Research in Industrial Projects for Students Final Report

Sponsor

The Aerospace Corporation

Regional Coverage from Space Systems

Student Members

Lisa Plucinski (Project Manager), Carleton College,
plucinski.lisa@gmail.com

Iain Carmichael (Report Coordinator), Cornell University

Jason Xu, University of Arizona

Louis Bohorquez, Cal Poly Pomona

Academic Mentor

Dr. Nam Lee, nhlee@jhu.edu

Sponsoring Mentors

Mr. Steven Hast (Lead), Steven.L.Hast@aero.org

Mr. James Gidney

Mr. Christopher Kobel

Dr. Gary Green

Mr. Gregory Henning

Dr. Ragini Joshi

Mr. Thomas Lang

Date: August 24, 2012

This project was jointly supported by The Aerospace Corporation and NSF Grant DMS-0931852.

Abstract

The Aerospace Corporation is interested in robust algorithms and mathematical solutions for regional coverage problems. This paper presents algorithms and analysis for new techniques developed by the 2012 RIPS Aerospace team toward solving two regional coverage problems. For the first problem of determining visibility periods during which any part of a region on the earth is visible from an orbiting satellite, we create a visibility function and corresponding root-finding scheme that yields precise visibility periods. For the second problem of computing visibility periods from a steerable satellite sensor to a set of target points, we reduce the problem of optimal pointing to solving a minimax problem. We detail two approaches to compute this optimal pointing direction, providing a precise mathematical solution as well as a simpler and fully implemented iterative algorithm. Analysis, results, software, and recommendations for future work are provided for each technique.

Acknowledgments

We would like to extend our gratitude towards everyone at The Aerospace Corporation and IPAM. We would especially like to thank our industrial mentors Steve Hast, Jim Gidney, Gary Green, Ragini Joshi, and Greg Henning as well as our academic mentor Nam Lee for technical support, feedback, and direction throughout our work on this project. We also appreciate the support from Mike Raugh and the rest of the IPAM staff.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Aerospace Corporation

The Aerospace Corporation was established in 1960 as a California nonprofit corporation and operates as a Federally Funded Research and Development Center (FFRDC) sponsored by the United States Air Force. The Aerospace Corporation provides technical research and hands-on engineering for space programs for national organizations including NASA and the NOAA, as well as international organizations and commercial companies.

The services that The Aerospace Corporation provides for these space programs include performance analysis, risk reduction, threat analysis, and concept design. This project focuses on one field that is salient to these services: developing and improving methods of regional coverage to and from space systems. The field has many applications including the observation and tracking of weather patterns, hurricanes, and tornadoes [8].

1.2 Overview of the Problem

Regional coverage problems can be divided into four main modeling categories:

1. visibility to *all* of a region from a sensor on an orbiting satellite,

2. visibility to *any* part of a region from an orbiting sensor,

3. visibility to an orbiting object from *all* of a region on the surface of the earth,

4. visibility to an orbiting object from *any* point within a region.

While The Aerospace Corporation currently has many regional coverage capabilities, further developing regional coverage techniques leads to robust algorithms that are more accurate and applicable on a wider class of examples.

Accurately modeling regional coverage problems also requires a variety of generalizations on the four main areas given above. These generalizations are concerned with different sensor and earth models. Many of the current techniques of The Aerospace Corporation for addressing regional coverage problems are accurate and can be generalized from their specific current applications, but in some cases The Aerospace Corporation is unsatisfied with the accuracy of their current techniques and the ability of these techniques to generalize. In addition, The Aerospace Corporation has identified some problems of interest that do not have any known solutions at present.

A past technique to model regional coverage problems is to represent the region of interest as a grid of points. To determine when a region is visible from an orbiting sensor, one needs to assess when the sensor is in line with the specified points on the grid. In practice, The Aerospace Corporation has found this technique both computationally intensive and susceptible to omitting visibility opportunities [5].

The Aerospace Corporation has improved upon these techniques and developed algorithms to compute periods of visibility for various regional coverage problems. Based on the current status of these techniques, The Aerospace Corporation proposed to the RIPS 2012 Aerospace Team three problems with the intention of refining or generalizing current methods:

1. *Satellite to Any Part of a Region*

   Refine the current technique of The Aerospace Corporation to solve for the time periods when *any* part of a region is visible from a sensor on an orbiting satellite. The current technique checks for visibility to the discretized boundaries of the region and uses a time step march along. The Aerospace Corporation posed this question to the RIPS 2012 Aerospace Team in order to improve the precision and mathematical elegance of their technique.

2. *Steerable Sensor*

   Develop a method to solve for the time periods when a set of points on the surface of the earth is simultaneously visible from an orbiting steerable sensor. The main component of this problem is to determine an optimal location to point the sensor given the set of points.

3. *Oblate Earth For Any Part of a Region to Satellite*

   The Aerospace Corporation is able to solve for visibility to an orbiting body from any point within a region using a spherical earth model. The third posed problem is to solve for these time periods using a more realistic oblate earth model.

The RIPS 2012 Aerospace Team conducted a preliminary investigation of these three problems in the order given above. In this report we discuss the development and implementation of our visibility function for a two-dimensional case for the first problem using Euclidean distances. We will also discuss how we generalized this solution to a three-dimensional spherical earth model using surface geometry. Another important component of this problem that we will discuss is a root-finding mechanism to solve for the time periods of visibility. In addition, we will discuss our developed algorithms for sensor and earth model generalizations for this first problem. In this report we also discuss the development and implementation of our algorithms for determining the optimal pointing direction for the steerable sensor problem. The first method uses projective geometry to work on a two-dimensional plane while the second method involves directly mathematically solving for the coordinates to point the steerable sensor.

1.3 Report Organization

In the following chapter, Chapter 2, we provide a technical background to the mathematical description of our approach, development of our algorithms, and description of our testing process for the regional coverage problems we present here. In Chapter 3 we state the satellite to *any* part of a region problem and steerable sensor problem and detail our mathematical approach. Chapter 4 covers the development, implementation, and testing of our algorithms for the satellite to region problem including our root finding algorithm. We also discuss applicable sensor and earth model generalizations. Chapter 5 addresses the development, implementation, and testing of algorithms for the steerable sensor problem. We begin by discussing the relevant geometry of the problem and giving an overview of our two approaches. The details of the non iterative approach are presented first followed by the iterative approach on the projective plane. We conclude this chapter with an analysis of these two approaches. In Chapter 6 we conclude this report by summarizing our accomplished work.

Chapter 2

Technical Background

In this chapter, we briefly outline the necessary background material and notation for developing, implementing, and analyzing regional coverage problems.

2.1 Earth Model and Satellite Orbit

The earth is not a perfect sphere, but much of the mathematical work is simplified when working on a sphere. The earth can be modeled more realistically by an oblate spheroid. In order to compare these two we use a flattening ratio of 0.003352859934 [1]. For our model we use a sphere with radius 6378 km, unless otherwise stated.

In order to mathematically model a satellite orbit we define six classical elements of aerodynamics used to describe the motion of an orbiting body [1]. An image depicting these elements is shown in Figure 2.1. Note that all orbits follow an elliptical path.

We now briefly describe some basic terminologies used in this report. *Semi-major axis* is the half of the orbit's longest diameter. *Eccentricity* is the amount by which an orbit deviates from a perfect circle. *Inclination* is the angle between the equatorial plane and the orbital plane. *Right Ascension of Ascending Node (RAAN)* is the ascending node such that it is rising through the equatorial plane. Also, ascending node is the pair of points where the satellite crosses the equatorial plane. *Argument of Perigee* defines where the perigee is located with respect to the ascending node. Then, perigee is the point in the orbit where the satellite is closest to the earth. *Mean Anomaly* is the angular displacement from the satellite to the perigee.

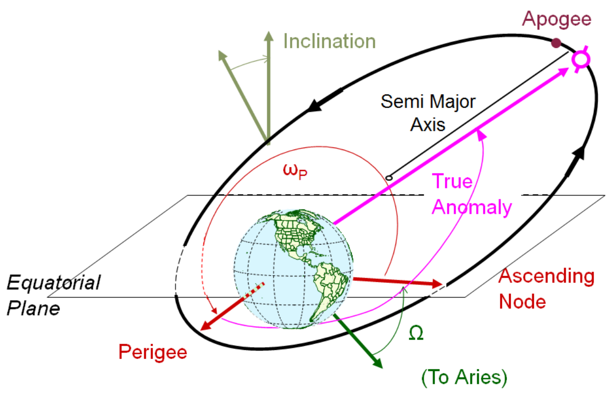


Figure 2.1: The six classical elements of astrodynamics.

In order to compare the position of the satellite with the region, we use the Earth-Centered Inertial (ECI) coordinate system to specify their positions. This coordinate system is shown in Figure 2.2 [6]. The XYZ system's origin is located at the earth's center and does not move. We define the xy plane as the equatorial plane. The x axis points towards the vernal equinox, and the y axis is 90° to the east of this. The z axis extends through the north pole from the origin. This coordinate system is used mainly because it is not fixed to the earth, so it does not change due to the earth's rotation. It deviates very slightly due to the precession of the equinox, but the effect is negligible [6].

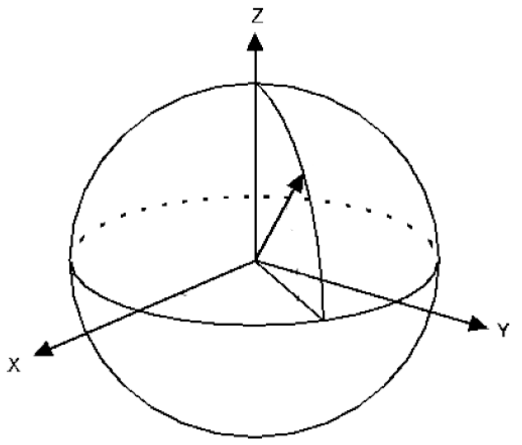


Figure 2.2: ECI coordinate system.

2.2 Convex Target Regions

A set $\mathbb{R} \subseteq \mathbb{R}^d$ is *convex* if every straight line segment connecting any pair of points in R is also contained within R. For example, circles and squares in the plane are convex, but a crescent region or an annulus is not. To mathematically describe this condition of containing all line segments between pairs of points, R is convex if for all x and y contained in R, and all $t \in [0, 1]$, the point $(1 - t)x + ty$ is still in R. Given a set of vertex points $x_1, \ldots, x_n$, the *convex hull* of these points is the smallest convex set containing all of them. Equivalently, the convex hull is the intersection of all convex sets containing the points. Convexity in the plane is a relatively simple and intuitive concept.

When discussing convex regions on the earth surface, we keep the same intuitive idea and generalize the planar definition by replacing straight line segments with geodesic segments. A geodesic is the generalization of a straight line on the surface of a manifold. It is well known that for any two points x, y on a sphere, there is a unique geodesic arc connecting the two. Then it is well defined to declare a set C on a spherical surface to be *convex* if for any x, y in C, the points $\gamma(t)$ lie in C for all $t \in [0, 1]$, where $\gamma(0) = x, \gamma(1) = y$, and $\gamma(t)$ is a parametrization for the geodesic through x and y.

2.3 Regional Coverage

The problems we focus on are all interested in when a region is visible from a sensor on a satellite. In order to understand regional coverage, here we introduce some basic terms of regional coverage. *Visibility Cone* is the cone pointing from the sensor to the earth.

Semi-Vertex Angle is the half of the angle found when looking at the cross section of a cone through the vertex and center of the base. *Footprint* is the intersection of visibility cone with the surface of the earth. We say that a sensor on a satellite is *nadir-pointing*, when the sensor points towards the center of the earth. *Center Beam* is the vector from sensor to center of earth (in nadir case). *Groundtrack*, $\mathbf{g}$ is the location where the center beam of the sensor intersects with the earth.

2.4 Visibility Function

A *visibility function* $V(t)$ is a function that can be used to determine visibility periods: intervals of time during which a satellite sensor can capture a target in question. Our target regions consist of a set of boundary vertex points and the geodesic arcs that connect them. The visibility function $V(t)$ must be continuous and take negative values precisely during time intervals in which visibility is obtained. Solving for the zeros of $V(t)$ then yields precise entry and exit times for region visibility. Some degree of differentiability is also desirable, as it allows for efficient root-finding algorithms such as Newton-Raphson.

Visibility to some part of a region occurs precisely when the satellite footprint overlaps with the region on the earth. Thus, we model our visibility function based on a notion of distance between convex bodies corresponding to the satellite footprint and target region. For our purposes, $V(t)$ must provide a consistent notion of distance between the convex bodies that becomes negative continuously as the intersection becomes non-empty. To do so, we define $V(t)$ as the minimal distance required to translate the footprint so that it is exterior and tangent to the region. The sign of $V(t)$ is negative when there is overlap and positive otherwise. By definition, $V(t)$ is zero at the border case when it is just about to overlap the region, and is thus continuous at these points. This characterization of a visibility function is general and applies over all cases, whether we are examining $V(t)$ on a planar example or on the surface of an earth model. Explicit formulae for $V(t)$ vary across cases, and will be provided in following sections accordingly.

