04/12/2023

Visión Por Ordenador

Informe de Proyecto Final

Lara Ocón Madrid
Lucía Prado Fernández-Vega

Visión Por Ordenador

Introducción

El alcance de la práctica abarca varias etapas fundamentales en el procesamiento de imágenes y visión por computadora. En primer lugar, se lleva a cabo una calibración (***calibration.py***) efectiva de la cámara para obtener sus parámetros intrínsecos y extrínsecos empleando imágenes de un patrón compuesto por 5x4 círculos.

Posteriormente, se implementa la detección patrones mediante SIFT (***detector.py***), que más tarde emplearemos para la ejecución del decodificador de secuencia. Dicho decodificador memoriza la cantidad de patrones consecutivos que desee el usuario y garantiza el paso al siguiente bloque si dicha secuencia se introduce correctamente. En caso de mostrar una carta incorrecta o desordenada, el proceso se reinicia.

Una vez superada la etapa de detección de secuencia, se introduce un módulo que emplea trackers para detectar y seguir figuras con colores específicos (***formas.py***). El usuario debe seleccionar la forma a mostrar por pantalla, que determinará la figura que el usuario deberá dibujar en la fase final de la práctica (***dibujar.py***). Al finalizar la trayectoria de dicha figura mostrada por pantalla, se calcula la similitud entre la figura mostrada y la dibujada para evaluar el rendimiento del usuario.

Todos estos módulos son incorporados por 1 solo en el cual se ejecutan las distintas fases secuencialmente, y se guarda la salida en un archivo mp4 (***practica.py***).

Requisitos y tareas

Calibración

Lo primero que llevaremos a cabo será la calibración de la cámara, con el objetivo de sacar sus parámetros intrínsecos y extrínsecos. Para ello, emplearemos un patrón de 5x4 círculos y una distancia de separación de 2cm. Realizaremos 23 fotos a dicho patrón desde distintos ángulos y compararemos los puntos imagen con los puntos reales del patrón.

Detección de secuencia de patrones con SIFT

Una vez hayamos calibrado la cámara, pasaremos a la detección de una secuencia de patrones mediante SIFT. Para ello, usaremos las cartas del juego *'UNO'* y el usuario deberá mostrar la secuencia 1, 2, 3, 4 (en ese mismo orden) para poder desbloquear la contraseña y pasar a la siguiente fase. En caso de que se muestre otra carta no perteneciente a la secuencia (por ejemplo, el 7 o la carta de cambio de sentido) o bien una carta de la secuencia, pero no en el orden que toca (por ejemplo, 1 y después 4), se reiniciará el proceso de desbloqueo obligando a tener que mostrar la secuencia nuevamente desde el principio.

Detección de formas y colores con Trackers

Desbloqueada la secuencia, se iniciará un módulo capaz de detectar 3 figuras con colores específicos (cuadrado amarillo, circulo azul y triángulo naranja) y. seguirlas a lo largo del vídeo. El objetivo de esta parte de la práctica es que el usuario elija una de las 3 formas, cada una de ellas indicará qué es lo que se va a dibujar en la siguiente parte de la práctica. Es decir, si el usuario elige el cuadrado, en la siguiente fase dibujará un cuadrado, si elige triángulo, un triángulo; y si elige círculo, pasará a dibujar el logo de

Disney. Para determinar la figura elegida por el usuario, este deberá mostrar en el vídeo una sola de las 3 formas y pulsar la tecla 's'. En caso de mostrar más de una figura a la vez en la pantalla o no mostrar ninguna, el programa seguirá en funcionamiento hasta que se cumpla el requisito (es decir, una única forma en la imagen al pulsar 's')

Comparación de contorno objetivo vs. dibujado con un Tracker

En la última parte del programa realizaremos un juego interactivo en el cual se mostrará una forma sobre el vídeo (cuadrado, triángulo o el logo de Disney) y se pedirá al usuario que replique esa forma, tratando de seguir los contornos con el uso de un tracker amarillo. Para comenzar a pintar el usuario deberá pulsar la tecla 's', y una vez haya terminado de delinear el contorno pulsará la tecla 'e'. Hecho esto, se calculará la similitud entre la figura mostrada y la pintada por el usuario para indicarle si lo ha hecho bien o mal.

Metodología

Calibración (calibration.py)

Como punto de partida de la práctica, hemos implementado la calibración de la cámara mediante el uso de un patrón compuesto por una red de 5x4 círculos a una distancia de 2cm entre ellos. Para tomar las imágenes del tablero eficientemente empleamos un código que permite tomar una ráfaga de fotos al presionar una determinada tecla, variando el ángulo desde el que tomamos cada imagen. Cargamos las imágenes de calibración y empleamos la función 'cv2.findCirclesGrid' para detectar los centros de los círculos en el patrón.

