lab3

May 19, 2021

[1]:
```
import os
from typing import List

import cv2
import numpy as np
from numpy import linalg as LA
from sympy import Matrix, Symbol

import matplotlib
matplotlib.use('qt5agg')
import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (12, 12)
plt.rcParams['figure.dpi'] = 120
```

Note: SymPy is optional. Without `sympy`, Some cells will fail to run, but this will not affect the result as SymPy is only used to display the matrices in LaTeX.

[2]:
```
# https://stackoverflow.com/questions/48491577/
 ↪printing-the-output-rounded-to-3-decimals-in-sympy
def round_expr(expr, num_digits):
    return expr.xreplace({n : round(n, num_digits) for n in expr.atoms(Number)})
```

1 Task 1: 3D-2D Camera Calibration

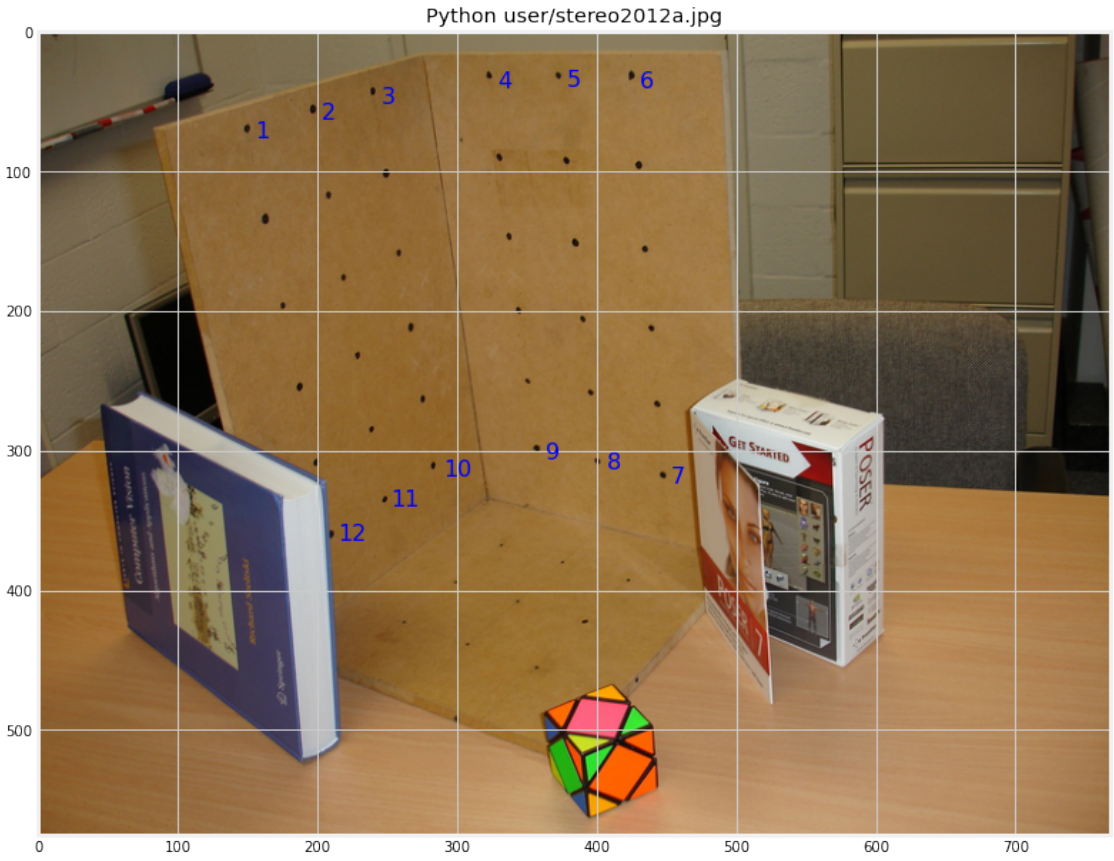
[3]:
```
path = 'Python user/stereo2012a.jpg'
img = plt.imread(path)
```

[4]:
```
# %matplotlib qt5
# plt.figure(figsize=(12,12))
# plt.title(path)
# plt.imshow(img)
# uv = plt.ginput(n=12)
# uv = np.array(uv)
# plt.close()
# np.save('uv', uv)
```

[5]:
```
%matplotlib inline
uv = np.load('uv.npy')
```

[6]:
```
plt.figure(figsize=(12,12))
plt.imshow(img)
# plt.xticks([])
# plt.yticks([])
for idx, pos in enumerate(uv, 1):
    plt.text(x=pos[0]+5, y=pos[1]+5, s=idx, color='blue', fontsize=16)
plt.title(path)
```

[6]: Text(0.5, 1.0, 'Python user/stereo2012a.jpg')



[7]:
```
# sorted from 1 - 12
XYZ = np.array([
    [ 0,42,21],
    [ 0,42,14],
    [ 0,42, 7],
    [ 7,42, 0],
    [14,42, 0],
```

```
    [21,42, 0],
    [21, 7, 0],
    [14, 7, 0],
    [ 7, 7, 0],
    [ 0, 7, 7],
    [ 0, 7,14],
    [ 0, 7,21]
])
```

[8]:
```
# # x, y, z = XYZ[0]
# # u, v = uv[0]

# xw = Symbol('x_w')
# yw = Symbol('y_w')
# zw = Symbol('z_w')
# u_var = Symbol('u')
# v_var = Symbol('v')
# A = Matrix([
#     [xw, yw, zw, 1, 0, 0, 0, 0, -u_var*xw, -u_var*yw, -u_var*zw, -u_var],
#     [0, 0, 0, 0, xw, yw, zw, 1, -v_var*xw, -v_var*yw, -v_var*zw, -v_var]
# ])
# # A = A.subs(xw, x)
# # A = A.subs(yw, y)
# # A = A.subs(zw, z)
# # A = A.subs(u_var, u)
# # A = A.subs(v_var, v)
# A
```

