INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Visión Artificial

Práctica 3

’Medición de distancia y dimensiones de varios objetos’

Profesor:

Anzueto Rios Álvaro

Grupo:

3BM5

Presenta:

Aguilar Gómez Pedro Nicolás

Sanchez Sanchez Luis Alberto

Fecha

2 de mayo del 2021

Contenido

Marco teórico

Los bordes de una imagen digital se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Suministran una valiosa información sobre las fronteras de los objetos y puede ser utilizada para segmentar la imagen, reconocer objetos, etc. La mayoría de las técnicas para detectar bordes emplean operadores locales basados en distintas aproximaciones discretas de la primera y segunda derivada de los niveles de grises de la imagen.

Desarrollo

Estimación de la profundidad

Para el desarrollo de la estimación de la profundidad se utilizó una cámara web con una resolución de 640x480 en conjunto de un láser color rojo. El primer paso fue graduar una superficie en marca de 5 cm de separación, empezando en 20 cm hasta los 110 cm. Una vez graduada la mesa se procedió a calcular la variación de pixeles que existía entre el centro de la imagen con el centroide del láser.

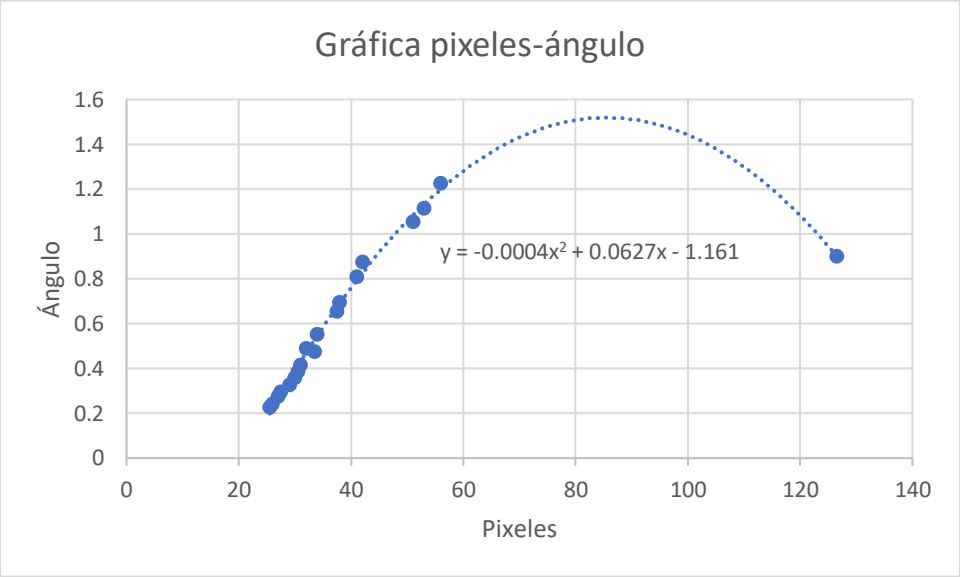
La cantidad de pixeles entre el láser y el centro de la imagen varia con respecto a la separación con la pared. Con los datos anteriores y mediante una relación trigonométrica se consiguió el valor del ángulo existente entre los catetos. La separación entre los centros es de aproximadamente 2 cm.

$$\theta = \tan^{-1}\frac{pixeles}{distancia}$$

Con dicha relación se obtuvo una tabla con los siguientes datos:

Distancia	Pixeles	Angulo
20	56	1.22777239
25	53	1.11502257
30	51	1.05572351
35	42	0.87605805
40	41	0.80978357
45	38	0.69473828
50	37.5	0.65617872
55	34	0.55368132
60	32	0.48995733
65	33.5	0.47587941
70	31	0.41689807
75	30.5	0.38624026
80	30	0.35877067
85	29	0.32879269
90	27.5	0.29654581
95	27	0.27690884
100	126.5	0.90186646
105	26	0.24273651
110	25.5	0.22779453

Con la información anterior de pixeles y ángulo se procede a realizar una regresión que nos permita obtener un polinomio que se adapte al comportamiento de los datos.



Todo esto nos permite calcular la distancia entre la cámara y la pared que posteriormente va a ser utilizada para el calculo de las medidas del objeto. El polinomio obtenido fue:

$$\theta = -0.0004x^2 + 0.0627x - 1.161$$

Esto nos permite obtener la relación

$$distancia = \frac{pixeles}{\tan\theta}$$

Código para el cálculo de la distancia

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, io, filters, color
import cv2
from PIL import Image

def colaser(filf,namec):
    cap = cv2.VideoCapture(namec)
    leido, frame = cap.read()
    if leido == True:
        cv2.imwrite('las.jpg',frame)
        imp=Image.open('las.jpg')
        imp=imp.resize((640,480))
        imp.save('las.jpg')
        print('Foto tomada exitosamente')
    else:
        print('Erro en la captura')

    cap.release()
```

```
    imal=io.imread('las.jpg')

    rojo=imal[:,:,0]-filf
    filred=filters.gaussian(rojo,sigma=10)
    binario=np.where(filred>10,1,0)
    fil1,col1=np.nonzero(binario)
    cf=((np.max(fil1)-np.min(fil1))/2)+np.min(fil1)
    cc=((np.max(col1)-np.min(col1))/2)+np.min(col1)

    return binario, cf,cc, imal

camname=0 #nombre de la camara

#como se reescala la imagen para procesamiento mas rapido podemos definir el
#centroide de la camara como a continuacion

