CIS 580: Machine Perception **Spring 2020**

Problem Set 5: PDF Submission

Name: *Jun Wang*

Academic Honesty Declaration:

By submitting this document, I certify that all solutions below are *my own work*. I have listed below all individuals (other than the course instructor/TAs) with whom I have discussed any of the problems and/or solutions; nevertheless, while I may have discussed some of the problems/solutions with these collaborators, I have written all solutions and code below *independently, on my own.*

Problem 1:

In problem 1, we are required to find the world coordinates based on the image given in the problem set.

The coordinates are:

$$a = \begin{pmatrix} -s/2 \\ -s/2 \\ 0 \end{pmatrix} \tag{1}$$

$$b = \begin{pmatrix} s/2 \\ -s/2 \\ 0 \end{pmatrix} \tag{2}$$

$$c = \begin{pmatrix} s/2 \\ s/2 \\ 0 \end{pmatrix} \tag{3}$$

$$d = \begin{pmatrix} -s/2 \\ s/2 \\ 0 \end{pmatrix} \tag{4}$$

```
Pw = np.zeros([4, 3])
Pw[0] = [-s/2, -s/2, 0] #a
Pw[1] = [s/2, -s/2, 0] #b
Pw[2] = [s/2, s/2, 0] #c
Pw[3] = [-s/2, s/2, 0] #d
```

Problem 2:

In PnP.py, we are required to recover the rotation matrix R and translation matrix T using PnP.

First here we have all the points in the world lie in the ground plane $Z = 0$.

```
world_c = np.zeros([4, 2])
world_c = Pw[:, :2]
```

So the transformation looks like:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \sim K \begin{pmatrix} r_1 & r_2 & T \end{pmatrix} \begin{pmatrix} X \\ Y \\ W \end{pmatrix} \tag{5}$$

Here H we define as:

$$H \sim K \begin{pmatrix} r_1 & r_2 & T \end{pmatrix} \tag{6}$$

```
H = est_homography(world_c, Pc)
```

Then name the columns of $K^{-1}H$:

$$K^{-1}H = \begin{pmatrix} h'_1 & h'_2 & h'_3 \end{pmatrix} \tag{7}$$

```
KInv = np.linalg.inv(K)

HP = KInv @ H
h1p = HP[:,0]
h2p = HP[:,1]
hcp = np.cross(h1p,h2p)

Tmp = np.zeros([3,3])
Tmp[:,0] = h1p
Tmp[:,1] = h2p
Tmp[:,2] = hcp
```

The optimization problem now becomes:

$$\underset{R \in SO(3)}{\arg\min} \quad \|R - (h'_1 \; h'_2 \; h'_1 \times h'_2)\|_F^2 \tag{8}$$

```
U, S, VT = np.linalg.svd(Tmp)
```

Then

$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(UV^T) \end{pmatrix} V^T \tag{9}$$

```
det_value = np.linalg.det(U @ VT)
ide = np.eye(3)
ide[2,2] = det_value
```

