# 1 Quaternion Multiplication

$$Q_{\mathrm{Wnew}} = Q_{\mathrm{W1}}(Q_{\mathrm{W2}}) - Q_{\mathrm{X1}}(Q_{\mathrm{X2}}) - Q_{\mathrm{Y1}}(Q_{\mathrm{Y2}}) - Q_{\mathrm{Z1}}(Q_{\mathrm{Z2}}) Q_{\mathrm{Xnew}} = Q_{\mathrm{W1}}(Q_{\mathrm{X2}}) - Q_{\mathrm{X1}}(Q_{\mathrm{W2}}) - Q_{\mathrm{Y1}}(Q_{\mathrm{Z2}}) - Q_{\mathrm{Z1}}(Q_{\mathrm{Y}}$$

$$(1)$$

# 2 Quaternion Division

$$
\begin{aligned}
Q_{\mathrm{Wnew}} &= \ Q_{\mathrm{W1}}(Q_{\mathrm{W2}}) + Q_{\mathrm{X1}}(Q_{\mathrm{X2}}) + Q_{\mathrm{Y1}}(Q_{\mathrm{Y2}}) + Q_{\mathrm{Z1}}(Q_{\mathrm{Z2}}) \\
Q_{\mathrm{Xnew}} &= -Q_{\mathrm{W1}}(Q_{\mathrm{X2}}) + Q_{\mathrm{X1}}(Q_{\mathrm{W2}}) + Q_{\mathrm{Y1}}(Q_{\mathrm{Z2}}) - Q_{\mathrm{Z1}}(Q_{\mathrm{Y2}}) \\
Q_{\mathrm{Ynew}} &= -Q_{\mathrm{W1}}(Q_{\mathrm{Y2}}) - Q_{\mathrm{X1}}(Q_{\mathrm{Z2}}) + Q_{\mathrm{Y1}}(Q_{\mathrm{W2}}) + Q_{\mathrm{Z1}}(Q_{\mathrm{X2}}) \\
Q_{\mathrm{Znew}} &= -Q_{\mathrm{W1}}(Q_{\mathrm{Z2}}) + Q_{\mathrm{X1}}(Q_{\mathrm{Y2}}) - Q_{\mathrm{Y1}}(Q_{\mathrm{X2}}) + Q_{\mathrm{Z1}}(Q_{\mathrm{W2}})
\end{aligned}
$$

# 3 Quaternion Conjugate

$$
\begin{aligned}
Q_{\mathrm{Wconjugate}} &= \ Q_{\mathrm{Winput}} \\
Q_{\mathrm{Xconjugate}} &= -Q_{\mathrm{Xinput}} \\
Q_{\mathrm{Yconjugate}} &= -Q_{\mathrm{Yinput}} \\
Q_{\mathrm{Zconjugate}} &= -Q_{\mathrm{Zinput}}
\end{aligned}
$$

# 4 Quaternion Normal

$$Normal_{\mathrm{rational}} = Q_W^2 + Q_X^2 + Q_Y^2 + Q_Z^2$$

# 5 Quaternion Multiplicative Inverse

$$Q_{\mathrm{reciprocal}} = Q_{\mathrm{conjugate}} \frac{1}{Q_{\mathrm{normal}}}$$

# 6 Quaternion Vector Rotation

$$
\begin{aligned}
Q_{\mathrm{Wbivector}} &= \ 0 \\
Q_{\mathrm{Xbivector}} &= V_X
\end{aligned}
$$

$Q_{\mathrm{Ybivector}} = V_Y$
$Q_{\mathrm{Zbivector}} = V_Z$

$Q_{\mathrm{rotation}} = Q_{\mathrm{current}}(Q_{\mathrm{bivector}})(Q_{\mathrm{reciprocal}})$

$V_{\mathrm{Xrotated}} = Q_{\mathrm{Xrotation}}$
$V_{\mathrm{Yrotated}} = Q_{\mathrm{Yrotation}}$
$V_{\mathrm{Zrotated}} = Q_{\mathrm{Zrotation}}$

# 7   Quaternion Vector Rotation Removal

$Q_{\mathrm{Wbivector}} = \ 0$
$Q_{\mathrm{Xbivector}} = V_X$
$Q_{\mathrm{Ybivector}} = V_Y$
$Q_{\mathrm{Zbivector}} = V_Z$

$Q_{\mathrm{rotation}} = Q_{\mathrm{conjugate}}(Q_{\mathrm{bivector}})(Q_{\mathrm{reciprocal}})$

$V_{\mathrm{Xrotated}} = Q_{\mathrm{Xrotation}}$
$V_{\mathrm{Yrotated}} = Q_{\mathrm{Yrotation}}$
$V_{\mathrm{Zrotated}} = Q_{\mathrm{Zrotation}}$

# 8   Unit Quaternion

$$Q_{\mathrm{Wunit}} = \frac{Q_{\mathrm{Winput}}}{Q_{\mathrm{normal}}}$$
$$Q_{\mathrm{Xunit}} = \frac{Q_{\mathrm{Xinput}}}{Q_{\mathrm{normal}}}$$
$$Q_{\mathrm{Yunit}} = \frac{Q_{\mathrm{Yinput}}}{Q_{\mathrm{normal}}}$$
$$Q_{\mathrm{Zunit}} = \frac{Q_{\mathrm{Zinput}}}{Q_{\mathrm{normal}}}$$

# 9   Quaternion Dot Product

$$D_{\mathrm{dot}} = Q_{\mathrm{W1}}(Q_{\mathrm{W2}}) + Q_{\mathrm{X1}}(Q_{\mathrm{X2}}) + Q_{\mathrm{Y1}}(Q_{\mathrm{Y2}}) + Q_{\mathrm{Z1}}(Q_{\mathrm{Z2}})$$