cxc=640/2
cyc=480/2

dat=[]

for k in range (1):
    disp=input('presiona tecla para obtener el fondo:')
    for i in range(10):
        cap = cv2.VideoCapture(camname)
        leido, frame = cap.read()
        if leido == True:
            cv2.imwrite('fon'+str(i)+'.jpg',frame)
            imp=Image.open('fon'+str(i)+'.jpg')
            imp=imp.resize((640,480))
            imp.save('fon'+str(i)+'.jpg')
            print('Fondo'+str(i))
        else:
            print('Error en la captura')
    cap.release()

    fondo=np.zeros((480,640,10)) #importante ver las dimenciones de la imagen de la camara
    for frame in range(10):
        imf=io.imread('fon'+str(frame)+'.jpg')
        fondo[:,:,frame]=imf[:,:,0]
    pf=np.mean(fondo,axis=2) #promedio del fondo
    filf=filters.gaussian(pf,sigma=5) #filtro gausiano al gondo
    disp=float(input('enciende laser e ingresa la distancia en profundidad:'))
    plt.figure(4)
    plt.ion() #interactive on xD
    ival=colaser(filf,camname)
    icx=ival[2]
    icy=ival[1]
```

```
dis=np.abs(cyc-icy)
teta=np.arctan(dis/disp)

ival[0][int(cyc),int(cxc)]=1
ival[0][int(icy),int(icx)]=0
dat.append([disp,dis,teta])
plt.imshow(ival[0],cmap='gray')
print('Volvemos a iniciar')
plt.pause(2)
```

Estimación de las dimensiones de un objeto

Para la calibración en el estimado de las dimensiones de un objeto se utilizó la mesa graduada de la estimación anterior, sin embargo, se utilizo un papel ilustración con medidas de 38x25.5 cm, de igual forma se midió un cuadrado pequeño dentro del mismo papel ilustración con dimensiones de 19x12.75 cm. Otro dato utilizado fue la hipotenusa del paralelogramo de menor dimensión, teniendo un tamaño de 22.88 cm. De igual forma se midió la variación de pixeles con respecto a la distancia de la pared. El cálculo de los pixeles existentes en el ancho y alto del objeto se realizó mediante la segmentación por bordes del objeto y posteriormente de forma manual se ubicó el puntero en puntos marcados y se realizó una resta.

Distancia	Alto Ref	Pixeles Alto	Ancho Ref	Pixeles Ancho	Alto Ref2	Pixeles Alto2	Ancho Ref 2	Pixeles ancho 2	Hipotenusa	Hipotenusa cm
40	25.5	361.583	38	519.138	12.75	176.768	19	249.275	305.5895146	22.88148815
45	25.5	324.624	38	468.586	12.75	158.406	19	225.586	275.6474274	22.88148815
50	25.5	294.153	38	427.309	12.75	145.723	19	206.189	252.4858342	22.88148815
55	25.5	270.058	38	393.5	12.75	132.602	19	190.913	232.4458302	22.88148815
60	25.5	250.19	38	363.028	12.75	123.445	19	177.341	216.0752145	22.88148815
65	25.5	231.804	38	338.928	12.75	115.038	19	164.948	201.1009303	22.88148815
70	25.5	216.408	38	315.905	12.75	106.977	19	153.151	186.813563	22.88148815
75	25.5	202.612	38	294.945	12.75	100.891	19	144.512	176.2461688	22.88148815
80	25.5	189.964	38	277.826	12.75	93.221	19	135.813	164.7280359	22.88148815
85	25.5	177.264	38	262.142	12.75	87.857	19	128.345	155.5354991	22.88148815
90	25.5	167.737	38	247.106	12.75	82.776	19	121.038	146.6358197	22.88148815
95	25.5	159.888	38	236.084	12.75	79.618	19	115.229	140.0598028	22.88148815
100	25.5	151.215	38	223.341	12.75	74.694	19	109.694	132.7100873	22.88148815
105	25.5	144.564	38	214.191	12.75	72.136	19	105.692	127.9624998	22.88148815
110	25.5	138.582	38	204.964	12.75	68.068	19	100.782	121.6152299	22.88148815

Con la información se realizó una regresión múltiple con la cual se obtuvo el polinomio que nos permite estimar las dimensiones a partir de la distancia a la pared y los pixeles calculados.

*Dimensión = -17.6900330972763 + 0.274823743132625*distancia + 0.104568304334793*pixeles*

Además, se realizó una manta de estimación en 3D para conocer el plano que nos permite predecir las dimensiones de un objeto a partir de los datos mencionados anteriormente.

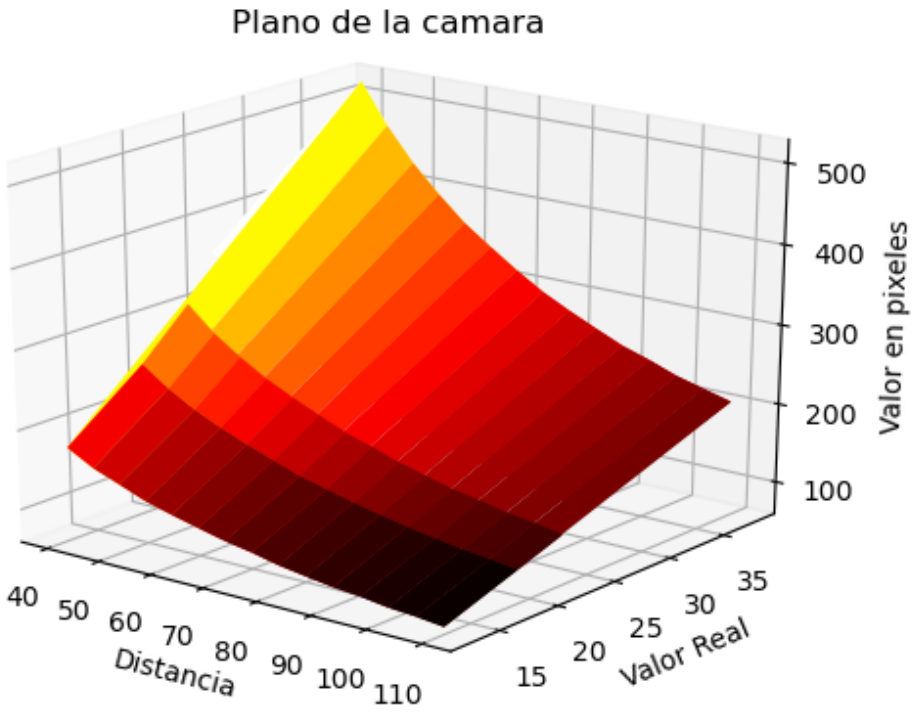


Ilustración 1 Manta de estimación de valores

Código para la estimación de las dimensiones de un objeto

