# 1 Cálculo de coordenadas en mm

A continuacion se resuelve la ecuacion que tiene como resultado el paso de coordenadas en unidades de pixel a unidades del mundo real. Finalmente, a partir de los valores conocidos de los bloques, se estiman los errores

[104]:
```
#Si queremos que las imágenes sean mostradas en una ventana emergente quitar el␣
 ↪inline
%matplotlib inline

# OpenCV-Python utiliza NumPy para el manejo de imágenes
import numpy as np
# cv2 es el módulo python para acceder a OpenCV
import cv2 as cv
# Usamos las poderosas herramientas de graficación de matplotlib para mostrar␣
 ↪imágenes, perfiles, histogramas, etc
import matplotlib.pyplot as plt
# para acceder a la ruta de las imagenes
import glob
```

[105]:
```
%store -r mtx
print("K= ",mtx,'\n')
%store -r dist
print("dist coeff =",dist,'\n')
%store -r R
print("R = ",R,'\n')
%store -r tvecs
print("traslacion= ",tvecs,'\n')
%store -r centros
print("centros (u,v) =",centros,'\n')
%store -r vertices
print("vertices =",vertices)
```

```
K=  [[812.10141061   0.         316.58328056]
 [  0.         812.44382868 247.47338425]
 [  0.           0.           1.        ]]

dist coeff = [[0.02314617 0.         0.         0.         0.        ]]

R =  [[ 0.99922992 -0.0077504   0.03846435]
 [ 0.00790713  0.99996104 -0.00392445]
 [-0.03843243  0.00422557  0.99925227]]

traslacion=  [[-109.36482532]
 [-153.68304707]
 [ 702.90119809]]

centros (u,v) = [(352.4909973144531, 129.95880126953125), (171.67794799804688,
```

```
145.92718505859375), (226.46409606933594, 130.07369995117188),
(270.3270568847656, 130.83819580078125), (366.16229248046875,
126.82916259765625), (305.78240966796875, 226.9926300048828),
(375.8201904296875, 319.88763427734375), (299.1384582519531, 157.4621124267578),
(375.82025146484375, 319.8876953125), (254.31228637695312, 126.7426986694336),
(415.2177429199219, 152.42982482910156), (179.4128875732422, 362.5978698730469),
(300.0484619140625, 356.8919677734375), (409.53369140625, 359.77203369140625),
(393.03448486328125, 272.4137878417969), (280.3117980957031, 267.0211181640625),
(151.64498901367188, 259.4842834472656)]

vertices = [array([[321.90222 , 208.31345 ],
       [271.39288 , 152.27963 ],
       [383.07977 ,  51.604156],
       [433.5891  , 107.63797 ]], dtype=float32), array([[168.04597 , 229.96355
],
       [106.09865 ,  93.25218 ],
       [175.30992 ,  61.890823],
       [237.25725 , 198.60219 ]], dtype=float32), array([[210.35684 , 212.46565
],
       [149.8064  , 164.3005  ],
       [242.57135 ,  47.681747],
       [303.1218  ,  95.84689 ]], dtype=float32), array([[247.81248 , 211.8193
],
       [218.95268 ,  64.313614],
       [292.8416  ,  49.857086],
       [321.70142 , 197.36278 ]], dtype=float32), array([[380.20132 , 210.45123
],
       [307.83524 , 188.37344 ],
       [352.12326 ,  43.207092],
       [424.48935 ,  65.28488 ]], dtype=float32), array([[348.8114 , 298.77014],
       [222.45775, 219.21414],
       [262.75342, 155.21512],
       [389.10706, 234.77112]], dtype=float32), array([[419.50555, 392.191  ],
       [291.68536, 312.30338],
       [332.13483, 247.58426],
       [459.95502, 327.4719 ]], dtype=float32), array([[320.88733, 238.0666 ],
       [221.37773, 127.07515],
       [277.3896 ,  76.85762],
       [376.89917, 187.84908]], dtype=float32), array([[419.50565, 392.19107],
       [291.6854 , 312.30344],
       [332.13486, 247.58432],
       [459.9551 , 327.47195]], dtype=float32), array([[231.82712, 207.81824],
       [202.87485,  60.16179],
       [276.79745,  45.66716],
       [305.74973, 193.32361]], dtype=float32), array([[411.7874 , 236.53778],
       [349.12018, 100.3047 ],
       [418.64807,  68.32187],
       [481.3153 , 204.55495]], dtype=float32), array([[173.06593 , 446.71292 ],
```

```
[113.903244, 309.45547 ],
[185.75984 , 278.48282 ],
[244.92253 , 415.74026 ]], dtype=float32), array([[271.8835 , 436.06635],
[219.66348, 381.38885],
[328.2134 , 277.7176 ],
[380.43344, 332.39508]], dtype=float32), array([[415.00522, 444.58032],
[338.40414, 313.26428],
[404.06216, 274.96375],
[480.66324, 406.2798 ]], dtype=float32), array([[400.34485, 356.13794],
[330.     , 328.     ],
[385.72412, 188.68964],
[456.06897, 216.82758]], dtype=float32), array([[270.98685, 350.1226 ],
[218.88757, 210.27719],
[289.63675, 183.91965],
[341.73602, 323.76505]], dtype=float32), array([[163.22197 , 342.64627 ],
[ 90.144585, 316.64758 ],
[140.06801 , 176.3223  ],
[213.14539 , 202.32098 ]], dtype=float32)]
```

