AE4879 Mission Geometry and Orbit Design

Assignment 8: Full-sky geometry

FULLSKY-2 and FULLSKY-5

Simon Billemont

November 19, 2010

Assignment 8: Full-sky geometry

This assignment deals with familiarization of using full-sky spherical geometry in order to solve problems instead of the traditional techniques. This document describes two implementations. The first concerns the airplane problem, of finding the direction back to a specific point. The second problem deals with a dual-axis spiral problem, with considering a satellite with a rotating sensor.

1. Airplane problem

Find all the solutions to airplane problem with the given parameters:

- **D1 = 30°, f1 = 300°, D2 = 20°**
- **D1 = 30°, f1 = 300°, D2 = 200°**

In the airplane problem, a fictitious airplane starts from a known position on the earth (or general sphere). It then flies in a random direction for a given distance (or angle) over the sphere. The the plane turns over a known angle, and continues on straight for another fixed distance. Then the plane must return to its starting point. The problem now arises to where it must turn and how far it must then fly in order to reach the starting point.

In this problem, the plane is on a sphere, and thus the application of spherical triangles is the logical solution. This specific problem is also known as a side-angle-side triangle, as these are known. This means that three quantities are known, and this is enough to compute all the other angles and sides of the spherical triangle.

The solution to this problem can easily be found using general full-sky geometry equations ([1] p390). The solution is summarized in eq 1-3. Here it is considered that B is the starting point, and the plane flies via side a to point C. Then onwards to point A over b and flies home via side c.

$$c^{(1)} = \mathrm{cos2}^{-1}[\cos a \cos b + \sin a \sin b \cos C, H(C)] \tag{1}$$

$$A^{(1)} = \mathrm{cos2}^{-1}\left[\frac{\cos a - \cos b \cos c^{(1)}}{\sin b \sin c^{(1)}}, H(a)\right] \tag{2}$$

$$B^{(1)} = \mathrm{cos2}^{-1}\left[\frac{\cos b - \cos a \cos c^{(1)}}{\sin a \sin c^{(1)}}, H(b)\right] \tag{3}$$

Where H is the hemisphere function and $\mathrm{cos2}^{-1}$ is a quadrant sensitive inverse cosine function defined as:

$$H(\varphi) = \begin{cases} +1 & 0 \leq \varphi \bmod (2\pi) < \pi \\ -1 & \pi \leq \varphi \bmod (2\pi) < 2\pi \end{cases} \tag{4}$$

$$\mathrm{cos2}^{-1}(\cos(\varphi), H(\varphi)) = \big(H(\varphi)\cos^{-1}(\cos(\varphi))\big) \bmod (2\pi) \tag{5}$$

Name	$C[°]$	$A[°]$	$B[°]$	$a[°]$	$b[°]$	$c[°]$
1.1	270.0	120.6	143.9	30.0	20.0	324.5
1.2	270.0	300.6	323.9	30.0	20.0	35.5
2.1	300.0	98.1	137.4	30.0	20.0	334.1
2.2	300.0	278.1	317.4	30.0	20.0	25.9
3.1	300.0	81.9	317.4	30.0	200.0	205.9
3.2	300.0	261.9	137.4	30.0	200.0	154.1

Table 1: Results for the three cases of the airplane problem

Note that there is also a second solution when the plane flies the other way around the globe. This solution is described by the following equations:

$$c^{(2)} = 2\pi - c^{(1)} \tag{6}$$

$$A^{(2)} = (A^{(1)} + \pi) \bmod (2\pi) \tag{7}$$

$$B^{(2)} = (B^{(1)} + \pi) \bmod (2\pi) \tag{8}$$

This calculation was done for three distinct cases, and the results can be found in table 1.