```
cap = cv2.VideoCapture(0)
leido, frame = cap.read()
if leido == True:
   cv2.imwrite('foto.png',frame)
   print('Foto tomada exitosamente')
else:
   print('Error en la captura')

cap.release()
ima = io.imread('foto.png')
ima=color.rgb2gray(ima)
plt.figure(0)
```

```
plt.imshow(ima,cmap='gray')

gx = [[-1,0,1],[-1,0,1],[-1,0,1]]
gy = [[-1,-1,-1],[0,0,0],[1,1,1]]
g45 = [[-1,-1,0],[-1,0,1],[0,1,1]]
g135 = [[0,1,1],[-1,0,1],[0,-1,-1]]


imaa = img_as_float32(ima)
gxx = filters.edges.convolve(imaa,gx)
borde = np.where(np.abs(gxx)>0.5,1,0)

gyy = filters.edges.convolve(imaa,gy)
borde = np.where(np.abs(gxx)>0.5,1,0)

g455 = filters.edges.convolve(imaa,g45)
borde = np.where(np.abs(g455)>0.5,1,0)

g335 = filters.edges.convolve(imaa,g135)
borde = np.where(np.abs(g335)>0.5,1,0)

salida = np.zeros(ima.shape)
for i in range(ima.shape[0]):
   for j in range(ima.shape[1]):
      vector = np.max([np.abs(gxx[i,j]),np.abs(gyy[i,j]),np.abs(g455[i,j]),np.abs(g335[i,j])])
      salida[i,j] = vector
borde = np.where(salida>0.4,1,0)