# 10   Quaternion Magnitude

$$M_{\mathrm{magnitude}} = \sqrt{Q_{\mathrm{normal}}}$$

# 11 Quaternion Additive Inverse

$$Q_{\text{Wnegative}} = -Q_{\text{Winput}}$$
$$Q_{\text{Xnegative}} = -Q_{\text{Xinput}}$$
$$Q_{\text{Ynegative}} = -Q_{\text{Yinput}}$$
$$Q_{\text{Znegative}} = -Q_{\text{Zinput}}$$

# 12 Quaternion Smooth Interpolation Between Quaternions

$Q_{\text{initial}} = Q_{\text{Unit initial}}$
$Q_{\text{final}} = Q_{\text{Unit final}}$

$D_{\text{dot}} = Q_{\text{initial}} \cdot Q_{\text{final}}$

$$Q_{\text{initial}} = \begin{cases} Q_{\text{initial}} = Q_{\text{initial(additive inverse)}}, & \text{if } D_{\text{dot}} < 0. \\ Q_{\text{initial}}, & \text{otherwise.} \end{cases}$$

$D_{\text{dot}} = |D_{\text{dot}}|$

$$D_{\text{dot}} = \begin{cases} 1, & \text{if } D_{\text{dot}} > 1. \\ D_{\text{dot}}, & \text{otherwise.} \end{cases}$$

$\theta = arccos(D_{\text{dot}}) \times ratio$

$Q_{\text{orthonomal}} = Q_{\text{final}} - Q_{\text{initial}} \times D_{\text{dot}}$
$Q_{\text{output}} = Q_{initial} \times cos(\theta) + Q_{\text{orthonomal}} \times sin(\theta)$

# 13 Quadcopter Combined Thrust Vector

$Q_{\text{change}} = (\frac{2 \times (Q_{\text{target}} - Q_{\text{current}}) \times Q_{\text{current conjugate}}}{dT})$

$V_{\text{Xchange}} = Q_{\text{Xchange}}$
$V_{\text{Ychange}} = Q_{\text{Ychange}}$
$V_{\text{Zchange}} = Q_{\text{Zchange}}$

$V_{\text{RotationOutput}} = FeedbackController_{\text{rotation}}.Calculate(0, V_{\text{change}})$

$$V_{\text{PositionOutput}} = FeedbackController_{\text{position}}.Calculate(0, V_{\text{CurrentPosition}} - V_{\text{TargetPosition}})$$

$$V_{\text{YThrusterBOutput}} = -V_{\text{XRotationOutput}} + V_{\text{ZRotationOutput}} - V_{\text{YRotationOutput}}$$
$$V_{\text{YThrusterCOutput}} = -V_{\text{XRotationOutput}} - V_{\text{ZRotationOutput}} + V_{\text{YRotationOutput}}$$
$$V_{\text{YThrusterDOutput}} = \phantom{-}V_{\text{XRotationOutput}} - V_{\text{ZRotationOutput}} - V_{\text{YRotationOutput}}$$
$$V_{\text{YThrusterEOutput}} = \phantom{-}V_{\text{XRotationOutput}} + V_{\text{ZRotationOutput}} + V_{\text{YRotationOutput}}$$

$$V_{\text{HoverAngles}} = RotationToHoverAngles(Q_{\text{CurrentRotation}})$$

$$V_{\text{PositionOutput}} = CalculateRotationOffset(Q_{\text{CurrentRotation}}).RotateVector(V_{\text{PositionOutput}})$$

$$V_{\text{XPositionOutput}} = V_{\text{XPositionOutput}} + V_{\text{ZHoverAngles}}$$
$$V_{\text{ZPositionOutput}} = V_{\text{ZPositionOutput}} - V_{\text{XHoverAngles}}$$

$$V_{\text{ThrusterBOutput}} = V_{\text{ThrusterBOutput}} + V_{\text{PositionOutput}}$$
$$V_{\text{ThrusterCOutput}} = V_{\text{ThrusterCOutput}} + V_{\text{PositionOutput}}$$
$$V_{\text{ThrusterDOutput}} = V_{\text{ThrusterDOutput}} + V_{\text{PositionOutput}}$$
$$V_{\text{ThrusterEOutput}} = V_{\text{ThrusterEOutput}} + V_{\text{PositionOutput}}$$

## 14 Quadcopter Thruster Position Calculation

$$V_{\text{ThrusterBPosition}} = Q_{\text{CurrentRotation}}.RotateVector(V_{\text{ThrusterBOffset}}) + V_{\text{TargetPosition}}$$
$$V_{\text{ThrusterCPosition}} = Q_{\text{CurrentRotation}}.RotateVector(V_{\text{ThrusterCOffset}}) + V_{\text{TargetPosition}}$$
$$V_{\text{ThrusterDPosition}} = Q_{\text{CurrentRotation}}.RotateVector(V_{\text{ThrusterDOffset}}) + V_{\text{TargetPosition}}$$
$$V_{\text{ThrusterEPosition}} = Q_{\text{CurrentRotation}}.RotateVector(V_{\text{ThrusterEOffset}}) + V_{\text{TargetPosition}}$$

## 15 Quadcopter Hover Angle Calculation

$$DA_{\text{Direction}} = RotationMatrix.RotateVector(EA_{\text{rotate}}(0, -90, 0), DA_{\text{Direction}})$$

$$DA_{\text{Direction}} = RotationMatrix.RotateVector(EA_{\text{rotate}}(0, DA_{\text{Rotation}}, 0), DA_{\text{Direction}})$$

$$D_{\text{InnerJoint}} = RadiansToDegrees(arcsin(D_{\text{DirectionVectorZ}}))$$
$$D_{\text{OuterJoint}} = RadiansToDegrees(arctan2(D_{\text{DirectionVectorX}}, D_{\text{DirectionVectorY}}))$$

