LVLH Transformations

Jacob Williams

April 19, 2014

Abstract

This technical note derives the equations for transformation of state vectors to and from a Local Vertical Local Horizontal (LVLH) frame. An implementation of the algorithms in Fortran is also included.

1 Preliminaries

The time derivative of a unit vector $\hat{\mathbf{u}}$ is given by:

$$\frac{d}{dt}(\hat{\mathbf{u}}) = \frac{d}{dt}\left(\frac{\mathbf{u}}{u}\right) \tag{1}$$

$$= \frac{\dot{\mathbf{u}}}{u} - \frac{\dot{u}}{u^2}\mathbf{u} \tag{2}$$

Noting that the dot product of a vector and its derivative is given by [1]:

$$\mathbf{u} \cdot \dot{\mathbf{u}} = u\dot{u} \tag{3}$$

we can combine Equations 2 and 3, eliminate $\dot{u}$, and produce the equation:

$$\frac{d}{dt}(\hat{\mathbf{u}}) = \frac{1}{u}\left[\dot{\mathbf{u}} - (\hat{\mathbf{u}} \cdot \dot{\mathbf{u}})\,\hat{\mathbf{u}}\right] \tag{4}$$

2 State Transformation Equations

Refer to Figure 1. We define two vehicles, the *Target* and the *Chaser*. A rotating frame is defined at the Target vehicle using that vehicle's state. A base inertial frame is centered at the central body. First, define the following two relative position and velocity

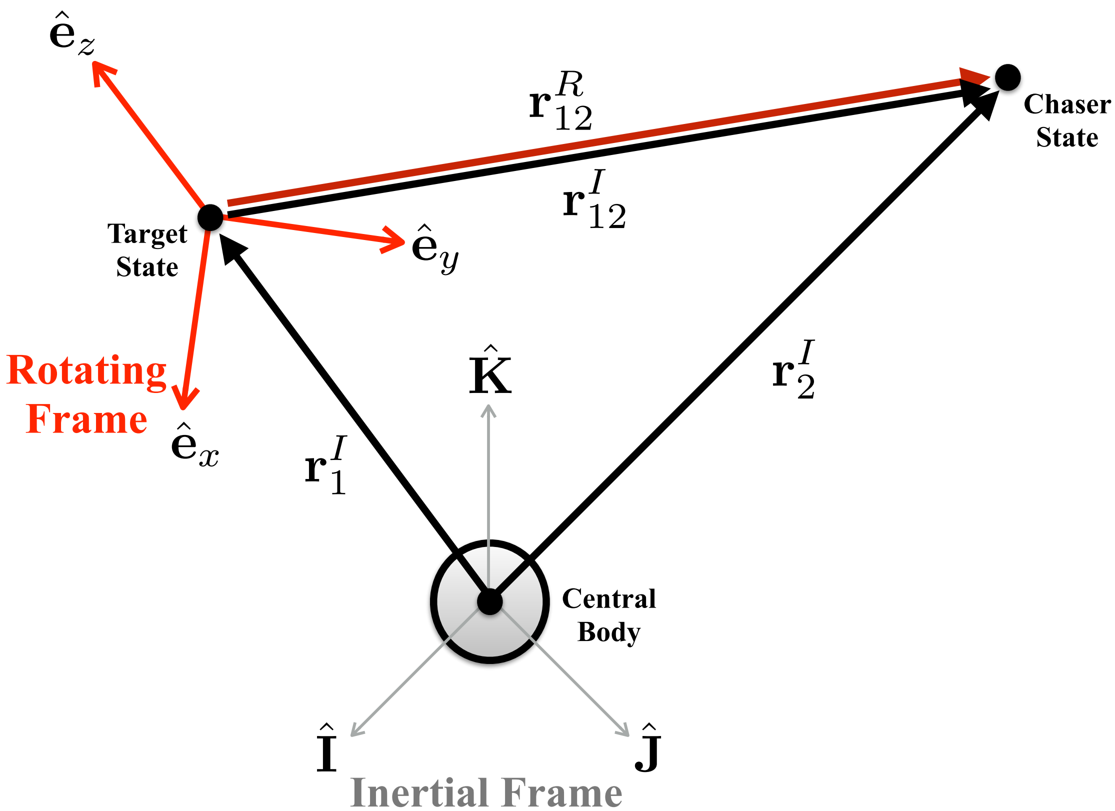

Figure 1: Vector Diagram: Target and Chaser Vehicle. Vectors with the superscript I are expressed in the inertial frame relative to the central body, and vectors with the superscript R are expressed in the rotating frame relative to the target vehicle.