plt.figure(1)
plt.imshow(borde, cmap = 'gray')
```

Código final para la medición de distancia y dimensiones de un objeto

El código utilizado para el desarrollo de la práctica utiliza la estimación de distancia de la cámara a la pared. Primero, calculando la cantidad de pixeles y mediante el uso del polinomio obtenido obteniendo un ángulo que será utilizado para estimar la distancia de la cámara al objeto. Todo esto se da mediante el cálculo del centroide del láser contra el punto central de la imagen, para evitar ruido se resta el fondo a la imagen y se realiza un filtrado por umbral.

La siguiente etapa consiste en tomar foto al objeto, donde mediante clics se estima la distancia primero al ancho y posteriormente a lo alto del objeto. Se pueden hacer las mediciones que uno necesite para buscar el mejor valor de pixeles para su posterior transformación a valor real. Para salir de la toma de dimensiones se realiza dos veces clic en el mismo punto. Finalmente, con el cálculo de pixeles a lo ancho y largo del objeto en conjunto con la distancia calculado con anterioridad se implementan en el polinomio de dos variables para la estimación de ambas dimensiones. Se imprimen los valores en la pantalla.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, io, filters, color
import cv2
from PIL import Image
from sklearn.cluster import KMeans
import cv2
from skimage.color import rgb2gray
from dtaidistance import dtw


#-----------------------------------------
#SECCION LASER
#-----------------------------------------

def                                                                                    labelling(ima):
##########################################################################################
##

   ai=np.zeros((ima.shape[0]+2,ima.shape[1]+2)) #se añaden dos bordes a la imagen para no
                                  #tener tantas condicionales de frontera
   for i in range(ima.shape[0]):
         ai[i+1,1:ai.shape[1]-1]=ima[i,:]

   # se realiza el primer ciclo
   sal1=np.zeros(ai.shape)
   c=2

   for i in range (1,ai.shape[0]-1,1):
      for j in range (1,ai.shape[1]-1,1):
         if ai[i,j]==1:
           vec=np.array([sal1[i-1,j-1],sal1[i-1,j],sal1[i-1,j+1],  #aqui preguntamos por los 8 vecinos
                     sal1[i  ,j-1],          sal1[i  ,j+1],  #del pixel para asignar etiqueta
                     sal1[i+1,j-1],sal1[i+1,j],sal1[i+1,j+1]])
           vec=np.where(vec<=0,1000,vec) #se sustituyen los ceros por un valor muy grande
diferente
           #print(vec)
           mn=np.min(vec) #se saca la etiqueta minima de los vecinos si es que hay
           if mn ==1000:
              sal1[i,j]=c #si el minimo es igual 100 se asigna una nueva etiqueta
           else:
              sal1[i,j]=mn #si un vecino ya tiene etiqueta, se asigna la menor
         else:
            if ai[i,j-1]!=0: #condicion que permite solo un incremento
             c=c+1

   #se realiza el segundo ciclo
```

```
   sal2=np.zeros(ai.shape)
   vp=[]
   for i in range (1,ai.shape[0]-1,1):
      for j in range (1,ai.shape[1]-1,1):
         if sal1[i,j]!=0:   #funcion solo aplicamos a los que sean diferente de cero
            vec=np.array([sal1[i-1,j-1],sal1[i-1,j],sal1[i-1,j+1],  #aqui preguntamos por los 8 vecinos
                     sal1[i  ,j-1],          sal1[i  ,j+1],  #del pixel para asignar etiqueta
                     sal1[i+1,j-1],sal1[i+1,j],sal1[i+1,j+1]])
            for k in range(vec.shape[0]): #preguntamos para cada vecino
               if sal1[i,j]!=vec[k] and vec[k]!=0: # si la etiqueta del vecino no es 0 y es diferente
                  sal1=np.where(sal1==vec[k],sal1[i,j],sal1) #se sustituyen las etiquetas los vecinos
que coincidan

   for i in range (1,ai.shape[0]-1,1):
      for j in range (1,ai.shape[1]-1,1):
         if sal1[i,j]!=0:
            if sal1[i,j] not in vp:
               vp.append(sal1[i,j])
   vp=np.array(vp)
   sal1=sal1[1:sal1.shape[0]-1,1:sal1.shape[1]-1]
   #print('sal'+str(vp.shape[0]))

   return sal1,vp
#############################################################################
###################################

def excol(name):
   ima=io.imread(name)
   plt.figure(0)
   plt.title('Da Click el color que deseas aislar y espera...')
   plt.imshow(ima)
   co=np.int32(plt.ginput(1))

   #--------------------Modelo de color RGB Kmeans--------------------
   ima33=io.imread(name)

   ima33=io.imread(name)
   ima3=color.rgb2luv(ima33)


   l=(ima3[:,:,0]) #convierte la matriz en un vector
   a=(ima3[:,:,1])
   b=(ima3[:,:,2])
   L=l.reshape((-1,1))
   A=a.reshape((-1,1))
   B=b.reshape((-1,1))
   datos3=np.concatenate((L,A,B),axis=1)
   clases=4
```

```
    salida3=KMeans(n_clusters=clases)
    salida3.fit(datos3)

    centros3=salida3.cluster_centers_
    aa2=color.lab2rgb(centros3[np.newaxis,:])
    etiquetas3=salida3.labels_ #volver a reconstruir como imagen

    for i in range (L.shape[0]): #asignar un color a cada posicion segun la etiqueta
        L[i]=aa2[0][etiquetas3[i]][0]
        A[i]=aa2[0][etiquetas3[i]][1]
        B[i]=aa2[0][etiquetas3[i]][2]

    L.shape=l.shape #redimencionar un vector a matriz
    A.shape=a.shape
    B.shape=b.shape

    L=L[:,:,np.newaxis]
    A=A[:,:,np.newaxis]
    B=B[:,:,np.newaxis]

    new3=np.concatenate((L,A,B),axis=2)
    gris = rgb2gray(new3)
    bina = np.where(gris == gris[co[0][1],co[0][0]],1,0)

    return bina

def colaser(filf,namec):
    cap = cv2.VideoCapture(namec)
    leido, frame = cap.read()
    if leido == True:
        cv2.imwrite('las.jpg',frame)
        imp=Image.open('las.jpg')
        imp=imp.resize((640,480))
        imp.save('las.jpg')
        print('Foto tomada exitosamente')
    else:
        print('Error en la captura')

    cap.release()
    imal=io.imread('las.jpg')

    rojo=imal[:,:,0]-filf
    filred=filters.gaussian(rojo,sigma=10)
    binario=np.where(filred>10,1,0)
    fil1,col1=np.nonzero(binario)
    cf=((np.max(fil1)-np.min(fil1))/2)+np.min(fil1)
    cc=((np.max(col1)-np.min(col1))/2)+np.min(col1)
```

```
    return binario, cf,cc, imal

camname=0 #nombre de la camara
#como se reescala la imagen para procesamiento mas rapido podemos definir el
#centroide de la camara como a continuacion

cxc=640/2
cyc=480/2

for k in range (1):
    disp=input('presiona tecla para obtener el fondo:')

    for i in range(10):
        cap = cv2.VideoCapture(camname)
        leido, frame = cap.read()
        if leido == True:
            cv2.imwrite('fon'+str(i)+'.jpg',frame)
            imp=Image.open('fon'+str(i)+'.jpg')
            imp=imp.resize((640,480))
            imp.save('fon'+str(i)+'.jpg')
            print('Fondo'+str(i))
        else:
            print('Error en la captura')

    cap.release()

    fondo=np.zeros((480,640,10)) #importante ver las dimenciones de la imagen de la camara
    for frame in range(10):
        imf=io.imread('fon'+str(frame)+'.jpg')
        fondo[:,:,frame]=imf[:,:,0]
    pf=np.mean(fondo,axis=2) #promedio del fondo
    filf=filters.gaussian(pf,sigma=5) #filtro gausiano al gondo

    print('Enciende el laser')
    plt.pause(10)

    plt.figure(4)
    plt.ion() #interactive on xD
    ival=colaser(filf,camname)
    icx=ival[2]
    icy=ival[1]
    dis=np.abs(cyc-icy)
    teta=-0.0004*(dis**2)+(0.0627*dis)-(1.161)
    ival[0][int(cyc),int(cxc)]=1
    ival[0][int(icy),int(icx)]=0
    plt.imshow(ival[0],cmap='gray')
    print('Continuamos...')
    fon = dis/np.tan(teta)
```

```
   print('La distancia a la pared es de: ',fon)
   plt.pause(2)
################################## captura de imagen ##################################
camname=0 #nombre de la camara

disp=input('presiona tecla para tomar la foto:')

cap = cv2.VideoCapture(camname)
leido, frame = cap.read()
if leido == True:
   cv2.imwrite('medicion.jpg',frame)
   imp=Image.open('medicion.jpg')
   imp=imp.resize((640,480))
   imp.save('medicion.jpg')
   print('capturado')
else:
   print('Error en la captura')

cap.release()

ima=io.imread('medicion.jpg')
pix=[]
x=5
while (x<10):
   plt.figure(0)
   plt.imshow(ima)
   plt.pause(0.5)
   pM=np.int32(plt.ginput(2))

   disd=np.sqrt((pM[0][0]-pM[1][0])**2+(pM[0][1]-pM[1][1])**2)
   pix.append(disd)
   #Nota: para romper el ciclo realiza la medicion en el mismo lugar
   if disd==0:
      break
   print(disd)

   #Dibujando lineas
   cv2.line(ima,(pM[0][0],pM[0][1]),(pM[1][0],pM[1][1]),(255,0,0),2)
   cv2.waitKey(0)
   cv2.destroyAllWindows()

Ancho = -17.6900330972763 + 0.274823743132625*fon + 0.104568304334793*(pix[0])

Alto = -17.6900330972763 + 0.274823743132625*fon + 0.104568304334793*(pix[1])
print('La distancia a la pared es: ',fon)
print('El ancho del objeto es: ',Ancho)
print('El alto del objeto es: ',Alto)
```