## 16 Quadcopter Estimate Position

$$V_{\text{TBThrust}} = Vector(0, ThrustBOutputY, 0)$$
$$V_{\text{TCThrust}} = Vector(0, ThrustCOutputY, 0)$$
$$V_{\text{TDThrust}} = Vector(0, ThrustDOutputY, 0)$$
$$V_{\text{TEThrust}} = Vector(0, ThrustEOutputY, 0)$$

$$Q_{\text{TBR}} = EA(ThrustBOutput.X, 0, -ThrustBOutput.Z)$$

$$Q_{\text{TCR}} = EA(ThrustCOutput.X, 0, -ThrustCOutput.Z)$$
$$Q_{\text{TDR}} = EA(ThrustDOutput.X, 0, -ThrustDOutput.Z)$$
$$Q_{\text{TER}} = EA(ThrustEOutput.X, 0, -ThrustEOutput.Z)$$

$$V_{\text{TBThrust}} = Q_{\text{TBR}}.RotateVector(TBThrust)$$
$$V_{\text{TCThrust}} = Q_{\text{TCR}}.RotateVector(TCThrust)$$
$$V_{\text{TDThrust}} = Q_{\text{TDR}}.RotateVector(TDThrust)$$
$$V_{\text{TEThrust}} = Q_{\text{TER}}.RotateVector(TEThrust)$$

$$V_{\text{ThrustSum}} = V_{\text{TBThrust}} + V_{\text{TCThrust}} + V_{\text{TDThrust}} + V_{\text{TEThrust}}$$
$$V_{\text{ThrustSum}} = Q_{\text{current}}.RotateVector(V_{\text{ThrustSum}})$$

$$V_{\text{XDragForce}} = D_{\text{AirDensity}} \times D_{\text{XCurrentVelocity}}^2 \times D_{\text{DragCoefficient}} \times Sign(XCurrentVelocity)$$
$$V_{\text{YDragForce}} = D_{\text{AirDensity}} \times D_{\text{YCurrentVelocity}}^2 \times D_{\text{DragCoefficient}} \times Sign(YCurrentVelocity)$$
$$V_{\text{ZDragForce}} = D_{\text{AirDensity}} \times D_{\text{ZCurrentVelocity}}^2 \times D_{\text{DragCoefficient}} \times Sign(ZCurrentVelocity)$$

$$V_{\text{CurrentAcceleration}} = V_{\text{ThrustSum}} + V_{\text{WorldAcceleration}}$$
$$V_{\text{CurrentVelocity}} = V_{\text{CurrentVelocity}} + V_{\text{CurrentAcceleration}} \times D_{\text{TimeDerivative}}$$
$$V_{\text{CurrentPosition}} = V_{\text{CurrentPosition}} + V_{\text{CurrentVelocity}} \times D_{\text{TimeDerivative}}$$

# 17   Quadcopter Estimate Rotation

$$V_{\text{TB}} = V_{\text{TBThrustVector}}$$
$$V_{\text{TC}} = V_{\text{TCThrustVector}}$$
$$V_{\text{TD}} = V_{\text{TDThrustVector}}$$
$$V_{\text{TE}} = V_{\text{TEThrustVector}}$$

$$V_{\text{TB}} = Q_{\text{CurrentRotation}}.RotateVector(TB)$$
$$V_{\text{TC}} = Q_{\text{CurrentRotation}}.RotateVector(TC)$$

$$V_{\text{TD}} = Q_{\text{CurrentRotation}}.RotateVector(TD)$$
$$V_{\text{TE}} = Q_{\text{CurrentRotation}}.RotateVector(TE)$$

$$D_{\text{Torque}} = D_{\text{ArmLength}} \times sin(180 - D_{\text{ArmAngle}})$$

$$V_{\text{XAngularAcceleration}} \quad = ( \quad V_{\text{TBY}} + V_{\text{TCY}} - V_{\text{TDY}} - V_{\text{TEY}}) \times D_{\text{Torque}}$$
$$V_{\text{YAngularAcceleration}} \quad = ( \quad V_{\text{TBX}} + V_{\text{TCX}} - V_{\text{TDX}} - V_{\text{TEY}}) \times D_{\text{Torque}}$$
$$V_{\text{YAngularAcceleration}} + = ( \quad V_{\text{TBZ}} - V_{\text{TCZ}} - V_{\text{TDZ}} + V_{\text{TEZ}}) \times D_{\text{Torque}}$$
$$V_{\text{ZAngularAcceleration}} \quad = (-V_{\text{TBY}} + V_{\text{TCY}} + V_{\text{TDY}} - V_{\text{TEY}}) \times D_{\text{Torque}}$$

$$V_{\text{XDragForce}} = D_{\text{AirDensity}} \times D^2_{\text{XAngularVelocity}} \times D_{\text{DragCoefficient}} \times Sign(X\,Angular\,Velocity)$$
$$V_{\text{YDragForce}} = D_{\text{AirDensity}} \times D^2_{\text{YAngularVelocity}} \times D_{\text{DragCoefficient}} \times Sign(Y\,Angular\,Velocity)$$
$$V_{\text{ZDragForce}} = D_{\text{AirDensity}} \times D^2_{\text{ZAngularVelocity}} \times D_{\text{DragCoefficient}} \times Sign(Z\,Angular\,Velocity)$$

$$V_{\text{DifferentialThrust}} = V_{\text{TB}} + V_{\text{TC}} - V_{\text{TD}} - V_{\text{TE}}$$
$$V_{\text{AngularAcceleration}} = V_{\text{AngularAcceleration}} + V_{\text{DifferentialThrust}}$$
$$V_{\text{AngularVelocity}} = V_{\text{AngularVelocity}} + (V_{\text{AngularAcceleration}} - V_{\text{DragForce}}) \times D_{\text{TimeDerivative}}$$
$$Q_{\text{AngularRotation}} = \frac{V_{\text{AngularVelocity}} \times D_{\text{TimeDerivative}}}{2}$$
$$Q_{\text{AngularPosition}} = Q_{\text{AngularPosition}} + Q_{\text{AngularRotation}} \times Q_{\text{AngularPosition}}$$