[9]:
```
def to_homogenous_coord(X: np.ndarray):
    """
    Parameters:
        X (np.ndarray): array of coordinates
    """
    n = X.shape[0]
    if len(X.shape) == 2 and 2 <= X.shape[1] <= 3:
        return np.hstack((X, np.ones(n).reshape(n, -1)))
    else:
        return X

def to_heterogenous_coord(X: np.ndarray):
    if len(X.shape) == 2:
        return X[:,:-1] / X[:,-1:]

def build_A(img_coord, world_coord):
    """Build A 2 rows at a time"""

    if len(img_coord) == 3:
```

```
        u, v, _ = img_coord
    else:
        u, v = img_coord

    if len(world_coord) == 4:
        x, y, z, _ = world_coord
    else:
        x, y, z = world_coord

    # build matrix A of 2n x 12
    A = np.array([
        [x, y, z, 1, 0, 0, 0, 0, -u*x, -u*y, -u*z, -u],
        [0, 0, 0, 0, x, y, z, 1, -v*x, -v*y, -v*z, -v]
    ])

    return A

def get_A(uv: np.ndarray, xyz: np.ndarray):
    """Iteratively build matrix A"""
    n = uv.shape[0]
    A = np.vstack([build_A(uv[i], xyz[i]) for i in range(n)])
    assert A.shape == (2*n, 12)
    return A

def calibrate(im: np.ndarray, XYZ: np.ndarray, uv: np.ndarray):
    """
    Compute the 3x4 camera calibration matrix C such that xi = C @ Xi, where Xi␣
 ↪is the world coordinate and xi is the image pixel coordinate

    Parameters:
        im: Image of the calibration target.
        XYZ: N x 3 array of XYZ coordinates of the calibration target points.
        uv: N x 2 array of image coordinates of the calibration target points.

    Returns:
        C (np.ndarray): 3 x 4 camera calibration matrix
    """
    assert XYZ.shape[0] >= 6
    assert uv.shape[0] == XYZ.shape[0]

    n = uv.shape[0]

    if uv.shape[1] == 2:
        uv = to_homogenous_coord(uv)

    if XYZ.shape[1] == 3:
        XYZ = to_homogenous_coord(XYZ)
```

```
    # construct matrix A and get the eigenvec corresponding to the smallest␣
↪non-zero eigenvalue
    A = get_A(uv, XYZ)
    U, S, VT = LA.svd(A)
    p = VT[-1]

    # normalize p to unit length
    p = p / (p @ p)
    C = p.reshape(3, 4)

    return C
```

1.1 1.3 Compute Calibration Matrix P and project correspondence points to pixel coordinate system

[10]:
```
XYZ = to_homogenous_coord(XYZ)
P = calibrate(img, XYZ, uv)
```

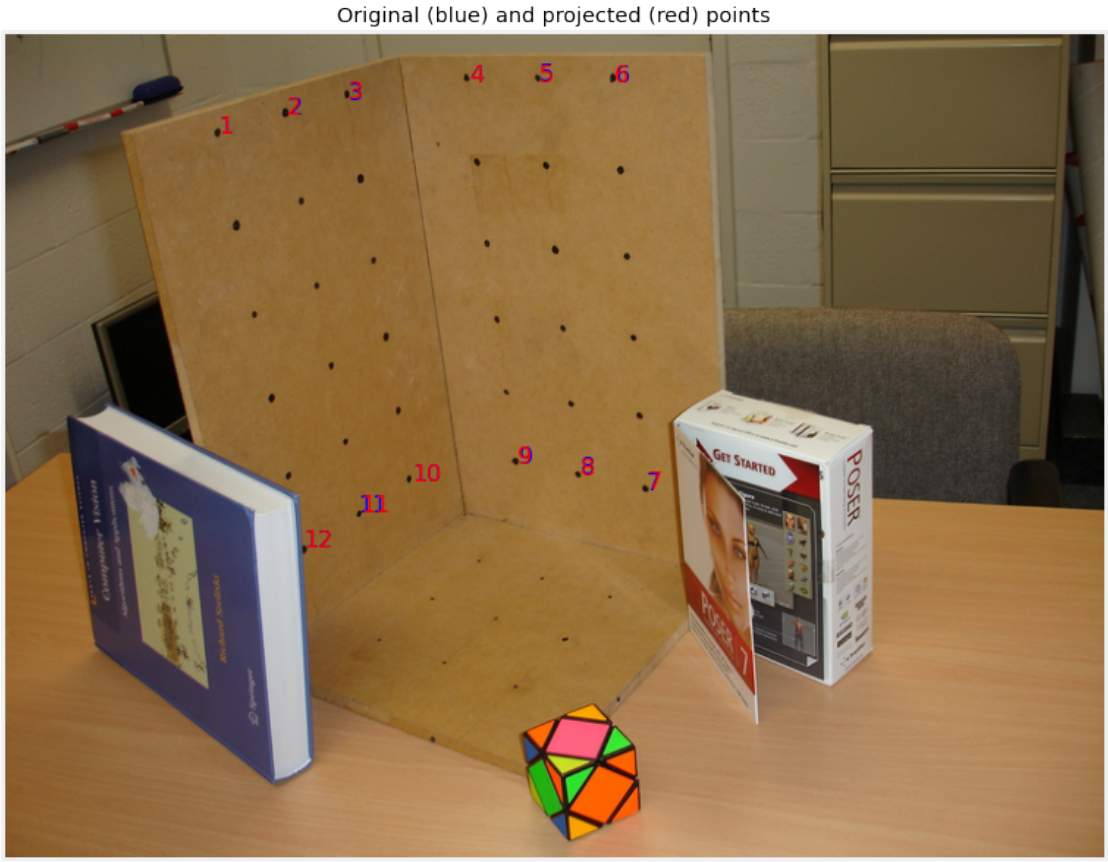
[11]:
```
Matrix(np.matrix(P).round(4))
```

