TP N°9: Medición de Objetos

Universidad Tecnológica Nacional, Facultad Regional Córdoba
Cátedra: Visión Por Computadora
Profesor: Araguás, Gastón
Redolfi, Javier Integrantes: Ruiz, Dante
Gomez, José Maria
Segovia, Franco

***Resumen*—Se realiza un script en lenguaje Phyton para procesar fotografías con diferentes objetos sobre una superficie. Con el uso de la biblioteca OpenCV [1] se puede realizar las transformaciones necesarias para tener todos los objetos en la misma escala, y realizar mediciones, a partir de un objeto conocido, con un margen de error aceptable.**

I. INTRODUCCIÓN

La *visión por computadora* permite que, partiendo de imágenes tomadas por cámaras de diferentes usos, y aplicando teoría de álgebra y óptica geométrica, se pueda obtener información útil para interpretar el mundo real, tanto por personas como por autómatas [3].

Dentro de las aplicaciones más comunes, se encuentra el reconocimiento y conversión de texto en una imagen, la identificación y seguimiento de objetos o personas en un video y las mas recientes aplicaciones de aprendizaje automático y asistencia en el manejo de automóviles [4].

Fig. 1: Reconocimiento de personas, basado en openCV [2]

En este práctico nos centraremos en el uso de propiedades geométricas de la proyección de planos, para obtener mediciones a partir de una referencia en la imagen. Tomaremos una imagen de un plano sobre la cual se afirma una figura de tamaño pre-establecido. Haciendo uso de esta figura como patrón, mediremos la distancia mas exacta posible que nace de la selección de dos puntos cualquiera que se encuentren en la imagen.

I-A. Marco teórico

I-A1. Transformación proyectiva: La geometría proyectiva equivale a la proyección sobre un plano de un subconjunto del espacio en la geometría euclidiana tridimensional. Las rectas que llegan al ojo del observador se proyectan en puntos. Los planos definidos por cada par de ellas se proyectan en rectas [6].

Una recta sobre un plano puede ser representado por el vector **l**, donde cada componente se corresponde con uno de los coeficientes de la ecuación general:

$$l = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} a & b & c \end{bmatrix}^T$$

Sin embargo, esta misma recta puede ser representada de infinitas formas. Si al vector asociado a la recta se lo multiplica por $\lambda \neq 0$, obtenemos un conjunto de vectores equivalentes, llamados homogéneos.

$$l = \begin{bmatrix} \lambda a \\ \lambda b \\ \lambda c \end{bmatrix} = \begin{bmatrix} \lambda a & \lambda b & \lambda c \end{bmatrix}^T$$

Resulta evidente que este mismo concepto se puede extender a los puntos que conforman tal recta. En efecto, si un punto $\mathbf{x} = (x_1, y_1)$ pertenece a una recta l, se cumple que:

$$\lambda\, a\, x_1 + \lambda\, b\, y_1 + \lambda\, c = 0$$

Si consideramos una recta perteneciente a un plano $\mathbb{P}^2$, y un punto perteneciente a esa recta (y al plano), el producto escalar entre ambos es nulo.

$$\mathbf{l} \cdot \mathbf{x} = \mathbf{0} \tag{1}$$

Si calculamos el producto vectorial de dos rectas perpendiculares entres si, que pertenezcan al mismo plano $\mathbb{P}^2$, el resultado es un vector perpendicular a ambas. Si llamamos $\mathbf{x}$ a ese vector:

$$\mathbf{x} = \mathbf{l_1} \times \mathbf{l_2} \tag{2}$$

Luego reemplazando 2 en 1:

$$\mathbf{l_1} \cdot (\mathbf{l_1} \times \mathbf{l_2} = \mathbf{l_2} \cdot (\mathbf{l_1} \times \mathbf{l_2} = 0 \tag{3}$$

Entonces $\mathbf{x}$ es el punto de intersección de dos rectas perpendiculares.

De igual forma,s e puede definir $\mathbf{l} = x_1 \times x_2$ como la recta que pasa por los puntos x_1 y x_2.

En el caso particular que $\lambda = 0$, tendremos puntos ideales, pertenecientes a la recta en el infinito.

$$\mathbf{x}^T \mathbf{l}_\infty = 0 \tag{4}$$

Donde:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & 0 \end{bmatrix}^T \tag{5}$$

$$\mathbf{l}_\infty = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{6}$$

Todos los puntos de intersección de rectas paralelas, pertenecen a l_∞, por lo tanto la recta en el infinito es el conjunto de todas las direcciones de las rectas de $\mathbb{R}^2$

Una transformación proyectiva, es un mapa que permite llevar un conjunto de puntos:

$$\{x_1, \ldots, x_n\} = \mathbb{X} \in \mathbb{R}^2$$

pertenecientes a una recta, a otro conjunto

$$h_1, \ldots, h_n = \mathbb{H} \in \mathbb{R}^2$$

que también pertenecen a una recta.

