JCTVC-AB0023-v3

Roll equations in omnidirectional projection indicator SEI message

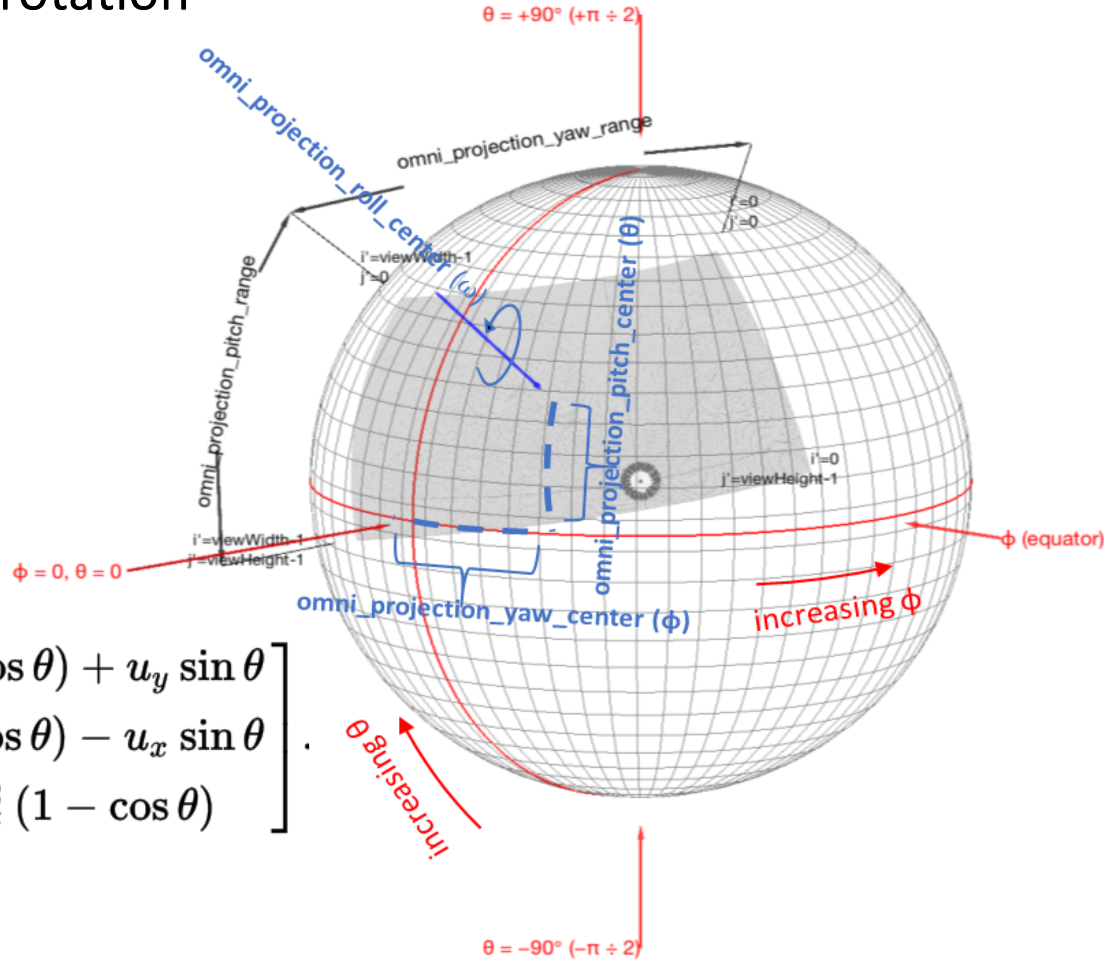
(referring to current SEI draft: JCTVC-AA1005)

Current state of roll equations:

- JCTVC-AA1005 currently defines roll as "…a clockwise rotation around the pitch_center when viewed from the origin looking outwards (anticlockwise when looking toward the origin), i.e., the forward vector after yaw and pitch rotation"

$$\phi = \mathrm{Cos}(\omega') * \alpha - \mathrm{Sin}(\omega') * \beta$$
$$\theta = \mathrm{Sin}(\omega') * \alpha + \mathrm{Cos}(\omega') * \beta$$

