Description of Script SO3 –by Elizabeth B. Torres, Perceptual Science 16:830:646:01 F11-

Rutgers U-Psych

This script generates 3 angles alpha, beta, gamma within the proper boundaries to first form a rotation matrix in SO(3) by first rotating around X, then around Y and then around Z. This is the Euler angle sequence that the Polhemus system that we have in the lab uses.

The function Angles2Matrix takes in the 3 angles within the proper bounds and outputs an element M of SO(3) with Det = 1, and inverse of M equals the transpose of M as we have seen before. Try some examples and compute det(M), M' and then inv(M), M' M or M inv(M) gives eyes(3) the multip identity of the group.

```
>>     alpha(i) = -pi + 2*pi*rand;     beta(i) = pi*rand;     ganma(i) = -pi +
2*pi*rand;

        %Convert from Euler to Quat

    [ M(:,:,i) ]=Angles2Matrix( alpha(i), beta(i), ganma(i) );

M =

    0.0392    0.9762    0.2133

    0.0856    0.2094   -0.9741

   -0.9956    0.0564   -0.0754

>> det(M)

ans =     1

>> M'*M

ans =

    1.0000         0         0

         0    1.0000   -0.0000

         0   -0.0000    1.0000

>> inv(M)*M

ans =

    1.0000    0.0000   -0.0000

   -0.0000    1.0000   -0.0000

         0    0.0000    1.0000

alpha =    1.1419

>> alpha*180/pi
```

```
ans =   65.4234 >> beta*180/pi ans =   84.5983 >> ganma*180/pi ans =  143.1967
```

Notice that the rotation vector is unitary (one of the standard basis in each case) and that we use the active perspective (point rotates). Also notice that the Ry matrix uses the opposite angle because of the limits for beta $0 \le \beta \le \pi$ which is always for positive values, whereas α, γ are as $-\pi < \alpha, \gamma \le \pi$

If you play with the function Angles2 Matrix and give values to these angles some ambiguities will emerge. For example if β is π R(α,β,γ) and R(α+ω,π,γ+ω) entail the same rotation for any arbitrary angle ω.

Given R(α,β,γ), its inverse, clearly is obtained by taking inverses in the opposite order

$$\left\{R\left(\alpha\beta\gamma\right)\right\}^{-1} = R\left(-\gamma,-\beta,-\alpha\right) = R\left(-\gamma \pm \pi,-\beta,-\alpha \pm \pi\right)$$

(Because of the ambiguity mentioned above, ignore the π when β=π).

The next function Get_Angle_Vector takes the Matrix M from SO(3) and outputs the angle-vector parametrization that we studied in class in detail.

```
[ delta(i), n(i,:) ]=Get_Angle_Vector( M(:,:,i) );

>> delta

delta =    1.9970 %radians

>> delta*180/pi

ans =  114.4206

the angle delta In degrees

%the unit vector using the norm or its definition in matlab

n =    0.5659    0.6638   -0.4891

>> norm(n)

ans =     1

sqrt(sum(n.^2)) %.^2 is the element wise power

ans =     1
```

You can also now obtain an orthogonal matrix A using the angle-vector param. This formula uses the skewed symmetric matrix and the matrix expansion. We did not go over it in class but it essentially builds a matrix of det +/- 1, but since we come from an SO(3) element which is a rot operator that preserves length and angles, these parametrizations are interchangeable so we end up with a matrix A in SO(3)

```
[ A(:,:,i) ]=Build_Ort_Matrix( delta(i), n(i,:) );
```

```
>> A


A =

    0.0392    0.9762    0.2133

    0.0856    0.2094   -0.9741

   -0.9956    0.0564   -0.0754

>> det(A)

ans =    1.0000

>> A'*A

ans =

    1.0000   -0.0000    0.0000

   -0.0000    1.0000   -0.0000

    0.0000   -0.0000    1.0000

>> inv(A)*A

ans =

    1.0000    0.0000   -0.0000

         0    1.0000   -0.0000

         0   -0.0000    1.0000
```

Finally we get to the function Quaternion_From_Angle_Vector which takes the unit vector n and the angle delta from M (which are the same as those from A since we constructed A from it) and output the angle theta and the unit quaternion

```
quat =  0.4757    0.5580   -0.4111    0.5416

>> norm(quat)

ans =     1
```

We will learn in class today all that we need to know about constructing this quaternion operator to rotate vectors in R3 but just notice that we have used the Euclidean norm here to obtain the norm of the unit quaternion

```
>> sum(quat.^2)


ans =     1
```

Notice inside this function the various parametrizations (commented out) that enable us to go from one representation to another while preserving the vector length, including from Euler angles to quaternions.

Finally we obtain the angle vector parametrization of the quaternion using the function Angle_Vector_from_Quat, which takes a unit quaternion and outputs a unit vector and an angle of rotation.

```
v =    0.5659    0.6638   -0.4891

>> theta

theta =    1.9970 %radians

>> theta*180/pi

ans =  114.4206  %degrees
```

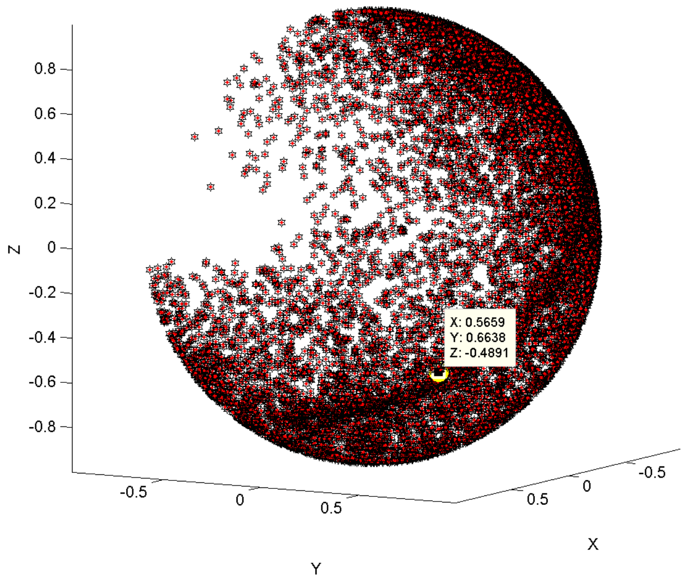
and these(v, theta) are the same as before, when we computed (delta, n) from M, built A with them and then built the quaternion with (delta, n)

```
>> n

n =    0.5659    0.6638   -0.4891

delta * 180/pi

ans =  114.4206
```



And here you have our unit vectors v and n (in yellow) along with other ~10,000 randomly generated examples to cover the unit sphere, thanks to Euler, Rodrigues and Hamilton. Math is always right!