AE4879 Mission Geometry and Orbit Design

Assignment 9: Earth coverage
EARTHCOV-2 and EARTHCOV-3

Simon Billemont

December 10, 2010

Assignment 9: Earth coverage

This assignment deals with how a satellite sees (maps) the Earth. This is done by analyzing the transform form the Earth reference sphere to the satellite reference sphere and vice-versa. Furthermore, an analysis of how geometric shapes get deformed when they are transformed will be done.

1. Mapping a point from the earth to a satellite

To see where a point on the earth surface is in the satellite unit sphere, the location needs to be transformed with a set of rules. In order to start the transform, we first define the position on the earth with a set of coordinates. This is done in the geocentric system with the points latitude ($\varphi_{P,E}$) and longitude ($\theta_{P,E}$). Also the location of the satellite needs to be known. The location is defined by the sub-satellite point latitude/longitude (φ_{SSP}, θ_{SSP}) and the satellite orbit height (H). This can be visualized as is figure 1.

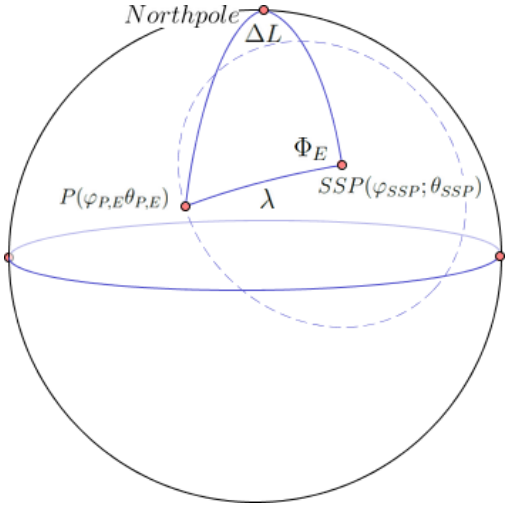


Figure 1: Earth coordinates (with [1])

The first step is to rewrite the location of the point with respect to the sub satellite point (SSP). This can be done with the spherical triangle made by the North pole, the point P and the sub satellite point. One wants to know the longitude difference between the SSP and P, and the angular distance between the SSP and P. These can be found using OCDM[2] eq A-20b and OCDM[2] eq B-19 (full sky geometry, see previous assignment)

$$b = \cos^{-1}\left(\sin c \sin a \cos B + \cos c \cos a\right) \tag{1}$$

$$\Delta\phi_2 = \cos 2^{-1}\left(\frac{\cos\delta_1' - \cos\rho\cos\delta_2'}{\sin\rho\sin\delta_2'}, H(\Delta\phi_1)\right) \tag{2}$$

These equations are rewritten with the proper symbols and yield eq3,4 . Note c and a are colatitude. These were converted using trigonometric co-function identities to

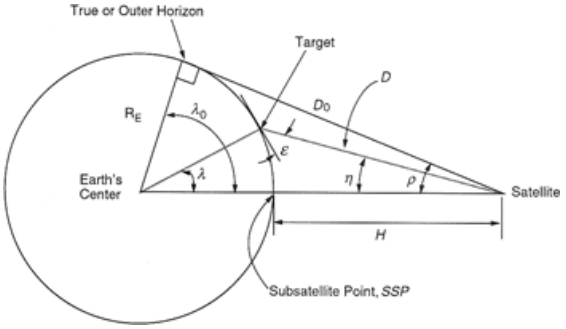


Figure 2: 2D intersection of the satellite-Earth 3D geometry

normal latitude [3]. Furthermore, the $\cos 2^{-1}$ and hemisphere function H where used from OCDM[2].

$$\lambda = \cos^{-1}\left(\cos(\Delta \mathrm{L})\cos\left(\varphi_{\mathrm{SSP}}\right)\cos\left(\varphi_{e,P}\right) + \sin\left(\varphi_{\mathrm{SSP}}\right)\sin\left(\varphi_{e,P}\right)\right) \tag{3}$$

$$\Phi_E = \cos 2^{-1}\left(\frac{\sin\left(\varphi_{e,P}\right) - \cos(\lambda)\sin\left(\varphi_{\mathrm{SSP}}\right)}{\sin(\lambda)\cos\left(\varphi_{\mathrm{SSP}}\right)}, H(\Delta \mathrm{L})\right) \tag{4}$$

$$\text{with } \Delta L = \left(\theta_{SSP} - \theta_{P,E}\right) \mod (2\pi) \tag{5}$$

Now the viewpoint is switched to the satellite viewpoint. This unit sphere has the zero latitude line (equator) parallel to that of the Earths equator. Also per definition, the line that connects the center of the earth and the satellite (creates the SSP on the Earth and nadir point on the satellite sphere) intersects the zero longitude line. Assume that the satellite has a positive latitude wrt the Earth. One sees that the connecting line departs the Earth at a positive latitude (SSP), and arrives at the satellite at a negative latitude (sat nadir). Thus the latitude switches sign. Summarized the SSP on the earth transforms into the satellite nadir point with eq 6, 7.

$$\theta_{nadir} = 0 \tag{6}$$

$$\varphi_{nadir} = -\varphi_{SSP} \tag{7}$$

Now the coordinates of the point can now be converted to the satellite unit sphere. To do this, one first defines eq 8:

$$\sin\rho = \frac{R_E}{R_E + H} \tag{8}$$

The angle at which the satellite sees the point (η, see fig 2;3) can now be determined by eq 9. The angles η on the right hand side can be rewritten with λ (eq 10). By substituting the previous equation 8, one can remove the explicit dependence on the distances between satellite and Earth.

