**RS-WISP-04-01: Two Impuse Rendezvous**

**WorkerInSpace**

**Hongseok Kim**

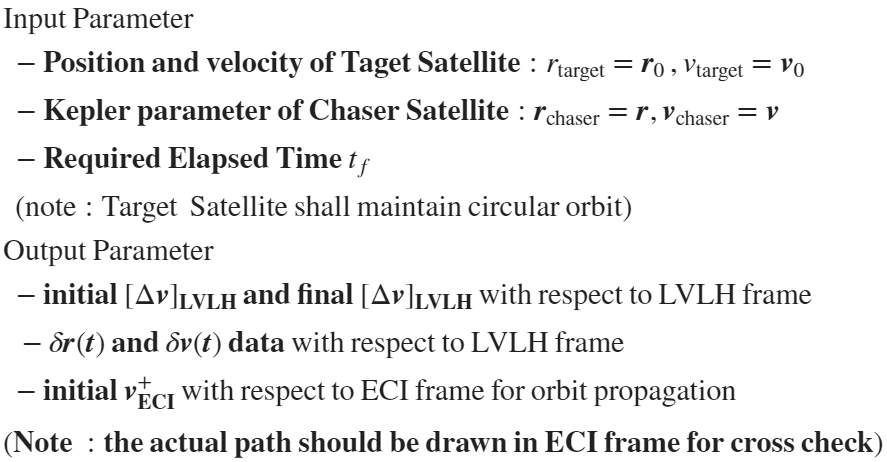
**2/5/2025**

# I. Scope

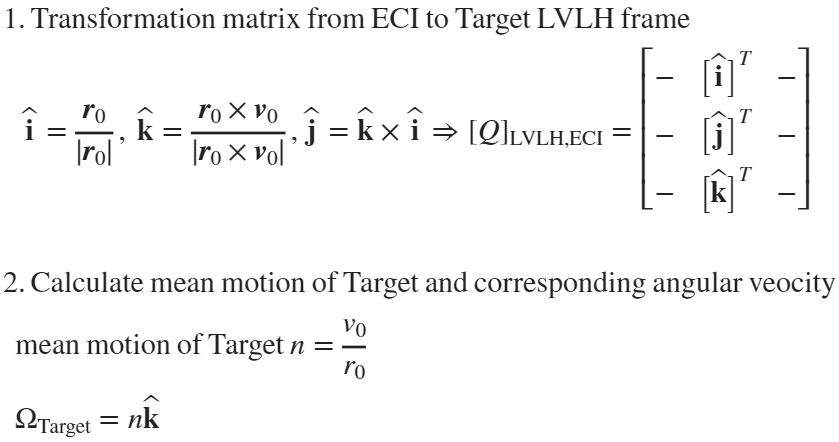
In RS-WISP-02, we have demonstrated the C-W equation for Two-Impuse Rendezvous. For defining operation modes, we have to define multiple functions for these delta-v maneuvers. Furthermore, we should structurize the process of rendezvous process in multiple block diagrams. In this document, the role of Two-Impuse Rendezvous and I/O structure is defined, and corresponding function is described. From these function, we can proceed the simulation in multiple environments and multiple simulation setups.