# 18 Quadcopter Calculate 3D Yaw

$$V_{\text{HoverRotation}} = HoverAnglesFromQuaternion(Q_{\text{CurrentRotation}})$$
$$Q_{\text{Hover}} = EA(V_{\text{HoverRotation}})$$
$$Q_{\text{Yaw3D}} = Q_{\text{Hover}} \times Q_{\text{CurrentRotation}}.MultiplicativeInverse()$$

# 19 Quadcopter Gimbal Locked Translation

$$D_{\text{Fade}} = TriangleWaveGenerator(-90 \longrightarrow 90) \implies (0 \longrightarrow 1)$$
$$D_{\text{InverseFade}} = 1 - D_{\text{fade}}$$
$$D_{\text{Rotation}} = 45 \times D_{\text{fade}}$$

$$V_{\text{RotatedControl}} = Q_{Calculate3DYaw(CurrentRotation)}.RotateVector(V_{\text{PositionFeedbackControlOutput}})$$

$$D_{\text{TBX}} = D_{\text{TBX}} \times D_{\text{InverseFade}} + D_{\text{RotatedControlX}} \times D_{\text{Fade}} + D_{\text{RotatedControlZ}} \times D_{\text{Fade}}$$
$$D_{\text{TCX}} = D_{\text{TCX}} \times D_{\text{InverseFade}} + D_{\text{RotatedControlX}} \times D_{\text{Fade}} + D_{\text{RotatedControlZ}} \times D_{\text{Fade}}$$

$$D_{\text{TDX}} = D_{\text{TDX}} \times D_{\text{InverseFade}} + D_{\text{RotatedControlX}} \times D_{\text{Fade}} + D_{\text{RotatedControlZ}} \times D_{\text{Fade}}$$
$$D_{\text{TEX}} = D_{\text{TEX}} \times D_{\text{InverseFade}} + D_{\text{RotatedControlX}} \times D_{\text{Fade}} + D_{\text{RotatedControlZ}} \times D_{\text{Fade}}$$

$$D_{\text{TBZ}} = D_{\text{TBZ}} \times D_{\text{InverseFade}} + D_{\text{Rotation}}$$
$$D_{\text{TCZ}} = D_{\text{TCZ}} \times D_{\text{InverseFade}} - D_{\text{Rotation}}$$
$$D_{\text{TDZ}} = D_{\text{TDZ}} \times D_{\text{InverseFade}} + D_{\text{Rotation}}$$
$$D_{\text{TEZ}} = D_{\text{TEZ}} \times D_{\text{InverseFade}} - D_{\text{Rotation}}$$

$$D_{\text{PostionControlX}} = D_{\text{PostionControlX}} \times D_{\text{InverseFade}}$$
$$D_{\text{PostionControlZ}} = D_{\text{PostionControlZ}} \times D_{\text{InverseFade}}$$

# 20    ADRC

$D_{\text{Amplificiation}}$
$D_{\text{Damping}}$
$D_{\text{Plant}}$
$P_{\text{PID}}$
$D_{\text{PrecisionModifier}}$

$$D_{\text{Precision}} = D_{\text{TimeDerivative}} \times D_{\text{PrecisionModifier}}$$
$$O_{\text{CurrentOutput}} = (P_{\text{PID}}.Calculate(D_{\text{SetPoint}}, D_{\text{ProcessVariable}}, D_{\text{TimeDerivate}}), O_{\text{PreviousOutput}})$$

$$S_{\text{State}} = ESO_{\text{ExtendedStateObserver}}.Observe(D_{\text{TimeDerivative}}, O_{\text{CurrentOutput}}, D_{\text{Plant}}, D_{\text{ProcessVariable}})$$

$$D_{\text{PreviousOutput}} = D_{\text{CurrentOutput}}$$
$$D_{\text{CurrentOutput}} = NLC_{\text{NonlinearCombiner}}.Combine(O_{\text{CurrentOutput}}, D_{\text{Plant}}, S_{\text{State}}, D_{\text{Precision}})$$

# 21    Setpoint Jump Prevention

$$D_{\text{Amp2Prec}} = D_{\text{Amplification}}^2 \times D_{\text{Precision}}$$
$$D_{\text{PrecTD}} = D_{\text{Precision}} \times D_{\text{TargetDerivative}}$$
$$D_{\text{TargetPrecTD}} = D_{\text{Target}} + D_{\text{PrecTD}}$$

$$D_{\text{A1}} = \sqrt{D_{\text{Amp2Prec}} \times (D_{\text{Amp2Prec}} + (8 \times |D_{\text{TargetPrecTD}}|))}$$

$$D_{\text{A2}} = \frac{D_{\text{PrecTD}} + Sign(D_{\text{TargetPrecTD}}) \times (D_{\text{A1}} - D_{\text{Amp2Prec}})}{2}$$

$$D_{\text{SignTargetPrecTD}} = \frac{Sign(D_{\text{TargetPrecTD}} + D_{\text{Amp2Prec}}) - Sign(D_{\text{TargetPrecTD}} - D_{\text{Amp2Prec}})}{2}$$

$$D_{\text{A}} = (D_{\text{PrecTD}} + D_{\text{TargetPrecTD}} - D_{\text{A2}}) \times D_{\text{SignTargetPrecTD}} + D_{\text{A2}}$$

$$D_{\text{SignA}} = \frac{Sign(D_{\text{A}} + D_{\text{Amp2Prec}}) - Sign(D_{\text{A}} - D_{\text{Amp2Prec}})}{2}$$

$$D_{\text{SetpointJumpPrevention}} = -D_{\text{Amplification}} \times (\frac{D_{\text{A}}}{D_{\text{Amp2Prec}}}) - Sign(D_{\text{A}}) \times D_{\text{SignA}} - D_{\text{Amplification}} \times Sign(D_{\text{A}})$$