[11]:
$$\begin{bmatrix} 0.0089 & -0.0041 & -0.0133 & 0.6928 \\ -0.0004 & -0.0156 & 0.0023 & 0.7208 \\ 0.0 & 0.0 & 0.0 & 0.0021 \end{bmatrix}$$

[12]:
```
# project coordinates from world to img
xyz_proj = P @ XYZ.T
xyz_proj = to_heterogenous_coord(xyz_proj.T)
```

[13]:
```
n = len(uv)
plt.figure(figsize=(12,12))
plt.imshow(img)
plt.xticks([])
plt.yticks([])
for i in range(n):
    plt.text(x=uv[i][0], y=uv[i][1], s=i+1, color='blue', fontsize=16)
    plt.text(x=xyz_proj[i][0], y=xyz_proj[i][1], s=i+1, color='red',␣
↪fontsize=16)
plt.title('Original (blue) and projected (red) points')
```

[13]:
```
Text(0.5, 1.0, 'Original (blue) and projected (red) points')
```



1.2 1.4 Extrinsic and Intrinsic Parameters

[14]:
```
# %load 'Python user/vgg_KR_from_P.py'
'''
%VGG_KR_FROM_P Extract K, R from camera matrix.
%
%    [K,R,t] = VGG_KR_FROM_P(P [,noscale]) finds K, R, t such that P =␣
 ↪K*R*[eye(3) -t].
%    It is det(R)==1.
%    K is scaled so that K(3,3)==1 and K(1,1)>0. Optional parameter noscale␣
 ↪prevents this.
%
%    Works also generally for any P of size N-by-(N+1).
%    Works also for P of size N-by-N, then t is not computed.

% original Author: Andrew Fitzgibbon <awf@robots.ox.ac.uk> and awf
% Date: 15 May 98

% Modified by Shu.
```

```
% Date: 8 May 20
'''
import numpy as np

def vgg_rq(S):
    S = S.T
    [Q,U] = np.linalg.qr(S[::-1,::-1], mode='complete')

    Q = Q.T
    Q = Q[::-1, ::-1]
    U = U.T
    U = U[::-1, ::-1]
    if np.linalg.det(Q)<0:
        U[:,0] = -U[:,0]
        Q[0,:] = -Q[0,:]
    return U,Q


def vgg_KR_from_P(P, noscale = True):
    N = P.shape[0]
    H = P[:,0:N]
    print(N,'|', H)
    [K,R] = vgg_rq(H)
    if noscale:
        K = K / K[N-1,N-1]
        if K[0,0] < 0:
            D = np.diag([-1, -1, np.ones([1,N-2])]);
            K = K @ D
            R = D @ R

            test = K*R;
            assert (test/test[0,0] - H/H[0,0]).all() <= 1e-07

    t = np.linalg.inv(-P[:,0:N]) @ P[:,-1]
    return K, R, t

K, R, t = vgg_KR_from_P(P)
```

```
3 | [[ 8.90629859e-03 -4.14692138e-03 -1.33119809e-02]
 [-3.82757009e-04 -1.55974745e-02  2.25583330e-03]
 [-9.30249262e-06 -6.83349696e-06 -1.21812069e-05]]
```

[15]:
```
Matrix(np.matrix(K).round(4))
```

[15]:
$$\begin{bmatrix} 908.7838 & -1.4141 & 382.2355 \\ 0.0 & 892.3563 & 293.5467 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

[16]:
```
Matrix(np.matrix(R).round(4))
```

[16]:
$$\begin{bmatrix} 0.8174 & -0.1021 & -0.567 \\ 0.1568 & -0.9076 & 0.3894 \\ -0.5543 & -0.4072 & -0.7259 \end{bmatrix}$$

