Kai Hirota

SID: u7233149

# Task 1: 3D-2D Camera Calibration

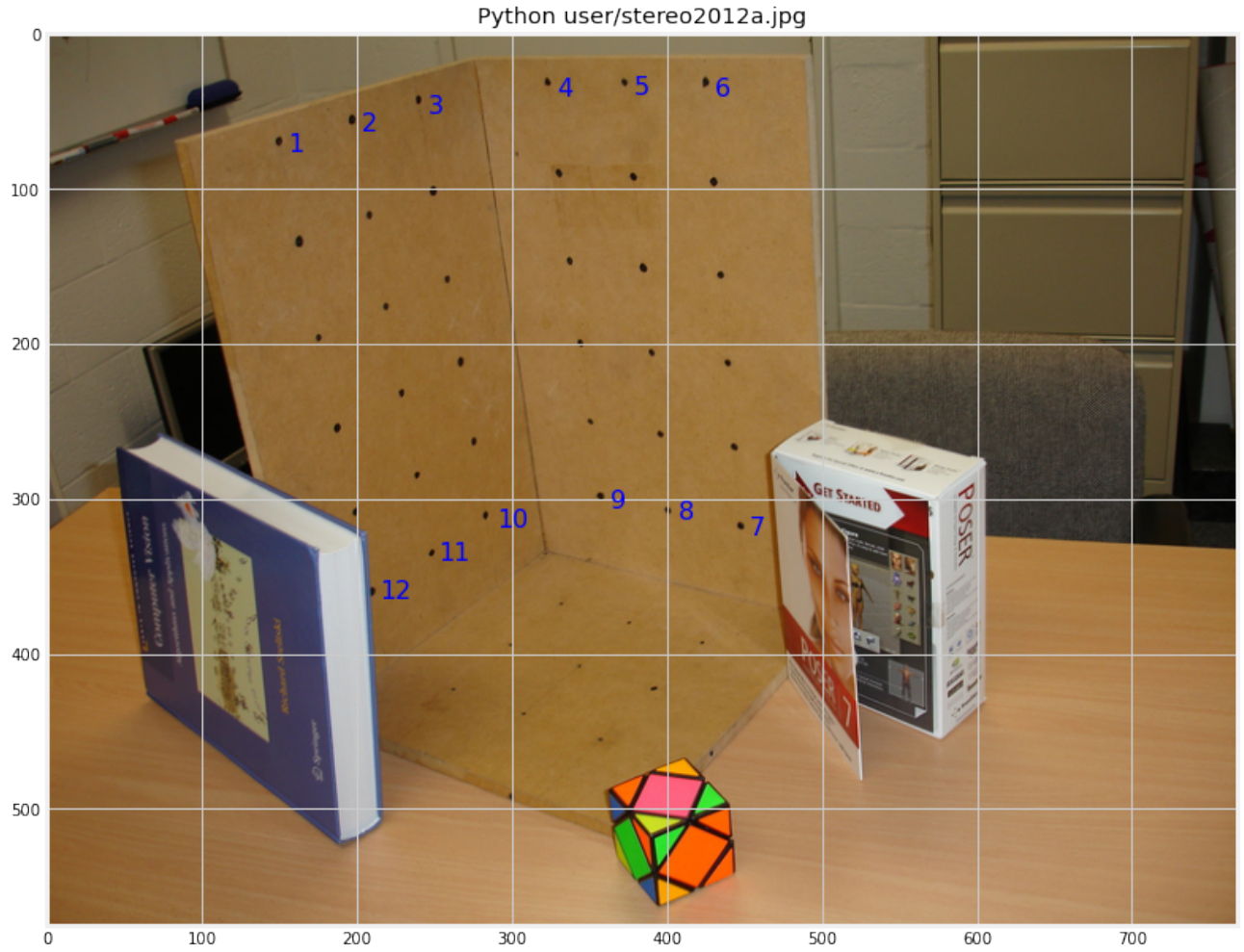
## 1.1 Code for calibration

```python
# helper functions not included: to_homogenous(), to_heterogenous()

def calibrate(im: np.ndarray, XYZ: np.ndarray, uv: np.ndarray):
    """
    Compute the 3×4 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]
```

```
30
31      # normalize p to unit length
32      p = p / (p @ p)
33      C = p.reshape(3, 4)
34
35      return C
36
37  def get_A(uv: np.ndarray, xyz: np.ndarray):
38      """Construct matrix A"""
39      n = uv.shape[0]
40      A = np.vstack([build_A(uv[i], xyz[i]) for i in range(n)])
41      assert A.shape == (2*n, 12)
42      return A
43
44  def build_A(img_coord, world_coord):
45      """Build A 2 rows at a time"""
46      if len(img_coord) == 3:
47          u, v, _ = img_coord
48      else:
49          u, v = img_coord
50
51      if len(world_coord) == 4:
52          x, y, z, _ = world_coord
53      else:
54          x, y, z = world_coord
55
56      # build matrix A of 2n x 12
57      A = np.array([
58          [x, y, z, 1, 0, 0, 0, 0, -u*x, -u*y, -u*z, -u],
59          [0, 0, 0, 0, x, y, z, 1, -v*x, -v*y, -v*z, -v]
60      ])
61
62      return A
63
64  ####### Normalized DLT #######
65  h, w, _ = img.shape
66
67  Tnorm = np.array([
68      [w+h, 0, w/2],
69      [0, w+h, h/2],
70      [0, 0, 1]
71  ])
72
73  eigvals, eigvecs = LA.eig( (XYZ - XYZ.mean(0)) @ (XYZ - XYZ.mean(0)).T )
74  eigvals = np.diag(1 / eigvals[:3])
75  eigvecs = eigvecs[:3, :3]
76
77  left_component = eigvecs @ eigvals @ LA.inv(eigvecs)
```