## 22   Nonlinear Combiner

$$D_{\text{EstimationE1}} = O_{\text{CurrentOutput}} - S_{\text{StateZ1}}$$
$$D_{\text{EstimationE2}} = O_{\text{PreviousOutput}} - S_{\text{StateZ2}}$$
$$D_{\text{NominalControlSignal}} = SetPointJumpPrevention(D_{\text{Target}}, D_{\text{TargetDerivate}}, D_{\text{Amplificiation}}, D_{\text{Precision}})$$

$$D_{\text{Combine}} = \frac{D_{\text{NominalControlSignal}} + S_{\text{StateZ3}}}{D_{\text{Plant}}}$$

## 23   Extended State Observer

$$S_{\text{Gain1}} = 1$$

$$S_{\text{Gain2}} = \frac{1}{2 \times dT^{0.5}}$$

$$S_{\text{Gain3}} = \frac{2}{25 \times dT^{1.2}}$$

$$D_{\text{E}} = D_{\text{StateZ1}} - D_{\text{ProcessVariable}}$$
$$D_{\text{FE}} = NonlinearFunction(D_{\text{E}}, 0.5, dT)$$
$$D_{\text{FE1}} = NonlinearFunction(D_{\text{E}}, 0.25, dT)$$

$$S_{\text{StateZ1}} = S_{\text{StateZ1}} + (dT \times S_{\text{StateZ2}}) - (S_{\text{GainZ1}} \times D_{\text{E}})$$
$$S_{\text{StateZ2}} = S_{\text{StateZ2}} + (dT \times (S_{\text{StateZ3}} + (D_{\text{Plant}} \times D_{\text{Output}}))) - (S_{\text{GainZ2}} \times D_{\text{FE}})$$
$$S_{\text{StateZ3}} = S_{\text{StateZ3}} - S_{\text{GainZ3}} \times D_{\text{FE1}}$$

## 24 Nonlinear Function

$$Output = \begin{cases} \frac{\eta}{\delta^{(1-\alpha)}}, & \text{if } |\eta| \leq \delta. \\ |\eta|^{\alpha} \times Sign(\eta), & \text{otherwise.} \end{cases}$$

## 25 Axis-Angle to Quaternion

$$D_{\text{Rotation}} = \frac{AA_{\text{AxisAngleRotation}} \times \pi}{180}$$

$$D_{\text{Scale}} = sin(\frac{D_{\text{Rotation}}}{2})$$

$$Q_{\text{W}} = cos(\frac{D_{\text{Rotation}}}{2})$$
$$Q_{\text{X}} = AA_{\text{AxisAngleX}} \times D_{\text{Scale}}$$
$$Q_{\text{Y}} = AA_{\text{AxisAngleY}} \times D_{\text{Scale}}$$
$$Q_{\text{Z}} = AA_{\text{AxisAngleZ}} \times D_{\text{Scale}}$$

## 26 Quaternion to Axis-Angle

$$AA_{\text{Rotation}} = 2 \times acos(Q_{\text{W}})$$

$$D_{\text{QuaternionCheck}} = \sqrt{1 - Q_{\text{W}}^2}$$

$$AA_{\text{AxisX}} = \begin{cases} \frac{Q_{\text{X}}}{D_{\text{QuaternionCheck}}}, & \text{if } D_{\text{QuaternionCheck}} \geq 0.001. \\ 0, & \text{otherwise.} \end{cases}$$
$$AA_{\text{AxisY}} = \begin{cases} \frac{Q_{\text{Y}}}{D_{\text{QuaternionCheck}}}, & \text{if } D_{\text{QuaternionCheck}} \geq 0.001. \\ 0, & \text{otherwise.} \end{cases}$$
$$AA_{\text{AxisZ}} = \begin{cases} \frac{Q_{\text{Z}}}{D_{\text{QuaternionCheck}}}, & \text{if } D_{\text{QuaternionCheck}} \geq 0.001. \\ 0, & \text{otherwise.} \end{cases}$$

# 27 Direction-Angle to Quaternion

$V_{\text{Right}} = Vector(1,0,0)$
$V_{\text{Up}} = Vector(0,1,0)$
$V_{\text{Forward}} = Vector(0,0,1)$

$V_{\text{RotatedUp}} = DA_{\text{DirectionAngleDirection}}$

$Q_{\text{Rotation}} = QuaternionFromDirectionVectors(V_{\text{Up}}, V_{\text{RotatedUp}})$

$V_{\text{RotatedRight}} = RM_{\text{RotationMatrix}}.Rotate(Vector(0, -DA_{\text{DirectionAngleRotation}}, 0), V_{\text{Right}})$
$V_{\text{RotatedForward}} = RM_{\text{RotationMatrix}}.Rotate(Vector(0, -DA_{\text{DirectionAngleRotation}}, 0), V_{\text{Forward}})$

$V_{\text{RotatedRight}} = Q_{\text{Rotation}}.RotateVector(V_{\text{RotatedRight}})$
$V_{\text{RotatedForward}} = Q_{\text{Rotation}}.RotateVector(V_{\text{RotatedForward}})$

$Q_{\text{Quaternion}} = RMToQuaternion(RotationMatrix(V_{\text{RotatedRight}}, V_{\text{RotatedUp}}, V_{\text{RotatedForward}}))$