[106]:
```python
# R^-1
Inv_R = np.linalg.inv(R)
#K^-1
Inv_K = np.linalg.inv(mtx)

def calcularCoordenadasDelMundo(u,v):
    #Ecuacion a resolver:

    #s * [uv1] = K * [R|t] * XYZ = K* (R*XYZ1 + t)

    #  (s * [uv1] * K^-1 - t) * R^-1 = XYZ1

    # s * [uv1] * K^-1 * R^-1 - t * R^-1 = XYZ1

    #genero el vector [uv1] y lo traspongo
    uv = np.array([[u,v,1]], dtype=np.float).T

    # t * R^-1
    res_parcial_d = Inv_R.dot(tvecs)
    # [uv1]*K^-1*R^-1
    res_parcial_i = Inv_R.dot(Inv_K.dot(uv))

    # Resuelvo s (asumo Z=0). cociente de z
    s = res_parcial_d[2][0]/res_parcial_i[2][0]


    XYZ1 = Inv_R.dot( s * Inv_K.dot(uv) - tvecs)
```

```
    return XYZ1
```

[111]:
```
puntos_mm = list()
print("CENTROS =",centros,'\n')
for centro in centros:
    #solo guardo los x e y
    puntos_mm.append(calcularCoordenadasDelMundo(centro[0],centro[1])[0:2])

print("CENTROS EN MM =",puntos_mm,'\n')

%store puntos_mm
```

```
CENTROS = [(352.4909973144531, 129.95880126953125), (171.67794799804688,
145.92718505859375), (226.46409606933594, 130.07369995117188),
(270.3270568847656, 130.83819580078125), (366.16229248046875,
126.82916259765625), (305.78240966796875, 226.9926300048828),
(375.8201904296875, 319.88763427734375), (299.1384582519531, 157.4621124267578),
(375.82025146484375, 319.8876953125), (254.31228637695312, 126.7426986694336),
(415.2177429199219, 152.42982482910156), (179.4128875732422, 362.5978698730469),
(300.0484619140625, 356.8919677734375), (409.53369140625, 359.77203369140625),
(393.03448486328125, 272.4137878417969), (280.3117980957031, 267.0211181640625),
(151.64498901367188, 259.4842834472656)]

CENTROS EN MM = [array([[140.72338402],
       [ 51.65307928]]), array([[-15.71482265],
       [ 65.8449864 ]]), array([[31.90295003],
       [52.00766637]]), array([[69.93027299],
       [52.57693898]]), array([[152.42881176],
       [ 48.94124876]]), array([[101.18660975],
       [135.25253347]]), array([[162.03617704],
       [214.5864881 ]]), array([[94.99508012],
       [75.42905962]]), array([[162.03622977],
       [214.58653993]]), array([[56.03995619],
       [49.07863356]]), array([[194.56264559],
       [ 70.75840939]]), array([[ -7.63206634],
       [253.54878862]]), array([[ 97.09898517],
       [247.22857946]]), array([[191.16873475],
       [248.46802507]]), array([[176.44864058],
       [173.68663093]]), array([[ 79.45305979],
       [169.91731958]]), array([[-32.54105325],
       [164.36697199]])]

Stored 'puntos_mm' (list)
```