1. Test case presented in [2] slide 19
2. a = 30°, C = 300°, b = 20°
3. a = 30°, C = 300°, b = 200°

2. Rotating sensor on a spinning spacecraft

Write a program that simulates an arbitrary dual-axis problem. Verify the contents of the table on sheet 37. Then consider a rotating sensor on a satellite which is spinning itself (ρ_1 = ρ_2 = 90°). Apply the concept of full-sky spherical geometry to derive the geometry (i.e. the path of elevation vs. azimuth) of the observations of the sensor.

- **Assess the coverage of the celestial sphere for the case that ρ_2 = ρ_1.**
- **Assess the coverage of the celestial sphere for the case that ρ_2 = $1.01 \cdot \rho_1$.**
- **Plot the coverage of the celestial sphere after 50 and 100 revolutions.**
- **Compare and discuss the results of the previous (sub)questions.**

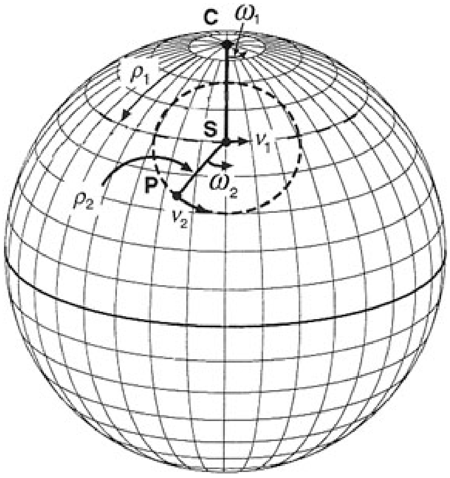

Figure 1: Geometry of the dual-axis spiral (from [1] Fig.8-11 p398)

In this section, a stable spacecraft is considered, spinning around a single axis. Furthermore, this spacecraft has a sensor, also spinning around is axis. Suppose that these spin axis are separated by an angle of 90°, and the angle from which the rotating sensor is scanning is also 90°. This configuration makes it possible for the sensor to scan the entire sky. In order to reconstruct path of the field of view (FOV) of the sensor, one can make use of a dual-axis spiral.

The dual axis spiral is basically one point P rotating about another point S, that in turn is rotating about again another point C (see fig 1). When applying these points to the satellite problem, there then is P representing the direction of the FOV. That is rotating about the sensor axis (S). Since the satellite is spinning, S rotates about the spin axis C. For more details about the dual-axis spiral see [1] section 8.2.

From this satellite configuration, one can formulate the angles ρ_1 (angle sensor, satellite spin axis) and ρ_2 (angle FOV with sensor spin axis) as 90°. Lastly defining ω_1 as the spin rate of the satellite and ω_2 as the sensor spin rate. Using [1] Table 8-8 as a guide, all the sides and angles in the spherical triangle can be found:

$$\varphi_1 = \varphi_{1,0} + t\omega_1 \tag{9}$$

$$\varphi_2 = \varphi_{2,0} + t\omega_2 \tag{10}$$

$$\Delta\alpha = \cos 2^{-1}\left(\frac{\cos(\rho_2) - \sin(\delta)\cos(\rho_1)}{\cos(\delta)\sin(\rho 1)}, -H(\varphi_2)\right) \tag{11}$$

$$\rho_e = \cos^{-1}\left(\cos(\delta)\cos(\Delta\alpha)\sin(\Delta\mathrm{E}') + \sin(\delta)\cos(\Delta\mathrm{E}')\right) \tag{12}$$

$$\omega_e = \sqrt{2\omega_2\omega_1\cos(\rho_1) + \omega_1^2 + \omega_2^2} \tag{13}$$

$$\Delta\mathrm{E}' = \tan^{-1}\left(\frac{\omega_2\sin(\rho_1)}{\omega_1 + \cos(\rho_1)\,\omega_2}\right) \bmod \pi \tag{14}$$