# 28 Quaternion to Direction-Angle

$V_{\text{Up}} = Vector(0,1,0)$
$V_{\text{Right}} = Vector(1,0,0)$
$V_{\text{RotatedUp}} = Q_{\text{Quaternion}}.RotateVector(V_{\text{Up}})$
$V_{\text{RotatedRight}} = Q_{\text{Quaternion}}.RotateVector(V_{\text{Right}})$

$Q_{\text{Rotation}} = QuaternionFromDirectionVectors(V_{\text{Up}}, V_{\text{RotateUp}})$

$V_{\text{RightCompensated}} = Q_{\text{Rotation}}.UnrotateVector(V_{\text{RotatedRight}})$

$D_{\text{RightAngle}} = \frac{atan2(D_{\text{RightZ}}, D_{\text{RightX}}) \times \pi}{180}$
$D_{\text{RightRotatedAngle}} = \frac{atan2(D_{\text{RightCompensatedZ}}, D_{\text{RightCompensatedX}}) \times \pi}{180}$

$DA_{\text{Rotation}} = D_{\text{RightAngle}} - D_{\text{RightRotateAngle}}$
$DA_{\text{Direction}} = V_{\text{RotatedUp}}$

# 29    Rotation Matrix to Quaternion

$$D_{\text{MatrixTrace}} = V_{\text{XAxisX}} + V_{\text{YAxisY}} + V_{\text{ZAxisZ}}$$

$$D_{\text{Square}} = \begin{cases} \sqrt{1 + MatrixTrace} \times 2, & \text{if } D_{\text{MatrixTrace}} > 0. \\ \sqrt{1 + D_{\text{XAxisX}} - D_{\text{YAxisY}} - D_{\text{ZAxisZ}}} \times 2, & \text{if } D_{\text{XAxisX}} > D_{\text{YAxisY}} \text{ and } D_{\text{XAxisX}} > D_{\text{ZAxisZ}}. \\ \sqrt{1 + D_{\text{YAxisY}} - D_{\text{XAxisX}} - D_{\text{ZAxisZ}}} \times 2, & \text{if } D_{\text{YAxisY}} > D_{\text{ZAxisZ}}. \\ \sqrt{1 + D_{\text{ZAxisZ}} - D_{\text{XAxisX}} - D_{\text{YAxisY}}} \times 2, & \text{otherwise.} \end{cases}$$

$$Q_{\text{W}} = \begin{cases} \frac{D_{\text{Square}}}{4}, & \text{if } D_{\text{MatrixTrace}} > 0. \\ \frac{D_{\text{ZAxisY}} - D_{\text{YAxisZ}}}{D_{\text{Square}}}, & \text{if } D_{\text{XAxisX}} > D_{\text{YAxisY}} \text{ and } D_{\text{XAxisX}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{XAxisZ}} - D_{\text{ZAxisX}}}{D_{\text{Square}}}, & \text{if } D_{\text{YAxisY}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{YAxisX}} - D_{\text{XAxisY}}}{D_{\text{Square}}}, & \text{otherwise.} \end{cases}$$

$$Q_{\text{X}} = \begin{cases} \frac{D_{\text{ZAxisY}} - D_{\text{YAxisZ}}}{D_{\text{Square}}}, & \text{if } D_{\text{MatrixTrace}} > 0. \\ \frac{D_{\text{Square}}}{4}, & \text{if } D_{\text{XAxisX}} > D_{\text{YAxisY}} \text{ and } D_{\text{XAxisX}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{XAxisY}} - D_{\text{YAxisX}}}{D_{\text{Square}}}, & \text{if } D_{\text{YAxisY}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{XAxisZ}} - D_{\text{ZAxisX}}}{D_{\text{Square}}}, & \text{otherwise.} \end{cases}$$

$$Q_Y = \begin{cases} \frac{D_{\text{XAxisZ}} - D_{\text{ZAxisX}}}{D_{\text{Square}}}, & \text{if } D_{\text{MatrixTrace}} > 0. \\ \frac{D_{\text{XAxisY}} - D_{\text{YAxisX}}}{D_{\text{Square}}}, & \text{if } D_{\text{XAxisX}} > D_{\text{YAxisY}} \text{ and } D_{\text{XAxisX}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{Square}}}{4}, & \text{if } D_{\text{YAxisY}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{YAxisZ}} - D_{\text{ZAxisY}}}{D_{\text{Square}}}, & \text{otherwise.} \end{cases}$$

$$Q_Z = \begin{cases} \frac{D_{\text{YAxisZ}} - D_{\text{ZAxisY}}}{D_{\text{Square}}}, & \text{if } D_{\text{MatrixTrace}} > 0. \\ \frac{D_{\text{XAxisZ}} - D_{\text{ZAxisX}}}{D_{\text{Square}}}, & \text{if } D_{\text{XAxisX}} > D_{\text{YAxisY}} \text{ and } D_{\text{XAxisX}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{YAxisZ}} - D_{\text{ZAxisY}}}{D_{\text{Square}}}, & \text{if } D_{\text{YAxisY}} > D_{\text{ZAxisZ}}. \\ \frac{D_{\text{Square}}}{4}, & \text{otherwise.} \end{cases}$$

$$Q_{\text{WXYZ}} = Conjugate(Q_{\text{WXYZ}})$$

# 30    Quaternion to Rotation Matrix

$V_{\text{Right}} = Vector(1, 0, 0)$
$V_{\text{Up}} = Vector(0, 1, 0)$
$V_{\text{Forward}} = Vector(0, 0, 1)$

$RM_{\text{XAxis}} = Q_{\text{Rotation}}.RotateVector(V_{\text{Right}})$
$RM_{\text{YAxis}} = Q_{\text{Rotation}}.RotateVector(V_{\text{Up}})$
$RM_{\text{ZAxis}} = Q_{\text{Rotation}}.RotateVector(V_{\text{Forward}})$