$$tan(\eta) = \frac{\sin\eta}{\cos\eta} \tag{9}$$

$$tan(\eta) = \frac{R_E \sin\lambda}{(R_E + H) - R_E\cos\lambda} \tag{10}$$

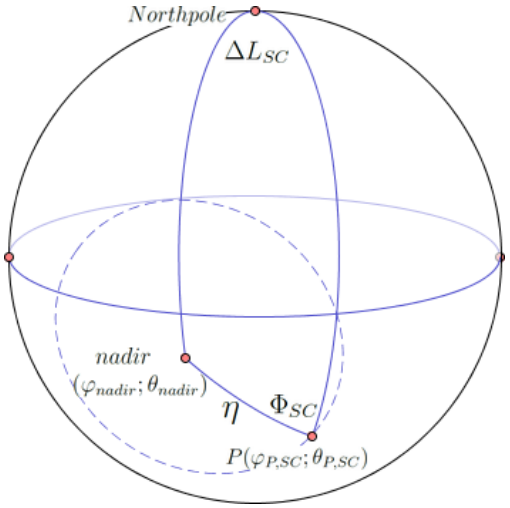


Figure 3: Satellite coordinates (with [1])

$$tan(\eta) = \frac{\sin\rho\sin\lambda}{1-\sin\rho\cos\lambda} \tag{11}$$

Using η, it can be determined if the respective point is below the horizon. This is done by computing the elevation ϵ (Simple sum of angles eq 12). The angle that gives the direction of point P wrt nadir (Φ_{SC}) flips sign. This is because of the viewpoint change. At first, one looks from the inside of the sphere of the Earth. Now one one moves outside of the sphere causing the flip of position. Consider that the target P is on the right side of the SSP. Moving outside of the sphere, causes the longitude line to flip to the left side of the SSP (or nadir).

$$el = pi/2 - \lambda - \eta \tag{12}$$

$$\Phi_{SC} = -\Phi_E \tag{13}$$

The coordinates (η;Φ_{SC}) can now be converted back to the generic latitude and longitude in the satellite frame. This is done similarly as to the forward conversion done before, with spherical triangles. The latitude of the target P on the spacecraft sphere can be found with OCDM[2] eq A20-c (see eq 14). Then the longitude between the nadir point and the target can be found with full sky geometry equations (OCDM[2] eq B-15)

$$c = \cos^{-1}\left(\sin a\sin b\cos C + \cos a\cos b\right) \tag{14}$$

$$\Delta\phi_1 = \text{cos2}^{-1}\left(\frac{\cos\delta_2' - \cos\rho\cos\delta_1'}{\sin\rho\sin\delta_1'}, H(\Delta\phi_1)\right) \tag{15}$$

Rewritten with the proper symbols and applying trigonometric co-function identities leads to eq 16;17. From the longitude difference between the target and the nadir point, the longitude of the target can then also be found.

$$\varphi_{P,SC} = \sin^{-1}\left(\sin\varphi_{nadir}\cos\eta + \cos\varphi_{nadir}\sin\eta\cos\Phi_{SC}\right) \tag{16}$$

$$\Delta L_{SC} = \text{cos2}^{-1}\left(\frac{\cos\eta - \sin\varphi_{P,SC}\sin\varphi_{nadir}}{\cos\varphi_{P,SC}\cos\varphi_{nadir}}, H(\Phi_{SC})\right) \tag{17}$$

$$\theta_{P,SC} = (\theta_{nadir} - \Delta L_{SC}) \mod (2\pi); \tag{18}$$

Now the coordinates ($\varphi_{P,SC}$, $\theta_{P,SC}$) in the spacecraft system are know for a given target on earth ($\varphi_{P,E}$, $\theta_{P,E}$). The conversion from satellite to the Earth celestial sphere is near identical, but earth and satellite are switched. Only the transformation of the viewpoint is a bit different. One can find the elevation using eq 19 (see OCDM[2] 9-12). Then, the distance between the SSP and the target (λ) can be found with a simple sum of angles in a triangle (see eq 20) .

$$\epsilon = \cos -1 \left(\frac{\sin \eta}{\sin \rho} \right) \tag{19}$$

$$\lambda = \pi - \frac{\pi}{2} - \eta - \epsilon \tag{20}$$

Furthermore in order to gain a bit of insight into the coordinate transformation I made a small interactive utility "Coverage2" , see appendix A .

2. Conversion tests

A program was written in MATLAB[4] to perform this conversion. For this program, a set of automated test cases where evaluated using the "MATLAB xUnit Test Framework"[5]. These test cases where based on the test cases presented in [6] slides 29-30. The first test case is defined by:

$H_{sc} = 1000$km; SSP: $\varphi_{SSP} = 20°, \theta_{SSP} = 270°$ and an target $\varphi_{P,E} = 40°, \theta_{P,E} = 290°$

The computed position of the target on the satellite celestial sphere was:

$$[\varphi_{P,SC} = 28.7055, \theta_{P,SC} = 324.5020]$$

The correct answer was (see slide 29):

$$[\varphi_{P,SC} = 28.72, \theta_{P,SC} = 324.49]$$

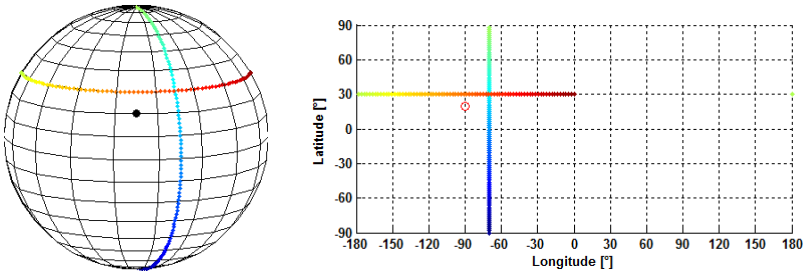
This means that the output differs only about $0.02°$, and confirms the validity of the Earth⇒Sat conversion. The second test case uses the previously developed DualAxis-Spiral solver to find the position on Earth of a circle (scan line) on the satellite. The specific parameters are:

$H_{sc} = 1000$km; SSP: $\varphi_{SSP} = 20°, \theta_{SSP} = 270°$ scan line $50°$ north of nadir

The specific point on satellite scan circle considered is $\phi_2 = 210°$. The computed coordinates of the target can be found in table 1. Note all values are latitude not colatitude. The source code for the tests can be found in appendix B.