Si la homografía es lineal, entonces se puede representar el mapa como el producto por una matriz de transformación. En el caso de $\mathbb{R}^2$, se trata de una matrz de 3×3 invertible.

$$h_i = H x_i$$

De los tipos de transformaciones lineales invertibles, la de mayor cantidad de grados de libertad, es la transformación proyectiva. La matriz de transformación está compuesta por 4 elementos correspondientes a la transformación afín, y elemento relacionado a la translación, y un vector denominado vector de perspectiva.

$$h_i = H x_i == \begin{bmatrix} A & t \\ \nu^T & 1 \end{bmatrix}$$

I-B. *Uso de OpenCV*

OpenCV permite obtener la matriz de transformación proyectiva, a partir de cuatro puntos identificados en una imagen, y sus correspondientes cuatro puntos en la imagen transformada (fig. 2) [5].

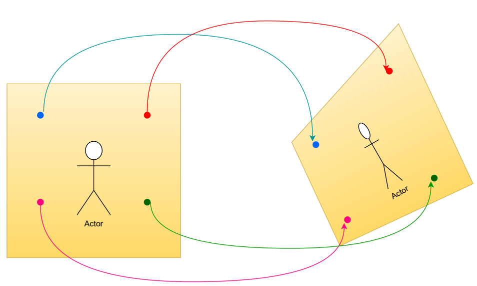

Fig. 2: Proyección de cuatro puntos de una imágen

De esta forma, se puede saber cual fue la transformación que sufrió la imagen y realizar la corrección necesaria sobre el resto de los puntos.

I-C. *Rectificación completa*

Consiste en transformar la imágen (fig. 3), de manera que queden dos conjuntos de rectas paralelas, y perpendiculares entre sí (una grilla rectangular).

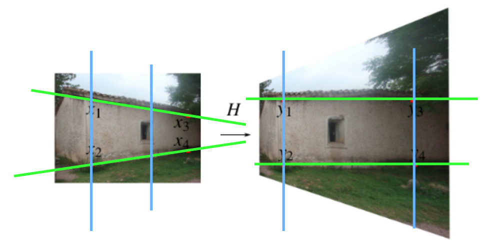

Fig. 3: Rectificación de una imagen, mediante proyección lineal de cuatro puntos

II. Desarrollo

Se plantea un programa que permita medir longitudes, a partir de un objeto de dimensiones conocidas, sobre un plano.

La resolución se desarrolla en dos partes, la primera es obtener la imagen rectificada, y la segunda es la medición propiamente dicha.

Son necesarias algunas condiciones para que se puedan obtener buenos resultados:

- Que todos los objetos se encuentren en un mismo plano, y que sean relativamente planos, y de poca altura con respecto a la imagen.
- Que el ángulo de donde se toma la fotografía con respecto a la normal al plano no sea demasiado grande.
- Contar en el plano con una figura u objeto cuadrado, lo mas plano posible (una hoja o un dibujo sobre el mismo plano), con la dimensión de sus lados conocida.
- Que la dimensión conocida no sea muy pequeña en relación a la imagen.

Para que se pueda corroborar mas fácilmente la rectificación, se utilizó una hoja cuadriculada A4, donde se marcaron cuatro puntos, donde se marcaron cuatro puntos, que describen un cuadrado de $80mm$ de lado (fig. 4).

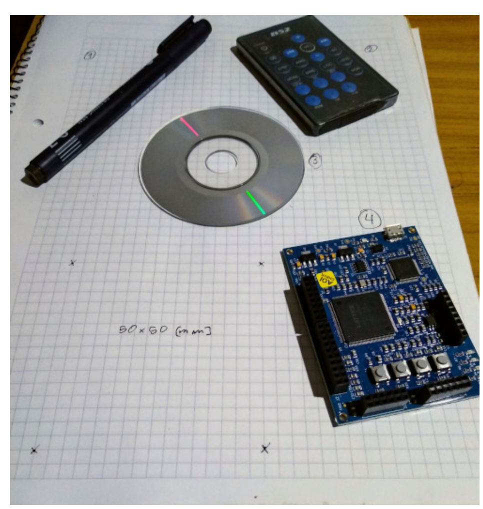

Fig. 4: Imagen Original

Primeramente, se realiza la rectificación, con el uso de las funciones **getPerspectiveTransform()** y **warpPerspective()** de openCV [7].