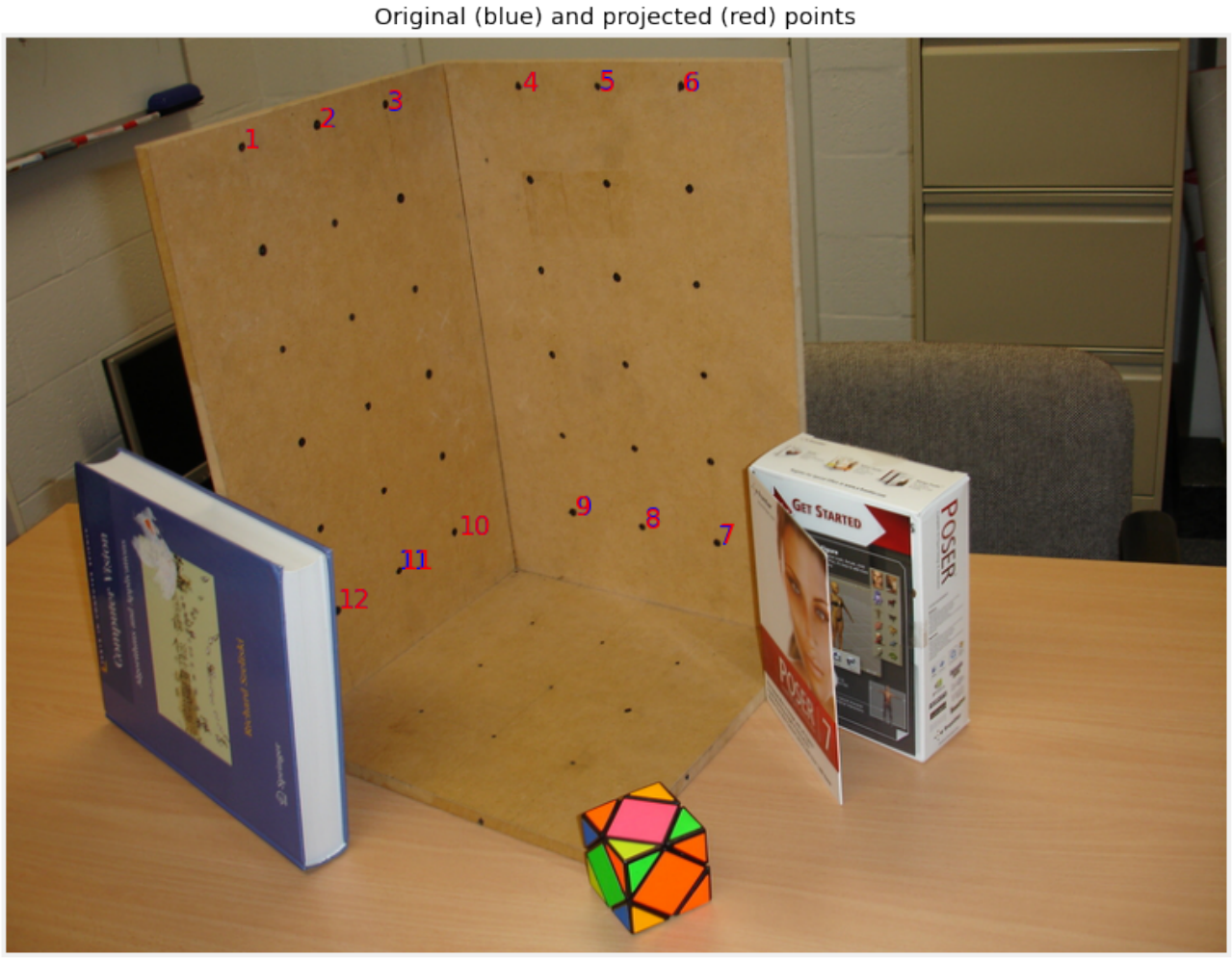
```
right_component = -eigvecs @ eigvals @ LA.inv(eigvecs) @ XYZ.mean(0)
right_component = right_component.reshape(len(right_component), -1)

Snorm = np.hstack([left_component, right_component])
Snorm = np.vstack([Snorm, [0,0,0,1]])

uv_norm = Tnorm @ to_homogenous_coord(uv).T
uv_norm = uv_norm.T

XYZ_norm = Snorm @ to_homogenous_coord(XYZ).T
XYZ_norm = XYZ_norm.T

##############################

# how to use the code
P = calibrate(img, XYZ, uv)

# if using Normalized DLT
# P = calibrate(img, XYZ_norm, uv_norm)
# P = LA.inv(Tnorm) @ P @ Snorm
# P = P / P[-1,-1]

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

## 1.2 Selected Image & Correspondence Points




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

$$P = \begin{bmatrix} 0.0089 & -0.0041 & -0.0133 & 0.6928 \\ -0.0004 & -0.0156 & 0.0023 & 0.7208 \\ 0 & 0 & 0 & 0.0021 \end{bmatrix} \quad (1)$$