	Computed [°]	Slides [°]
$\varphi_{P,SC}$	31.9686	31.97
$\theta_{P,SC}$	22.2625	22.26
$\varphi_{P,E}$	36.1932	36.17
$\theta_{P,E}$	261.5742	261.59
φ_{SSP}	20	20

Table 1: Test case 2 results



(a) (b)

Figure 4: Latitude and longitude line on the Earth unit sphere

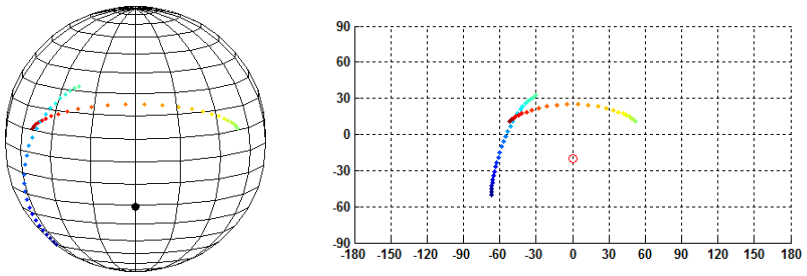
3. Latitude and longitude lines

For a situation (OCDM Figure 9-8), a satellite at 20 N, 90 W (H=1000 km).

- **Develop a program that projects Earth-centered lat/long coordinates into spacecraft-centered lat/long coordinates, as defined in the previous sheets.**
- **Verify the Earth-spc transformation as on the previous sheets ($\Rightarrow$ the equations used and the outcome by your program).**
- **Verify the projection of the longitude line (70° W) in an Earth-fixed system into the spc-centered celestial sphere. Verify if the direction is reversed.**
- **Verify the projection of the latitude line at (30° N) in an Earth-fixed system into the spc-centered celestial sphere. Verify if the direction is reversed.**

Consider a satellite at $\varphi_{SSP} = 20^\circ$ N and $\theta_{SSP} = 90^\circ$ W. Now we are interested in how latitude and longitude lines are projected onto the spacecraft unit sphere. The specific lines considered are the great-circle segments that define a constant longitude of 290° and latitude of 20°. In the earth reference frame that gives fig 4. The projection on the satellite reference frame is given in fig 5.

The results are similar to what is given in OCDM[2] fig 9-8. What can be seen is that both the latitude and longitude line is bended around the nadir point (points that are visible). What we can see is that from the SSP to nadir, the latitude of the point flipped. Furthermore, points that where above the SSP remain above the nadir point. This means that the latitude is not flipped for the individual points. What however does flip is the longitude. Point on the right of the SSP are on the left of the nadir point and vice-versa.



(a) (b)

Figure 5: Latitude and longitude line on the satellite unit sphere

	$\Phi[^\circ]$	$\lambda_1[^\circ]$	$\lambda_2[^\circ]$
A	45	15	20
B	80	20	30
C	20	28	30
D	50	28	30

Table 2: Area definitions

4. Area transformation

Consider a situation like OCDM Figure 9-10. Assume the characteristics for the various areas A-D as in table 2.Compute the surface areas of A-D individually, on an Earth-centered sphere, as well as on the spacecraft-centered celestial sphere (expressed in steradians in both cases). Assume a satellite altitude of 1000 km.

To compute the area of a annular segment around the nadir or SSP, one needs to know three parameters. The length of the annular section (Φ), the inner radius (λ_1) and the outer radius (λ_2). Then with the basic formula (eq 21, see OCDM[2] Table 9-1) to compute the area of an annular segment one can find the area in steradian. For the equivalent area on the satellite centered celestial sphere, one needs to convert the radiuses from the Earth system to the satellite system (see 10. The segment length remains constant (it only changes sign, see 13)

$$A = \Phi \left(\cos \lambda_1 - \cos \lambda_2 \right) \tag{21}$$

The results for the areas defined in table 2, the following areas where found:

- $\Phi = 45^\circ, \lambda_1 = 15^\circ, \lambda_2 = 20^\circ : A_{earth} = 0.0206sr, A_{sat} = 0.0453sr$
- $\Phi = 80^\circ, \lambda_1 = 20^\circ, \lambda_2 = 30^\circ : A_{earth} = 0.1029sr, A_{sat} = 0.0465sr$
- $\Phi = 20^\circ, \lambda_1 = 28^\circ, \lambda_2 = 30^\circ : A_{earth} = 0.0059sr, A_{sat} = 0.0004sr$
- $\Phi = 50^\circ, \lambda_1 = 28^\circ, \lambda_2 = 30^\circ : A_{earth} = 0.0148sr, A_{sat} = 0.0010sr$

References

[1] D. Austin and W. Dickinson. (2009) Spherical easel. A spherical drawing program. [Online]. Available: http://merganser.math.gvsu.edu/easel/

[2] J. R. Wertz, *Orbit & Constellation Design & Management*, second printing ed. El Segundo, California: Microcosm Press, 2009.

[3] E. W. Weisstein. (2010) Trigonometry. MathWorld. [Online]. Available: http://mathworld.wolfram.com/Trigonometry.html

[4] MathWorks. (2010a) Matlab 7.11. Natick, MA.

[5] S. Eddins. (2010, Nov) Matlab xunit test framework. MATLAB Central File Exchange. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/22846

[6] R. Noomen, *AE4-879 Optimisation V3.2*, TUDelft Lecture Slides, 2010.