La primera obtiene la matriz de transformación, a partir de cuatro pares de puntos en la imagen original y sus correlativos en la imagen corregida, que debería ser un cuadrado perfecto. Para obtener los puntos de la imagen original, se uso una variante del programa del Práctico 8, donde se marcaron con el mouse los 4 puntos y se obtuvieron 4 pares coordenados.

La segunda función aplica la transformación utilizando la matriz, la imagen y las dimensiones deseadas en la imagen de salida. Fue necesario aplicar un escalamiento y un offset a la imagen para que el resultado abarque las zonas de interés.

El imagen rectificada se puede ver en la imagen 5.

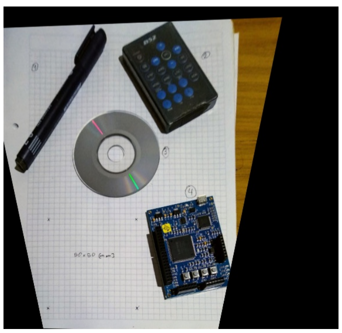

Fig. 5: Imagen Rectificada

Finalmente, para la medición, se utilizó una función que captura los eventos del ratón. Esta función registra las coordenadas de dos puntos donde se haga click izquierdo. Luego se calcula la distancia euclídea con la ecuación pitagórica. Antes de presentar el resultado, se vuelve a convertir utilizando el factor de escala.

A continuación se presenta el código y luego se presentan los resultados obtenidos.

II-A. Código

```
import cv2
import numpy as np
import math

## Variables
l = 80      # lado del cuacrado en mm
esc = 2    # factor de escala pixels/mm
c = l*esc   # lado en pixels
osx =  50 #offset_x
osy =  50 #offset_y

p,x1,y1,x2,y2,i,medida=0,0,0,0,0,0,0
menu = "MENU: (m) medir  (g) guardar  (q) salir"

# Puntos de referencia obtenidos con tp8 sobre la imagen
puntos = np.float32([[ 27 , 465 ],[ 66 , 266 ],[ 268 , 267 ],[ 270 , 463 ]]);

xd = puntos[0,0]+osx    # corrimiento para evitar perdida de margenes
yd = puntos[0,1]+osy

# puntos de destino del mapeo
destino = np.float32([[ xd , yd ],[ xd , yd-c ],[ xd+c , yd-c ],[ xd+c , yd]]);

## Funciones

# corrige proyeccion
def homografia(image,puntos,destino):
     (h, w) = image.shape[:2]
     M=cv2.getPerspectiveTransform(puntos,destino)
     rect=cv2.warpPerspective(image,M,(h+osy,w+osx))
     return rect

# marca puntos de medicion
def selec_p(event,x,y,flag,params):
  global p,x1,y1,x2,y2,long
  if event==cv2.EVENT_LBUTTONDOWN:
    cv2.circle(img_transf,(x,y),3,(255,0,0),-1)
    cv2.imshow('Transformada',img_transf)
  if event==cv2.EVENT_LBUTTONUP:
    if p==0:
      (x1,y1)=(x,y)
    if p==1:
      (x2,y2)=(x,y)
      long=np.float32(math.sqrt((x2-x1)**2 + (y2-y1)**2)/esc)
      print("La longitud es ",long)
      print()
    p += 1

# auxiliar - suspende eventos del mouse
def dummy(event,x,y,flag,params):
   event==cv2.EVENT_LBUTTONDOWN
   event==cv2.EVENT_LBUTTONUP

# imprime texto en imagen a diferente altura
def texto(image,text,posy):
   fuente = cv2.FONT_HERSHEY_SIMPLEX
   linea = cv2.LINE_AA
   colort = (0, 250, 250)
   colorb = (10, 10, 10)
   pt1 = (10,4+posy)
   pt2 = ((10+11*len(text)),25+posy)
   image = cv2.rectangle(image, pt1, pt2, colorb, -1)
   image = cv2.rectangle(image, pt1, pt2, colort, 1)
   image = cv2.putText(image , text, (10,20+posy), fuente, 0.6, colort, 1 , linea)
   return image


img1 = cv2.imread('medir_1.jpg', 1)    # Carga imagen original
img_transf = homografia(img1, puntos, destino) #corrige
(h, w) = img1.shape[:2]
img_copy = img_transf.copy()     # Crea copia limpia

img1 = texto(img1, "Imagen Original",0)
cv2.imshow('Original', img1)
cv2.waitKey(2000)

cv2.namedWindow('Transformada')
img_transf = texto(img_transf, menu,h-10)
cv2.imshow('Transformada', img_transf)
print (menu)

while(1):

    k = cv2.waitKey(100) & 0xFF #verifica teclas precionadas

    if k == ord('m'):      # mide distancia entre dos puntos seleccionados

        img_transf = img_copy.copy()
        img_transf = texto(img_transf, "Elija dos puntos",0)
        cv2.imshow('Transformada',img_transf)
        cv2.setMouseCallback('Transformada',selec_p)
        p = 0
        while (1):

              k = cv2.waitKey(1) & 0xFF
              if p == 2:
                 img_transf = cv2.line(img_transf, (x1,y1), (x2,y2), (0,255,0), 2)
                 cv2.setMouseCallback('Transformada',dummy)
                 # Imprime valor medido
                 img_transf = texto(img_transf, "Longitud: "+str(long)+" mm",0)
                 img_transf = texto(img_transf, menu,h-10)
                 cv2.imshow('Transformada',img_transf)
                 print (menu)
                 break


    elif k == ord('g'):  # Guarda imagen con la medicion
       cv2.imwrite('medida.png',img_transf)
       img_guardado = texto(img_transf, "Guardado...",h-100)
       cv2.imshow('Transformada',img_guardado)


    elif k == ord('q'): # sale del programa
        break

cv2.destroyAllWindows()
```

