Navigation and Guidance Systems

Course 086759- Spring 2018

Assignment #7

Inertial Navigation Equations (StrapDown)

Daniel Engelsman 300546173

Part A : Theory

1.

Given a system at rest where $\lambda = \Lambda = 0 \Rightarrow (\dot{\lambda} = \dot{\Lambda} = 0)$. Compute $\omega_{BI}^{B} = f(C_{L}^{B})$:

Orientation Derivative in Strap-Down equations : $\dot{C_{L}^{B}} = C_{L}^{B}\omega_{LI}^{L}{}^{\wedge} - \omega_{BI}^{B}{}^{\wedge}C_{L}^{B}$.

Recall $\dot{C_{L}^{B}} = 0$, so we get $\omega_{BI}^{B}{}^{\wedge}C_{L}^{B} = C_{L}^{B}\omega_{LI}^{L}{}^{\wedge} \quad \backslash \cdot(C_{B}^{L})$

And then isolating : $\omega_{BI}^{B}{}^{\wedge} = C_{L}^{B}\omega_{LI}^{L}{}^{\wedge}C_{B}^{L} = (C_{L}^{B}\omega_{LI}^{L})^{\wedge}$

Now let us compute $\omega_{LI}^{L} = C_{E}^{L}\omega_{EI}^{E} + \cancelto{0}{\omega_{LE}^{L}} = \omega_{EI}^{L}$

Where $\omega_{EI}^{E} = \begin{bmatrix} 0 \\ 0 \\ \omega_e \end{bmatrix}$ and $C_{E}^{L}(\lambda, \Lambda) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$ so $\omega_{LI}^{L} = \begin{bmatrix} \omega_e \\ 0 \\ 0 \end{bmatrix}$

Eventually : $\omega_{BI}^{B}{}^{\wedge} = (C_{L}^{B}\omega_{LI}^{L})^{\wedge} = (C_{L}^{B}\begin{bmatrix} \omega_e \\ 0 \\ 0 \end{bmatrix})^{\wedge}$

2.

Let us write down the following equation :

$$\dot{V^N} = C_B^L a_{SF}^B + g_P^L - (\omega_{LE}^L + 2\omega_{EI}^I) \times V^L \quad \text{where} \quad a_{SF}^B = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The only elements that are suspicious to involve the horizontal components are those whose operation projects on different coordinates system $\Rightarrow$ $g_P^L \neq f(\hat{X}, \hat{Y})$

Accelerometer gives us :

$a_{SF,Z}^B = -sin(\theta) \cdot a_1 + cos(\theta)sin(\phi) \cdot a_2 + cos(\theta)cosn(\phi) \cdot a_3$ (Dependent!)

And the Ω product :

$$(\omega_{LE}^L + 2\omega_{EI}^I) \times V^L = \begin{bmatrix} 2\omega_e cos(\lambda) - cos(\lambda)\dot{\Lambda} \\ \dot{\lambda} \\ sin(\lambda)(\dot{\Lambda} - 2\omega_e) \end{bmatrix} \times \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix} \Rightarrow \begin{bmatrix} \cdots \\ \cdots \\ V_E cos(\lambda)(2\omega_e - \dot{\Lambda}) - V_N \dot{\lambda} \end{bmatrix}$$

Let us sum up the Z components :

$$V_Z^L = -sin(\theta) \cdot a_1 + cos(\theta)sin(\phi) \cdot a_2 + cos(\theta)cosn(\phi) \cdot a_3 + g_E + V_E cos(\lambda)(2\omega_e - \dot{\Lambda}) - V_N \dot{\lambda}$$

One **Cannot claim for independence** in Horizontal components.

3.

An aircraft flies north ($\dot{\Lambda} = 0$) at constant H and $V^L = [V_N \quad 0 \quad 0]^T$.

