INFORME PROYECTO VISIÓN

Enrique Fernández-Baíllo Rodríguez de Tembleque

Jacobo Banús Bertram

Introducción

El proyecto consiste en tres partes:

Calibración: En la que se obtienen los parámetros intrínsecos, extrínsecos, los coeficientes de distorsión y el root mean squared reprojection error de la cámara para la posterior corrección de la distorsión de la cámara.

Sistema de seguridad: Donde hay que mostrar a la cámara una secuencia de 4 figuras, y solo se podrá acceder a la siguiente fase del programa si la secuencia mostrada es correcta.

Tracker: Nuestro sistema basado en un tracker consiste en una ventana en negro, en la cual se puede pintar si se mueve cualquier objeto delante de la cámara. El programa seguirá al objeto e irá trazando una línea en blanco en la pantalla proporcional a la posición donde se vaya encontrando el objeto seguido por la cámara. La línea se trazará mientras se presione el click izquierdo en el ratón, para así permitir saltos en el trazo y pintar formas más complejas.

Metodología

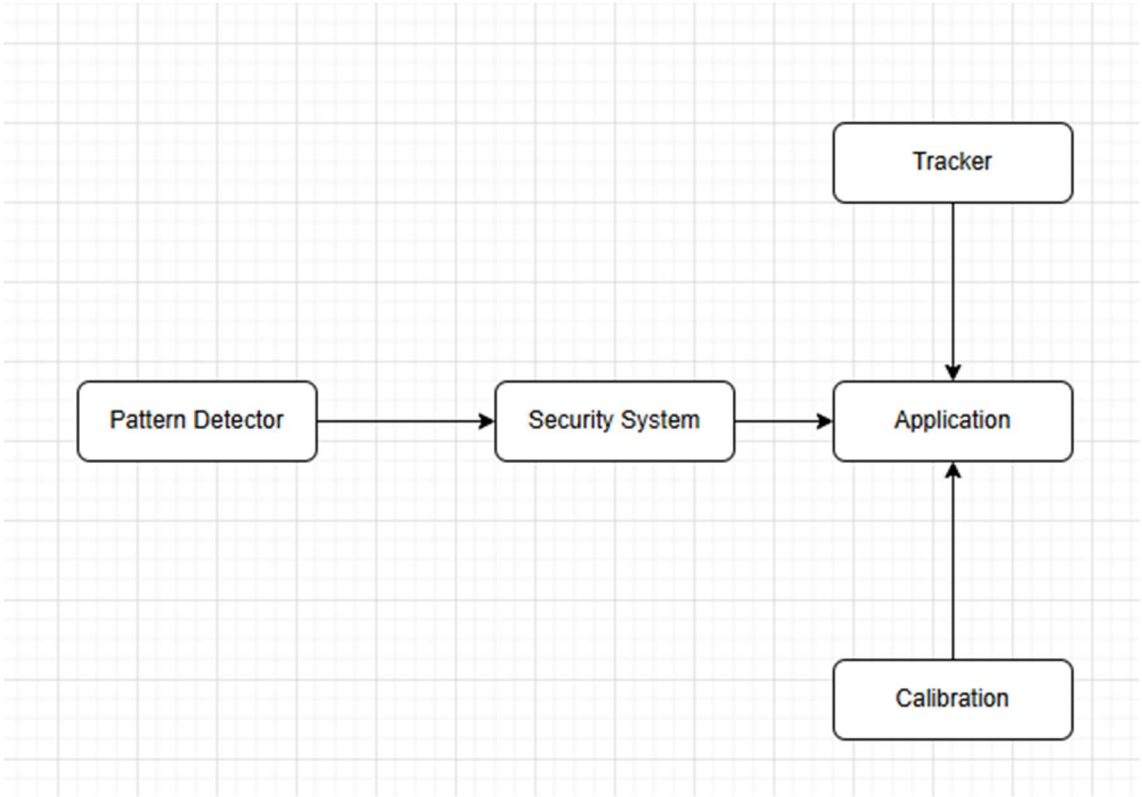
El script test.py cumple la función de comprobar rápidamente el correcto funcionamiento de la cámara.

La calibración de la cámara se encuentra en el script calibration.py. Para calibrar la cámara, se hizo un script llamado photos.py y se hicieron 16 fotos a un tablero de ajedrez desde distintos ángulos. Estas fotos se guardaron en la carpeta data/calibration_images. La cámara se calibró leyendo estas imágenes mediante imageio.imread, y se implementó el siguiente código:

```python
def get_chessboard_points(chessboard_shape, dx, dy):
    points = []
    for i in range(chessboard_shape[1]):
        for j in range(chessboard_shape[0]):
            points.append([i*dx, j*dy, 0])
    return np.array(points, np.float32)

path = '../data/calibration_images/'
imgs_path = [path + file_name for file_name in os.listdir(path)]
imgs = load_images(imgs_path)
imgs_copy = copy.deepcopy(imgs)

corners = [cv2.findChessboardCorners(img, (7, 7)) for img in imgs]
corners_for_calibration = [corner[1] for corner in corners]

imgs_gray = [cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) for img in imgs]
imgs_copy_gray = copy.deepcopy(imgs)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.01)

imgs_corners = [cv2.drawChessboardCorners(img, (7, 7), cor[1], cor[0]) for img, cor in zip(imgs_copy_gray, corners)]

for i, img in enumerate(imgs_corners):
    show_image(img)
    write_image(i, img, 'corners_detected')

chessboard_points = [get_chessboard_points((7, 7), 30, 30)] * len(imgs_gray)

rms, intrinsics, dist_coeffs, rvecs, tvecs = cv2.calibrateCamera(chessboard_points, corners_for_calibration, imgs_gray[0].shape[::-1], np.zeros((3,3)), np.zeros((1,4)))

# Obtain extrinsics
extrinsics = list(map(lambda rvec, tvec: np.hstack((cv2.Rodrigues(rvec)[0], tvec)), rvecs, tvecs))
```