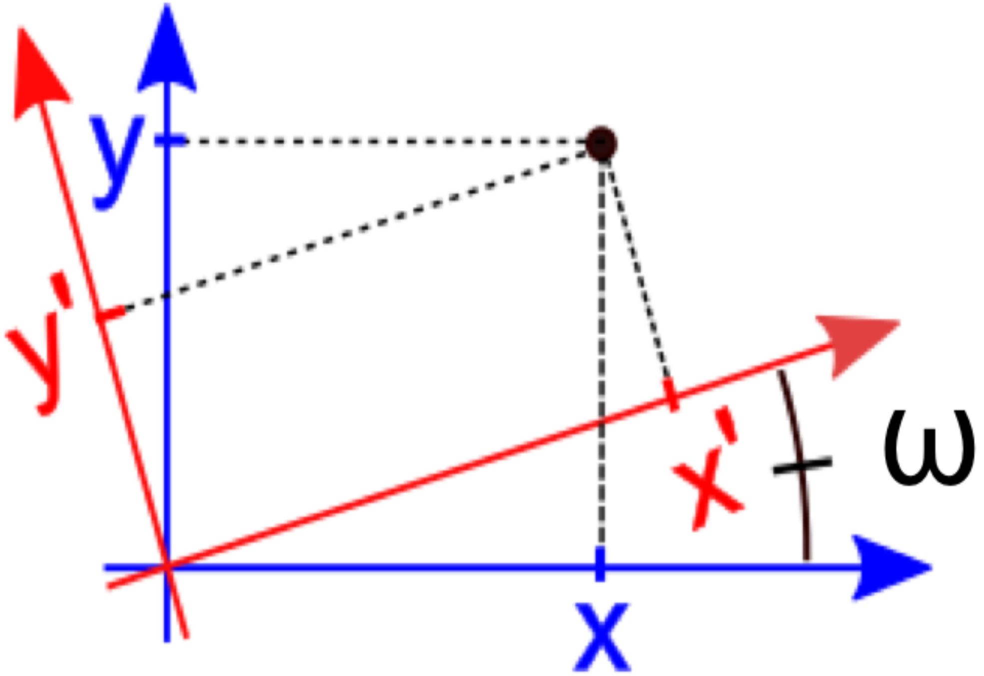
The above two equations do not reflect intent of the roll operation… Euler equations about vector u (roll center) needed instead…]



$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}.$$

- JVET-F1003 defines rotations about X,Y,Z axis

Counter-clockwise 2D rotation



x' = x * cos(ω) - y * sin(ω)
y' = x * sin(ω) + y * cos(ω)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\omega) & -\sin(\omega) \\ \sin(\omega) & \cos(\omega) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

counter-clockwise:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\omega) & -\sin(\omega) \\ \sin(\omega) & \cos(\omega) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