Mean Squared Error (pixels): 0.5426

## 1.4 Extrinsic and Intrinsic Parameters

$$K = \begin{bmatrix} 908.7838 & -1.4141 & 382.2355 \\ 0 & 892.3563 & 293.5467 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R = \begin{bmatrix} 0.8174 & -0.1021 & -0.567 \\ 0.1568 & -0.9076 & 0.3894 \\ -0.5543 & -0.4072 & -0.7259 \end{bmatrix} \tag{3}$$

$$t = \begin{bmatrix} 76.4326 & 56.7049 & 85.5121 \end{bmatrix} \tag{4}$$

## 1.5 Focal length & pitch angle

### Focal Length

Since $K_{1,1} \neq K_{2,2}$ and $K_{1,2} \neq 0$, we have an intrinsic parameter matrix where the aspect ratio is not 1.

$$K = \begin{bmatrix} \alpha & \gamma & x_0 \\ 0 & \beta & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where

- $(x_0, y_0) =$ Principal point
- $\alpha = \alpha_x$
- $\beta = \frac{\alpha_y}{\sin\theta}$
- $\alpha_x,\ \alpha_y$ are horizontal and vertical focal length in pixel units
- $\theta = \pi/2 \implies \beta = \alpha_y$

Assuming $\theta = \pi/2$, $\alpha_x = f_x$ and $\alpha_y = f_y$. Referring to $K$ defined in (2) in section 1.4, we have

- $f_x = 908.7838$ (pixels)
- $f_y = 892.3563$ (pixels)

### Pitch Angle with respect to X-Z plane

The position, $C$, of the camera expressed in world coordinates is $C = -R^{-1}T = -R^{T}T$. Let $\vec{p}$ be a vector that lie on the X-Z plane and $\vec{c}$ be the position of the camera, with both vectors having the same basis (coordinate system). We have the geometric property of vectors:

$$\cos\theta = \frac{\vec{c} \cdot \vec{p}}{\|\vec{c}\| \cdot \|\vec{p}\|} \tag{6}$$

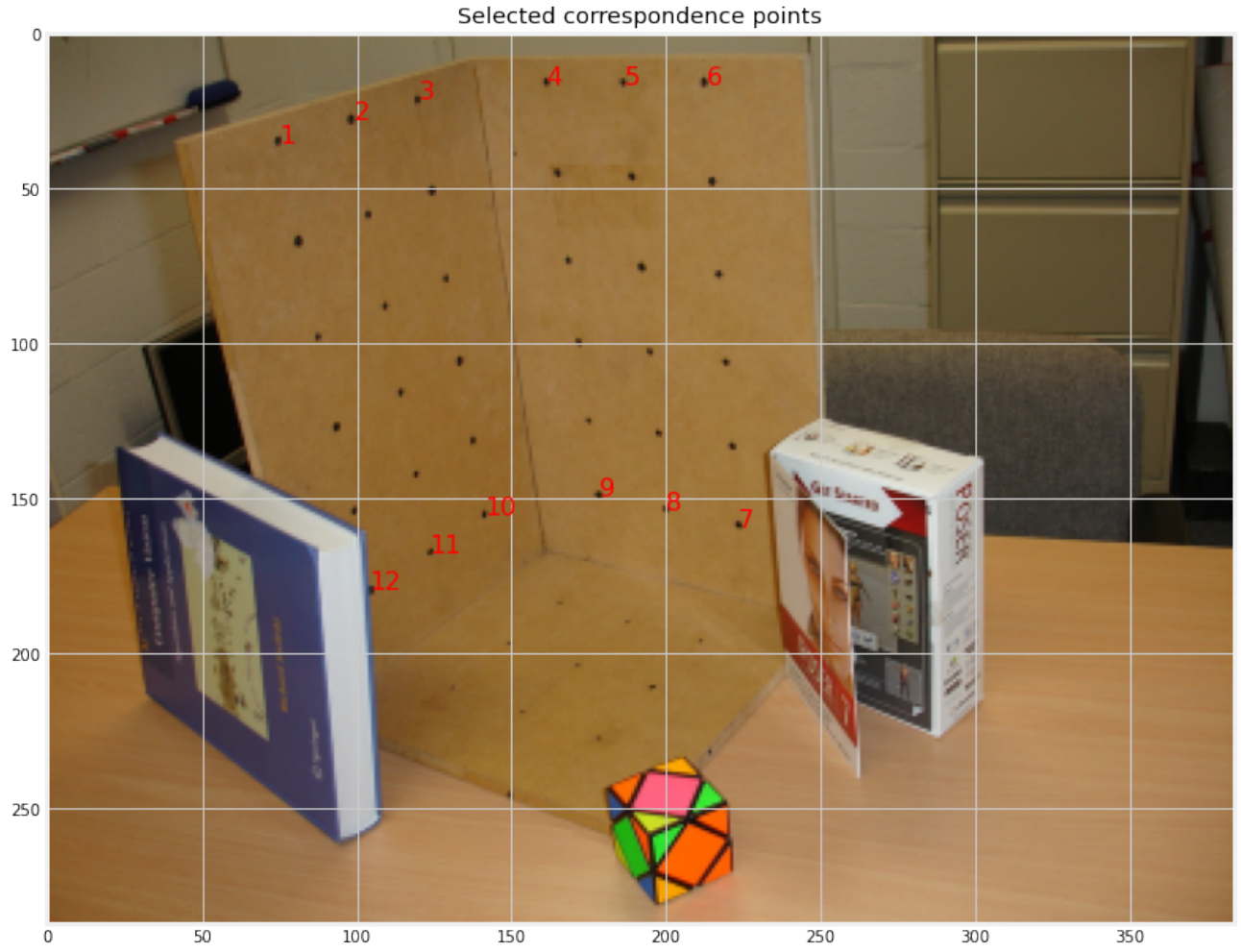
where $\vec{c}, \vec{p} \in \mathbb{R}^3$

The angle $\theta$ between $C$ and the X-Z plane can be calculated as $\theta = \arccos(\cos(\theta))$.