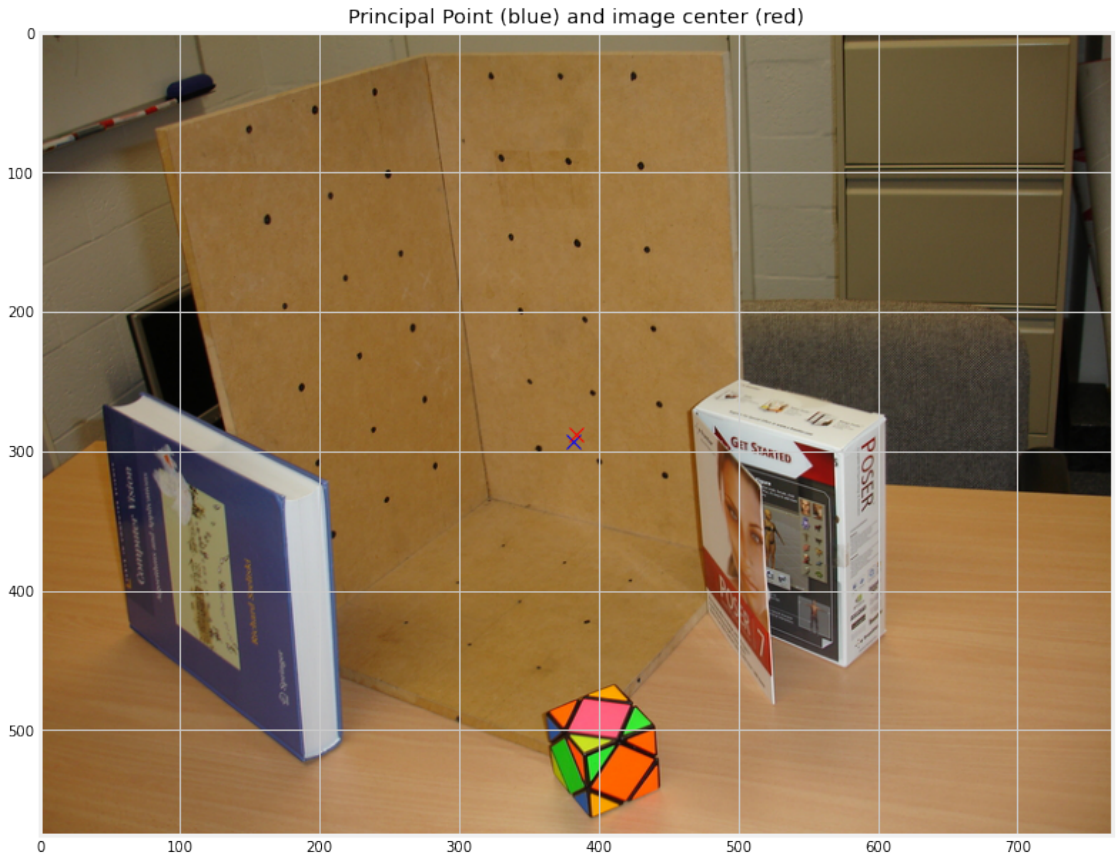
[17]:
```
Matrix(np.matrix(t).round(4))
```

[17]:
$$\begin{bmatrix} 76.4326 & 56.7049 & 85.5121 \end{bmatrix}$$

[18]:
```
h, w, _ = img.shape
```

[19]:
```
plt.figure(figsize=(12,12))
plt.imshow(img)
plt.plot(K[0][2], K[1][2], marker='x', markersize=10, color='blue')
plt.plot(w/2, h/2, marker='x', markersize=10, color='red')
plt.title('Principal Point (blue) and image center (red)')
```

[19]:
```
Text(0.5, 1.0, 'Principal Point (blue) and image center (red)')
```



1.3 1.5: Focal Length and Pitch Angle

[20]:
```
ax = K[0][0]
ay = K[1][1]
print('ax:', ax)
print('ay:', ay)
print('focal length:', np.mean([ax, ay]))
print('aspect ratio:', ay / ax)
```

```
ax: 908.7838201708981
ay: 892.356291375963
focal length: 900.5700557734306
aspect ratio: 0.9819236121613103
```

[21]:
```
# theta_abs = np.arccos((np.trace(R) - 1) / 2)
# theta_abs
```

[22]:
```
# np.pi - theta_abs
```

[23]:
```
# np.degrees(np.pi - theta_abs)
```

The position, C, of the camera expressed in world coordinates is $C = -R^{-1}T = -R^{T}T$.

$$\cos\theta = \frac{\vec{c} \cdot \vec{p}}{\|\vec{c}\| \cdot \|\vec{p}\|}$$

where $c, p \in \mathbb{R}^3$

We can get the angle using

$$\theta = \arccos(\cos(\theta))$$

[24]:
```
C = -R.T @ t.reshape(len(t), -1)
C.T
```

[24]:
```
array([[-23.96402805,  94.08800595,  83.32400034]])
```

[25]:
```
plane = np.array([7,0,7])
```

[26]:
```
theta = np.arccos((C.T @ plane) / (LA.norm(C) * LA.norm(plane)))
theta
```