Resultados

Los objetos propuestos para la prueba del sistema fueron un estuche de xbox, un Nintendo Ds y un libro. Se realizó una prueba para el DS y para el libro, mientras que para el estuche se realizaron dos para verificar el funcionamiento del sistema. Las dimensiones y las distancias propuestas fueron:

Ilustración 2 Distancia de la cámara al estuche (50 cm)

Ilustración 3 Distancia de la cámara al Nintendo DS (40 cm)

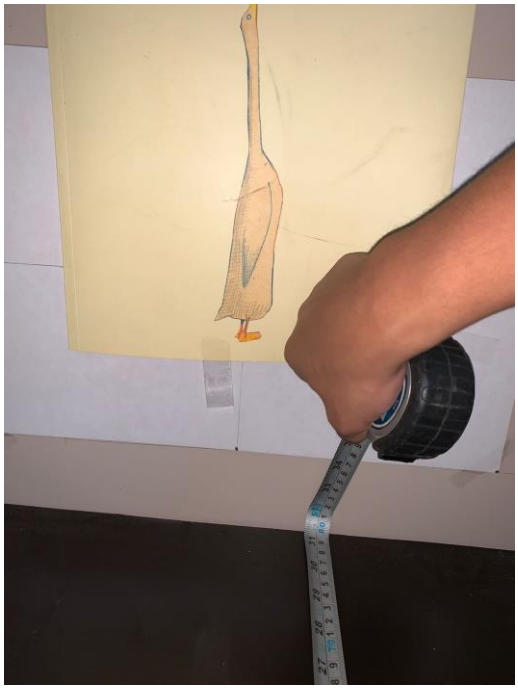


Ilustración 4 Distancia de la cámara al libro (80 cm)

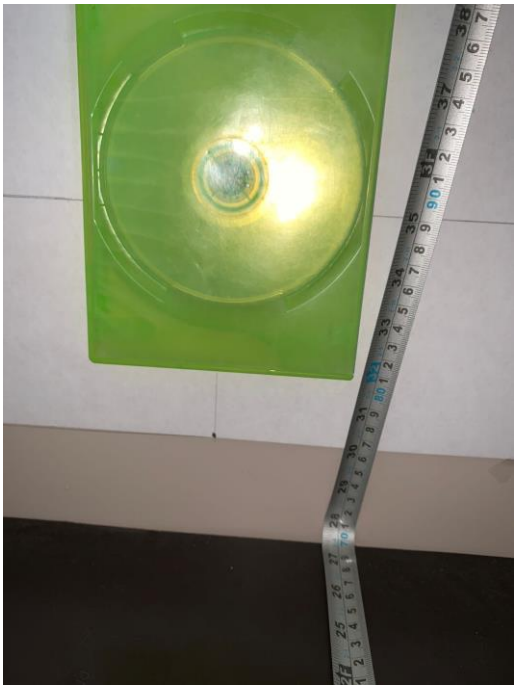


Ilustración 5 Segunda distancia al estuche (70 cm)