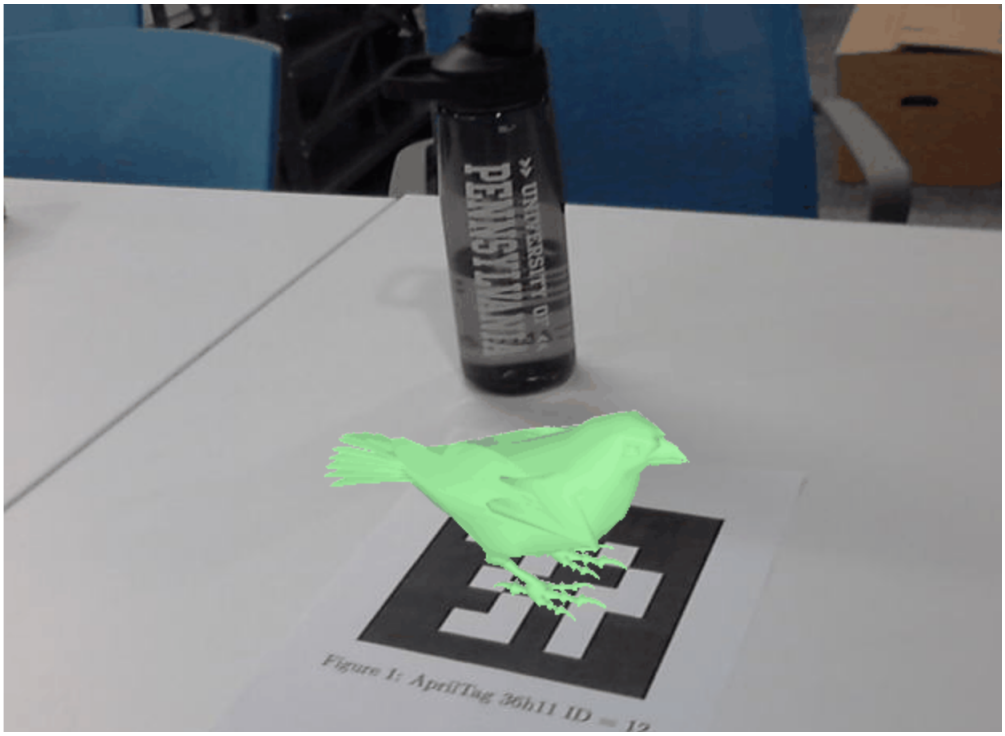
$$T = \frac{h_3'}{\|h_1'\|} \tag{10}$$

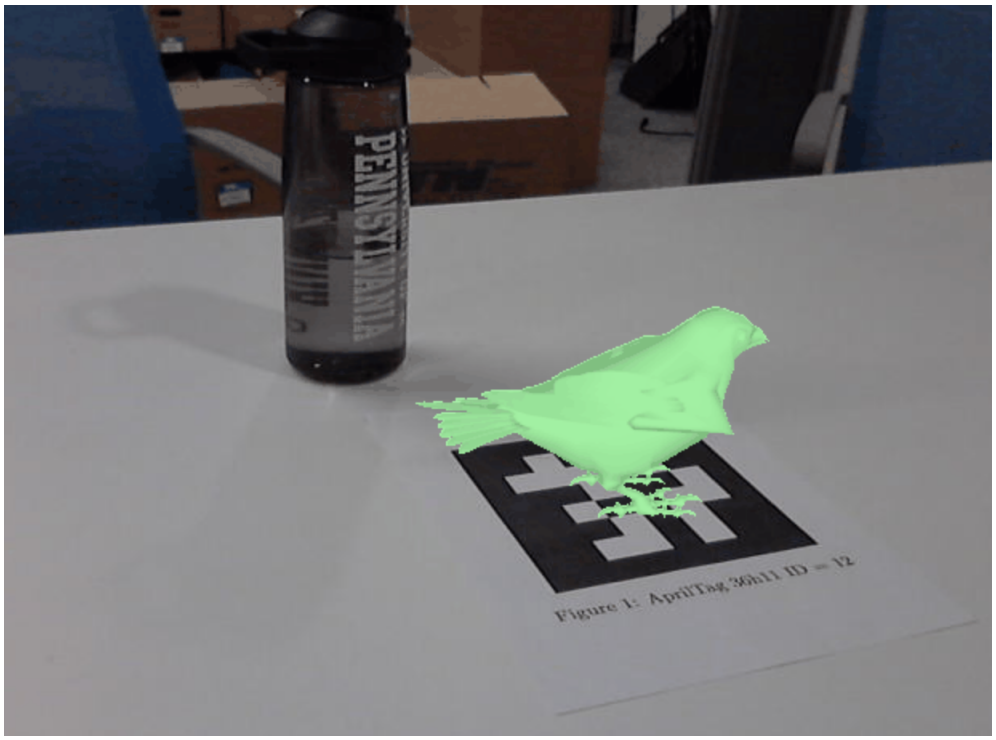
Also we need to change the transformation into " from camera to world ", so we have:

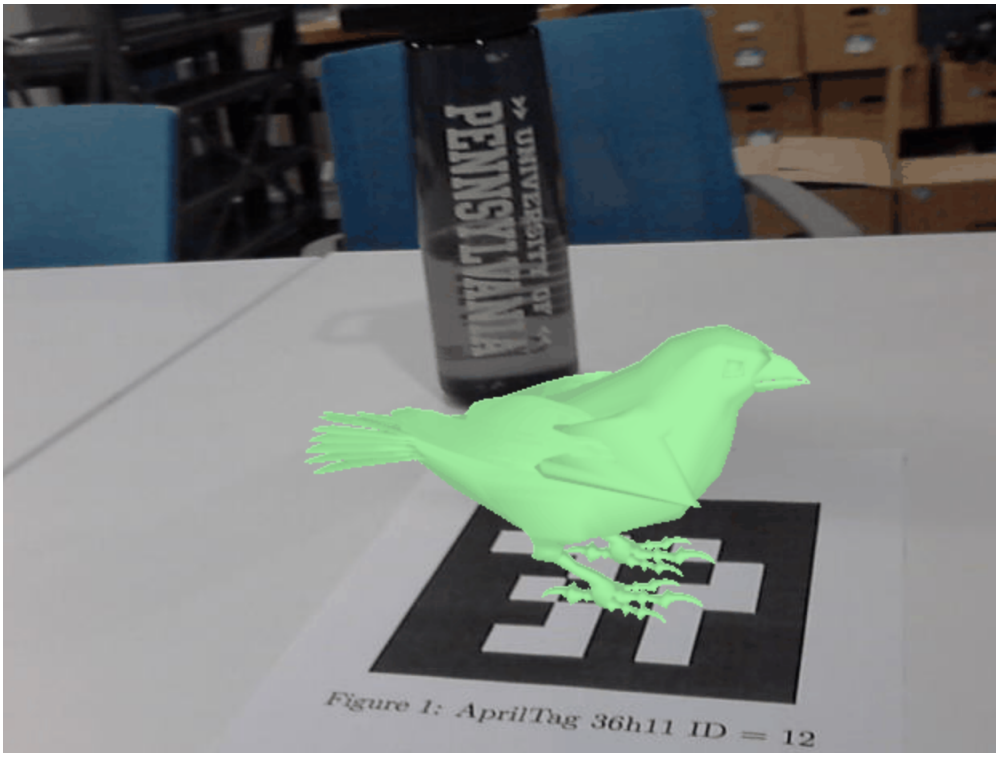
$$\begin{aligned} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} &= \lambda R^T K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} - R^T T \\ &= R^T \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} - R^T T \end{aligned} \tag{11}$$

```
R = U @ ide @ VT
h1_norm = np.linalg.norm(h1p)
t = HP[:,2] / h1_norm
R = R.transpose()
t = -1 * R @ t
```

Result




PENNSYLVANIA
Figure 1: AprilTag 36h11 ID = 12


PENNSYLVANIA
Figure 1: AprilTag 36h11 ID = 12

Problem 3:

In this problem, we are required to solve the perspective 3-point problem.

First get the side lengths of the triangle:

$$a = \|p_2 - p_3\| \tag{12}$$

$$b = \|p_1 - p_3\| \tag{13}$$

$$c = \|p_1 - p_2\| \tag{14}$$

```
Pw_T = Pw.transpose()
a = np.linalg.norm(Pw_T[:,1]-Pw_T[:,2])
b = np.linalg.norm(Pw_T[:,0]-Pw_T[:,2])
c = np.linalg.norm(Pw_T[:,0]-Pw_T[:,1])
```

Then get the j_1, j_2 and j_3.