# 31    Direction Vectors to Quaternion

$V_{\text{XAxis}} = Vector(1, 0, 0)$
$V_{\text{YAxis}} = Vector(0, 1, 0)$
$D_{\text{Dot}} = V_{\text{Initial}} \dot{V}_{\text{Final}}$

$$C_{\text{Cross}} = \begin{cases} V_{\text{XAxis}} \times V_{\text{Initial}}, & \text{if } D_{\text{Dot}} < -0.999. \\ Vector(0, 0, 0), & \text{if } D_{\text{Dot}} > 0.999. \\ V_{\text{Initial}} \times V_{\text{Final}}, & \text{otherwise.} \end{cases}$$

$$C_{\text{Cross}} = \begin{cases} V_{\text{YAxis}} \times V_{\text{Initial}}, & \text{if } D_{\text{CrossLength}} < 0.001. \end{cases}$$

$$AA_{\text{AxisAngle}} = AxisAngle(\pi, V_{\text{Cross}})$$

$$Q_W = \begin{cases} D_{\text{AxisAngleRotation}}, & \text{if } D_{\text{Dot}} < -0.999. \\ 1, & \text{if } D_{\text{Dot}} > 0.999. \\ 1 + D_{\text{Dot}}, & \text{otherwise.} \end{cases}$$

$$Q_{\mathrm{X}} = \begin{cases} D_{\mathrm{AxisAngleAxisX}}, & \text{if } D_{\mathrm{Dot}} < -0.999. \\ 0, & \text{if } D_{\mathrm{Dot}} > 0.999. \\ D_{\mathrm{CrossX}}, & \text{otherwise.} \end{cases}$$

$$Q_{\mathrm{Y}} = \begin{cases} D_{\mathrm{AxisAngleAxisY}}, & \text{if } D_{\mathrm{Dot}} < -0.999. \\ 0, & \text{if } D_{\mathrm{Dot}} > 0.999. \\ D_{\mathrm{CrossY}}, & \text{otherwise.} \end{cases}$$

$$Q_{\mathrm{Z}} = \begin{cases} D_{\mathrm{AxisAngleAxisZ}}, & \text{if } D_{\mathrm{Dot}} < -0.999. \\ 0, & \text{if } D_{\mathrm{Dot}} > 0.999. \\ D_{\mathrm{CrossZ}}, & \text{otherwise.} \end{cases}$$

# 32 Euler Angles to Quaternion

$D_{\mathrm{Rotating}} = D_{\mathrm{EulerX}}$

$$D_{\mathrm{EulerX}} = \begin{cases} D_{\mathrm{EulerZ}}, & \text{if } F_{\mathrm{Frame}} = F_{\mathrm{Rotating}} \\ D_{\mathrm{EulerX}}, & \text{otherwise.} \end{cases}$$

$$D_{\mathrm{EulerZ}} = \begin{cases} D_{\mathrm{Rotating}}, & \text{if } F_{\mathrm{Frame}} = F_{\mathrm{Rotating}} \\ D_{\mathrm{EulerZ}}, & \text{otherwise.} \end{cases}$$

$$D_{\mathrm{EulerY}} = \begin{cases} -D_{\mathrm{EulerY}}, & \text{if } P_{\mathrm{AxisPermutation}} = P_{\mathrm{Odd}} \\ D_{\mathrm{EulerY}}, & \text{otherwise.} \end{cases}$$

$D_{\mathrm{SineX}} = sin(\frac{D_{\mathrm{EulerX}}}{2})$
$D_{\mathrm{SineY}} = sin(\frac{D_{\mathrm{EulerY}}}{2})$
$D_{\mathrm{SineZ}} = sin(\frac{D_{\mathrm{EulerZ}}}{2})$

$D_{\mathrm{CosineX}} = cos(\frac{D_{\mathrm{EulerX}}}{2})$

$$D_{\text{CosineY}} = cos\frac{D_{\text{EulerY}}}{2})$$
$$D_{\text{CosineZ}} = cos\frac{D_{\text{EulerZ}}}{2})$$

$$D_{\text{CC}} = D_{\text{CosineX}} \times D_{\text{CosineZ}}$$
$$D_{\text{CS}} = D_{\text{CosineX}} \times D_{\text{SineZ}}$$
$$D_{\text{SC}} = D_{\text{SineX}} \times D_{\text{CosineZ}}$$
$$D_{\text{SS}} = D_{\text{SineX}} \times D_{\text{SineZ}}$$

$$Q_{\text{W}} = \begin{cases} D_{\text{CosineY}} \times (D_{\text{CC}} - D_{\text{SS}}), & \text{if } A_{\text{AxisRepetition}} = A_{\text{Yes}} \\ D_{\text{CosineY}} \times D_{\text{CC}} + D_{\text{SineY}} \times D_{\text{SS}}, & \text{otherwise.} \end{cases}$$

$$Q_{\text{X}} = \begin{cases} D_{\text{CosineY}} \times (D_{\text{CS}} + D_{\text{SC}}), & \text{if } A_{\text{AxisRepetition}} = A_{\text{Yes}} \\ D_{\text{CosineY}} \times D_{\text{SC}} - D_{\text{SineY}} \times D_{\text{CS}}, & \text{otherwise.} \end{cases}$$

$$Q_{\text{Y}} = \begin{cases} D_{\text{SineY}} \times (D_{\text{CC}} + D_{\text{SS}}), & \text{if } A_{\text{AxisRepetition}} = A_{\text{Yes}} \\ D_{\text{CosineY}} \times D_{\text{SS}} + D_{\text{SineY}} \times D_{\text{CC}}, & \text{otherwise.} \end{cases}$$

$$Q_{\text{Z}} = \begin{cases} D_{\text{SineY}} \times (D_{\text{CS}} - D_{\text{SC}}), & \text{if } A_{\text{AxisRepetition}} = A_{\text{Yes}} \\ D_{\text{CosineY}} \times D_{\text{CS}} - D_{\text{SineY}} \times D_{\text{SC}}, & \text{otherwise.} \end{cases}$$

$$Q_{\text{WXYZ}} = Q_{\text{WXYZ}}.Permutate(V_{\text{Permutation}})$$

$$Q_{\text{Y}} = \begin{cases} -Q_{\text{Y}}, & \text{if } P_{\text{AxisPermutation}} = P_{\text{Odd}} \\ Q_{\text{Y}}, & \text{otherwise.} \end{cases}$$

