University of Colorado - Boulder

ASEN 3200 - Orbital Mechanics

Asteroid Constellation Design

Kuat Drive Yards

The goal of this project is to design a satellite constellation about the asteroid Bennu that optimizes two objectives: number of spacecraft, and science yield. Our team decided to use the minimum allowable number of spacecraft as a fixed design point, meaning our design space was centered on the orbit design. After some sensitivity analyses, our team was able to confidently design an inclined orbit for each S/C that maximized our science gain, while avoiding collisions with either Bennu, or the other spacecraft. This, however, comes at the cost of lower overall science value.

Team Members:
Ian McCarty
Jacob Ecks
Hayden Gebhardt

Instructors:
Dr. Casey Heidrich
Dr. Mark Moretto

May 4, 2023

Ann and H.J. Smead
Aerospace Engineering Sciences
UNIVERSITY OF COLORADO **BOULDER**

Contents

I. Nomenclature

TBP	=	Two Body Problem
r_a	=	Radius at Apoapsis
r_p	=	Radius at Periapsis
e	=	Eccentricity
SRP	=	Solar Radiation Perturbation
P_{SR}	=	Solar Radiation Pressure
ACI	=	Asteroid Centered Inertial
$ACAF$	=	Asteroid Centered Accelerating Frame
AU	=	Astronomical Unit
C_R	=	Coefficient of Reflectivity = 1.2
μ_{Bennu}	=	Gravitational Parameter of Bennu = $4.892 \frac{m^2}{s^2}$
P_{Bennu}	=	Period of Bennu = 4.297 Hours
A_*	=	Satellite Area Exposed to the Sun

II. Introduction and Design Constraints

A. Introduction

In orbital mechanics, there is a multitude of different ways to design an orbit between 2 bodies. Mathematically, the relationship between these two astronomical bodies is dependent on a variety of different parameters related to the orbit of the second. In reality, these parameters are usually narrowed down to a specific range due to design constraints. In this project, we were tasked with determining a variety of these parameters given our first body, the asteroid Bennu, and our second body, a satellite tasked with observing different visual elements of Bennu. At first, the problem was a direct TBP between a single satellite and the asteroid, and given some general parameters, the orbit could be propagated about Bennu for a given time span. Eventually, we expanded upon this by allowing this addition of more satellites. The problem is still manageable even with this addition due to the fact the satellites have a negligible effect on each other due to gravity. Thus, the scenario remains a relatively straightforward TBP. Finally, the overarching goal of these satellite additions was to maximize the amount of scientific data pulled from Bennu. However, this extra data is not without its trade-offs, as we will discuss in detail throughout this report.

B. Design Constraints

Space is an incredibly unforgiving environment, and trying to perform research in such a domain poses many challenges. In this project, we neglected a variety of different environmental conditions that would make this analysis much more involved. Ultimately, we only included the effect of solar radiation perturbations. This effect will accelerate our spacecraft in a constant direction as long as it is in view of the sun.

The mission proposal itself also imposes some design constraints. Perhaps most obvious, our satellite(s) must not hit Bennu throughout the week they are orbiting. Additionally, for our constellation design, we must ensure that none of the satellites hit each other, which is a valid concern given how the orbit planes will drift due to SRP. Additionally, we have a few design constraints given the geometry and mass of the satellites. For one, our given mission design mass is 1000 kilograms. So for a singular satellite, our mass would be equal to 1000 kilograms. Once we add satellites to our design, our mass will begin to drop off proportionally with the number of spacecraft we've added. This effect is given by the equation below.

$$m = \frac{1000}{1.05 \cdot N_{sc}} [\text{kg}] \tag{1}$$

Additionally, we must consider how the area of each spacecraft will change with the number of spacecraft due to payload size constraints. A simple model for this is as follows:

$$A = 5 - \frac{1}{5} \left(N_{sc} - 1\right) [m^2] \tag{2}$$

This calculated area is the area that is acted upon by solar radiation pressure, the acceleration of which is determined

from the following:

$$\vec{a}_{SRP} = -\frac{p_{SR} C_R A_*}{m_{SAT}} \frac{\vec{r}_{*SAT}}{|\vec{r}_{*SAT}|} [\text{km/s}^2] \tag{3}$$

III. Design Objectives

We will now discuss what we want to achieve within this mission. The first choice left up to our group to optimize was the amount of spacecraft present in our constellation design. As discussed earlier, we have weight limitations for our mission design and our "effective" weight decreases for every added spacecraft, as we must account for payload weight limitations. We also need to account for payload space limitations, meaning our effective spacecraft area will change as the number of spacecraft changes. Conversely, more spacecraft makes the system less susceptible to total failure. If one spacecraft fails or malfunctions, having multiple satellites makes that loss significantly less impactful to the mission objectives. The second design choice left up to us to optimize was the amount of valuable scientific information to extract. Since we are interested in visual information of Bennu, having more satellites in orbit gives access to more visual information, and therefore a higher "science" value. It is clear that both of our objectives are at odds with each other, and trade-offs must be made.

IV. Design Approach

As stated previously, the two goals of this project were to create a satellite constellation system that both minimizes the number of spacecraft in the constellation and maximizes the cost function seen in the following equation.

$$J_{sv} = \sum_{i=1}^{N_{targ}} \sum_{j=1}^{N_{sc}} \sum_{k=1}^{N_{V,i,j}} \left[\frac{0.007263}{G R_{i,j,k}^2} \sin \phi_{i,j,k} H_i(t_k) \right] \tag{4}$$

Looking at the summations involved in the cost function, it is easy to see that the variable we can affect the most is the number of required spacecraft. Our design approach began with testing a series of orbits and comparing their behavior to an overall cost value. This value was compared to other arbitrary cost values for different initial orbit states to develop a baseline for our orbit propagation method. From this, we found that angles of inclination set at $\pm 45^o$ yielded consistently high cost values for two spacecraft. With this knowledge, we then focused on the ground resolution and phase angle mask functions within our cost function to analytically determine necessary restrictions on our inclined orbits. This resulted in a decision to set each spacecraft periapsis on the dark side of Bennu, to maximize our camera visibility window on the light side of the asteroid. We then tailored values for the radius of apoapsis and periapsis to the minimum and maximum design constraints listed in the problem statement ($r_a < 3$ km, $r_b > 0.3$ km). These restrictions are convenient because they not only prevent our asteroid from crashing into Bennu, but also act as excellent reference bounds for an iterative orbit determination method. We recognized this during our design process and were able to make use of for loops to determine the appropriate radii that balanced a good minimum ground resolution with an efficient camera boresight visibility window.

Through iteration, we were able to find two sets of initial conditions that allowed both spacecraft to achieve periapsis on the dark side of Bennu, while remaining within the required bounds. We could not simply accept these values, however, because we needed to ensure that the orbits could hold for a duration of one Earth week, This would not be an issue if we did not need to account for solar radiation pressure, but our constellation incurs a drift velocity in the $-\hat{X}_{ACI}$ direction. To account for this in code, we created cell arrays for each spacecraft in the constellation and calculated the distance from Bennu for each position state along the orbit duration. This comes at a high computational cost, so after analysis, we determined that the best approach to accounting for solar radiation pressure was to supply a positive offset to each minimum and maximum radius condition by the absolute value of the difference between the closest pass of Bennu and the minimum or maximum condition. Since we only need both orbits to meet these conditions for one week, we can neglect their behavior after the final second of our mission window.