Here notice that the pixel coordinates need to be calibrated with (u_0, v_0), and the focal length in intrinsic matrix K needs to find the average value of it.

```
f = (K[0][0] + K[1][1])/2
for k in range(4):
    Pc[k][0] = Pc[k][0] - K[0][2]
    Pc[k][1] = Pc[k][1] - K[1][2]
```

$$j_1 = \frac{1}{\sqrt{u_1^2 + v_1^2 + f^2}} \begin{pmatrix} u_1 \\ v_1 \\ f \end{pmatrix} \tag{15}$$

$$j_2 = \frac{1}{\sqrt{u_2^2 + v_2^2 + f^2}} \begin{pmatrix} u_2 \\ v_2 \\ f \end{pmatrix} \tag{16}$$

$$j_3 = \frac{1}{\sqrt{u_3^2 + v_3^2 + f^2}} \begin{pmatrix} u_3 \\ v_3 \\ f \end{pmatrix} \tag{17}$$

```
j = np.zeros([3,3])
sqrt_s = np.zeros([3,1])
for k in range(0,3):
    j[0,k] = Pc[k][0]
    j[1,k] = Pc[k][1]
    j[2,k] = f
    sqrt_s[k] = 1/((Pc[k][0]**2 + Pc[k][1]**2 + f**2) ** 0.5)

j1 = sqrt_s[0] * j[:,0]
j2 = sqrt_s[1] * j[:,1]
j3 = sqrt_s[2] * j[:,2]
```

Then we have angles:

$$\cos\alpha = j_2 \cdot j_3 \tag{18}$$

$$\cos\beta = j_1 \cdot j_3 \tag{19}$$

$$\cos\gamma = j_1 \cdot j_2 \tag{20}$$

```
c_alpha = np.dot(j2,j3)
c_beta = np.dot(j1,j3)
c_gamma = np.dot(j1,j2)
```

Using law of cosines:

$$s_2^2 + s_3^2 - 2s_2 s_3 \cos\alpha = a^2 \tag{21}$$

$$s_1^2 + s_3^2 - 2s_1 s_3 \cos\beta = b^2 \tag{22}$$

$$s_1^2 + s_2^2 - 2s_1 s_2 \cos\gamma = c^2 \tag{23}$$

Let

$$s_2 = us_1 \tag{24}$$

$$s_3 = vs_1 \tag{25}$$

Then we formulate a fourth order polynomial of v:

$$A_4 v^4 + A_3 v^3 + A_2 v^2 + A_1 v + A_0 = 0 \tag{26}$$

Where:

$$A_4 = (\frac{a^2 - c^2}{b^2} - 1)^2 - \frac{4c^2}{b^2}\cos^2\alpha \tag{27}$$

```
A4 = ((a**2 - c**2)/(b**2) - 1)**2 - (4 * c**2)/(b**2) * (c_alpha**2)
```

$$A_3 = 4(\frac{a^2 - c^2}{b^2}(1 - \frac{a^2 - c^2}{b^2})\cos\beta - (1 - \frac{a^2 + c^2}{b^2})\cos\alpha\cos\gamma + 2\frac{c^2}{b^2}\cos^2\alpha\cos\beta) \tag{28}$$

```
A3 = 4 * ( (a**2 - c**2)/(b**2) * (1 - (a**2 - c**2)/(b**2)) * c_beta - (1 - (a**2 + ...
    c**2)/(b**2)) * c_alpha * c_gamma + 2 * (c**2)/(b**2) * c_alpha**2 * c_beta )
```

$$A_2 = 2((\frac{a^2-c^2}{b^2})^2 - 1 + 2(\frac{a^2-c^2}{b^2})^2 \cos^2\beta + 2(\frac{b^2-c^2}{b^2})\cos^2\alpha - 4(\frac{a^2+c^2}{b^2})\cos\alpha\cos\beta\cos\gamma + 2(\frac{b^2-a^2}{b^2})\cos^2\gamma) \tag{29}$$

```
A2 = 2 * ( ((a**2 - c**2)/(b**2))**2 - 1 + 2 * ((a**2 - c**2)/(b**2))**2 * c_beta**2 + 2 * ...
    (b**2 - c**2)/(b**2) * c_alpha**2 - 4 * ((a**2 + c**2)/(b**2)) * c_alpha * c_beta * ...
    c_gamma + 2 * ((b**2 - a**2)/(b**2)) * c_gamma**2 )
```