Encontrando primero las esquinas con cv2.findChessboardCorners, mostrándolas con cv2.drawChessboardCorners, y calibrando la cámara haciendo uso de las esquinas, las imágenes en escala de grises y la función get_chessboard_points definida. Para encontrar los extrínsecos, se usó el método cv2.Rodrigues. Los parámetros resultantes se encuentran almacenados en output/calibration_results.txt.

El diagrama de cajas es el siguiente:



Por lo que respecta a la secuencia de transformación de la imagen, podemos diferenciar principalmente entre el reconocimiento de patrones y el seguimiento de objetos. En el reconocimiento de patrones, en primer lugar, debemos pasar la imagen a escala de grises, como es común en este tipo de procedimientos. Además, realizamos un suavizado

gausiano para reducir el ruido de la imagen, y es esta imagen la que pasamos al detector shiTomasi. Las esquinas detectadas se dibujan encima de la imagen.

Por otro lado, para el tracker, en primer lugar, giramos la imagen, para que la posición de la mano en la mesa coincida con la del tracker en la imagen. Seguidamente, aplicamos la sustracción de fondo con un modelo de mixtura de gausianas . Además, aplicamos el operador morfológico de apertura (erosión + dilatación) a la imagen para que los contornos sean mas fáciles de detectar.

En cuanto al sistema de seguridad, este parte de una contraseña de cuatro formas, en este caso cuadrado-triángulo-cuadrado-pentágono. La introducción de la contraseña por parte del usuario procede de la siguiente manera: Para cada parte de la contraseña, se debe situar la forma (dibujada en un papel, por ejemplo) delante de la camara, esperar a que el porcentaje en pantalla llegue al 100%, y pasar a la siguiente forma. En caso de que las 4 formas coincidan con la contraseña, se pasará a la siguiente parte, y, en caso contrario, se debe volver a introducir. Internamente, durante la exposición de cada forma (que dura hasta que el porcentaje para esa parte de la contraseña llegue al 100%, el programa cuenta las esquinas de cada frame mediante el detector shi Tomasi, guarda el numero en una lista, y se queda con la moda de la esa lista. De esta forma, obtenemos una medida muy robusta para introducir la contraseña de manera fiable.

Por lo que respecta al tracker, como ya hemos explicado, primero se invierte la imagen para concidir con la localización física de los objetos, y se aplica una sustracción de fondo mediante una mixtura de gausianas. Aplicamos además el operador morfológico de apertura para facilitar la detección de contornos. El seguimiento se realiza con el filtro de Kalman, que busca predecir la trayectoria del objeto en la imagen y se actualiza constantemente.

El “paint" es una aplicación directa de esto: El filtro de Kalman nos permite calcular la posición del objeto seguido en la imagen, con lo que creamos un pequeño programa que pone la pantalla en negro y dibuja en el punto correspondiente a la posición del objeto seguido. Para empezar a dibujar, basta con mantener pulsado el click izquierdo del ratón y mover el objeto (la mano, por jemplo) delante de la cámara.

Resultados

Los parámetros obtenidos durante la calibración indican que la distorsión de la cámara era mínima, al obtener unos coeficientes de distorsión cercanos a 0, y se llevó a cabo de forma precisa, ya que el root mean squared reprojection error es de 0.65, lo que indica que la cantidad de fotos y la variedad de perspectivas entre ellas fue suficiente para una calibración adecuada.

El sistema de seguridad funciona de forma eficiente si se le presenta una forma geométrica bien definida y se acerca el papel lo suficiente a la cámara para que ocupe todo el campo de visión, evitando que se detecten esquinas adicionales y evitando el desenfoque de la forma debido a la distancia respecto a la cámara. También ayuda una iluminación uniforme, aunque tolera cierto nivel de sombras, como se puede comprobar en el vídeo de demostración, donde la iluminación no era perfecta.

Por último, el tracker detecta eficientemente cualquier objeto en movimiento frente a la cámara. Se intentó que el fondo fuese lo más despejado posible y se probó moviendo la mano frente a la cámara y también moviendo un cargador de Raspberry Pi, tirando del cable para que el tracker detectase la cabeza del enchufe en movimiento. Ambos funcionaron correctamente. En el video de demostración se muestra cómo, si se procede suficientemente despacio como para que la cámara siga al objeto con la suficiente precisión, se es capaz de pintar formas complejas. En este caso, Enrique escribió el nombre Enrique.

Futuros desarrollos

Inicialmente, el proyecto iba a tratar de poder usar la mano, o cualquier objeto, como alternativa al ratón del ordenador, moviéndolo conforme se moviese el objeto frente a la cámara. Por limitaciones de software de la Raspberry Pi, ya que no funciona la interfaz gráfica de la librería estándar que permite desplazar el ratón a través de la pantalla (pyautogui), la implementación final ha sido la explicada anteriormente. De todas formas, sería muy sencillo implementar la idea original con el código desarrollado y, si se pudiese intentase que la cámara siguiese un objeto concreto, esto podría ser más preciso y más rápido, suponiendo una alternativa al mouse relativamente eficiente.