Chapter 3

Problem Statement

3.1 Satellite to Region

The *satellite to region problem* asks the question: when can an orbiting satellite see any part of a given region? This paper provides a full solution to the case when the visibility cone is nadir pointing and circular. The Aerospace Corporation currently has capabilities to approximate this solution using discrete boundaries and a march along method for root-finding. Our solution implements a faster root-finding algorithm and finds the exact value of the roots down to numerical errors.

Our general approach to solving the satellite to region problem is to first calculate the visibility function then to find the roots of this function. The root-finding scheme is a robust version of Newton's method that exploits characteristics of the satellite orbit to efficiently find every root.

A region is defined by the convex hull of N boundary points. Recall that the satellite footprint of a nadir pointing circular visibility cone is a minor circle with the center located at $\mathbf{g}$, the groundtrack position. To calculate the visibility function we need some notion of distance between the footprint and the region. Our definition of distance is: the minimum distance required to translate the groundtrack position along the surface of the sphere such that the footprint is just exterior to the region.

We have designed an algorithm to compute the visibility function for a nadir pointing circular visibility cone. Our root-finding algorithm computes the zeros of this visibility function for a given orbit, visibility cone, and region. Additionally, our testing has confirmed that this algorithm in fact solves for the correct visibility intervals.

There are several generalizations to the simple case of the satellite to region problem. These include: an elliptical visibility cone, a non-nadir pointing visibility cone, and an oblate earth. We have investigated all of these generalizations and provide our results.

3.2 Steerable Sensor

Suppose there are several fixed points on the earth surface to be observed by a sensor on a satellite. This satellite sensor is of fixed size and shape, which translates to our assumptions of a constant semi-vertex angle and right-circular visibility cone. Determining time periods during which all points are simultaneously visible to the sensor is of interest to The Aerospace Corporation.

The Aerospace Corporation has capabilities to effectively solve this problem when the pointing direction of the sensor is fixed. This method involves calculating satellite-to-point visibility time intervals for each of the points. The intersection of these time intervals then yields the intervals during which all points can be captured.

Our problem is a generalization of this special case. We assume that the user is permitted to steer the satellite sensor to best capture all points. This advantage allows for visibility from more space locations than in the fixed sensor problem. The same question is posed, and we wish to find time intervals during which all points are visible to the sensor, which can now be steered to allow visibility at times that a nadir-pointing sensor may fail to capture all targets. Thus, the solution to the steerable problem is a superset of the solution to the fixed pointing problem. Figure 3.1 provides an illustration of satellites placed in two distinct locations, steered to capture all target points. The black visibility cone representing nadir pointing would fail to capture points at that instant in time, while the steered sensor allows for visibility from the same position.

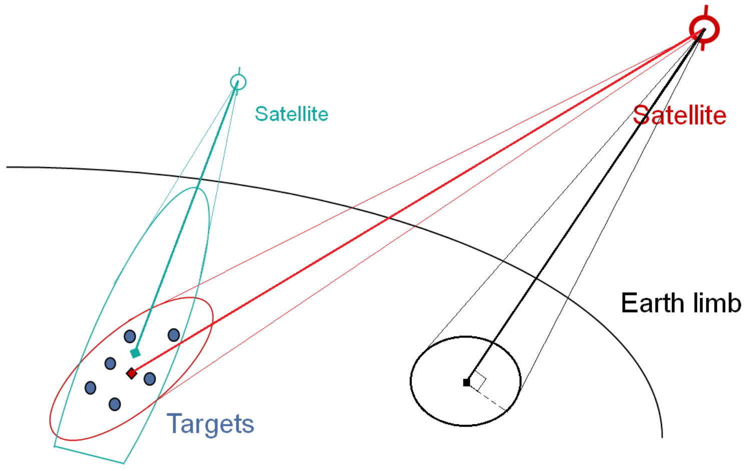


Figure 3.1: A satellite sensor steered to capture a set of target points.

The core of this problem is determining the optimal direction to steer the sensor at any given time. The RIPS Aerospace team has focused efforts on this pointing problem, whose solution can be reduced to solving a minimax problem. At a fixed time t, we wish to minimize the furthest distance from the center of the footprint to each target point with respect to an appropriate metric: call this minimax distance $g(t)$. With a solution to this subproblem, we can then create a visibility function $V(t)$ that becomes positive when $g(t)$ becomes too large that visibility is no longer obtainable from any steering direction at that time. The remainder of the problem can then be reduced to a root-finding problem, requiring the zeros of $V(t)$. The corresponding time intervals on which $V(t) < 0$ precisely yield the time periods when all target points can be captured by the steerable satellite sensor.

Chapter 4

Satellite to Region

4.1 Satellite to Region Algorithm

For this problem, we seek time intervals during which any part of a target region on the earth is visible from an orbiting satellite. We assume that the satellite has a fixed circular sensor pointed toward nadir, and a spherical model for the earth. Our solution to this problem requires first defining a visibility function whose zeros correspond precisely to the rise and set times when visibility is obtained. With a well-defined and general visibility function, we introduce a root-finding algorithm to locate its zeros. Our choice of visibility function and the corresponding root-finding scheme are tested through Matlab implementation. Our code is tested over various orbits, and results are compared with data generated by current software capabilities from The Aerospace Corporation.

4.1.1 Visibility Function

Here we introduce the visibility function used to solve for visibility periods. Because the sensor is a nadir pointing circular cone, its footprint on the earth is a minor circle. Visibility is obtained exactly when this circle overlaps the convex target region. Thus, we choose to define our visibility function based on a notion of distance between the footprint and region on the earth surface. Distance is measured in terms of surface distance and region boundaries are composed of geodesic segments.

At any time t, the absolute value of the visibility function $V(t)$ is the smallest surface distance required to translate the groundtrack position such that the satellite footprint is just exterior to the region. This quantity is defined to be positive when the footprint is outside of the region and becomes negative when the footprint overlaps with the region. Figure 4.1 illustrates this translation distance. Green circles represent the footprint location, and red circles represent where the footprint needs to be translated. The blue line shows the minimal translation distance being measured.

Define $d(t)$ as a distance function measuring the distance from the groundtrack position $\mathbf{g}(t)$ to the region boundary at time t. This function is a component of the larger visibility function, and d is given by the smallest over the distances between $\mathbf{g}(t)$ and each boundary segment:

$$d(t) = \min_i d(\mathbf{g}(t), B_i(t)), \tag{4.1}$$

where $d(\mathbf{g}(t), B_i(t))$ denotes the distance from $\mathbf{g}(t)$ to the i-th boundary, B_i. Figure 4.2

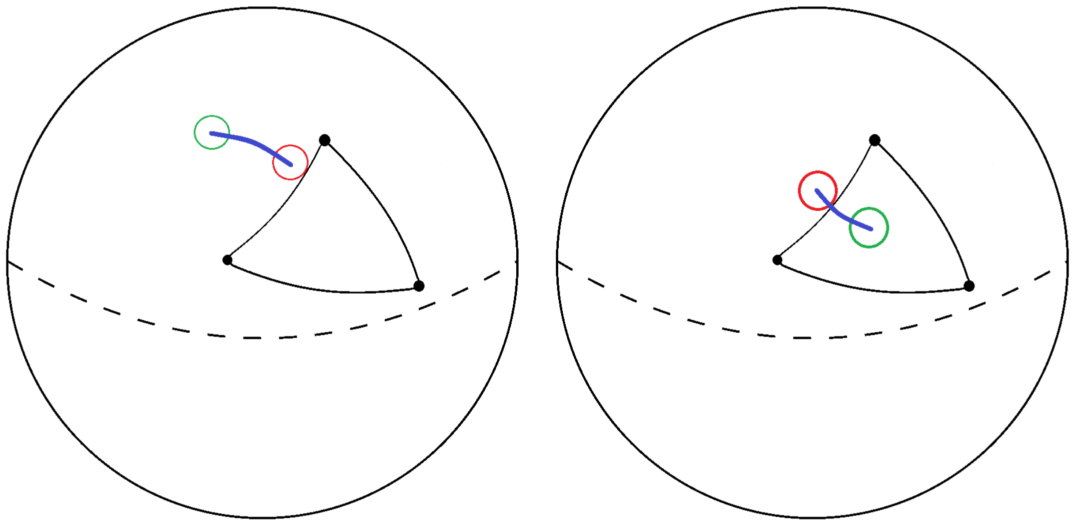

(a) Footprint entering the region (b) Footprint exiting the region.

Figure 4.1: The translation distance for a footprint.

illustrates how the value of the distance function is determined given the groundtrack position $\mathbf{g}(t)$. The green dot represents the center of the footprint or groundtrack position $\mathbf{g}(t)$ and the blue lines represent the distance from the $\mathbf{g}(t)$ to each boundary. The red line denotes the smallest of these distances.

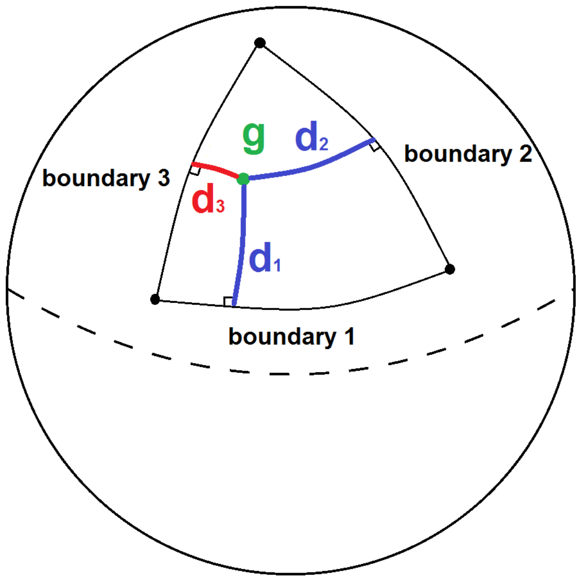


Figure 4.2: The distance function as a solution to a minimization problem.

Now let $r(t)$ define the surface radius of the footprint at time t. We define the visibility function

$$V(t) = \begin{cases} d(t) - r(t) & \text{if } \mathbf{g}(t) \notin R, \\ -d(t) - r(t) & \text{if } \mathbf{g}(t) \in R. \end{cases} \tag{4.2}$$

This equation agrees with the conceptual characterization of the visibility function presented in terms of minimal translation distances. Recall from the technical background that we desire several properties of $V(t)$. Most importantly, it must take negative values precisely during visibility intervals. It must be differentiable to some extent and continuous so that efficient root-finding algorithms that use derivative information can be applied. In particular, our definition for $V(t)$ is one that satisfies the conditions, and we give a sketch proof of this claim.

$V(t)$ is a proper distance function. *Proof sketch.* $V(t)$ is positive if the footprint is outside of the region because $d(t) > r(t)$ and therefore $V(t) = d(t) - r(t) > 0$. $V(t)$ is negative if the footprint overlaps the region. If $\mathbf{g}(t)$ is outside the region but the footprint overlaps with the region then $d(t) < r(t)$ and $V(t) = d(t) - r(t) < 0$. Furthermore, if $\mathbf{g}(t)$ is inside the region then $V(t) = -d(t) - r(t) < 0$. $V(t)$ is zero when the footprint is exterior and tangent to the region because $d(t) = r(t)$. Therefore both definitions agree at this point; $V(t) = d(t) - r(t) = 0$. □

$V(t)$ is continuous. *Proof sketch.* The two component functions $d(t)$ and $r(t)$ are continuous. Since $V(t)$ is a piecewise function it remains to check continuity at the piecewise switch. This switch happens exactly when $\mathbf{g}(t)$ lies on a region boundary. Since $d(t) = 0$ in this case, we have $d(t) - r(t) = -d(t) - r(t) = -r(t)$, so both pieces of $V(t)$ agree. Therefore $V(t)$ is continuous. □

$V(t)$ is piecewise differentiable. *Proof sketch.* $V(t)$ is calculated using $r(t)$ and $d(t)$. Both $r(t)$ and $d(t)$ are piecewise differentiable. $d(t)$ is defined in equation (4.1) as the minimum of component functions $d_i(t)$. Each $d_i(t)$ is differentiable, however taking the minimum of all these differentiable function puts kinks in $d(t)$. These correspond to times when $\mathbf{g}(t)$ is the same distance from multiple boundaries. □

Calculating the visibility function Computing the visibility function requires calculating the distance between the footprint boundary and the region boundary. Because the footprint is a minor circle, the visibility function can be calculated based on the distance from the footprint center $\mathbf{g}(t)$ to the region boundary. The footprint radius is then either added or subtracted to this distance depending on whether the region is inside or outside of the region respectively. The circular symmetry allows for this simplified point-to-curve calculation, rather than requiring a minimum curve-to-curve distance. In addition to these distance calculations, we must calculate the footprint radius $r(t)$ in terms of surface distance given the sensor's semi-vertex angle and distance from satellite to earth at time t. This radius increases with the satellite distance, but must be capped so that it can never exceed the boundary of the earth. As $V(t)$ is piecewise and depends on whether $\mathbf{g}(t)$ lies in R, we create an additional subroutine checking this condition using angular calculations. The details of these intermediate methods are included in Appendix A.