Additional information

Estimated work time:

$\sim$ 3h Studying theory + $\sim$ 4h making assignment + $\sim$ 4h writing report = $\sim$ 11h

Made by

Simon Billemont

Stud Nr: 1387855

s.billemont@student.tudelft.nl

License and notices

Version history

Version 1: Initial document

A. Coverage2

Coverage 2 is an interactive MATLAB[4] GUI tool to get insight into how geometry on the earth is transformed when is being inspected by a satellite. Converts points form the earth reference frame to the satellite reference frame. It shows a rotatable 3D scene and projections on both celestial spheres. The script is available on:

http://projects.angelcorp.be/?project=Coverage%202

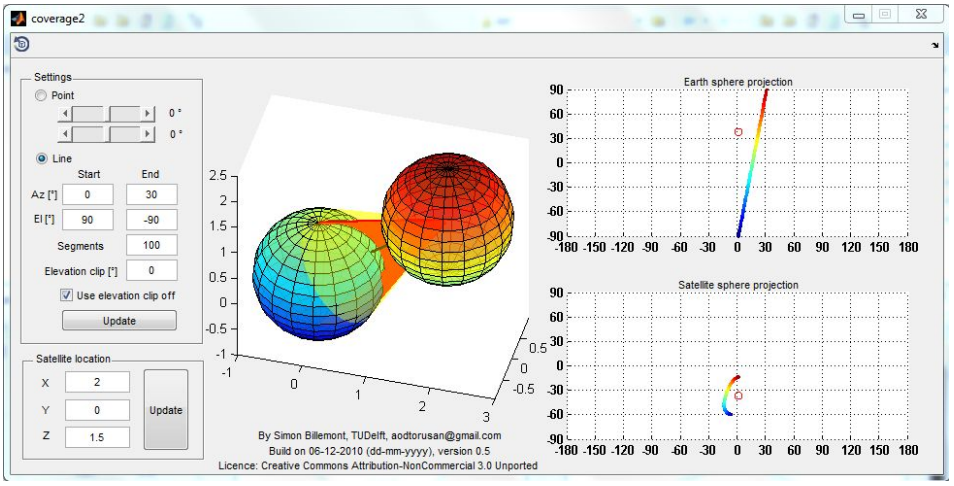


Figure 6: Coverage2 an interactive MATLAB[4] geometry projection visualization tool

B. Matlab source code

The code written to implement the three described optimizers was written in MATLAB 7.11 (2010b)[4]. A structured overview of the dependencies is given below:

- EARTHCOV2.m
  - greatCircle.m
  - EarthSatReferenceConverter.m
    - H.m
    - acos2.m
  - globePlot.m
- EARTHCOV3.m
  - annulusSegment.m
- test.m
  - EarthSatReferenceConverter.m
  - DualAxisProblem.m

Listing 1: EARTHCOV2.m: Compute the transform of a latitude and longitude line

```
%% By: Simon Billemont, sbillemont, 1387855
%   Contact: aodtorusan@gmail.com or s.billemont@student.tudelft.nl
%       Solve the Side-Angle-Side problem on a shperical triagle
%       Equations from Orbit & Constellation Design & Management (ocdm)
%   Made on: 04-12-2010 (dd-mm-yyyy)
%   This work is licensed under the
%       Creative Commons Attribution-NonCommercial 3.0 Unported License.
%       To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/
%% Setup environment
clc
clearvars
close all

% Make a new utility for saving pictures
saver = ImSav();
saver.c_plotsDir = '../images/';            % Where the plots will go
saver.c_changec_figSize = 0;                % Do not resize
saver.c_appendFigureNr = 0;                 % Do not append stuff the the name
saver.c_saveFigure = 0;

%% Only procede if all the unit tests are succesfull
if (~runtests)
    error('There are errors in the test cases. Fix these first')
end

%% Compute

% Create the longditude line of 70 W
co = greatCircle.fromRange(deg2rad(290),[-pi/2 pi/2], 100);
% Create the latitude line of 30 N
co = [co, greatCircle.fromRange([pi, 2*pi], deg2rad(30), 80)];

% Reformat lat and long of all the points
lon_E = co(1,:);
lat_E = co(2,:);

% The sub satillite point
lat_SSP = deg2rad(ones(size(lat_E))*(20));
lon_SSP = deg2rad(ones(size(lat_E))*(360-90));

% Convert the coordinates
converter = EarthSatReferenceConverter([lat_E; lon_E; lat_SSP; lon_SSP]', true);
converter.setRho(6371E3, 1000E3);
converter.c_elevarionCutOff = 0; % elevation cutoff angle
converter.c_filerElevations = 1; % do elevation cutoff
converter.compute();             % Convert the coordinates
sat_co = converter.sc_co;


% Make 3d globe plots
figure
globePlot('point',   [0.3491, 4.7124, 1.02], 'filled', 'k',...
          'segment', [lat_E; lon_E; ones(1,length(lon_E))]');
view(0,10);
saver.saveImage('earth-3d'); close;
globePlot('point',   [sat_co(1,3:4), 1.02], 'filled', 'k', ...
          'segment', [sat_co(:,1:2)'; ones(1,size(sat_co,1))]');
view(90,10);
saver.saveImage('sat-3d'); close;

% Convert to fancy degrees
sat_co = rad2deg(sat_co);  % Sat coordinates to degree
sat_co(sat_co > 180) = sat_co(sat_co > 180) - 360; % Center around 0 in the middle
co = rad2deg(co);                   % Earth coordinates to degree
co(co > 180) = co(co > 180) - 360;  % Center around 0 in the middle

% Make 3D plots
plot2k(co(1:2,:)',1:size(co,2));
hold on
    scatter(rad2deg(lon_SSP)-360, rad2deg(lat_SSP),'ro');
hold off
set(gca, 'YTick', -90:30:90)
set(gca, 'YLim',  [-90, 90])
set(gca, 'XTick', -180:30:180)
set(gca, 'XLim',  [-180, 180])
grid on
xlabel('Longitude [ ]');
ylabel('Latitude [ ]');
saver.saveImage('earth-2d')
undo(2) %remove last 2 plots
plot2k(sat_co(:,2:-1:1),1:size(sat_co,1));
hold on
    scatter(sat_co(:,4),sat_co(:,3),'ro');
hold off
saver.saveImage('sat-2d');
```

