Proyecto#4: Sistema de realidad aumentada

Autores: Grupo#1

- Juan David Aleman Falco – T00062639
- Donovan Gómez Medrano – T00061057

Asignatura:

Visión Artificial

Docente:

Yady Tatiana Solano Correa

NRC:

2451

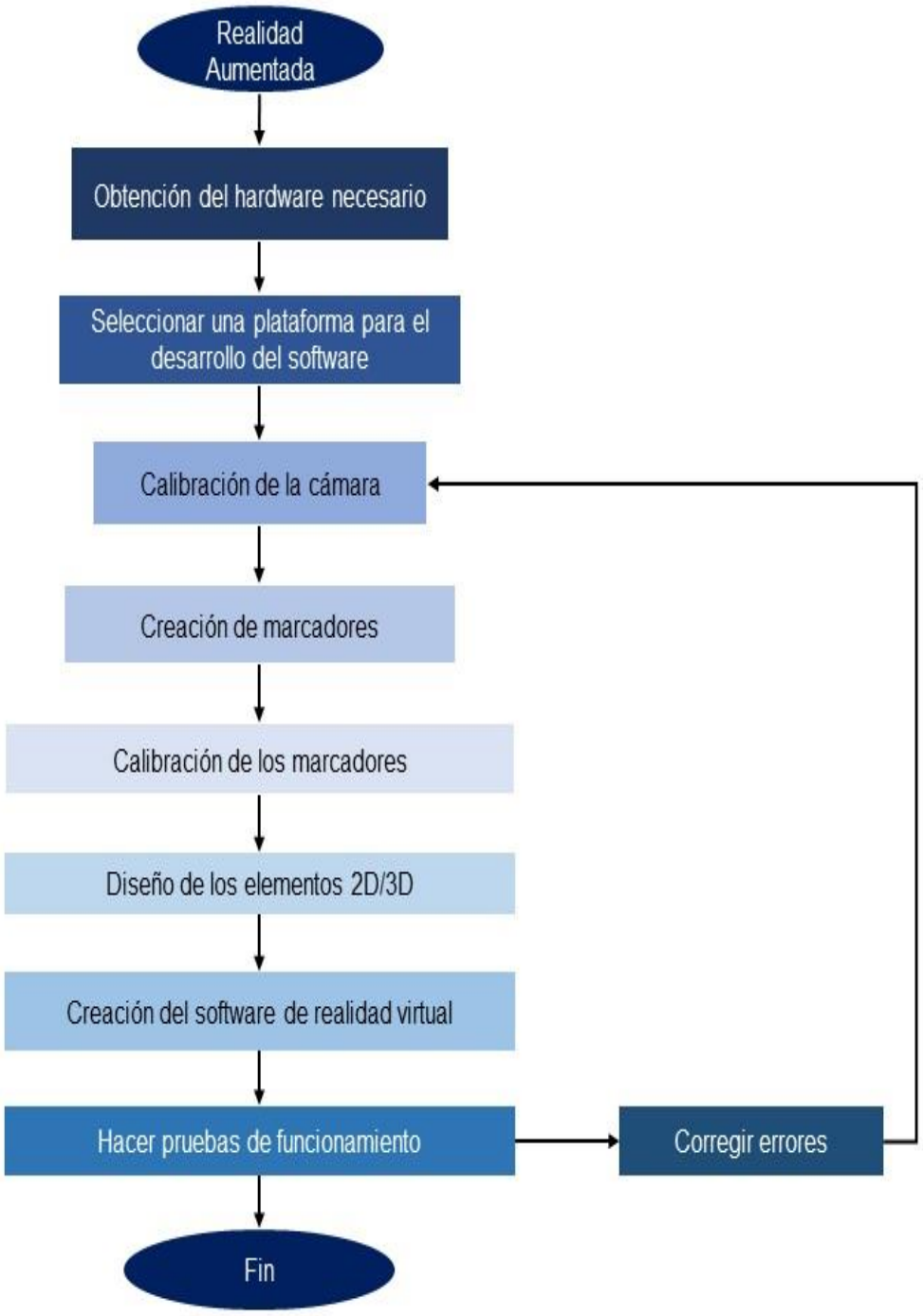
Tema:

Realidad Aumentada

Universidad Tecnológica de Bolívar

13/04/2023

La realidad aumentada es una tecnología que ha ganado mucha popularidad en los últimos años y es utilizada en diversas áreas como juegos, publicidad y educación. Esta tecnología permite proyectar elementos 2D o 3D sobre un objeto de referencia, creando experiencias inmersivas e interactivas para el usuario. Sin embargo, su implementación puede presentar desafíos técnicos importantes, como la identificación precisa del objeto de referencia, la sincronización de la proyección y la calidad de la imagen proyectada. Además, es importante considerar la compatibilidad con diferentes dispositivos y plataformas, así como la privacidad y seguridad de los usuarios.



Materiales:

- Cámara
- Cheesboard (para Calibración de la cámara)
- Marcadores Arauco
- Dispositivos de procesamiento (Portátil u Ordenador)
- Software de programación (Jupyter)

Recursos necesarios:

1. Entorno de desarrollo integrado (IDE): Necesitarás un IDE para escribir y depurar tu código, como VScode, PyCharm o Sublime Text.
2. Bibliotecas de Python: Es posible que también se necesiten instalar varias bibliotecas de Python, como OpenCV o NumPy. Estas bibliotecas permitirán realizar tareas como la detección y el procesamiento de imágenes y gráficos en 3D.
3. Aruco: Descargar e instalar la biblioteca Aruco, que es una biblioteca de marcadores de realidad aumentada de código abierto. Los marcadores Aruco son imágenes cuadradas con un patrón de puntos blancos y negros, que se utilizan para detectar la posición y orientación de la cámara en relación con la escena.
4. Comprensión de los conceptos fundamentales de la realidad aumentada, como la calibración de la cámara, la detección de marcadores y la superposición de objetos virtuales.
5. Recursos adicionales, como tutoriales en línea, videos instructivos, documentación y ejemplos de código.
    - "Augmented Reality with Python and OpenCV" por Joseph Howse (2018) - Este libro cubre los conceptos básicos de la realidad aumentada, cómo usar Python y OpenCV para construir aplicaciones de realidad aumentada y cómo integrar la tecnología en una variedad de proyectos.
    - "Real-time Augmented Reality for Python Developers" por Ankit Dangi (2021) - Este libro se centra en cómo crear aplicaciones de realidad aumentada en tiempo real con Python y las bibliotecas de visión artificial.

Procedimiento:

1. **Obtención del hardware:** Obtendremos los equipos necesarios para el proyecto como los son cámaras y dispositivos de procesamiento los cuales permitan correr el programa (Ej: Un teléfono celular)
2. **Seleccionar una plataforma para el desarrollo del software:** En esta parte escogeremos un programa o software el cual nos permita crear el entorno de realidad virtual en Python (Jupyter).
3. **Calibración de la cámara:** En este paso calibraremos la cámara para que pueda ser utilizada por el programa.