4.2 Root Finding

We need to solve for the zeros of our visibility function to obtain the precise visibility time periods. In order to do this we implemented a root-finding algorithm that uses a combination of binary search and the Newton-Raphson method. The first step is to divide our visibility function into intervals. We chose to divide the observation time period of interest into intervals the size of the orbital period. For example, suppose that we want to study the visibility to a region for a day. If our satellite orbits the earth twice in a day, then we would break our visibility function into two intervals.

For a given orbital period, we then first conduct a binary accurate to a predetermined time threshold. This binary search looks for a sign change in the value of the visibility function, which indicates the intervals that contain exactly one zero. We require the binary search threshold to be small enough so that the intervals it returns contain exactly one zero and so that we do not miss any zeros. Conversely, we do not want the intervals to be so small that the program runs inefficiently. Once the binary search has determined the intervals containing exactly one zero, we use a robust Newton-Raphson method to find the value of the zero. The values of the zeros yield the visibility time intervals.

4.3 Analysis

In this section we present our results and testing procedures for the satellite to region problem. We begin by introducing software resources provided to us by The Aerospace Corporation to be used in our testing process.

4.3.1 Astrolib and SOAP

Our algorithm for this visibility problem is implemented in Matlab. To test the algorithm we used two vital resources from The Aerospace Corporation. The first is Astrolib, a software library designed to assist The Aerospace Corporation in answering a wide range of orbital analysis problems. Astrolib is commonly used for satellite propagation, coordinate transformations, and sensor-pointing models [6]. This is a toolbox that is accessible through Matlab and gave us access to many of the functions crucial for accurate testing. Astrolib allows us to initialize satellite orbits defined using the six classical elements of aerodynamics. Given these initial conditions, Astrolib can provide ECI coordinates of the satellite necessary for calculating its footprint on the earth. Another capability in Astrolib utilized in our testing process is boundary point and satellite propagation. This allows us to consider the change in boundary point locations with respect to the ECI coordinate system due to the earth's rotation. These points are defined in ECI coordinates so that they are consistent with the satellite location defined in the same coordinate system.

The second resource we utilized was an interactive software system called the Satellite Orbit Analysis Program (SOAP), a visualization tool that lets us view the earth, regions, and satellite orbits in three-dimensional graphics [3]. Using SOAP, we were able to input all of our initial conditions for both the satellite orbit and the boundary points that we used in our Matlab code. We were then able to visualize the orbit we were testing. In addition, we obtained from SOAP a Boolean statement on when any part of the region we defined on the surface of the earth is visible from a sensor on the orbiting satellite. We then compared the time intervals obtained by SOAP to the time intervals obtained from our code. This gave us a metric for determining the accuracy of our algorithms. However, one limitation to this testing process is that SOAP solves for the time intervals using a brute force method that checks for visibility to any of the boundary points at each time step [3]. This is in contrast to our method which solves for visibility time periods to any of the boundary points as well as the lines that connect the boundary points and define the region. This means that while we were able to accurately visualize the periods of visibility using SOAP, the time outputs of visibility are not guaranteed to be accurate.

Semi-Major Axis	26000km
Eccentricity	.65
Inclination	63.435°
RAAN	0°
Arg of Periapsis	270°
Mean of Anomaly	0°

Table 4.1: Parameters of Molniya orbit.

4.3.2 Molniya Test

Our first nontrivial experiment is a Molniya orbit satellite, illustrated in Figure 4.3, where the sensor on this satellite has a nadir-pointing visibility cone with a semi-vertex angle of 10 degrees. The target region is defined by the four points specified in Table 4.2. The orbit of a Molniya orbit satellite is specified by the parameters in Table 4.1. A Molniya orbit is a high eccentricity orbit meaning that it is highly elliptical. The orbital period is approximately 12 hours and due to the high eccentricity, a satellite in this orbit spends the majority of this period covering the northern hemisphere [8]. We placed our four boundary points defining our region of interest in the northern hemisphere, obtaining large time periods of visibility.

In our test we examined the behavior of our visibility function for one day. We expected to have two large time periods of visibility given that the Molniya orbit is approximately 12 hours. Our resulting visibility function is shown in Figure 4.4.

Figure 4.4 shows two large time periods of visibility as we expected. However, in order to ensure that our algorithm is accurate we need to compare the exact time periods of

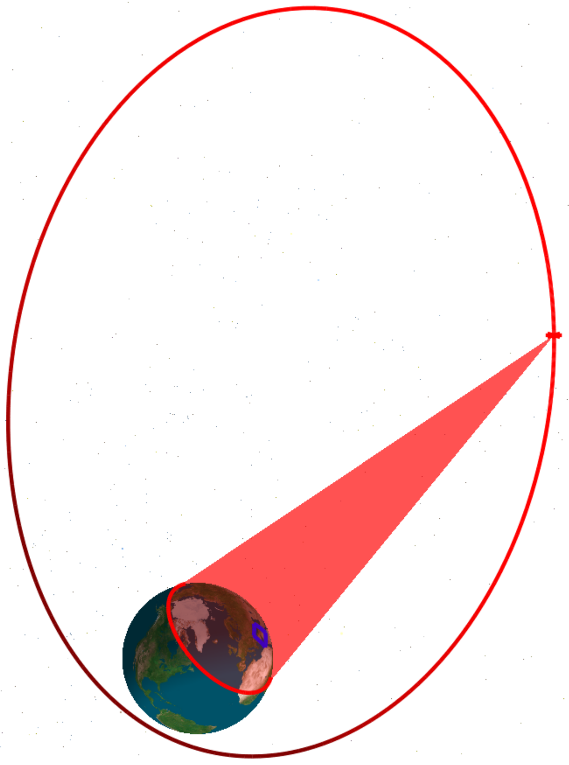


Figure 4.3: Image of a Molniya orbit, region, and satellite generated in SOAP.

Boundary Points	Lat	Lon
Point 1	30°	30°
Point 2	30°	45°
Point 3	45°	45°
Point 4	45°	30°

Table 4.2: Boundary points determining Molniya target region.

Rise	Set
194.3698	501.6042
750.0245	1320.5401
Is_True	60.96%

Table 4.3: Molniya rise and set times from Matlab.

visibility to the time periods obtained in SOAP. We know that when the visibility function yields negative values, the region is visible and when the visibility function yields positive values, the region is not visible. We implemented our root-finder in Matlab and this allowed us to solve for the zeros of our visibility function. The visibility periods corresponding to the zeros of our visibility function are shown in Table 4.3.

In Table 4.3, rise denotes the time when visibility to a region begins, and set denotes the time when visibility to a region ends. As can be seen, we have two rise times and two set times corresponding to two visibility periods. The other piece of information that is given in this table is the percentage of the time that the region is visible. In this case, the percentage is a percentage of our time period of a day. This is denoted by “Is_True”. We used these terms to denote visibility in order to match the report output from SOAP. The SOAP output for the same parameters we used in Matlab and the same time period is shown in Figure 4.5.

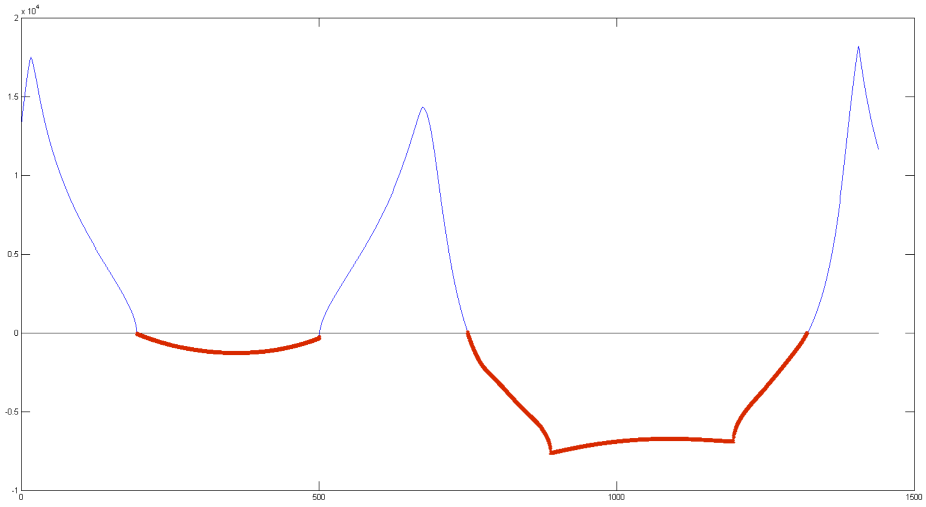
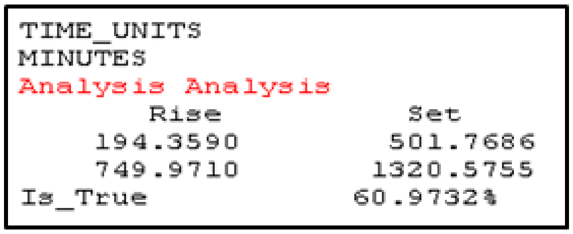


Figure 4.4: Graph of the visibility function for Molniya orbit over one day.



```
TIME_UNITS
MINUTES
Analysis Analysis
        Rise               Set
     194.3590          501.7686
     749.9710         1320.5755
Is_True               60.9732%
```

Figure 4.5: Molniya rise and set times from SOAP report.

It is clear that the two results are not a perfect match. The discrepancies between the two outputs range from 0.1644 minutes to 0.0108 minutes. The average difference of the four results is .066 minutes or about 3.96 seconds. There are a number of factors that could contribute to these differences between the two outputs. One of these is that SOAP is an approximation method based on the fact that a rise time is registered only if a boundary point falls within the visibility cone footprint. In contrast, our visibility function records a rise time when any part of the region falls within the visibility cone footprint. The accuracy of our results can be reinforced by the fact that the percentage of time during a day that the region is visible according to our code is 60.96% while the percentage of time during the day that the region is visible according to SOAP is 60.97%. This means that the two differ by only 0.01%.

4.3.3 Low Earth Orbit Test

In our second test we made changes to the satellite orbit, the boundary points defining the region, and the semi-vertex angle defining the sensor's visibility cone. We used a low earth orbit (LEO) satellite and defined our region of interest by six boundary points in

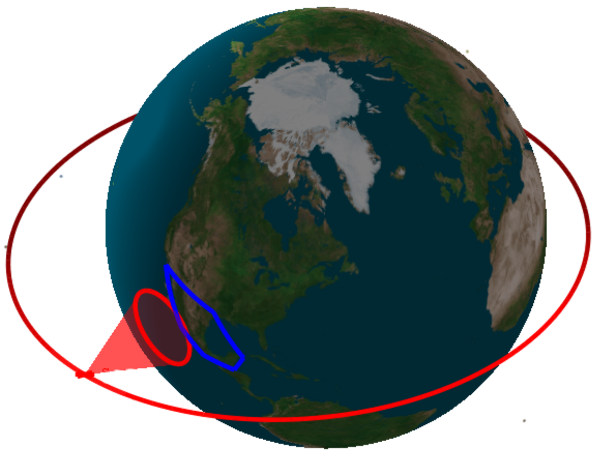


Figure 4.6: Image of a LEO satellite, region, and sensor generated in SOAP.

Semi-Major Axis	8378km
Eccentricity	.1
Inclination	22.5°
RAAN	0°
Arg of Periapsis	180°
Mean of Anomaly	0°

Table 4.4: Parameters of LEO satellite orbit.

Boundary Points	Lat	Lon
Bp1	32°	-117°
Bp2	30°	-97°
Bp3	21°	-86°
Bp4	16°	-88°
Bp5	16°	-97°
Bp6	20°	-105°

Table 4.5: Boundary points defining LEO target region.

the northern hemisphere. Table 4.5 provides the coordinates of the six boundary points, and Table 4.4 provides the parameters for the LEO satellite orbit that we studied. The LEO satellite is shown in Figure 4.6. The sensor's visibility cone was nadir-pointing with a semi-vertex angle of 22.5 degrees. The exact parameters defining the LEO satellite orbit and our boundary points are listed below.