$$A_1 = 4(-(\frac{a^2-c^2}{b^2})(1 + \frac{a^2-c^2}{b^2})\cos\beta + 2\frac{a^2}{b^2}\cos^2\gamma\cos\beta - (1 - (\frac{a^2+c^2}{b^2}))\cos\alpha\cos\gamma) \tag{30}$$

```
A1 = 4 * ( -1 * ((a**2 - c**2)/(b**2)) * (1 + (a**2 - c**2)/(b**2)) * c_beta + 2 * a**2 / ...
    b**2 * c_gamma**2 * c_beta - (1 - ((a**2 + c**2)/b**2)) * c_alpha * c_gamma)
```

$$A_0 = (1 + \frac{a^2-c^2}{b^2})^2 - 4\frac{a^2}{b^2}\cos^2\gamma \tag{31}$$

```
A0 = (1 + (a**2 - c**2) / b**2)**2 - 4 * a**2 / b**2 * c_gamma**2
```

Here we may get 4 roots of v, we need to choose the first root that satisfy the condition that the root itself is real number and the distance s_1, s_2 and s_3 are all positive.

Then we have equation of u as:

$$u = \frac{(-1 + \frac{a^2-c^2}{b^2})v^2 - 2(\frac{a^2-c^2}{b^2})\cos\beta v + 1 + \frac{a^2-c^2}{b^2}}{2(\cos\gamma - v\cos\alpha)} \tag{32}$$

Using u and v, we can have value of s_1

$$s_1 = \sqrt{\frac{a^2}{u^2 + v^2 - 2uv\cos\alpha}} \tag{33}$$

```
coef = [A4, A3, A2, A1, A0]
v_tmp = np.roots(coef)
for k in range(len(v_tmp)):
    if np.imag(v_tmp[k]) == 0:
        v = np.real(v_tmp[k])
        u = ( (-1 + (a**2 - c**2)/b**2) * (v**2) - 2 * ((a**2 - c**2) / b**2) * c_beta * v + ...
            1 + ((a**2 - c**2) / b**2) ) / ( 2 * (c_gamma - v * c_alpha) )

        s1 = (a**2 / (u**2 + v**2 - 2 * u * v * c_alpha)) ** 0.5
        s2 = u * s1
        s3 = v * s1
        if s1 > 0:
            if s2 > 0:
                if s3 > 0:
                    vres = v
                    ures = u
                    s1res = s1
```

```
                s2res = s2
                s3res = s3
                break
```

With the result of s_1, s_2 and s_3, we know that the coordinate of three point in the camera coordinate system:

$$p_1 = s_1 j_1 \tag{34}$$

$$p_2 = s_2 j_2 \tag{35}$$

$$p_3 = s_3 j_3 \tag{36}$$

```
p1 = np.zeros([3,1])
p2 = np.zeros([3,1])
p3 = np.zeros([3,1])

p1 = s1res * j1
p2 = s2res * j2
p3 = s3res * j3

camera_c = np.zeros([3,3])
camera_c[:,0] = p1
camera_c[:,1] = p2
camera_c[:,2] = p3
```

Then according to the procrustes method, we first give out a minimization problem:

$$\min_{R,T} \sum_{i}^{N} \|A_i - RB_i - T\|^2 \tag{37}$$

Where X is the camera coordinate system and Y is the world coordinate system. They need to be calibrated with the centroid of the coordinates. So we have $A = Y$ and $B = X$.