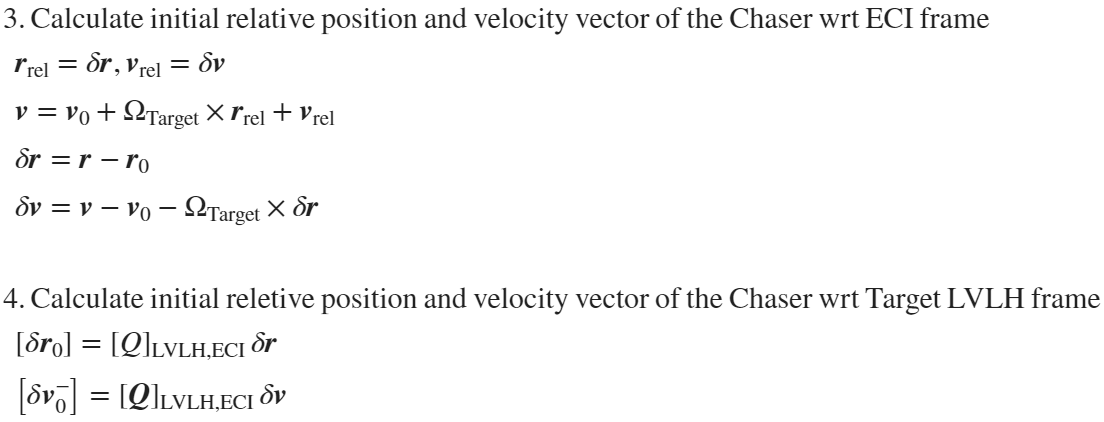
# II. Two Impulse Rendezvous Procedure

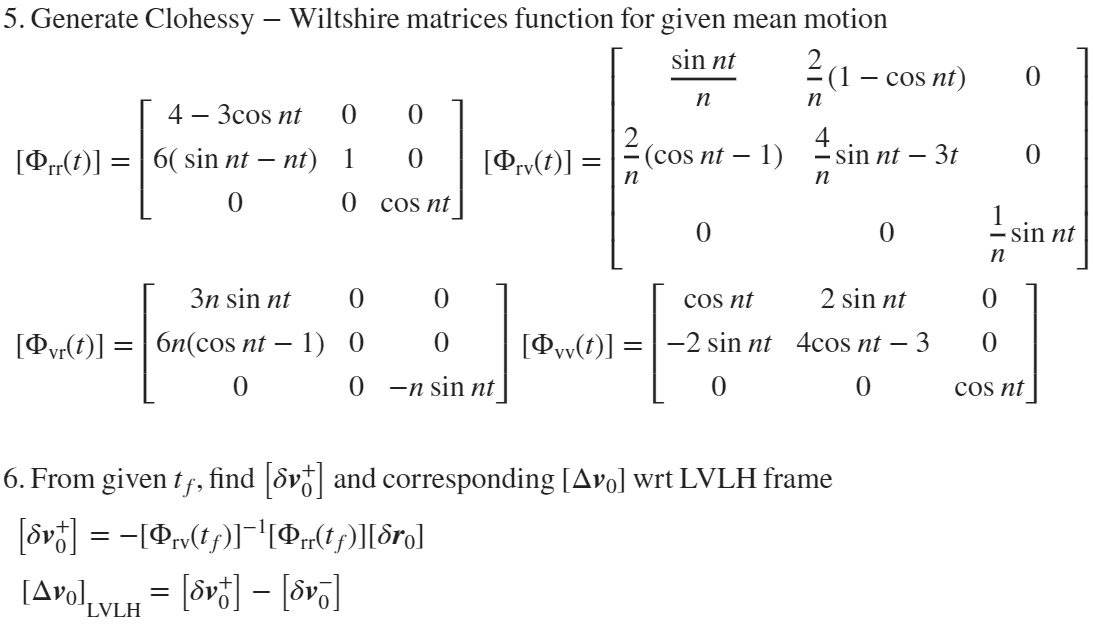
## II.1 I/O Structure

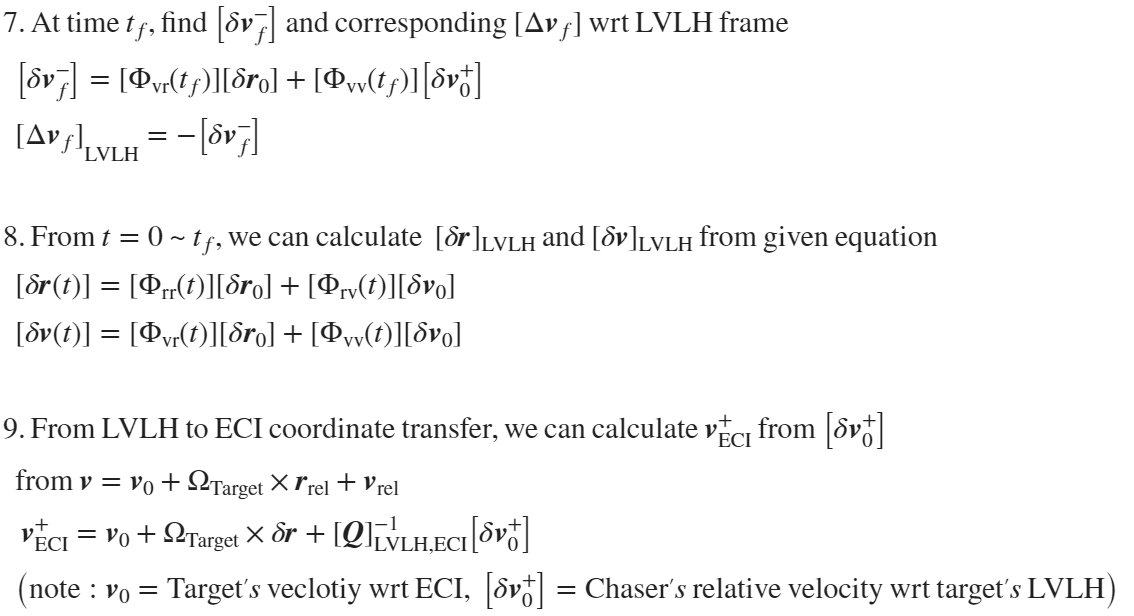


## II.2 Two Impulse Rendezvous Algorithm









# III. Matlab function demonstration

function [Delta\_v\_0, Delta\_v\_f, delta\_r\_t\_mat, delta\_v\_t\_mat, v\_plus\_ECI] = two\_impulse\_rendezvous(r\_target, v\_target, r\_chaser, v\_chaser, tf)

r\_0 = r\_target;

v\_0 = v\_target;

r = r\_chaser;

v = v\_chaser;