Para calibrar la cámara, se emplean los centros de los círculos detectados ('image points') junto con los puntos reales (que hemos calculado desarrollando la función 'get_circle_centers', dadas las dimensiones del patrón y la separación entre elementos del patrón, devuelve una lista con los centros) calculamos el rms, parámetros intrínsecos, extrínsecos y los coeficientes de distorsión con la función 'cv2.calibrateCamera'. Los valores obtenidos son los siguientes:

```
Calibration matrix:
-------------------
Intrinsic parameters:
[[963.99000049   0.         315.69719588]
[  0.         961.31983307 228.85835581]
[  0.           0.           1.        ]]

Distortion coefficients:
[[ 6.94492612e-02 -8.56240698e-01 -1.86174421e-03 -3.81191727e-03 4.92245679e+00]]

RMS:
0.1989979308762485
```

Detección de secuencia de patrones con SIFT (detector.py)

Para el módulo del decodificador de secuencia dentro del desarrollo de la práctica, hemos empleado el algoritmo SIFT para la detección de patrones.

Para empezar, en la llamada inicial a la función de *obtener patrones* se realiza la carga de las imágenes correspondientes a los posibles patrones de la secuencia. En este caso, los patrones se tratan de las cartas del juego UNO, previamente tomadas con la cámara, pudiendo seleccionar la secuencia que se desee introducir como contraseña. Una vez cargadas las imágenes, se obtienen los keypoints y descriptores de cada una de ellas mediante el algoritmo *detect and compute* de *SIFT*, y esta información se organiza en

dos diccionarios distintos según si forman parta de la secuencia (*patron*) o no (*not patron*). En ambos diccionarios, la clave corresponde al número de la imagen, y el valor asociado a su lista de descriptores.

Habiendo obtenido los diccionarios, se inicia la cámara y comienza la ejecución del bucle que deberá capturar frames consecutivos hasta que el sistema sea desbloqueado con la introducción de la secuencia correcta de patrones. En primer lugar, se inicializan las variables del estado de desbloqueo (*False*) y el indicador para salir del bucle principal (*salir*), además de las instancias de los objetos SIFT y BFMatcher para la detección y comparación de descriptores, respectivamente. Para rastrear el patrón siguiente y el patrón actual en la secuencia, se establecen los índices *next_pattern* y *current_pattern,* que permiten determinar si la secuencia introducida es la correcta.

El bucle principal se ejecuta mientras el sistema no esté desbloqueado y no se decida salir, capturando en cada iteración un nuevo frame de la cámara para extraer sus keypoints y descriptores mediante el algoritmo de SIFT. Una vez obtenidos los descriptores, se lleva a cabo una comparación exhaustiva de estos con los descriptores almacenados en los diccionarios de los patrones y no patrones utilizando el BFMatcher. En cada caso, se busca la mejor coincidencia con los patrones mediante la implementación del *ratio test de Lowe*, para asegurar que solo se dé un emparejamiento en caso de que verdaderamente se trate del patrón que le corresponde.

Si se encuentra un emparejamiento significativo, se evalúa si corresponde a un patrón, no patrón o si no se ha detectado nada. En caso de que corresponda a un patrón, se comprueba si se trata del siguiente elemento de la secuencia, en cuyo caso se actualizan los índices de *next_pattern* y *current_pattern* para pasar a detectar el siguiente elemento: si el índice que corresponde al siguiente patrón a detectar vale 5, se ha completado la secuencia correctamente y se sale del bucle, de lo contrario, se vuelve a entrar al bucle y continua el proceso. Si se introduce un patrón que no corresponde al siguiente elemento, la búsqueda de la secuencia se reinicia y debemos introducir la contraseña de cero. A lo largo del proceso de detección, se muestra visualmente el estado del reconocimiento de patrones en tiempo real mediante la función *mostrar_texto*, que coloca un texto en el frame. El bucle continúa hasta que el sistema está desbloqueado o hasta que se decide salir mediante la tecla 'q'.