[26]:
```
array([1.23654328])
```

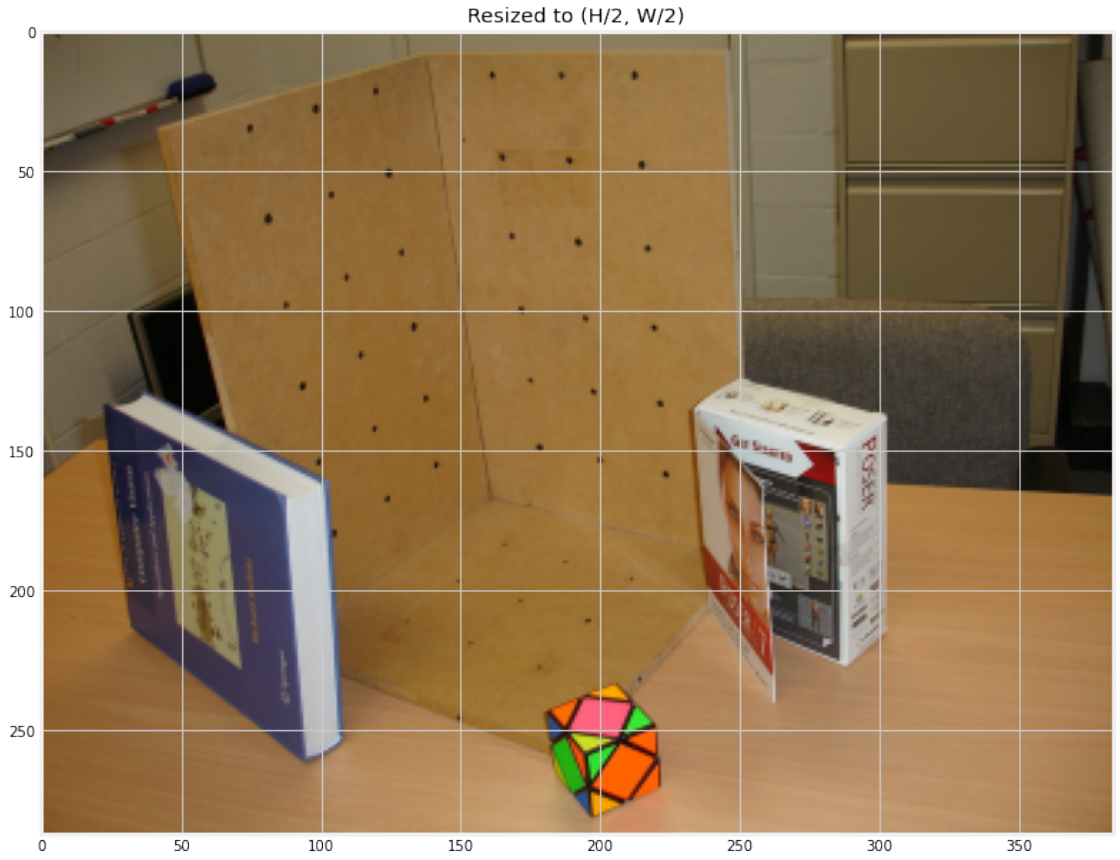
[27]:
```
np.degrees(theta)
```

[27]:
```
array([70.84871102])
```

1.4 1.6 Resize

[28]:
```
# resize to (H/2, W/2)
h, w, _ = img.shape
resized = cv2.resize(img, dsize=(w//2, h//2))
plt.figure(figsize=(12,12))
plt.imshow(resized)
plt.title('Resized to (H/2, W/2)')
```

[28]:
```
Text(0.5, 1.0, 'Resized to (H/2, W/2)')
```



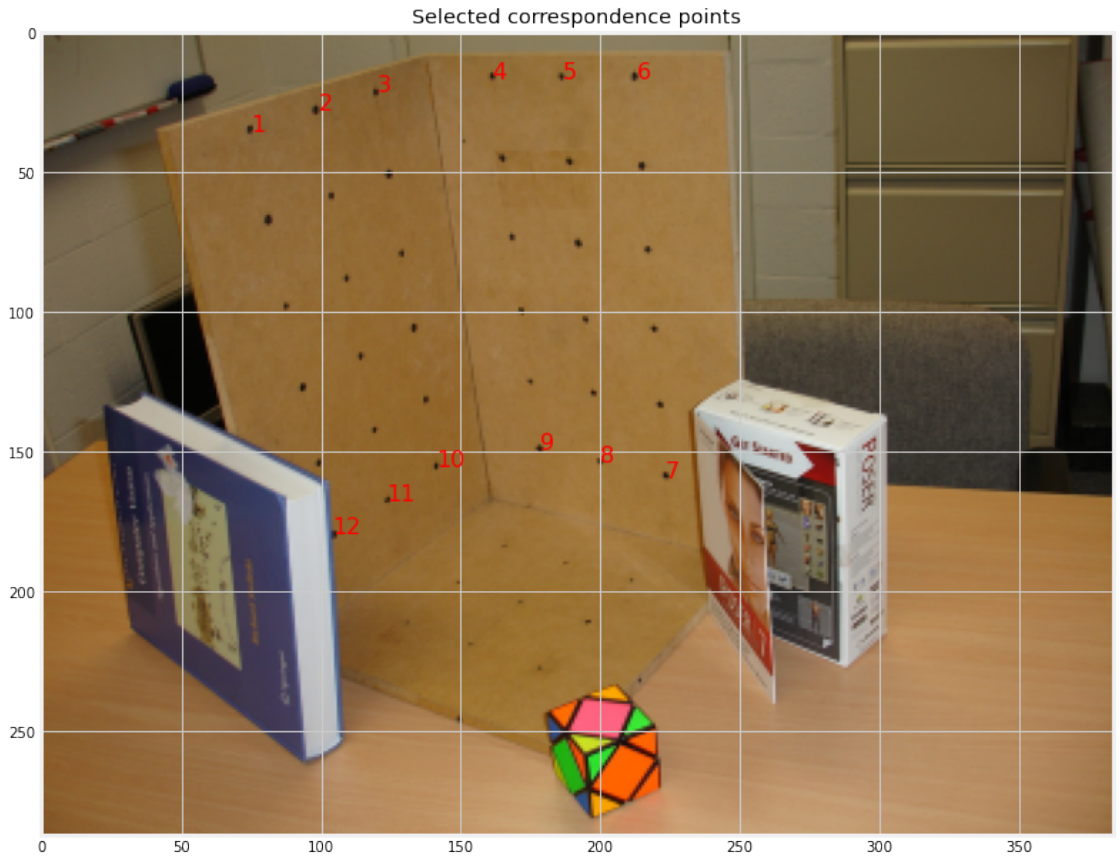
[29]:
```
img2 = resized
```

[30]:
```
# %matplotlib qt5
# plt.figure(figsize=(12,12))
# plt.imshow(img2)
# uv2 = plt.ginput(n=12)
# uv2 = np.array(uv2)
# plt.close()
# np.save('uv2', uv2)
```