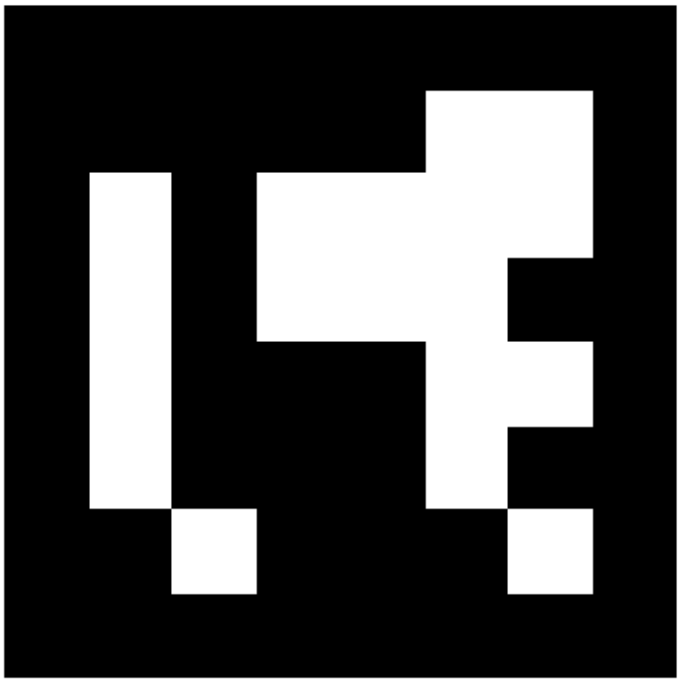
```
def calibration(images): #Definition of the

function for calibration

  pattern_shape = (9, 6)
  print('=== Camera Calibration ===')

  DISPLAY_CORNERS = True
  # Define the world coordinates of the checkerboard corners
  objp = np.zeros((pattern_shape[0] * pattern_shape[1], 3),
np.float32)
  objp[:, :2] = np.mgrid[0:pattern_shape[0],
0:pattern_shape[1]].T.reshape(-1, 2)
  # Create a list to store the object points and the image
points
  objp_list = [] # 3D points in real world space
  imgp_list = [] # 2D points in image plane
  img_shape = None

  # Detect the corners in each image
  for fname in range(len(images)):
    print(fname)
    img = images[fname]
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    if img_shape is None:
        img_shape = gray.shape
    elif img_shape != gray.shape:
        print('Mismatch size')
        continue
    # Find the checkerboard corners
    ret, corners = cv2.findChessboardCorners(gray,
pattern_shape, None) # Find the chess board corners

    # If the corners are found
    if ret:
        objp_list.append(objp)
        imgp_list.append(corners)
```

```
        if DISPLAY_CORNERS:
            cv2.drawChessboardCorners(img, pattern_shape,
corners, ret) #function is then used to draw the corners on the
original image
            plt.imshow(img)
            plt.show()

            cv2.waitKey(0) #the program waits for 0 milliseconds
    else:
        print('Could not find corners in image', fname)

  CALIBFLAG = 0  # cv2.CALIB_FIX_K3
  print(' ', len(objp_list), 'images are used')
  rms, cam_int, cam_dist, rvecs, tvecs =
cv2.calibrateCamera(objp_list, imgp_list, img_shape, None, None,
None, None, CALIBFLAG)
  np.set_printoptions(suppress=True)
  return cam_int,cam_dist
```

4. **Creación de marcadores:** Definiremos los marcadores a utilizar, ya sean tipo QR, Aruco, imágenes, etc.

```
import numpy as np #we import the libraries
import cv2, PIL
import cv2
from cv2 import aruco
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
import glob
from skimage import io
import urllib.request

aruco_dict =aruco.Dictionary_get(aruco.DICT_6X6_250)
#Dictionary/Set of markers, it contains the inner
codification.

fig = plt.figure()
nx = 1
ny = 1
for i in range(1, nx*ny+1):
    ax = fig.add_subplot(ny,nx, i)
    img = aruco.drawMarker(aruco_dict,i, 700) #we draw marker
    plt.imshow(img, cmap = mpl.cm.gray, interpolation =
"nearest") #print marker
    ax.axis("off")
#we save the marker in the folder of our device
plt.savefig("C:/Users/USER/Downloads/Vision/aruco_markers.jpeg
")
plt.show()
```

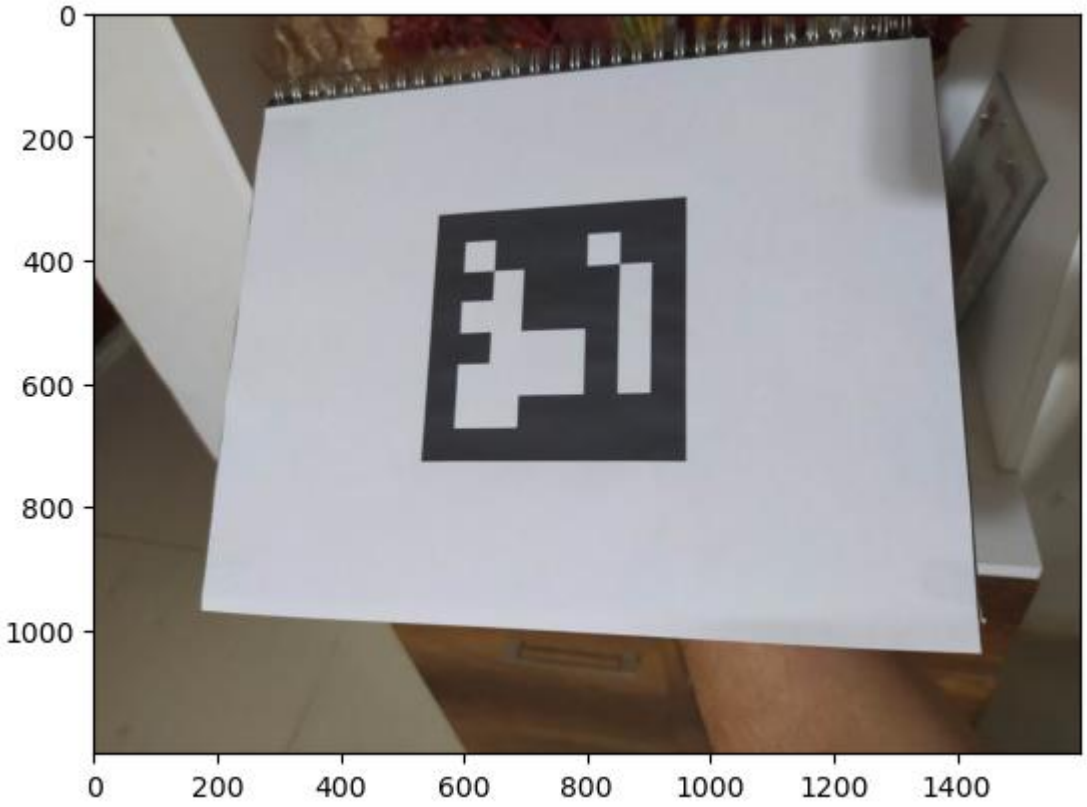
Al ejecutar el código obtenemos el siguiente marcador:

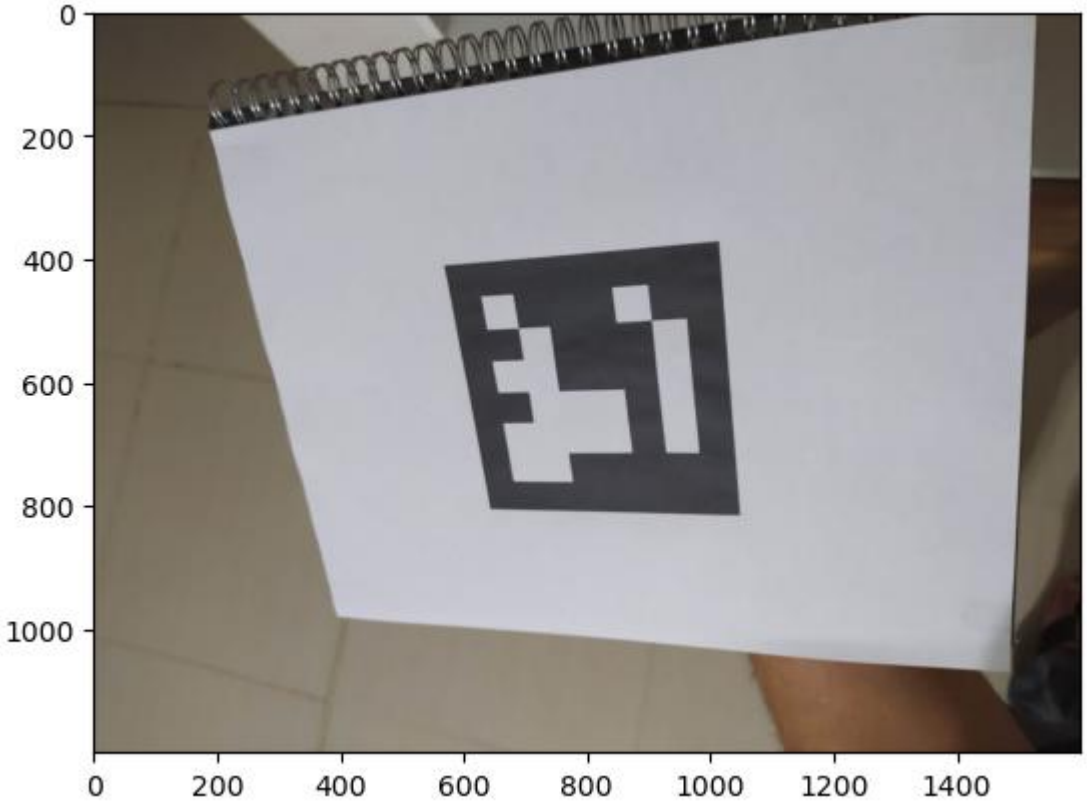


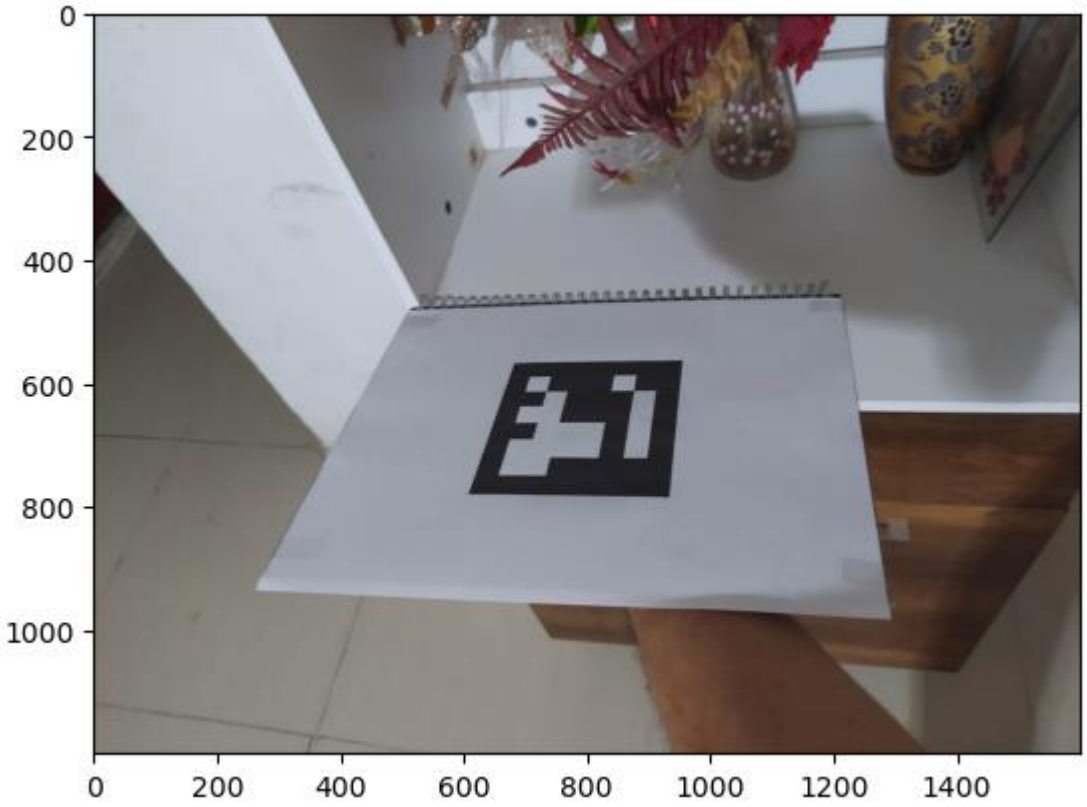
5. **Calibración de los marcadores:** En esta sección nos encargaremos de que el programa sea capaz de reconocer nuestros marcadores e identificarlos correctamente.

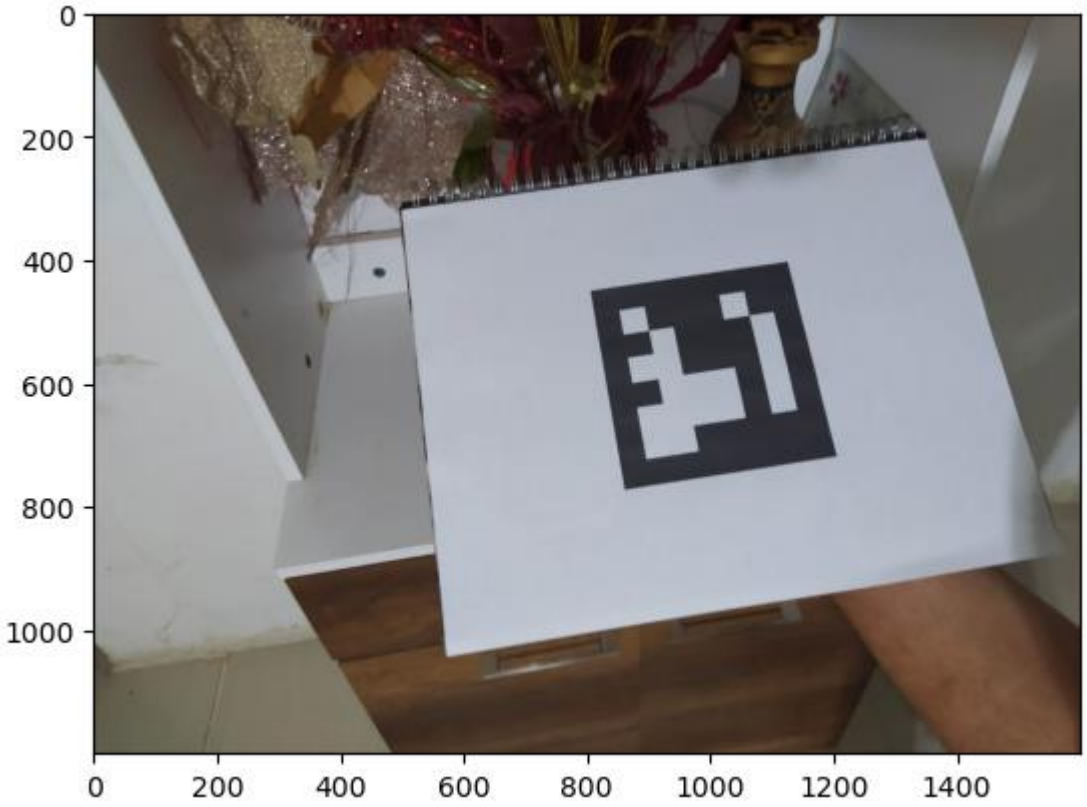
```
#We read the photos of the markers made in real life
img1 = cv2.imread('C:/Users/USER/Downloads/Vision/1.jpeg')
img2 = cv2.imread('C:/Users/USER/Downloads/Vision/2.jpeg')
img3 = cv2.imread('C:/Users/USER/Downloads/Vision/3.jpeg')
img4 = cv2.imread('C:/Users/USER/Downloads/Vision/4.jpeg')
frame = [img1,img2,img3,img4]

#We print these images
for i in range(len(frame)):
    print(i)
    img = frame[i]
    color = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(color)
    plt.show()
```

Lectura de las imágenes:


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

```
#we load the images to calibrate the camera
a= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/a.jpeg')
b= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/b.jpeg')
c= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/c.jpeg')
d= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/d.jpeg')
e= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/e.jpeg')
f= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/f.jpeg')
g= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/g.jpeg')
h= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/h.jpeg')
i= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/i.jpeg')
j= cv2.imread('C:/Users/USER/Desktop/Juan David/nuevo
semestre/Vision/Actividad 10/Nueva carpeta (5)/j.jpeg')

images = [a,b,c,d,e,f,g,h,i,j]

#we obtain the values of the matrix and distortions
matrix, dist=calibration(images)
print(matrix)
print(dist)
```