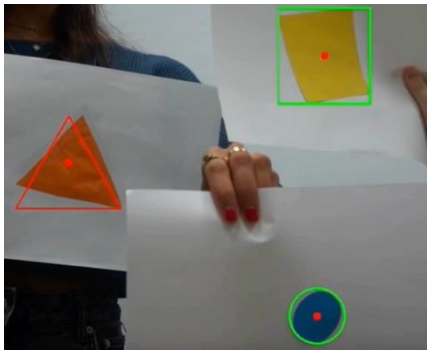
Detección de formas y colores con Trackers (formas.py)

En este módulo, hemos decidido trackear tres formas distintas de 3 colores diferentes a lo largo del vídeo. Estas han sido: cuadrado amarillo, círculo azul y triángulo naranja. En los 3 casos hemos aplicado en primer lugar una máscara de color específico. Esta mascara la hemos conseguido aplicando a la imagen en HSV la función 'cv2.inRange' pasándole los 'upper-range' y 'lower-range' de ese color específico. Una vez hemos hecho esto, obtenemos los contornos de la máscara mediante 'cv2.findContours'. En caso de haber encontrado contornos, tomamos el máximo de estos (comparando por el área del contorno), encontramos el perímetro de dicho contorno con la función 'cv2.arcLength' y aproximamos dicho perímetro a un polígono 'cv2.approxPolyDP'. Hasta aquí el proceso es el mismo para las 3 formas (con la distinción de que en cada uno el color aplicado para obtener la máscara difiere).

Si el polígono tiene 4 vértices quiere decir que hemos encontrado un cuadrado, de forma que obtenemos el rectángulo más pequeño que contiene al polígono con la función 'cv2.boundingRect', que nos devuelve x,y,w,h (esquina superior izquierda del rectángulo, width y height) información que nos servirá para más adelante mostrar el seguimiento del tracker por el vídeo. Además, con ello también podemos actualizar las coordenadas previas del tracker de la siguiente forma: prev_x, prev_y = x + w//2, y + h//2.

En el caso de que el polígono tenga 3 vértices hacemos un proceso muy parecido, obteniendo el rectángulo más pequeño que contiene al triángulo y así obtener x,y,w,h información que nos servirá para dibujar el contorno el triángulo sobre el vídeo cuando sea detectado.

En el caso del círculo, el proceso difiere un poco. Una vez hemos obtenido los contornos, recorremos cada uno de ellos y tomamos su perímetro y su área, con ello, calculamos su 'circularity', para medir como de 'circulo ideal' es. Si para un círculo perfecto 2*pi*r = P y pi*r^2 = A, entonces, r = P/(2pi) y sustituyendo en la segunda obtenemos que (P^2)/(4pi) = área => 4*pi*área / P^2 = 1. Es asi como definimos la circularity: circularity = 4*pi*área / P^2. De forma que cuanto más cercana a 1, mejor circulo es. Una vez calculado este valor, definimos que si es superior a un threshold (en nuestro caso 0.85) lo damos por bueno y calculamos el centro y el radio del mínimo círculo que encierra ese contorno con 'cv2.minEnclosingCircle'. Esto lo hacemos para todos los contornos, y finalmente, nos quedamos con aquel círculo (con circularidad > 0.85) y de mayor radio.



A la hora de dibujar las formas, hemos desarrollado una función a la cual le pasas el tipo de forma ('cuadrado', 'triangulo' y 'circulo'), las coordenadas (x,y) de la forma y el w, h en caso del cuadrado y triángulo, y el radio en caso del círculo. En función de la forma que sea, pinta el cuadrado con 'cv2.rectangle'; para el triángulo, calculamos los 3 vértices y los pintamos con 'cv2.drawContours'; y finalmente, para el círculo, lo dibuja con 'cv2.circle'.

Dado que el propósito de este módulo es que el usuario elija una de las 3 formas para dibujar en la siguiente fase, el bucle principal de esta parte funciona de la siguiente forma: para cada frame, se llama a la función específica de cada tracker, devolviendo cada una de estas las coordenadas y dimensiones de dichas formas. Para cada una de ellas, en caso de que se hayan detectado se llama a la función 'dibujar_forma' para mostrarla sobre el video. Después, se comprueba si el usuario ha pulsado la tecla 's', en caso de hacerlo y que en dicho momento haya 1 y solo una de las figuras en la imagen, se devolverá el nombre de la forma y finalizará la ejecución del programa. En caso de haber más de una o ninguna, no pasará nada, seguirá llevando a cabo la lectura de frames y detección de formas. Por otro lado, si en cualquier momento se pulsa 'q', se finalizará la captura de frames y la ejecución del programa.