At this point, we had completed the majority of our orbit determination analysis, creating a variety of iteration tools along the way. We decided to employ these iteration tools as part of a Monte Carlo simulation to further hone our orbit parameters. Since we were confident on our method for determining the shape of the orbit, we wrapped our code with conditions to compare a range of orbit inclinations to overall mission cost in an attempt to numerically validate our geometric approach to finding the angle of inclination for each orbit. Our simulation very closely validated the expected

$\pm 45^o$ angles of inclination we chose initially. After this, we decided we were prepared to run our final simulation with these calculated initial orbit states.

V. Final Solution

Our final optimized satellite constellation is composed of two spacecraft. Each of these spacecraft have an orbit inclined at 45 degrees to the Bennu equatorial plane, but in opposite directions. These initial orbits are nearly circular, with a radius of apoapsis of 1.5 km, and a radius of periapsis of 1.3 km. Over one week, they tend to migrate, but never get too close or too far from Bennu. With this configuration, the science value we obtained, was **258504**. This cost is much higher than the test case, which is to be expected because the test case only considers one spacecraft on a polar orbit, which is not an ideal orbit.

VI. Justification Behind Our Final Solution

To begin, we discuss the accuracy of the simulations performed. All the important pieces (orbit propagation, facet observability, and the obj reader) were all checked by the provided MATLAB grader pages, confirming the functionality of our core simulation code. Any code we added, such as the orbit constructor, was checked with a simpler, known problem. In the case of the orbit constructor, we applied a simple polar orbit (as these remain the most constant under SRP) and checked the calculations by hand to confirm the functionality. When implementing each of these pieces, we used a provided test case to ensure each piece was working together correctly. This test case consisted of a given initial state for a polar orbiting spacecraft, and a given science value of 90,000. When input to our code, we obtained a science value of 89,400, which we considered to be well within tolerances. From these results, we are confident that our code is accurate and can perform a simulation to meet desired objectives.

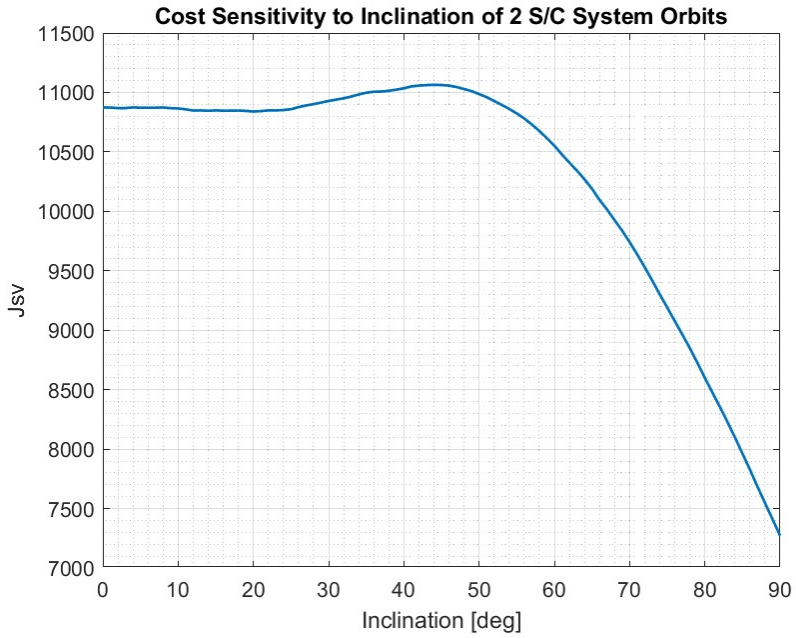


Fig. 1 Results from inclination sensitivity analysis

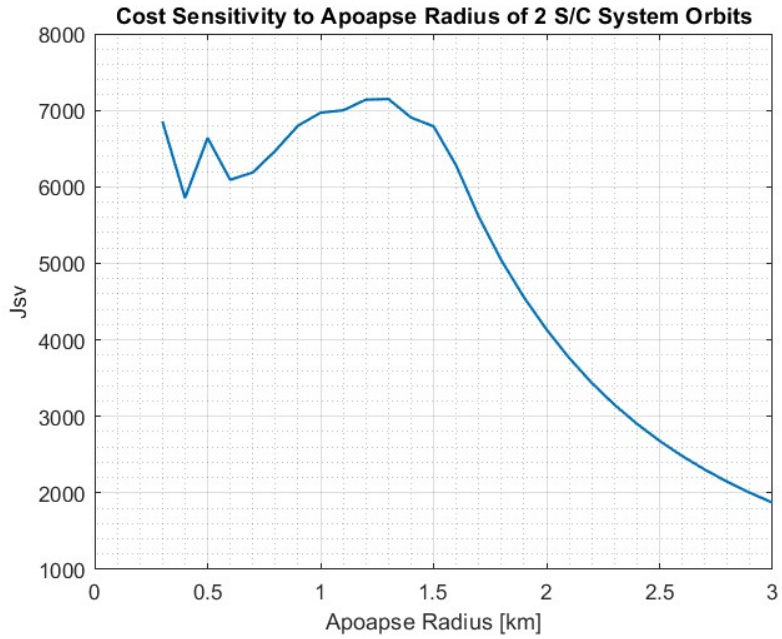


Fig. 2 Results from apoapse radius sensitivity analysis

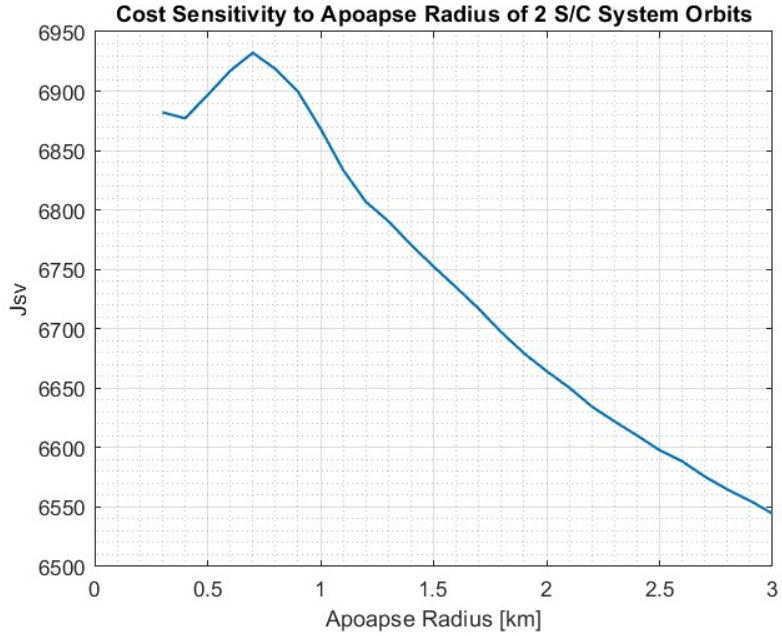


Fig. 3 Results from periapse radius sensitivity analysis

The above figures are results from our sensitivity, or Monte Carlo, analysis. The inclination was easy to chose, as there is a distinct peak at 45 degrees. The radii of apoapse and periapse were more difficult to choose, as the choice of one affects the other. To choose optimal values, we had to start with peak values then slowly work our way towards the true optimal case, while also considering cases in which the spacecraft get too close or collide with Bennu. We used 1.5 km for the apoapse and 1.3 km for the periapse, both of which are close to, but not at the peaks. Another way, though much more computationally intense, would have been to iterate over both of the parameters at the same time to generate

a 3D distribution that we can extrapolate the optimal combination from. The above plots were generated over a time span of 4 hours, explaining the low cost values shown on the vertical axis. We started with 1 hour, then increased the time, noting that the behavior of the parameters was not changing significantly.