Prueba 1: estuche disco XBOX

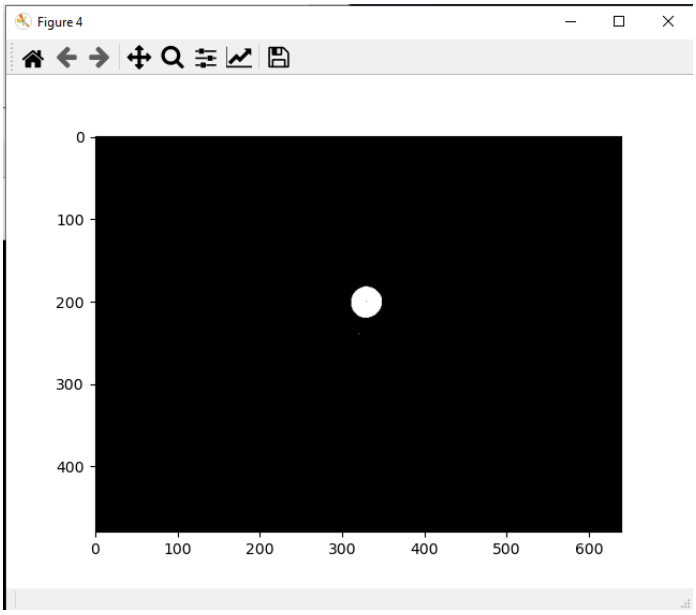


Ilustración 6 Pixeles entre el láser y el centro de la imagen

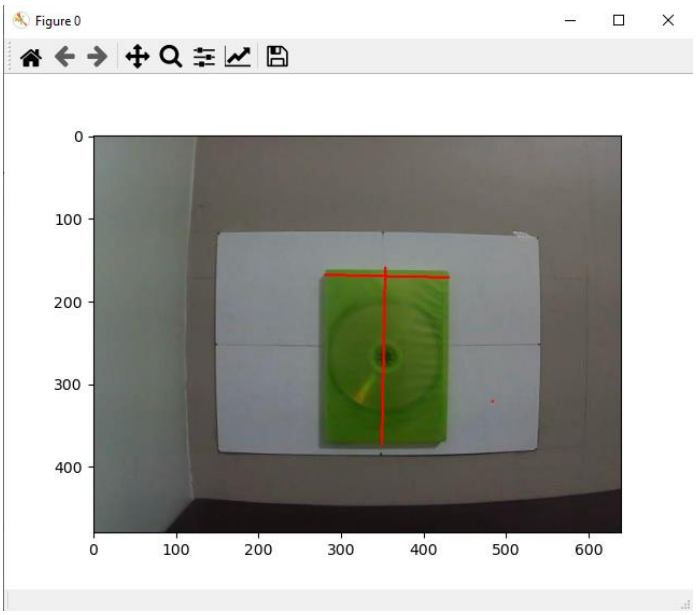


Ilustración 7 Toma de medidas por píxel del objeto

```
La distancia a la pared es:  50.61404092983709
El ancho del objeto es:  11.803742215165878
El alto del objeto es:  18.07868494015686
```

Ilustración 8 Resultados obtenidos

Prueba 2: Nintendo DS

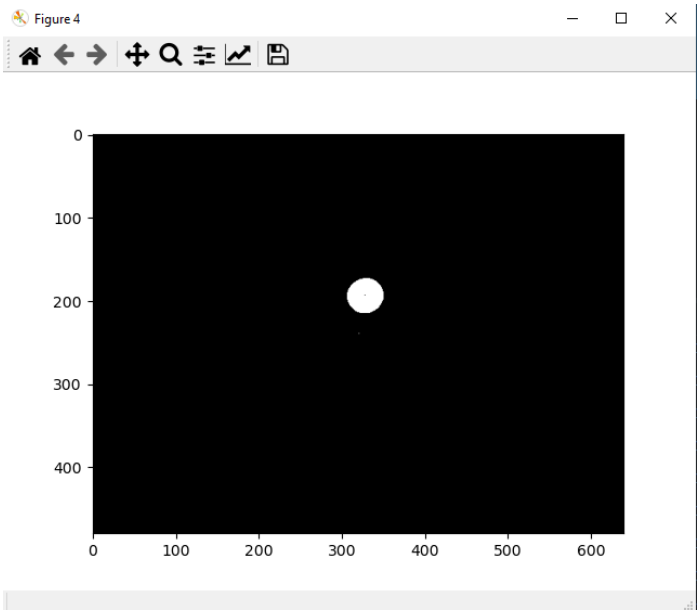


Ilustración 9 Pixeles entre el láser y el centro de la imagen

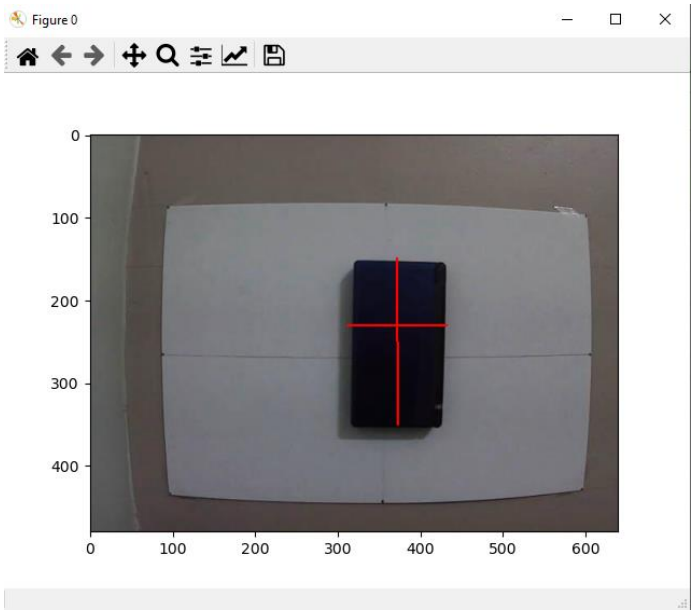


Ilustración 10 Toma de medidas por píxel del objeto

```
La distancia a la pared es:  38.87546571749132
El ancho del objeto es:  5.542064427403865
El alto del objeto es:  14.01235719707254
```