Once again we chose to examine the behavior of our visibility function for a time period of one day. In contrast to the Molniya orbit, the orbital period of the LEO orbit is approximately 150 minutes [8]. Due to the smaller orbital period we expect to obtain a greater number of time periods of visibility, but we expect each time period to be of shorter duration than we obtained with a Molniya orbit. Our visibility function for this test is shown in Figure 4.7.

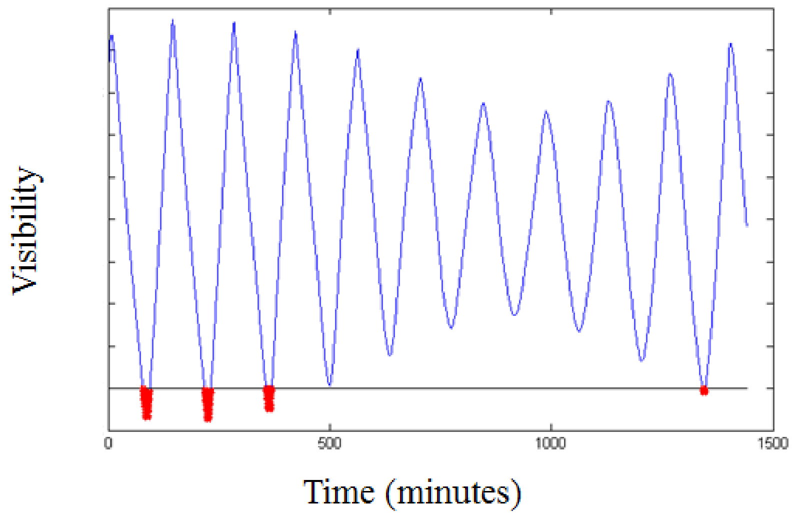
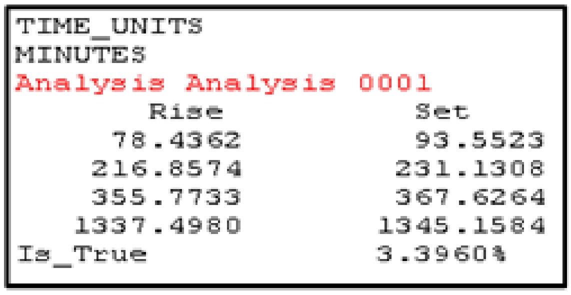


Figure 4.7: Graph of visibility function for LEO over one day.

Rise	Set
78.6313	93.7017
217.4714	231.4806
356.8522	368.1577
1341.283	1346.955
Is_True	3.21%

Table 4.6: LEO rise and set times from Matlab.



```
TIME_UNITS
MINUTES
Analysis Analysis 0001
            Rise                 Set
          78.4362              93.5523
         216.8574             231.1308
         355.7733             367.6264
        1337.4980            1345.1584
Is_True                        3.3960%
```

Figure 4.8: LEO rise and set times from SOAP report.

As can be seen in Figure 4.7, we obtain four time intervals of visibility. Once again we use the implementation of our root-finding algorithm to solve for the roots of the visibility function. The results are shown in Table 4.6.

We then obtained a report from SOAP in order to compare to our results. The results from SOAP are shown in Figure 4.8.

We notice discrepancies when comparing our results to the report from SOAP. The range in these discrepancies in this case is .1494 minutes to 3.785 minutes and the average discrepancy is 1.063 minutes. This is a much greater average difference than we attained from the Molniya orbit. We speculate that this difference cannot solely be attributed to the differences between SOAP and our code, or the fact that SOAP is only an approximation. We also have a discrepancy of .18 in the percentage of time when a region is visible throughout the day compared to SOAP. This is a significant difference when compared with the Molniya test.

4.3.4 Testing Outcomes

Overall, our testing process revealed some advantages to our algorithm, but also some limitations to our algorithm and our testing process. The number of time intervals of visibility matched between our algorithm and the SOAP report for both cases. Although some periods of visibility were slightly inaccurate we always had a corresponding period of visibility from our output to SOAP. The corresponding intervals overlapped. In addition the percentages of time when a region is visible in a day were very similar.

For the Molniya orbit it is feasible that the discrepancies between our time periods and the time periods reported by SOAP could be attributed to the approximation methods in SOAP, but the results for the LEO test will require further investigation. We noticed that when we increased the semi-vertex angle of the visibility cone our discrepancies also increased, and the largest discrepancies occurred when the visibility cone radius was greater then the earth's radius. We suspect that there is a difference in how SOAP and our algorithm calculates the radius of the footprint on the surface of the earth. We recommend this to be investigated.

Chapter 5

Steerable Sensor

5.1 Problem Framework: Steerable Sensor

The heart of this problem lies in choosing an optimal direction to point the satellite sensor. We will refer to the vector from the center of the satellite to the earth as the *center beam*. At any given time, the goal is to determine a center beam $\mathbf{C}$ such that all target points $\mathbf{x}_i$ lie interior to the visibility cone around $\mathbf{C}$. If such a beam exists, then visibility is obtainable at that time.

When the sensor is steered so that it does not point toward nadir, the satellite footprint on the earth surface is no longer a circle and need not even be an ellipse. This complicates any approach involving surface calculations to determine whether target points lie inside the satellite footprint. Instead, we choose to examine the orthographic projections of the target points onto the tangent plane T orthogonal to $\mathbf{C}$.

The satellite footprint on T is now a circle whose radius is determined by the semi-vertex angle and satellite-to-earth distance. By examining the projections of target points on this plane, we can determine whether they are captured within the footprint much more easily. We simply compare the projection distances from each point to the footprint center with the radius of the footprint. In this framework, all target points are visible if and only if no projection distance is greater than the radius of the footprint on T.

Two approaches From this point there are two possible ways to proceed. Our first approach is to choose an initial center beam $\mathbf{C}_0$ at any given time, and iteratively improve this guess with a sequence of adjustments $\mathbf{C}_1, \mathbf{C}_2, \ldots$ until all targets are captured. This idea faces the problem of being computationally expensive: each adjustment requires a recalculation of a tangent plane T_i corresponding to $\mathbf{C}_i$, and recalculation of all projections and their respective distances. A more serious flaw in this approach arises at times when it is impossible to capture all points. In this case, steering the sensor iteratively will never converge to a solution. Imposing any stopping criterion to prevent an infinite loop is either computationally wasteful after an inordinate number of iterations, or risks missing visibility intervals by terminating the algorithm prematurely. Nonetheless, we investigate this approach as it may be the simplest or most efficient method in cases that are not problematic.

The second approach is to derive a mathematical solution that yields the optimal pointing direction at any given time. Immediately locating the optimal direction avoids the iteration problem of the previous approach. At a fixed time, we may simply calculate the optimal center beam $\mathbf{C}_M$, and compare the projected distances to target points on the corresponding tangent plane T_M. If they are all less than the radius of the footprint, visibility is obtained. Otherwise, it is impossible to capture all points at that time, and no computation is wasted in hopeless attempts to correct the pointing direction. One potential drawback of this approach is that solving the optimization problem at each time step may be computationally expensive when implemented in a larger scheme.

Below we begin by discussing the non-iterative second approach to this problem. We then detail our initial approach. In practice, we recommend beginning with the iterative approach, as it is straightforward and often converges after a small number of steps. In cases that the sequence of guesses generated by the iterative approach exhibits convergence issues, the non-iterative solution provides a precise alternative. The non-iterative approach is technically also an "approximation" in that numerical calculations and methods for locating the minimum cannot guarantee perfect precision, but the bound on its error can be made arbitrarily small. As it provides a mathematically correct solution without potential convergence issues, the non-iterative approach allows for a "truth standard" for the optimal pointing problem. The iterative algorithm as well as any future algorithms towards this problem can be compared with the solution provided by our non-iterative scheme.

5.2 Approach I: Non Iterative Algorithm

5.2.1 Overview

In this approach, we reduce the optimal pointing solution to solving a minimax problem. Here we discuss the requisite geometry and definitions before detailing the optimal pointing algorithm. Given satellite and target positions as inputs at a fixed time, we create a function P_i of $\mathbf{c}$ for each target $\mathbf{x}_i$ that yields the distance between $\mathbf{c}$ and the projection of $\mathbf{x}_i$ onto the tangent plane T_c. The objective function g is then defined to be the largest of these component functions: the problem then becomes minimization of $g = \max_i P_i$, that is, minimizing the distance from $\mathbf{c}$ to the furthest target. As the beam location on the earth $\mathbf{c}$ is the variable, the value, arg min(g), that minimizes g gives the optimal pointing direction. The minimum value of g yields the smallest radius of the footprint on T_c that allows for visibility.

5.2.2 Choice of Minimax Problem

Here we briefly digress to discuss the choice of a minimax framework as opposed to other characterizations of optimization. For instance, it would be simpler to minimize an objective function f that outputs the average distance to each target rather than the maximum of each distance. However, we choose to model the problem as a minimax because it is the only framework giving a precise, *necessary and sufficient* condition for visibility. If the largest projected distance is less than the radius $r(t)$ of the footprint on T at a given time t, then all distances are less than $r(t)$ and thus all targets are captured by the sensor beam. This shows sufficiency. To see that it is necessary, if any projected distance $P_i(t)$ is greater than $r(t)$, then the sensor beam fails to capture $\mathbf{x}_i$. Thus, all points can be captured precisely when the largest projected distance, characterized by the value of $g(t)$, is less than or equal to $r(t)$. When implementing this pointing algorithm to a larger scheme to solve for visibility time intervals, this choice of g leads to a root-finding problem requiring zeros of a visibility function $V(t) = g(t) - r(t)$. Our choice of g as a minimax function ensures that this resulting visibility function $V(t)$ is negative precisely when all targets can be captured by the steerable sensor.

5.2.3 Geometry and Components of g

Before solving for the minimum of the objective function g, we need to understand the component functions P_i and how they encompass the geometry of the problem. Again, $\mathbf{C}$ denotes the center beam vector from the sensor, and $\mathbf{c} \in \mathbb{R}^3$ denotes the intersection point of $\mathbf{C}$ with the earth surface. The center beam vector can be thought of as $\mathbf{C} = \mathbf{s} - \mathbf{c}$, where $\mathbf{s}$ is the location of the satellite. As g is a function of the pointing direction, $\mathbf{c}$ is treated as a variable.

To reduce dimension, we apply an orthographic projection[1] from a downward vertical perspective to obtain the shadow of $\mathbf{c}$ on the xy plane. This is equivalent to temporarily disregarding the z-coordinate of $\mathbf{c}$: this initial orthographic projection step maps $\mathbf{c} = (c_1, c_2, c_3) \mapsto (c_1, c_2)$ [2]. Figure 5.1 illustrates the projection of a point directly down onto its shadow on the xy plane. The z-coordinate can then be recovered when necessary by using the equation of the earth model, whether sphere or oblate spheroid, given the xy plane coordinates. Reducing the variable domain to $\mathbb{R}^2$ allows each P_i component and hence also g to become functions of two variables, taking values as a surface in $\mathbb{R}^3$.

Recall that at any location of $\mathbf{c}$, we wish to compare the projected distances from $\mathbf{c}$ to each target $\mathbf{x}_i$ on the instantaneous tangent plane T_c. We want each component function P_i to contain these projected distances requiring tangent plane information, but do not wish to recalculate the tangent plane at each step. It is possible to bypass the tangent plane calculations, however, by directly calculating the projected distances involving dot products. First, define $D_i := \mathbf{x}_i - \mathbf{c}$, the vector pointing directly from $\mathbf{c}$ to vertex $\mathbf{x}_i$. Then a closed form formula for the projected distance between $\mathbf{x}_i$ and $\mathbf{c}$ is given by

$$P_i = \cos(\theta)\|D_i\|,$$

where θ is the angle between T_c and D_i. See Figure 5.2.

While this is the simplest geometric characterization of P_i, it is not how we choose to calculate its value. Instead, we want to use the quantity $\cos(\phi)$ where ϕ is the angle between $\mathbf{C}$ and D_i. This value of $\cos(\phi)$ follows directly from a normalized dot product, and avoids rounding errors resulting from inverse trigonometric functions required to calculate θ. Using a trigonometric identity, we can equivalently calculate P_i with the formula:

$$P_i = \sqrt{(1 - \cos^2(\phi))}\|D_i\|.$$

In terms of dot products, this is equivalent to

$$P_i = \sqrt{1 - \left(\frac{D_i \cdot \mathbf{C}}{\|D_i\| \cdot \|\mathbf{C}\|}\right)^2}\|D_i\|.$$

Expanding one more time so that the expression can be written explicitly in terms of $\mathbf{c}$, we obtain

[1]Note that this is simply a step to simplify the problem into a two dimensional domain, and the choice of projection plane is arbitrary. The xy axis was chosen here because it is easy to implement, as the projection of a point is equivalent to simply disregarding its z-coordinate, but another reference plane such as the tangent plane on the earth orthogonal to a nadir-pointing vector would work as well.