VII. Appendix

A. MATLAB Plots

1. Time Series

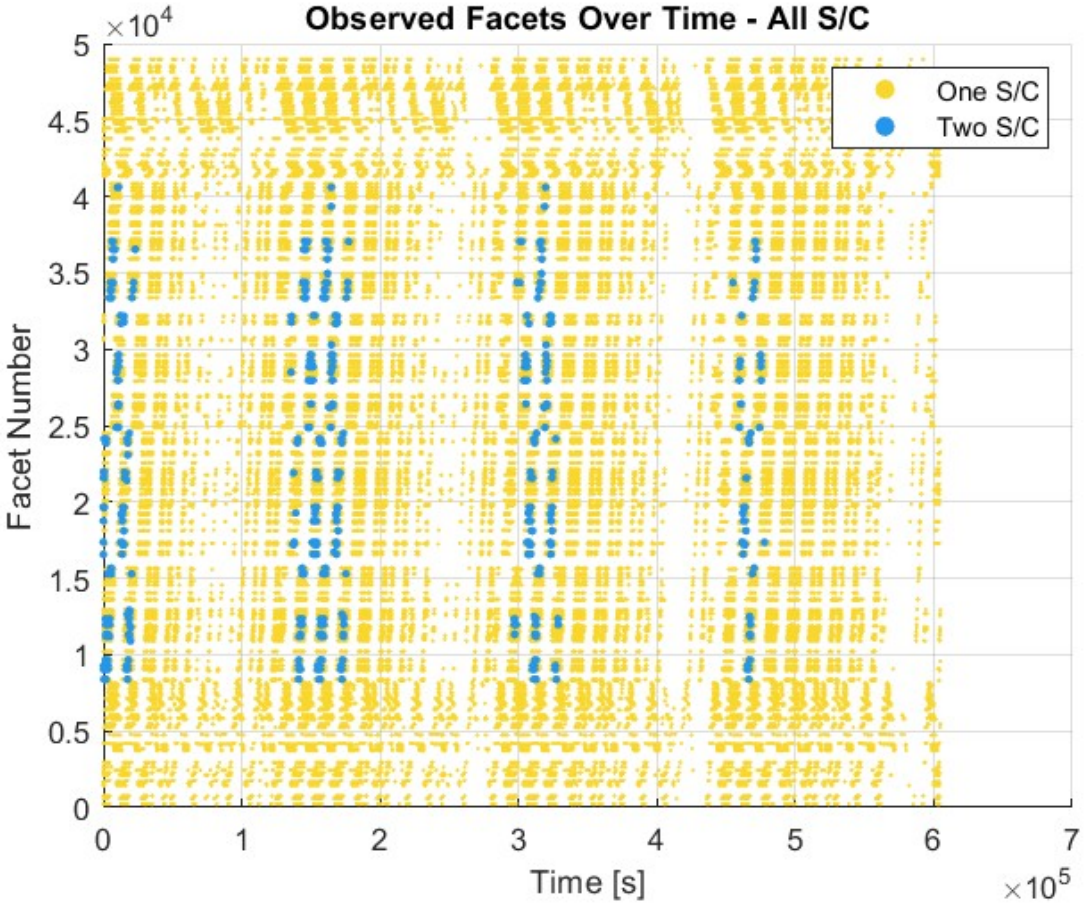


Fig. 4 Time Series of S/C Observing Facets

As shown in the plot, our configuration gives 100 percent coverage of all the desired targets, with most being observed multiple times throughout the week. There is only a small amount of overlap, due to the fact that we designed around two spacecraft. If we were to increase the number of spacecraft, we would see more overlap, and likely a more optimized cost. Regions of whitespace are either facets that are not targets of interest, or are not observed by any spacecraft at that time.

2. Groundtracks

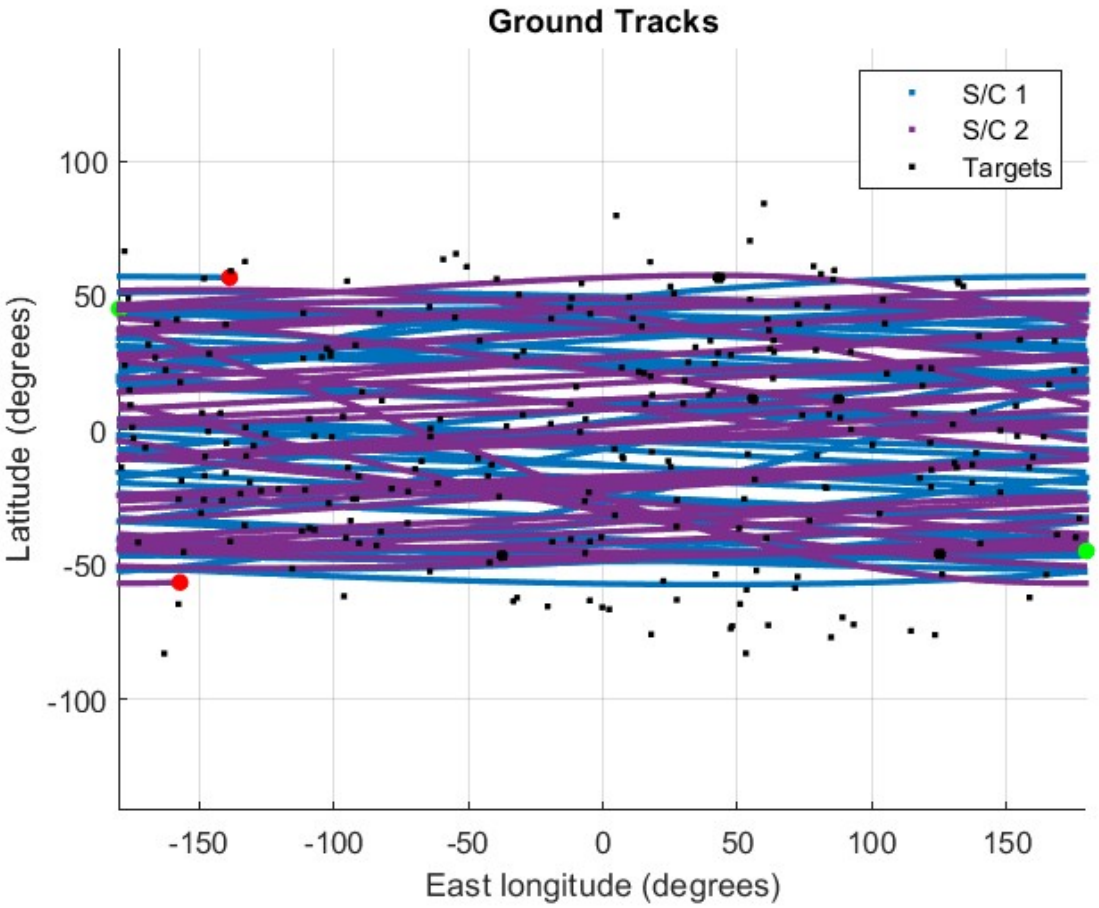


Fig. 5 Ground Tracks for One Week

Our configuration gives very good coverage over a band from about -50 to 50 degrees latitude. This region also contains a majority of the targets, so we were sure to design an orbit that didn't have too high of an inclination, otherwise we would waste time over targets that are not of interest.