Result	$\varphi_1[°]$	$\varphi_2[°]$	$\delta[°]$	$\Delta\alpha[°]$	$\alpha[°]$	$\delta'_E[°]$	$\rho_e[°]$	$\omega_e[rad/s]$	$v[rad/s]$	$\Delta\psi[°]$	$\psi[°]$
$test-1$	0.00	90.00	46.04	330.48	330.48	40.00	20.00	3.00	1.03	292.18	202.18
$test-2$	0.00	90.00	46.04	330.48	330.48	30.31	22.14	3.82	1.44	318.70	228.70
$test-3$	0.00	100.00	42.97	332.59	332.59	30.31	23.61	3.82	1.53	324.54	234.54

Table 2: Testing and verification results

$$\Delta\psi = \cos2^{-1}\left(\frac{\cos\left(\Delta\mathrm{E}'\right) - \sin(\delta)\cos\left(\rho_e\right)}{\cos(\delta)\sin\left(\rho_e\right)}, H(\Delta\alpha)\right) \tag{15}$$

Using these intermediate values, the actual field of view path can be found in terms azimuth (α) and elevation (δ). Furthermore also the instantaneous velocity of the FOV over the sky sphere v and the direction of motion of the FOV ψ

$$\alpha = (\Delta\alpha + \varphi_1) \bmod (2\pi) \tag{16}$$

$$\delta = \frac{\pi}{2} - \cos^{-1}\left(\sin\left(\rho_1\right)\sin\left(\rho_2\right)\cos\left(\varphi_2\right) + \cos\left(\rho_1\right)\cos\left(\rho_2\right)\right) \tag{17}$$

$$v = \omega_e \sin\left(\rho_e\right) \tag{18}$$

$$\psi = \left(\Delta\psi - \frac{\pi}{2}\right) \bmod (2\pi) \tag{19}$$

To test the workings of these equations, several test cases presented in [2] slide 37 where re-simulated. The results of this testing procedure where near identical to the results found on the slide (± 0.02°) and can be found in table 2.

To evaluate the coverage of the satellite considered in this problem, ω_1 and ω_2 need to be defined. We consider two distinct cases. The first where both are equal and a second where they are non equal (eg $\omega_2 = 1.01\omega_1$). For both, the coverage was plotted (see fig 2 to 6)

When both rotation speeds are equal, one notices that a pure 8 figure is traced by the FOV of the sensor (see fig 2). This means that only a small section of the sky sphere is visible to the sensor. The figure can easily be explained. If the sensor of is facing a particular direction, the satellite ten turns 180°. The the senor is looking at the opposite site it was looking before.However not only the satellite spins, also the sensor itself. Since both rotation rates are equal, the sensor should also rotate 180°, meaning that it is facing the same azimuth again, but inverted elevation.

When the rotation speeds are not equal, instead of the sensor rotating 180°, it rotates a bit more (or less). This causes the sensor to just 'miss', creating a small offset (see fig 3). This small offset causes the sensor to systematically scan the entire sky. It still repeats the same track, after 100 rotations (when both ω_1 and ω_2 are integers again).

What can also be seen is that after 50 revolutions, the sensor is scanning 180°further then when it started. This because it has a repeat orbit of 100 revolutions. Furthermore, when looking at 50 rotations, one notices that the sensor scanned $3/4$ of the sky, but of that area, it scanned $1/4$ of the sky twice. When increasing the revolutions to 100 (when the sensor starts repeating its track precisely), it scanned the entire sky twice.

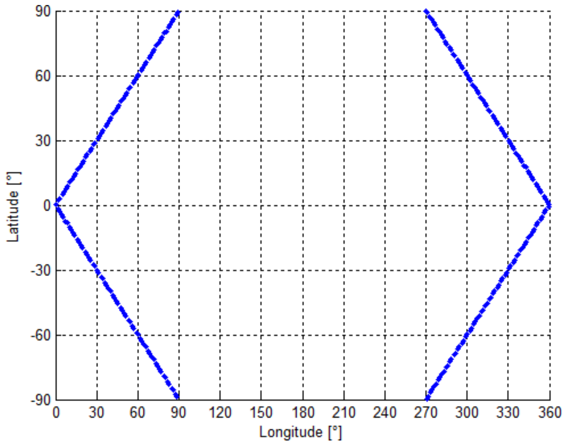

Figure 2: Rotating sensor on a spinning spacecraft, $\omega_2 = \omega_1$, 2 revolutions