vectors:

$$\mathbf{r}_{12}^I = \mathbf{r}_2^I - \mathbf{r}_1^I \tag{5}$$
$$\dot{\mathbf{r}}_{12}^I = \dot{\mathbf{r}}_2^I - \dot{\mathbf{r}}_1^I \tag{6}$$

Vectors with the superscript I are expressed in the inertial frame relative to the central body, and vectors with the superscript R are expressed in the rotating frame relative to the target vehicle. Transformations from the inertial frame to the rotating frame for position and velocity are given by:

$$\mathbf{r}_{12}^R = [\mathbf{C}]\mathbf{r}_{12}^I \tag{7}$$
$$\dot{\mathbf{r}}_{12}^R = [\dot{\mathbf{C}}]\mathbf{r}_{12}^I + [\mathbf{C}]\dot{\mathbf{r}}_{12}^I \tag{8}$$

The matrix $[\mathbf{C}]$ is the 3x3 rotation matrix from the inertial to the rotating frame, and $[\dot{\mathbf{C}}]$ is the derivative of this matrix. Equation 8 is simply the time derivative of Equation 7. The inverse transformation (from the rotating frame to the inertial frame), is obtained by pre-multiplying Equation 7 by $[\mathbf{C}]^T$ (which is equivalent to $[\mathbf{C}]^{-1}$) and differentiating:

$$\mathbf{r}_{12}^I = [\mathbf{C}]^T\mathbf{r}_{12}^R \tag{9}$$
$$\dot{\mathbf{r}}_{12}^I = [\dot{\mathbf{C}}]^T\mathbf{r}_{12}^R + [\mathbf{C}]^T\dot{\mathbf{r}}_{12}^R \tag{10}$$

3 Derivation of $[\mathbf{C}]$ and $[\dot{\mathbf{C}}]$

The three basis vectors that define the rotating frame are [$\hat{\mathbf{e}}_x$, $\hat{\mathbf{e}}_y$, and $\hat{\mathbf{e}}_z$]. Where:

$$\hat{\mathbf{e}}_x = \hat{e}_{x_i}\hat{\mathbf{I}} + \hat{e}_{x_j}\hat{\mathbf{J}} + \hat{e}_{x_k}\hat{\mathbf{K}} \tag{11}$$
$$\hat{\mathbf{e}}_y = \hat{e}_{y_i}\hat{\mathbf{I}} + \hat{e}_{y_j}\hat{\mathbf{J}} + \hat{e}_{y_k}\hat{\mathbf{K}} \tag{12}$$
$$\hat{\mathbf{e}}_z = \hat{e}_{z_i}\hat{\mathbf{I}} + \hat{e}_{z_j}\hat{\mathbf{J}} + \hat{e}_{z_k}\hat{\mathbf{K}} \tag{13}$$

The transformation matrix $[\mathbf{C}]$ is thus [2]:

$$[\mathbf{C}] = \begin{bmatrix} (\hat{\mathbf{e}}_x \cdot \hat{\mathbf{I}}) & (\hat{\mathbf{e}}_x \cdot \hat{\mathbf{J}}) & (\hat{\mathbf{e}}_x \cdot \hat{\mathbf{K}}) \\ (\hat{\mathbf{e}}_y \cdot \hat{\mathbf{I}}) & (\hat{\mathbf{e}}_y \cdot \hat{\mathbf{J}}) & (\hat{\mathbf{e}}_y \cdot \hat{\mathbf{K}}) \\ (\hat{\mathbf{e}}_z \cdot \hat{\mathbf{I}}) & (\hat{\mathbf{e}}_z \cdot \hat{\mathbf{J}}) & (\hat{\mathbf{e}}_z \cdot \hat{\mathbf{K}}) \end{bmatrix} = \begin{bmatrix} \hat{e}_{x_i} & \hat{e}_{x_j} & \hat{e}_{x_k} \\ \hat{e}_{y_i} & \hat{e}_{y_j} & \hat{e}_{y_k} \\ \hat{e}_{z_i} & \hat{e}_{z_j} & \hat{e}_{z_k} \end{bmatrix} \tag{14}$$

And the transformation matrix derivative is:

$$[\dot{\mathbf{C}}] = \frac{d}{dt}[\mathbf{C}] \tag{15}$$

which requires the time derivatives of the three basis vectors.