Like previously : $\cancelto{0}{\dot{V^L}} = C_B^L a_{SF}^B + g_P^L - (\omega_{LE}^L + 2\omega_{EI}^I) \times V^L$ given $(C_B^L = I)$ we'll get :

$a_{SF}^B = -g_P^L + (\omega_{LE}^L + 2\omega_{EI}^I) \times V^L$

$$(\omega_{LE}^L + 2\omega_{EI}^I) \times V^L = \begin{bmatrix} 2\omega_e cos(\lambda) - cos(\lambda)\dot{\Lambda} \\ \dot{\lambda} \\ sin(\lambda)(\dot{\Lambda} - 2\omega_e) \end{bmatrix} \times \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix} = \begin{bmatrix} 2\omega_e cos(\lambda) - cos(\lambda)0 \\ \dot{\lambda} \\ sin(\lambda)(0 - 2\omega_e) \end{bmatrix} \times \begin{bmatrix} V_N \\ 0 \\ 0 \end{bmatrix}$$

Compute the vector product : $\begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ 2\omega_e cos(\lambda) & \dot{\lambda} & -2\omega_e sin(\lambda) \\ V_N & 0 & 0 \end{vmatrix} = \begin{bmatrix} 0 \\ 2\omega_e V_N sin(\lambda) \\ -\dot{\lambda} V_N \end{bmatrix}$

Plug $\dot{\lambda} = \frac{V_N}{R_M + H}$ and we get $a_{SF}^B = \begin{bmatrix} 0 \\ 2\omega_e V_N sin(\lambda) \\ -\frac{V_N^2}{R_M + H} - g_E \end{bmatrix}$

4.

An IMU is at rest where $[\lambda, \Lambda] = [32°, 35°]$ and $h_0 = 0.0[m]$ @ 100 [Hz].

Since no force applies, the accelerometer measures : $\Delta u \sim \frac{[0 \quad 0 \quad g_E]^T}{100} \; [\frac{1}{sec}]$

And its orientation changes : $\Delta\theta = \frac{[\phi \quad 0 \quad \psi] \cdot 10^{-5}}{100} \; [rad]$

Since only gravity applies, the orientation components nullify as :

$\phi = \tan^{-1}(\frac{a_y}{a_Z}) = 0$ and $\theta = \sin^{-1}(\frac{a_X}{g}) = 0.$

$\omega_{BI}^B = \Delta\theta \cdot \mathbf{f} = \omega_{EI}^B$ where $\mathbf{f}$ denotes frequency.

Next we should compute :

$$\omega_{EI}^{B} = C_{L}^{B} C_{E}^{L} \omega_{EI}^{E} = C_{B}^{L} \cdot \begin{bmatrix} \omega_e cos(\lambda) \\ 0 \\ -\omega_e sin(\lambda) \end{bmatrix}$$

C_{E}^{B} is a **VERY** long computation that I skipped writing in LATEX, but eventually got :

$$\omega_{EI}^{B} = \omega_e \begin{bmatrix} cos(\theta)cos(\psi)cos(\lambda) + sin(\theta)sin(\lambda) \\ \cdots \\ \cdots \end{bmatrix}. \text{ (only the 1st row is usable...)}$$

And now equating to $\hat{X}$ sampling :

$$\psi = cos^{-1}\left[\frac{\Delta\theta \cdot \mathbf{f}}{\omega_E cos(\theta)cos(\lambda)} - tan(\theta)tan(\lambda)/\omega_E\right] \text{ when } \phi = \theta = 0.$$

Part B : Strap-Down

5. MATLAB Code

Given the initial conditions, we'll implement the Strap-Down equations using state vector :

$$\bar{x} = [\lambda \quad \Lambda \quad h \quad V_N \quad V_E \quad V_D \quad C^B_{L\,9x1} \quad]^T.$$

Whereas its derivative $\dot{\bar{x}}$ is the argument of the integration. Let us first initialize time boundary, and function handle calling :

```
% ———————— State Vector Implementation ———————— %
dt = 1;                          % Integration step size [sec]
t_0 = 0; t_F = 20*60;            % Total run : 20 minutes
N = ( t_F - t_0 )/dt;
t = linspace(t_0, t_F, N);  % Spanning time on vector
IC = [P_0; V_n_0; T_B2L_0; P_NED_0];     % IC - IVP

% ———— Define F(t,x) Function Handle using RK4 ———— %
F = @(tt,x) dx(tt,x);
X = RK_4( t, F, dt, IC );
```