[31]:
```
%matplotlib inline
uv2 = np.load('uv2.npy')
```

[32]:
```
n = len(uv2)
plt.figure(figsize=(12,12))
plt.imshow(img2)
for i in range(n):
    plt.text(x=uv2[i][0], y=uv2[i][1], s=i+1, color='red', fontsize=16)
# plt.xticks([])
# plt.yticks([])
plt.title('Selected correspondence points')
```

[32]:
```
Text(0.5, 1.0, 'Selected correspondence points')
```



[33]:
```
XYZ2 = to_homogenous_coord(XYZ)
P2 = calibrate(img2, XYZ2, uv2)
K2, R2, t2 = vgg_KR_from_P(P2)
```

```
3 | [[ 8.90660663e-03 -4.14295362e-03 -1.32672048e-02]
 [-1.54871877e-04 -1.57692989e-02  2.55101048e-03]
 [-1.93034663e-05 -1.40615018e-05 -2.49253860e-05]]
```

[34]:
```
Matrix(np.matrix(P2).round(4))
```

[34]:
$$\begin{bmatrix} 0.0089 & -0.0041 & -0.0133 & 0.693 \\ -0.0002 & -0.0158 & 0.0026 & 0.7206 \\ 0.0 & 0.0 & 0.0 & 0.0043 \end{bmatrix}$$

[35]:
```
Matrix(np.matrix(K2).round(4))
```

[35]:
$$\begin{bmatrix} 442.1749 & 1.4406 & 182.1194 \\ 0.0 & 442.5788 & 135.2311 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

[36]:
```
Matrix(np.matrix(R2).round(4))
```

[36]:
$$\begin{bmatrix} 0.8133 & -0.1007 & -0.5731 \\ 0.1607 & -0.9077 & 0.3876 \\ -0.5592 & -0.4073 & -0.7221 \end{bmatrix}$$

[37]:
```
Matrix(np.matrix(t2).round(4))
```

[37]:
$$\begin{bmatrix} 73.4314 & 58.4466 & 83.2765 \end{bmatrix}$$

2 Task 2: Two-View DLT based Homography Estimation

2.1 2.1 Code, images used, and correspondence points

[38]:
```
l = plt.imread('Python user/Left.jpg')
r = plt.imread('Python user/Right.jpg')
```

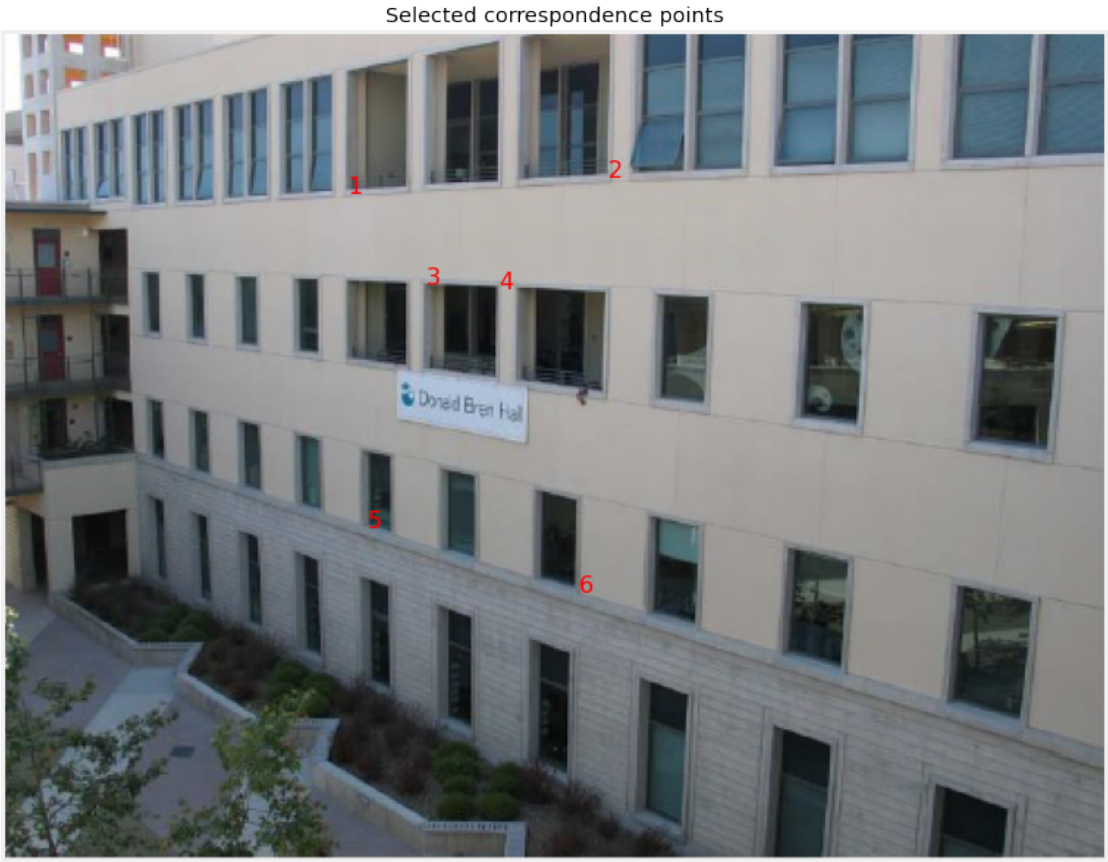
[39]:
```
n = 6
```

[40]:
```
# %matplotlib qt5
# plt.figure(figsize=(12,12))
# plt.imshow(l)
# uv_left = np.array(plt.ginput(n=n))
# plt.close()
# np.save('uv_left', uv_left)
```

