ipvive challenge

Yuqiu Yang
yyqengels@gmail.com

Math

Euclidean geometry

A

Let two points $P = (x_1, y_1, z_1, w_1)$ and $Q = (x_2, y_2, z_2, w_2)$ with $w_1 = w_2 = 1$ be given in homogeneous coordinates, then one commonly used formula to find the distance between P and Q is $d(P, Q) = \sqrt{(P-Q) \bullet (P-Q)} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ since $w_1 = w_2$.

B

For simplicity, we will represent P and Q by column vectors, namely, $P = \begin{bmatrix} x_1, y_1, z_1, 1 \end{bmatrix}^T$ and $Q = \begin{bmatrix} x_2, y_2, z_2, 1 \end{bmatrix}^T$.
It is well known that the isometry group of a 3-dimensional Euclidean space is $E(3)$ which has as subgroups the translational group $T(3)$ and the orthogonal group $O(3)$. Since the composition of two isometries is still an isometry, we only need to show that the distance is invariant under the same translation or under the same orthogonal transformation.

"Translation"

Let A an arbitrary translation be given. Then A can be written in a matrix form: $\begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Let $P' = AP$ and $Q' = AQ$, then $d(P', Q') = \sqrt{(P' - Q')^T (P' - Q')}$

$= \sqrt{\left(\begin{bmatrix} x_1 + a_1 \\ y_1 + a_2 \\ z_1 + a_3 \\ 1 \end{bmatrix} - \begin{bmatrix} x_2 + a_1 \\ y_2 + a_2 \\ z_2 + a_3 \\ 1 \end{bmatrix} \right)^T \left(\begin{bmatrix} x_1 + a_1 \\ y_1 + a_2 \\ z_1 + a_3 \\ 1 \end{bmatrix} - \begin{bmatrix} x_2 + a_1 \\ y_2 + a_2 \\ z_2 + a_3 \\ 1 \end{bmatrix} \right)}$

$= \sqrt{\left(\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} - \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} \right)^T \left(\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} - \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} \right)}$

$= d(P, Q)$

"Orthogonal transformation"

Let B be an arbitrary 3×3 orthogonal matrix, and let $B' = \begin{bmatrix} B & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$, where $\mathbf{0}$ is a 3×1 zero matrix, then applying the orthogonal transformation, B, to P and Q is equivalent to premultiplying P and Q by B'. Let $P' = B'P$ and $Q' = B'Q$, then $d(P', Q') = \sqrt{(P' - Q')^T (P' - Q')} = \sqrt{(B'P - B'Q)^T (B'P - B'Q)} = \sqrt{(P - Q)^T B'^T B' (P - Q)}$
$= \sqrt{(P - Q)^T (P - Q)} = d(P, Q)$. The fourth equality follows from the orthogonality of B and the multiplication rule of block matrices.

Spherical geometry

A

Let two points $P = (x_1, y_1, z_1, w_1)$ and $Q = (x_2, y_2, z_2, w_2)$ with $x_1^2+y_1^2+z_1^2+w_1^2 = x_2^2+y_2^2+z_2^2+w_2^2 = 1$ be given, one commonly used formula to find the distance between P and Q is $d(P,Q) = \arccos(\frac{P \bullet Q}{\|P\| \, \|Q\|}) = \arccos(P \bullet Q)$, where "$\bullet$" is the dot product and "$\|\cdot\|$" is the norm induced by the dot product.

B

For simplicity, we will represent P and Q by column vectors, namely, $P = \begin{bmatrix} x_1, y_1, z_1, w_1 \end{bmatrix}^T$ and $Q = \begin{bmatrix} x_2, y_2, z_2, w_2 \end{bmatrix}^T$. Denote by $\mathbb{S}$ the unit 3-sphere $\{(x,y,z,w) \in \mathbb{R}^4 : x^2+y^2+z^2+w^2 = 1\}$ and let U be an arbitrary 4×4 orthogonal matrix.
First, we show that if $P \in \mathbb{S}$, then $P' = UP \in \mathbb{S}$.

Proof. To see this, we only need to show that $\|P'\| = 1$. Since $\|P'\| = \sqrt{(P')^T P'} = \sqrt{(UP)^T UP} = \sqrt{P^T U^T UP} = \sqrt{P^T P} = \|P\| = 1$. The fourth equality holds since U is orthogonal. ∎

Let $P' = UP$ and $Q' = UQ$, then we show that $d(P,Q) = d(P',Q')$, where $d(\cdot,\cdot)$ is the distance given in part A.

