PROGRAMA ACADÉMICO

Departamento de ciencias aplicadas

UNIDAD DE ESTUDIO

Robótica-Diseño-Robots

INTEGRANTE

Kevyn Estiven Marín Nivia

David Leonardo Carrillo Sarmiento

Juan Pablo Veloza Chaves

DOCENTE

PhD. Edwin Nikolay Prieto Parrado

Contenido

1. Mecánica base soporte, ¿Se compra o se hace?

Para la compra de la base con características móviles y de fácil manipulación, se le recomendó al cliente la adquisición de una base para televisión capaz de sostener un objeto suspendido a una altura de 1.397 metros y soporta un peso de hasta 39.9 kilogramos. *(ver Figura 1)*

Figura 1: Soporte Móvil.

Fuente: https://www.amazon.com/

Sin embargo, el cliente opta por verificar la creación de un acople en la base/hombro destinado a una superficie plana, asegurando que luego él verificaba a que base lo acoplaría.

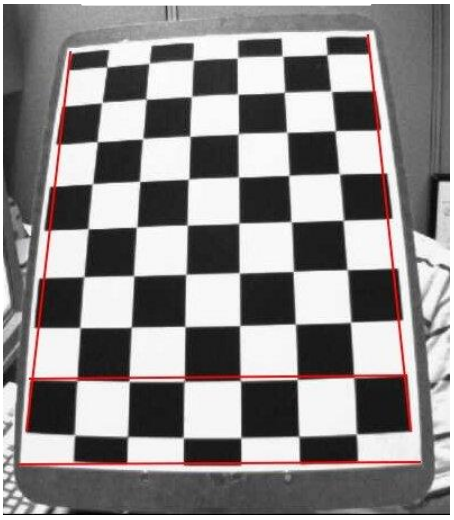
2. Robótica - Obtención de coordenadas X Y y Z de un objeto con cámara estero

2.1 Calibración Cámara

La calibración de la cámara es esencial para evitar problemas como lo son la distorsión debido al lente y los defectos de fabricación. Como lo plantea OpenCv las dos distorsiones principales son la distorsión radial y la distorsión tangencial.

- **Distorsión radial:** las líneas rectas aparecerán curvas. Su efecto es mayor a medida que nos alejamos del centro de la imagen. *(ver figura 2)*

Figura 2: Tablero.



Fuente: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

Como se puede ver en la imagen los trazos rectos de la línea roja no coinciden con la línea del cuadro esto debido a la distorsión provocada por el lente. Sin embargo, para solucionar esta problemática se puede efectuar el siguiente calculo. *(ver figura 3)*

Fuente: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

- **Distorsión Tangencial:** Se produce debido a que la cámara no se encuentra alineada perfectamente paralela al plano de formación de la imagen Esto provoca que algunas partes de la imagen se puedan ver mas grandes de lo real. Sin embargo, para solucionar esta problemática se puede efectuar el siguiente calculo. *(ver figura 4)*

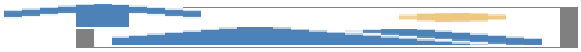
Figura 4: Ecuaciones.

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Fuente: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

Según lo anterior podemos concluir que es necesario encontrar los coeficientes de distorsión dados por. *(ver figura 5)*

Figura 5: Variables.