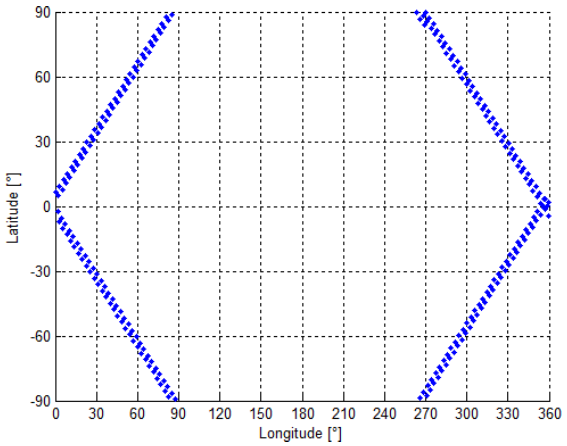

Figure 3: Rotating sensor on a spinning spacecraft, $\omega_2 = 1.01\omega_1$, 2 revolutions

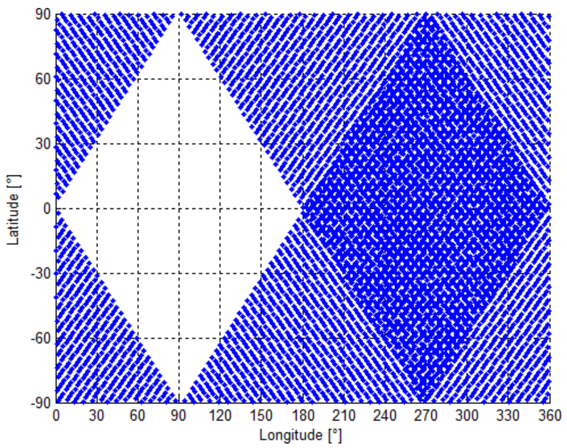

Figure 4: Rotating sensor on a spinning spacecraft, $\omega_2 = 1.01\omega_1$, 50 revolutions

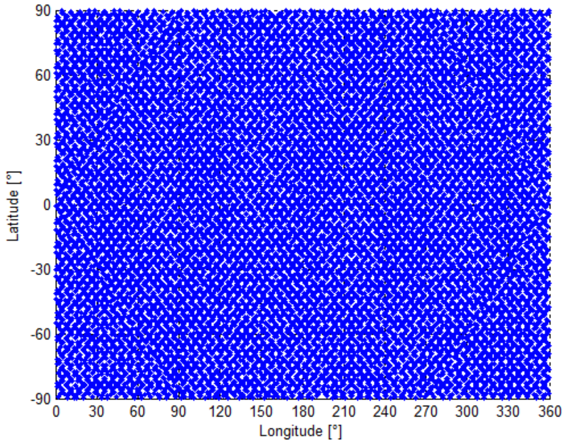

Figure 5: Rotating sensor on a spinning spacecraft, $\omega_2 = 1.01\omega_1$, 100 revolutions