$$A = \begin{pmatrix} A_1 - \bar{A} & \dots & A_N - \bar{A} \end{pmatrix} \tag{38}$$

$$B = \begin{pmatrix} B_1 - \bar{B} & \dots & B_N - \bar{B} \end{pmatrix} \tag{39}$$

```
X = X.transpose()
Y = Y.transpose()
A = Y
B = X

A_center = np.zeros([3,1])
B_center = np.zeros([3,1])

for k in range(3):
```

```
        A_center[k] = (A[k][0] + A[k][1] + A[k][2])
        B_center[k] = (B[k][0] + B[k][1] + B[k][2])

    A_center = A_center / 3
    B_center = B_center / 3

    A_calc = np.zeros([3,3])
    B_calc = np.zeros([3,3])

    for k in range(3):
        A_calc[k] = A[k] - A_center[k]
        B_calc[k] = B[k] - B_center[k]
```

Then we have:

$$\min_{R,T} \sum_{i}^{N} \|A_i - RB_i - T\|^2 \tag{40}$$

```
    tmp = B_calc @ A_calc.transpose()
    U, S, VT = np.linalg.svd(tmp)
```

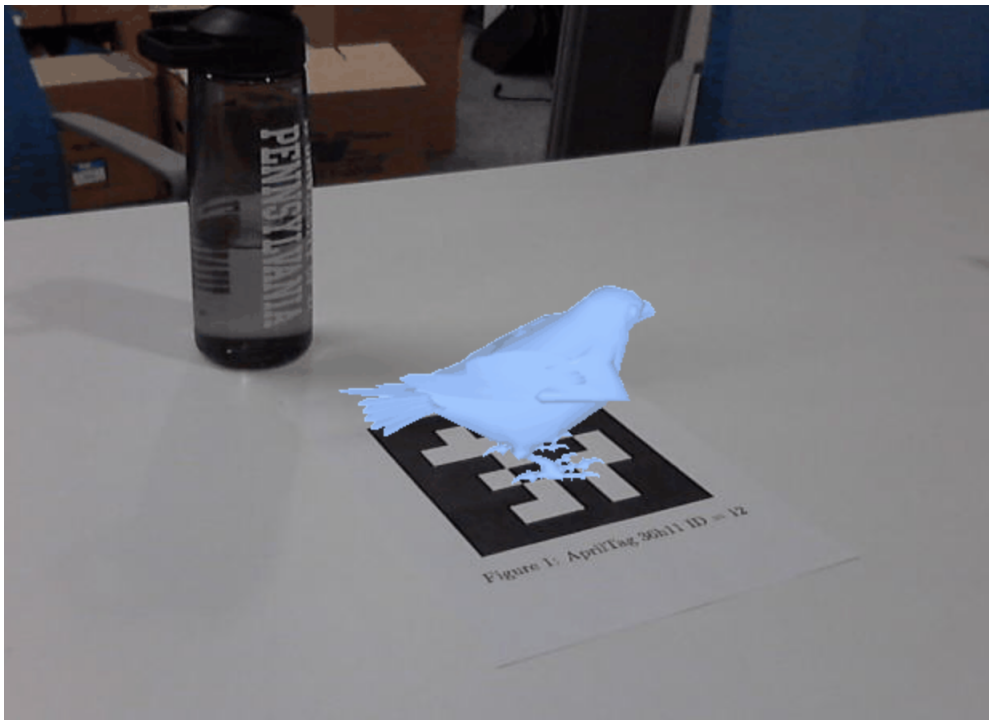
So according to the SVD of XY^T, we have:

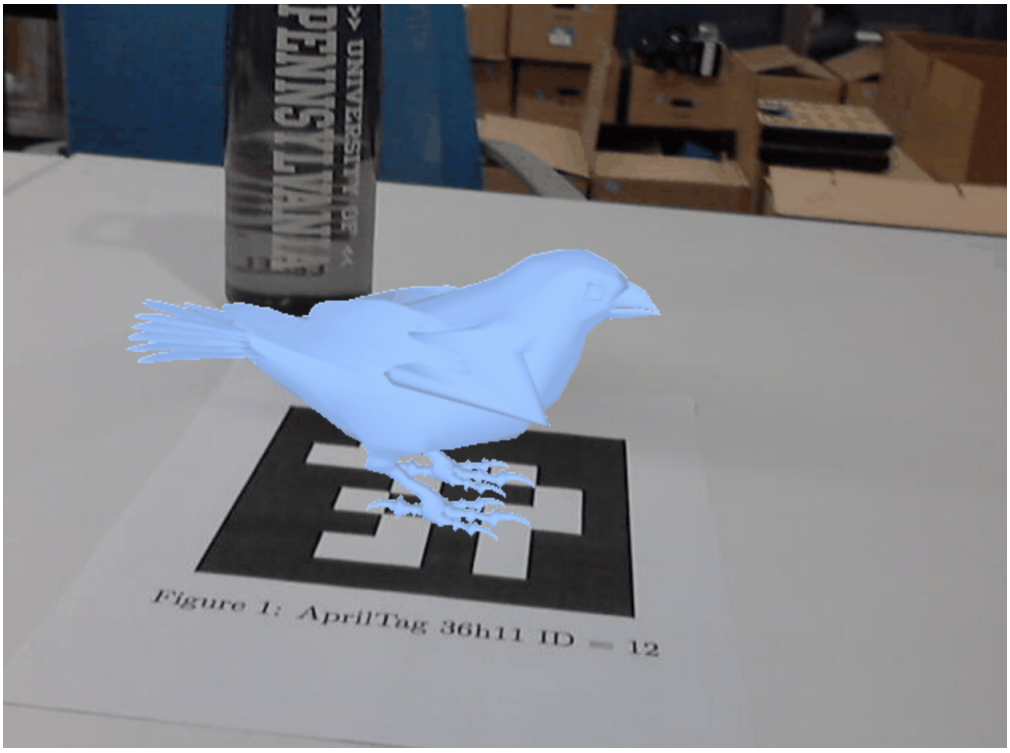
$$R = V \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(VU^T) \end{pmatrix} U^T \tag{41}$$