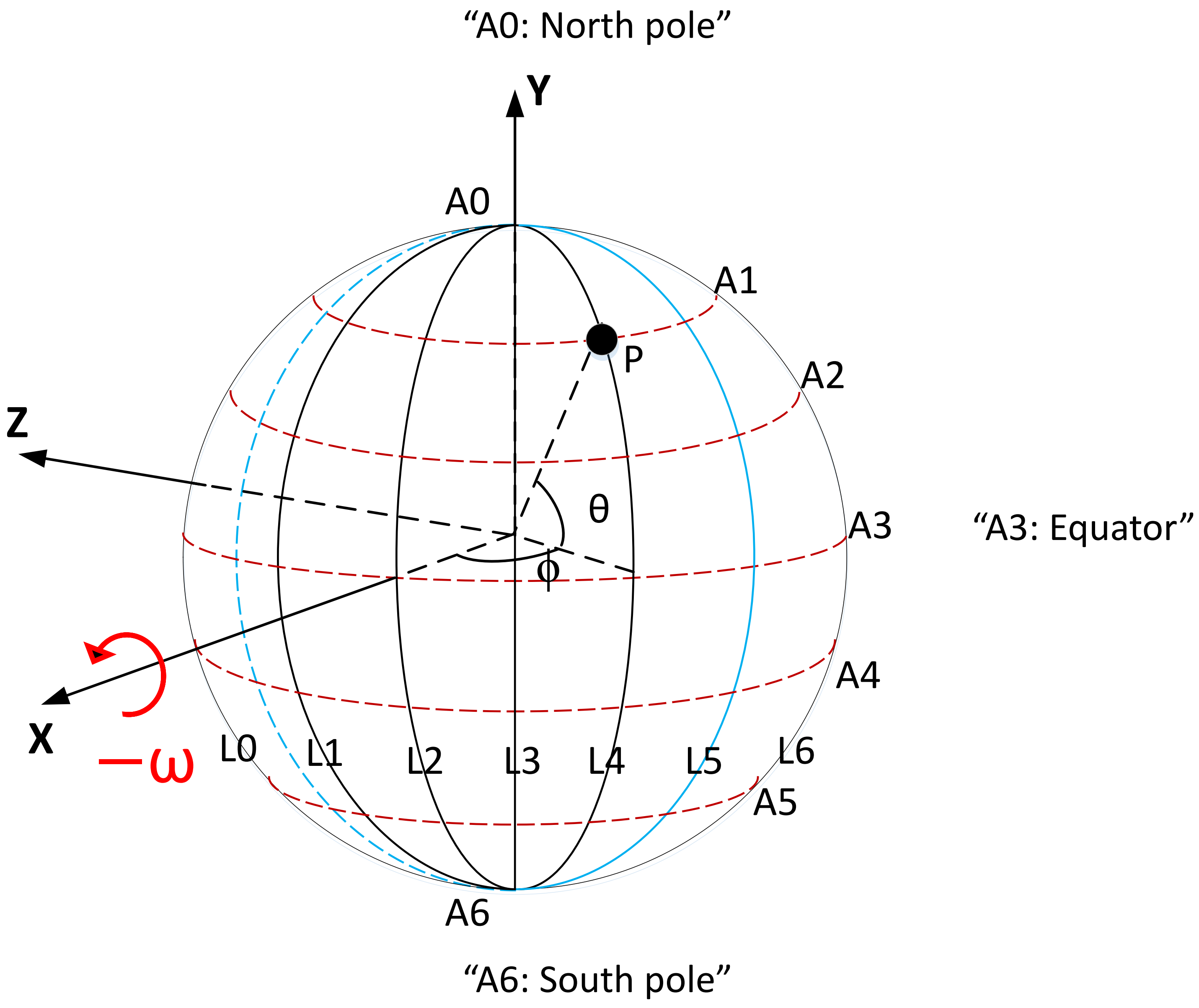
clockwise (ω’ = -ω):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\omega') & \sin(\omega') \\ -\sin(\omega') & \cos(\omega') \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

generalized rotation in 3D XYZ Cartesian space, where a clockwise rotation is performed of two axis (a,b) around a third axis (c) normal to the (a,b) plane

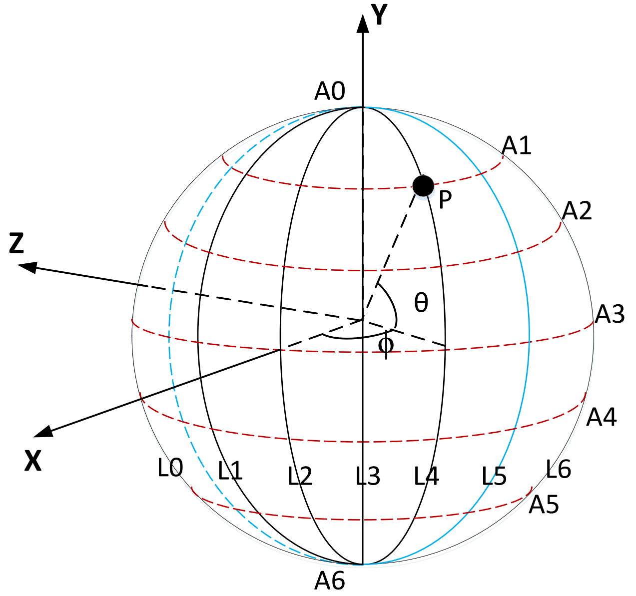
$$\begin{bmatrix} a' \\ b' \\ c' \end{bmatrix} = \begin{bmatrix} \cos(\omega) & \sin(\omega) & 0 \\ -\sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