Then we can use numerical integration RK-4, whose length fits the problem :

```
function Y = RK_4( X, F, h, IC )
% ———————— Runge Kutta Algorithm ———————— %
x_len = length(X);
Y(:,1) = IC;                     % Initial Value Problem

for i = 1:x_len-1                % Each K gets 2 rows state vector
    k1 = h*F( X(i)       , Y(:,i)       );
    k2 = h*F( X(i)+0.5*h, Y(:,i)+0.5*k1 );
    k3 = h*F( X(i)+0.5*h, Y(:,i)+0.5*k2 );
    k4 = h*F( X(i)+    h, Y(:,i)+    k3 );
    Y(:,i+1) = Y(:,i) + (1/6)*( k1 + 2*k2 + 2*k3 + k4 );
```

```
end
```

Using function *dx* which defines the problem :

```
function dX = dx(~, x)
global Ro e omg_IE
% ------------------ Assigning updated values --------------- %
[phi, lambda, h] = deal(x(1), x(2), x(3) );
V_n              =      [x(4), x(5), x(6)]';
C_L2B = vec2mat( x(7:15), 3);            % Unfolding C_L2B
C_B2L = C_L2B';                          % Define C_B2L
% ------------------ Geographic Coordinates ----------------- %
Rn = Ro/(1-(e*sin(phi))^2)^0.5;          % Meridian Radius
Rm = Ro*(1-e^2)/(1-(e*sin(phi))^2)^1.5; % Normal Radius
% ------------ Transformation to Navigation Frame ---------- %
C_E2L = [   -sin(phi)*cos(lambda) -sin(phi)*sin(lambda)  cos(phi)
    ;
    -sin(lambda)           cos(lambda)              0          ;
    -cos(phi)*cos(lambda) -cos(phi)*sin(lambda)  -sin(phi)];
%% ------------------- Inertial Sensors ------------------- %
% ---------------- Accelerometer Properties ---------------- %
g_P = [0 0 g_L( phi, h )]';              % g as f(phi, h)
a_SF2B = -g_P;                            % Acc. Measurements (B)
w_a = [0,0,0]';                          % Acc. White Noise
b_a = [0,0,0]';                          % Acc. Bias
a_SF = a_SF2B + w_a + b_a;               % Acc. Data (Total)
% ------------------- Gyro Properties ------------------- %
omg_x = 0; omg_y = 0; omg_z = 0;         % No angular rates given
omg_LI2B = [omg_x omg_y omg_z]';         % Gyro sensor measurements
w_g = [0,0,0]';                          % Gyro White Noise
b_g = [0,0,0]';                          % Gyro Bias
omg_BI2B = omg_LI2B + w_g + b_g;         % Gyro Data (Total)
Omg_BI2B = skew_symmetric(omg_BI2B);     % Skew Symmetric Form
%% ------------------- d_Position ------------------- %
D  = [ 1/(Rm+h) 0                    0;
       0        1/(cos(phi)*(Rn+h)) 0;
```

```
        0          0                    -1];
% ---------- Assigning P_n(t) ---------- %
dX(1:3) = D*[V_n(1) V_n(2) V_n(3)]';     % P Result
%% ---------- Earth's Angular rate's Properties ---------- %
Omg_LE2L = 0;                            % Given I.C.
omg_EI2E = [0 0 omg_IE]';                % Earth's Angular rate
omg_LI2L = C_E2L*omg_EI2E + Omg_LE2L;
Omg_LI2L = skew_symmetric(omg_LI2L);
dC_L2B   = C_L2B*Omg_LI2L - Omg_BI2B*C_L2B;
%% ---------- d_Velocity ---------- %
omg_EI2L = C_E2L*omg_EI2E;
Omg_EI2L = skew_symmetric(omg_EI2L);
dV = C_B2L*a_SF + g_P - ( Omg_LE2L + 2*Omg_EI2L )*V_n; % V Result
% ---------- Assigning V_n(t) ---------- %
dX(4:6) = [dV(1)  dV(2)  dV(3)]';
dX(16:18) = x(4:6);      % P_NED = Integral of NED velocity
%% ---------- d_Orientation ---------- %
% ---------- Assigning T_b_n(t) ---------- %
dX(7:15) = reshape( dC_L2B', 9, 1);      % Folding dC_L2B
% ---------- Assigning Total State Vector ---------- %
dX = dX';
```

5. Code Implementation

Ideal Results

In an ideal world, where the Earth stays still and not rotating, one get the following results:

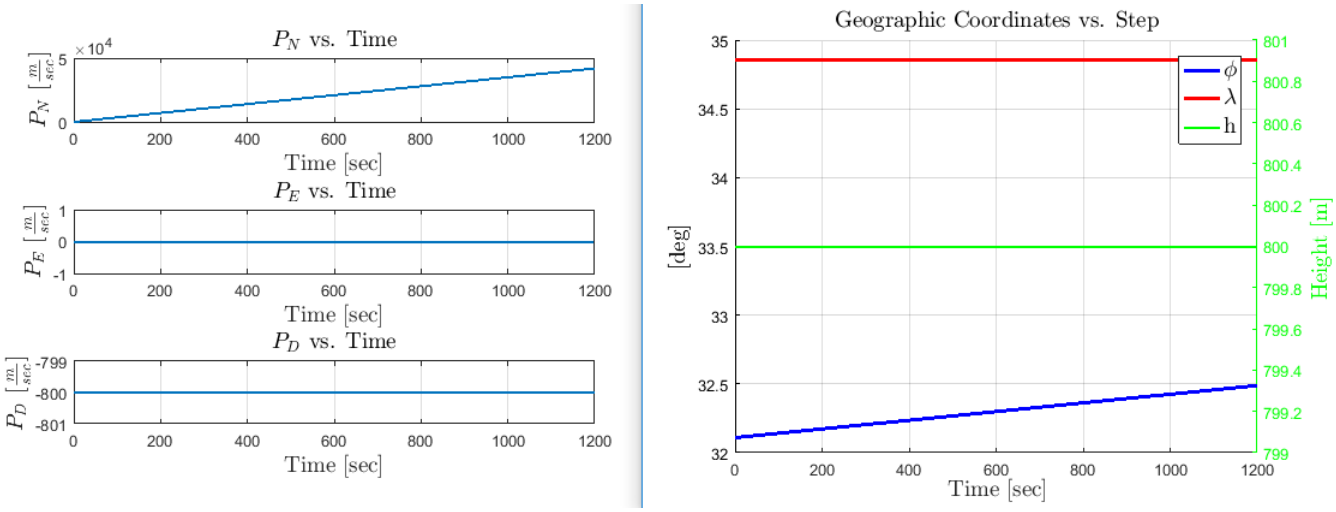

Figure 1: Aircraft location (**Left**) and Geodetic Coordinates (**Right**) vs. Time

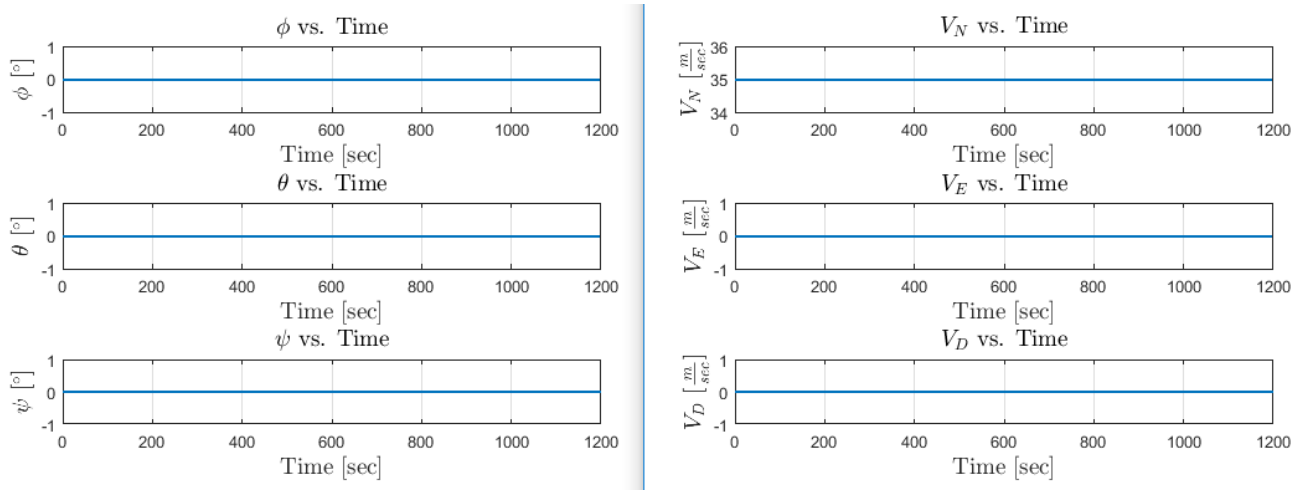

Figure 2: Body Orientation (**Left**) and V_{NED} (**Right**) vs. Time

The distance the Plane would pass equals to the pure product of $P = V_N \cdot \Delta t$.