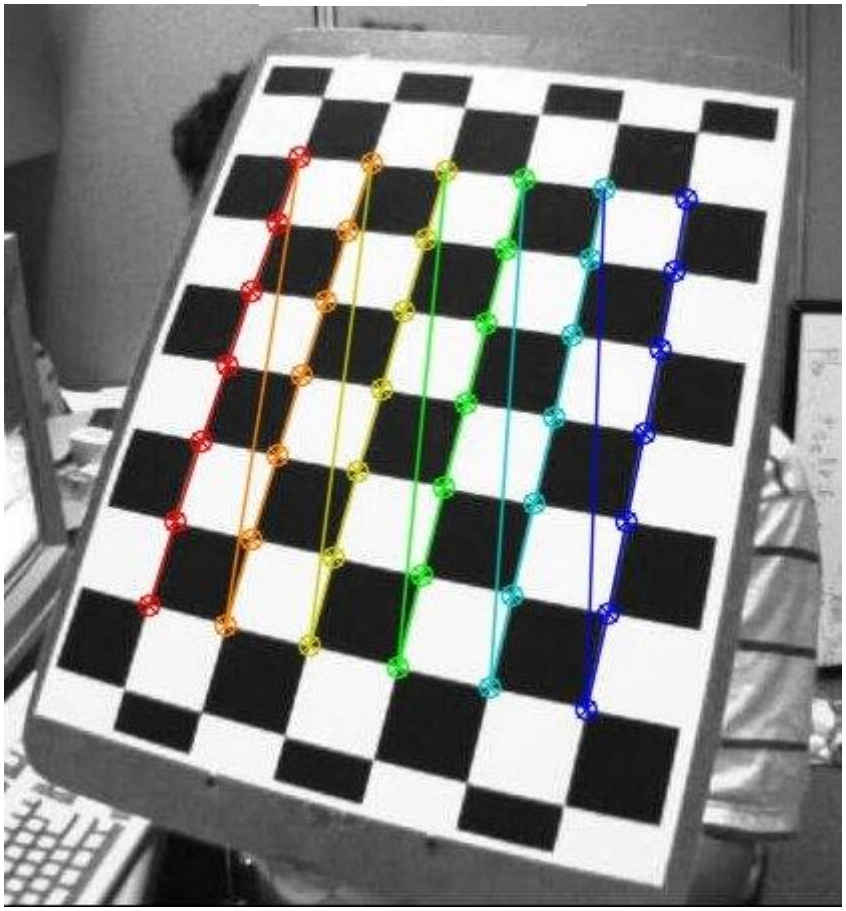
Fuente: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

2.2 Procedimiento

La primera fase es encontrar los parámetros intrínsecos y extrínsecos de la cámara "Los parámetros intrínsecos son específicos de una cámara. Incluye información como distancia, centros, etc." que dirá como está construida y dará la característica para calibrar la distorsión causada por la curvatura. Los parámetros extrínsecos corresponden a los vectores de rotación y traslación que traducen las coordenadas de un punto 3D a un sistema de coordenadas del lente. Para ello usaremos el tutorial suministrado por OpenCV "Camera Calibration" (OpenCV, 2014)

Para el caso del robot manipulador se usará una cámara estéreo para ello tendremos que proporcionar algunas imágenes de prueba que estén bien definidas. Lo aconsejable es usar un tablero de ajedrez. Necesitaríamos al menos 10 patrones de prueba. *(ver figura 6)*

Figura 6: Tablero #2.



Fuente: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

2.3 Código para mejorar la precisión

```
importar numpy como np
importar cv2
importar glob

# criterio de terminación
```

```
        criterio = ( cv2 . TERM_CRITERIA_EPS + cv2 . TERM_CRITERIA_MAX_ITER , 30 , 0.001 )

        # preparar puntos de objeto, como (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
        objp = np . ceros (( 6 * 7 , 3 ), np . float32 )
        objp [:,: 2 ] = np . mgred [ 0 : 7 , 0 : 6 ] . t _ reformar ( - 1 , 2 )

        # Matrices para almacenar puntos de objetos y puntos de imagen de todas las imágenes.
        objpoints = [] # Punto 3d en el espacio del mundo real
        imgpoints = [] # Puntos 2d en el plano de la imagen.

        imágenes = globo . globo ( '*.jpg' )

        para fname en imágenes :
            img = cv2 . imread ( fname )
            gris = cv2 . cvtColor ( img , cv2 . COLOR_BGR2GRAY )

            # Encuentra las esquinas del tablero de ajedrez
            ret , corners = cv2 . findChessboardCorners ( gris , ( 7 , 6 ), Ninguno )

            # Si lo encuentra, agregue puntos de objeto, puntos de imagen (después de refinarlos)
            if ret == True :
                objpoints . agregar ( objp )

                cv2 . cornerSubPix ( gris , esquinas , ( 11 , 11 ), ( -1 , -1 ) , criterios ) imgpoints . agregar ( esquinas )

                # Dibuja y muestra las esquinas
```

cv2 . dibujarEsquinasTableroDeAjedrez (img , (7 , 6), esquinas2 , ret)

cv2 . imshow ('img' , img)

cv2 . tecla de espera (500)

cv2 . destruir todas las ventanas ()

2.4 Calibración:

Entonces, ahora que tenemos nuestros puntos de objeto y puntos de imagen, estamos listos para la calibración. Para eso usamos la función, cv2.calibrateCamera() . Devuelve la matriz de la cámara, los coeficientes de distorsión, los vectores de rotación y traslación, etc.