Proof. First of all, $d(P',Q')$ now makes sense since we just showed that an orthogonal transformation maps $\mathbb{S}$ into $\mathbb{S}$.
Then $d(P',Q') = \arccos((P')^T Q') = \arccos((UP)^T UQ) = \arccos(P^T U^T UQ) = \arccos(P^T Q) = d(P,Q)$. ∎

Hyperbolic geometry

A

Let two points $P = (x_1, y_1, z_1, w_1)$ and $Q = (x_2, y_2, z_2, w_2)$ with $x_1^2+y_1^2+z_1^2-w_1^2 = x_2^2+y_2^2+z_2^2-w_2^2 = -1$ and $w_1, w_2 > 0$ be given, one commonly used formula to find the distance between P and Q is $d(P,Q) = \operatorname{arccosh}(-P * Q)$, where arccosh is the inverse hyperbolic cosine function and $*$ is the inner product for Minkowski space-time model. Specifically, $P * Q = x_1x_2 + y_1y_2 + z_1z_2 - w_1w_2$.

B

Let $\eta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$, then $d(P,Q) = \operatorname{arccosh}(-P^T \eta Q)$.
Let M be an element of the proper orthochronous Lorentz group $SO(3,1)$, then M satisfies (1). $M^T \eta M = \eta$; (2). $m_{4,4} > 0$, where $m_{4,4}$ is the element of M in row 4 and column 4; and (3). $\det(M) = 1$.
Denote by $\mathbb{L}$ the unit 3-hyperboloid $\{(x,y,z,w) \in \mathbb{R}^4 : x^2+y^2+z^2-w^2 = 1 \text{ and } w > 0\}$.

First, we show that if $P \in \mathbb{L}$, then $P' = MP \in \mathbb{L}$.

Proof. First of all, we show that $P' * P' = -1$.
$P' * P' = P'^T \eta P' = P^T M^T \eta M P = P^T \eta P = P * P = -1$ since $P \in \mathbb{L}$.
Secondly, we show that the last coordinate of P' is greater than 0.
Let $P' = \begin{bmatrix} x', y', z', w' \end{bmatrix}^T$, $P = \begin{bmatrix} x, y, z, w \end{bmatrix}^T$ and $M = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{bmatrix}$
We first partition M into 4 blocks: $M = \begin{bmatrix} m & c \\ b^T & a \end{bmatrix}$, where m is a 3×3 matrix, a is a scalar, and b and c are both 3×1 matrices. We can then partition η accordingly, namely, $\eta = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0}^T & -1 \end{bmatrix}$, where $\mathbf{1}$ is the

3×3 identity matrix and $\mathbf{0}$ is the 3×1 zero matrix.
Since $M^T \eta M = \eta$, we have $\eta^{-1} = M\eta^{-1}M^T$.
Since $\eta^{-1} = \eta$, we have $\begin{bmatrix} m & c \\ b^T & a \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0}^T & -1 \end{bmatrix} \begin{bmatrix} m^T & b \\ c^T & a \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0}^T & -1 \end{bmatrix}$, which gives us $b^T b = a^2 - 1$.
Since $w' = aw + b^T \left[x, y, z\right]^T$, it suffices to show that $aw + b^T \left[x, y, z\right]^T > 0$.
$-2\frac{b^T \left[x, y, z\right]^T}{aw} \leq \frac{b^T b}{a^2} + \frac{\left[x, y, z\right] \left[x, y, z\right]^T}{w^2} < \frac{a^2 - 1}{a^2} + 1 < 2$
$\Rightarrow aw + b^T \left[x, y, z\right]^T > 0$. The second last inequality holds since $x^2 + y^2 + z^2 + 1 = w^2$.

■

Now let $P' = MP$ and $Q' = MQ$, then we show that $d(P', Q') = d(MP, MQ) = d(P, Q)$.

Proof. $d(P', Q')$ now makes sense since we just showed that M maps $\mathbb{L}$ into $\mathbb{L}$.
$d(P', Q') = \text{arccosh}(-P' * Q') = \text{arccosh}(-P'^T \eta Q') = \text{arccosh}(-P^T M^T \eta M Q) = \text{arccosh}(-P^T \eta Q) = \text{arccosh}(-P * Q) = d(P, Q)$. ■