[2]Technically, we must map (c_1, c_2, c_3) to $(c_1, c_2, \text{sign}(c_3))$. The sign of c_3 is necessary so that the sphere equation recovers $\mathbf{c}$ on the correct hemisphere of the earth, as a square root is involved.

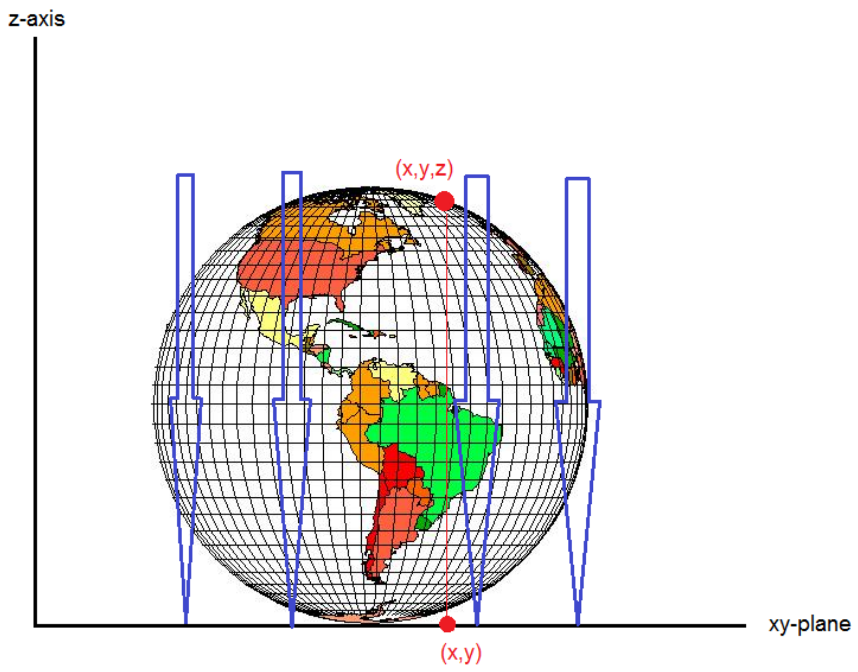


Figure 5.1: Downward orthographic projection.

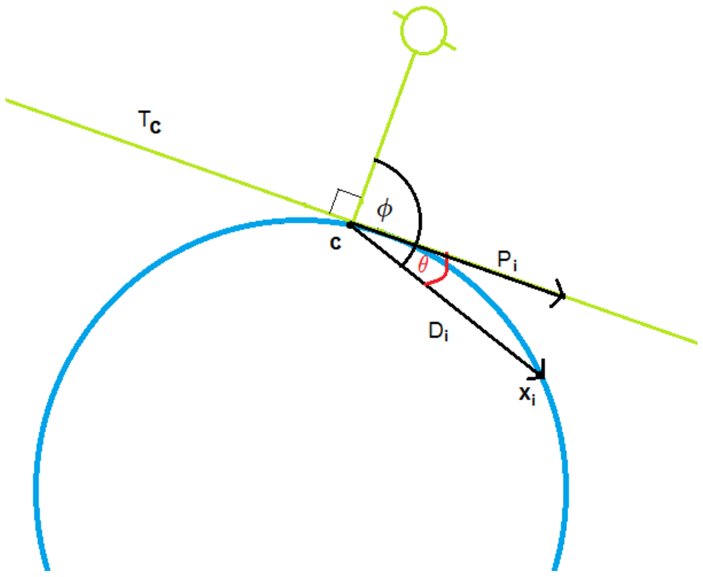


Figure 5.2: The projections P_i onto the instantaneous tangent plane T_c.

$$P_i(\mathbf{c}) = \sqrt{1 - \left(\frac{(\mathbf{x}_i - \mathbf{c}) \cdot (\mathbf{s} - \mathbf{c})}{\|\mathbf{x}_i - \mathbf{c}\| \cdot \|\mathbf{s} - \mathbf{c}\|} \right)^2} \|\mathbf{x}_i - \mathbf{c}\|. \tag{5.1}$$

In this form, the only variable is **c**. We have already noted how **c** can be represented as a variable in the xy plane. Thus, the component functions are functions of two variables $P_i(x, y)$ whose graph is a surface in $\mathbb{R}^3$, and similarly the objective function $g(x, y) = \max_i P_i(x, y)$ is also a function of two variables.

Locating the minimum of g To locate the minimum of g, one can apply general optimization methods such as Nelder-Mead method or minimizing by majorization [7]. Derivative-based methods such as Steepest Descent and Newton's optimization algorithm are likely to fail because g may not have a smooth gradient; it has a cusped nature as the maximum of several distinct functions. Similarly, standard constrained optimization techniques such as the Lagrange multiplier method do not apply as they rely on a well-behaved gradient.

As of now, implementation of this algorithm uses a simplistic method to locate the minimum. Recall that the problem is being solved on a planar domain; we have reduced dimension of the center beam argument $\mathbf{c}$ as well as of the target points by considering their shadow on the xy plane via an orthographic projection.

The current Matlab code simply evaluates each P_i over the area of a rectangular hull of xy points. Denote the smallest x-coordinate of target points $\mathbf{x}_i$ by x_m, and the largest x-coordinate among the target points by x_M. Similarly, define y_m and y_M: then Matlab evaluates each P_i over a meshgrid over the region $[x_m, x_M] \times [y_m, y_M]$. The surface for the objective function g is then constructed over a loop by assigning the largest P_i value to g at each point in the meshgrid. The user can fine tune the meshgrid spacing as an input, trading off precision for runtime or vice versa.

When constructed in this way, it is straightforward to locate the minimum of g by searching and returning the smallest entry among the array containing calculated values of g. This is a relatively inefficient approach, but as a first implementation, this method is straightforward and transparent, avoiding potential conceptual flaws. Furthermore, it is desirable to generate all values for each surface at this stage to generate graphics of the entire surfaces. These graphs allow verification of conjectures into the geometry of the problem by visualizing the true geometry generated by Matlab over various test cases.

Minima along intersections We may exploit the nature of our particular problem to locate the minimum of g more directly. While we have not implemented the following in software, we can significantly reduce the search space by showing that the minimum of g is guaranteed to occur at intersections of the P_i functions. Because each $P_i(x, y)$ function characterizes distance from $\mathbf{c}$ to some fixed $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3})$, they each have a minimum value of zero at $\mathbf{c}_{\min} = (x_{i1}, x_{i2})$. This is the case when the sensor beam points directly toward the target point $\mathbf{x}_i$. When $\mathbf{c}$ varies away from this minimum, the value of P_i continuously increases in every direction: the resulting graph of each P_i surface is thus a bowl-shaped sheet. Figure 5.3 provides a visualization of these surfaces. As the objective function g is defined as the largest of each of these P_i surfaces, it also has a "bowl" shape that has cusped edges at intersections of P_i functions. The following image was generated by our Matlab code as an intermediate step to solve one case of this problem.

The minimum of g must occur along an intersection of the P_i surfaces. While we have not found a way to nicely implement the following calculation in Matlab, one may proceed by calculating the curves of intersection between the P_i surfaces; we may call this family of curves $\ell_1, \ldots, \ell_n$. As each ℓ_i is simply a curve in $\mathbb{R}^3$, it is not computationally expensive to locate its minimum m_i, given an expression for ℓ_i.

The next step is to only consider minima that are "dominating" in the sense that the function value $\ell_i(m_i)$ is no less than any other surface value $P_j(m_i)$ for surfaces other than

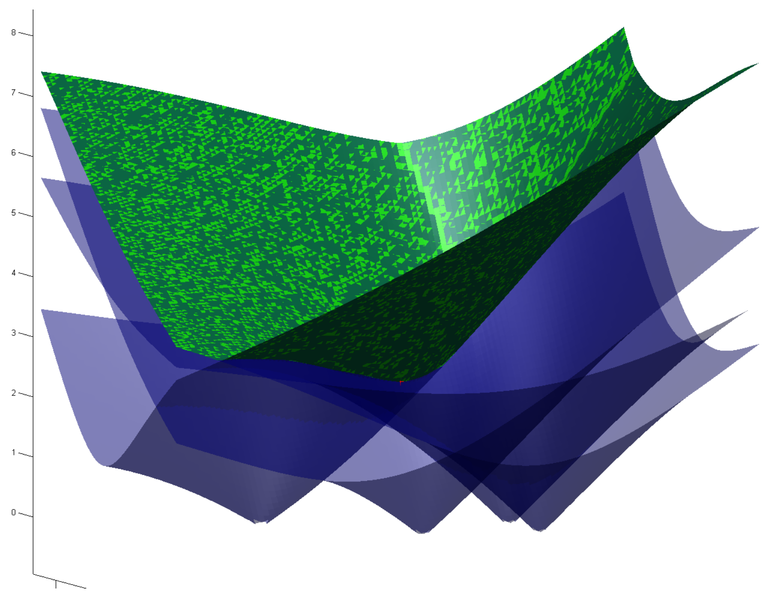


Figure 5.3: Blue P_i surfaces and green dominating surface g.

the two that intersect along ℓ_i. The geometry of the problem ensures that the smallest of these "dominating" minima corresponds to the minimum value of g. Figure 5.4 provides a cross-sectional view of the P_i bowl surfaces; each corresponding cross section is labeled Proj_i. The stars represent "dominating" minima, and the black crosses denote intersections that can be disregarded. The gold star is the smallest of the dominating minima, and is thus the solution to the minimax problem.

There may not exist a general formula to analytically calculate these curves of intersection, and we have not been successful in finding such an expression. However, it may be

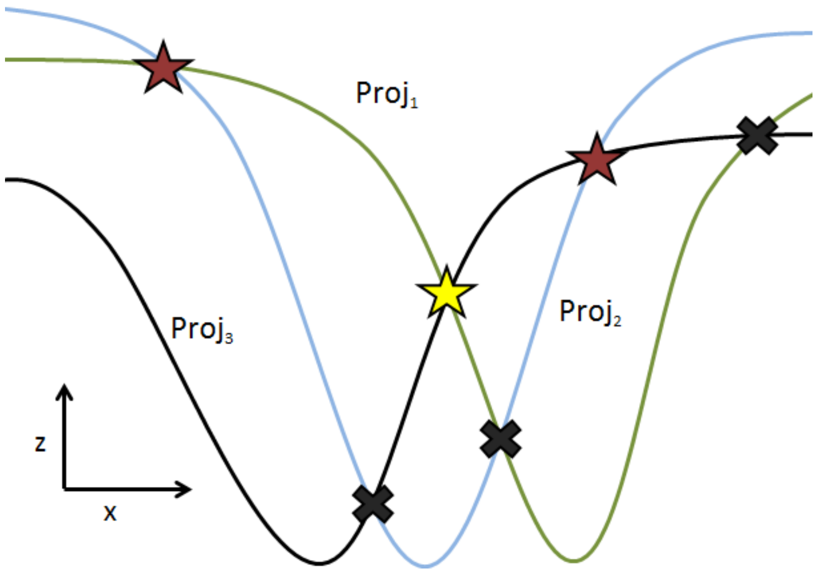


Figure 5.4: Cross section showing minima along intersections of P_i.

possible to accomplish this using mathematical software such as Maple or Mathematica.

Smoothing the objective function The primary reason most general and widely used minimization methods do not apply to our objective function g is because of non-differentiability. Because g is not smooth, straightforward optimization with constraint techniques such as Lagrange multiplier method cannot be applied. However, we may provide approximations of arbitrary precision by exploiting an alternative definition of the L_∞ norm [7]. For any vector $\mathbf{z} \in \mathbb{R}^N$, one has

$$\max_{1 \le j \le N} \mathbf{z}[j] = \lim_{s \to 0+} s \log \left(\sum_{j=1}^{N} e^{\frac{\mathbf{z}[j]}{s}} \right),$$

where $\mathbf{z}[j]$ denotes the j-th coordinate of $\mathbf{z}$. While strict equality only holds in the limit, for small positive s, we can approximate the non-smooth objective function g using this formula, replacing g by a family of smooth functions g_s indexed by the parameter $s > 0$. Thus, as $g = \max_i P_i$, we have

$$g = \max_{1 \le i \le N} P_i = \lim_{s \to 0+} s \log \left(\sum_{i=1}^{N} e^{\frac{P_i}{s}} \right)$$

These approximations g_s are infinitely differentiable, and arbitrary degrees of precision can be attained by choosing appropriately small parameters s. That is, for any desired ε-threshold of accuracy, there is a corresponding sufficiently small $s > 0$ such that the error between $g(\mathbf{c})$ and the approximating function $g_s(\mathbf{c}) = s \log \left(\sum_{i=1}^{N} e^{\frac{P_i(\mathbf{c})}{s}} \right)$ is within ε. By representing g with a smooth approximation g_s, standard smooth optimization techniques can be applied. We may solve for the minimum by setting $\nabla g_s = 0$, applying any boundary constraints or checking boundaries when appropriate.