Listing 2: greatCircle.m: Creates GreatCircle segment points

```
classdef greatCircle
% Wrapper for common greatCircle opperations
```

```
% By: Simon Billemont (s.billemont@student.tudelft.nl)
% Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.
    methods (Static)
        function [ co ] = fromRange( lonRange, latRange, samples )
            % Create a great circle segment based on two points
            % Simple interpolation is done
            % If the second entry in lon/latRange is missing, it is asumed to be constant
            if (length(lonRange) == 1 || lonRange(1) == lonRange(2))
                lon = ones(1, samples)*lonRange(1);
            else
                dLon = (max(lonRange)-min(lonRange)) / (samples-1);
                lon  = min(lonRange):dLon:max(lonRange);
            end

            if (length(latRange) == 1 || latRange(1) == latRange(2))
                lat = ones(1, samples)*latRange(1);
            else
                dLat = (max(latRange)-min(latRange)) / (samples-1);
                lat  = min(latRange):dLat:max(latRange);
            end
            co = [lon; lat];
        end
    end
end
```

Listing 3: EarthSatReferenceConverter.m: Convert between satellite and Earth celestial sphere

```
classdef EarthSatReferenceConverter < handle
%EARTHSATREFERENCECONVERTER Converts a series of points between two celstial spheres
%   Converts between Earth<=>Sat celestial sphere.
% function this = EarthSatReferenceConverter(coordinates, earthCentered)
%       coordinates:    [Lat_P-E,  Lon_P-E,  Lat_SSP-E,  Lon_SSP-E ] (or equivalent)
%       earthCentered:  Boolean true if the given coordinates are on the earth
%   Note YOU MUST MANUALLY SET RHO
% By: Simon Billemont (s.billemont@student.tudelft.nl), on: 04-12-2010 (dd-mm-yyyy)
% Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.

    properties (Access=public)
        %% Configuration settings
        c_filerElevations = false; % Remove elements under the elevation cutoff angle
        c_elevarionCutOff = 0;     % Define the elevation cutoff angle (+ above horizon) [rad]

        %% list of variables
        rho;        % Sattillite FOV
        accepted;   % Points that have been accepted due to the elevation cutoff

        % Earth centered
        earth_co;           % [Lat_P-E,  Lon_P-E,  Lat_SSP-E,  Lon_SSP-E ]
        earth_center_co;    % [lambda, Phi_E,  el]
        % Spacecraft centered
        sc_center_co;       % [eta,    Phi_SC, el]
        sc_co;              % [Lat_P-SC, Lon_P-SC, Lat_SSP-SC, Lon_SSP-SC]
    end

    methods
        % Generic coinstuctor for the converter
        function this = EarthSatReferenceConverter(coordinates, earthCentered)
            this.clearAll();
            if (earthCentered)
                this.earth_co = coordinates;
            else
                this.sc_co    = coordinates;
            end
        end
        % Sets rho ; ). based on the earth radius and sat altitude.
        function setRho(this, Re, h)
            this.rho = asin(Re/(Re+h));
        end
        % Do the conversion process (wrapper for the actual converter)
        function compute(this)
            if (isempty(this.sc_co))
                this.earth2sat();
            elseif (isempty(this.earth_co))
                this.sat2earth();
            end
        end
    end
    methods (Access=private)
        % Clear all internal conversion values
        function clearAll(this)
            % Clears all intermediate values
            this.earth_co = []; this.earth_center_co = [];
            this.sc_center_co = []; this.sc_co = [];
            this.accepted = [];
        end
        %Convert from the earth celestial sphere to the satellite celstial sphere
        function earth2sat(this)
            Lat_P_E =   this.earth_co(:,1);           % Simple name
            Lon_P_E =   this.earth_co(:,2);           % Simple name
            Lat_SSP_E = this.earth_co(:,3);           % Simple name
```

```
            Lon_SSP_E = this.earth_co(:,4);            % Simple name
            Lat_nadir_SC = -Lat_SSP_E;                 % Latitude flips
            Lon_nadir_SC = zeros(size(Lat_nadir_SC));  % definition, lon=0

            DL_earth = mod(Lon_SSP_E - Lon_P_E, 2*pi); % Longditude difference P, SSP

            % To Earth centered
            [lambda_E, Phi_E] = this.toUnitCentered(Lat_P_E, Lat_SSP_E, DL_earth);
            this.earth_center_co = [lambda_E, Phi_E]; % save intermediate result

            % To spacecraft centered
            eta = atan((sin(this.rho).*sin(lambda_E))./(1-sin(this.rho).*cos(lambda_E)));
            el = pi/2 - lambda_E - eta;
            Phi_SC = mod(-Phi_E, 2*pi);

            if (this.c_filerElevations)              % Set elevation filter
                idx = (el > this.c_elevarionCutOff); % Remove all elevations below the cutoff
            else
                idx = true(size(el));                % No elevations are filtered
            end
            this.accepted = idx; % Save the indexes that satisfy the elevation cutoff

            % Remove the points under the cutoff
            eta = eta(idx);
            Phi_SC = Phi_SC(idx);
            Lat_nadir_SC = Lat_nadir_SC(idx);
            Lon_nadir_SC = Lon_nadir_SC(idx);

            % To spacecraft lat, long
            [Lat_P_SC, Lon_P_SC] = this.fromUnitCentered(eta, Phi_SC, Lat_nadir_SC, Lon_nadir_SC);
            this.sc_co = [Lat_P_SC, Lon_P_SC, Lat_nadir_SC, Lon_nadir_SC]; %save the final values
        end
        %Convert from the Satillite celestial sphere to the earth celstial sphere
        function sat2earth(this)
            Lat_P_SC =   this.sc_co(:,1);
            Lon_P_SC =   this.sc_co(:,2);
            Lat_nadir =  this.sc_co(:,3);
            Lon_nadir =  this.sc_co(:,4);
            Lat_SSP  =   -Lat_nadir;        % Latitude flips
            Lon_SSP  =   0;                 % definition, lon=0

            DL_SC = mod(Lon_nadir - Lon_P_SC, 2*pi);

            % To sat centered
            [eta, Phi_SC] = this.toUnitCentered(Lat_P_SC, Lat_nadir, DL_SC);
            el = acos(sin(eta)./sin(this.rho));
            this.sc_center_co = [eta, Phi_SC, el];   % Save intermediate result

            if (this.c_filerElevations)              % Set elevation filter
                idx = (el > this.c_elevarionCutOff); % Remove all elevations below the cutoff
            else
                idx = true(size(el));                % No elevations are filtered
            end
            this.accepted = idx;  % Save the indexes that satisfy the elevation cutoff

            % Remove the points under the cutoff
            eta = eta(idx);
            Phi_SC = Phi_SC(idx);
            Lat_SSP = Lat_SSP(idx);
            Lon_SSP = Lon_SSP(idx);

            % To earth centered
            lambda = pi/2 - eta - el(idx);
            Phi_E = -Phi_SC;

            % To earth lat, long
            [Lat_P_E, Lon_P_E] = this.fromUnitCentered(lambda, Phi_E, Lat_SSP, Lon_SSP);
            this.earth_co = [Lat_P_E, Lon_P_E, Lat_SSP, Lon_SSP]; %save the final values
        end
        %% Conversion functions
        function [lambda, Phi] = toUnitCentered(~, Lat_P, Lat_Dir, DeltaLon)
             % Convert the given latitudes and dLong to celestial sphere
             % centered coordinates (both earth or sat)
            lambda = acos(sin(Lat_P).*sin(Lat_Dir) + ...
                          cos(Lat_P).*cos(Lat_Dir).*cos(DeltaLon));
            Phi    = acos2((sin(Lat_P)-sin(Lat_Dir).*cos(lambda))./ ...
                          (cos(Lat_Dir).*sin(lambda)), H(DeltaLon));
        end
        function [Lat_P, Lon_P] = fromUnitCentered(~, eta, Phi, Lat_Dir, Lon_Dir)
             % Convert the given celestial sphere coordinates
             % to latitude and longditude (both earth or sat)
            Lat_P = asin(sin(Lat_Dir).*cos(eta) + ...
                         cos(Lat_Dir).*sin(eta).*cos(Phi));
            DL = acos2((cos(eta)-sin(Lat_P).*sin(Lat_Dir)) ./ ...
                          (cos(Lat_P).*cos(Lat_Dir)), H(Phi));
            Lon_P = mod(Lon_Dir - DL, 2*pi);
        end
    end

end
```