Coding

We first define some functions that will be used later on.
The function used to generate a random point.

```python
def genPoint(n, method = "Homogeneous"):
    # Generate random points in R^n, S^(n-1), or L^(n-1)
    import numpy as np
    P = np.random.randn(n)
    if method == "Hyperbolic":
        P[-1] = np.sqrt(1 + np.dot(P[0 : -1], P[0 : -1]))
        return P
    elif method == "Spherical":
        # Make sure that P is not the origin
        while all(P == 0):
            P = np.random.randn(n)
        P = P / np.sqrt(np.dot(P, P))
        return P
    else:
        # when method is Homogeneous
        # n is the length of the desired
        # vector - 1
        tempP = np.zeros(n + 1)
        tempP[0:-1] = P
        tempP[-1] = 1
        return tempP
```

The function used to find the distance between two points.

```python
def distance(P1, P2, method = "Homogeneous"):
    # Find the distance between two points
    import numpy as np
    import math
    P1 = np.asarray(P1)
    P2 = np.asarray(P2)
    if method == "Hyperbolic":
        eta = np.eye(len(P1))
        eta[-1,-1] = -1
        return math.acosh(abs(np.dot(P1,eta).dot(P2)))
    elif method == "Spherical":
        return math.acos(np.dot(P1,P2))
    else:
        return np.sqrt(np.dot(P1-P2,P1-P2))
```

The function used to generate a random element of O(n) or SO(n) (in the hyperbolic case)

```python
def orthoTrans(n, method = "Homogeneous"):
    # Generate elements in O(n) and SO(n) (hyperbolic)
    import numpy as np
    from scipy.stats import ortho_group
    from scipy.stats import special_ortho_group
    if method == "Spherical":
        Q = ortho_group.rvs(n)
```

```
        return Q
    elif method == "Hyperbolic":
        # generate pure rotation which
        # leaves the time coordinate unchanged
        Q = np.zeros((n, n))
        Q[:-1,:-1] = special_ortho_group.rvs(n - 1)
        Q[- 1, - 1] = 1
        return Q
    else:
        # when method is Homogeneous
        # n is the length of the vector - 1
        Q = np.zeros((n + 1, n + 1))
        Q[:-1,:-1] = ortho_group.rvs(n)
        Q[n, n] = 1
        return Q
```

The function used to generate a random translation.

```
def translation(n):
    import numpy as np
    U = np.eye(n + 1)
    U[0:-1, -1] = np.random.randn(n)
    return U
```

The function used to generate a Lorentz boost.

```
def boost(beta):
    # beta 1 by 3 ndarray whose norm is < 1
    import numpy as np
    Beta = beta.dot(beta.transpose())
    gamma = 1/np.sqrt(1-Beta)
    B = np.zeros((4,4))
    B[3,3] = gamma
    B[3, 0:-1] = -gamma * beta
    B[0:-1, 3] = B[3, 0:-1].transpose()
    B[0:-1,0:-1] = np.matmul(beta.transpose(), beta)
    B[0:-1, 0:-1] *= ((gamma - 1) / Beta)
    B[0:-1, 0:-1] += np.eye(3)
    return B
```

The test function

```
def test(P1,P2, trans,method, precision = 3):
    import numpy as np
    import math
    try:
        d = distance(math.inf,math.inf, method = method)
        if not(math.isnan(d)):
            return "Did not pass."
            raise ValueError("distance function seems to return a constant")
    except:
        pass
    d1 = distance(P1, P2, method = method)
    UP1 = trans.dot(P1)
    UP2 = trans.dot(P2)
    d2 = distance(UP1, UP2, method = method)
    d1 = round(d1, precision)
    d2 = round(d2, precision)
    if d1 == d2:
        return "Passed."
    else:
        return "Did not pass."
```

Euclidean geometry

We will first generate two points $P1$ and $P2$, an orthogonal transformation, and a translation. The affine transformations we chose to apply to $P1$ and $P2$ are (1). O, an orthogonal transformation; (2). T, a translation; (3). OT, a translation followed by an orthogonal transformation; and (4). TO, an orthogonal transformation followed by a translation. The results are shown in the following tables: ("+" in the column name stands for function composition "$\circ$")

	Coordinate
P1	[0.16539479 -0.42516585 -0.00659501 1.]
P2	[1.54678283 -0.23006551 1.29197053 1.]
OP1	[-0.42018259 -0.15084892 -0.09410701 1.]
OP2	[-1.41210367 1.40374058 0.38746709 1.]
TP1	[0.19117562 1.14420415 -0.34154722 1.]
TP2	[1.57256367 1.33930449 0.95701832 1.]
OTP1	[0.78735548 0.25765499 0.88096028 1.]
OTP2	[-0.2045656 1.81224448 1.36253438 1.]
TOP1	[-0.39440176 1.41852109 -0.42905921 1.]
TOP2	[-1.38632283 2.97311059 0.05251488 1.]