5.3 Approach II: Projective Plane

5.3.1 Minimax Location Problem

In this second approach we once again use the minimax location method for optimal pointing for the steerable sensor. This means that we want to minimize

$$f = \max_i d(c, \mathbf{x}_i) \tag{5.2}$$

where c is the variable point, $\mathbf{x}_i$ denote the target points, and $d(c, \mathbf{x}_i)$ denotes the distance between the two points c and $\mathbf{x}_i$.

In this method we minimize Equation (5.2) by minimizing the maximum 2D Euclidean distance between a variable point c' and the projected target points $\mathbf{x}'_i$ on a projective plane. We define the projective plane by a viewing axis pointing from the sensor on the orbiting satellite to the center of the earth. We project the $\mathbf{x}_i$ defined in 3D space to 2D space meaning that $\mathbf{x}'{}_i = (a'_i, b'_i)$ are the points on interest on the plane and $c' = (x', y')$ is the variable point. We can then optimize Equation (5.2) using Euclidean distances on this projective plane such that

$$d(c', \mathbf{x}'_i) = \sqrt{(x' - a'_i)^2 + (y' - b'_i)^2}.$$

5.3.2 Projective Plane Method

The $\mathbf{x}_i$ are defined in 3D space on the surface of the earth with respect to the center of the earth e. We conduct a 3D to 2D projection using a vertical perspective projection. We first use a camera transform matrix to define the $\mathbf{x}_i$ with respect to the location of the sensor s. We then project the $\mathbf{x}_i$ on a projective plane n defined by a viewing axis that points from s to e. The $\mathbf{x}_i$ on the projective plane are denoted by $\mathbf{x}'_i$. Once the $\mathbf{x}_i$ are defined on the projective plane, we take the convex hull to obtain the extreme points t_i. In order to minimize the maximum 2D Euclidean distance on the projective plane, we find the circle of smallest radius that circumscribes all t_i on the projective plane. The center of this smallest circle is the location c' where we point $\mathbf{C}$ on the projective plane. We then take the inverse projection of c' to obtain the location to point $\mathbf{C}$ with respect to e. In order to check for simultaneous visibility, we compare the largest projection distance on T to the radius of the visibility cone on T. The projection distances for a given $c = (x, y, z)$ are calculated using the form of Equation (5.1) where c, $\mathbf{x}_i$, and s are all defined in $\mathbb{R}^3$.

This becomes an iterative method due to the fact that the projective plane n is different from T. In particular, this means that the optimal location c' to point $\mathbf{C}$ on the projective plane does not directly correspond to the optimal location c to point $\mathbf{C}$ on the surface of the earth. We then invoke an iterative method where we first compute c' on the projective plane and take the inverse projection to define this point in $\mathbb{R}^3$ with respect to e. We then compute another vertical perspective projection defining a viewing axis pointing from c to e. The iterations continue in this way until the latitude and longitude of c_{i+1} is the same at the latitude and longitude of c_i, or until c is projected back onto the surface of the earth. When either of these occur, this means that this c directly corresponds to c' on the projective plane in the sense that both are the optimal pointing direction for $\mathbf{C}$. At the end of the iterative process, the largest projection distance on T is compared to the visibility cone radius on T. If the largest projection distance is less than or equal to the visibility cone radius, then simultaneous visibility occurs. If not, then simultaneous visibility cannot occur with the given visibility cone.

5.3.3 Vertical Perspective Projection

The location of $\mathbf{x}_i$ on the surface of the earth and the location of the sensor on the orbiting satellite are given in terms of latitude, longitude, and altitude. We begin by converting these to Cartesian (x, y, z) coordinates in the ECI coordinate frame F_E. We then project the $\mathbf{x}_i$ from 3D space to a 2D projective plane by a vertical perspective projection. This projection method is analogous to taking a photograph of the surface of the earth from a camera in space [2]. In this case the camera location h is the location of the sensor s, the viewing axis is the negative z' axis defined by the vector from s to the center of the earth e, the near plane n is defined orthogonal to the viewing axis and tangent to the surface of the earth, f is defined orthogonal to the viewing axis and tangent at e. The field of view $a = \pi/2$ allowing the sensor to point in any direction on one hemisphere of the earth.

Note that we define n and f with respect to s and that by defining n at the surface of the earth and f at e, any point on the surface of the earth in the same hemisphere as the sensor will be located within the viewing frustum of the sensor. A representation of the vertical perspective projection and the viewing frustum is shown in Figure 5.5. A two-dimensional cross-section of the vertical perspective projection of a point on the surface of a sphere is shown on the left. The point $\mathbf{x}_i$ is a point defined in 3D space and $\mathbf{x}'_i$ is the projection of this point onto n. A representation of the three-dimensional viewing frustum is shown on the right. As can be seen, the visible points on the surface of the sphere are bound by $x \in [r, l]$, $y \in [t, b]$ and $z \in [n, f]$.

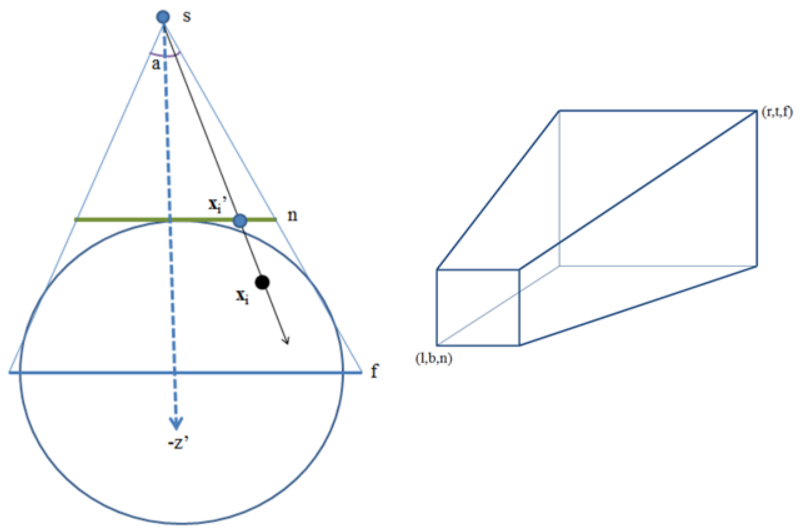


Figure 5.5: Vertical perspective projection and viewing frustum.

In order to complete a vertical perspective projection two matrices are necessary. The first is a camera transform matrix. This will take the $\mathbf{x}_i$ in the ECI coordinate system F_E centered at e and define them in homogeneous coordinates in F_H centered at s [2]. Before using the camera transform matrix we need to define a new coordinate system with the sensor at the origin. To do this we define the negative z' axis by the vector pointing from s to e. The only conditions on the positive y' and x' axes are that they define the vertical and horizontal directions, respectively. Our method defines y' in the plane defined by y in F_E and z' in F_H. The vectors $\mathbf{x}'$ and $\mathbf{y}'$ are then defined by

$$\mathbf{x}' = \frac{\mathbf{y} \times \mathbf{z}'}{|\mathbf{y} \times \mathbf{z}'|}$$

and

$$\mathbf{y}' = \mathbf{z}' \times \mathbf{x}'.$$

By defining F_H in this way, $\mathbf{y}'$ is in the $\mathbf{y} - \mathbf{z}'$ plane, all vectors are unit vectors, and all axes are mutually perpendicular. This means that F_H is an orthonormal frame [2]. The only case where this method breaks down is if $\mathbf{y} = \pm\mathbf{z}'$ meaning that $\mathbf{y} \times \mathbf{z}' = 0$. We then have to define a different vector in the vertical direction besides $\mathbf{y}$. Once we define these three axes of F_H, we can solve for the camera transform matrix.

We then calculate the projection matrix based on the parameters defined above. We know that our projected coordinates can be determined by the properties of similar triangles. However, in order to define our projection matrix we also need to confine the potential input and output coordinates to the geometry of our problem by a process called *clipping* [2]. After this process the final projected coordinates $(x_p z', y_p z', z_p z', wz')$ are then given by

$$\begin{bmatrix} x_p z' \\ y_p z' \\ z_p z' \\ w z' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} A, \tag{5.3}$$

where each point of interest is defined in homogeneous coordinates in the coordinate system F_H and

$$A = \begin{bmatrix} \frac{1}{r}\cot(\frac{a}{2}) & 0 & 0 & 0 \\ 0 & \cot(\frac{a}{2}) & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & -\frac{fn}{f-n} & 0 \end{bmatrix}.$$

We obtain the points $\mathbf{x}_i' = (a_i', b_i')$ on the projective plane in F_H by dividing by the fourth coordinate:

$$a_i' = \frac{x_p z'}{w z'},$$

and

$$b_i' = \frac{y_p z'}{w z'}.$$

5.3.4 Smallest Circle Algorithm

Now that the $\mathbf{x}_i$ are defined in 2D space on the projective plane, we minimize the maximum Euclidean distance from the variable point c' to the given target points $\mathbf{x'}_i$. Minimizing the maximum Euclidean distance on a 2D plane is a problem that has been solved before. We begin by taking the convex hull of the $\mathbf{x'}_i$ to reduce the number of calculations necessary to the extreme points t_i. We are able to do this because we know that the maximum distance from c' to $\mathbf{x'}_i$ concerns the extreme points alone.

We now want to optimize by minimizing Equation (5.2). Minimizing the maximum Euclidean distance is equivalent to $\min_{r,x,y} r$ such that

$$\sqrt{(x-a_i)^2 + (y-b_i)^2} \leq r \tag{5.4}$$

where $(i = 1, \ldots, m)$. The solution to minimizing the maximum Euclidean distance on a 2D plane is then to compute the circle of smallest radius that encloses all points of interest. An algorithm that finds this circle of smallest radius that circumscribes all points is called a smallest circle algorithm. In order to determine the circle with the smallest radius that will encompass all t_i on the projective plane we implemented a smallest circle algorithm developed by Emo Welzl in 1991 [9]. This is a recursive algorithm that runs in linear time.

5.3.5 Projection Distances

The output of the smallest circle algorithm gives the radius and the center of the smallest circle that circumscribes the t_i. The center of the circle gives the $c' = (c_x', c_y')$ coordinates of where to point $\mathbf{C}$ on the projective plane. We then take the inverse projection of c' to obtain c in $\mathbb{R}^3$ with respect to the center of the earth. The inverse projection is done in a series of steps. The first is to compute

$$\begin{bmatrix} x z_p & y z_p & z z_p & w z_p \end{bmatrix} = (\begin{bmatrix} c_x' & c_y' & -n & 1 \end{bmatrix} A^{-1}) F^{-1}, \tag{5.5}$$

where F is the camera transform matrix. We then divide by the fourth coordinate to obtain $c = (x, y, z)$ in F_E. We then want to determine if simultaneous visibility occurs to all $\mathbf{x}_i$ when we point $\mathbf{C}$ at this location. In order to determine if simultaneous visibility occurs we work on the tangent plane T orthogonal to $\mathbf{C}$ because the visibility cone footprint is a circle on this plane. We compute the largest projection distance on T using Equation (5.1) and compare this to the footprint radius on T. An issue that arises here is that the projective plane and T are defined at two different locations.

The projective plane is defined to be tangent to the surface of the earth and orthogonal to the viewing axis. In contrast T is defined tangent to the surface of the earth and orthogonal to the vector from s to c. This means that the computed location c' to point $\mathbf{C}$ on the projective plane does not directly correspond to the computed location c to point $\mathbf{C}$ on the surface of the earth. The point t_i that defines the maximum distance on the projective plane may not correspond to the same point $\mathbf{x}_i$ that defines the maximum distance on T.

This means that if the largest projection distance on T is less than or equal to the radius of the visibility cone, then we have not necessarily determined the optimal pointing direction c_M for $\mathbf{C}$, although we have determined a pointing direction that can capture all of the $\mathbf{x}_i$ simultaneously for the given radius of the visibility cone. Even more worrisome is the fact that we could miss a visibility period when the largest projection distance on T is greater than the radius of the visibility cone. In order to fix both of these issues at once, we employ an iterative method that improves upon the pointing direction at each iteration and converges to the optimal location c_M to point the sensor on the surface of the earth.