# 33 Quaternion to Euler Angles

$D_{\text{Norm}} = D_{\text{QuaternionNormal}}$

$D_{\text{Scale}} = \begin{cases} \frac{2}{D_{\text{Norm}}}, & \text{if } D_{\text{Norm}} > 0 \\ 0, & \text{otherwise.} \end{cases}$

$V_{\text{SB}} = Vector(Q_{\text{X}} \times D_{\text{Scale}}, Q_{\text{Y}} \times D_{\text{Scale}}, Q_{\text{Z}} \times D_{\text{Scale}})$
$V_{\text{W}} = Vector(Q_{\text{W}} \times SB_{\text{X}}, Q_{\text{W}} \times SB_{\text{Y}}, Q_{\text{W}} \times SB_{\text{Z}})$
$V_{\text{X}} = Vector(Q_{\text{X}} \times SB_{\text{X}}, Q_{\text{X}} \times SB_{\text{Y}}, Q_{\text{X}} \times SB_{\text{Z}})$
$V_{\text{Y}} = Vector(0, Q_{\text{Y}} \times SB_{\text{Y}}, Q_{\text{Y}} \times SB_{\text{Z}})$
$V_{\text{Z}} = Vector(0, 0, Q_{\text{Z}} \times SB_{\text{Z}})$

$V_{\text{XAxis}} = Vector(1 - D_{\text{YY}} + D_{\text{ZZ}}, D_{\text{XY}} - D_{\text{WZ}}, D_{\text{XZ}} + D_{\text{WY}})$
$V_{\text{YAxis}} = Vector(D_{\text{XY}} + D_{\text{WZ}}, 1 - D_{\text{XX}} - D_{\text{ZZ}}, D_{\text{YZ}} - D_{\text{WX}})$
$V_{\text{ZAxis}} = Vector(D_{\text{XZ}} - D_{\text{WY}}, D_{\text{YZ}} - D_{\text{WX}}, 1 - D_{\text{XX}} + D_{\text{YY}})$

$RM_{\text{RotationMatrix}} = RotationMatrix(V_{\text{XAxis}}, V_{\text{YAxis}}, V_{\text{ZAxis}})$

$D_{\text{SineY}} = \sqrt{(D_{\text{XY}}^2) + (D_{\text{XZ}}^2)}$
$D_{\text{CosineY}} = \sqrt{(D_{\text{XX}}^2) + (D_{\text{YX}}^2)}$

$D_{\text{EAX}} = \begin{cases} atan2(D_{\text{XY}}, D_{\text{XZ}}), & \text{if Initial Axis is repeated and} D_{\text{SineY}} > 32 \times D_{\text{Epsilon}} \\ atan2(-D_{\text{YZ}}, D_{\text{YY}}), & \text{if Initial Axis is repeated and} D_{\text{SineY}} \leq 32 \times D_{\text{Epsilon}} \\ atan2(D_{\text{ZY}}, D_{\text{ZZ}}), & \text{if Initial Axis is not repeated and} D_{\text{SineY}} > 32 \times D_{\text{Epsilon}} \\ atan2(-D_{\text{YZ}}, D_{\text{YY}}), & \text{if Initial Axis is not repeated and} D_{\text{SineY}} \leq 32 \times D_{\text{Epsilon}} \end{cases}$

$D_{\text{EAY}} = \begin{cases} atan2(D_{\text{SineY}}, D_{\text{XX}}), & \text{if Initial Axis is repeated} \\ atan2(-D_{\text{ZX}}, D_{\text{CosineY}}), & \text{if Initial Axis is not repeated} \end{cases}$

$D_{\text{EAZ}} = \begin{cases} atan2(D_{\text{YX}}, -D_{\text{ZX}}), & \text{if Initial Axis is repeated and} D_{\text{SineY}} > 32 \times D_{\text{Epsilon}} \\ 0, & \text{if Initial Axis is repeated and} D_{\text{SineY}} \leq 32 \times D_{\text{Epsilon}} \\ atan2(D_{\text{YX}}, D_{\text{XX}}), & \text{if Initial Axis is not repeated and} D_{\text{SineY}} > 32 \times D_{\text{Epsilon}} \\ 0, & \text{if Initial Axis is not repeated and} D_{\text{SineY}} \leq 32 \times D_{\text{Epsilon}} \end{cases}$

$D_{\text{EAX}} = \begin{cases} -D_{\text{EAX}}, & \text{if Axis Permutation is Odd} \end{cases}$

$D_{\text{EAY}} = \begin{cases} -D_{\text{EAY}}, & \text{if Axis Permutation is Odd} \end{cases}$

$D_{\text{EAZ}} = \begin{cases} -D_{\text{EAZ}}, & \text{if Axis Permutation is Odd} \end{cases}$

$D_{\text{TempX}} = D_{\text{EAX}}$
$D_{\text{EAX}} = \begin{cases} D_{\text{EAZ}}, & \text{if Frame Taken is Rotating} \end{cases}$
$D_{\text{EAZ}} = \begin{cases} D_{\text{TempX}}, & \text{if Frame Taken is Rotating} \end{cases}$