JVET-F1003



"A0: North pole"
Y
A0
A1
P
A2
Z
θ
A3
"A3: Equator"
φ
A4
X
−ω
L0
L1
L2
L3
L4
L5
L6
A5
A6
"A6: South pole"

JVET-F1003



Forward (pre-processing on encoder):

$$R_{XYZ} = R_Y(yaw) \cdot R_Z(-\text{pitch}) \cdot R_X(roll)$$

Inverse (post-processing "render" after decoder):

$$R'_{XYZ} = R_X(-roll) \cdot R_Z(pitch) \cdot R_Y(-yaw)$$

(Step 3) (Step 2) (Step 1)

JVET-F1003

Convert spherical surface coordinates to Cartesian XYZ

$$X = \cos(\theta)\cos(\varphi)$$

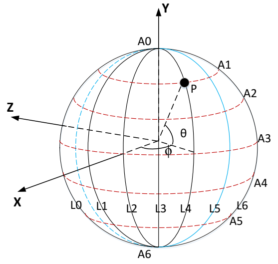
$$Y = \sin(\theta)$$

$$Z = -\cos(\theta)\sin(\varphi)$$

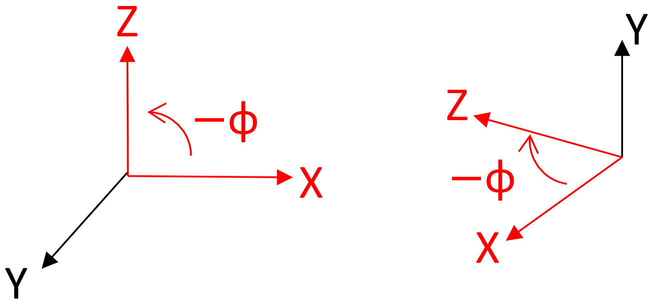
Inverse:

$$\varphi = \tan2^{-1}(-Z, X)$$

$$\theta = \sin^{-1}(Y/(X^2+Y^2+Z^2)^{1/2})$$

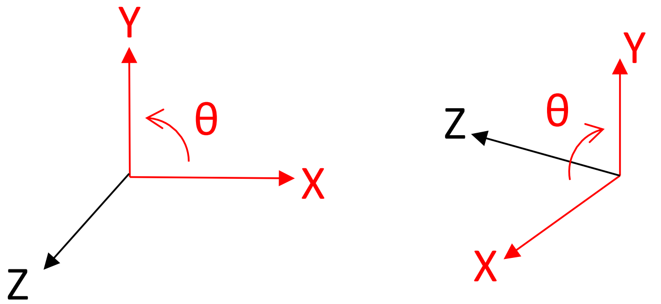


$$R'_{XYZ} = R_X(-roll) \cdot R_Z(pitch) \cdot R_Y(-yaw)$$



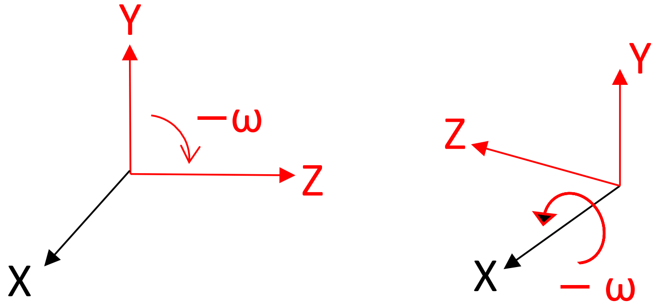
$$R_Y(yaw) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(yaw) & 0 & \sin(yaw) \\ 0 & 1 & 0 \\ -\sin(yaw) & 0 & \cos(yaw) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

yaw: clockwise
−yaw: counter-clockwise



$$R_Z(pitch) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(pitch) & -\sin(pitch) & 0 \\ \sin(pitch) & \cos(pitch) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