5.3.6 Iterations and Stopping Criteria

The first step of the iteration process is to compute c'_0 as described above using a vertical perspective projection with the viewing axis defined by the vector from s to e. We then take the inverse projection of c'_0 to obtain c_0 with respect to e. This point c_0 is given in Cartesian coordinates in F_E. We convert these to spherical coordinates to obtain the latitude, longitude, and altitude of this point. The second iteration is to conduct another vertical perspective projection, but this time the viewing axis is defined from the vector connecting c_0 to e. On this new projective plane we use the smallest circle algorithm to compute a c'_1. We then take the inverse projection of c'_1 to obtain c_1 with respect to e. We convert the coordinates of c_1 to spherical coordinates and compare with the coordinates of c_0. If the latitude and longitude is the same, then we have determined c_M and we exit the iterative process. If the two pairs of latitude and longitude differ, then we proceed with another iteration.

We conduct another vertical perspective projection, this time defining the viewing axis by the vector from c_1 to e. From the smallest circle algorithm we obtain a point c'_2 to point $\mathbf{C}$ on the projective plane. We take the inverse of this point to obtain c_2 with respect to e. Once again we compare the latitude and longitude of c_2 and c_1. This then becomes an iterative method where each time the pointing direction is improved by a small amount. These iterations continue until the latitude and longitude of c_{i+1} is equal to the latitude and longitude of c_i, or until the altitude of c_i puts the point on the surface of the earth. Both of these indicate that the optimal location c' to point $\mathbf{C}$ on the projective plane corresponds directly to the optimal location c_M to point $\mathbf{C}$ on the earth's surface.

Once the iterations end, we are then able to compare the largest projection distance on T to the radius of the visibility cone on T. If the largest projection distance is less than or equal to the radius of the visibility cone, then simultaneous visibility occurs and if the largest projection distance is greater than the visibility cone's radius, then simultaneous visibility is impossible with the given semi-vertex angle of the visibility cone.

We anticipate that once c_M is obtained we will be able to use a process similar to one described in the first approach to solve for the time periods of simultaneous visibility. This function will be continuous and piecewise differentiable. We expect it to be piecewise differentiable due to the fact that the $\mathbf{x}_i$ that is farthest from c will change as the sensor on the orbiting satellite moves with respect to the $\mathbf{x}_i$.

There are potential issues with utilizing an iterative method. We need to determine effective stopping criteria for cases where simultaneous visibility is impossible. There are two main cases where simultaneous visibility is not possible. The first is where the points are not all within view of the steerable sensor and the second is where the visibility cone is not large enough to capture all $\mathbf{x}_i$. In the first case, if a point is not in the viewing frustum of the steerable sensor, and therefore not on the same hemisphere as the satellite, then we ignore this point. One simple check for if it is impossible to contain all $\mathbf{x}_i$ with the given visibility cone requires choosing any point inside of the $\mathbf{x}_i$ defined on the surface of the earth and comparing the diameter of the visibility cone on the plane tangent to this point to the sum of the two greatest projection distances on this plane. If the diameter is less than this sum, then simultaneous visibility is impossible. We do not know if this covers all cases and we still need to consider whether this iterative method always converges.

5.4 Analysis

5.4.1 Non-iterative Algorithm

The mathematical derivation of the non-iterative algorithm guarantees optimality of the pointing solution. We can verify this by generating the intermediate surfaces and figures of the pointing solution through Matlab and visualizing the solution. Since there is no current optimal pointing capability in software used by The Aerospace Corporation, we analyze the performance of this algorithm by heuristically evaluating the visualizations of its results.

First, we consider three target points spaced evenly and symmetrically about the North Pole. In Figure 5.6, these targets are denoted by red stars, and the transparent blue cone corresponds to the smallest visibility cone necessary to capture all points. The green satellite is placed directly above the north pole. From a top view provided in Figure 5.7, it is clear that the optimal pointing direction is calculated to be directly on the North Pole. This is the obvious correct answer for this simple case, allowing us to check against intuition.

Figure 5.8 provides a visualization of the intermediate step in the algorithm. The transparent blue surfaces correspond to the $P_i(\mathbf{c})$ functions that characterize distance on the tangent plane $T_\mathbf{c}$ from $\mathbf{c}$ to each respective $\mathbf{x}_i$, and the green surface is the objective function $g(\mathbf{c})$. Again, these surfaces are calculated pointwise over a grid of points. The minimum of g is then located by returning the smallest value in the array containing values of g over the grid, and is marked below by a red cross. We see that the minimum occurs along the intersections of P_i, as mentioned during the discussion of this algorithm.

For our next example, consider four points whose locations have no obvious symmetry and a non-trivial satellite location. Nevertheless, the algorithm holds for an arbitrary number of points. While the optimal pointing location is not obvious in this example, Figure 5.9 shows that the Matlab solution very tightly captures all target points. Any smaller radius would fail to capture the two points that are just touching the footprint.

Figure 5.10 illustrates a feature of the algorithm to automatically detect which pair of points determines the largest projected distance, thus determining the smallest necessary

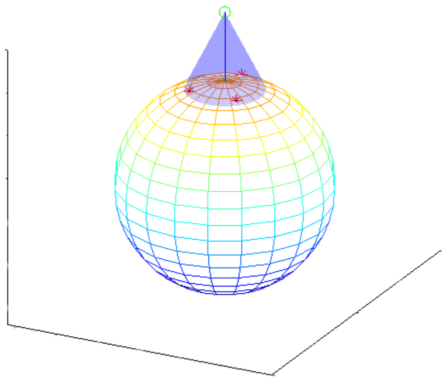


Figure 5.6: A satellite capturing three symmetrically placed targets.

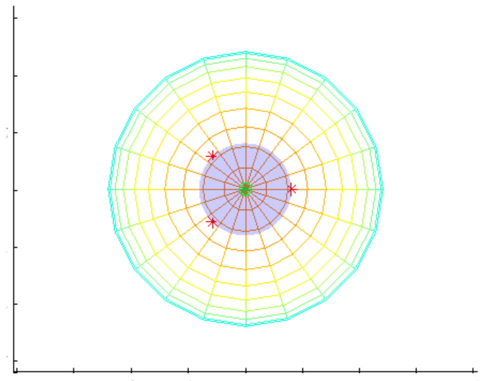


Figure 5.7: Top view of Figure 5.6.

footprint radius. We consider visibility cones from two distinct satellite positions in space capturing the same set of three target points. In earlier approaches to this problem involving planar calculations, this salient pair of points was not easy to identify. However, the current solution automatically does so as shown in the following set of figures. We see in Figure 5.10 that the furthest pair of points from the satellite's point of view changes at some time between the first satellite position and second satellite position. Notice how our solution has accounted for this change, and in each case the satellite footprint captures all target points in what visually appears to be a minimal way.

Even with this rather rudimentary approach, the optimal point calculation does not require much computational time. The images presented above were generated with a meshgrid spacing of .1 units, but the earth radius was artificially small to speed up calculations. Each was generated within five seconds. Even using a more appropriate spherical earth model with radius 6378 units (representing kilometers) and target points spaced hundreds of units apart, the minimum of g is located in under one minute with a meshgrid spacing of 1 unit. When target points are closer together and the rectangular domain is

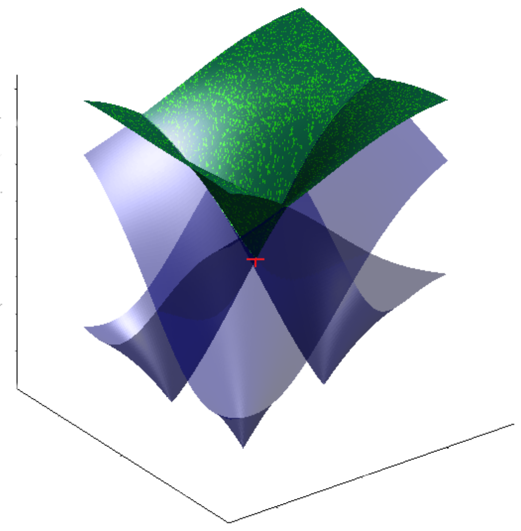


Figure 5.8: Visualizing the minimization of g.

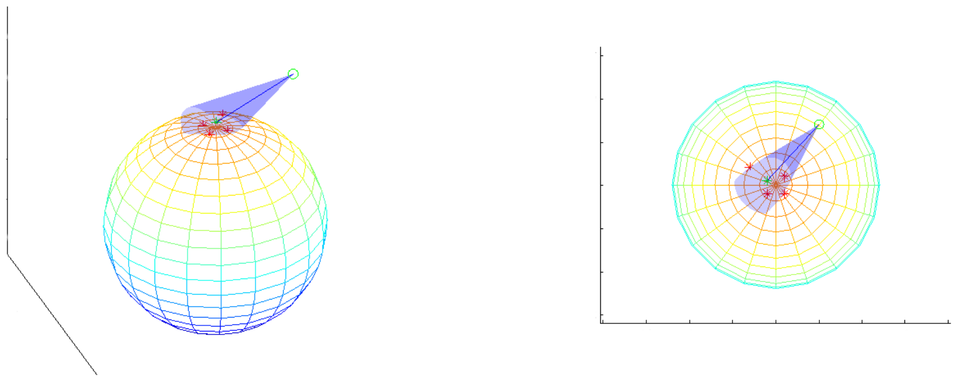


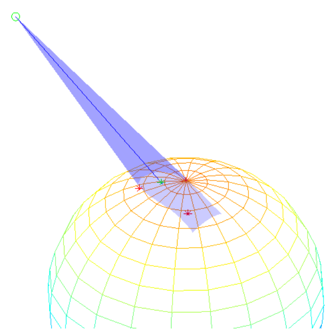
Figure 5.9: Optimal pointing to four target points.

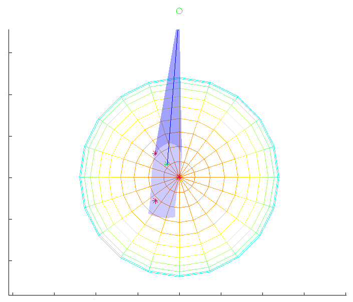
thus smaller, the calculation again only takes several seconds [3].

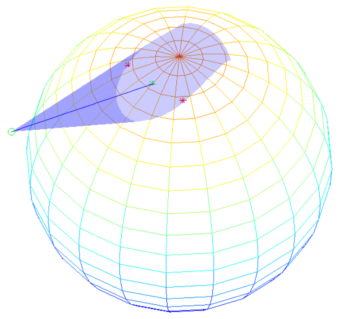
It should be noted here that a two-step implementation of this method can further reduce computations significantly. An initial rough calculation of the optimal point to reduce the domain of the meshgrid followed by a second, refined meshgrid calculation on the smaller region allows for both accuracy and efficiency. For example, setting a large grid spacing of, say, 20 units allows for an extremely quick calculation that estimates the minimum within 20 units of error. Then calculating the g values over a small domain of length 20 around this guess with a much finer meshgrid spacing will yield a very precise answer, and will not require much runtime as the domain of the meshgrid has already been dramatically reduced.

When the Matlab code for this algorithm is fully incorporated with Astrolib and can be propagated along an orbit, this solution can be used to check other possibly faster and simpler algorithms such as our iterative scheme that aim to solve the same problem.

[3]All Matlab trials mentioned were conducted on a standard Windows 7 Enterprise desktop computer with Intel Core2 Duo CPU at 3.16 GHz and 16 GB RAM.


(a) View from first satellite position.


(b) Top view from first satellite position.


(c) View from second satellite position.

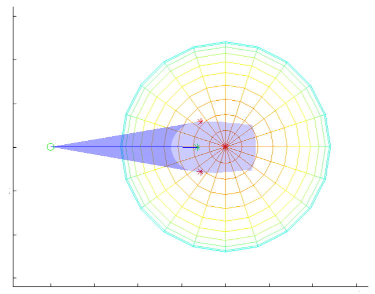

(d) Top view from second satellite position.

Figure 5.10: Optimal pointing to a set of targets from two satellite positions.

5.4.2 Optimal Pointing on the Projective Plane

We implemented all methods described in the previous section for optimal pointing on the projective plane in Matlab. The Aerospace Corporation has no current capabilities for optimal pointing [5]. This makes it difficult to develop a metric to evaluate our current methods. In this discussion of the analysis of our iterative approach we will calculate the optimal pointing directions c_M for a set of points $\mathbf{x}_i$ on the surface of the earth and a set of satellites frozen in time at different locations with respect to the $\mathbf{x}_i$. All results presented here were found using a spherical earth model with a radius of 6378 km and a visibility cone with a semi-vertex angle of $\pi/8$ or 22.5 degrees. We computed all optimal pointing locations using Matlab and then used The Aerospace Corporation's SOAP software to visualize the optimal pointing for satellite locations that approximately matched what we used in our calculations. The latitude and longitude defining our set of $\mathbf{x}_i$ is given in Table 5.1. All points are defined on the surface of the earth at an altitude of 6378 km.