% Reference Frame of the Target

i\_hat = r\_0/norm(r\_0);

j\_hat = v\_0/norm(v\_0);

k\_hat = cross(i\_hat, j\_hat);

% Transformation Matrix from ECI to Space Station Frame

Q\_LVLH\_ECI = [i\_hat';j\_hat';k\_hat'];

% Position vector of the spacecraft relative to the space station (ECI)

delta\_r = r - r\_0;

n = norm(v\_0)/norm(r\_0);

Omega\_target = n \* k\_hat;

delta\_v = v - v\_0 - cross(Omega\_target, delta\_r);

% Relative position vector at the beginning of the rendezvous maneuver

delta\_r\_0 = Q\_LVLH\_ECI \* delta\_r;

% Relative velocity just before launch into the rendezvous trajectory is

delta\_v\_0\_minus = Q\_LVLH\_ECI \* delta\_v;

% Calculate Clohessy-Whiltshire matrix for t = t\_f = 28800s and n

nt = n\*tf;

Phi\_rr\_tf = [4 - 3\*cos(nt), 0, 0;

6\*(sin(nt) - nt),1,0;

0,0, cos(nt)];

Phi\_rv\_tf = [sin(nt)/n, 2/n \* (1 - cos(nt)), 0;

2/n\*(cos(nt)-1), 4/n\*sin(nt)-3\*tf,0;

0,0,1/n\*sin(nt)];

Phi\_vr\_tf = [3\*n\*sin(nt),0,0;

6\*n\*(cos(nt)-1),0,0;

0,0,-n\*sin(nt)];

Phi\_vv\_tf = [cos(nt), 2\*sin(nt),0

-2\*sin(nt), 4\*cos(nt)-3, 0;

0,0,cos(nt)];

delta\_v\_0\_plus = -inv(Phi\_rv\_tf) \* Phi\_rr\_tf \* delta\_r\_0;

delta\_v\_f\_minus = Phi\_vr\_tf \* delta\_r\_0 + Phi\_vv\_tf \* delta\_v\_0\_plus;

% Delta-v at the beginning of the rendezvous maneuver

Delta\_v\_0 = delta\_v\_0\_plus - delta\_v\_0\_minus;

% Delta-v at the conclusion of the maneuver

Delta\_v\_f = [0;0;0] - delta\_v\_f\_minus;

% Drawing Reference Chaser Trajector wrt Target

t\_vector = linspace(0,tf);

delta\_r\_t\_mat = zeros(length(t\_vector),3);

delta\_v\_t\_mat = zeros(length(t\_vector),3);

for timestep = 1:length(t\_vector)

t = t\_vector(timestep);

nt = n\*t;

Phi\_rr\_t = [4 - 3\*cos(nt), 0, 0;

6\*(sin(nt) - nt),1,0;

0,0, cos(nt)];

Phi\_rv\_t = [sin(nt)/n, 2/n \* (1 - cos(nt)), 0;

2/n\*(cos(nt)-1), 4/n\*sin(nt)-3\*t,0;

0,0,1/n\*sin(nt)];

Phi\_vr\_t = [3\*n\*sin(nt),0,0;

6\*n\*(cos(nt)-1),0,0;

0,0,-n\*sin(nt)];

Phi\_vv\_t = [cos(nt), 2\*sin(nt),0

-2\*sin(nt), 4\*cos(nt)-3, 0;

0,0,cos(nt)];

delta\_r\_t = Phi\_rr\_t \* delta\_r\_0 + Phi\_rv\_t \* delta\_v\_0\_plus;

delta\_v\_t = Phi\_vr\_t \* delta\_r\_0 + Phi\_vv\_t \* delta\_v\_0\_plus;

delta\_r\_t\_mat(timestep,:) = delta\_r\_t';

delta\_v\_t\_mat(timestep,:) = delta\_v\_t';

end

v\_plus\_ECI = v\_0 + cross(Omega\_target, delta\_r) + delta\_v\_0\_plus\Q\_LVLH\_ECI;

end

# IV. Matlab Function Test

%% Input Parameters

% Target Information

Target.a = 300+6378;

Target.e = 1e-5;

Target.i = 40;

Target.RAAN = 20;

Target.AOP = 0;

Target.TA = 60;

% Chaser Information

Chaser.a = 6378 + (318.5 + 515.51)/2;