Figure 1: The coordinates of the original points and the points after transformations.

Original	Orthogonal	Translation	Ortho + Translation	Translation + Ortho
1.906	1.906 Passed.	1.906 Passed.	1.906 Passed.	1.906 Passed.

Figure 2: Tests showing that the distance is invariant under these aforementioned transformations.

As we can see from Figure 2, the distance between $P1$ and $P2$ is invariant under these four affine transformations.

Spherical geometry

We will generate two points $P1$ and $P2$ and two orthogonal transformations O and $O1$. The results are shown in the following tables: ("+" in the column name stands for function composition "$\circ$")

	Coordinate
P1	[0.82863676 -0.40006619 -0.18354435 0.34586071]
P2	[0.12639021 0.86426412 0.06226809 -0.48290343]
OP1	[0.75627383 -0.29471153 -0.57365179 0.11008468]
OP2	[0.11518995 -0.02727687 0.77893621 -0.61582922]
O1P1	[-0.07399994 -0.82690456 -0.05912556 0.55430771]
O1P2	[-0.01623616 0.4739988 -0.86844302 -0.14445847]

Figure 3: The coordinates of the original points and the points after transformations.

Original	Orthogonal	Orthogonal1
2.004	2.004 Passed.	2.004 Passed.

Figure 4: Tests showing that the distance is invariant under these aforementioned transformations.

As we can see from Figure 4, the distance between $P1$ and $P2$ is invariant under these two orthogonal transformations.

Hyperbolic geometry

Analogous to the Euclidean geometry case, we will first generate two points $P1$ and $P2$, a pure spatial rotation that will not change the time coordinate, and a Lorentz boost. The Lorentz transformations we chose to apply to $P1$ and $P2$ are (1). O, an pure rotation; (2). T, a Lorentz boost; (3). OT, a Lorentz boost followed by a pure rotation; and (4). TO, a pure rotation followed by a Lorentz boost. The results are shown in the following tables: ("+" in the column name stands for function composition "$\circ$")

	Coordinate
P1	[-0.07772535 -0.78670724 -1.0205745 1.63294881]
P2	[-1.44195049 -1.11198336 -0.83212132 2.23789055]
OP1	[0.58892278 1.14651959 0.0720042 1.63294881]
OP2	[-0.32007762 1.70290202 1.00291034 2.23789055]
TP1	[-0.0084158 -0.12115067 -0.86888004 1.33030103]
TP2	[-1.34745055 -0.20453186 -0.62529387 1.80234533]
OTP1	[0.68223317 0.49089948 0.25154808 1.33030103]
OTP2	[-0.1928538 0.80899835 1.24770901 1.80234533]
TOP1	[0.68323052 2.05212545 0.27841099 2.39906971]
TOP2	[-0.18838627 2.9674903 1.29113684 3.39242132]

Figure 5: The coordinates of the original points and the points after transformations.

Original	Rotation	Boost	Rotation + Boost	Boost + Rotation
1.205	1.205 Passed.	1.205 Passed.	1.205 Passed.	1.205 Passed.

Figure 6: Tests showing that the distance is invariant under these aforementioned transformations.

As we can see from Figure 6, the distance between $P1$ and $P2$ is invariant under these four Lorentz transformations.

Flexibility

A

No. In fact any function that returns a constant will not pass the test since none of these distance formulas are defined for points at infinity and if the inputs of the python function participate in the calculation, the function should either raise an error or return the python object “math.nan”. However, in python, the expression “math.nan == math.nan” returns False, so a function that always returns “math.nan” will not pass the test.

B

If the transformations generated are always identity, then they can pass the test. Or if the random points generated are always identical, then they can pass the test.

