Práctico: Medición de objetos usando CV

Visión por Computadoras

Afur Matías y Santamarina Gaspar

2018

1. Introducción

El objetivo de este trabajo es diseñar un programa que permita realizar la medición en escala real de objetos que se encuentran sobre un plano conocido. Para ello se utiliza un método donde no es necesario que el plano de medición se encuentre paralelo al palno focal de la cámara. Esto se lleva a cabo usando un patrón de calibración en el plano de medición, permitiendo que la cámara pueda moverse libremente mientras realiza mediciones. El script es desarrollado en Python 3.0, utilizando la librería OpenCV 4.1.0.

2. Desarrollo

Para la medición es necesario primero hallar el patrón de calibración en la escena y efectuar una transformación perspectiva para traerlo al plano focal. Una vez obtenido esto, se buscan contornos en la imagen que puedan representar a los objetos que se están por medir. A estos contornos se les dibuja un rectángulo en sus límites y se mide el alto y ancho del objeto. En el script se han definido dos funciones que realizan estas tareas.

2.1. Búsqueda y transformación: función *rectify()*

Se definió la función *rectify(frame)* a la que se le pasa como argumento el cuadro de video (*frame*) en el cual se efectúa la búsqueda del plano de medición y la transformación de perspectiva. Esta función devuelve un valor booleano y el cuadro (*True* y el cuadro rectificado o *false* y el cuadro sin modificaciones).

Para hallar el plano de medición se utilizó un patrón de calibración tipo tablero de ajedrez de 4x5 como el de la figura 1, y se llama a la función *cv2.findChessboardCorners(image, patternSize, corners, flags)* a la que se le debe pasar como argumentos, la imagen donde debe buscar, el tamaño del patrón (esquinas internas), la matriz de salida con las esquinas encontradas y *flags* de configuración. Una vez encontradas las esquinas se refina la ubicación de estas mediante *cv2.cornerSubPix()* y se las dibuja con *cv2.drawChessboardCorners()*.

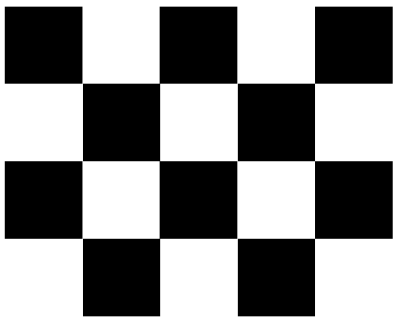


Figura 1: Patrón de calibración 4x5 o 3x4 esquinas internas

Los puntos encontrados pertenecientes a las esquinas son mapeados en un rectángulo de dimensiones conocidas en *píxeles* que respetan la relación *ancho/alto* del patrón. Para obtener la matriz de transformación se llama a la función *cv2.getPerspectiveTransform(src, dst)*, donde los argumentos son los vectores de puntos de las esquinas halladas y los puntos del rectángulo donde se las desea mapear, respectivamente. Esta matriz se pasa como argumento a la función *cv2.warpPerspective(drawedFrame, M, imgSize)* la cual devuelve la imagen rectificada.

Código python:

```
def rectify(frame):
global n, h, v

drawedFrame = frame.copy()
grayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # Escala de grises
ret, corners = cv2.findChessboardCorners(grayFrame, chessboardSize, None, cv2
    .CALIB_CB_FAST_CHECK)

if ret is True:
    refinedCorners = cv2.cornerSubPix(grayFrame, corners, (11, 11), (-1, -1),
          criteria)
    cv2.drawChessboardCorners(frame, (chessboardSize), refinedCorners, ret)
    imgSize = (frame.shape[1], frame.shape[0])

    # Puntos a mapear
    src = np.float32([corners[0], corners[3], corners[8], corners[11]])
    dst = np.float32([[508, 0], [640, 0], [508, 88], [640, 88]])

    # Matriz de transformacion
    M = cv2.getPerspectiveTransform(src, dst)
    rectifiedFrame = cv2.warpPerspective(drawedFrame, M, imgSize)

    return True, rectifiedFrame

else:
    return False, frame
```