Real Results

Let us now "turn on" Earth's angular rate switch and we get:

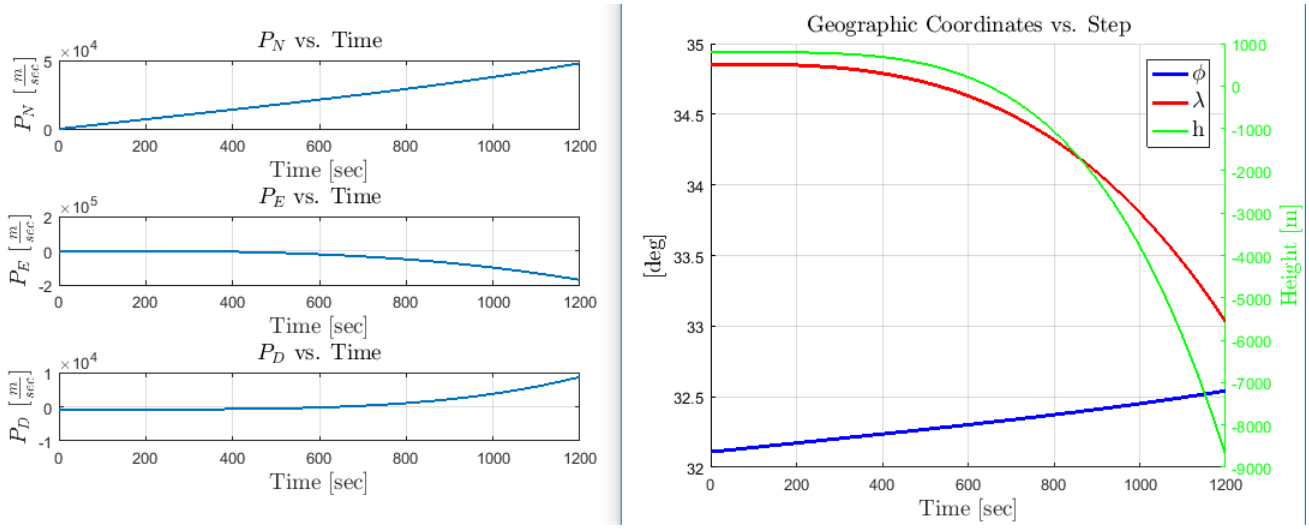

Figure 3: Aircraft location (**Left**) and Geodetic Coordinates (**Right**) vs. Time