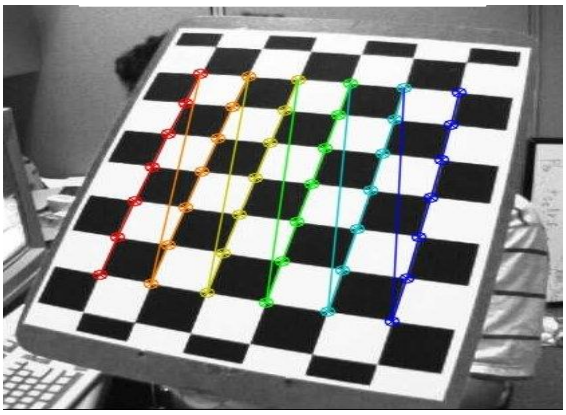
```
ret , mtx , dist , rvecs , tvecs = cv2 . calibrateCamera ( objpoints , imgpoints , gray . forma [:: - 1 ], Ninguno , Ninguno )
```

Usamos cv2.undistort() para la no distorsionar la imagen de la siguiente manera.

```
# no distorsionar
dst = cv2 . no distorsionar ( img , mtx , dist , Ninguno , newcameramtx )

# recortar la imagen
x , y , w , h = roi
dst = dst [ y : y + h , x : x + w ]
cv2 . imwrite ( 'calibresult.png' , dst )
```

2.5 Resultado. *(ver figura 7)*

Figura 7: Resultado Tablero.

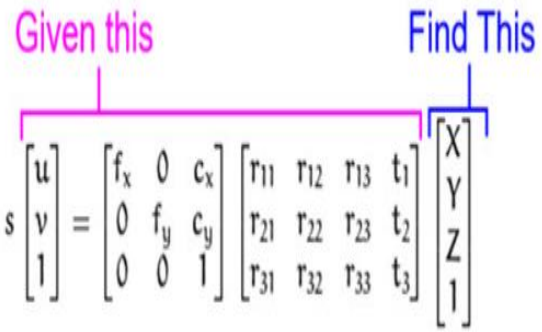


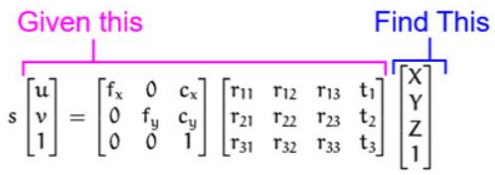
Fuente: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

3. Reconstrucción 3D

Validando los análisis facilitados en los repositorios de OpenVC, es crucial comprender el funcionamiento de las cámaras y como procesan e interpretan las imágenes. *(ver figura 8)*

Figura 8: Gráfica de calculos.





Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

Donde:

(X, Y, Z) son coordenadas de un punto en el plano cartesiano

(u,v) son coordenadas de un punto proyectado en pixeles

(T) es una matriz de parámetros intrínsecos

(Cx, Cy) es un punto principal usado en el centro de la imagen

(Fx, Fy) es una distancia focal expresada en pixeles

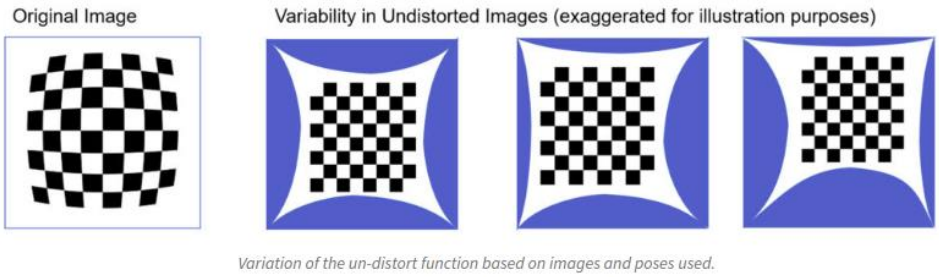
Para hacerlo funcionar, se añade un factor de escala y una matriz de cámara, para llegar a “u” y “v”, el cual permitirá solucionar para X, Y y Z de la siguiente manera:

$$\left(s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} A^{-1} - t \right) R^{-1} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

3.1 Calibración de la Cámara Intrínseca

Inicialmente se tienen que utilizar los parámetros intrínsecos de la cámara que se esté usando para lograr calibrar la cámara, ya que el mismo lente de la cámara, por su curvatura provoca la distorsión de imagen del tablero y la interpreta de diferentes maneras. *(ver figura 9)*