For convenience, we will drop the superscript and subscripts for the state variables of the target vehicle in the inertial frame, and define:

$$\mathbf{r} \equiv \mathbf{r}_1^I \tag{16}$$

$$\mathbf{v} = \dot{\mathbf{r}} \equiv \mathbf{v}_1^I \tag{17}$$

$$\mathbf{a} = \ddot{\mathbf{r}} = \mathbf{f}(t, \mathbf{r}, \dot{\mathbf{r}}) \tag{18}$$

$$\mathbf{h} = \mathbf{r}_1^I \times \mathbf{v}_1^I \tag{19}$$

where $\mathbf{r}$ is the target position vector, $\mathbf{v}$ is the target velocity vector, $\mathbf{a}$ is the target acceleration vector, and $\mathbf{h}$ is the target specific angular momentum. All are expressed in the inertial frame relative to the central body (as shown in Figure 1).

For the LVLH frame [3], the basis vector definitions are:

$$\hat{\mathbf{e}}_z = -\hat{\mathbf{r}} \tag{20}$$

$$\hat{\mathbf{e}}_y = -\hat{\mathbf{h}} \tag{21}$$

$$\hat{\mathbf{e}}_x = \hat{\mathbf{e}}_z \times \hat{\mathbf{e}}_y \tag{22}$$

Using Equation 4, the derivative of the $\hat{\mathbf{e}}_z$ vector is:

$$\frac{d}{dt}(\hat{\mathbf{e}}_z) = \frac{d}{dt}(-\hat{\mathbf{r}}) \tag{23}$$

$$= -\frac{1}{r}\left[\mathbf{v} - (\hat{\mathbf{r}} \cdot \mathbf{v})\,\hat{\mathbf{r}}\right] \tag{24}$$

Again using Equation 4, the derivative of the $\hat{\mathbf{e}}_y$ vector is:

$$\frac{d}{dt}(\hat{\mathbf{e}}_y) = \frac{d}{dt}\left(-\hat{\mathbf{h}}\right) \tag{25}$$

$$= -\frac{1}{h}\left[\dot{\mathbf{h}} - \left(\hat{\mathbf{h}} \cdot \dot{\mathbf{h}}\right)\hat{\mathbf{h}}\right] \tag{26}$$

Where:

$$\dot{\mathbf{h}} = \frac{d}{dt}(\mathbf{r} \times \mathbf{v}) \tag{27}$$

$$= \left[\frac{d}{dt}(\mathbf{r}) \times \mathbf{v}\right] + \left[\mathbf{r} \times \frac{d}{dt}(\mathbf{v})\right] \tag{28}$$

$$= \mathbf{r} \times \mathbf{a} \tag{29}$$

For a force field where $\mathbf{r} \| \mathbf{a}$ (e.g., two-body motion), $\dot{\mathbf{h}} = \mathbf{0}$, so Equation 26 reduces to:

$$\frac{d}{dt}(\hat{\mathbf{e}}_y) = \mathbf{0} \tag{30}$$

Finally, the derivative of the $\hat{\mathbf{e}}_x$ vector is:

$$\frac{d}{dt}(\hat{\mathbf{e}}_x) = \frac{d}{dt}(\hat{\mathbf{e}}_y \times \hat{\mathbf{e}}_z) \tag{31}$$

$$= \left[\frac{d}{dt}(\hat{\mathbf{e}}_y) \times \hat{\mathbf{e}}_z\right] + \left[\hat{\mathbf{e}}_y \times \frac{d}{dt}(\hat{\mathbf{e}}_z)\right] \tag{32}$$

Which contains already-derived quantities. For two-body motion, Equation 32 reduces to:

$$\frac{d}{dt}(\hat{\mathbf{e}}_x) = -\hat{\mathbf{h}} \times \frac{d}{dt}(\hat{\mathbf{e}}_z) \tag{33}$$

4 Test Case

Assume two-body motion. Given the target and chaser inertial position and velocity vectors:

$$\begin{array}{lcllll}
\mathbf{r}_1^I & = & [& -2301672.24489839, & -5371076.10250925, & -3421146.71530212 &] \text{ m} \\
\mathbf{v}_1^I & = & [& 6133.8624555516, & 306.265184163608, & -4597.13439017524 &] \text{ m/s} \\
\mathbf{r}_2^I & = & [& -2255213.51862763, & -5366553.94133467, & -3453871.15040494 &] \text{ m} \\
\mathbf{v}_2^I & = & [& 6156.89588163809, & 356.79933181917, & -4565.88915429063 &] \text{ m/s}
\end{array}$$