3. 3D Bennu Track

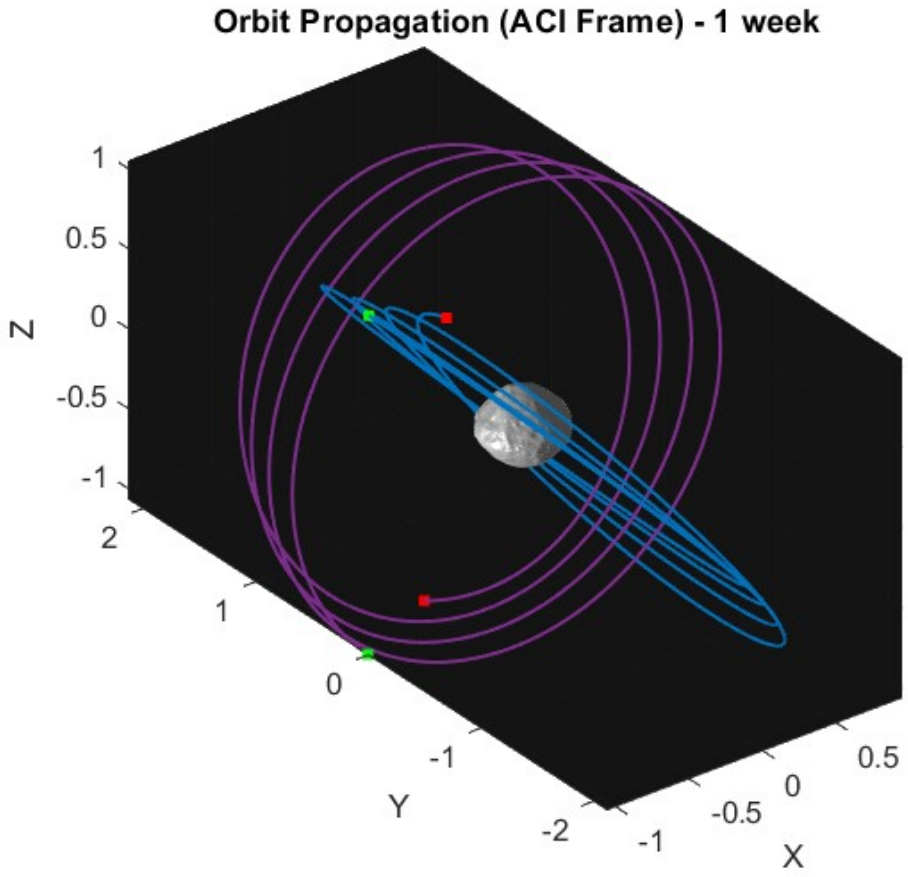


Fig. 6 Orbit Propagation for One Week in Inertial Space

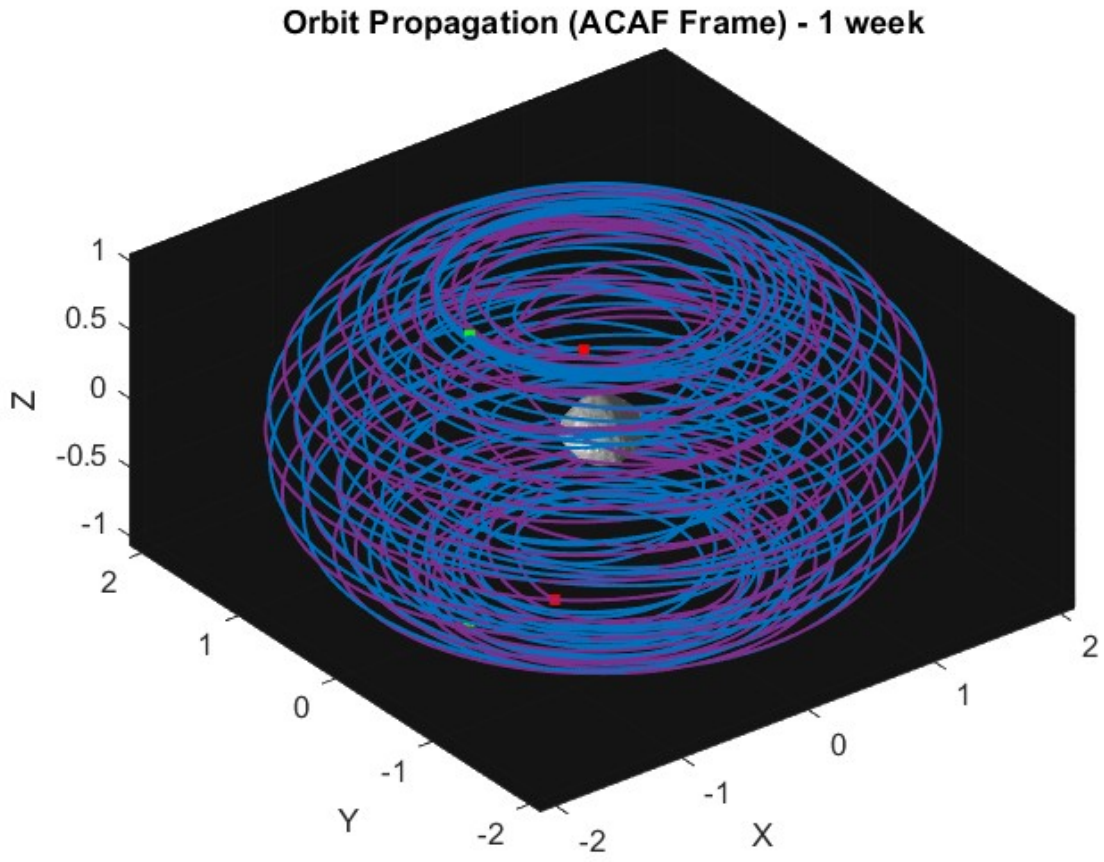
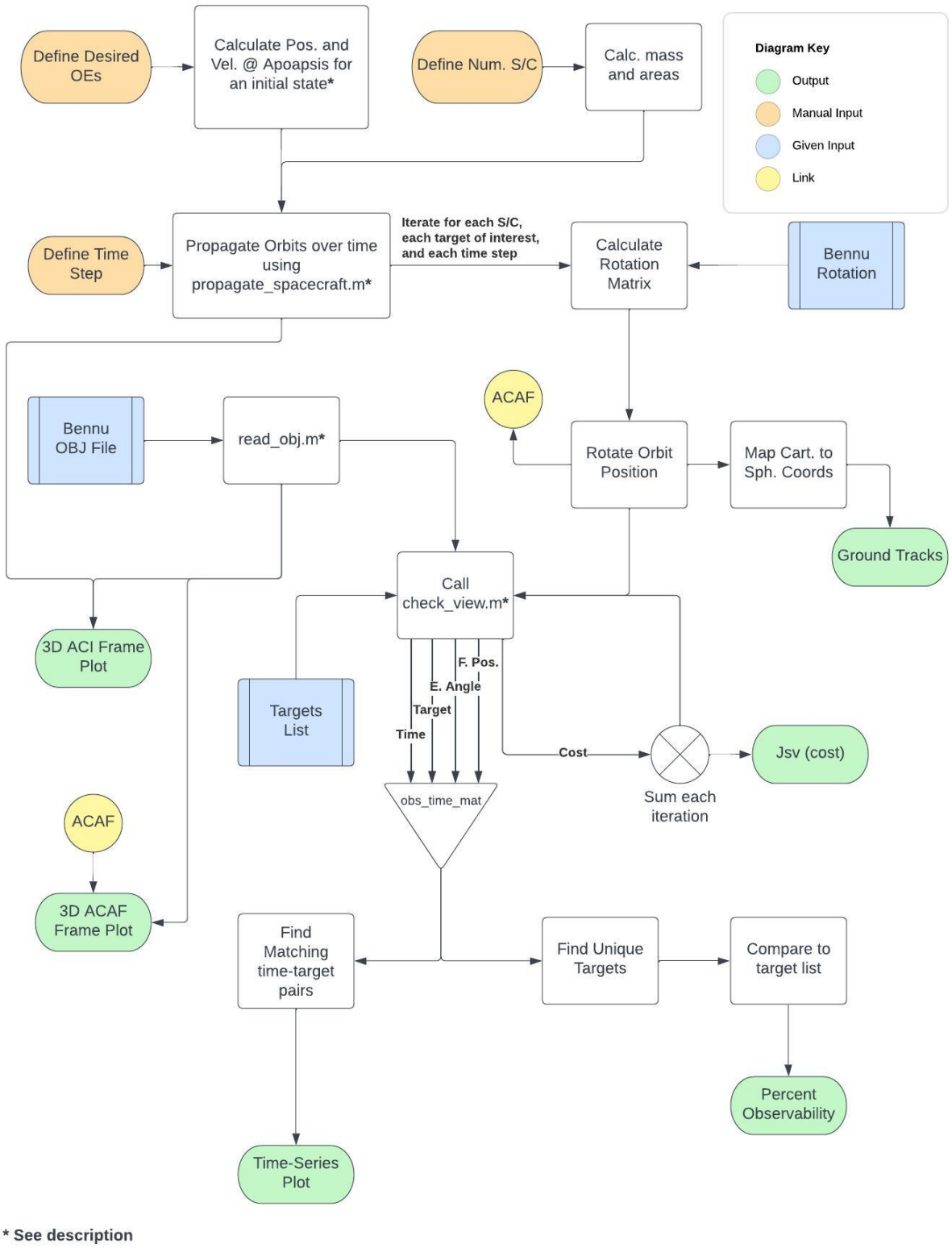


Fig. 7 Orbit Propagation for One Week in Accelerating Space

Both figures above confirm that our orbits are within the valid parameters of minimum and maximum range (also confirmed computationally). More importantly, they show that the orbits are stable enough for one week that they don't precess into the main body. The ACAF frame plot also emphasizes our coverage over a week, in that there is almost a full oblate spheroid encasing the main body.

B. Block Diagram



C. Block Diagram Explanation

We begin the computational process by defining our initial desired orbital elements, the number of spacecraft, and the time step for simulation. The orbital elements consist of the radii of apoapse and periapse, true anomaly, inclination, RAAN, and argument of periapse. In our case, we set the true anomaly to 180° to represent an initial position at apoapse. Due to space constraints, the complex processes used to calculate parameters is not included in the diagram. This process* utilizes a function describing a set of calculations that outputs a state vector given a set of orbital elements. First, we calculate the eccentricity, semi-major axis, and semilatus rectum. From these, we can find the velocity and position components in the perifocal frame. The remaining orbital elements are used to calculate rotation matrices to rotate from the perifocal frame into the ACI frame, giving a usable initial state. Inclination was the most important parameter, as we found it affected our cost the most compared to the others. From the number of spacecraft, we calculate the mass of each, as well as the effective solar radiation pressure area of each. In our case, the time vector is defined as one week discretized into 60 second increments. All these inputs (initial state, time, mass, and area) are used to propagate the orbit over the time span. Within this function†, there is a function defining the equations of motion (two-body problem, including SRP) that are used in the numerical integrator. Over each iteration, a new set of 60 second intervals is passed in, and the numerical integrator propagates the orbit over this time. Then, only the last point is saved and output from the function, and is now set as the initial state for the next time interval. Using this last point, the orbital elements are computed in a process similar to the one used to calculate the inital state, but somewhat reversed. This process outputs a matrix of state vectors and orbital elements at each time. Using the given rotation rate of Bennu, we can calculate a rotation matrix at each time step in order to rotate the ACI positions into the ACAF frame. The resultant positions can then be mapped from cartesian coordinates to spherical coordinates, to show the ground tracks. The ACAF positions are also input to the check_view.m function, along with the data obtained via read_obj.m (the Bennu OBJ file) and the targets list. This function‡ takes a 3-vertex obj file as input, filters out comment lines, and discretizes the data into a set of facets and a set of vertices. The check_view function§ first calculates two vectors between one vertex and the other two for a given facet. By taking the cross product, we can determine the normal vector to the facet. We can also calculate the center of the facet via geometry. The range, or distance between a target and the S/C is the difference between the position of the S/C and the position of the target. From all these calculations, we find the elevation and camera angles. We also calculate the phase angle mask function at this point. Now, we check that the elevation angle is greater than or equal to 15°, the camera angle is less than or equal to the FOV (calculated earlier from the number of S/C), and that the phase angle function returns a 1 (illuminated). If these conditions are met, a target is considered observable, and the ground resolution and cost are calculated. The function outputs a logical representing the observability, the elevation angle (E. Angle), the cost, and the location of the center (F. Pos.). Over each iteration, only the observable facets are stored in a large cell array, along with an associated time, the associated elevation angle, and the associated position of the target. This large cell array is used for two things. First, we use it to create a time series plot, showing the targets and how many S/C are observing them at a given time. This is done by finding the "intercepts" between the data sets and plotting those points as a different color than the separate S/C data sets (which represent only 1 S/C observing a target). We also use the large cell array output to calculate a percent observability - a measure of how many targets are observed over a certain period of time. This is done by concatenating each S/C dataset vertically, then finding the unique targets. The length of the resultant vector is then compared to the length of the initial target list to give a ratio of observed to total targets. The final step we can take is to plot the orbits and Bennu as an object in 3D space. We make two plots of this nature: one in the ACI frame, and one in the ACAF frame. The process is similar for both, but we just need to choose the array of states corresponding to each frame. We can take these plots one step further, and create an animated video of the orbits. This is done by creating a MATLAB VideoWriter object, creating a 3D plot at each time step, then writing each time step as a frame to the video. We also need to rotate Bennu at each time step by multiplying a rotation matrix by each of the vertices given by the .OBJ file. This is a computationally intense process, so the computation was performed once, and the results were saved as a .mat file to be loaded in when necessary.

Note: The core of this code flow was used in a pseudo Monte-Carlo analysis, by wrapping it in iteration to determine the effect different parameters would have on the value of the cost function.

*Initial State

†propagate_spacecraft.m

‡read_obj.m

§check_view.m

D. MATLAB Code

1. Main Script

```
% ASEN 3200 - Bennu Constellation Design
% Written by: Hayden Gebhardt, Jacob Ecks
% Housekeeping
clc; clear; close all;
tic;
%fprintf("Beginning Simulation...\n")

% Sound for code completion
load("CompleteSound.mat")
finished = audioplayer(y, Fs);

%% Part 1 - Read OBJ and setup constellation parameters
filename = "g_06290mm_spc_obj_0000n00000_v008.obj";
[facets, vertices] = read_obj(filename);

% Initial State Vectors
X0(1:6,1) = orbits_construction(1.5,1.5,180,45,270,90);
X0(1:6,2) = orbits_construction(1.5,1.5,180,-45,270,90);
X0(4:5,2) = -X0(4:5,2);
% Test Case - X0(1:6,1) = [ 0 -1 0 0 0 0.6994e-4]';

% Num Spacecraft - mass and area calculations
Nsc = numel(X0)/6;
m_orbiter = 1000/(1.05*Nsc); % [kg]
A_srp = (5 - 0.2*(Nsc - 1)) * 1e-6; % [km^2]

% INTEGRATION TIME SPAN
time = 0:60:3600*24*7;

% Bennu rotation properties
period = 4.297461 * 3600;
n = 2*pi/period;

%% Part 2 - Propagate orbit for 1 week in 1 minute increments
% Iterate over each time step, propagating the orbit
for j = 1:Nsc
    for i = 1:length(time)-1

        [Xout{j}(1:6,i), OEout{j}(1:6,i)] = propagate_spacecraft(X0(1:6,j), time(i), ...
            time(i+1), A_srp, m_orbiter);
        X0(1:6,j) = Xout{j}(1:6,i);

    end
end

%% Part 3 - Check Observability and angles at each time step for each facet
% get target list
load("Target_list.mat")

% initialize cost
Jsv = 0;
% Iterate for each S/C
for k = 1:Nsc
    cnt = 1;
    % iterate for each target
    for i = 1:length(targets)


        % iterate for each time step
        for j = 1:length(Xout{k})
            % calculate change in angle about z axis
            Δ_psi = n * time(j);
```

```
            % calculate rotation matrix at that angle
            rotate{j} = [cos(Δ_psi), -sin(Δ_psi), 0; sin(Δ_psi), cos(Δ_psi), 0; 0, 0, 1];
            Xout_ACAF{k}(1:3,j) = rotate{j}*Xout{k}(1:3,j);
            % check view for each iteration for each spacecraft
            [observable, elevationAngle, ¬, cost, center] = ...
                check_view(rotate{j}*Xout{k}(1:3,j), targets(i), facets, vertices, Nsc, ...
                rotate{j});

            Jsv = Jsv + cost;

            % Only keep the observed facets and timestamp them. Also store
            % angles with them
            if observable == 1
                obs_time_mat{k}(cnt,1) = time(j);
                obs_time_mat{k}(cnt,2) = targets(i);
                obs_time_mat{k}(cnt,3) = elevationAngle;
                obs_time_mat{k}(cnt,4:6) = center;
                cnt = cnt + 1;
            end
        end

    end
end
fprintf("Cost Value: %.3f\n", Jsv)

%% Percent Observability
% Find the percent observability
temp=[];
% Append targets list into one nx1 vector
for i = Nsc
    temp = [temp; obs_time_mat{i}(:,2)];
end
% find unique values
obs_targets = unique(temp);
% find the length of the unique values and the full targets list to get
% percent observability
percent_obs = length(obs_targets)/length(targets);
fprintf("Percent Observability: %.3f\n",percent_obs*100)

%% Ground Track Calculations
% Convert cartesian to spherical
for i = 1:Nsc
    [TH, PHI, ALT] = cart2sph(Xout_ACAF{i}(1,:),Xout_ACAF{i}(2,:),Xout_ACAF{i}(3,:));
    lat{i} = rad2deg(TH);
    lon{i} = rad2deg(PHI);
    [TH_tar, PHI_tar, ¬] = cart2sph(obs_time_mat{i}(:,4), obs_time_mat{i}(:,5), ...
        obs_time_mat{i}(:,6));
    lat_tar{i} = rad2deg(TH_tar);
    lon_tar{i} = rad2deg(PHI_tar);
end

figure();
hold on;
xlabel("East longitude (degrees)"); ylabel("Latitude (degrees)"); title("Ground Tracks")
grid on;
for i = 1:Nsc
    scatter(lat{i}, lon{i},'.', 'DisplayName',strcat("S/C ",num2str(i)))
    plot(lat{i}(1), lon{i}(1), 'g.','MarkerSize', 20, 'HandleVisibility','off')
    plot(lat{i}(end), lon{i}(end), 'r.','MarkerSize', 20, 'HandleVisibility','off')
end
scatter(lat_tar{1}, lon_tar{1}, '.k', 'DisplayName', 'Targets')
axis ([-180 180 -90 90]); axis equal; legend("show")

%% Time Series Plot
[C, ia, ib] = intersect(obs_time_mat{1}(:,1:2),obs_time_mat{2}(:,1:2),'rows');
figure();
hold on;
for i = 1:Nsc
```

```
    sc= scatter(obs_time_mat{i}(:,1), ...
        obs_time_mat{i}(:,2),20,'MarkerFaceColor',[250/255,215/255,44/255], ...
        'MarkerEdgeColor',[250/255,215/255,44/255],'DisplayName','One S/C');
    if i == 1
        sc.HandleVisibility = "off";
    end
    grid on; grid minor; xlabel("Time [s]"); ylabel("Facet Number"); title("Observed ...
        Facets Over Time - All S/C")
end
scatter(C(:,1),C(:,2),20, 'MarkerFaceColor',[38/255,150/255,235/255], ...
    'MarkerEdgeColor',[38/255,150/255,235/255],'DisplayName','Two S/C');
legend('show')


%% Part 4 - Plot Bennu at final time and the orbits of each S/C
% Produce scatter plots with facet number and time observed
for i = 1:Nsc
    figure();
    scatter(obs_time_mat{i}(:,1), obs_time_mat{i}(:,2),[],obs_time_mat{i}(:,3)*180/pi,'.');
    a = colorbar; ylabel(a,"Elevation Angle [deg]")
    grid on; grid minor; xlabel("Time [s]"); ylabel("Facet Number"); ...
        title(strcat("Observed Facets Over Time - S/C ",num2str(i)))
end

%% ACI Frame Plotting
% Plot the facets and vertices
figure();
hold on;
view(3)
% Bennu
patch('vertices',vertices,'faces',facets, 'FaceColor', [230/255,230/255,230/255], ...
    'EdgeAlpha',0);

% plot orbit of each S/c
for i = 1:Nsc
    plot3(Xout{i}(1,:),Xout{i}(2,:),Xout{i}(3,:),'LineWidth',1.3)
    plot3(Xout{i}(1,1),Xout{i}(2,1),Xout{i}(3,1), 'g.','MarkerSize', 10)
    plot3(Xout{i}(1,end),Xout{i}(2,end),Xout{i}(3,end), 'r.','MarkerSize', 10)
end

% Formatting
hold off; grid on;
xlabel("X");ylabel("Y");zlabel("Z")
set(gca,"Color", [20/255, 20/255, 20/255])
axis square;
axis equal;
lt = light;
lt.Position = [-1 0 0];
title("Orbit Propagation (ACI Frame) - 1 week")

%% ACAF Frame Plotting
% Plot the facets and vertices
figure();
hold on;
view(3)
% Bennu
patch('vertices',vertices,'faces',facets, 'FaceColor', [230/255,230/255,230/255], ...
    'EdgeAlpha',0);

% plot orbit of each S/c
for i = 1:Nsc
    plot3(Xout_ACAF{i}(1,:),Xout_ACAF{i}(2,:),Xout_ACAF{i}(3,:),'LineWidth',1.3)
    plot3(Xout_ACAF{i}(1,1),Xout_ACAF{i}(2,1),Xout_ACAF{i}(3,1), 'g.','MarkerSize', 10)
    plot3(Xout_ACAF{i}(1,end),Xout_ACAF{i}(2,end),Xout_ACAF{i}(3,end), 'r.','MarkerSize', 10)
end

% Formatting
hold off; grid on;
```

```
xlabel("X");ylabel("Y");zlabel("Z")
set(gca,"Color", [20/255, 20/255, 20/255])
axis square;
axis equal;
lt = light;
lt.Position = [-1 0 0];
title("Orbit Propagation (ACAF Frame) - 1 week")


%% Animating Orbit Propagation and rotation of bennu
v = VideoWriter('bennu_orbits.mp4','MPEG-4');
open(v)

% Rotate the vertices at each time
fprintf("Rotating Verticess...\n")
% for i = 1:length(rotate)
%     for j = 1:length(vertices)
%         vert_rot{i}(1:3,j) = rotate{i}*vertices(j,1:3)';
%     end
% end
load("BennuRot.mat")

for i=1:length(vert_rot)
    figure(100);
    hold on;
    pt = patch('vertices',vert_rot{i}','faces',facets, 'FaceColor', ...
        [230/255,230/255,230/255], 'EdgeAlpha', 0.1);
    sc1 = ...
        plot3(Xout{1}(1,1:i),Xout{1}(2,1:i),Xout{1}(3,1:i),Xout{1}(1,i),Xout{1}(2,i),Xout{1}(3,i), ...
        'g.', 'LineWidth',1.3);
    sc2 = ...
        plot3(Xout{2}(1,1:i),Xout{2}(2,1:i),Xout{2}(3,1:i),Xout{2}(1,i),Xout{2}(2,i),Xout{2}(3,i), ...
        'g.', 'LineWidth',1.3);
    hold off;
    view(3)

    xlabel('X [m]'); ylabel('Y [m]'); zlabel('Z [m]');
    axis equal; grid on;
    set(gca,"Color", [20/255, 20/255, 20/255])
    title("ACI Frame - One Week")
    lt = light;
    lt.Position = [-1 0 0];
    drawnow

    frame = getframe(gcf);
    writeVideo(v,frame);
    if i < length(vert_rot)
        clf
    end
end
close(v)

%% Animate in ACAF Frame
v2 = VideoWriter('bennu_orbits_acaf.mp4','MPEG-4');
open(v2)

for i=1:length(Xout_ACAF{1})
    figure(200);
    hold on;
    pt = patch('vertices',vertices,'faces',facets, 'FaceColor', [230/255,230/255,230/255], ...
        'EdgeAlpha', 0.1);
    sc1 = ...
        plot3(Xout_ACAF{1}(1,1:i),Xout_ACAF{1}(2,1:i),Xout_ACAF{1}(3,1:i),Xout_ACAF{1}(1,i),Xout_ACAF{1}(2,i)
        'g.', 'LineWidth',1.3);
    sc2 = ...
        plot3(Xout_ACAF{2}(1,1:i),Xout_ACAF{2}(2,1:i),Xout_ACAF{2}(3,1:i),Xout_ACAF{2}(1,i),Xout_ACAF{2}(2,i)
        'g.', 'LineWidth',1.3);
    hold off;
```

```
    view(3)

    xlabel('X [m]'); ylabel('Y [m]'); zlabel('Z [m]');
    axis equal; grid on;
    set(gca,"Color", [20/255, 20/255, 20/255])
    title("ACAF Frame - One Week")
    lt = light;
    lt.Position = [-1 0 0];
    drawnow

    frame = getframe(gcf);
    writeVideo(v2,frame);
    if i < length(Xout_ACAF{1})
        clf
    end
end
close(v2)

%% Animating Ground Tracks
v3 = VideoWriter('bennu_groundtracks.mp4','MPEG-4');
open(v3)

for i = 1:length(lat{1})
    figure(300);
    hold on;
    xlabel("East longitude (degrees)"); ylabel("Latitude (degrees)"); title("Ground Tracks ...
        w/ Targets - One Week")
    grid on;

    scatter(lat{1}(1:i), lon{1}(1:i),'.')
    scatter(lat{2}(1:i), lon{2}(1:i), '.')
    scatter(lat_tar{1}, lon_tar{1}, '.k', 'DisplayName', 'Targets')
    hold off;
    axis ([-180 180 -90 90]); axis equal;

    drawnow
    frame = getframe(gcf);
    writeVideo(v3,frame)
    if i < length(lat{1})
        clf
    end
end
close(v3)

%% End
play(finished)
runtime = toc;
fprintf("Simulation Complete...\n")
fprintf("Run Time: %.3f s - %.3f hr\n", runtime, runtime/3600)
```