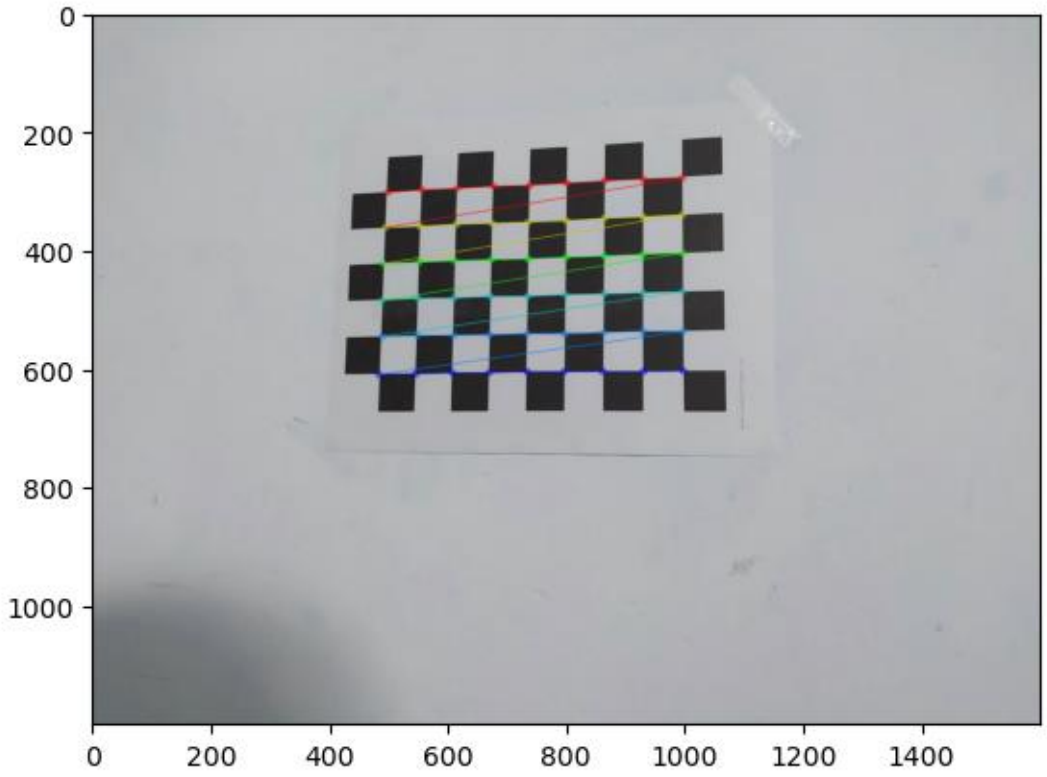
Calibración de la cámara:

=== Camera Calibration ===

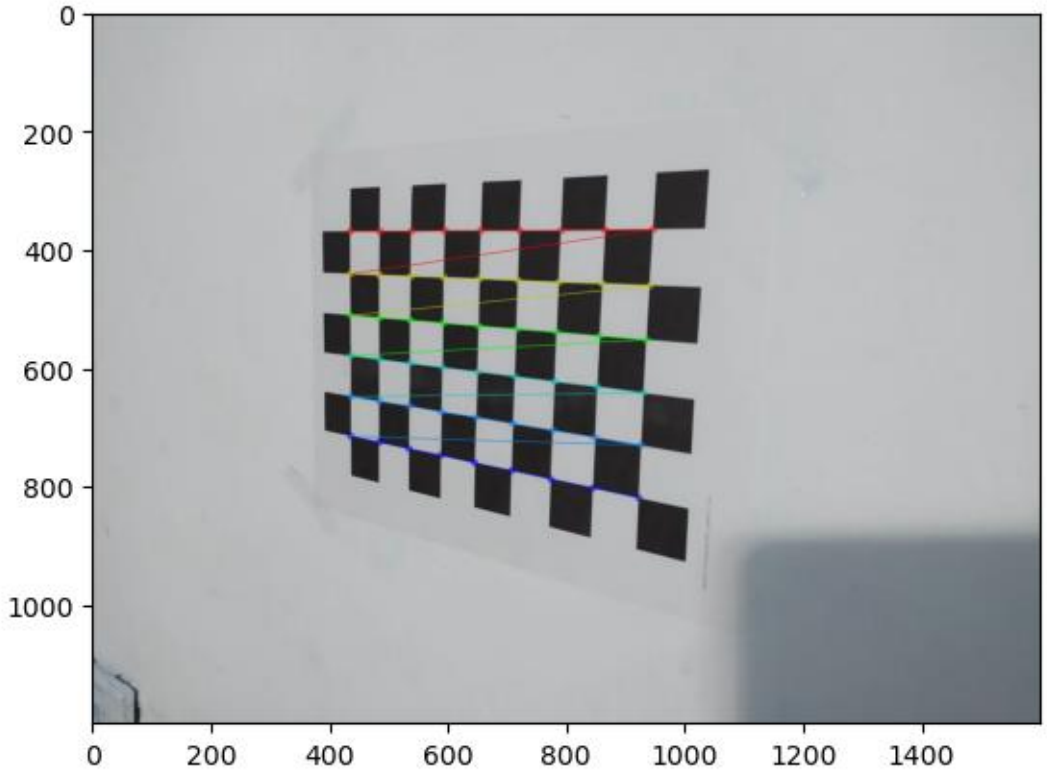
0

Could not find corners in image 0

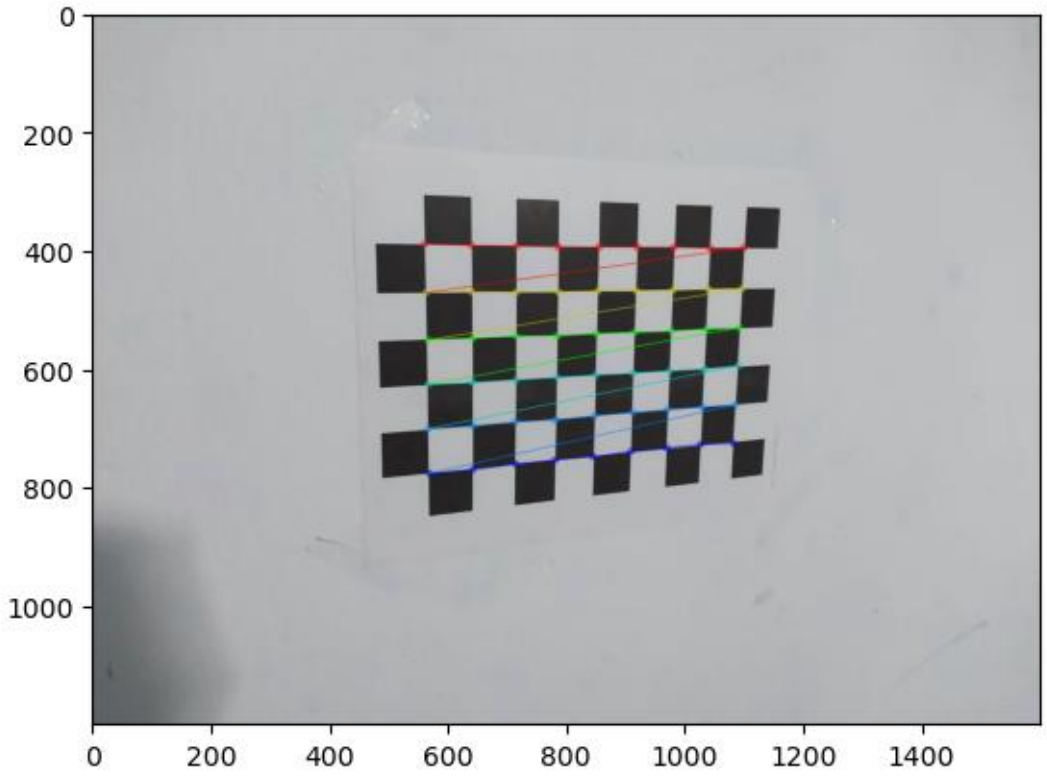
1


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

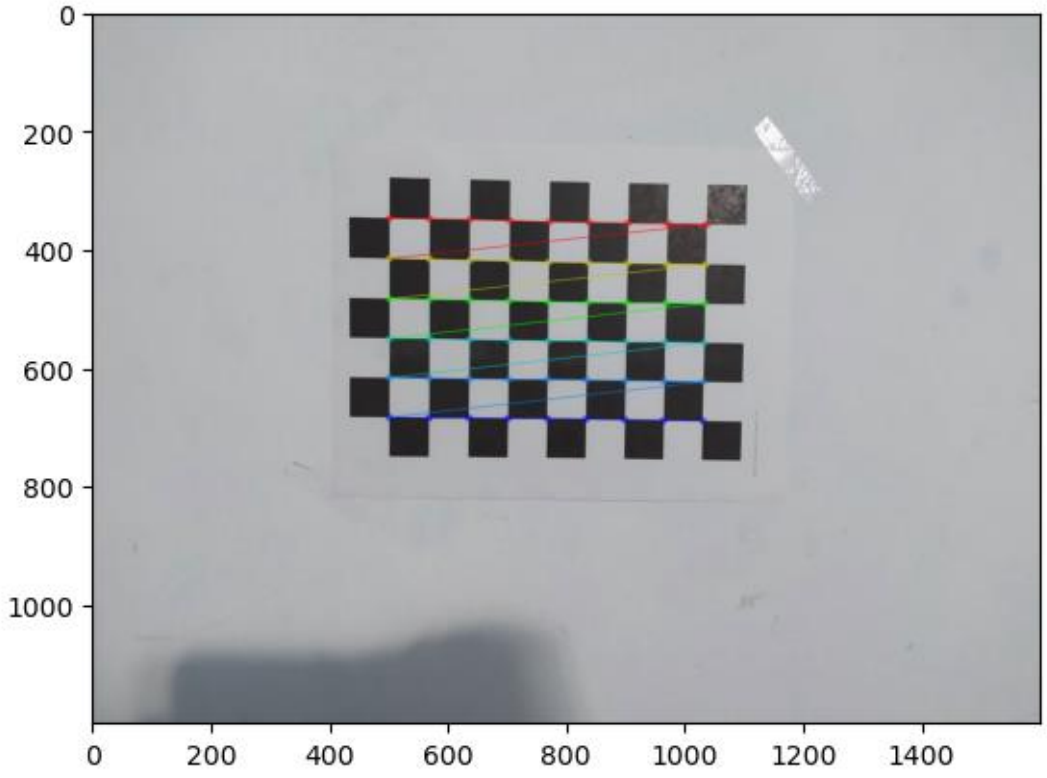
2


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

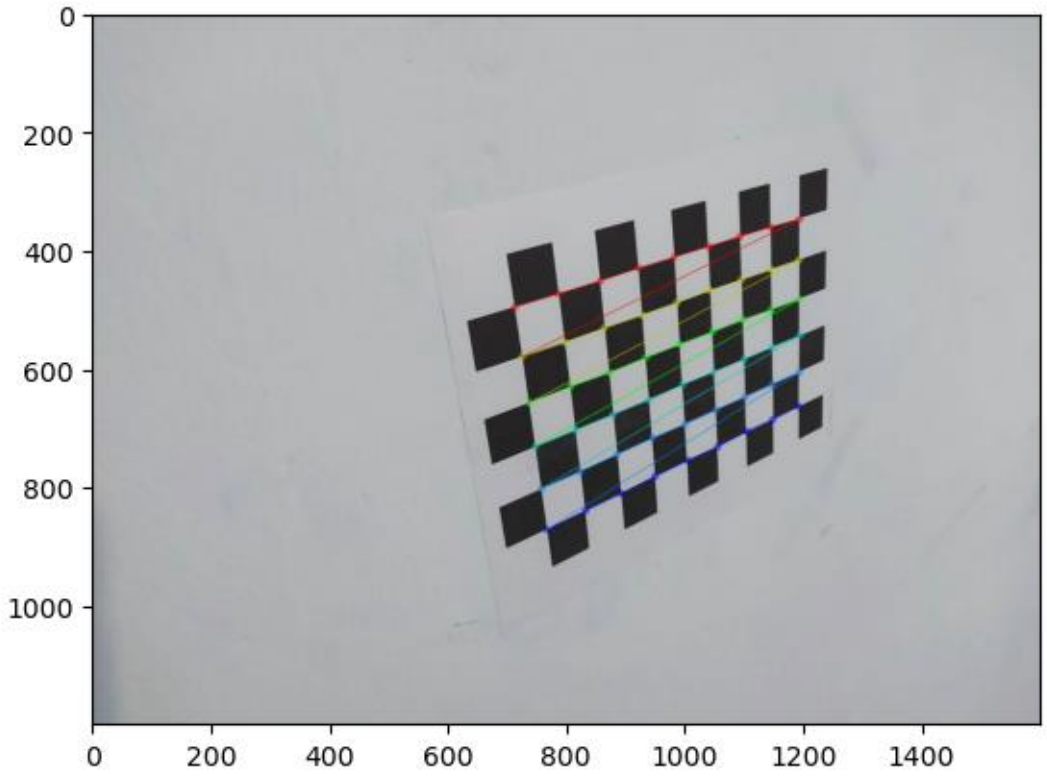
3


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

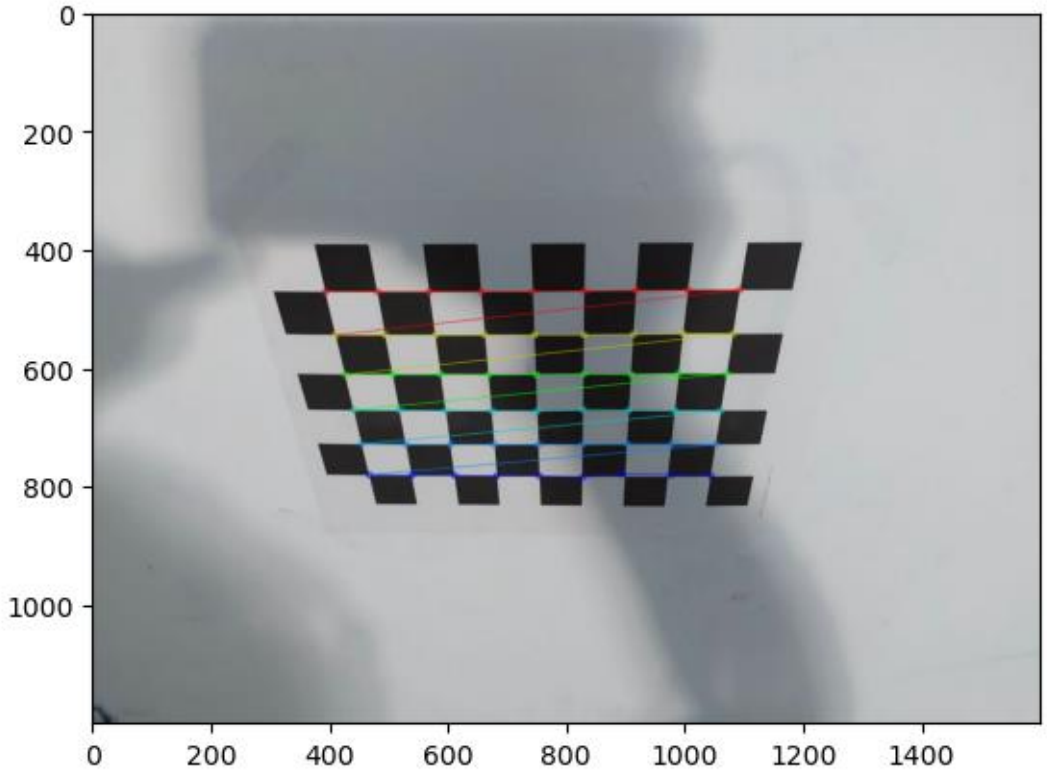
4


0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400

5

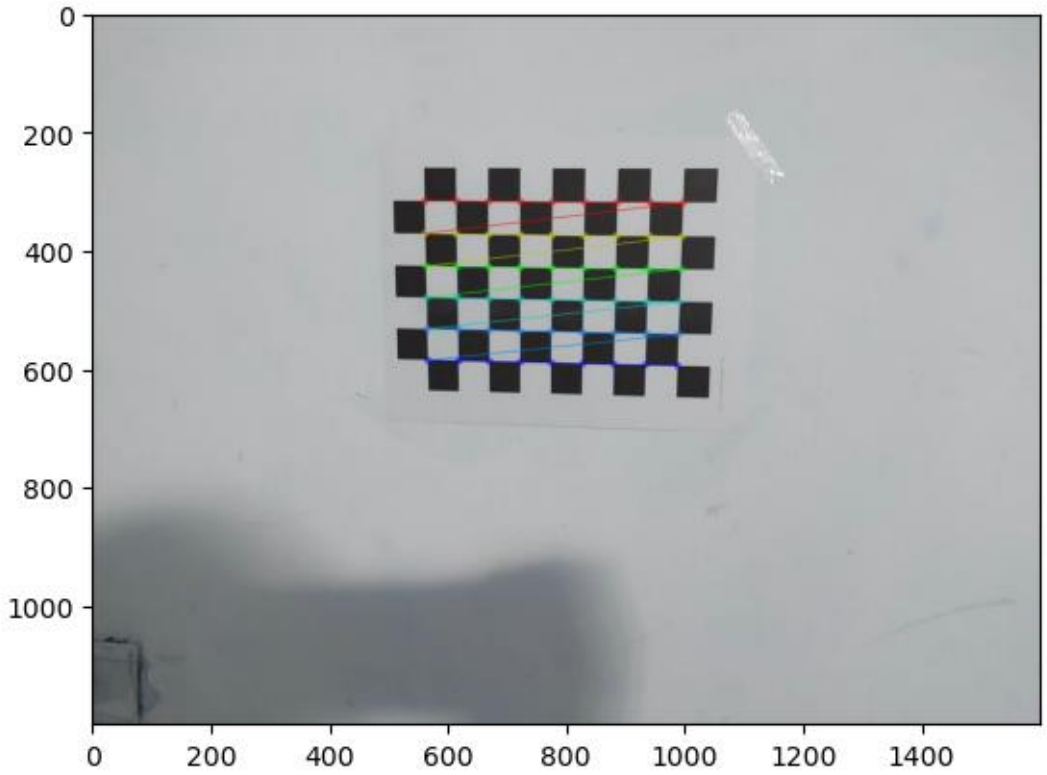


6




```
7
Could not find corners in image 7
8
Could not find corners in image 8
```