Listing 4: H.m: OCDM hemisphere function

```
function [ H ] = H( phi )
    %H Hemisphere function
    %       H(phi) = +1 if (0   <= ?modulo360 < 180 [deg])
    %       H(phi) = -1 if (180 <= ?modulo360 < 360 [deg])
    %       Based on OCDM p389 eq 8-1
    % By: Simon Billemont (s.billemont@student.tudelft.nl), on: 12-2010 (mm-yyyy)
    % Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.
    phi = mod(phi, 2*pi);

    H = ones(size(phi));
    H(pi <= phi & phi < 2*pi) = -1;
end
```

Listing 5: acos2.m: OCDM acos2 function

```
function [ acos2 ] = acos2( cosPhi, H )
    %ACOS2 Quadrant sensative arcCosine, uses hemisphere function H
    %     acos2[cos(phi),H(phi)] = {H(phi) acos(cos(phi)}modulo360
    %     Based on OCDM p389 eq8-2
    % By: Simon Billemont (s.billemont@student.tudelft.nl), on: 12-2010 (mm-yyyy)
    % Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.
    if (any(cosPhi > 1))
        warning('MATLAB:acos2:bounds', ...
            'cosPhi values where clamped to [0, 1] <min:%f max:%f>\n', min(cosPhi), max(cosPhi));
        cosPhi(cosPhi > 1) = clamp(cosPhi(cosPhi > 1),0,1);
    end
    acos2 = mod(H .* real(acos(cosPhi)), 2*pi);
end
```

Listing 6: globePlot.m: 3D plotting utility for data on globes

```
function [ h ] = globePlot( varargin )
%GLOBEPLOT plot stuff on a globe
%   Provides 3d ploting functions
% By: Simon Billemont (s.billemont@student.tudelft.nl)
% Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.

plotCommands = splitStack(varargin{:});
hold on
    colormap('default')
    for i=1:length(plotCommands)
        hold on
        command = plotCommands{i};
        func = command{1};
        func(command{2:end})
    end
    hold on
    sphere(18) % Add a sphere
    colormap('white') % Grey spheres not to intervene with the colors of the plot
hold off
view(45,10) % Rotate the view of the camera
axis equal  % doent squeeze the earth
axis off    % Axis are useless, unit sphere
colorbar off
end

function commandStack = splitStack(varargin)
    cmdName =    {'point',   'segment'};
    cmdCommand = {@pointPlot, @segmentPlot};

    commandStack = {};
    i = 1;
    while(i <= length(varargin))
        currentCommand = cmdCommand(findCell(cmdName, varargin{i})); i=i+1;
        while (i <= length(varargin) && ~findCellContains(cmdName, varargin{i}))
            currentCommand(end+1) = varargin(i); i=i+1;
        end
        commandStack(end+1) = {currentCommand};
    end
end

function pointPlot(varargin)
    point = varargin{1};
    [x,y,z] = sph2cart(point(2), point(1), point(3));
    scatter3(x,y,z, varargin{2:end})
end
function segmentPlot(varargin)
    coordinates = varargin{1};
    [x,y,z] = sph2cart(coordinates(:,2), coordinates(:,1), coordinates(:,3)); % Convert az, el, 1 => x y z
    % Plot the points in 3d space, with color information the sample nr
    plot3k([x';y';z']', [1:length(x)]', varargin{2:end})
end
```