```
C = -R.T @ t.reshape(len(t), -1)
C.T
>> array([[-23.96402805,  94.08800595,  83.32400034]])

plane = np.array([7,0,7])
theta = np.arccos((C.T @ plane) / (LA.norm(C) * LA.norm(plane)))
theta
>> 1.23654328

np.degrees(theta)
>> 70.84871102
```

The pitch angle, or the angle between the camera's optical axis and the ground-plane, is approximately 70.85 degrees.

## 1.6 Resize




Resized image

### Calibration Matrix P' and Parameters K', R', t'

$$P' = \begin{bmatrix} 0.0089 & -0.0041 & -0.0133 & 0.693 \\ -0.0002 & -0.0158 & 0.0026 & 0.7206 \\ 0.0 & 0.0 & 0.0 & 0.0043 \end{bmatrix} \tag{7}$$

$$K' = \begin{bmatrix} 442.1749 & 1.4406 & 182.1194 \\ 0.0 & 442.5788 & 135.2311 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{8}$$

$$R' = \begin{bmatrix} 0.8133 & -0.1007 & -0.5731 \\ 0.1607 & -0.9077 & 0.3876 \\ -0.5592 & -0.4073 & -0.7221 \end{bmatrix} \tag{9}$$

$$t' = \begin{bmatrix} 73.4314 & 58.4466 & 83.2765 \end{bmatrix} \tag{10}$$

- Intrinsic parameters $K$
    - Focal length $K'_{1,1}, K'_{2,2}$ are approximately half of $K_{1,1}, K_{2,2}$ due to width and height being halved.
    - Skew parameter $|K_{1,2}| \approx |K'_{1,2}|$ since the width and height were scaled by the same factor. In other

words, resizing while keeping the aspect ratio should not skew the image.
    - Principal point $K'_{1,3}, K'_{2,3}$ are approximately half of $K_{1,3}, K_{2,3}$ due to the width and height being halved.
- Extrinsic parameters
    - $R \approx R'$ since rotation is scale-invariant if width and height are scaled by the same factor.
    - $t \approx t'$ for the same reason as $R, R'$ above.

# Task 2: Two-View DLT based homography estimation

## 2.1 Code, images used, and correspondence points

```python
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

# how to use the code
# uv_left, uv_right: numpy array containing xy coordinates of correspondence points
H = homography(
    u2Trans=uv_left[:, 0].tolist(),
    v2Trans=uv_left[:, 1].tolist(),
    uBase=uv_right[:, 0].tolist(),
    vBase=uv_right[:, 1].tolist()
)

# warp the source image l using H
# l = Left.jpg = warp source
# r = Right.jpg = warp destination
x, y, _ = r.shape
size = (y, x)
l_warped = cv2.warpPerspective(l, H, dsize=size)