2.2. Contornos: función *getContours()*

Una vez hecha la transformación de perspectiva de la imagen se deben ubicar los objetos a medir en al escena. La estrategia para encontrarlos aquí es utilizar contornos. Para esto se define *getCountours(frame)* la cual realiza los siguientes pasos:

1. El cuadro o imagen se pasa a escala de grises.
2. Se le aplica un filtro para difuminar el ruido.
3. Se le aplica un límite binario para pasar a blanco y negro y sea mas fácil la detección de los contornos.
4. Se llama a la función *cv2.findContours(image, mode, method)* la cual devuelve *contours* y *hierarchy*, donde *contours* son los contornos detectados

almacenados en un vector de puntos y *hierarchy* contiene información a cerca de la topología de la imagen.

5. Por último se dibujan los contornos en la imagen mediante la función *cv2.drawContours(image, contours, contourldx, color, thickness).*

Código python:

```
def getContours(frame):
    global threshold

    # Escala de grises
    imgray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Filtro blur para disminuir ruido
    imblur = cv2.blur(imgray, (10,10))

    # Threshold para pasar la imagen a B/N
    ret, imthresh = cv2.threshold(imblur, threshold, 255, cv2.THRESH_BINARY)

    # Parche
    cv2.rectangle(imthresh, (400, 0), (640, 150), (255,255,255), -1)

    # Calculo de contornos
    contornos, hierarchy = cv2.findContours(imthresh, cv2.RETR_TREE, cv2.
        CHAIN_APPROX_SIMPLE)
    cv2.drawContours(frame, contornos, -1, (0,255,0), 3)

    return contornos, frame
```

2.3. Medición

Para medir el objeto se obtiene un rectángulo al rededor de su contorno mediante *cv2.boundingRect(contour).* Esta función devuelve las coordenadas *x, y* y el ancho y alto del rectángulo en *píxeles.* Estas mediciones se dibujan en la imagen como lineas al rededor del objeto mediante *cv2.line()* y su valor con *cv2.putText()* previamente transformada a *mm* mediante algún factor *mm/píxel.*

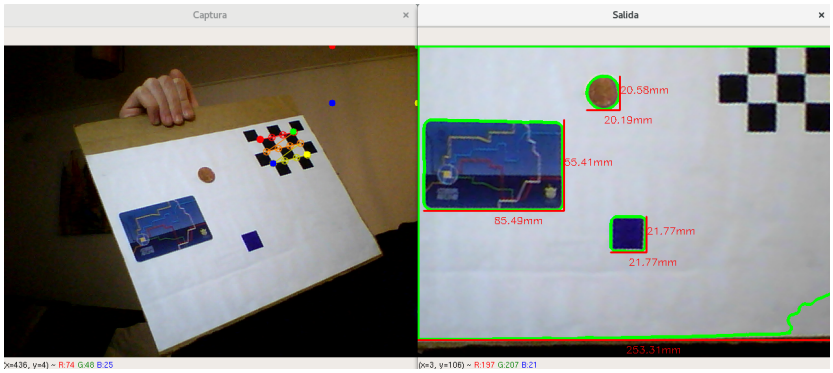
A continuación se muestra el código y las imágenes de las mediciones.

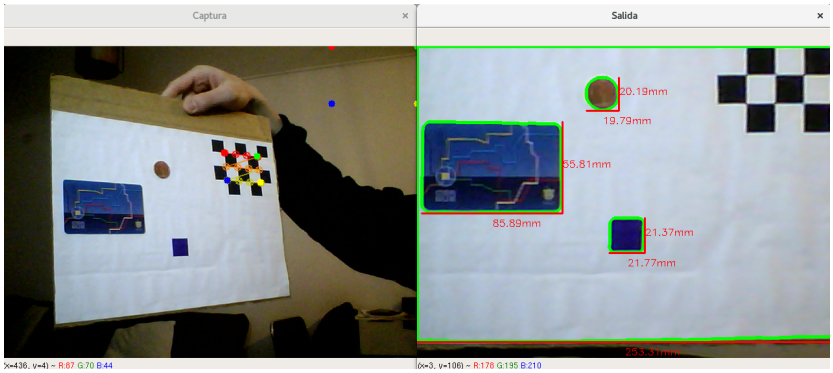
Código python:

