  - Cartas comunitarias: Valor: 5/5 Palo: 5/5
  - Cartas del jugador: Valor: 2/2 Palo: 2/2

Figura 5-8: Prueba 8

Detección:

- Tiempo: 0.03893 s
- Precisión:
  - Cartas comunitarias: Detectadas: 4/4 Falsos positivos: 0
  - Jugadores: Detectados: 0/4 Falsos positivos: 1
  - Mesa: Detectada
  - Cartas del jugador: No detectadas

Reconocimiento:

- Tiempo: 1.08102 s
- Precisión:
  - Cartas comunitarias: Valor: 4/4 Palo: 4/4
  - Cartas del jugador: No procede

5.2 Resultados

Resumen (media de los resultados):
Detección:

- Tiempo: 0.03628 s
- Precisión:
    - Cartas comunitarias: Detectadas: 100% Falsos positivos: 0
    - Jugadores: Detectados: 45.83% Falsos positivos: 4
    - Mesa: Detectada: 100%
    - Cartas del jugador: Detectadas: 40%

Reconocimiento:

- Tiempo: 0.84943
- Precisión:
    - Cartas comunitarias: Valor: 83.75% Palo: 91.25%
    - Cartas del jugador: Valor: 100% Palo: 100%

Discusión:
Estos datos, a pesar de ser algo irregulares, nos dan una idea aproximada de las capacidades del sistema. En cuanto a eficiencia, se puede ver que el sistema puede funcionar bien en tiempo real, ya que de media el reconocimiento no alcanza el segundo. En cuanto a la detección, funciona a la perfección tanto para las cartas de la mesa como la propia mesa. También el reconocimiento de las cartas es muy bueno, aunque algo lento, principalmente debido a la librería de OCR.

Los resultados son bastante más dispares en cuanto a la detección de los jugadores y las cartas del jugador. Aunque el sistema no se diseñó con una plataforma específica en mente, se puede apreciar que está sesgado por la plataforma que se utilizó de referencia principal durante el desarrollo, y es uno de los primeros problemas que se debería resolver en un trabajo futuro.

A pesar de ello, la detección de jugadores es considerablemente buena en 3 plataformas y la detección de las cartas de jugador en 2, así que el objetivo de un sistema general para estos dos elementos se da por conseguido parcialmente.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Este trabajo de fin de grado partió de la idea de crear un sistema capaz de extraer información de una partida de póker de la misma forma que lo haría una persona, interpretando las imágenes de la interfaz.

Cómo es normal, los primeros pasos consistieron en detenerse a estudiar el problema y desgranarlo en varias partes y también en estudiar procedimientos comunes en materia de CV. Este además ha sido uno de los objetivos desde el principio, profundizar en el conocimiento de formas de preprocesamiento de imágenes, detección de puntos de interés, obtención de contornos, etc. y ver cómo funcionan en la práctica.

Parte de la fase de diseño también supuso decidir si usaría algún patrón de diseño, optando finalmente por programación orientada a objetos, y se buscó la manera de hacer el código más eficiente, surgiendo de ahí la división del sistema en dos etapas.

Estoy contento con cómo ha ido el desarrollo ya que fue bastante fluido, pero sin estar exento de retos. Algunos "pequeños" como resolver la oclusión en cartas o mejorar el reconocimiento, y otros, como la detección de jugadores principalmente, que supuso probar varios métodos uno tras otro hasta dar con el actual, con unos resultados bastante satisfactorios, pero mejorables. En el resto de elementos (cartas y mesa), la detección es muy buena, y con el estado actual del proyecto ya se podría dar lugar a un asistente que, por ejemplo, calculase la fuerza de una mano.

Dejando de lado los resultados, también valoro los conocimientos adquiridos. Sin duda, me ha servido para mejorar en Python y en el manejo de algunas librerías populares como OpenCV y NumPy. He aprendido varias técnicas útiles en procesamiento de imágenes y una intuición de cuando es mejor usar una y cuando es mejor usar otra. Y, sobre todo, ir descubriendo esta área, bastante desconocida para mí e ir viendo avanzar el proyecto ha sido una experiencia muy gratificante.

6.2 Trabajo futuro

A pesar de que en general los resultados han sido satisfactorios, aún hay mucho margen de mejora, más en algunas partes que en otras. Estos serían los principales aspectos a mejorar en un futuro:

- Mejorar la detección de los jugadores, incluso con un enfoque totalmente distinto, en el que se tenga en cuenta que los jugadores son un patrón que se repite en la imagen, y no solo bloques de texto alrededor de un perímetro.
- Conseguir un sistema más general. Minimizando restricciones, ajustando parámetros, etc. Principalmente para los jugadores, y las cartas del jugador.

- Reducir al mínimo los inputs del usuario. Actualmente son tres: coordenadas de la ventana a analizar al inicio, posición del usuario en la mesa (solicitado en la fase de detección). Los tres podrían ser inferidos haciendo un sistema más inteligente.
- Utilizar una librería de OCR más eficiente y capaz de reconocer texto en escenarios complejos y/o mejorar las etapas de preprocesamiento para facilitar la tarea.

Referencias

[1] D. Molin, T. F. Rosin, “Determining the state of thexas hold ’em in almost to real time”, 2014.
[2] J. Hrdlicka, M. Hlavač, “Poker Cards Recognition using Neural Networks”, 2017.
[3] P. Martins , L. P. Reis, L. Teófilo, “Poker Vision: Playing Cards and Chips Identification based on Image”, 2012.
[4] ProcessingOpenCV, “http://www.opencv.org/”.
[5] Pytesseract, “https://github.com/madmaze/pytesseract”.

Glosario

CV	Computer Vision
OCR	Optical Character Recognition
AI	Artificial Intelligence