Listing 7: EARTHCOV3.m: Computes celestial sphere areas

```
%% By: Simon Billemont, sbillemont, 1387855
%   Contact: aodtorusan@gmail.com or s.billemont@student.tudelft.nl
%       Solve the Side-Angle-Side problem on a shperical triagle
%       Equations from Orbit & Constellation Design & Management (ocdm)
%   Made on: 06-12-2010 (dd-mm-yyyy)
%   This work is licensed under the
%       Creative Commons Attribution-NonCommercial 3.0 Unported License.
%       To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/
%% Setup environment
clc
clearvars
close all

% Make a new utility for saving pictures
saver = ImSav();
saver.c_plotsDir = '../images/';            % Where the plots will go
saver.c_changec_figSize = 0;                % Do not resize
saver.c_appendFigureNr = 0;                 % Do not append stuff the the name

%% Only procede if all the unit tests are succesfull
if (~runtests)
    error('There are errors in the test cases. Fix these first')
end

%% Setup constants
global Re H;
Re = 6371E3; % m
H  = 1000E3; % m

segments = [
    45, 15, 20;
    80, 20, 30;
    20, 28, 30;
    50, 28, 30;
]; % deg

%% Compute

[ aEarth, aSat ] = annulusSegment( deg2rad(segments) );

fprintf(['\t\\item $\\Phi=%d^{\\circ}\t,\\lambda_1=%d^{\\circ}\t,\\lambda_2=%d^{\\circ}:', ...
         ' A_{earth}=%.4fsr\t,A_{sat}=%.4fsr$\n'], ...
            [segments, aEarth, aSat]');

% ASSIGNMENT EARTHCOV-3:
% Consider a situation like OCDM Figure 9-10. Assume the following
% characteristics for the various areas A-D:
%
% Compute the surface areas of A-D individually, on an Earth-centered sphere,
% as well as on the spacecraft-centered celestial sphere (expressed in
% steradians in both cases). Assume a satellite altitude of 1000 km.
```

Listing 8: annulusSegment.m: Computes area of ring segments on Earth and satellite

```
function [ aEarth, aSat ] = annulusSegment( Phi, l1, l2 )
%ANNULUSSEGMENT Compute the surface area of a given ring segment on both
%the Earth and satellite celstial sphere.
%   function [ aEarth, aSat ] = annulusSegment( Phi, l1, l2 )
%       Phi:    Segment length
%       l1:     Start radius
%       l2:     End radius
%     Note: uses global Re, h to determine the satille positioon wrt the earth
% By: Simon Billemont (s.billemont@student.tudelft.nl), on: 07-12-2010 (dd-mm-yyyy)
% Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.

global Re H;

if nargin == 1 % Vector input
    l1 = Phi(:,2);l2 = Phi(:,3);Phi = Phi(:,1);
end

sinRho = Re / (Re+H);                           % ocdm eq 9-2
eta1 = atan2(sinRho*sin(l1), 1-sinRho*cos(l1)); % ocdm eq 9-3
eta2 = atan2(sinRho*sin(l2), 1-sinRho*cos(l2)); % ocdm eq 9-3

aEarth = Phi .* (cos(l1) - cos(l2));            % ocdm table 9-1
aSat   = Phi .* (cos(eta1) - cos(eta2));

end
```

Listing 9: test.m: xUnit tests for EarthSatReferenceConverter

```
function test()
    global Re h
    Re = 6371E3; % m
    h  = 1000E3; % m
    initTestSuite;
end

function testEarth2Sat()
    global Re h
    % Test case sheet 29 Space Mission Design Earth coverage
    ssp =   deg2rad([20, 270]);
    point = deg2rad([40, 290]);

    %Lat_P-E,  Lon_P-E,  Lat_SSP-E,  Lon_SSP-E
    converter = EarthSatReferenceConverter([point, ssp], true);
    converter.setRho(Re, h);
    converter.compute();

    sat_co = rad2deg(converter.sc_co);
    % Check the lat and long of the point in the spacecraft system with that of
    % the sheets (sheet 29)
    assertVectorsAlmostEqual(sat_co(1:2), [28.72, 324.49], 'absolute', 0.02 )
end

function testSat2Earth()
    global Re h
    % Test case sheet 30 Space Mission Design Earth coverage
    ssp    =     deg2rad([20, 270]);
    nadir =      [-ssp(1), 0 ];
    scanline =  deg2rad(50);

    %Dual axis spiral param
    r1     = -nadir(1);        % latNadir + pi/2 + r1 = pi/2 => r1 = -latNadir
    r2     = pi/2 - scanline;  % Scanline 50  north of nadir, 90 = nardir=>Scan + r2
    phi1_0 = 0;                %
    phi2_0 = deg2rad(210);     % Position considered
    omega1 = 1;                % Does not matter t=0
    omega2 = 0;                % Single axis spiral

    % Find lon lat of P
    dap = DualAxisProblem(r1, r2, phi1_0, phi2_0, omega1, omega2);
    s = dap.eval();
    point = [s.d, s.a];% [elevation, azimuth]

    % Check the lat and long of the point are compute correct from the dual
    % axis spiral (note i use latitude not colatitude and long not DeltaLong)
    assertVectorsAlmostEqual(rad2deg(point), [90-58.03, 360-337.74], 'absolute', 0.05 )

    % Lat_P-E,  Lon_P-E,  Lat_SSP-E,  Lon_SSP-E
    converter = EarthSatReferenceConverter([point, nadir], false);
    converter.setRho(Re, h);
    converter.compute();

    earth_co = converter.earth_co;        % Retrieve the coordinates
    % fix the ssp as it cannot be determined from nadir
    earth_co(:,[2 4]) = mod(earth_co(:,[2 4])+ssp(2), 2*pi);
    earth_co = rad2deg(earth_co);         % fancy degrees

    % Check the lat and long of the point in the earth system with that of
    % the sheets (sheet 30) (no solution given for lon_ssp
    assertVectorsAlmostEqual(earth_co(1:3), [36.17, 261.59, 90-70], 'absolute', 0.05 )
end
```