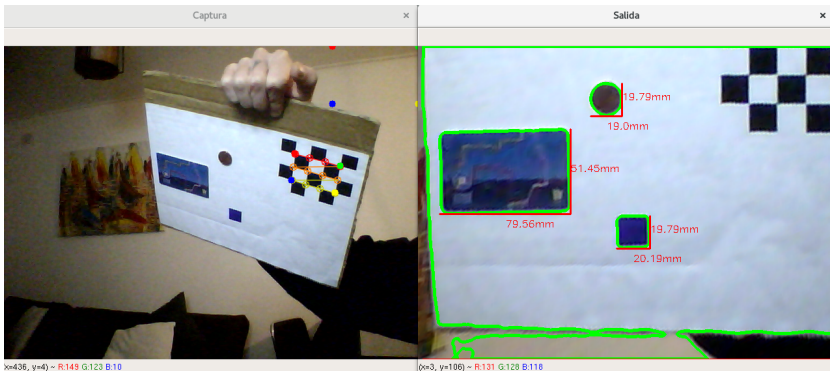
```
valid, rectifiedFrame = rectify(frame)

if(valid):
    contornos, drawedFrame = getContours(rectifiedFrame.copy())

    # Medicion
    for contorno in contornos:
        x,y,contourW,contourH = cv2.boundingRect(contorno) # Rectangulo
            dentro del cual esta comprendido el objeto a medir

        cv2.line(drawedFrame, (x, y+contourH), (x+contourW, y+contourH),
            (0,0,255), 2) # Ancho
        cv2.line(drawedFrame, (x+contourW, y), (x+contourW, y+contourH),
            (0,0,255), 2) # Alto

        # Paso a mm
        contourW_mm = round(mmpx * contourW, 2)
        contourH_mm = round(mmpx * contourH, 2)
```

```
            # Etiquetas
            cv2.putText(drawedFrame, str(contourW_mm)+"mm", (round(x+contourW/2),
                  y+contourH+20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255)
            cv2.putText(drawedFrame, str(contourH_mm)+"mm", (x+contourW, round(y+
                 contourH/2)), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))

    else:
        drawedFrame = frame.copy()
```

Imágenes:







3. Conclusión

A modo de concluir, se puede nombrar las ventajas y desventajas de este método y como se podría mejorar el algoritmo. Una desventaja que se presentó fueron las sombras. Al tratarse de un plano móvil este genera distintas sombras, de acuerdo a su orientación, sobre los objetos a medir. Esta sombra modifica las mediciones, ya que de acuerdo al nivel de *threshold,* la sombra pasa a formar parte de las dimensiones del objeto alterando las mediciones. La iluminación debe ser uniforme y pareja en el plano, para que este efecto no influya drásticamente en los errores de medición o también se podrían promediar las mediciones.

Otra desventaja que se observa en las imágenes es que las mediciones varían de acuerdo a la orientación, esto es debido al propio error de la transformación perspectiva. Una de las mejoras posibles para solucionar este efecto es la de realizar la búsqueda de los contornos antes de efectuar la transformación; se transforman los puntos de los contornos en vez de buscar los puntos en la imagen transformada.

Con respecto a las ventajas, existen algunas y son más obvias:

- No hace falta tener un plano fijo para realizar las mediciones.
- Se puede obtener mediciones de un objeto desde varias perspectivas.
- Si las condiciones de iluminación son buenas las mediciones son confiables.

Este tipo de prácticas son de suma utilidad en el mundo moderno donde la visión por computadora está en auge. Este algoritmo cumple con las pautas planteadas por la consigna y se desenvuelve correctamente, pero siempre se puede seguir aplicando mejoras. Queda claro que el uso de OpenCV es ilimitado y conforma una herramienta poderosa en constante crecimiento. Su difusión es importante para la ingeniería y una solución a problemas actuales.