pitch: counter-clockwise



$$R_X(roll) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

roll: counter-clockwise
-roll: clockwise

https://jvet.hhi.fraunhofer.de/svn/svn_360Lib/tags/360Lib-3.0rc1 /source/Lib/TLib360/TGeometry.cpp

```
Void TGeometry::invRotate3D(SPos& sPos, Int iRoll, Int iPitch, Int iYaw){
  POSType x = sPos.x; POSType y = sPos.y;  POSType z = sPos.z;
  if(iYaw){
    POSType rcos = scos((POSType)(iYaw*S_PI/(180.0*SVIDEO_ROT_PRECISION)));
    POSType rsin = ssin((POSType)(iYaw*S_PI/(180.0*SVIDEO_ROT_PRECISION)));
    POSType t1 = rcos*x + rsin*z;
    POSType t2 = -rsin*x + rcos*z;
    x = t1; z = t2;
  }
  if(iPitch){
    POSType rcos = scos((POSType)(-iPitch*S_PI/(180.0*SVIDEO_ROT_PRECISION)));
    POSType rsin = ssin((POSType)(-iPitch*S_PI/(180.0*SVIDEO_ROT_PRECISION)));
    POSType t1 = rcos*x - rsin*y;
    POSType t2 = rsin*x + rcos*y;
    x = t1;  y = t2;
  }
  if(iRoll){
    POSType rcos = scos((POSType)(iRoll*S_PI/(180.0*SVIDEO_ROT_PRECISION)));
    POSType rsin = ssin((POSType)(iRoll*S_PI/(180.0*SVIDEO_ROT_PRECISION)));
    POSType t1 = rcos*y - rsin*z;
    POSType t2 = rsin*y + rcos*z;
    y = t1;  z = t2;
  }
  sPos.x = x; sPos.y = y; sPos.z = z;
}
```

JVET-F0065 / JCTVC-AA0033

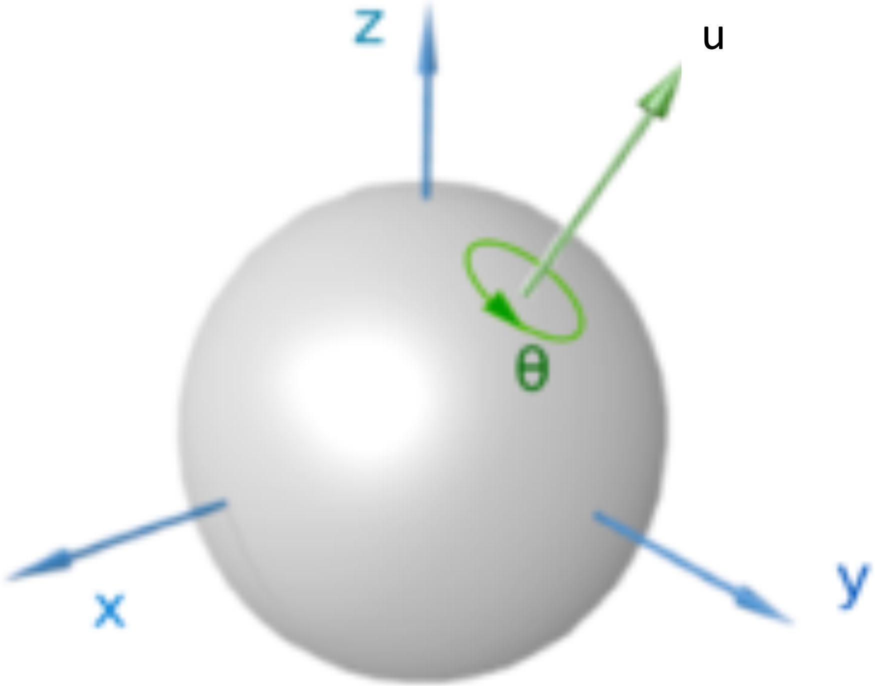
https://jvet.hhi.fraunhofer.de/svn/svn_360Lib/tags/360Lib-3.0rc1/source/Lib/TLib360/TGeometry.cpp

```
S_EPS = 1.0e-6
#define satan2(y, x)    atan2((Double)(y), (Double)(x))

yaw  =  (-atan2(z,  x))   * 180.0/ pi;
len  =  sqrt(x*x  +  y*y  +  z*z);
pitch  =  90.0  -((len < S_EPS? 0.5 : acos(y/len)/pi)*180.0);
```

Euler's rotation: math reflecting intent of JCTVC-AA1005 roll...

Perform yaw, pitch rotations, thus producing a new X,Y,Z axis..



Compute the vector u (center_roll)

… then apply R:

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}.$$

Where to find more information on rotation…

- https://en.wikipedia.org/wiki/Rotation_matrix