II-B. Resultados

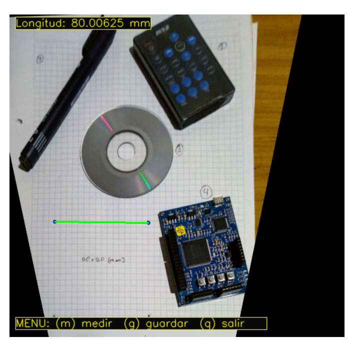

Fig. 6: Medición del lado del cuadrado patrón

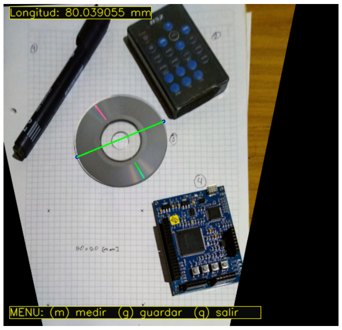

Fig. 7: Medición del diámetro de un minidisk

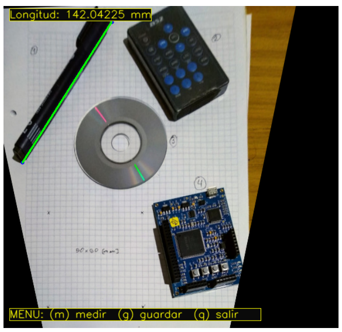

Fig. 8: Medición del largo de una lapicera

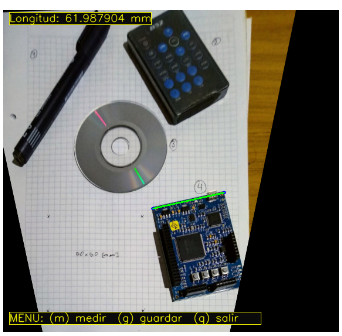

Fig. 9: Medición del ancho de una EduFPGA

III. CONCLUSIONES

Los valores medidos obtenidos han sido mas que aceptables, sobre todo teniendo en cuenta que el relevamiento de las medidas se realiza de forma manual, lo que le da mayor margen de error. Las mediciones se contrastaron con otras hechas con regla, teniendo una diferencia de $\pm 2mm$ en el peor de los casos.

Si bien el sistema resulta útil para medir objetos que se ubiquen sobre el plano de referencia, también es de resaltar que si se desea utilizar el programa para otra escena, sería necesario reescribir los valores en el código. Una mejora posible, sería inicializar con un script que habrá la imagen original, y permita marcar en ella los vértices del cuadrado de referencia, e ingresar la longitud de su lado.

También se puede suponer la facilidad de agregar la función de medir áreas, con el correspondiente incremento en las incertidumbres.

REFERENCIAS

[1] OpenCV web page
opencv.org
[2] Why Is OpenCV Gaining Prominence?, Junio 2020
www.analyticsindiamag.com
[3] OpenCV By Example. Prateek Joshi, David Millán Escrivá, Vinícius Godoy. Packt Publishing, 2016.
[4] OpenCV: Computer Vision Projects with Python. Joseph Howse, Prateek Joshi, Michael Beyeler . Packt Publishing, 2016.
[5] OpenCV-Python Tutorials Documentation. Alexander Mordvintsev, Abid K. 2017.
[6] Transformaciones en imágenes. Gastón Araguás, Javier Redolfi, 2020.
[7] Introducción a las OpenCV, Redolfi, Javier, 2020.