Ilustración 11 Resultados obtenidos

Prueba 3: libro

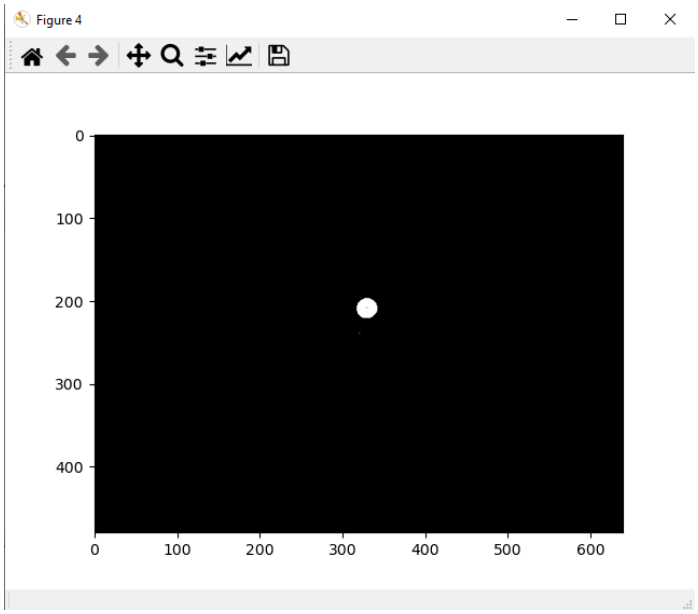


Ilustración 12 Pixeles entre el láser y el centro de la imagen

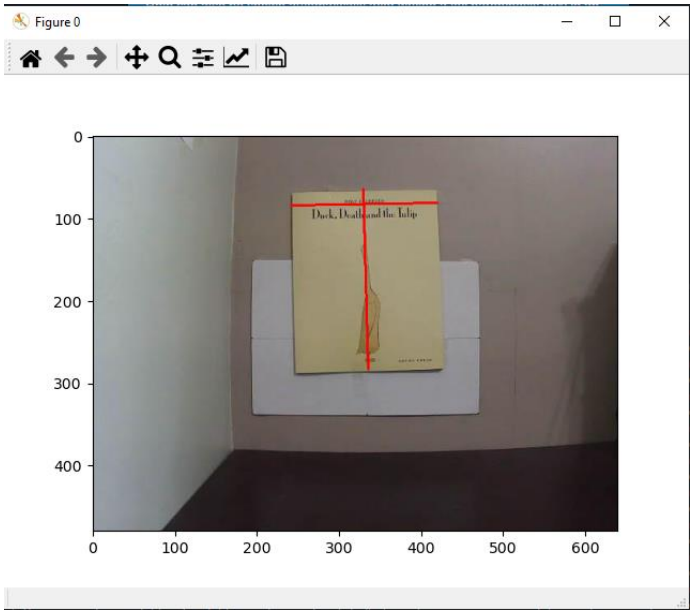


Ilustración 13 Toma de medidas por píxel del objeto

```
La distancia a la pared es:  76.52869189089132
El ancho del objeto es:  21.957670029838585
El alto del objeto es:  26.250920157489052
```