9



7 images are used

[[1221.55974445 0. 783.86430096]

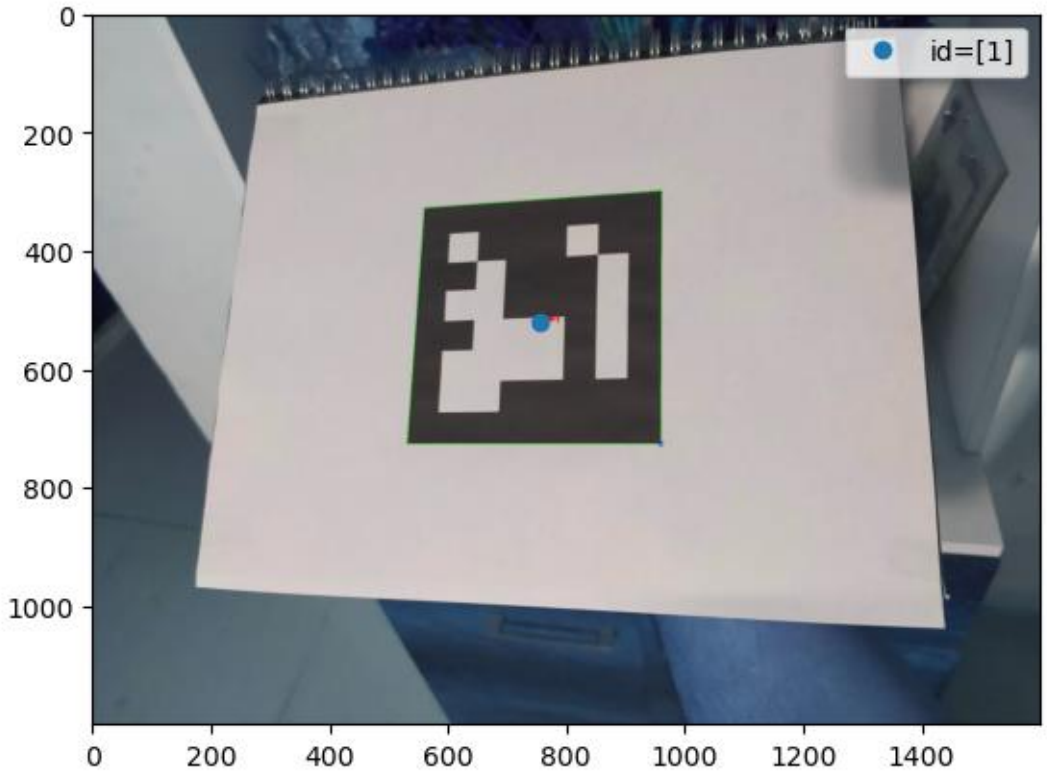
[0. 1222.10977754 595.59726311]

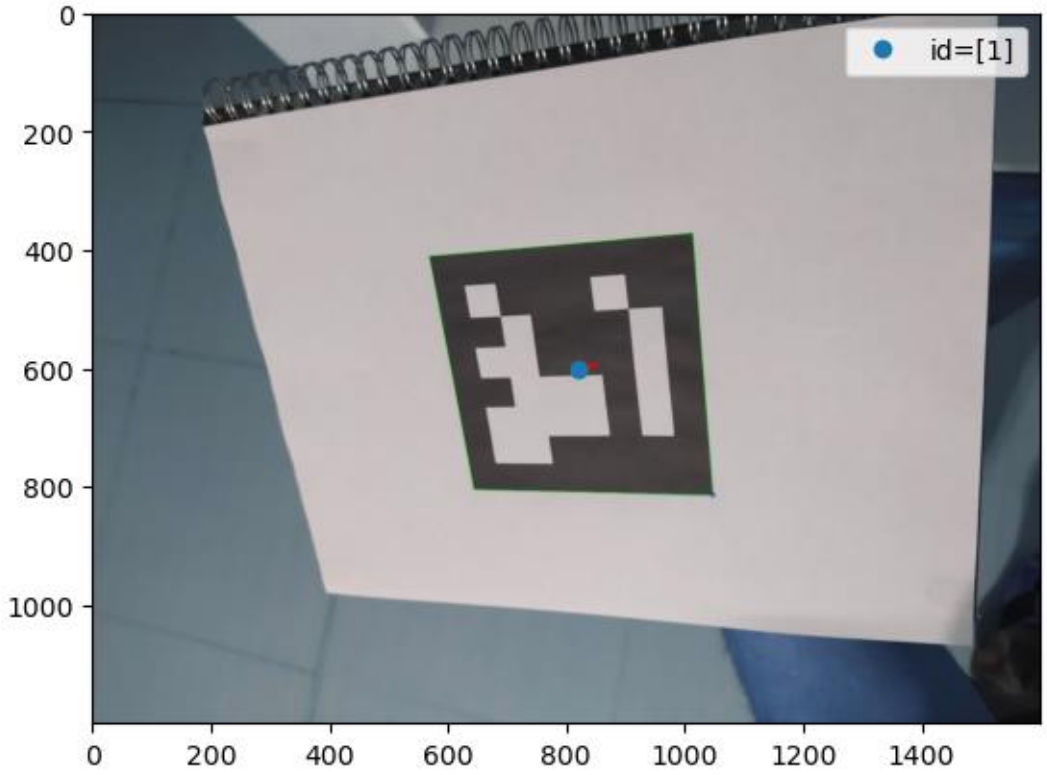
[0. 0. 1.]]

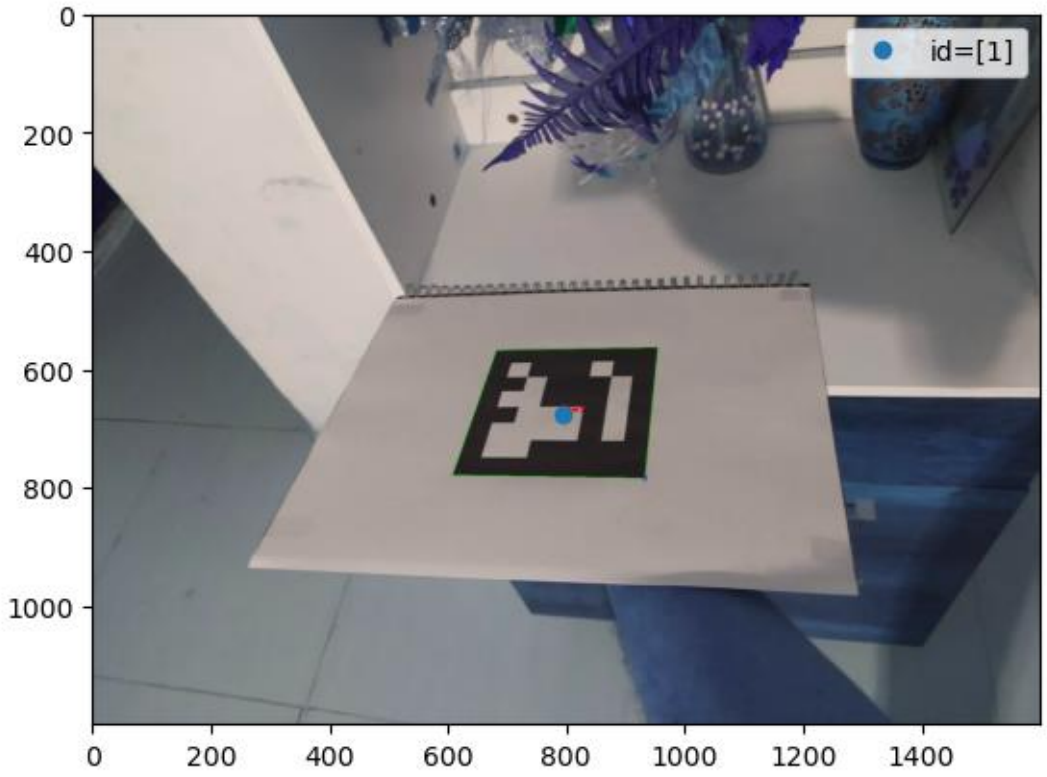
[[0.23143999 -2.49382049 -0.00464086 -0.00567373 8.01389779]]