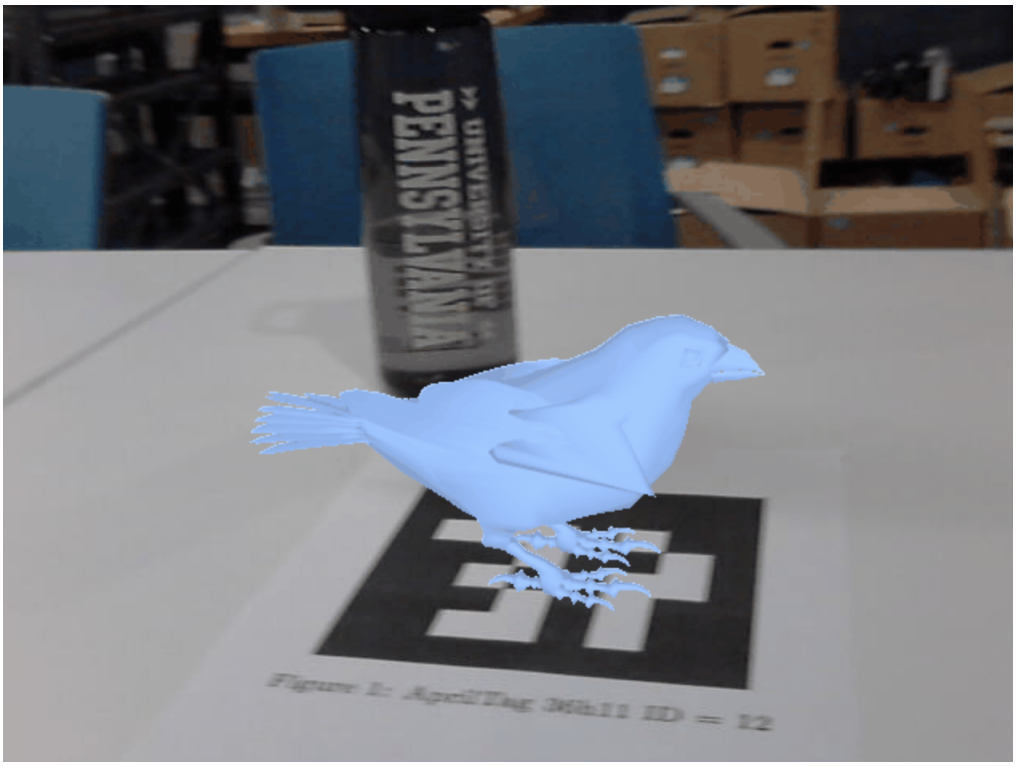
$$T = \bar{A} - R\bar{B} \tag{42}$$

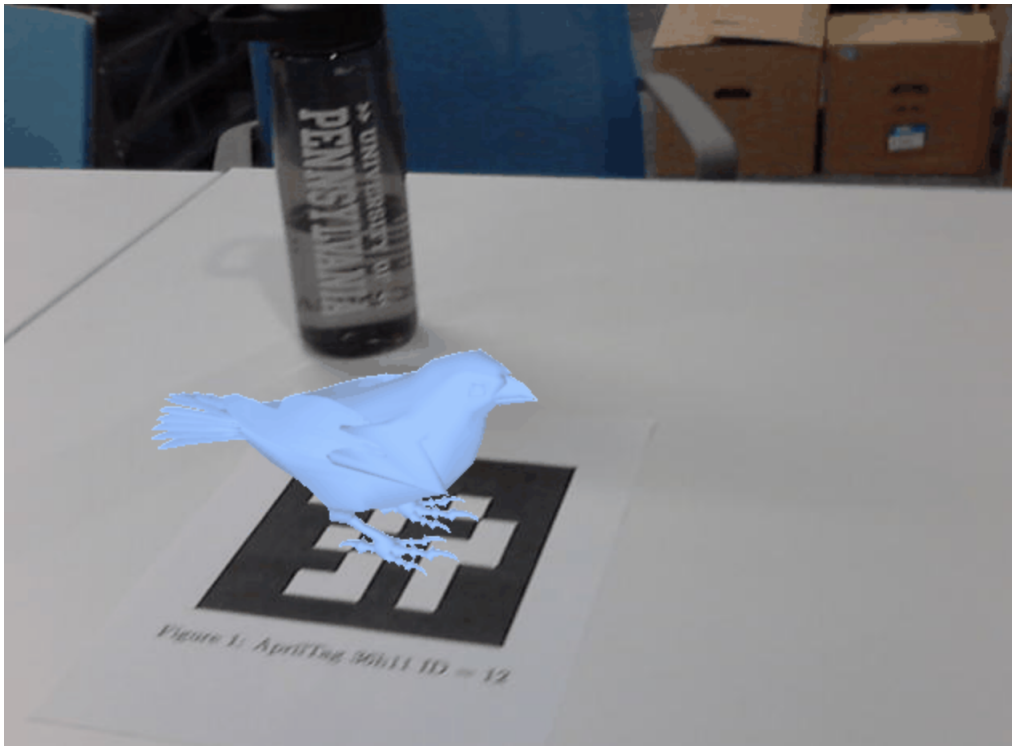
```
    ide = np.eye(3)
    ide[2][2] = np.linalg.det(VT.transpose() @ U.transpose())
    R = VT.transpose() @ ide @ U.transpose()
    t = A_center - R @ B_center
```

Result


PENNSYLVANIA
Figure 1: AprilTag 36h11 ID = 12


PENNSYLVANIA
Figure 1: AprilTag 36h11 ID = 12


PENNSTATE
Figure 1: AprilTag 36h11 ID = 12


PENNSTATE
Figure 1: AprilTag 36h11 ID = 12