The chaser relative state in the LVLH frame centered on the target is:

$$\begin{array}{lcllll}
\mathbf{r}_{12}^{LVLH} & = & [& 56935.52933486611, & 38.16029598938621, & 2845.326754409645 &] \text{ m} \\
\mathbf{v}_{12}^{LVLH} & = & [& 4.890395234717321, & -0.09759947085768417, & -0.8044815052666578 &] \text{ m/s}
\end{array}$$

5 Program Listing

Fortran implementations of the algorithms in this paper (using Fortran 2003/2008 standards) [4] are presented in the `lvlh_module` below. The main transformation subroutines are `from_ijk_to_lvlh` and `from_lvlh_to_ijk`. Note that the acceleration vector **a** is an optional argument to these routines. If it is not present, then a radial force vector is assumed (the simplifications given in Equations 30 and 33). Three supporting routines (`cross`, `unit`, and `uhat_dot`) are also included. The `test_case` routine shows how to compute the results shown in Section 4. All the code is fairly straightforward, and could be easily converted to other languages such as Matlab.

```fortran
!*****************************************************************
 module lvlh_module
!*****************************************************************

 implicit none

 private

 !public routines:
 public :: from_ijk_to_lvlh, from_lvlh_to_ijk
 public :: test_case

 !working precision (double reals):
 integer,parameter :: wp = selected_real_kind(15, 307)

 !constants:
 real(wp),parameter :: zero = 0.0_wp

 contains
!*****************************************************************

!**********************************************
 pure function cross(r,v) result(rxv)
!**********************************************
! Vector cross product
 implicit none

 real(wp),dimension(3),intent(in) :: r    !vector r
 real(wp),dimension(3),intent(in) :: v    !vector v
 real(wp),dimension(3)            :: rxv  !cross product r×v

 rxv(1) = r(2)*v(3)-v(2)*r(3)
 rxv(2) = r(3)*v(1)-v(3)*r(1)
 rxv(3) = r(1)*v(2)-v(1)*r(2)

!**********************************************
 end function cross
!**********************************************