[108]:
```
vertices_mm= list()

for conjunto_vertices in vertices:
```

```
    V = list()

    vertice0 =␣
↪calcularCoordenadasDelMundo(conjunto_vertices[0][0],conjunto_vertices[0][1])[0:
↪2]
    vertice1 =␣
↪calcularCoordenadasDelMundo(conjunto_vertices[1][0],conjunto_vertices[1][1])[0:
↪2]
    vertice2 =␣
↪calcularCoordenadasDelMundo(conjunto_vertices[2][0],conjunto_vertices[2][1])[0:
↪2]
    vertice3 =␣
↪calcularCoordenadasDelMundo(conjunto_vertices[3][0],conjunto_vertices[3][1])[0:
↪2]

    V.append(vertice0)
    V.append(vertice1)
    V.append(vertice2)
    V.append(vertice3)

    vertices_mm.append(V)
```

[109]:
```
#mido dimensiones
dimensiones_mm = list()

for vers in vertices_mm:
    dimension = list()
    ancho = np.sqrt((vers[1][0]-vers[0][0])**2+(vers[1][1]-vers[0][1])**2)
    alto =  np.sqrt((vers[3][0]-vers[0][0])**2+(vers[3][1]-vers[0][1])**2)
    tmp=0
    if(ancho>alto):
        tmp = alto
        alto = ancho
        ancho = tmp
    dimension.append(alto)
    dimension.append(ancho)
    dimensiones_mm.append(dimension)
```

[110]:
```
base_real = 65
altura_real = 130

error_relativo_x = list()
error_relativo_y = list()
muestras = list()

for i in range(len(dimensiones_mm)):
```

```
    muestras.append(i)
    error_relativo_x.append((abs(altura_real - dimensiones_mm[i][0])/
 ↪dimensiones_mm[i][0])*100)
    error_relativo_y.append((abs(base_real - dimensiones_mm[i][1])/
 ↪dimensiones_mm[i][1])*100)

plt.plot(muestras,error_relativo_x, label = 'errores relativos en base (%)')
plt.ylabel('error relativo porcentual')

plt.plot(muestras,error_relativo_y, label = 'errores relativos en altura (%)')
plt.legend()
plt.show()
```

## 2 Algoritmo de validación

Para corroborar la posición de los centros, por ejemplo, podría hacerse de la siguiente manera. Medir desde el origen de la terna de trabajo hasta el centro correspondiente (en unidades de pixel). Medir los lados del bloque en unidades de pixel con el objetivo de obtener la relacion mm/pixel (son sabidas las dimensiones de los bloques). Multiplicar la distancia original por el factor mm/pixel. Sobre una imagen del patrón usado para la calibración extrinseca, medir desde el origen de la terna en unidades de mm y ubicar el centro.

Aqui podria verse su ubicación en unidades de pixel y compararla con la original, o si se hallan los vertices en vez del centro, superponer las imagenes (máscara mediante) y verificar que el cociente entre las areas de los bloques sea lo mas cercana a 1 posible.