Technical University of Denmark

Xiao Hu

# 30550: Satellite Based Positioning

## Report for assignment A-C

September 23, 2018

**DTU Space**
National Space Institute

**30550: Satellite Based Positioning, Report for assignment A-C**

**Author(s):**
Xiao Hu

**Supervisor(s):**
Dr. Anna B. O. Jensen

**National Space Institute**
Technical University of Denmark
Elektrovej Building 328, room 007
2800 Kongens Lyngby
Denmark
www.space.dtu.dk

E-mail: xiahaa@space.dtu.dk

# ABSTRACT

This report will present the procedures, results accomplished for assignments A-C which mainly includes transformation among common used coordinate frames and prediction of satellite position using Kepler theorem.

# Table of Contents

# 1 Assignment A: Geographic and Cartesian Coordinates

The main objective of the assignment is to understand the two commonly used coordinate frames for satellite-based positioning and grasp the transformations between the two coordinate frames.

## 1.1 Theory

According to [1], the transformation from the ellipsoidal coordiante frame ($\phi,\ \lambda,\ h$) to the earth-centered, earth-fixed (ECEF) Cartesian coordinate frame ($x,\ y,\ z$) can be computed as follows:

$$N = \frac{a^2}{(1 - e^2 sin^2\phi)^{\frac{1}{2}}} \tag{1.1}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (N+h)cos(\phi)cos(\lambda) \\ (N+h)cos(\phi)sin(\lambda) \\ (N(1-e^2)+h)sin(\phi) \end{bmatrix} \tag{1.2}$$

where, $a$ is the semi-major axis of ellipsoid (6378137.0 m), $e$ is the eccentricity of the ellipsoid calculated by $e^2 = 2f - f^2$ with $f$ being the flattening ($\frac{1}{298.257223563}$). This transformation can be computed directly.

The backward transformation from the ECEF to the ellipsoidal coordinate frame is given by:

$$p = \sqrt{x^2 + y^2} \tag{1.3}$$

$$\begin{bmatrix} \phi \\ \lambda \\ h \end{bmatrix} = \begin{bmatrix} atan(\frac{z}{p(1-e^2\frac{N}{N+h})}) \\ atan(\frac{y}{x}) \\ \frac{p}{cos(\phi)} - N \end{bmatrix} \tag{1.4}$$

As can be seen (1.4), the $h$ and $\phi$ correlate with each other, which means it can only be solved iteratively.

## 1.2 Tasks

- Implement a Matlab function for conversion from Latitude, Longitude, and Height to X, Y, and Z (in WGS84), and vice versa.
- Test with positions selected from the internet.
- Benchmark the results with results converted by the KMSTrans[1] program.

[1] http://valdemar.kms.dk/trf/

## 1.3 Code

Complementary functions:

**deg2rad: conversion from degree to radian**:

```
function rad = deg2rad(deg)
%deg2rad   conversion from degree to radian.
%   rad = deg2rad(deg) produces a corresponding radian value to a
%   given degree value.
%   Author: xiahaa@space.dtu.dk
    rad = deg * pi / 180.0;
end
```

**rad2deg: conversion from radian to degree**:

```
function deg = rad2deg(rad)
%rad2deg   conversion from radian to degree.
%   deg = rad2deg(rad) produces a corresponding degree value to a
%   given radian value.
%   Author: xiahaa@space.dtu.dk

    deg = rad * 180.0 / pi;
end
```

**llhtoCartesian: conversion from latitide, longitude, altitude to Cartesian coordiantes**:

```
function [x,y,z] = llhtoCartesian(lat, lon, height, consParams)
%   [x,y,z] = llhtoCartesian(lat, lon, height, consParams) does the
%   conversion from latitude, longitude, altitude to the Cartesian
%   coordiantes in the ECEF coordiante system.
%   lat, lon, height: coordinates in the ellipsoidal coordinate system.
%   consParams: struct of relative constant parameters
%       a: length of semi-major axis in meters.
%       f: flattening of the ellipsoid.
%   Author: xiahaa@space.dtu.dk

    a = consParams.a;
    f = consParams.f;
    % fistly, convert f to e
    e = sqrt(2*f-f*f);
    rlat = deg2rad(lat);
    rlon = deg2rad(lon);
    e2 = e*e;
    % then compute N
    N = a / sqrt(1-e2*sin(rlat)*sin(rlat));
    % XYZ, zc = height*sin(rlat);
    hsum = N+height;
    x = hsum*cos(rlat)*cos(rlon);
    y = hsum*cos(rlat)*sin(rlon);
    z = (N*(1-e2)+height)*sin(rlat);
```

```
end
```

**cartesian2llh: conversion from Cartesian coordiantes to latitide, longitude, altitude**:

```
function [lat, lon, height] = Cartesian2llh(x,y,z,consParams)
%   [lat, lon, height] = Cartesian2llh(x,y,z,consParams) does the
%   conversion from Cartesian coordiantes to latitude, longitude,
%   altitude.
%   x, y, z: coordinates in the ECEF coordinate system.
%   consParams: struct of relative constant parameters
%       a: length of semi-major axis in meters.
%       f: flattening of the ellipsoid.
%   Author: xiahaa@space.dtu.dk

    a = consParams.a;
    f = consParams.f;
    % fistly, convert f to e
    e = sqrt(2*f-f*f);
    e2 = e*e;
    x2y2sqr = sqrt(x*x+y*y);

    iter = 1;
    vold = [0,0,0];
    vnew = zeros(1,3);
    tolerance = 1e-6;
    maxIter = 1e6;

    while iter < maxIter
        disp(strcat('iter:',num2str(iter)));
        N = a / sqrt(1-e2*sin(vold(1))*sin(vold(1)));
        % update
        vnew(1) = atan2(z, x2y2sqr*(1-e2*N/(N+vold(3)+1e-6)));
        vnew(2) = atan2(y,x);
        vnew(3) = x2y2sqr/cos(vnew(1)) - N;
        % check tolerance
        verr = vnew - vold;
        if norm(verr) < tolerance
            break;
        else
            vold = vnew;
            iter = iter + 1;
        end
    end
    % to deg
    lat = rad2deg(vnew(1));
    lon = rad2deg(vnew(2));
    height = vnew(3);
end
```

```matlab
function ex1_coordinate_transformation
    clear all;
    clc;
    consParams = struct('a',6378137.0,'f',1/298.257223563);

    %% add utils path
    addpath('../utils/')

    %% test 1, deci
    testCase(1,:) = [55.78575300466123,12.525384183973078,40];% DTU 101
    testCase(2,:) = [0,12.525384183973078,40];%equator
    testCase(3,:) = [90,0,40];
    testCase(4,:) = [-90,-15,40];

    testid = 1;% choose the desired point you want to test.

    lat = testCase(testid,1);% latitude
    lon = testCase(testid,2);% longitude
    height = testCase(testid,3);% altitude
    [x,y,z] = llhtoCartesian(lat, lon, height, consParams); % do forward transformation
    %% show
    disp(strcat('x: ',num2str(x),'; y: ',num2str(y),'; z: ',num2str(z)));
    %% reverse
    [latr, lonr, heightr] = Cartesian2llh(x,y,z,consParams);

    disp(strcat('true lat: ',num2str(lat),'; true lon: ',num2str(lon),'; true height: ',num2str(height)));% '; ellipsoid Height',num2str(heightr+N)));
    disp(strcat('lat: ',num2str(latr),'; lon: ',num2str(lonr),'; height: ',num2str(heightr)));% '; ellipsoid Height',num2str(heightr+N)));
    error = abs(lat-latr)+abs(lon-lonr)+abs(height-heightr);
    errorRMS = sqrt(mean(([lat, lon, height] - [latr, lonr, heightr]).^2));
    disp(strcat('error: ',num2str(error)));
    disp(strcat('RMS: ', num2str(errorRMS)));

    %% test case for deg minute second to decimal
    latd = 55;latm = 47;lats = 8.7108;
    lond = 12;lonm = 31;lons = 31.3824;
    lart = deg2deci(latd,latm,lats);
    lont = deg2deci(lond,lonm,lons);
    disp(strcat('lat: ', num2str(lat), '; latt', num2str(lart)));
    disp(strcat('lon: ', num2str(lon), '; lont', num2str(lont)));

end
```

## 1.4 Experiments

### 1.4.1 Simple Test

#### Setup

In this simple test, I manually selected several points among which there is one point representing the DTU 101, one point located on the equator, one point in the North pole and one in the South pole. Meanwhile, in order to validate the correctness given the inputs contain negative values, the fourth point contains negative values for latitude, longitude. Below is a table listing the four sample points.

***Table 1.1.*** Test cases

| **ID** | **{lati, longi, alti}** |
|---|---|
| 1 | {55.78575300466123,12.525384183973078,40} |
| 2 | {0,12.525384183973078,40} |
| 3 | {90,0,40} |
| 4 | {-90,-15,40} |

#### Result

***Table 1.2.*** Backward Results

| $LLA$ | $LLA$ by KMSTrans | $RMS$ with KMSTrans |
|---|---|---|
| {55.7858, 12.5254, 40} | {55.5626, 12.3355, 40} | 0.1692 |
| {0, 12.5254, 40} | {0, 12.3355, 40} | 0.1096 |
| {90, 0, 40} | {90.1478, 0, 40} | 0.0853 |
| {-90, -15, 40} | {N/A[2]} | N/A |

***Table 1.3.*** Backward Results

| **Original** $LLA$ | **Reconstructed** $LLA$ | $RMS$ |
|---|---|---|
| {55.7858, 12.5254, 40} | {55.7858, 12.5254, 40} | $2.1508e^{-09}$ |
| {0, 12.5254, 40} | {0, 12.5254, 40} | $5.377e^{-10}$ |
| {90, 0, 40} | {90, 0, 40} | 0 |
| {-90, -15, 40} | {-90, -15, 40} | $1.0256e^{-15}$ |

The forward result is shown in the following Table 1.4 where the first column lists the transformed $XYZ$ values, the second column lists the results from the KMSTrans website, the third column lists the error between the computed values and the results from the KMSTrans website in terms of Root-Mean-Squares $RMS$.
The backward result is shown in the Table 1.2, where the first column lists the re-computed values for latitude, longitude, altitude $LLA$, the second column lists the results from the KMSTrans website, the last column lists the error between the re-computed values and the refereed values from KMSTrans for latitude, longitude, altitude in terms of Root-Mean-Squares $RMS$. A further test is done to evaluate the accuracy of the backward

***Table 1.4.*** Forward Results

| $XYZ$ | $XYZ$ by KMSTrans | $RMS$ **with KMSTrans** |
|---|---|---|
| {3509064.2531, 779572.0321, 5251099.25} | {3509064.253, 779572.032, 5251099.25} | $8.1650e^{-05}$ |
| {6226376.5177, 1383248.822, 0} | {6226376.518, 1383248.822, 0.00} | $1.7321e^{-04}$ |
| {3.9186e-10, 0, 6356792.3142} | {0, 0, 6356792.31} | 0.0024 |
| {3.7851e-10, -1.0142e-10, -6356792.3142} | {N/A[a] } | N/A |

[a]KMSTrans does not support negative latitude, longitude inputs.

transformation: with given $LLA$, firstly do the forward transformation and then do the backward transformation, then take the difference. Theoretically, the reconstructed result should be the same as the original values. The real result is shown in Table 1.3.

### 1.4.2 Brute-force Test

In order to do more brute force tests conveniently, a MATLAB GUI[3] is designed and shown in Fig 1.1. Latitude, longitude, and altitude can be arbitrarily selected simply by sliding corresponding sliders. Transformation results will be shown once the "**Calculate**" button is pushed down. KMSTrans result (if available) can be automatically accessed by click "**web check**" button.
By playing with this GUI, I did a brute-force test. Generally speaking, if KMSTrans result is achievable, the results are similar with a small difference.

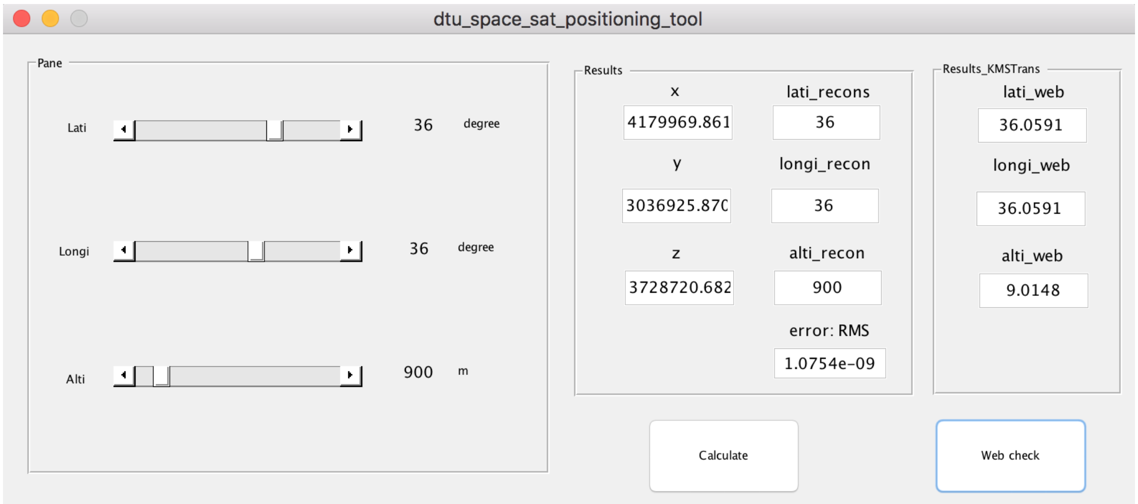


***Figure 1.1.*** Snapshot of the designed GUI.

### 1.4.3 Conclusion

Though various tests of my program, I believe that this program can correctly compute the coordinate transformation between the ECEF coordinate frame and the ellipsoidal coordinate frame.

[3]Code is not attached here since there are a lot of GUI related codes.

# 2 Assignment B: Satellite Coordinates from Kepler Elements

The main objective of the assignment is to know how to compute and predict the satellites' position by using Kepler elements.

## 2.1 Theory

According to [1], the orbital position of one satellite can be computed with

$$n = \sqrt{\frac{GM}{a^3}} \tag{2.1}$$

$$M = n(t - t_p) \tag{2.2}$$

$$M = E - esinE \tag{2.3}$$

$$\mathbf{r} = \begin{bmatrix} acosE - ae \\ a\sqrt{1-e^2}sinE \\ 0 \end{bmatrix} \tag{2.4}$$

where, $n$ is the mean motion, $GM$ is the earth's gravitational constant $3986004.418 * 10^8\ m/s^2$, $a$ is the length of the semi-major axis of ellipsoid (6378137.0 m), $e$ is the eccentricity of the ellipsoid calculated by $e^2 = 2f - f^2$ with $f$ being the flattening $(\frac{1}{298.257223563})$.

The transformation from the orbital coordinate frame to the ECEF coordinate frame can be computed using the inclination $i$, the right ascension of the ascending node $\Omega$ and the argument of perigee $\omega$.

$$\mathbf{r}_{ECEF} = \mathbf{R}(-\Omega)\mathbf{R}(-i)\mathbf{R}(-\omega)\mathbf{r}_{orbit} \tag{2.5}$$

## 2.2 Tasks

- Implement a Matlab function for estimating satellite positions in the orbital coordinate system.
- Convert the positions to the inertial system.
- Visualization of satellite positions for an interval.

## 2.3 Code

Complementary functions:

**deg2rad: conversion from degree to radian** as previous one.

**rad2deg: conversion from radian to degree** as previous one.
**consR: construct a rotation matrix**:

```
function R = consR(rad, axis)
%consR: construct a rotation matrix.
%   R = consR(rad, axis) return the corresponding rotation matrix.
%   rad: rotation angle.
%   axis: 1-x axis, 2-y axis, 3-z axis.
%   R: rotation matrix.
%   Author:xiahaa@space.dtu.dk
    if axis == 1
        R = [1 0 0; ...
             0 cos(rad) sin(rad); ...
             0 -sin(rad) cos(rad)];
    elseif axis == 3
        R = [cos(rad) sin(rad) 0; ...
            -sin(rad) cos(rad) 0; ...
            0 0 1];
    elseif axis == 2
        R = [cos(rad) 0 -sin(rad); ...
             0 1 0; ...
             sin(rad) 0 cos(rad)];
    end
end
```

**estimateEccAnomaly: estimate eccentric anomaly**:

```
function E = estimateEccAnomaly(M, e)
%estimateEccAnomaly: estimate eccentric anomaly iteratively.
%   E = estimateEccAnomaly(M, e) returns estimated eccentric anomaly.
%   M: mean anomaly.
%   e: eccentricity.
%   E: eccentric anomaly.
%   Author:xiahaa@space.dtu.dk
    E = 0;
    iter = 1;
    maxIter = 1e3;
    error = 1e6;
    while error > 1e-3 && iter <= maxIter
        newE = M + e*sin(E);
        error = abs(newE) - E;
        E = newE;
    end
end
```

**lutOmegaAndAnomaly: lookup ascension and anomaly**:

```
function [OmegaDeg, AnomalyDeg] = lutOmegaAndAnomaly(slotID)
%lutOmegaAndAnomaly: lookup ascension and anomaly
%   [OmegaDeg, AnomalyDeg] = lutOmegaAndAnomaly(slotID) returns
%   ascension and anomaly for a given satellite.
```

```
%   slotID: satellite ID, now support all 24 satellites of GPS.
%   OmegaDeg: ascension in degree.
%   AnomalyDeg: anomaly in degree.
%   Author: xiahaa@space.dtu.dk
    i = 0; j = 0;
    if slotID(1) == 'A' || slotID(1) == 'a'
        i = 1;
    elseif slotID(1) == 'B' || slotID(1) == 'b'
        i = 2;
    elseif slotID(1) == 'C' || slotID(1) == 'c'
        i = 3;
    elseif slotID(1) == 'D' || slotID(1) == 'd'
        i = 4;
    elseif slotID(1) == 'E' || slotID(1) == 'e'
        i = 5;
    elseif slotID(1) == 'F' || slotID(1) == 'f'
        i = 6;
    end
    j = str2num(slotID(2));

    %% lookup table
    lutOmega = [272.85, 332.85, 32.85, 92.85, 152.85, 212.85];
    lutAnomaly = [11.68 41.81 161.79 268.13; ...
                  80.96 173.34 204.38 309.98; ...
                  111.88 241.57 339.67 11.80; ...
                  135.27 167.36 265.45 35.16; ...
                  197.05 302.60 333.69 66.07; ...
                  238.89 345.23 105.21 135.35];

    OmegaDeg = lutOmega(i);
    AnomalyDeg = lutAnomaly(i,j);
end
```

**calcSatPosition: compute satellite positions**:

```
function [q1, q2, q3, i1, i2, i3] = calcSatPosition(slotID, t0, t)
%calcSatPosition: compute satellite position in orbital and CIRS coordinate frame
%   [q1, q2, q3, i1, i2, i3] = calcSatPosition(slotID, t0, t) compute
%   satellite position in orbital and CIRS coordinate frame.
%   slotID: satellite ID, now support all 24 satellites of GPS.
%   t0: initial time, 0.
%   t: time interval starting from t0.
%   q1, q2, q3: return value, satellite positions in orbital frame.
%   i1, i2, i3: return value, satellite positions in ECEF frame.
%   Author: xiahaa@space.dtu.dk
    a = 26559.8*1e3;% semi-major axis
    e = 0;% eccentricity
    inc = deg2rad(55); % GPS inclination
    argPerigee = deg2rad(0);% argument perigee
    % look-up corresponding information for given satellite
    [OmegaDeg, AnomalyDeg] = lutOmegaAndAnomaly(slotID);
    % conversion
```

```
    Omega = deg2rad(OmegaDeg);
    M0 = deg2rad(AnomalyDeg);
    % earth gravitational constant
    GM = 3986004.418*1e8;
    meanMotion = sqrt(GM)*a^(-3/2);% mean motion
    % mean anomaly
    M = M0 + meanMotion*(t-t0);
    % accentric anomaly
    E = estimateEccAnomaly(M, e);
    % orbital positions
    q1 = a*cos(E) - a*e;
    q2 = a*sqrt(1-e^2)*sin(E);
    q3 = 0;

    % constuct rotation matrix
    ROmega = consR(Omega,3);
    Rinc = consR(inc,1);
    RargPerigee = consR(argPerigee,3);

%    ROmega = [cos(Omega) sin(Omega) 0;
%              -sin(Omega) cos(Omega) 0;
%              0 0 1];
%    Rinc = [1 0 0;
%            0 cos(inc) sin(inc);
%            0 -sin(inc) cos(inc)];
%    RargPerigee = [cos(argPerigee) sin(argPerigee) 0;
%              -sin(argPerigee) cos(argPerigee) 0;
%              0 0 1];

    Rqs = RargPerigee * Rinc * ROmega;
    Rsq = Rqs';
    % transformation to ECEF
    v = Rsq * [q1;q2;q3];
    i1 = v(1);
    i2 = v(2);
    i3 = v(3);
end
```

**ExampleHelperSat: visualization.**:

```
classdef ExampleHelperSat < handle
%Create a simulated sat plot.
% Author xiahaa@space.dtu.dk.
    properties(SetAccess = public)
        %Trace Pose history of the sat
        Traces = [];
        len = 0;
        sats = [];
        init = 0;
    end

    properties(Access = private)
```

```
        %HTrajectory Graphics handle for the trajectory of the sat
        HTrajectories

        %HEarth Graphics handle for earth
        HEarth

        %HParticles Graphics handle for sats
        HSats

        %FigureHandle the handle of the figure
        FigureHandle
    end


    methods
        function obj = ExampleHelperSat(numSat, sats)
            %ExampleHelperSat Constructor
            obj.Traces = zeros(numSat,1,3);
            for i=1:numSat
                obj.Traces(i,1,:) = sats(i,:);
            end

            obj.len = 1;
            obj.sats = zeros(numSat,3);

            obj.FigureHandle = figure('Name', 'Satellite Orbit Gif ');
            % clear the figure
            ax = axes(obj.FigureHandle);
            cla(ax)

            % customize the figure
            obj.FigureHandle.Position = [100 100 500 500];
            axis(ax, 'equal');
            grid(ax, 'on');
            box(ax, 'on');

            hold(ax, 'on')
            [x,y,z] = sphere;
            x = x.*6371000;
            y = y.*6371000;
            z = z.*6371000;

            obj.HEarth = surf(x,y,z); % earth
            color = jet(24);
            markers = ['o','*','s','d'];
            obj.HSats = gobjects(numSat);
            obj.HTrajectories = gobjects(numSat);
            for i = 1:numSat
                if mod(i,4) == 0
                    id = 4;
                else
                    id = mod(i,4);
```

```
                end
                obj.HSats(i) = scatter3(ax, 0,0,0,'MarkerEdgeColor',color(i,:),
                    'Marker', markers(id));
                obj.HTrajectories(i) = plot3(ax, obj.Traces(i,:,1),
                    obj.Traces(i,:,2),
                    obj.Traces(i,:,3),'Color',color(i,:),'LineWidth',2);
            end

            title(ax, strcat('Satellite Orbit Gif', 't = 0'));
            xlabel(ax, 'x (m)');
            ylabel(ax, 'y (m)');
            zlabel(ax, 'z (m)');
            hold(ax, 'off');

            view(3)
        end

        function updatePlot(obj, sats, t)
            % updatePlot
            % render sats
            obj.sats = sats;
            for i = 1:size(sats,1)
                obj.HSats(i).XData = obj.sats(i,1);
                obj.HSats(i).YData = obj.sats(i,2);
                obj.HSats(i).ZData = obj.sats(i,3);
            end

            if obj.len == 100
                obj.Traces(:,1:99,:) = obj.Traces(:,2:100,:);
                id = 100;
            else
                id = obj.len + 1;
                obj.len = obj.len + 1;
            end
            for i = 1:size(sats,1)
                obj.Traces(i,id,:) = obj.sats(i,:);
            end

            % draw trajectories
            for i = 1:size(sats,1)
               obj.HTrajectories(i).XData = obj.Traces(i,:,1);
               obj.HTrajectories(i).YData = obj.Traces(i,:,2);
               obj.HTrajectories(i).ZData = obj.Traces(i,:,3);
            end

            ax = get(obj.FigureHandle, 'currentaxes');
            title(ax, strcat('Satellite Orbit Gif ',' t = ',num2str(t)));
        end

    end

end
```

**Entry**:

```
function ex2_satellite_position_calc
    close all;
    clc;clear all;
    orbitName = ['A','B','C','D','E','F'];% satellite orbital name
    satId = [1,2,3,4];% satellite id

    addpath('../utils/')

    % Q1,2,3
    initTatPos = zeros(numel(orbitName)*numel(satId),3);
    k = 1;
    for i = 1:numel(orbitName)
        for j = 1:numel(satId)
            slotID = strcat(orbitName(i),num2str(satId(j)));% satellite name
            [q1, q2, q3, i1, i2, i3] = calcSatPosition(slotID, 0, 0);% positions for given satellite
            initTatPos(k,:) = [i1,i2,i3];
            k = k + 1;
        end
    end
    figure
    color = jet(24);
    markers = ['o','*','s','d'];
    for i = 1:numel(orbitName)
        for j = 1:numel(satId)
            k = (i - 1)*numel(satId)+j;
            plot3(initTatPos(k,1),initTatPos(k,2),initTatPos(k,3),'Marker',markers(j),'Color',co
                on;
        end
    end
    grid on;
    title('Satellite Positions','Interpreter','latex');

    %% Q4
    t = linspace(0,60*60*12,50);
    satPosX = zeros(numel(orbitName)*numel(satId),numel(t));
    satPosY = zeros(numel(orbitName)*numel(satId),numel(t));
    satPosZ = zeros(numel(orbitName)*numel(satId),numel(t));

    for i = 1:numel(t)
        for j = 1:numel(orbitName)
            for k = 1:numel(satId)
                slotID = strcat(orbitName(j),num2str(satId(k)));
                [q1, q2, q3, i1, i2, i3] = calcSatPosition(slotID, 0, t(i));
                satPosX((j-1)*numel(satId)+k,i) = i1;
                satPosY((j-1)*numel(satId)+k,i) = i2;
                satPosZ((j-1)*numel(satId)+k,i) = i3;
            end
        end
    end
```

```
figure
color = jet(24);
markers = ['o','*','s','d'];
for i = 1:numel(orbitName)
    for j = 1:numel(satId)
        k = (i - 1)*numel(satId)+j;
        plot3(satPosX(k,:),satPosY(k,:),satPosZ(k,:),'Marker',markers(j),'Color',color(k,:))
            on;
    end
end
[x,y,z] = sphere;
x = x.*6371000;
y = y.*6371000;
z = z.*6371000;
surf(x,y,z);
grid on;
title('Satellite Orbits','Interpreter','latex');

drawGif = 0;
axis tight manual % this ensures that getframe() returns a consistent size
giffilename = 'satellite.gif';

t = linspace(0,60*60*12,12*60);
satPosX = zeros(numel(orbitName)*numel(satId),1);
satPosY = zeros(numel(orbitName)*numel(satId),1);
satPosZ = zeros(numel(orbitName)*numel(satId),1);
iter = 1;
init = 0;
frames = [];
for i = 1:numel(t)
    for j = 1:numel(orbitName)
        for k = 1:numel(satId)
            slotID = strcat(orbitName(j),num2str(satId(k)));
            [q1, q2, q3, i1, i2, i3] = calcSatPosition(slotID, 0, t(i));
            satPosX((j-1)*numel(satId)+k,1) = i1;
            satPosY((j-1)*numel(satId)+k,1) = i2;
            satPosZ((j-1)*numel(satId)+k,1) = i3;
        end
    end
    sats = [satPosX satPosY satPosZ];
    if init == 0
        init = 1;
        satsh = ExampleHelperSat(24,sats);
    else
        updatePlot(satsh, sats, t(i));
    end
    drawnow;
    frame = getframe(gcf);
    frames = [frames;frame];
end

if drawGif == 1
```

```
        for i = 1:size(frames,1)
            im = frame2im(frames(i));
            [imind,cm] = rgb2ind(im,256);
%             imwrite(imind,cm,strcat(num2str(i),'.png'));
            if iter == 1
                iter = 2;
                imwrite(imind,cm,giffilename,'gif', 'Loopcount',inf);
            else
                imwrite(imind,cm,giffilename,'gif','WriteMode','append');
            end
        end
    end
end
```

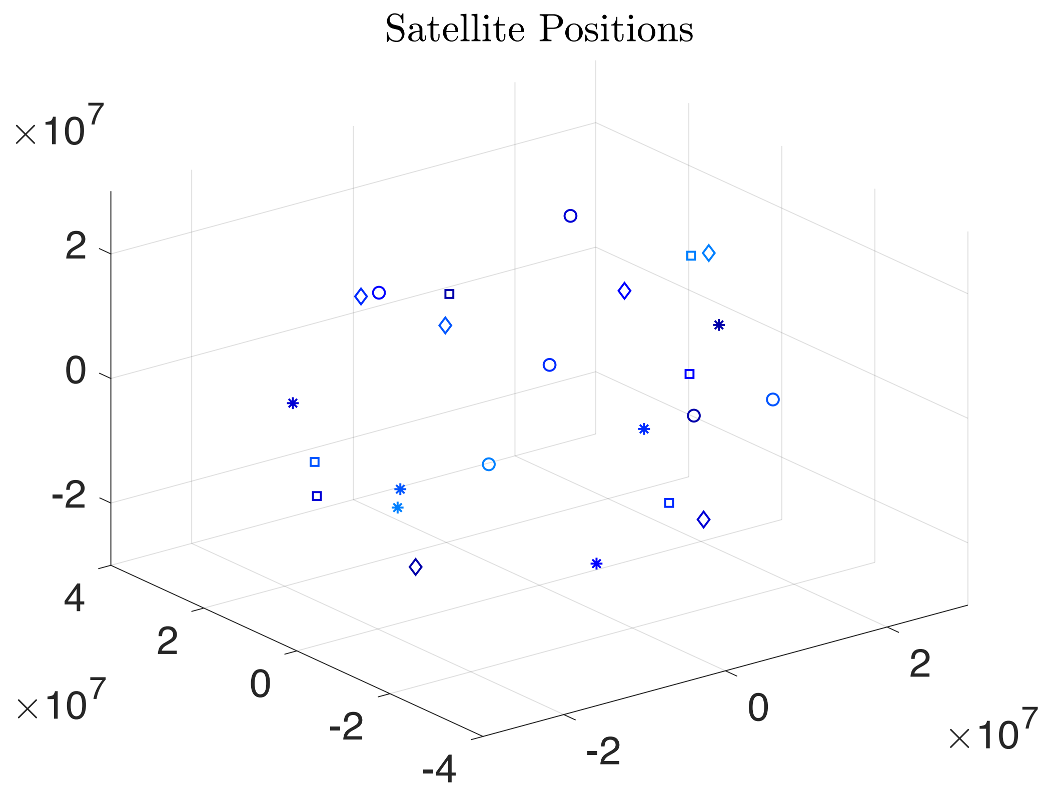


***Figure 2.1.*** Satellite Positions at given epoch.

## 2.4 Experiments

### 2.4.1 Single Epoch

The result is shown in Fig 2.1

### 2.4.2 Time Interval

The result is shown in Fig 2.2

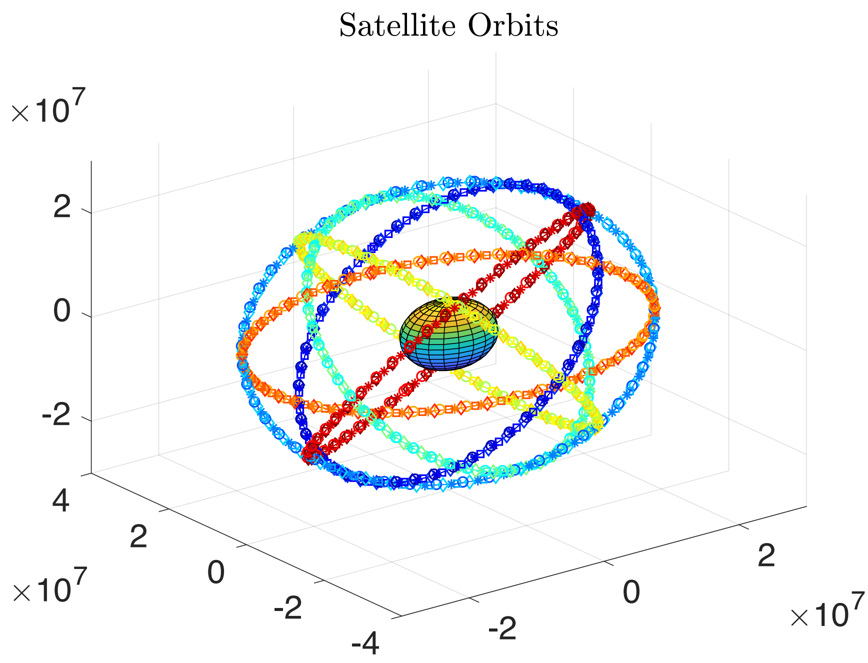


***Figure 2.2.*** Satellite orbits for 12 hours.

### 2.4.3 DEMO Video

A animation that demonstrates the motion of 24 satellites can be watched by the following link: https://youtu.be/PjKFGVKaSvQ.

### 2.4.4 Conclusion

Through this assignment, I grasp how to do the predictions of satellite positions using the Kepler elements.

# 3 Assignment C: Global and Local Coordinate Systems

The main objective of the assignment is to grasp the transformation from the global coordinate system (WGS84) to the local East-North-Up (ENU) coordinate system. With this relationship, satellite positions can be projected to the local ENU coordinate system. Furthermore, the visibility of a given satellite can be determined by checking its elevation angle.

## 3.1 Theory

According to [1], the transformation from the WGS84 frame to the ENU frame can be computed as:

$$\begin{bmatrix} x_{ENU} \\ y_{ENU} \\ z_{ENU} \end{bmatrix} = \mathbf{R}_{WGS84}^{ENU} \begin{bmatrix} x_{WGS84} \\ y_{WGS84} \\ z_{WGS84} \end{bmatrix} \tag{3.1}$$

$$\mathbf{R}_{WGS84}^{ENU} = \mathbf{R_1}(90^\circ - \phi)\mathbf{R_3}(90^\circ + \lambda) \tag{3.2}$$

$$\mathbf{R_1}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{3.3}$$

$$\mathbf{R_3}(\alpha) = \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Once the coordinates in the ENU frame is obtained, the azimuth, elevation, and zenith angles can be computed as follows:

$$tan(\alpha_{azimuth}) = \frac{x_{ENU}}{y_{ENU}} \tag{3.5}$$

$$cos(\alpha_{zenith}) = \frac{z_{ENU}}{\sqrt{x_{ENU}^2 + y_{ENU}^2 + z_{ENU}^2}} \tag{3.6}$$

$$\alpha_{elevation} = 90^\circ - \alpha_{zenith} \tag{3.7}$$

## 3.2 Tasks

- Implement a Matlab function for transforming a point in the WGS84 frame to the local ENU frame.

- Implement a MATLAB function that computes the azimuth, elevation and zenith angle.
- Read satellite positions for a given time epoch from a SP3 file and then convert the satellite positions to the ENU coordinates. Determine elevation and azimuth in the local coordinate system for the vectors to all GPS satellites at the given time epoch. Finally, determine the distances from origo to the satellites visible.

## 3.3 Code

Complementary functions:

**WGS842ENU: convert coordinates from WGS84 to ENU**:

```
function [e,n,u] = WGS842ENU(lati, longi, dx, dy, dz)
%WGS842ENU: convert coordinates from WGS84 to ENU.
%   [e,n,u] = WGS842ENU(lati, longi, dx, dy, dz) does the conversion
%   from WGS84 coordinates to ENU coordinates.
%   Author: xiahaa@space.dtu.dk
    R1 = consR(deg2rad(90-lati),1);
    R3 = consR(deg2rad(90+longi),3);
    R = R1*R3;
    v1 = R*[dx;dy;dz];
    e = v1(1);n = v1(2);u = v1(3);
end
```

**calcAzimuthZenithElevation: compute the azimuth, zenith, elevation angle for a given point**:

```
function [azimuth, zenith, elevation] = calcAzimuthZenithElevation(e,n,u)
%   [azimuth, zenith, elevation] = calcAzimuthZenithElevation(e,n,u) does
%   the computation of the azimuth, zenith, elevation angle for a given
%   point.
%   Author: xiahaa@space.dtu.dk
    azimuth = rad2deg(atan2(e,n));
    zenith = 90 - rad2deg(asin(u / sqrt(e^2+n^2+u^2)));
    elevation = rad2deg(asin(u / sqrt(e^2+n^2+u^2)));
end
```

**sp3fileParser: parsing sp3 file.**:

```
function content = sp3fileParser(path)
%sp3fileParser: parsing sp3 file.
%   sp3fileParser(path) parses a given sp3 file.
%   path: path to the sp3 file.
%   content: struct
%       firstline: struct
%               versionSym: see sp3 definition.
%               PosOrVel
%               YearStart
%               MonthStart
%               DayStart
```

```
%               HoureStart
%               MinuteStart
%               SecondStart
%               NumOfEpochs
%               DataUsed
%               CoordianteSys
%               OrbitType
%               Agency
%        satNum: int, number of satellites.
%        satNames: array, satellite name.
%        accuracy: array, satellite accuracy.
%        sections: cells of struct, each element is a struct defined as
%        follows.
%             section: struct.
%               year
%               Month
%               Day
%               Hour
%               Minute
%               Second
%               satPos: array of satNumx4 [x,y,z,clocktime].
%    Author: xiahaa@space.dtu.dk

    %% fopen file
    fid = fopen(path,"r");
    linenum = 1;
    satNum = 0;
    shiftbase = 23;

    sections = {};
    k = 1;
    while ~feof(fid)
        tline = fgetl(fid);
        if linenum == 1
            firstline = parseFirstLine(tline);
            content.firstline = firstline;
        elseif linenum == 3
            thirdline = parseThirdLine(tline);
            satNum = str2num(thirdline.NumSats);
            content.satNum = satNum;
            content.satNames = thirdline.SatNames;
        elseif linenum >= 4 && linenum <= 7
            res = parse4to7Line(tline);
            content.satNames = [content.satNames;res.SatNames];
        elseif linenum >= 8 && linenum <=12
            res = parse8to12FifthLine(tline);
            if linenum == 8
                content.accuracy = res.Accuracy;
            else
                content.accuracy = [content.accuracy;res.Accuracy];
            end
        elseif linenum >= (shiftbase) && linenum <= (shiftbase+satNum)
```

```
            if strcmp(tline(1:3), 'EOF') == 1
                break;
            end

            if linenum == shiftbase
                Year = tline(4:7);
                Month = tline(9:10);
                Day = tline(12:13);
                Hour = tline(15:16);
                Minute = tline(18:19);
                Second = tline(21:31);
                section.year = str2num(Year);
                section.Month = str2num(Month);
                section.Day = str2num(Day);
                section.Hour = str2num(Hour);
                section.Minute = str2num(Minute);
                section.Second = str2num(Second);
            else
                satPos = parseSatPos(tline);
                satPos.x = str2num(satPos.x);
                satPos.y = str2num(satPos.y);
                satPos.z = str2num(satPos.z);
                satPos.clock = str2num(satPos.clock);
                section.satPos(linenum - (shiftbase),1) = satPos;
            end

            %% valid
            if linenum == (shiftbase+satNum)
                shiftbase = shiftbase + satNum + 1;
                content.sections{k,1} = section;
                k = k + 1;
            end
        end
        linenum = linenum + 1;
    end

end

function satPos = parseSatPos(line)
    satPos.symbol = line(1);
    satPos.name = line(2:4);
    satPos.x = line(5:18);%km
    satPos.y = line(19:32);%km
    satPos.z = line(33:46);%km
    satPos.clock = line(47:60);%micro second
end

function firstline = parseFirstLine(line)
    firstline.versionSym = line(1:2);
    firstline.PosOrVel = line(3);
    firstline.YearStart = line(4:7);
    firstline.MonthStart = line(9:10);
```

```
    firstline.DayStart = line(12:13);
    firstline.HoureStart = line(15:16);
    firstline.MinuteStart = line(18:19);
    firstline.SecondStart = line(21:31);
    firstline.NumOfEpochs = line(33:39);
    firstline.DataUsed = line(41:45);
    firstline.CoordianteSys = line(47:51);
    firstline.OrbitType = line(53:55);
    firstline.Agency = line(57:60);
end

function thirdline = parseThirdLine(line)
    thirdline.NumSats = line(5:6);
    thirdline.SatNames = [];
    for i = 1:17
        thirdline.SatNames = [thirdline.SatNames;line(10+(i-1)*3:10+(i-1)*3+2)];
    end
end

function res = parse4to7Line(line)
    res.SatNames = [];
    for i = 1:17
        res.SatNames = [res.SatNames;line(10+(i-1)*3:10+(i-1)*3+2)];
    end
end

function res = parse8to12FifthLine(line)
    res.Accuracy = [];
    for i = 1:17
        res.Accuracy = [res.Accuracy;line(10+(i-1)*3:10+(i-1)*3+2)];
    end
end
```

**Entry**:

```
function ex3_enu_conversion

    addpath('../utils/');

    % fetched satellite position from igs at * 2018 9 18 5 30 0.00000000
    data = [-14084.965421 -13335.977939 -18444.389348; ...
            -8412.178797 14931.928610 20810.047600; ...
            -13642.570569 -22690.514868 2505.504310; ...
            -390.404350 21748.781259 14982.038597; ...
            -22918.304145  6890.582919 11571.183052; ...
            -25196.382195 -6397.745158 6768.549040; ...
            2796.631366 -22395.316629 -13830.185289; ...
            -13947.715293 -6618.056480 21577.959060; ...
            15947.740945 -1338.501133 -21183.204680; ...
            15947.740945 -1338.501133 -21183.204680; ...
            9161.084379 24938.647725 -2378.996417; ...
            -11093.440122 22608.322634 -8521.726915; ...
```

```
            18485.647522 -18026.093616 -5511.098696; ...
            3157.925831 20557.837992 -16521.290198; ...
            3402.684664 -20247.785807 16449.616936; ...
            -19351.327692 10977.974490 -14018.876526; ...
            -4035.301059 -15495.645056 -21685.320944; ...
            -19015.304410 17811.742561 -5797.886412; ...
            19708.841479  7637.705325 -16267.555982; ...
            26132.764030  2961.941803  4275.236351; ...
            -9918.277161 -24070.081964 -5110.381059; ...
            -7975.048024 -15242.870677 20435.202393; ...
            9811.195637 14662.117690 -20056.505271; ...
            16842.930627 18008.705148 9990.949356; ...
            10105.122386 -12626.042864 21040.684542; ...
            11417.769712 -23432.342744 -4392.131096; ...
            -15790.334354 -2544.884817 -20573.112489; ...
            12353.007291  7962.709748 22097.350060; ...
            -26312.156995   524.630853 -4199.834644; ...
            21865.789063 -7671.878821 13389.204161; ...
            19113.591091 -12285.821643 -13637.262853; ...
            ];
    data = data.*1000;% km to m
    % local position llh
    testCase(1,:) = [55.78575300466123,12.525384183973078,0];% DTU 101
    consParams = struct('a',6378137.0,'f',1/298.257223563); % some constants
    testid = 1;
    lat = testCase(testid,1);%
    lon = testCase(testid,2);%
    height = testCase(testid,3);%
    [xo,yo,zo] = llhtoCartesian(lat, lon, height, consParams);% to ECEF

    % satpos in enu
    satPosENU = zeros(size(data,1),3);
    azimuths = zeros(size(data,1),1);
    zeniths = zeros(size(data,1),1);
    visibles = zeros(size(data,1),1);
    dists = ones(size(data,1),1).*-1;
    for i = 1:size(data,1)
        satPos = data(i,:);
        dx = satPos(1) - xo;
        dy = satPos(2) - yo;
        dz = satPos(3) - zo;
        % conversion
        [e,n,u] = WGS842ENU(lat, lon, dx, dy, dz);
        satPosENU(i,:) = [e,n,u];

        %% compute azimuth and zenith
        [azimuth, zenith, elevation] = calcAzimuthZenithElevation(e,n,u);
        azimuths(i) = azimuth;
        zeniths(i) = zenith;

        if elevation > 5 % visibility threshold
            % identify as visible
```

```
            visibles(i) = 1;
            dists(i) = sqrt(dx^2+dy^2+dz^2);
        else
            visibles(i) = 0;
        end
    end

    % earth
    [xe,ye,ze] = sphere;
    xe = xe.*6371000;
    ye = ye.*6371000;
    ze = ze.*6371000;
    surf(xe,ye,ze);
    grid on;
    hold on;

    % local point
    plot3(xo,yo,zo,'m*','MarkerSize',10);

    for i = 1:size(data,1)
        if visibles(i) == 1
            % shown as a diamond with red color
            plot3(data(i,1),data(i,2),data(i,3),'rd','MarkerSize',10);
            % line satellite to local point
            line = [[xo, yo, zo];[data(i,1),data(i,2),data(i,3)]];
            plot3(line(:,1),line(:,2),line(:,3),'LineWidth',3);
            % show the distance
            text(0.5*(xo+data(i,1)),0.5*(yo+data(i,2)),0.5*(zo+data(i,3)),strcat('dist=',num2str
        else
            % if not visible, display as black square.
            plot3(data(i,1),data(i,2),data(i,3),'ks','MarkerSize',10);
        end
    end
    title('Assignment3','Interpreter','latex');
end
```

## 3.4 Experiments

### 3.4.1 Single Epoch

The result is shown in Fig 3.1. From the website of GPS.gov[1], it says that "GPS satellites fly in medium Earth orbit (MEO) at an altitude of approximately 20,200 km (12,550 miles)." Based on this data, it can be seen the calculated distances are reasonable since they are close to 20,200 km.

[1] https://www.gps.gov/systems/gps/space/

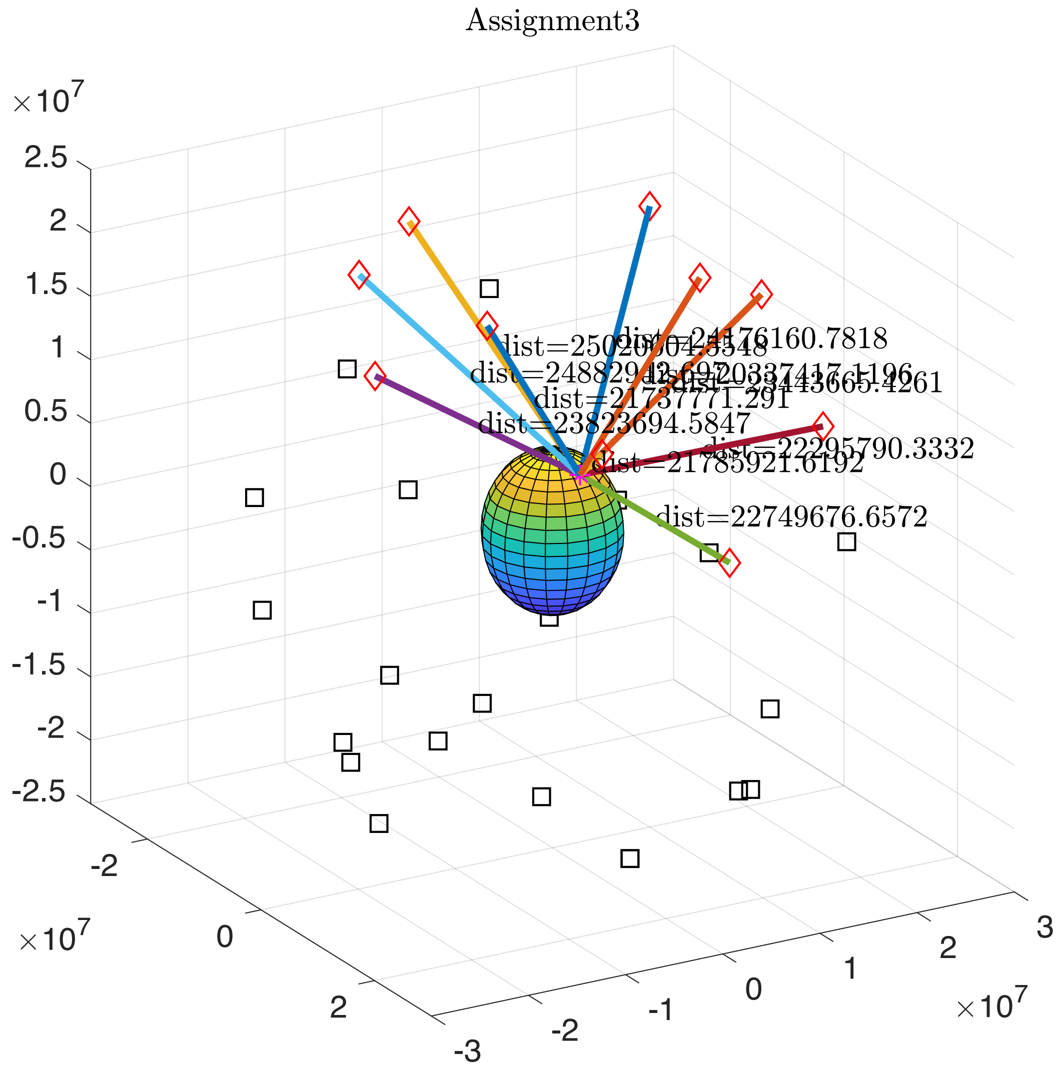


***Figure 3.1.*** Satellite visibili at a given epoch.

### 3.4.2 Brute-force test

In order to conveniently do more brute-force test, a MATLAB GUI program is designed. The GUI program supports the following features:

1. Automatically parse a sp3 file and list all relevant information.
2. List all satellite positions' observations in a list box for the user to select.
3. Input arbitrary latitude, longitude, and altitude.
4. Visualization.

Snapshots of the GUI program are shown as follows:
**Initialization**:
**After loading a sp3 file**:
**Results of 2 examples**:

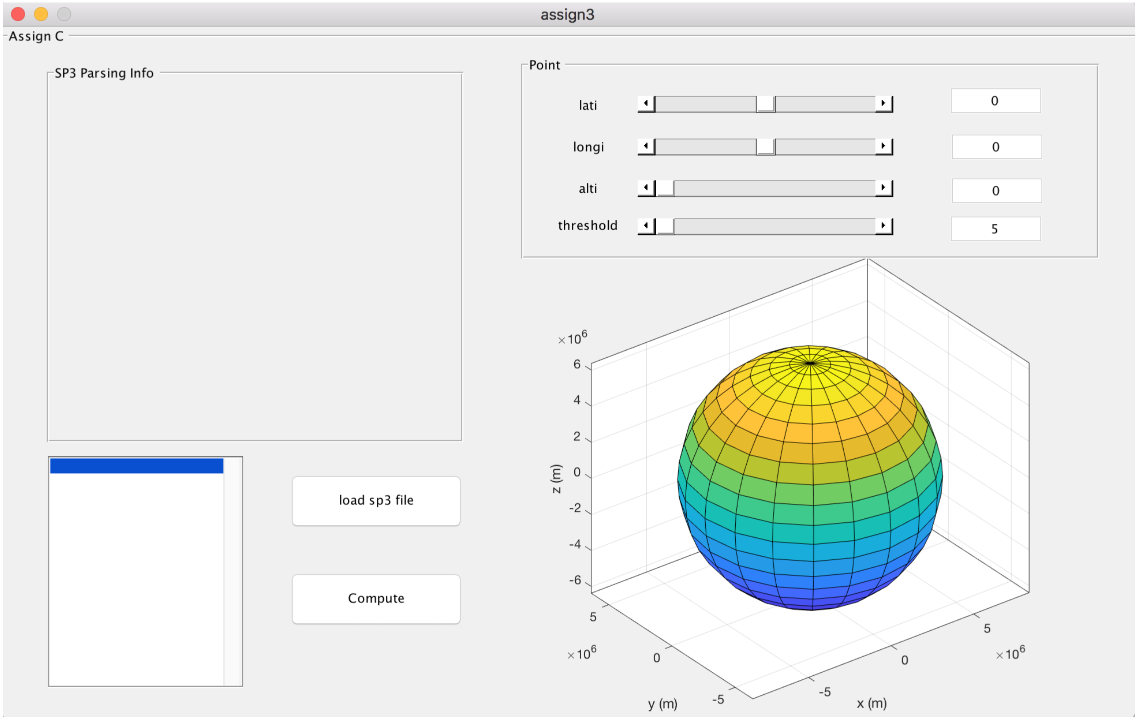


***Figure 3.2.*** Initialization.

### 3.4.3 Conclusion

Through this assignment, I grasp how to do the conversion from WGS84 to ENU frame and verify the visibility of a given satellite by checking its elevation angle.
I designed a GUI program that can easily parse a sp3 file, and do the conversion automatically. The program is easy-to-use.

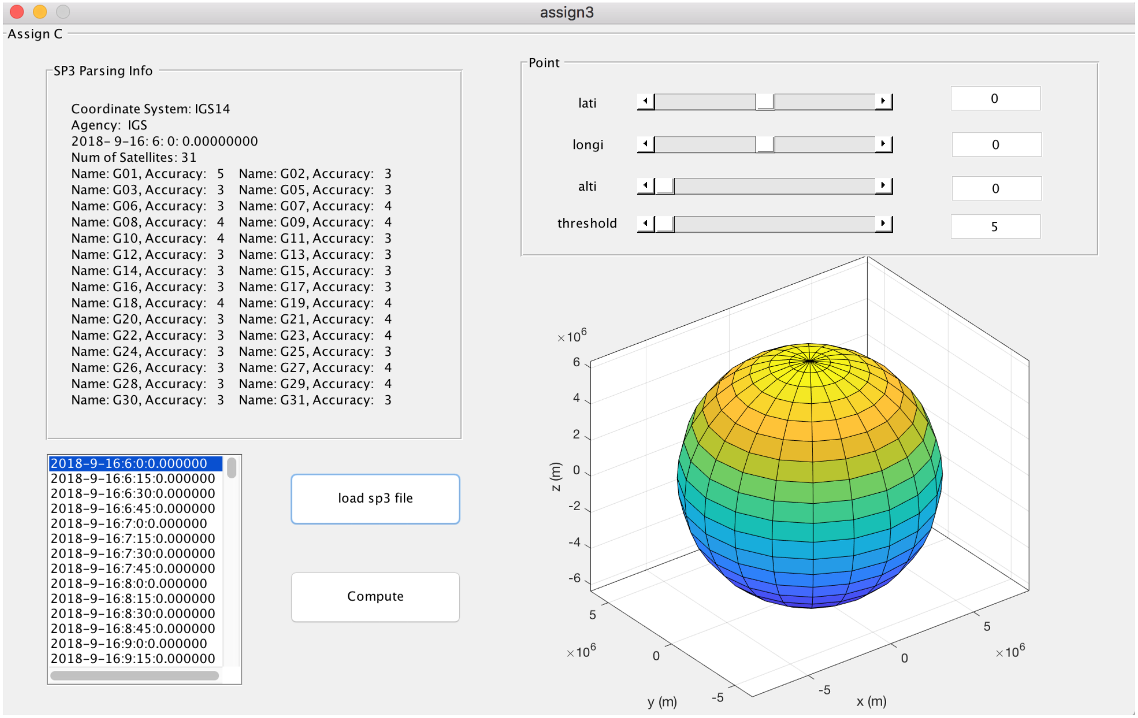


*Figure 3.3.* Loading a sp3 file.

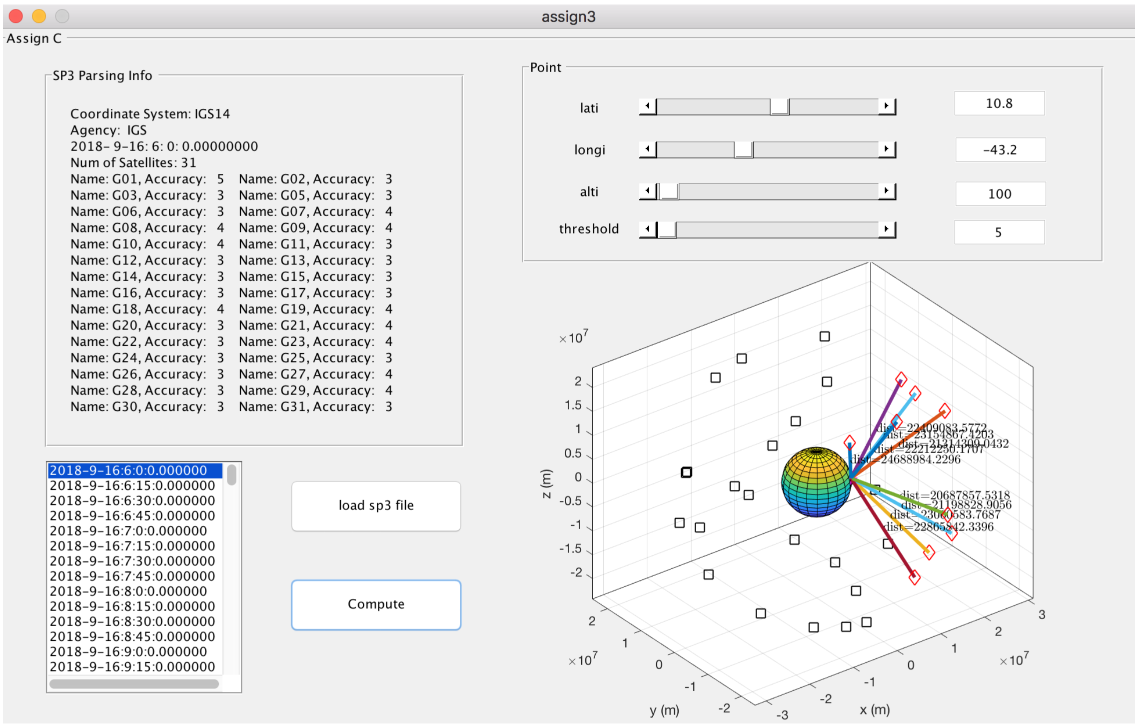


*Figure 3.4.* Snapshot of the result 1.

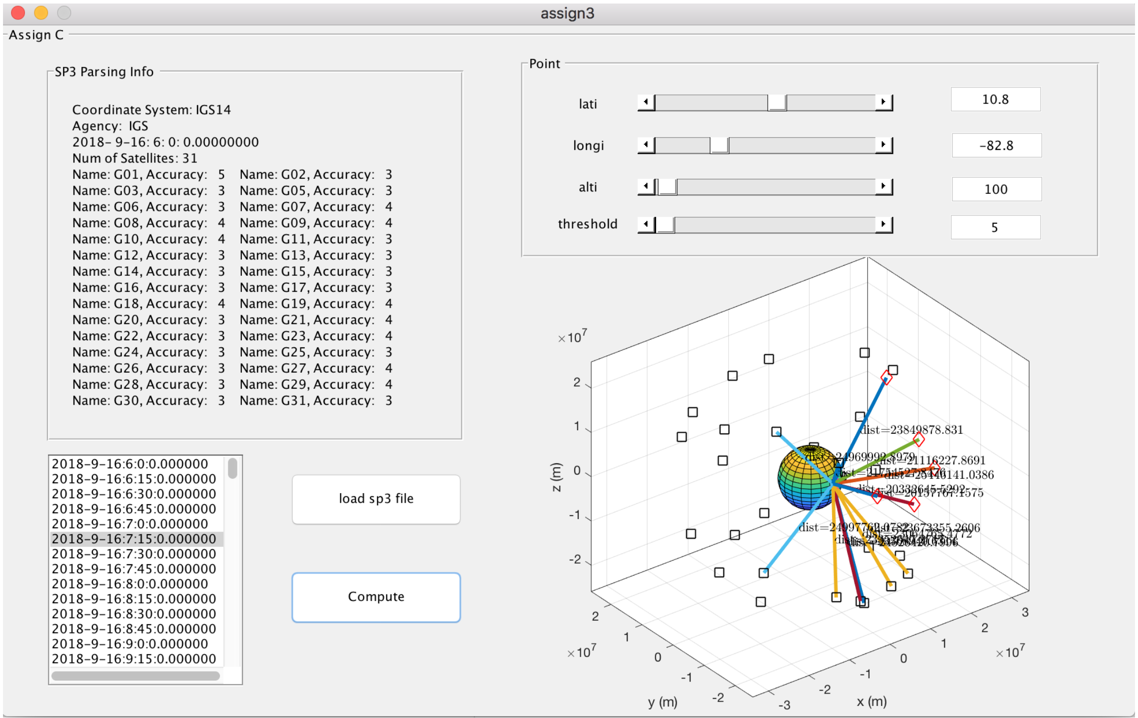


*Figure 3.5.* Snapshot of the result 2.

# References

[1] Pratap Misra and Per Enge. Global positioning system: signals, measurements and performance second edition.

**National Space Institute**
Technical University of Denmark
Elektrovej Building 328, room 007
2800 Kongens Lyngby
Denmark www.space.dtu.dk

E-mail: xiahaa@space.dtu.dk