2. Function to Define Initial Orbit

```
function X0 = orbits_construction(ra, rp, TA, inc, RAAN, w)
    % Fully define the orbit
    e = (ra-rp)/(ra+rp);
    a = (ra+rp)/2;
    p = a*(1-e^2);
    mu = 4.892e-9;

    % Calculate state components in perifocal frame
    rPQW = [p*cosd(TA)/(1+e*cosd(TA)); p*sind(TA)/(1+e*cosd(TA)); 0];
    vPQW = [-sqrt(mu/p) * sind(TA); sqrt(mu/p) * (e + cosd(TA)); 0];

    % Calculate rotation matrices using other OEs
    R1 = [1 0 0;
```

```
          0 cosd(inc) sind(inc);
          0 -sind(inc) cosd(inc)];

    R3 = [cosd(RAAN) sind(RAAN) 0;
          -sind(RAAN) cosd(RAAN) 0;
          0 0 1];

    R3w = [cosd(w) sind(w) 0;
          -sind(w) cosd(w) 0;
          0 0 1];

    % Rotate Perifocal components into ACI frame
    r = R3w*R1*R3*rPQW;
    v = R3w*R1*R3*vPQW;

    % combine position and velocity into one state vector
    X0 = [r; v];
```

3. Function to Read an OBJ File

```
function [facets, vertices] = read_obj( obj_filename )
% Any line that starts with    #    is a comment
%     A set of lines defining vertices start with    v    and are followed by 3 floating point
% numbers to define a vertex position
%     v -0.1187999993562698 0.1491799950599670 0.1283600032329559
%     A set of lines defining facets start with a    f    and are followed by 3 integers
%     f 425 382 791
%     Note: defined in CCW order!

% Open file
obj_file = fopen(obj_filename);
scan = textscan(obj_file, "%c %.16f %.16f %.16f", 'CommentStyle','#');

% Find which rows are for the vertices and which are for the facets
v_idxs = find(scan{1}=='v');
f_idxs = find(scan{1}=='f');

% read in data for the facets and vertices and combine into single matrix
% for each
vertices = [scan{2}(v_idxs(1):v_idxs(end)), scan{3}(v_idxs(1):v_idxs(end)), ...
    scan{4}(v_idxs(1):v_idxs(end))];
facets = [scan{2}(f_idxs(1):f_idxs(end)), scan{3}(f_idxs(1):f_idxs(end)), ...
    scan{4}(f_idxs(1):f_idxs(end))];

fclose(obj_file);
```

4. Function to Propagate Orbit and Calculate Orbital Elements

```
function [Xout,OEout] = propagate_spacecraft(X0,t0,tf,A,m)
    % Description: This function accepts an initial state, initial time, final time, SRP ...
        area, and spacecraft mass in km,
    %   kg, s units and outputs the state and orbital elements at the final time.

    % Inputs:
    % X0 - [6 by 1 ] spacecraft Cartesian state vector in the ACI frame [x, y, z,   x ,  ...
        y ,   z ]T at the initial time t0
    % t0 - scalar, initial time in seconds
    % tf - scalar, final time in seconds
    % A - scalar, SRP area in km2
    % m - scalar, spacecraft mass in kg

    % Outputs:
```

```
% Xout - [6 by 1 ] spacecraft Cartesian state vector in the ACI frame [x, y, z,   x ,  ...
     y ,   z ]T at the final time tf
% OEout - [6 by 1 ] spacecraft orbital elements relative to the ACI frame [a, e, i, ...
       ,   ,   ]T at the final
% time tf in km and degrees on [  180  , 180]

% 60 sec. Time span
tspan = [t0 tf];

% Integration Tolerances
tol = 1e-10;
options = odeset('AbsTol',tol, 'RelTol',tol);

% Propagate over T.S. and save last point as output and as new initial state
[¬, Xout] = ode45(@(t,x) EOM(t, x, A, m), tspan, X0, options);
Xout = Xout(end,:)';

% Ensure orbit stays within bounds
if norm(Xout) > 3 || norm(Xout) < 0.3
    fprintf("Invalid orbit...\n")
end

% computing OE
mu = 4.892/1e9; % km^3/s^2

% position and velocity vectors
R = Xout(1:3);
V = Xout(4:6);
r = norm(R);
v = norm(V);

vr = dot(R,V)/r;

% Angular momentum
H = cross(R,V);
h = norm(H);

% compute inclination
inclination = acos(H(3)/h);

% Line of nodes
N = cross([0 0 1],H);
n = norm(N);

% compute RAAN
if n ≠ 0
    RAAN = acos(N(1)/n);
    if N(2) < 0
        RAAN = 2*pi - RAAN;
    end
else
    RAAN = 0;
end

% Eccentricity Vector
E = 1/mu*((v^2 - mu/r)*R - r*vr*V);
e = norm(E);

% Compute argument of periapse
if n ≠ 0
    if e > tol
        w = acos(dot(N,E)/n/e);
        if E(3) < 0
            w = 2*pi - w;
        end
    else
        w = 0;
    end
```

```
    else
        w = 0;
    end

    % Compute True anomaly
    if e > eps
        theta = acos(dot(E,R)/e/r);
        if vr < 0
            theta = 2*pi - theta;
        end
    else
        cp = cross(N,R);
        if cp(3) ≥ 0
            theta = acos(dot(N,R)/n/r);
        else
            theta = 2*pi - acos(dot(N,R)/n/r);
        end
    end

    % Semi major axis
    a = h^2/mu/(1 - e^2);

    % Function output
    OEout = [a, e, inclination, RAAN, w, theta]';

    % EOM function for numerical integration
    function dydt = EOM(t, X0, A, m)
        %  Extract state vector
        [X, Y, Z, VX, VY, VZ] = deal(X0(1),X0(2),X0(3),X0(4),X0(5),X0(6));

        % SRP parameters
        C_R = 1.2;
        psr = 4.57e-3; % w/r to km
        r_sun_sat = [-1, 0, 0];
        mu = 4.892/1e9; % km^3/s^2

        % relative position between parent and S/C
        rvec = [X,Y,Z];
        r = norm(rvec);

        % accelaration due to Solar radiation pressure
        Asrp = -psr*C_R*A/m * r_sun_sat;

        % r double dot eqn for the S/C
        AX = -mu*(X)/r^3 - Asrp(1);
        AY = -mu*(Y)/r^3 - Asrp(2);
        AZ = -mu*(Z)/r^3 - Asrp(3);

        % time derivative of state
        dydt = [VX, VY, VZ, AX, AY, AZ]';
    end

end
```

5. *Function to Check Target Observability*

```
function [observable, elevationAngle, cameraAngle, cost, center] = check_view(r, ...
    facetNumber, F, V, Nsc, rotate)
% Description: This function accepts an initial state, facet number, list of facets, and a ...
    list of verticies in km and
%   outputs the if the facet is observable, the elevation angle of the facet, and the ...
    camera angle of the facet.

% Inputs: r - [3 by 1 ] spacecraft Cartesian position vector in the body frame [x, y, z]T
%         facetNumber - scalar, facet index
```

```
%          F - [n x 3] matrix of vertices that form each face
%          V - [m x 3] matrix of vertex locations in implied body frame

% Outputs: observable - int, 0 for unobservable, 1 for observable
%           elevationAngle - scalar, elevation of spacecraft relative to the facet plane in ...
    radians
%          cameraAngle - scalar, angle of facet center relative to camera boresight
r = r';
% Observability Limits
bound = 15 * pi/180;
FOV = pi/ (9*Nsc);

% Pull out the individual facet numbers
F1 = F(facetNumber, 1);
F2 = F(facetNumber, 2);
F3 = F(facetNumber, 3);

% Calculate vectors between two sides of the facet
vec1 = V(F2,:) - V(F1,:);
vec2 = V(F3,:) - V(F1,:);

% Find the location of the center of the facet form geometry
center = 1/3 * (V(F1,:)+V(F2,:)+V(F3,:));

% Find the normal vector of the facet
facetNorm = cross(vec1, vec2, 2);
% Find the range, or distance, between the facet and the S/C
range = r - center;

% Calc. vector norms for use later
normFacetNorm = vecnorm(facetNorm);
normRange = vecnorm(range);
normR = vecnorm(r);

% Phase angle mask function
H = ceil(dot(rotate*-[1 0 0]', center'/vecnorm(center)));

% Calculate elevation and camera angle from geometry
elevationAngle = asin(dot(facetNorm,range,2) ./ (normFacetNorm .* normRange));
cameraAngle = acos(dot(r,range,2) ./ (normR .* normRange));

% Check observability against conditions
if elevationAngle ≥ bound && H && cameraAngle ≤ FOV
    observable = 1;
else
    observable = 0;
end

% Calculate the GR and the cost for each corresponding observable target
if observable
    GR = (FOV * normRange*1e3)/(2048 - (1408*(Nsc - 1)/9));
    cost = 0.007263/GR^2 * sin(elevationAngle);
else
    cost = 0;
end
```