[41]:
```
uv_left = np.load('uv_left.npy')
```

[42]:
```
%matplotlib inline
plt.figure(figsize=(12,12))
plt.imshow(l)
for i in range(n):
    plt.text(x=uv_left[i][0], y=uv_left[i][1], s=i+1, color='red', fontsize=16)
plt.xticks([])
plt.yticks([])
plt.title('Selected correspondence points')
```

[42]:
```
Text(0.5, 1.0, 'Selected correspondence points')
```



[43]:
```
# %matplotlib qt5
# plt.figure(figsize=(12,12))
# plt.imshow(r)
# uv_right = np.array(plt.ginput(n=n))
# plt.close()
# np.save('uv_right', uv_right)
```

[44]:
```
uv_right = np.load('uv_right.npy')
```

[45]:
```
%matplotlib inline
plt.figure(figsize=(20, 8))
plt.suptitle('Selected correspondence points', fontsize=20)

# 1
plt.subplot(1, 2, 1)
plt.imshow(l)
for i in range(n):
    plt.text(x=uv_left[i][0], y=uv_left[i][1], s=i+1, color='red', fontsize=16)
plt.xticks([])
plt.yticks([])
```

```
plt.title('Left (source)')

# 2
plt.subplot(1, 2, 2)
plt.imshow(r)
for i in range(n):
    plt.text(x=uv_right[i][0], y=uv_right[i][1], s=i+1, color='red',␣
 ↪fontsize=16)
plt.xticks([])
plt.yticks([])
plt.title('Right (dest)')

plt.tight_layout()
```

[46]:
```
def homography(u2Trans: List[float], v2Trans: List[float], uBase: List[float],␣
 ↪vBase: List[float]):
    """
    Computes the homography H applying the Direct Linear Transformation

    Parameters:
        u2Trans : Vectors with coordinates u and v of the transformed image
        v2Trans : point (p')
        uBase : vectors with coordinates u and v of the original base
        vBase : image point p
    """
    uv_src = np.vstack([u2Trans, v2Trans]).T
    uv_dest = np.vstack([uBase, vBase]).T
    return _homography(uv_src, uv_dest)

def _homography(uv_src: np.ndarray, uv_dest: np.ndarray):
    assert uv_src.shape[0] >= 6
```

```
    assert uv_src.shape[0] == uv_dest.shape[0]

    if uv_src.shape[1] == 2:
        uv_src = to_homogenous_coord(uv_src)

    if uv_dest.shape[1] == 2:
        uv_dest = to_homogenous_coord(uv_dest)

    A = get_A(uv_src, uv_dest)
    U, S, VT = LA.svd(A)
    p = VT[-1]

    # normalize p to unit length
    p = p / (p @ p)
    H = p.reshape(3, 3)

    return H

def get_A(uv_src: np.ndarray, uv_dest: np.ndarray):
    """Iteratively build matrix A for homography"""
    n = uv_src.shape[0]
    A = np.vstack([build_A_homography(uv_src[i], uv_dest[i]) for i in range(n)])
    assert A.shape == (2*n, 9)
    return A

def build_A_homography(uv_src, uv_dest):
    """Build A 2 rows at a time"""
    assert len(uv_src) == 3 and len(uv_dest) == 3

    u, v, _ = uv_src
    x, y, _ = uv_dest

    A = np.array([
        [u, v, 1, 0, 0, 0, -x*u, -x*v, -x],
        [0, 0, 0, u, v, 1, -y*u, -y*v, -y]
    ])

    return A
```

2.2 2.2 Compute homography matrix H

[47]:
```
H = homography(
    u2Trans=uv_left[:, 0].tolist(),
    v2Trans=uv_left[:, 1].tolist(),
    uBase=uv_right[:, 0].tolist(),
    vBase=uv_right[:, 1].tolist()
)
```

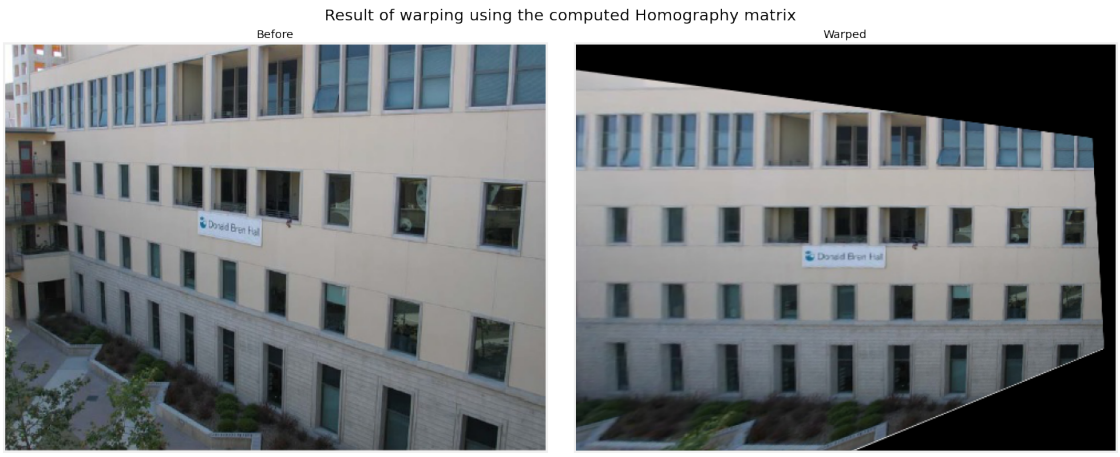
[48]:
```
Matrix(np.matrix(H).round(4))
```