Comparación de contorno objetivo vs. dibujado con un Tracker (dibujar.py)

Para esta parte de la práctica, hemos llevado a cabo el seguimiento de un puntero amarillo. Para llevar a cabo dicho tracker, hemos aplicado una máscara de color amarilla y nos hemos quedado con el mayor de los contornos (además, dicho contorno solo nos vale si tiene un radio mayor que 10, para asegurarnos de no estar detectando el tracker en momentos en los que no se está mostrando). Por otro lado, hemos controlado que hasta que no se pulse la tecla 's' no se comenzará a guardar la trayectoria del tracker, y que una vez se pulse la tecla 'e' se deje de guardar.

Para mostrar la trayectoria sobre el vídeo desarrollamos la función 'draw_trajectory(frame, trajectory)' en cual hacemos lo siguiente: En primer lugar, suavizamos la trayectoria original, iterando sobre todos los puntos de la trayectoria y calculando el punto medio entre cada par de puntos consecutivos y agregándolo a la lista 'smooth_trajectory'. Hecho esto, pasamos a mostrar la trayectoria mediante la función cv2.line, la cual conecta los puntos consecutivos que le hayamos pasado con una línea del grosor y color que le indiquemos.

De forma paralela, superponemos sobre la imagen de video mostrada la figura objetivo que se va a dibujar. En el caso del rectángulo, usamos la función "cv2.rectangle", pasándole el frame sobre el cual lo vamos a pintar, coordenadas de la esquina superior izquierda, coordenadas de la esquina inferior derecha, color, grosor. En el caso del triángulo, calculamos los vértices de este, hacemos un reshape de los puntos para que sea un array de 3x1x2 y finalmente lo mostramos sobre el frame con la función

cv2.polylies, pasándole el frame, los puntos, si está cerrado o no el contorno, el color y el grosor. Finalmente, para mostrar el logo tipo de Disney, el proceso es algo más complicado. En primer lugar, cargamos el logo del fichero 'disney-channel-logo.png' con la función cv2.imread. Después flipeamos el logo en espejo (ya que después al mostrar el video se volverá a flipear la imagen). Extraemos los canales R, G,B y alpha del logo, el canal alpha nos dará la máscara del logo. Además, redimensionamos el logo y la máscara para que tenga el tamaño deseado mediante la función cv2.resize. Después, especificamos las coordenadas donde situaremos el logo para que se muestre en la esquina inferior derecha (recordar que al mostrar el video se verá en la izquierda). Finalmente, mediante un bucle para cada canal de color (R, G, B) usaremos la transparencia del canal alfa ('logo_mask_resized') para combinar los píxeles del logo con los píxeles del marco del video.

Una vez se hayan superpuesto tanto la trayectoria, como la forma que se vaya a dibujar sobre el frame, flipeamos dicho frame en espejo, y sobre este agregamos un texto dando la indicación necesaria al usuario (Pulsa 's' para comenzar a dibujar, pulsa 'e' para terminar de dibujar, la figura está bien, la figura está mal…) para después mostrar dicho frame. De esta forma, el bucle se ejecutará hasta que el usuario termine de dibujar y pulse 'e', donde se mostrará si ha dibujado bien o mal la forma, se esperarán 5 segundos y se finalizará la ejecución del programa. También, podrá salir del programa de forma inmediata pulsado 'q'.

Para determinar si el usuario ha dibujado bien o mal la forma, tras probar distintos métodos, finalmente nos decantamos por el siguiente que se define en la función 'compare_trajectory(trajectory, shape)'. A este método le pasamos tanto la trayectoria como la forma 'objetivo' y dibuja sobre 2 imágenes en blanco, en una la trayectoria, y en otra la forma. Después, pasa las imágenes a escala de grises y busca los contornos sobre ambas imágenes, quedándose con todos los contornos de la imagen 1 (la de la trayectoria) y solo con el contorno de mayor área de la segunda imagen (obtiene el contorno de la figura perfecta que se quiere dibujar). Hecho esto, calcula la distancia Hu entre cada uno de los contornos de la imagen con la trayectoria con el contorno de la figura deseada mediante la función cv2.matchShapes. La distancia Hu es una métrica propuesta por Ming-Kuei en 1962 que se basa en los momentos Hu, invariantes en escala, rotación y traslación. La idea principal detrás de los momentos de Hu es capturar características importantes de la forma del contorno que sean invariantes a ciertas transformaciones. Estos momentos se utilizan para calcular un conjunto de siete valores, cada uno representa una característica específica de la forma del contorno. La distancia de Hu se calcula como la diferencia entre los momentos de Hu de dos contornos. Cuando se utiliza la función cv2.matchShapes, se comparan los momentos de Hu de dos contornos para medir qué tan diferentes son. Si los contornos son idénticos o muy similares, la distancia de Hu será cercana a cero, lo que indica una mayor similitud entre las formas. Por otro lado, cuanto mayor sea el valor de la distancia de Hu, mayor será la diferencia entre las formas. De esta forma, una vez calculadas las distancias Hu entre cada uno de los contornos, con el contorno de la figura 'perfecta', cogemos el menor de todos, y si este está por debajo de un threshold determinado devolvemos True (se ha pintado bien la forma) o False (no se ha pintado bien la forma).