Figura 9: Vizualización de la cámara



Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

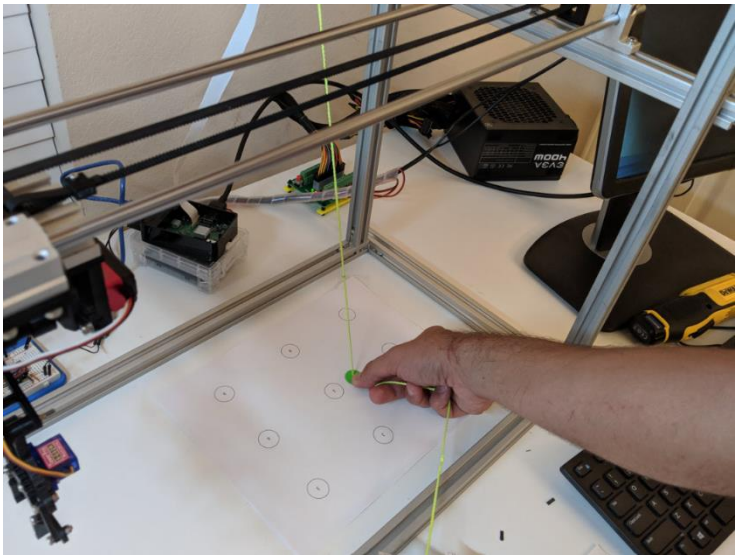
Para solucionar este error, se sugiere hacer un entrenamiento de al menos unas 40 imágenes para que se logre enfocar bien el tablero y establecerle al tablero las mismas coordenadas iniciales de la cámara.

3.2 Calibración de la perspectiva.

Primero se identifican los valores de Cx, Cy y z, si se refiere al de la cámara posicionada son iguales a Cx=628 y Cy= 342.

Luego se mide con un string el valor de Z. *(ver figura 10)*

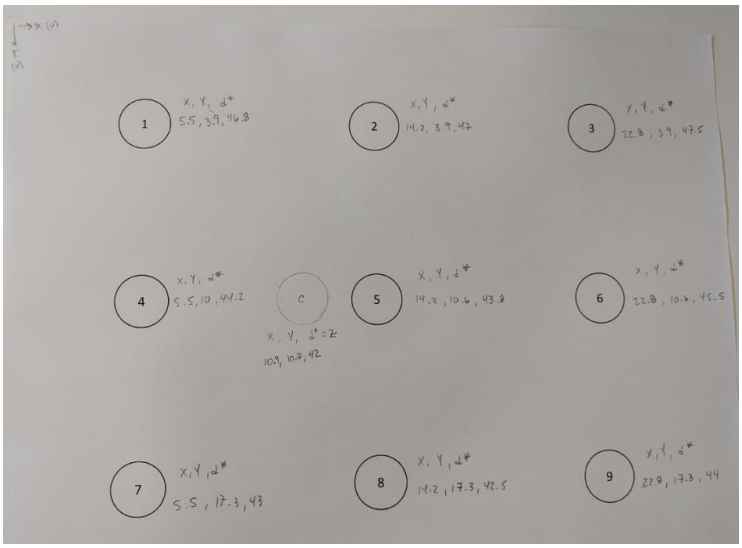
Figura 10: Visualización de entrenamiento.



Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

Para el caso de ejemplo este fue de alrededor de 43 cm el valor de Z para luego ser proyectado en la pose. *(ver figura 11)*

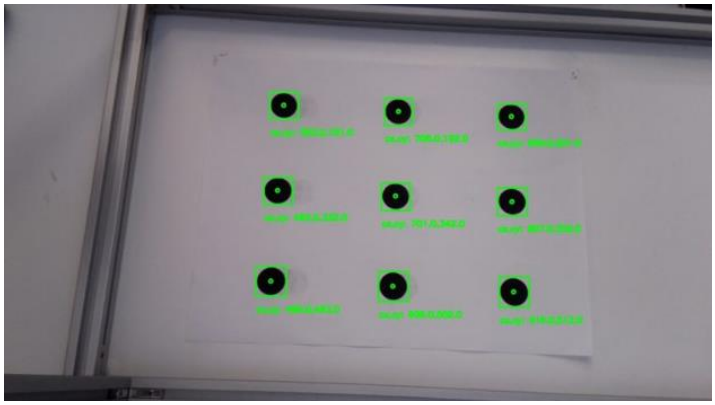
Figura 11: Visualización de entrenamiento.



Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

Luego se repite el procedimiento, el cual dará las coordenadas para los centros de las circunferencias, para cualquier otro parámetro en el espacio de trabajo se emplea trigonometría con los datos ya obtenidos y automáticamente con programación. *(ver figura 12)*

Figura 12: Visualización de entrenamiento.



Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

Con los 9 datos obtenidos ya se puede proceder con el entrenamiento del código para obtener la perspectiva. *(ver figura 13)*

Figura 13: Visualización de código.

```
#ENTER (X,Y,d*)
#d* is the distance from your point to the camera lens. (d* = Z for the
camera center)

X_center=10.9
Y_center=10.7
Z_center=43.4
worldPoints=np.array([[X_center,Y_center,Z_center],
                      [5.5,3.9,46.8],
                      [14.2,3.9,47.0],
                      [22.8,3.9,47.4],
                      [5.5,10.6,44.2],
                      [14.2,10.6,43.8],
                      [22.8,10.6,44.8],
                      [5.5,17.3,43],
                      [14.2,17.3,42.5],
                      [22.8,17.3,44.4]], dtype=np.float32)

#[u,v] center + 9 Image points
imagePoints=np.array([[cx,cy],
                      [502,185],
                      [700,197],
                      [894,208],
                      [491,331],
                      [695,342],
                      [896,353],
                      [478,487],
                      [691,497],
                      [900,508]], dtype=np.float32)
```

Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

3.2 Cálculo de X Y y Z del mundo real, a partir de la calibración de la imagen o base.

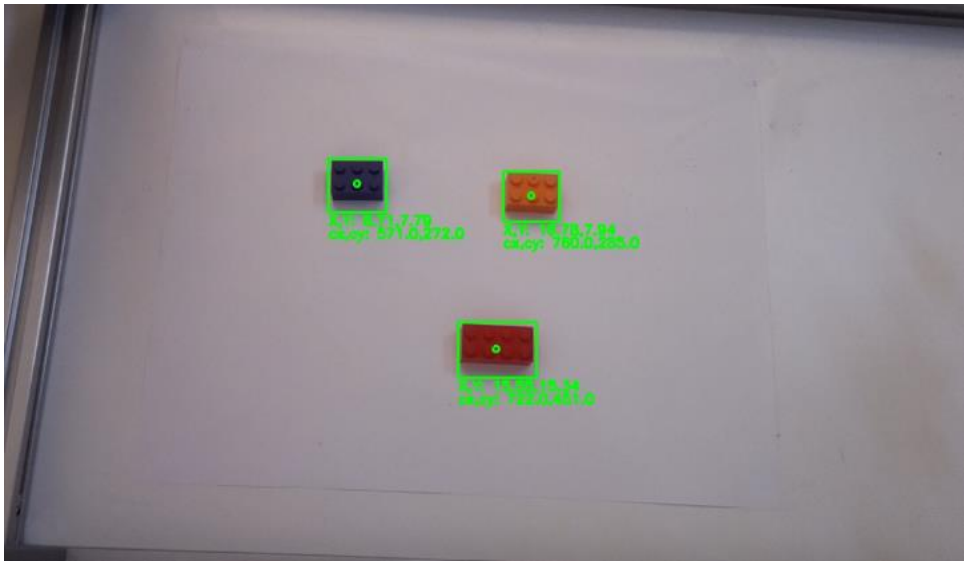
Figura 14: Visualización de código.

```
def calculate_XYZ(self,u,v):

        #Solve: From Image Pixels, find World Points

        uv_1=np.array([[u,v,1]], dtype=np.float32)
        uv_1=uv_1.T
        suv_1=self.scalingfactor*uv_1
        xyz_c=self.inverse_newcam_mtx.dot(suv_1)
        xyz_c=xyz_c-self.tvec1
        XYZ=self.inverse_R_mtx.dot(xyz_c)

return XYZ
```

Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

Una vez validado el modelo de entrenamiento, se crean 2 archivos, el del modelo ya entrenado y de las nuevas a procesar. *(ver figura 15)*

Figura 15: Resultado.



Fuente: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

Brindando como resultado las coordenadas en aplicación de objetos reales.

4. Referencias

fdxLABS. (2019, Abril 10). Recuperado de https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

OpenCV. (2014, Noviembre 10). Recuperado de https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration