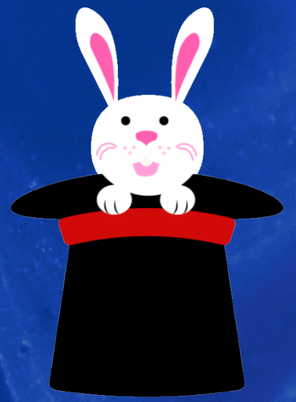
Rigid Motion Transformation Using SVD



Ofir Herrera
Shaked weis

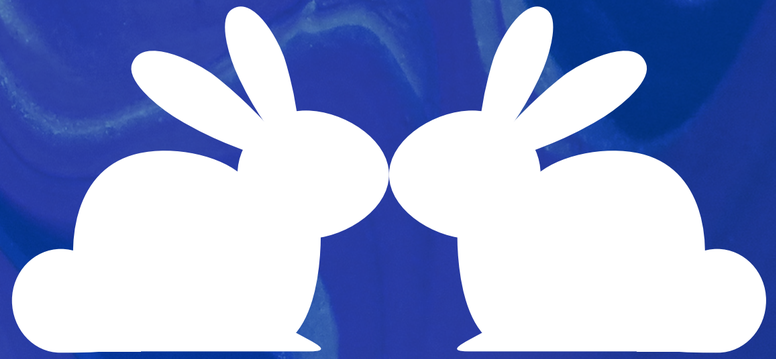
Least-Squares Rigid Motion Using SVD
Olga Sorkine-Hornung and Michael Rabinovich
January 16, 2017

Types of Rigid Motion

Translation
(slide)

Rotation
(turn)

Reflection
(mirror)



Problem:

Let P={p1,p2,...,pn} and Q={q1,q2,...,qn} be two sets of corresponding points in R^d. In our case, d=2.

We wish to find a rigid transformation that optimally aligns the two sets in the least squares

sense, i.e., we seek a rotation R and a translation vector t such that.

In order to solve this problem first, we would like to define P, R, T and find Q.
After we will find Q, we would like to solve the problem and find R1, t and compare them to R,T.

We have defined a function "calculate_Q_matrix".

Input: matrix P, angle, transletion.
Outpot: returns matrix Q.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

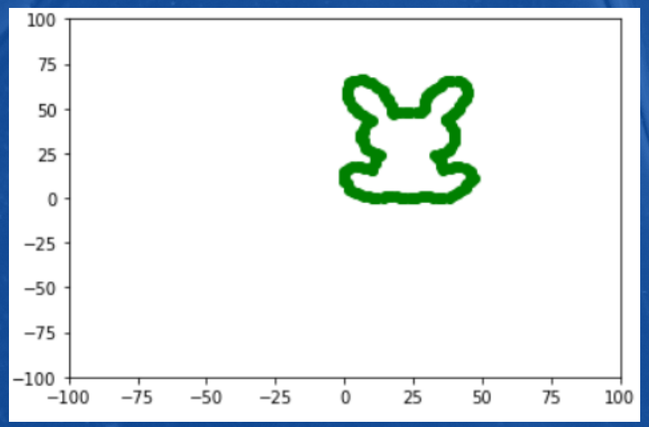
$$\theta = -\frac{\pi}{6}$$

```
Rotation matrix:
[[0.866025403784439 0.500000000000000]
 [-0.500000000000000 0.866025403784439]]
```

```
the Transletion: [-99  30]
```

P

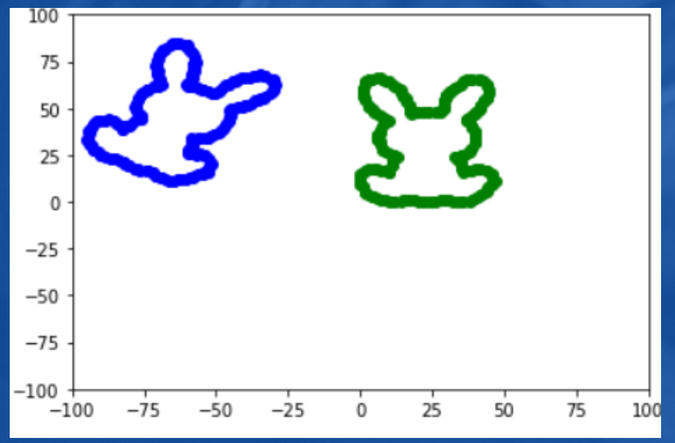
(200,2)



R*P+T=Q

Q

(200,2)



```python
def calculate_Q_matrix(P,theta1,T):
        # define angle and R
    scos=lambda theta: sp.N(sp.cos(theta))
    ssin=lambda theta: sp.N(sp.sin(theta))

    def angle(num):
        #  counterclockwise- positive angle
        if(num > 0):
            return(sp.pi/num)
        #  clockwise- negative angle
        return((2*sp.pi)+(sp.pi/num))

    theta = angle(theta1) # positive angle(counter-clockwise rotation), negative angle(clockwise)
    c = scos(theta)
    s=  ssin(theta)

    R = np.array(((c, -s), (s, c)))
    print('Rotation matrix:')
    print(R)

    # Data after the rotation
    Q = np.zeros((P.shape[0],P.shape[1]))

    for i in range(P.shape[0]):
        x=R.dot(P[i].reshape(2,1))+T.reshape(2,1)
        Q=np.append(Q,x.reshape(1,2),axis=0)

    # delete the first insertion dot (0,0)
    Q= np.delete(Q, np.s_[0:P.shape[0]], axis = 0)

    print("Shape of Q Matrix:", Q.shape)

    return Q

```

Now, we will solve the problem:
find R1,t and check whether they are approximately equal to R and T.

≈

We have defined a function "Rigid_Motion"
Input: P and Q matrices
Outpot: prints the R1 and T1 according to the algorithm represented.
Step 1: Compute the weighted centroids of both point sets.

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i}{\sum_{i=1}^{n} w_i}, \quad \bar{\mathbf{q}} = \frac{\sum_{i=1}^{n} w_i \mathbf{q}_i}{\sum_{i=1}^{n} w_i}$$

```python
def Rigid_Motion(P,Q):

    #Step 1: Compute the weighted centroids of both point sets
    centroid_P=np.mean(P,axis=0)
    centroid_Q=np.mean(Q,axis=0)
    centroid_P=centroid_P.reshape(2,1)
    centroid_Q=centroid_Q.reshape(2,1)

    centroid_P=centroid_P.astype(float)
    centroid_Q=centroid_Q.astype(float)
```

Step 2: Compute the centered vectors.

We can thus concentrate on computing the rotation R by restating the problem such that the translation would be zero.

$$\mathbf{x}_i := \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{y}_i := \mathbf{q}_i - \bar{\mathbf{q}}, \qquad i = 1, 2, \ldots, n.$$

```python
# extend the centroid to the same shape of matrix  P
centroid_P_Matrix=centroid_P.T
concat_P=centroid_P_Matrix[0].reshape(1,2)
print("The centroid of P Matrix: {} with shape:{}".format(centroid_P,centroid_P.shape))
print("The centroid of Q Matrix: {} with shape:{}".format(centroid_Q,centroid_Q.shape))

for i in range(P.shape[0]-1):
    centroid_P_Matrix=np.append(centroid_P_Matrix,concat_P,axis = 0)
centroid_P_Matrix=(np.array)(centroid_P_Matrix)
centroid_P_Matrix=centroid_P_Matrix.astype(float)
print("Shape centroid_P_Matrix:\n",centroid_P_Matrix.shape)

# extend the centroid to the same shape of matrix  Q
centroid_Q_Matrix=centroid_Q.T
concat_Q=centroid_Q_Matrix[0].reshape(1,2)
for i in range(Q.shape[0]-1):
    centroid_Q_Matrix=np.append(centroid_Q_Matrix,concat_Q,axis = 0)
centroid_Q_Matrix=(np.array)(centroid_Q_Matrix)
centroid_Q_Matrix=centroid_Q_Matrix.astype(float)
print("Shape centroid_Q_Matrix:\n",centroid_Q_Matrix.shape)

#calculate the centerd
centered_P= P-centroid_P_Matrix
centered_Q= Q-centroid_Q_Matrix

centered_P=centered_P.astype(float)
centered_Q=centered_Q.astype(float)
```

Step 3: Compute the 2 × 2 "covariance" matrix

The optimal rotation R1 would be:

$$R = \underset{R \in SO(d)}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \left\| R\mathbf{x}_i - \mathbf{y}_i \right\|^2$$

After simplifying the expression and excluding the expressions which do not depend on R at all we get:

$$\underset{R \in SO(d)}{\operatorname{argmin}} \left(-2 \sum_{i=1}^{n} w_i \mathbf{y}_i^{\mathsf{T}} R\mathbf{x}_i \right) = \underset{R \in SO(d)}{\operatorname{argmax}} \sum_{i=1}^{n} w_i \mathbf{y}_i^{\mathsf{T}} R\mathbf{x}_i.$$

We note that:

$$\sum_{i=1}^{n} w_i \mathbf{y}_i^{\mathsf{T}} R\mathbf{x}_i = \operatorname{tr}\left(WY^{\mathsf{T}} RX \right)$$

Therefore we are looking for a rotation R that maximizes $\operatorname{tr}\left(WY^{\mathsf{T}}RX\right)$

Matrix trace has the property: $\operatorname{tr}(AB) = \operatorname{tr}(BA)$

for any matrices A, B of compatible dimension.

Therefor: $\operatorname{tr}\left(WY^{\mathsf{T}}RX\right) = \operatorname{tr}\left((WY^{\mathsf{T}})(RX)\right) = \operatorname{tr}\left(RXWY^{\mathsf{T}}\right)$

Define matrix $S = XWY^{\mathsf{T}}$

Now, we will compute S:

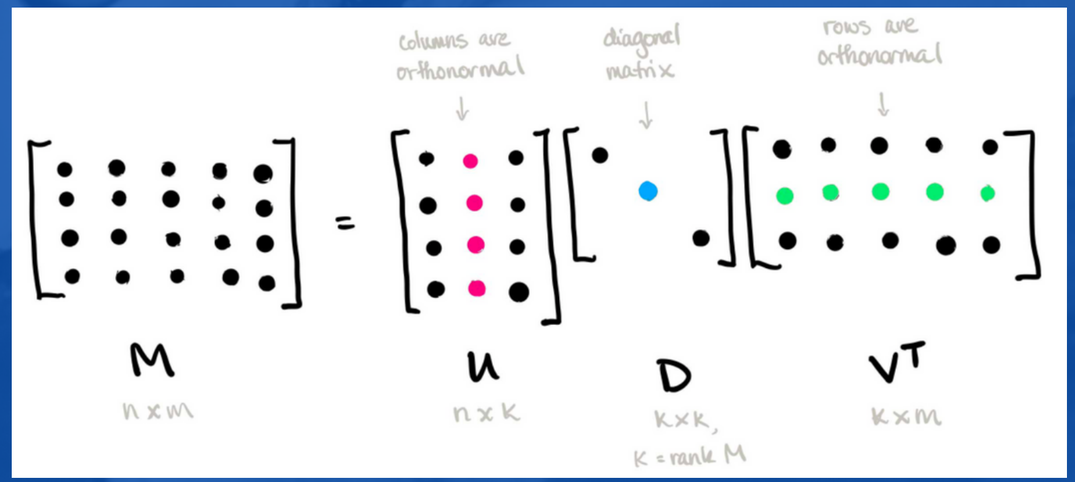
```python
#Step 3: Compute the d × d covariance matrix
W= np.eye(P.shape[0],P.shape[0])

print("centered_P: ",centered_P.shape)
print("centered_Q: ", centered_Q.T.shape)
S= np.dot(np.dot(centered_P.transpose(), W), centered_Q)
S=S.astype(float)

print("S Matrix:" ,S)
print("S shape:" ,S.shape)
```

```
S Matrix: [[   637.465  48147.555]
 [-97864.795   -637.465]]
S shape: (2, 2)
```

SVD

The singular value decomposition (SVD) is one of the most useful matrix decompositions, particularly for numerical computations.
Its most common application is in the solution of over-determined systems of equations.

U	Sigma	V
Orthonormal matrix	Diagonal matrix	Orthonormal matrix



In favor of computing the SVD of matrix S we can get:

$$\operatorname{tr}\left(RXWY^{\mathsf{T}}\right) = \operatorname{tr}(RS) = \operatorname{tr}\left(RU\Sigma V^{\mathsf{T}}\right) = \operatorname{tr}\left(\Sigma V^{\mathsf{T}} RU\right).$$

$$M = V^{\mathsf{T}} RU$$

Ortogonal matrix which=>

$$1 = \mathbf{m}_j^{\mathsf{T}}\mathbf{m}_j = \sum_{i=1}^{d} m_{ij}^2 \;\Rightarrow\; m_{ij}^2 \leq 1 \;\Rightarrow\; |m_{ij}| \leq 1$$

mj= column of M

$$I = M = V^{\mathsf{T}} RU \;\Rightarrow\; V = RU \;\Rightarrow\; R = VU^{\mathsf{T}}.$$

Step 4: Compute the singular value decomposition S
The rotation we are looking for is then:

```
U, sigma, V= svd(S)
M= np.eye(P.shape[1],P.shape[1])
R1= ((V).dot(M)).dot(U.T)
```

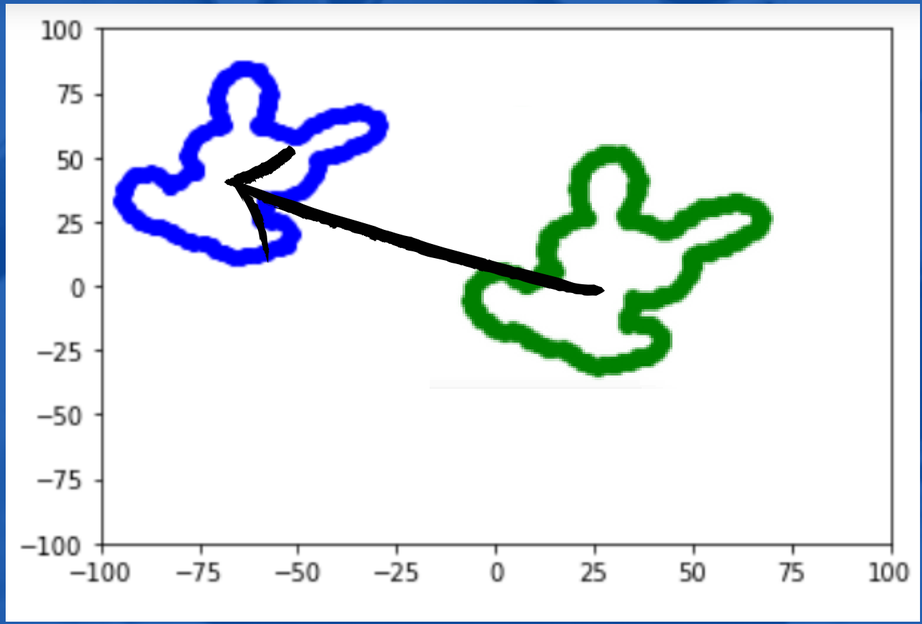
```
The Rotation Matrix according to the algorithm:
[[0.866 0.500]
 [-0.500 0.866]]
```

```
Rotation matrix:
[[0.866025403784439 0.500000000000000]
 [-0.500000000000000 0.866025403784439]]
```

Step 5: After we found the rotation matrix, R1, we would like to Compute the optimal translation:

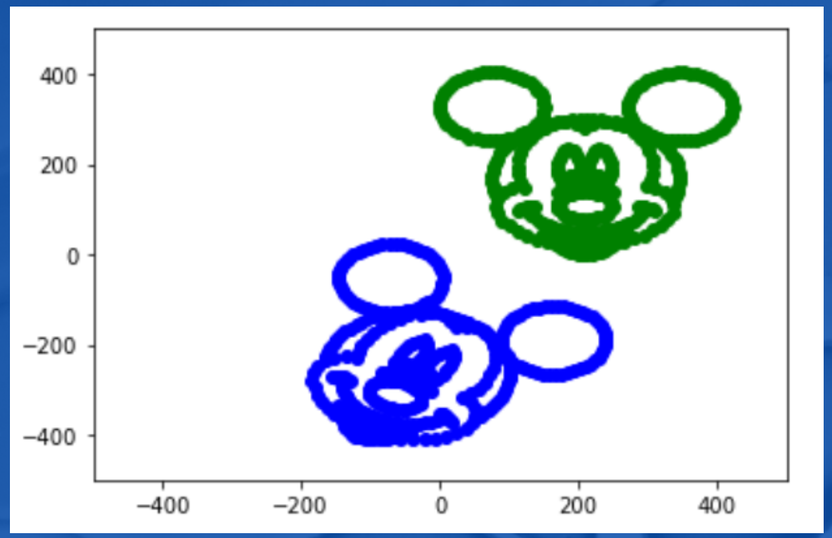
```
#5. Compute the optimal translation
t=centroid_Q-(R1.dot(centroid_P))
```

```
The transletion according to the algorithm:
[[-99.000]
 [30.000]]
```




```
the Transletion: [-99  30]
```

Mickey Mouse example



θ=45

```
Rotation matrix:
[[0.707106781186548 0.707106781186548]
 [-0.707106781186548 0.707106781186548]]
Shape of Q Matrix: (392, 2)
the Transletion: [-300 -300]
```

```
The Rotation Matrix according to the algorithm:
[[0.707 0.707]
 [-0.707 0.707]]
The transletion according to the algorithm:
[[-8.000]
 [-300.000]]
```

SOURCES

Least-Squares Rigid Motion Using SVD
Olga Sorkine-Hornung and Michael Rabinovich
Department of Computer Science, ETH Zurich January 16, 2017

Multiple View Geometry in Computer Vision Second Edition
Richard Hartley Australian National University, Canberra, Australia
Andrew Zisserman University of Oxford, UK