```

```fortran
!**********************************************
 pure function unit(u) result(u_hat)
!**********************************************
! Unit vector
 implicit none

 real(wp),dimension(3), intent(in) :: u      !vector u
 real(wp),dimension(3)             :: u_hat  !unit vector û

 real(wp) :: umag  !vector magnitude u

 umag = norm2(u)
 if (umag == zero) then
   u_hat = zero ! error
 else
   u_hat = u / umag
 end if

!**********************************************
 end function unit
!**********************************************
!**********************************************
 pure function uhat_dot(u,udot) result(uhatd)
!**********************************************
! Time Derivative of a Unit Vector
 implicit none

 real(wp),dimension(3),intent(in) :: u      !vector u
 real(wp),dimension(3),intent(in) :: udot   !u̇
 real(wp),dimension(3)            :: uhatd  !d(û)/dt

 real(wp)              :: umag  !vector magnitude u
 real(wp),dimension(3) :: uhat  !unit vector û

 umag = norm2(u)

 if (umag == zero) then  !singularity
  uhatd = zero
 else
  uhat = u / umag
  uhatd = (udot-dot_product(uhat,udot)*uhat)/umag
 end if

!**********************************************
 end function uhat_dot
!**********************************************
```

```
!********************************************
 pure subroutine from_ijk_to_lvlh(r,v,a,c,cdot)
!********************************************
! if a is not present, a radial acceleration is assumed

 implicit none

 real(wp),dimension(3),intent(in)           :: r      !r vec.
 real(wp),dimension(3),intent(in)           :: v      !v vec.
 real(wp),dimension(3),intent(in),optional  :: a      !a vec.
 real(wp),dimension(3,3),intent(out)        :: c      ![C]
 real(wp),dimension(3,3),intent(out)        :: cdot   ![Ċ]

 real(wp),dimension(3) :: ex_hat,ex_hat_dot
 real(wp),dimension(3) :: ey_hat,ey_hat_dot
 real(wp),dimension(3) :: ez_hat,ez_hat_dot
 real(wp),dimension(3) :: h,h_hat,h_dot

 h           = cross(r,v)
 h_hat       = unit(h)
 ez_hat      = -unit(r)
 ey_hat      = -h_hat
 ex_hat      = cross(ey_hat,ez_hat)

 ez_hat_dot = -uhat_dot(r,v)
 if (present(a)) then
   h_dot      = cross(r,a)
   ey_hat_dot = -uhat_dot(h, h_dot)
   ex_hat_dot = cross(ey_hat_dot,ez_hat) + &
                cross(ey_hat,ez_hat_dot)
  else !assume no external torque
   ey_hat_dot = zero
   ex_hat_dot = cross(ey_hat,ez_hat_dot)
 end if

 c(1,:)     = ex_hat
 c(2,:)     = ey_hat
 c(3,:)     = ez_hat
 cdot(1,:)  = ex_hat_dot
 cdot(2,:)  = ey_hat_dot
 cdot(3,:)  = ez_hat_dot

!********************************************
 end subroutine from_ijk_to_lvlh
!********************************************
```

```fortran
!*********************************************
 pure subroutine from_lvlh_to_ijk(r,v,a,c,cdot)
!*********************************************
! if a is not present, a radial acceleration is assumed

 implicit none

 real(wp),dimension(3),intent(in)           :: r     !r vec.
 real(wp),dimension(3),intent(in)           :: v     !v vec.
 real(wp),dimension(3),intent(in),optional :: a     !a vec.
 real(wp),dimension(3,3),intent(out)        :: c     ![C]
 real(wp),dimension(3,3),intent(out)        :: cdot  ![Ċ]

 call from_ijk_to_lvlh(r,v,a,c,cdot)

 !matrices for reverse transformation:
 c    = transpose(c)
 cdot = transpose(cdot)

!*********************************************
 end subroutine from_lvlh_to_ijk
!*********************************************
```

```fortran
!*********************************************
 subroutine test_case()
!*********************************************

 implicit none

 real(wp),dimension(6),parameter :: target_rv = &
  [-2301672.24489839_wp, &
   -5371076.10250925_wp, &
   -3421146.71530212_wp ,&
   6133.8624555516_wp,&
   306.265184163608_wp,&
   -4597.13439017524_wp ]

 real(wp),dimension(6),parameter :: chaser_rv = &
  [-2255213.51862763_wp,&
   -5366553.94133467_wp,&
   -3453871.15040494_wp,&
   6156.89588163809_wp,&
   356.79933181917_wp,&
   -4565.88915429063_wp ]

 real(wp),dimension(3) :: r_12_I, r1_I, r2_I
 real(wp),dimension(3) :: v_12_I, v1_I, v2_I
 real(wp),dimension(3) :: r_12_R, v_12_R
 real(wp),dimension(3,3) :: c,cdot

 r1_I = target_rv(1:3)     !see Figure 1 in paper
 v1_I = target_rv(4:6)
 r2_I = chaser_rv(1:3)
 v2_I = chaser_rv(4:6)

 r_12_I = r2_I - r1_I      !equations 5,6 in paper
 v_12_I = v2_I - v1_I

 call from_ijk_to_lvlh(r1_I,v1_I,c=c,cdot=cdot)   !compute [C] and [Ċ]

 r_12_R = matmul( c, r_12_I)    !equations 7,8 in paper
 v_12_R = matmul( cdot, r_12_I) + matmul( c, v_12_I)

 write(*,'(A,1x,*(E30.16,1X))') 'r_12_R:', r_12_R
 write(*,'(A,1x,*(E30.16,1X))') 'v_12_R:', v_12_R

!*********************************************
 end subroutine test_case
!*********************************************
```

```
!*******************************************************************
 end module lvlh_module
!*******************************************************************
```

References

[1] P. R. Escobal, *Methods of Orbit Determination*. Krieger Publishing Company, 1976. (Equation 3.131).

[2] D. Vallado and W. McClain, *Fundamentals of Astrodynamics and Applications*. Microcosm Press, 2001.

[3] K. H. Bhavnani and R. P. Vancour, “Coordinate systems for space and geophysical applications,” tech. rep., RADEX, Inc., December 1991.

[4] M. Metcalf, J. Reid, and M. Cohen, *Modern Fortran Explained*. Numerical Mathematics and Scientific Computation, Oxford: Oxford Univ. Press, 2011.