# project the correspondence points as well
uv_left = to_homogenous_coord(uv_left)
uv_left_proj = H @ uv_left.T
uv_left_proj = to_heterogenous_coord(uv_left_proj.T)
```

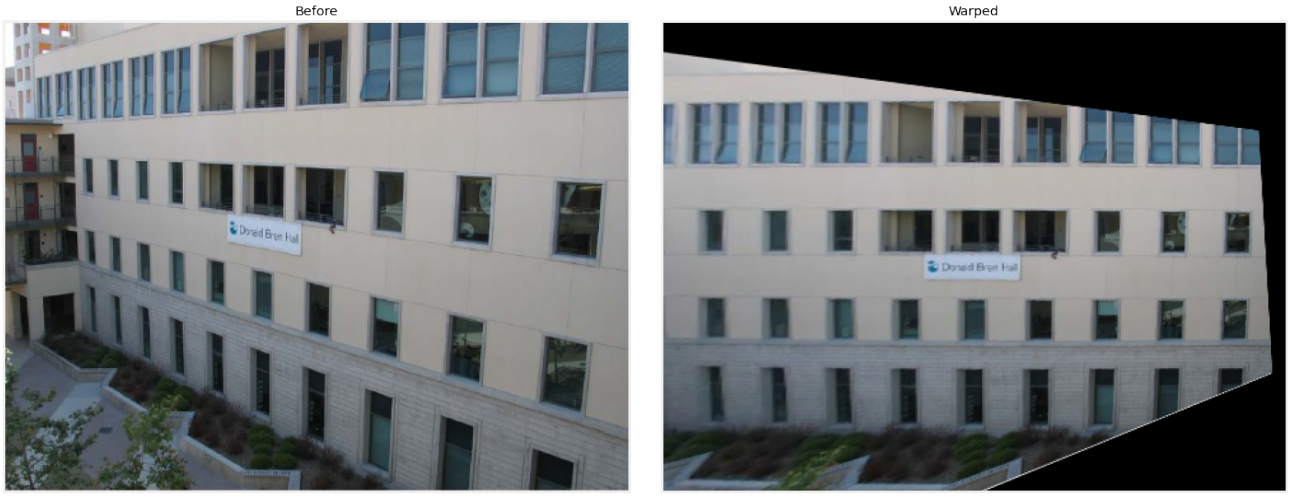
Selected correspondence points

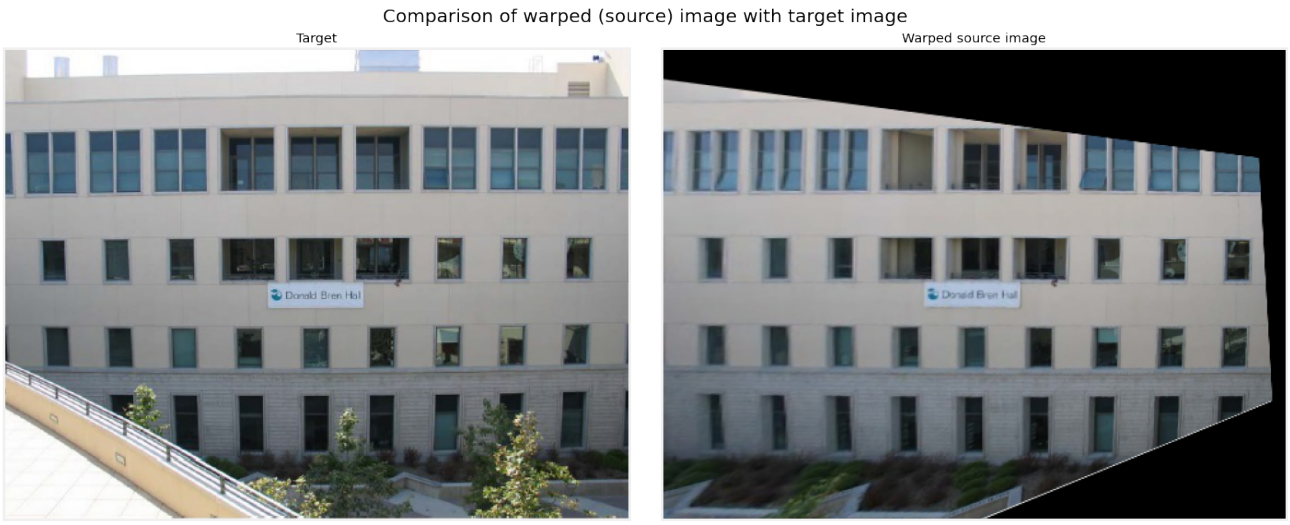

## 2.2 Compute homography matrix H

$$H = \begin{bmatrix} -0.0149 & 0.0004 & 0.9997 \\ -0.0024 & -0.0065 & 0.0169 \\ 0 & 0 & -0.0047 \end{bmatrix} \tag{11}$$

## 2.3 Warp image using H

Result of warping using the computed Homography matrix







```
# euclidean distance between target and warped correspondence points in pixels
dist = np.sqrt(np.sum(np.square(uv_right - uv_left_proj), 1))
dist
>> array([0.48353272, 0.59920041, 0.27162193, 1.05412312, 0.54916902,
       0.2332485 ])
```

Mean Squared Error (pixels): 0.3556

The small distance between the target and warped points are result of manual input of correspondence points using `plt.ginput()`.