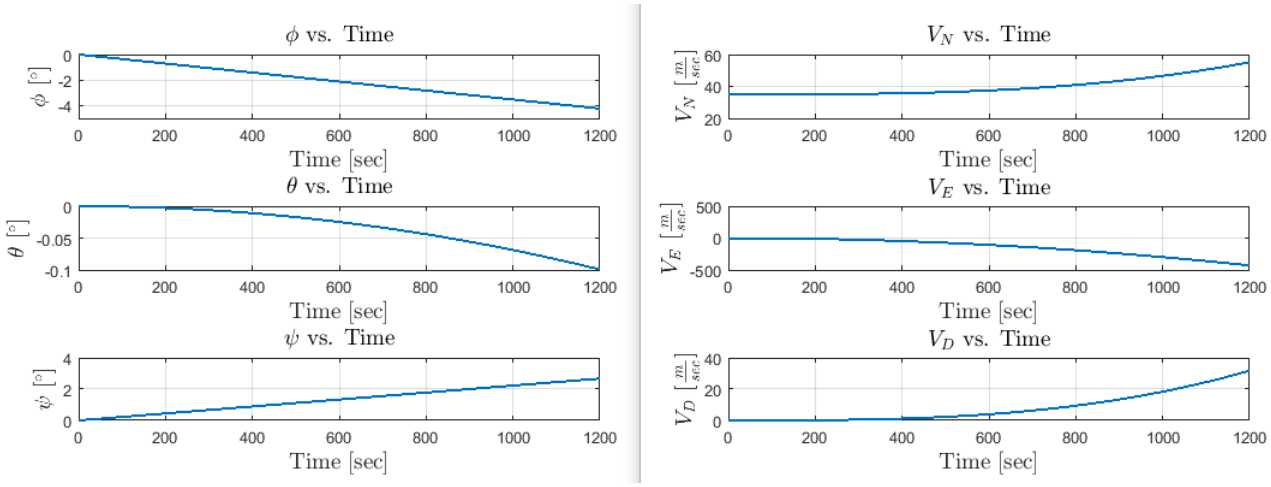

Figure 4: Body Orientation (**Left**) and V_{NED} (**Right**) vs. Time

One can see the inevitable noise it causes, which in turn drifts the results significantly.

6. Aircraft Position and Velocity after 20 mins

Both position and velocity have been presented above, but now I will show the location :

- Initial Coordinates: [32.109333, 34.855499]
- Final **Ideal** Coordinates: [32.488103, 34.855499], Azimuth = 0 [°]
- Final **Real** Coordinates: [32.542615 33.032042], Azimuth = -74.994 [°]

Using the Google Maps ©, we'll plug the results :

Figure 5: Final Aircraft destinations due to drift in Time

As seen on the graphs above, there is a strong West velocity drift which sweeps the aircraft leftward, all across the sea.

7. V_E change vs. Time

No doubts there is a strong difference between active Earth rotated by ω_{IE} and Earth still.

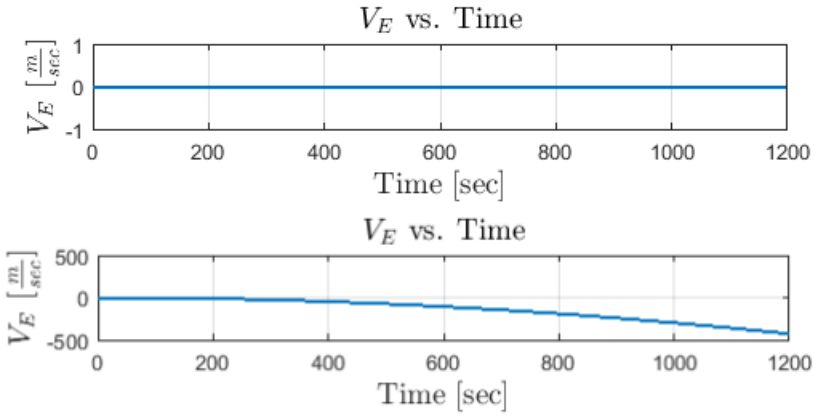

Figure 6: V_E Comparison between Ideal **(Top)** and Real simulation **(Bottom)**

Explanation : 20 Minutes is quite a long time where error evolves and diverges drastically. When no estimator or external Data is combined to "truncate" the errors, the system loses track quickly.

8. Ideal Case with h = 3000 [m]

In order to isolate the altitude variable, I will run the simulation when $\omega_{IE} = 0$. Let us run few different altitudes to demonstrate its influence :

- Initial Coordinates: [32.1093, 34.8555], Distance = 0 [m]
- Final **Ideal** Co. (h=3,000 [m]) : [32.4880 34.8555], Distance = 4.1992e+04 [m]
- Final **Ideal** Co. (h=10,000 [m]) : [32.4876 34.8555], Distance = 4.1946e+04 [m]
- Final **Ideal** Co. (h=100,000 [m]) : [32.4823 34.8555], Distance = 4.1360e+04 [m]
- Final **Ideal** Co. (h=1,000,000 [m]) : [32.4366 34.8555], Distance = 3.6293e+04 [m]

The usual suspect that might cause that change, is probably the gravity function :

$$g_{\lambda h} = 9.780327\left(1 + 0.0053024\sin^2\lambda\right)\left(1 - 2h/R_e\right)$$

One can see that the further the altitude goes, the further the gravity decays, as it depends on the geodetic coordinates. In turn it affects on the different parameters which are used in the Code.