Appendix

This the code we used to for the tests.

```
import numpy as np
from prettytable import PrettyTable
###################
# Euclidean
###################
P1 = genPoint(3)
P2 = genPoint(3)
# Orthogonal transformation
O = orthoTrans(3)
# Translation
T = translation(3)
# Translation followed by an Orthogonal transformation
OT = O.dot(T)
# Orthogonal transformation followed by a Translation
TO = T.dot(O)
#
OP1 = O.dot(P1)
OP2 = O.dot(P2)
TP1 = T.dot(P1)
TP2 = T.dot(P2)
OTP1 = OT.dot(P1)
OTP2 = OT.dot(P2)
TOP1 = TO.dot(P1)
TOP2 = TO.dot(P2)
# Construct tables
# The table that shows the coordinates of these points
t = PrettyTable(['', 'Coordinate'])
t.add_row(['P1', P1])
t.add_row(['P2', P2])
t.add_row(['OP1', OP1])
t.add_row(['OP2', OP2])
t.add_row(['TP1', TP1])
t.add_row(['TP2', TP2])
t.add_row(['OTP1', OTP1])
t.add_row(['OTP2', OTP2])
t.add_row(['TOP1', TOP1])
t.add_row(['TOP2', TOP2])
print(t)
# The table that shows the distances and if they passed the test
t1 = PrettyTable(['Original', 'Orthogonal', 'Translation',
                  'Ortho + Translation', 'Translation + Ortho'])
t1.add_row([round(distance(P1,P2),3),
            round(distance(OP1,OP2),3),
            round(distance(TP1,TP2),3),
            round(distance(OTP1,OTP2),3),
            round(distance(TOP1,TOP2),3)])
t1.add_row(['',
            test(P1,P2, O,'Homogenous'),
            test(P1,P2, T,'Homogenous'),
            test(P1,P2, OT,'Homogenous'),
```

```
            test(P1,P2, TO,'Homogenous')])
print(t1)


###################
# Spherical
###################
P1 = genPoint(4, method = "Spherical")
P2 = genPoint(4, method = "Spherical")
O = orthoTrans(4, method = "Spherical")
O1 = orthoTrans(4, method = "Spherical")

OP1 = O.dot(P1)
OP2 = O.dot(P2)
O1P1 = O1.dot(P1)
O1P2 = O1.dot(P2)

# Construct tables
# The table that shows the coordinates of these points
t = PrettyTable(['', 'Coordinate'])
t.add_row(['P1', P1])
t.add_row(['P2', P2])
t.add_row(['OP1', OP1])
t.add_row(['OP2', OP2])
t.add_row(['O1P1', O1P1])
t.add_row(['O1P2', O1P2])
print(t)
# The table that shows the distances and if they passed the test
t1 = PrettyTable(['Original', 'Orthogonal', 'Orthogonal1'])
t1.add_row([round(distance(P1,P2, method = "Spherical"),3),
            round(distance(OP1,OP2, method = "Spherical"),3),
            round(distance(O1P1,O1P2, method = "Spherical"),3)])
t1.add_row(['',
            test(P1,P2, O,"Spherical"),
            test(P1,P2, O1,"Spherical")])
print(t1)


###################
# Hyperbolic
###################
P1 = genPoint(4, method = "Hyperbolic")
P2 = genPoint(4, method = "Hyperbolic")
# Pure rotation
O = orthoTrans(4, method = "Hyperbolic")
# Pure boost
beta = np.zeros((1,3))
beta[0,:] = (2*np.random.rand(1)-1) * genPoint(3, method = "Spherical")
T = boost(beta)
# Boost followed by a rotation
OT = O.dot(T)
# Rotation followed by a boost
TO = T.dot(O)
#
OP1 = O.dot(P1)
OP2 = O.dot(P2)
TP1 = T.dot(P1)
TP2 = T.dot(P2)
OTP1 = OT.dot(P1)
OTP2 = OT.dot(P2)
TOP1 = TO.dot(P1)
TOP2 = TO.dot(P2)
# Construct tables
# The table that shows the coordinates of these points
t = PrettyTable(['', 'Coordinate'])
t.add_row(['P1', P1])
t.add_row(['P2', P2])
t.add_row(['OP1', OP1])
t.add_row(['OP2', OP2])
t.add_row(['TP1', TP1])
t.add_row(['TP2', TP2])
t.add_row(['OTP1', OTP1])
t.add_row(['OTP2', OTP2])
```

```python
t.add_row(['TOP1', TOP1])
t.add_row(['TOP2', TOP2])
print(t)
# The table that shows the distances and if they passed the test
t1 = PrettyTable(['Original', 'Rotation', 'Boost',
                  'Rotation + Boost', 'Boost + Rotation'])
t1.add_row([round(distance(P1,P2, method = "Hyperbolic"),3),
            round(distance(OP1,OP2, method = "Hyperbolic"),3),
            round(distance(TP1,TP2, method = "Hyperbolic"),3),
            round(distance(OTP1,OTP2, method = "Hyperbolic"),3),
            round(distance(TOP1,TOP2, method = "Hyperbolic"),3)])
t1.add_row(['',
            test(P1,P2, O,"Hyperbolic"),
            test(P1,P2, T,"Hyperbolic"),
            test(P1,P2, OT,"Hyperbolic"),
            test(P1,P2, TO,"Hyperbolic")])
print(t1)
```