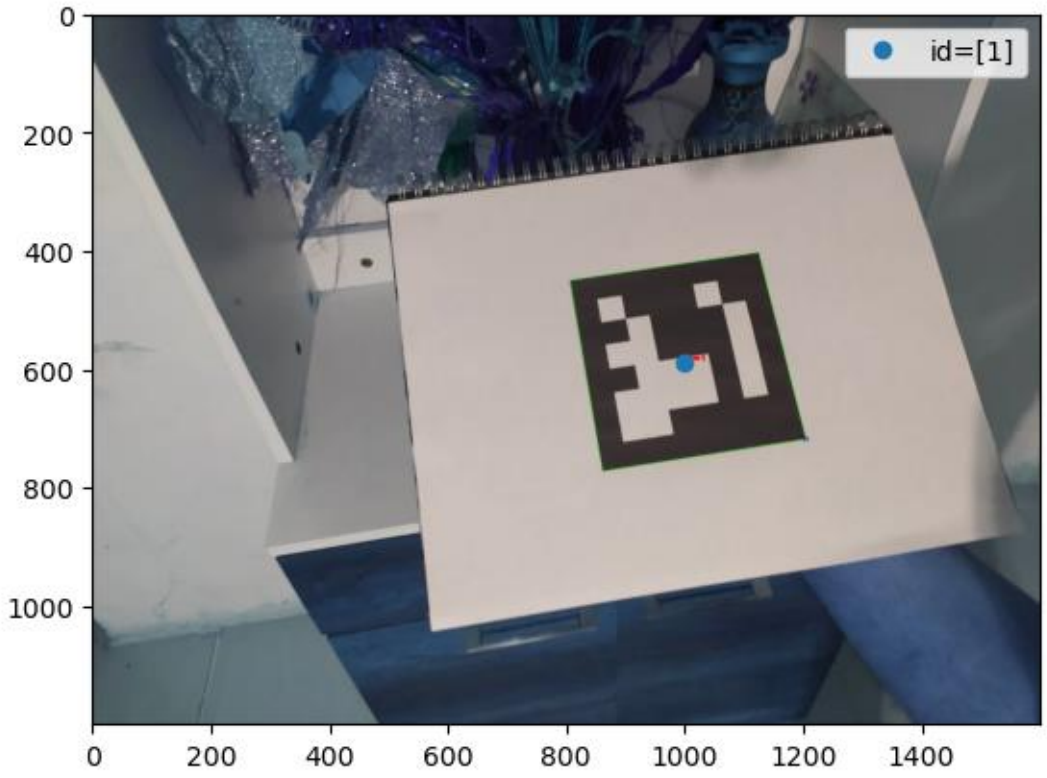
```
%%time
for i in range(len(frame)):
    gray = cv2.cvtColor(frame[i], cv2.COLOR_BGR2GRAY) #convert
images to grayscale
    aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
#Arauco dictionary
    parameters =  aruco.DetectorParameters_create() #arauco
parameter detection
    corners, ids, rejectedImgPoints = aruco.detectMarkers(gray,
aruco_dict, parameters=parameters) #Marker detection
    frame_markers = aruco.drawDetectedMarkers(frame[i].copy(),
corners, ids)#we draw on the marker
    plt.figure()
    plt.imshow(frame_markers)
    for i in range(len(ids)):
        c = corners[i][0] #we use the corners of the marker
        plt.plot([c[:, 0].mean()], [c[:, 1].mean()], "o", label
= "id={0}".format(ids[i])) #we graphed on this
    plt.legend()
    plt.show()
```

Calibración del marcador:









CPU times: total: 1.78 s

Wall time: 1.71 s

6. **Diseño de los elementos 2D/3D:** Para esta fase crearemos los modelos 2D o 3D que vamos a superponer en realidad aumentada.

    - Para este caso diseñaremos un cubo:

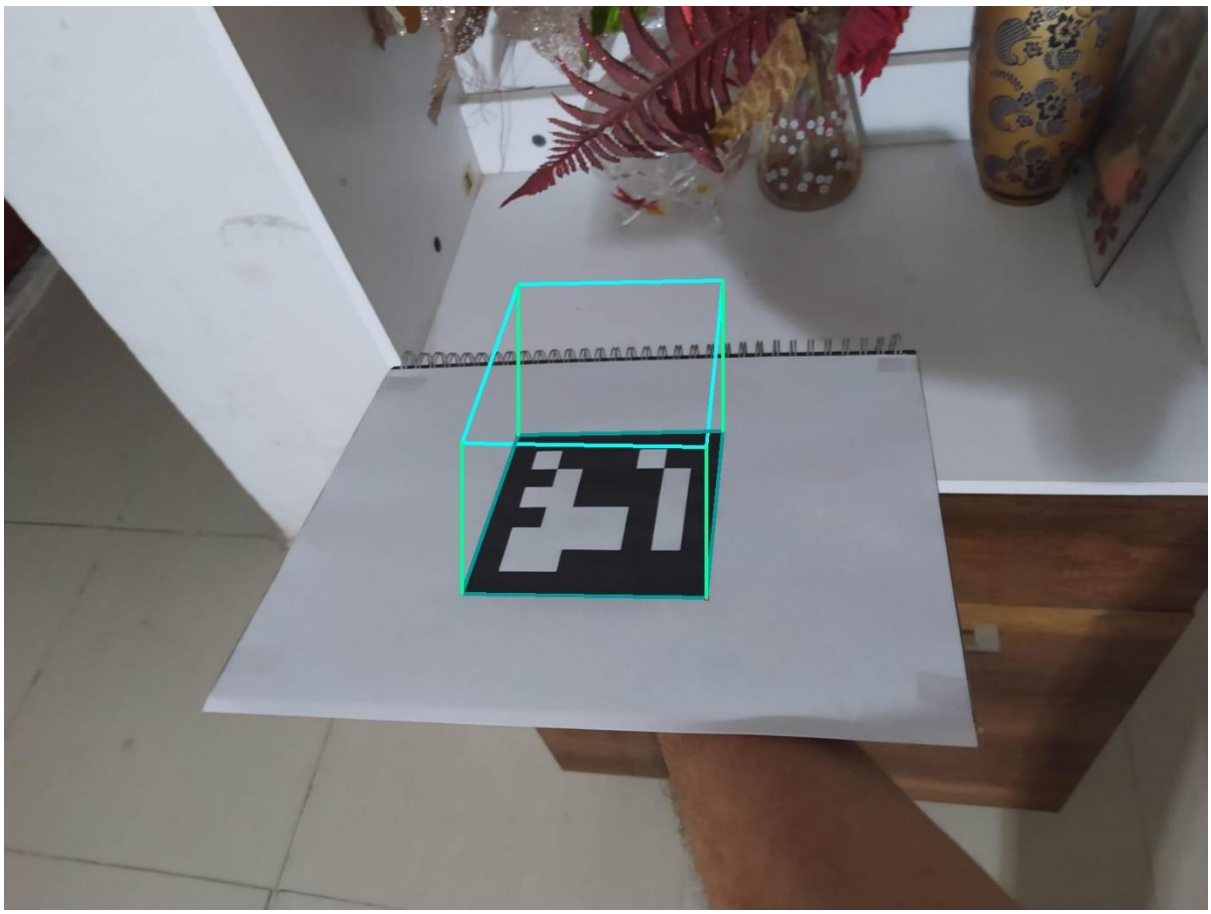
```
frame=img3
marker_id = 0
marker_size = 700

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #convert
images to grayscale
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250) #Arauco
dictionary
parameters =  aruco.DetectorParameters_create() #arauco
parameter detection
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray,
aruco_dict, parameters=parameters)#Marker detection
frame_markers=cv2.aruco.drawDetectedMarkers(frame, corners)
#we draw on the marker

#definition corners
c1=(corners[0][0][0][0],corners[0][0][0][1])
c2=(corners[0][0][1][0],corners[0][0][1][1])
c3=(corners[0][0][2][0],corners[0][0][2][1])
c4=(corners[0][0][3][0],corners[0][0][3][1])

#definition vertices
v1,v2=c1[0],c1[1]
v3,v4=c2[0],c2[1]
v5,v6=c3[0],c3[1]
v7,v8=c4[0],c4[1]

#Cube top face
cv2.line(frame,(int(v1),int(v2-200)),(int(v3),int(v4-
200)),(255,255,0),3)
cv2.line(frame,(int(v5),int(v6-200)),(int(v7),int(v8-
200)),(255,255,0),3)
cv2.line(frame,(int(v1),int(v2-200)),(int(v7),int(v8-
200)),(255,255,0),3)
cv2.line(frame,(int(v3),int(v4-200)),(int(v5),int(v6-
200)),(255,255,0),3)

#Bottom face of the cube
cv2.line(frame,(int(v1),int(v2)),(int(v3),int(v4)),(155,155,0
),3)
cv2.line(frame,(int(v5),int(v6)),(int(v7),int(v8)),(155,155,0
),3)
cv2.line(frame,(int(v1),int(v2)),(int(v7),int(v8)),(155,155,0
),3)
cv2.line(frame,(int(v3),int(v4)),(int(v5),int(v6)),(155,155,0
),3)

#Side faces of the cube
cv2.line(frame,(int(v1),int(v2-
200)),(int(v1),int(v2)),(155,255,0),3)
```

```
cv2.line(frame,(int(v3),int(v4-
200)),(int(v3),int(v4)),(155,255,0),3)
cv2.line(frame,(int(v5),int(v6-
200)),(int(v5),int(v6)),(155,255,0),3)
cv2.line(frame,(int(v7),int(v8-
200)),(int(v7),int(v8)),(155,255,0),3)
```

7. **Creación del software de realidad virtual:** Para esta fase lo que haremos es combinar la parte real con la virtual a través del uso del software haciendo uso de los diseños creados anteriormente superponiéndolos sobre los marcadores designados.

```
#we draw the 3d cube
cv2.imshow("Dibujo cubo 3D",frame)
cv2.waitKey(0)
cv2.DestroyAllWindows()
```

8. **Pruebas de funcionamiento:** En esta parte se hará todo lo relacionado con el control de calidad, por lo cual se harán diversas pruebas y estudios para confirmar el correcto funcionamiento del programa.

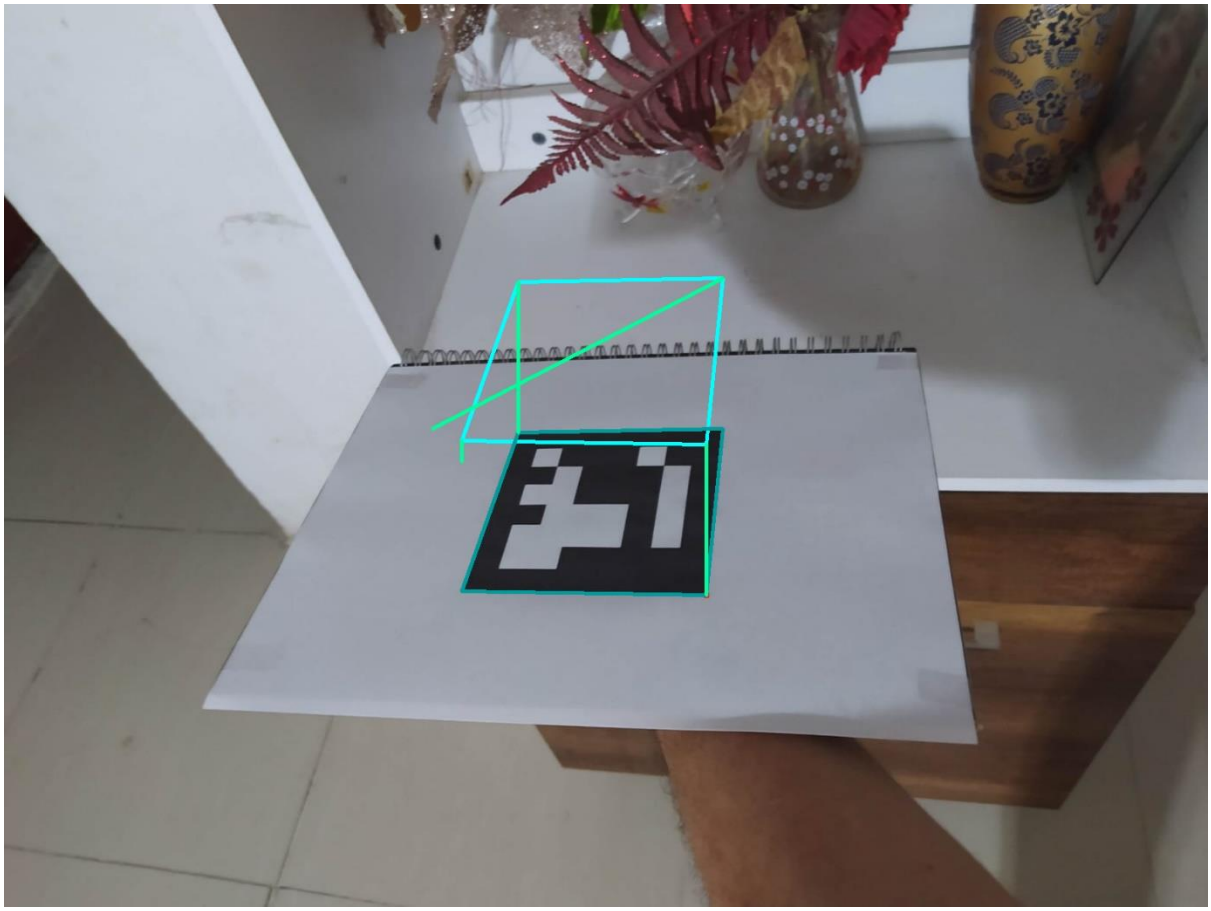


9. **Corrección de errores:** En caso tal de que en las pruebas de funcionamiento tengamos errores por parte del programa, lo que haremos es revisar el código desde

sus fases iniciales en la búsqueda del error el cual no permite el correcto funcionamiento del programa.

Observaciones y dificultades:

- Hardware:
    1. Cámara: Las cámaras de los dispositivos de video y captura de imágenes actuales son muy sofisticados, por lo cual la resolución de estos es muy grande lo cual puede provocar problemas al momento del procesamiento de las imágenes. Razón por la cual es necesario realizar comprensiones en la imagen con el fin de facilitar el procesamiento de las imágenes.
    2. Dispositivo de procesamiento: Para poder usar el código de la manera mas optima posible y relacionado con lo anterior, es necesario que este tenga la capacidad de procesamiento suficiente para ejecutar todas las funciones la manera más optima posible.
    3. Chessboard: Es necesario que las imágenes sean tomadas adecuadamente de lo contrario puede haber errores al momento de la calibración del software.
- Software:
    1. Calibración de la cámara: Si no se tomaron unas fotos adecuadas esto puede dificultar el proceso de calibración de la cámara.
    2. Detección de los marcadores Arauco: Si no se sabe usar adecuadamente la librería Arauco puede haber muchas dificultades al momento de tratar de detectar el marcador
    3. Dibujar sobre el Arauco: Para esto hay que saber cómo funciona las funciones de dibujo de Arauco, así como definir bien sus valores de lo contrario pueden darse errores al momento de dibujar. Ejemplo:



En este caso al momento de colocar los valores dentro de la función ***cv2.line()*** se definieron de manera incorrecta los valores de los vértices, dando como resultado que el dibujo del cubo no fuera lo que se esperaba.

Bibliografía:

[1] "Augmented Reality with Python and OpenCV" por Joseph Howse (2018)

[2] "Real-time Augmented Reality for Python Developers" por Ankit Dangi (2021)

[3] ARUCO markers: basics — Scientific Python: a collection of science oriented python examples documentation. (s/f). Readthedocs.Io. Recuperado el 13 de abril de 2023, de https://mecaruco2.readthedocs.io/en/latest/notebooks_rst/Aruco/aruco_basics.html

[4] Ingenia, A. e. [@AprendeIngenia]. (2021, septiembre 12). REALIDAD AUMENTADA EN 3D en TIEMPO REAL Aruco Markers | Calibración de Camara Python OpenCV. Youtube. https://www.youtube.com/watch?v=pwtiJ5CsvaI