Ilustración 14 Resultados obtenidos

Prueba 4: estuche a diferente distancia

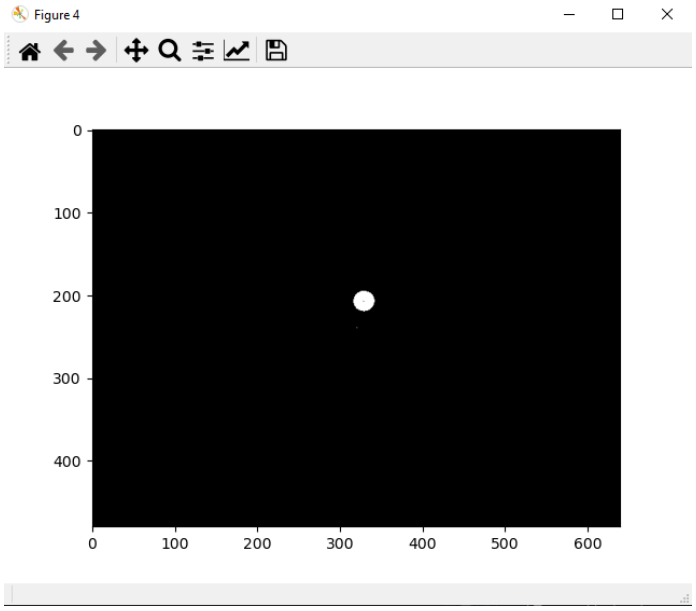


Ilustración 15 Pixeles entre el láser y el centro de la imagen

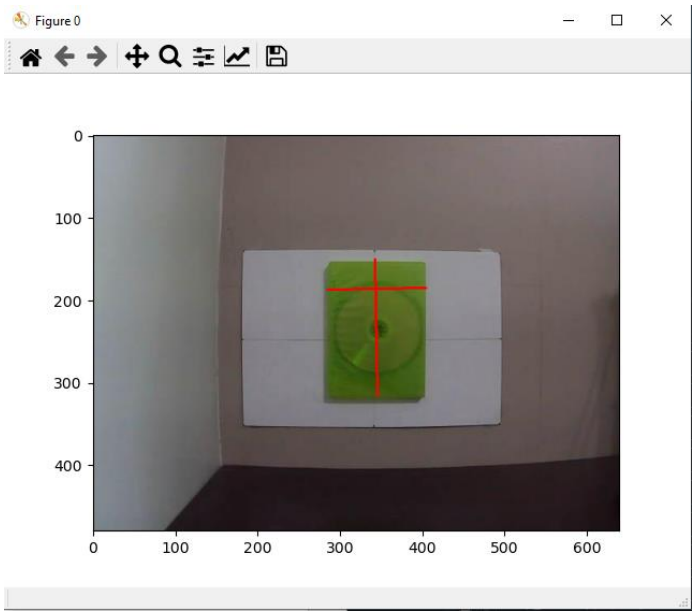


Ilustración 16 Toma de medidas por píxel del objeto

```
La distancia a la pared es:  68.71970238258369
El ancho del objeto es:  13.745711942701401
El alto del objeto es:  18.556945709570307
```

Ilustración 17 Resultados obtenidos

Las medidas reales de los objetos:

	Estuche	Nintendo DS	Libro	Estuche
Ancho	13.5 cm	7.3 cm	24.2 cm	13.5 cm
Altura	19 cm	13.3 cm	29.7 cm	19 cm
Distancia	50 cm	40 cm	80 cm	70 cm

Los resultados obtenidos de las 4 pruebas se muestran en la siguiente tabla:

	Estuche	Nintendo DS	Libro	Estuche
Ancho	11.80 cm	5.54 cm	21.95 cm	13.74 cm
Alto	18.07 cm	14.01 cm	26.25 cm	18.55 cm
Distancia	50.61 cm	38.87 cm	76.52 cm	68.71 cm

Los resultados obtenidos tuvieron una ligera variación a los valores reales, se estima que se encuentre un error entre el 9 y 15% entre la medida obtenida mediante el programa y los valores reales de los objetos y su distancia a la cámara. De igual forma entre más alejada la cámara estuviera del objeto los resultados muestran un mejor acercamiento al valor real. Esto se debe principalmente a que, a distancias lejanas tanto el objeto como el punto proyectado por el laser tienden a localizarse en la sección más céntrica de la imagen, que es donde la lente de la cámara presenta un comportamiento más lineal.

Conclusiones

La calibración de la cámara requirió su tiempo debido a que el tomar distintas fotos y buscar valores de pixeles para obtener un valor lo más promedio posible que se puede utilizar en varios momentos es un trabajo tedioso. Sin embargo, los valores obtenidos mediante la calibración ya sea en profundidad o largo y ancho fueron muy confiables en el desarrollo de práctica. Ya que los resultados finales presentaban poca diferencia a los valores reales. Es importante mantener las condiciones de distancia posición de la cámara y el láser, ya que los mejores resultados se ven presenten en situaciones homogéneas, aunque la calibración ajusta los valores mantener las condiciones puede asegurar menor variación con la realidad. En las primeras pruebas de distancia los valores se mantuvieron y era demasiado exactos, sin embargo, en la segunda prueba el láser empezó a tener batería baja por lo que se sospecha que dicho fenómeno pudo afectar la potencia de la luz y a su vez realizar un cambio en los cálculos. Debo concluir que quizá con una mayor toma de datos el sistema puede mejorar bastante, asimismo, una segmentación y conteo de pixeles que sea automatizado podría mitigar errores humanos.

-Aguilar Gómez Pedro Nicolás

Como se mencionó de forma teórica, al realizar la calibración de la cámara se reafirmó que la relación entre la distancia en pixeles y la distancia real no es de carácter lineal, esto se observa claramente a través del plano del lente obtenido, ya que este no es “liso” si no más bien presenta curvaturas. Este efecto de la curvatura de la lente también se vio reflejado al calibrar la sección de medición de profundidad con el láser, por lo que para ambos casos es fundamental ajustar un polinomio para obtener resultados acertados. Cabe denotar que los polinomios obtenidos para una cámara no funcionan en otra, por lo que para cada cámara es necesario realizar su debida calibración. Aunque existen varios métodos para realizar la calibración, el método empleado es bastante sencillo y produce buenos resultados, sin embargo, en este caso el porcentaje máximo de error fue del 15% debido a que influyen múltiples factores desde la etapa de calibración hasta el momento de realizar una medición. En la etapa de la calibración, el error puede ser disminuido al tener un mejor control de las distancias para la toma de datos, así como incrementar el número de tomas, ya que de esta forma el polinomio describiría en mayor medida el efecto de la lente. En la etapa de medición debido a que se realiza de forma manual esta esta influenciada a posibles errores humanos por lo que implementar una etapa automática para la medición de distancia en pixeles de los bordes del objeto ayudaría a incrementar la precisión en las mediciones.

- Sanchez Sanchez Luis Alberto

Referencia

Departamento de Ingeniería electrónica, Telecomunicación y Automática. Área de Ingeniería de Sistemas y Automática. (s. f.). *Detección de bordes en una imagen*. UJAEN. Recuperado 1 de mayo de 2021, de http://www4.ujaen.es/~satorres/practicas/practica3_vc.pdf