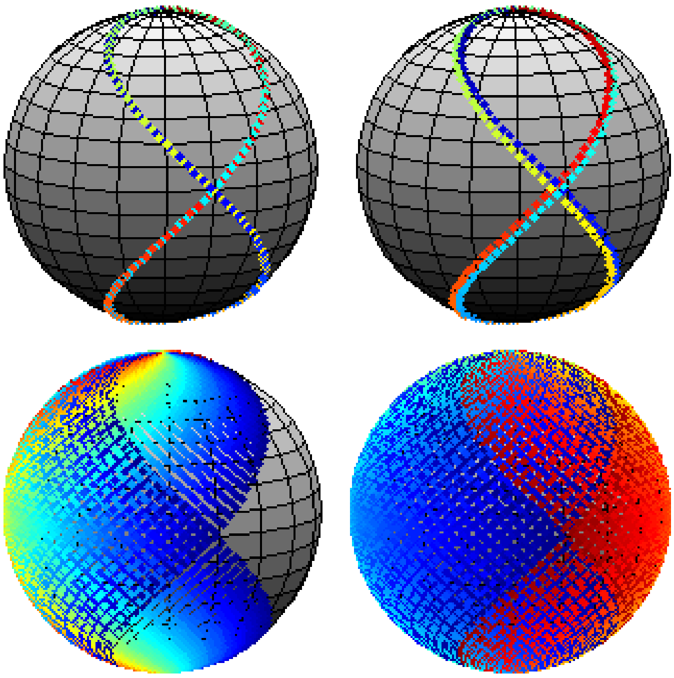

Figure 6: 3D overview for the 4 different cases, the color represents sample epoch (blue to red)

References

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

[2] R. Noomen, *AE4-879 Full-sky spherical geometry V3.1*, TUDelft Lecture Slides, 2010.

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

Additional information

Estimated work time:
$\sim$ 3h Studying theory + $\sim$ 3h making assignment + $\sim$ 4h writing report = $\sim$ 10h

Made by

Simon Billemont

Stud Nr: 1387855

s.billemont@student.tudelft.nl

License and notices

A. Matlab source code

The code written to implement the three described optimizers was written in MATLAB 7.11 (2010b)[3]. A structured overview of the dependencies is given below:

- FULLSKY2.m
  - acos2.m
  - H.m
- FULLSKY5.m
  - DualAxisProblem.m
  - acos2.m
  - H.m

The script FULLSKY2.m solves the spherical triangle side-angle-side problem and FULLSKY5.m solves the spinning spacecraft problem. It does this using a dual-axis spiral problem solver (DualAxisProblem.m). Then there are the specific spherical triangle function acos2.m and H.m.

Listing 1: FULLSKY2.m: Find the set of solutions for a spherical triangle size-angle-side

```
%% 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: 15-11-2010 (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/matlab/';     % Where the plots will go
saver.cleanPlots;                           % Remove any plots already made
saver.c_changec_figSize = 0;                % Do not resize
saver.c_appendFigureNr = 0;                 % Do not append stuff the the name

%% Setup constants
a = 30  * (pi/180); % D1,   side
C = 300 * (pi/180); % phi1, angle
b = 20  * (pi/180); % D2,   side

%% Compute
% First set of results (ocdm p390 eq 8-4a to 8-4c)
c(1) = acos2( cos(a)*cos(b) + sin(a)*sin(b)*cos(C)    , H(C));
A(1) = acos2((cos(a) - cos(b)*cos(c))/(sin(b)*sin(c)), H(a));
B(1) = acos2((cos(b) - cos(a)*cos(c))/(sin(a)*sin(c)), H(b));

% Second set of results (ocdm p390 eq 8-5a to 8-5c)
c(2) = 2*pi-c(1);
A(2) = mod(A(1) + pi,2*pi);
B(2) = mod(B(1) + pi,2*pi);

%% Fix units to degrees
a = a * (180/pi);b = b * (180/pi);c = c * (180/pi);
A = A * (180/pi);B = B * (180/pi);C = C * (180/pi);

%% Output in latex table form
line = '\t\t &\t %.1f &\t %.1f &\t %.1f &\t %.1f &\t %.1f &\t %.1f \t \\\\ \\hline \n';
fprintf(line, C(1), A(1), B(1), a(1), b(1), c(1));
fprintf(line, C(1), A(2), B(2), a(1), b(1), c(2));
```

Listing 2: acos2.m: Compute the quadrant sensitive arc cosine

```
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, on: 15-11-2010 (dd-mm-yyyy)
%   Licence: Creative Commons Attribution-NonCommercial 3.0 Unported License.

acos2 = mod(H .* acos(cosPhi), 2*pi);

end
```