Listing 10: DualAxisProblem.m: Solves the duals axis spiral problem

```
classdef DualAxisProblem < handle
%DUALAXISPROBLEM Solve the Dual-Axis spiral problem
%   Find the angles and rates of point P in function of the given times
%   For details see Orbit & Constellation Design & Management (ocdm) chapter 8
% By: Simon Billemont (s.billemont@student.tudelft.nl), on: 16-11-2010 (dd-mm-yyyy)
% Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.
%
    properties (Access=public)
        r1      % Initial condition; distance C => S [rad]
        r2      % Initial condition; distance S => P [rad]
        phi1_0  % Initial condition; start angle around C between North and S [rad]
        phi2_0  % Initial condition; start angle around S between North and P [rad]
        omega1  % Rate of ratation of S around C [rad/s]
        omega2  % Rate of ratation of P around S [rad/s]
    end

    methods (Access=public)
        function obj = DualAxisProblem(r1, r2, phi1_0, phi2_0, omega1, omega2)
            % Constructor, sets the IC, based on the given values
            if nargin == 1 % Seperate values given
                obj.r1 = r1(1);             obj.r2 = r1(2);
                obj.phi1_0 = r1(3);         obj.phi2_0 = r1(4);
                obj.omega1 = r1(5);         obj.omega2 = r1(6);
            else % Values given as a vector
                obj.r1 = r1;                obj.r2 = r2;
```

```
                obj.phi1_0 = phi1_0;        obj.phi2_0 = phi2_0;
                obj.omega1 = omega1;        obj.omega2 = omega2;
            end
        end
        function s = eval(obj, t)
            % Find the angles and rates of P for the given timesamples
            if nargin==1
                t=0; % Default time sample
            end

            s = struct(); % Stores all the results
            % Compute the angles and rates in the order descibed in slide 36
            s.phi1 = obj.azimuthS_C(t);
            s.phi2 = obj.azimuthP_S(t);
            s.d =    obj.elevationP_C(s.phi2);
            s.da =   obj.changeAzimuthP_C(s.d, s.phi2);
            s.a =    obj.azimuthP_C(s.phi1, s.da);
            s.dE_ =  obj.angleC_E(s.phi1);
            s.rE =   obj.angleP_E(s.dE_, s.d, s.da);
            s.wE =   obj.rotationE(s.phi1);
            s.v =    obj.velocityP(s.wE, s.rE);
            s.dPsi = obj.changeMotionP(s.dE_, s.rE, s.d, s.da);
            s.psi =  obj.directionP(s.dPsi);
        end
    end

    methods (Access=private)
        %% Intermediate values
        function phi1 = azimuthS_C(obj, t)
            % Azimuth of S around C relative to alpha = 0; ocdm eq 8-27
            phi1 = obj.phi1_0 + obj.omega1 .* t;
        end
        function phi2 = azimuthP_S(obj, t)
            % Azimuth of P around S realtive to C; ocdm eq 8-27
            phi2 = obj.phi2_0 + obj.omega2 .* t;
        end
        function da = changeAzimuthP_C(obj, d, phi2)
            % Delta alpha: Change in azimuth of P around C; ocdm eq 28-a
            da = acos2( ...
                (cos(obj.r2)-cos(obj.r1).*sin(d))./(sin(obj.r1).*cos(d)), ...
                -H(phi2));
        end
        function rE = angleP_E(obj, dE_, d, da)
            % rho E: Angle from P to E; ocdm eq  8-24
            rE = acos(cos(dE_).*sin(d)+sin(dE_).*cos(d).*cos(da));
        end
        function wE = rotationE(obj, phi1)
            % omega E: Rate of ratation about E; ocdm eq  8-23b
            wE = ones(size(phi1)) * ...
                    sqrt(obj.omega1.^2 +obj.omega2.^2 + 2.*obj.omega1.*obj.omega2.*cos(obj.r1));
        end
        function dE_ = angleC_E(obj, phi1)
            % Angle from C to E; ocdm eq  8-23a
            dE_ = ones(size(phi1)) * mod( ...
                    atan((obj.omega2.*sin(obj.r1))./(obj.omega1+obj.omega2.*cos(obj.r1))) ...
                    , pi);
        end
        function dPsi = changeMotionP(obj, dE_, rE, d, da)
            % Change in direction of motion of P; ocdm eq  8-25a
            dPsi = acos2( ...
                (cos(dE_)-cos(rE).*sin(d))./(sin(rE).*cos(d)), ...
                H(da));
        end
        %% Results
        function a = azimuthP_C(obj, phi1, da)
            % Azimuth of P around C (final  azimuth); ocdm eq  8-28b
            a = mod(phi1 + da, 2.*pi);
        end
        function d = elevationP_C(obj, phi2)
            % Elevation of P relative to C (final elevation); ocdm eq 8-28c
            d = pi./2 - acos(cos(obj.r1).*cos(obj.r2)+sin(obj.r1).*sin(obj.r2).*cos(phi2));
        end
        function v = velocityP(obj, wE, rE)
            % Velocity of P; ocdm eq  8-26
            v = wE .* sin(rE);
        end
        function psi = directionP(obj, dPsi)
            % Direction of motion of P; ocdm eq  8-25b
            psi = mod(dPsi - pi./2, 2.*pi);
        end
    end

end
```