Chaser.e = (6378+515.51)/Chaser.a-1;

Chaser.i = 40.130;

Chaser.RAAN = 19.819;

Chaser.AOP = 70.662;

Chaser.TA = 349.65;

% Timing of Docking

t\_f = 28800;

%% Main Algorithm

% Change to ECI frame

% rv data of the target wrt ECI

[r\_target, v\_target] = kepler2ijk\_hs(Target.a, Target.e, Target.i, Target.RAAN, Target.AOP, Target.TA);

% rc data of the chaser wrt ECI

[r\_chaser, v\_chaser] = kepler2ijk\_hs(Chaser.a, Chaser.e, Chaser.i, Chaser.RAAN, Chaser.AOP, Chaser.TA);

[Delta\_v\_0, Delta\_v\_f, delta\_r\_t, delta\_v\_t, v\_plus\_ECI] = two\_impulse\_rendezvous(r\_target, v\_target, r\_chaser, v\_chaser, t\_f)

Delta\_v\_0 = *3×1*

0.0294

-0.0667

0.0130

Delta\_v\_f = *3×1*

0.0258

0.0005

0.0244

delta\_r\_t = *100×3*

20.0303 20.2865 19.9531

21.5214 6.0269 21.1097

20.5068 -8.3946 19.8978

17.1003 -21.3140 16.4532

11.6842 -31.2357 11.1624

4.8661 -37.0005 4.6191

-2.5889 -37.9157 -2.4425

-9.8442 -33.8328 -9.2300

-16.0859 -25.1640 -14.9818

-20.6135 -12.8361 -19.0526

⋮

delta\_v\_t = *100×3*

0.0093 -0.0468 0.0080

0.0008 -0.0503 -0.0001

-0.0077 -0.0479 -0.0082

-0.0155 -0.0400 -0.0153

-0.0214 -0.0275 -0.0207

-0.0250 -0.0117 -0.0238

-0.0258 0.0055 -0.0243

-0.0236 0.0223 -0.0220

-0.0189 0.0368 -0.0172

-0.0120 0.0472 -0.0105

⋮

v\_plus\_ECI = *3×3*

13.2732 -7.4962 -8.9281

21.0754 0.3060 -1.1259

23.0776 2.3083 0.8764

# Appendix 1: Bundle functions (kepler2ijk, ijk2kepler)

function [r,v] = kepler2ijk\_hs(a,e,inc,RAAN,AOP,theta)

mu = 398600.4415;

RAAN = RAAN/180\*pi;

inc = inc/180\*pi;

AOP = AOP/180\*pi;

theta = theta/180\*pi;

R\_3\_AOP = [cos(AOP), sin(AOP),0;

-sin(AOP), cos(AOP),0;

0, 0,1];

R\_1\_inc = [1, 0, 0;

0, cos(inc), sin(inc);

0,-sin(inc), cos(inc)];

R\_3\_RAAN = [cos(RAAN), sin(RAAN),0;

-sin(RAAN), cos(RAAN),0;

0, 0,1];

Q\_rotation = R\_3\_AOP \* R\_1\_inc \* R\_3\_RAAN;

p = Q\_rotation(1,:);

q = Q\_rotation(2,:);

w = Q\_rotation(3,:);

p\_value = a\*(1-e^2);

r = p\_value\* (cos(theta)\*p + sin(theta) \*q)/(1+e\*cos(theta));

r = r';

v = sqrt(mu/p\_value)\*(-sin(theta)\*p+(e+cos(theta))\*q)';

end

function [a, e, inc, RAAN, AOP, theta] = ijk2kepler\_hs(r,v)

mu = 398600.4415;

h = cross(r,v); % Angular Velocity Vector

z = [0,0,1]';

n = (cross(z,h))/(norm(cross(z,h))); % Normal Vector

e\_vector = ((cross(v,h))/mu) - r/(norm(r)); % Eccentricity vector

a = (-mu)/((norm(v))^2 - 2\*mu/norm(r)); % 1. Semi-major axis

e = norm(e\_vector); % 2. Eccentricity

inc = acos(dot(z,h)/norm(h)) / pi \*180; % 3. Inclination

RAAN = atan2(n(2),n(1)); % 4. RAAN

RAAN = RAAN / pi \* 180;

AOP = acos((dot(n,e\_vector))/norm(e\_vector)); % 5. Argument of periapse

if(e\_vector(3)<0)

AOP = 2\*pi - AOP;

end

AOP = AOP/pi\*180;

theta = acos((dot(r,e\_vector))/(norm(r)\*norm(e\_vector))); % 6. True anomaly

if(dot(r,v) < 0)

theta = 2\*pi-theta;

end

theta = theta / pi \* 180;

end