Listing 3: H.m: 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, on: 15-11-2010 (dd-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 4: FULLSKY5.m: Solve the rotating sensor/spinning spacecraft problem

```
%% By: Simon Billemont, sbillemont, 1387855
%   Contact: aodtorusan@gmail.com or s.billemont@student.tudelft.nl
%       Find the path of the FOV of a sensor on a spinning spacecraft, with
%       a rotating sensor.
%       Equations from Orbit & Constellation Design & Management (ocdm)
%   Made on: 16-11-2010 (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_saveFigure = 0;                     % Disable it to not overwrite
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

%% Setup constants

labels = {'test-1','test-2','test-3', 'sensor-equal', ...
    'sensor-non-equal-2', 'sensor-non-equal-50', 'sensor-non-equal-100'};

%        r1, r2, phi1_0, phi2_0, omega1, omega2
param = [40, 20,   0,     90,     0,      3;         %test
         40, 20,   0,     90,     1,      3;         %test
         40, 20,   0,     100,    1,      3;         %test
         90, 90,   0,     0,      1,      1;         %w2 = 1    w1
         90, 90,   0,     0,      1,      1.01];     %w2 = 1.01 w1
param(:,1:4) = param(:,1:4) * pi/180; % Convert the first 4 rows to rad

%% Tests

for i=1:3
    % Create the dual-axis spiral problem with the given test parameter
    dap = DualAxisProblem(param(i,:));
    % Compute all the angles and rates (for t=0, default)
    results(i) = dap.eval();
end
% Output in latex table form
for i=1:length(results)
    fprintf('\t\t %s & ', labels{i});
    s = results(i);
    fn = fieldnames(s);
    for n = 1:length(fn)
        if (any(strcmp(fn{n}, {'wE', 'v'})))
            fprintf('%.2f & ', s.(fn{n}));
        else
            fprintf('%.2f & ', rad2deg(s.(fn{n})));
        end
    end
    fprintf('\b\b \\\\ \\hline \n');
end

%% Rotating sensor
```

```
% w2 = w1
n = 2;                                  % Rotations
t = 0:0.05:n*param(4,5)*2*pi;           % Individual epochs
dap = DualAxisProblem(param(4,:));      % Dual-axis problem solver
s(1) = dap.eval(t);                     % Get the resulting variables

% w2 = 1.01 w1
rotations = [2 50 100];                 % Compute for all these rotations
for n=rotations
    t = 0:0.05:n*param(4,5)*2*pi;       % Time samples ...
    dap = DualAxisProblem(param(5,:)); % Solver with the correct initial conditions
    s(end+1) = dap.eval(t);             % Get the resulting variables
end

%% 2D plot
% Make a plot of long vs lat (az vs el)
for i=1:length(s)
    figure
    % Make the plot of all the points that where tracked
    scatter(rad2deg(s(i).a) , rad2deg(s(i).d), '.');
    % Fix the plot axis range and ticks (looks better this way)
    set(gca, 'YTick', -90:30:90)
    set(gca, 'YLim',  [-90, 90])
    set(gca, 'XTick', 0:30:360)
    set(gca, 'XLim',  [0, 360])
    grid on
    xlabel('Longitude [ ]')
    ylabel('Latitude [ ]')
    saver.saveImage(labels{3+i});       % Save it to disk
    close
end

%% 3D plot (globes)

figure
for i=1:length(s)
    subplot(2,length(s)/2,i) % Put each globe in a subplot
    [x,y,z] = sph2cart(s(i).a, s(i).d, ones(size(s(i).a))); % 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))
    hold on
        sphere % Add a sphere
    hold off
    view(70,10) % Rotate the view of the camera
    axis equal  % doent squeeze the earth
    axis off    % Axis are useless, unit sphere
    colorbar off
end
colormap('gray') %Grey spheres not to intervene with the colors of the plot
saver.saveImage('globes') % Save it to disk
```

Listing 5: DualAxisProblem.m: Generic solver for dual-axis spiral problems

```
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
```