[48]:
$$\begin{bmatrix} -0.0149 & 0.0004 & 0.9997 \\ -0.0024 & -0.0065 & 0.0169 \\ 0.0 & 0.0 & -0.0047 \end{bmatrix}$$

2.3 2.3 Warp image using H

[49]:
```
x, y, _ = r.shape
size = (y, x)
l_warped = cv2.warpPerspective(l, H, dsize=size)
```

[50]:
```
plt.figure(figsize=(20, 8))
plt.suptitle('Result of warping using the computed Homography matrix',␣
 ↪fontsize=20)

# 1
plt.subplot(1, 2, 1)
plt.imshow(l)
plt.xticks([])
plt.yticks([])
plt.title('Before')

# 2
plt.subplot(1, 2, 2)
plt.imshow(l_warped)
plt.xticks([])
plt.yticks([])
plt.title('Warped')

plt.tight_layout()
```



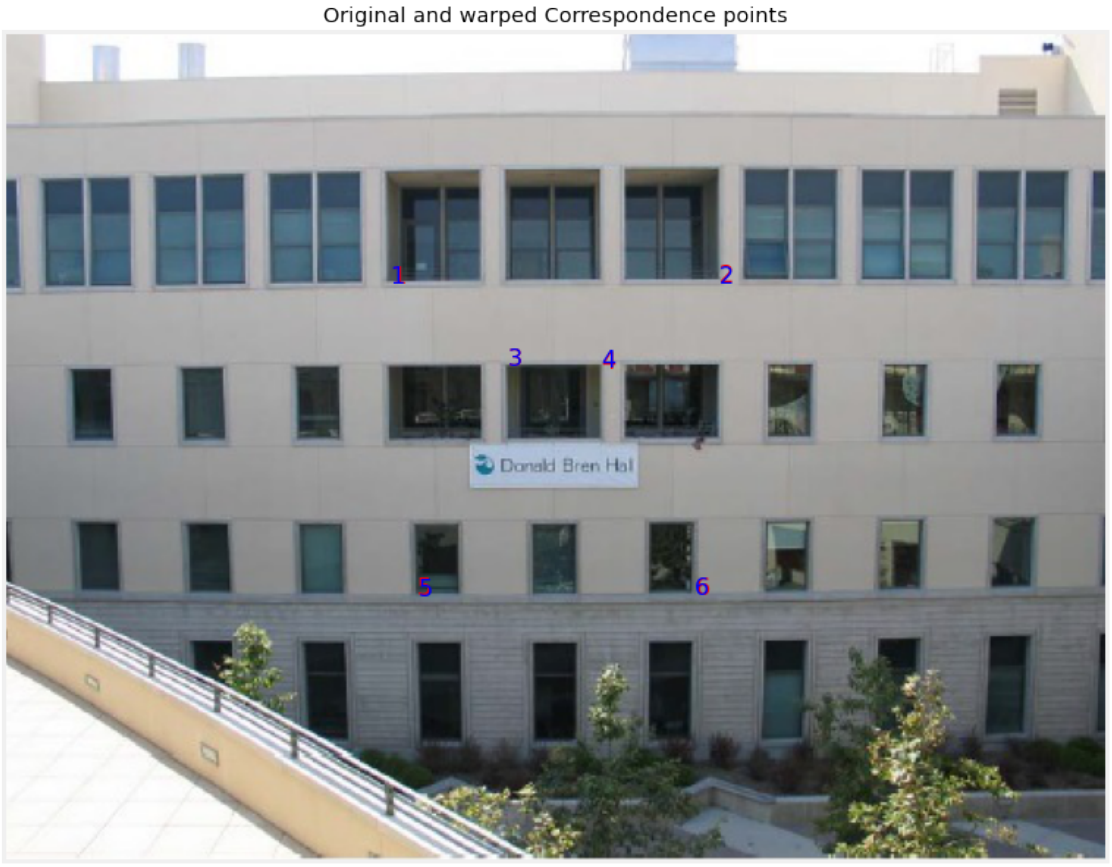
[51]:
```
plt.figure(figsize=(20, 8))
plt.suptitle('Comparison of warped (source) image with target image',␣
 ↪fontsize=20)

plt.subplot(1, 2, 1)
plt.imshow(r)
plt.xticks([])
plt.yticks([])
plt.title('Target')

plt.subplot(1, 2, 2)
plt.imshow(l_warped)
plt.xticks([])
plt.yticks([])
plt.title('Warped source image')

plt.tight_layout()
```

[52]:
```
# project img coordinates using H
uv_left = to_homogenous_coord(uv_left)
uv_left_proj = H @ uv_left.T
uv_left_proj = to_heterogenous_coord(uv_left_proj.T)
```

[53]:
```
plt.figure(figsize=(12,12))
plt.imshow(r)
plt.xticks([])
plt.yticks([])
for i in range(n):
    plt.text(x=uv_right[i][0], y=uv_right[i][1], s=i+1, color='red',␣
 ↪fontsize=16)
```

```
    plt.text(x=uv_left_proj[i][0], y=uv_left_proj[i][1], s=i+1, color='blue',␣
 ↪fontsize=16)
plt.title('Original and warped Correspondence points')
```

[53]:
```
Text(0.5, 1.0, 'Original and warped Correspondence points')
```



[54]:
```
# distance in pixels
dist = np.sqrt(np.sum(np.square(uv_right - uv_left_proj), 1))
dist
```

[54]:
```
array([0.48353272, 0.59920041, 0.27162193, 1.05412312, 0.54916902,
       0.2332485 ])
```

[ ]:
```

```