Tras hacer varias pruebas vimos que la distancia Hu para dibujar un cuadrado aceptable podía dar valores muy bajos (de 0.004 aproximadamente), mientras que, con el logo de Disney, al ser más complicado de dibujar, no lográbamos bajar de 2. Es por ello que finalmente decidimos establecer distintos valores de threshold para cada una de las 3 formas.

En todos los módulos descritos, la imagen mostrada en el vídeo ha sido flippeada (en espejo), de forma que el usuario pueda seguir sus movimientos en el vídeo de forma más intuitiva.

Resultados y futuros desarrollos

Resultados

La práctica aborda de manera incremental el procesamiento de imágenes y visión por computadora mediante diversas etapas. En primer lugar, la calibración de la cámara se realiza con éxito, extrayendo parámetros intrínsecos y extrínsecos necesarios para una representación precisa del entorno visual. La implementación de SIFT para la detección de secuencias de patrones, utilizando cartas del juego 'UNO', agrega un elemento interactivo y de validación al proceso de manera original, permitiendo la asignación de un amplio rango de secuencias como contraseña de acceso al sistema.

La introducción de un módulo de seguimiento con *trackers* para detectar figuras con colores específicos añade complejidad y creatividad al proyecto, además de ofrecer al usuario la capacidad de seleccionar una determinada forma según la figura que desee dibujar en la última fase, proporcionando una experiencia personalizada.

Finalmente, la fase interactiva de comparación de contornos mediante un tracker amarillo cierra la práctica con un juego que evalúa la habilidad del usuario para replicar formas mostradas. La implementación de la similitud entre la figura mostrada y la pintada proporciona una retroalimentación inmediata.

Futuros Desarrollos:

Como futuros desarrollos para ampliar el despliegue de la práctica y extender su aplicación a nuevas tecnologías y enfoques en el campo de la visión por computadora, se plantean las siguientes funciones:

- ***Ampliación de Figuras***: Además de cuadrados, triángulos y el logo de Disney, podrían agregarse más opciones de figuras o incluso permitir al usuario diseñar su propia figura.
- ***Optimización del Tracker***: Se podría trabajar en la optimización del tracker amarillo para mejorar la precisión y la fluidez en el seguimiento de contornos.
- ***Incorporación de Machine Learning***: La implementación de algoritmos de aprendizaje automático podría permitir una mayor adaptabilidad a diferentes estilos de dibujo del usuario y proporcionar evaluaciones más personalizadas.
- ***Aumento de la Complejidad del Juego***: Se podrían agregar niveles de dificultad progresivos o desafíos adicionales en el juego interactivo para mantener el interés y proporcionar una experiencia más enriquecedora.

A partir de estas mejoras, algunas de las posibles aplicaciones prácticas del proyecto en campos como la salud, la educación y la terapia, así como en el desarrollo personal y creativo, son las siguientes:

- ***Evaluación Psicomotora***: El sistema se podría ampliar con pruebas psicomotoras más avanzadas. Por ejemplo, el seguimiento de movimientos del tracker podría ser utilizado como herramienta de evaluación psicomotora para medir la coordinación mano-ojo y la precisión en tareas específicas.
- ***Terapia de Rehabilitación***: La aplicación podría utilizarse en terapias de rehabilitación física o cognitiva. Por ejemplo, el seguimiento y la evaluación de la precisión de los trazos podría ser empleado para monitorizar la coordinación motora de pacientes que lo requieran.
- ***Aplicaciones en Educación Infantil***: Adaptar el sistema para entornos educativos con el fin de fomentar la creatividad y la interacción en niños. La aplicación principal sería la del

módulo de selección de formas y colores como herramienta educativa de los conceptos básicos de geometría y colores.

- ***Simulador de Cirugía***: Para campos médicos, el sistema podría desarrollarse como un simulador de cirugía para que los estudiantes practiquen movimientos precisos y coordinación durante procedimientos quirúrgicos.
- ***Aplicaciones en Terapia Ocupacional***: En el ámbito de la terapia ocupacional, el sistema podría ser adaptado para ayudar a las personas con discapacidades físicas a mejorar sus habilidades motoras y su independencia.