Based on these points we then defined six locations for a satellite at an altitude of 10204.8 km with respect to e. These satellite locations were chosen such that each satellite had a different reference position to the $\mathbf{x}_i$. The longitude, latitude, and altitude coordinates for these six satellite locations is listed in Table 5.2.

We used SOAP as a visualization tool to display approximations of optimal pointing. The set of points $\mathbf{x}_i$ that we used with corresponding latitudes and longitudes listed in Table 5.1 are shown in green below in Figure 5.11 on the surface of the earth. Also shown is a

Point	Latitude	Longitude
1	$\pi/8$	$\pi/10$
2	$\pi/10$	$\pi/9$
3	$\pi/10$	$\pi/7$
4	$\pi/7$	$\pi/6$
5	$\pi/6$	$\pi/7$
6	$\pi/8$	$\pi/8$

Table 5.1: Target point coordinates for optimal pointing test.

Satellite	Longitude	Latitude	Altitude (km)
1	$\pi/7$	$\pi/8$	10204.8
2	$\pi/9$	$\pi/9$	10204.8
3	$\pi/5$	$\pi/10$	10204.8
4	$\pi/5$	$\pi/5$	10204.8
5	$-\pi/10$	$\pi/4$	10204.8
6	$\pi/8$	$\pi/5$	10204.8

Table 5.2: Satellite coordinates for optimal pointing test.

satellite orbit in red with a semi-major axis of 10204.8 km and an eccentricity of 0, which are the parameters we used in our visualization of our optimal pointing. This orbit has an inclination of 50 degrees and the epoch is defined at midnight on January 27, 2005.

In the first iteration, we project the $\mathbf{x}_i$ onto the projective plane defined by a viewing axis pointing from the sensor's location on the satellite to the center of the earth. We then take the convex hull of the projected points $\mathbf{x}_i$. Figure 5.12 shows the convex hull of the projected points on the projective plane of the first iteration corresponding to each satellite given in Table 5.2. Each convex hull is labeled with the longitude and latitude pair that define the satellite position.

In order to show the iterative process we will focus on the iterations for a single satellite. We will then give a table of the results for the optimal pointing location c_M on the surface of the earth for each satellite. The single satellite we focus on is satellite 4 in Table 5.2. This satellite has a latitude and longitude of $\pi/5$. This means that the location of the satellite with respect to the $\mathbf{x}_i$ is in the upper right corner. This satellite location required nine iterations before converging to the optimal pointing direction c_M. The information of interest for each of the iterations for this satellite position is given in Table 5.3. In this table "I" denotes the iteration number and "MP" denotes the minimax point, which is the $\mathbf{x}_i$ farthest from c on T. "PD" denotes the largest projection distance, which is the distance from c to the minimax point on T. Radius denotes the visibility cone radius on T. The longitude, latitude, and altitude give the spherical coordinates of c with respect to e. As seen here, for satellite 4 there are nine iterations ending in a optimal pointing location of $c_M = (.425, .434, 6399.685)$. This result requires that we then manipulate the coordinates such that the longitude and latitude stay the same, but c_M is place on the surface of the earth.

This shows a case where after the first iteration the largest projection distance on T is greater than the visibility cone radius on T. However, after the fourth iteration the

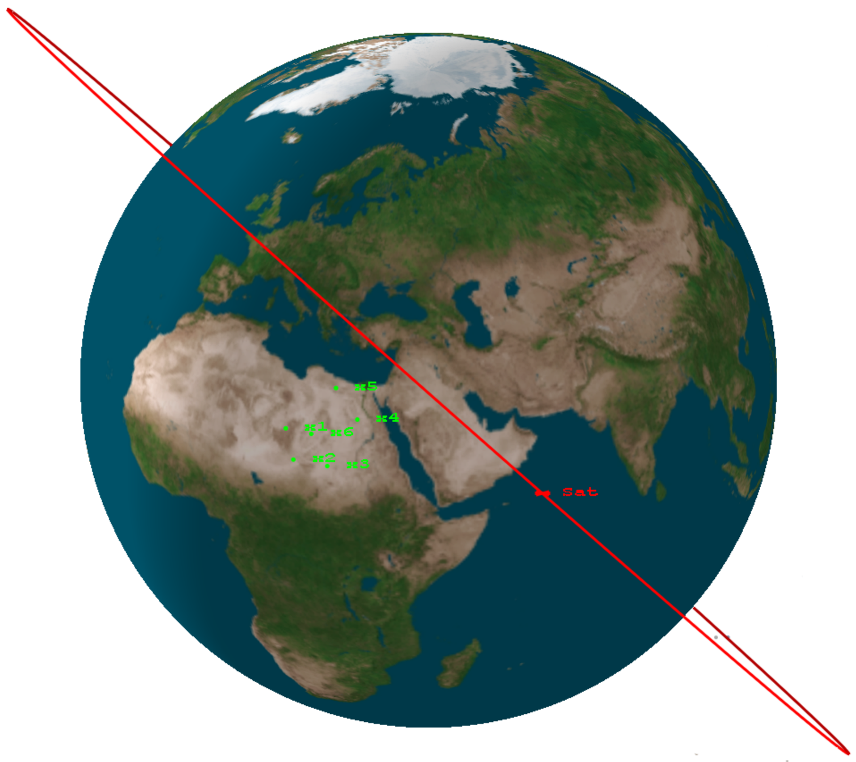


Figure 5.11: SOAP image of target points.

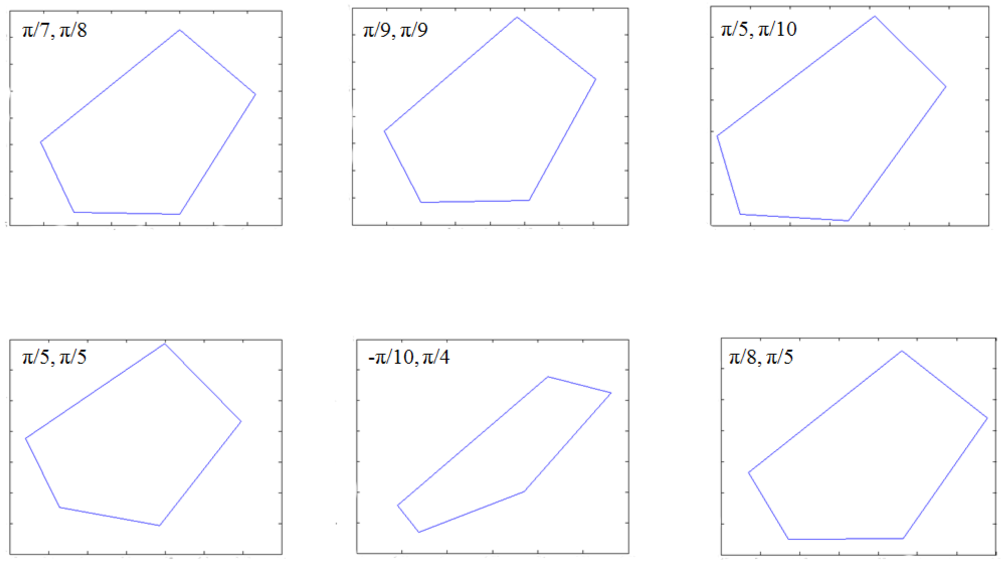


Figure 5.12: Convex hulls on projective plane.

largest projection distance on T is equal to the visibility cone radius. The largest projection distance then proceeds to decrease with each iteration and the visibility cone radius proceeds to increase. This is also a case where the point that defines the largest projection distance

I	MP	PD (km)	Radius (km)	Longitude	Latitude	Altitude (km)
1	2	1799.234	775.565	0.553	0.561	8397.275
2	2	1309.403	803.628	0.501	0.509	7463.536
3	2	1013.464	828.502	0.468	0.476	6976.256
4	2	846.361	846.361	0.447	0.456	6714.758
5	2	753.879	857.841	0.435	0.445	6570.616
6	3	716.887	864.219	0.429	0.439	6488.977
7	3	705.407	867.408	0.426	0.435	6442.418
8	3	700.304	868.684	0.425	0.434	6415.631
9	3	700.304	868.684	0.425	0.434	6399.685

Table 5.3: Iterations required for optimal pointing.

Sat	Longitude	Latitude	Altitude (km)	Opt Long	Opt Lat	I
1	$\pi/7$	$\pi/8$	10204.8	0.406	0.415	10
2	$\pi/9$	$\pi/9$	10204.8	0.398	0.411	10
3	$\pi/5$	$\pi/10$	10204.8	0.419	0.409	10
4	$\pi/5$	$\pi/5$	10204.8	0.425	0.434	9
5	$-\pi/10$	$\pi/4$	10204.8	0.365	0.462	8
6	$\pi/8$	$\pi/5$	10204.8	0.406	0.433	9

Table 5.4: Optimal pointing coordinates for six different satellite locations.

on T changes from $\mathbf{x}_2$ to $\mathbf{x}_3$ after five iterations. The trends of the iterations listed in Table 5.3 are representative of common trends in the behavior of the iterations of the vertical perspective projections for all cases.

We used the SOAP software in order to visualize the optimal pointing direction. Although we were unable to evaluate our coordinates for c_M for the exact corresponding satellite position, we were able to approximate this position by initially setting a center point at a latitude and longitude of $\pi/5$. We then set the satellite in orbit until it was nadir to this point. We then changed the center point to be the optimal pointing direction and examined the visibility behavior. Figure 5.13 shows the result of this visualization test for satellite 4. The semi-vertex angle of the visibility cone was reduced to 12 degrees to help with visualization. From our data we know that $\mathbf{x}_3$ is the point farthest from c, however we see that the points $\mathbf{x}_3$ and $\mathbf{x}_5$ define the maximum semi-vertex angle of the visibility cone that is able to enclose all $\mathbf{x}_i$ due to the position of the satellite relative to the $\mathbf{x}_i$.

In Figure 5.13 we see that when the semi-vertex angle of the visibility cone is changed such that the visibility cone just barely encloses the $\mathbf{x}_i$, the pointing direction c_M does appear to be optimal. This is because the visibility cone could not be any smaller or moved in any direction and still enclose the points $\mathbf{x}_3$ and $\mathbf{x}_5$. A table of optimal pointing locations c_M for all six satellites is shown in Table 5.4. The longitude and latitude of c_M is denoted by "Opt Long" and "Opt Lat", respectively. "I" gives the number of iterations necessary for each satellite position.

When we do the same approximation visualization tests with the remaining c_M we obtain similar results as that shown in Figure 5.13. To emphasize this, Figure 5.14 shows

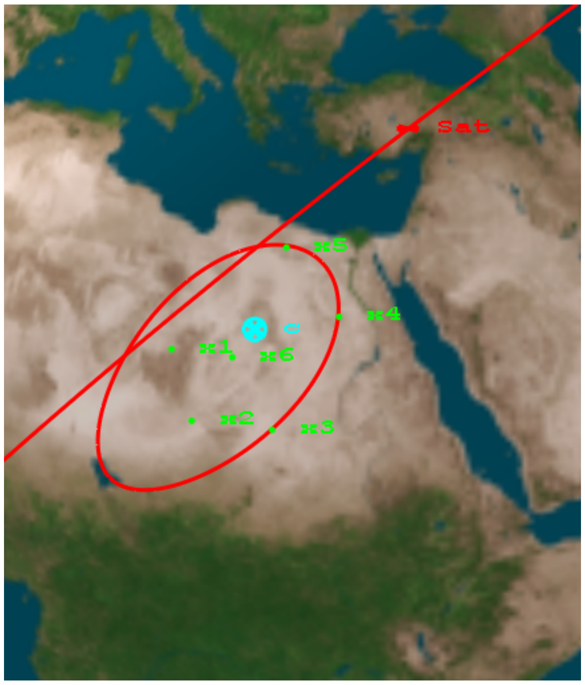


Figure 5.13: A visualization of optimal pointing.

a result of a visualization test for satellite 5 at longitude $-\pi/10$ and latitude $\pi/4$. In this case $c_M = (0.365, 0.462, 6399.685)$. The semi-vertex angle is reduced to 6 degrees for visualization purposes. As can be seen, the visibility cone footprint on the surface of the earth just encloses $\mathbf{x}_2$, $\mathbf{x}_4$, and $\mathbf{x}_5$ meaning that if the center beam vector was pointing in a different direction then simultaneous visibility would be lost for a visibility cone with a semi-vertex angle of 6 degrees.

We chose to show this example because this location for the satellite is the furthest away from the set of $\mathbf{x}_i$ when compared with the other satellite locations, making the optimal pointing direction difficult to predict and visualize. Because this visualization is an approximation we cannot be certain that we have found the exact optimal pointing direction with this iterative method, but the results are promising.

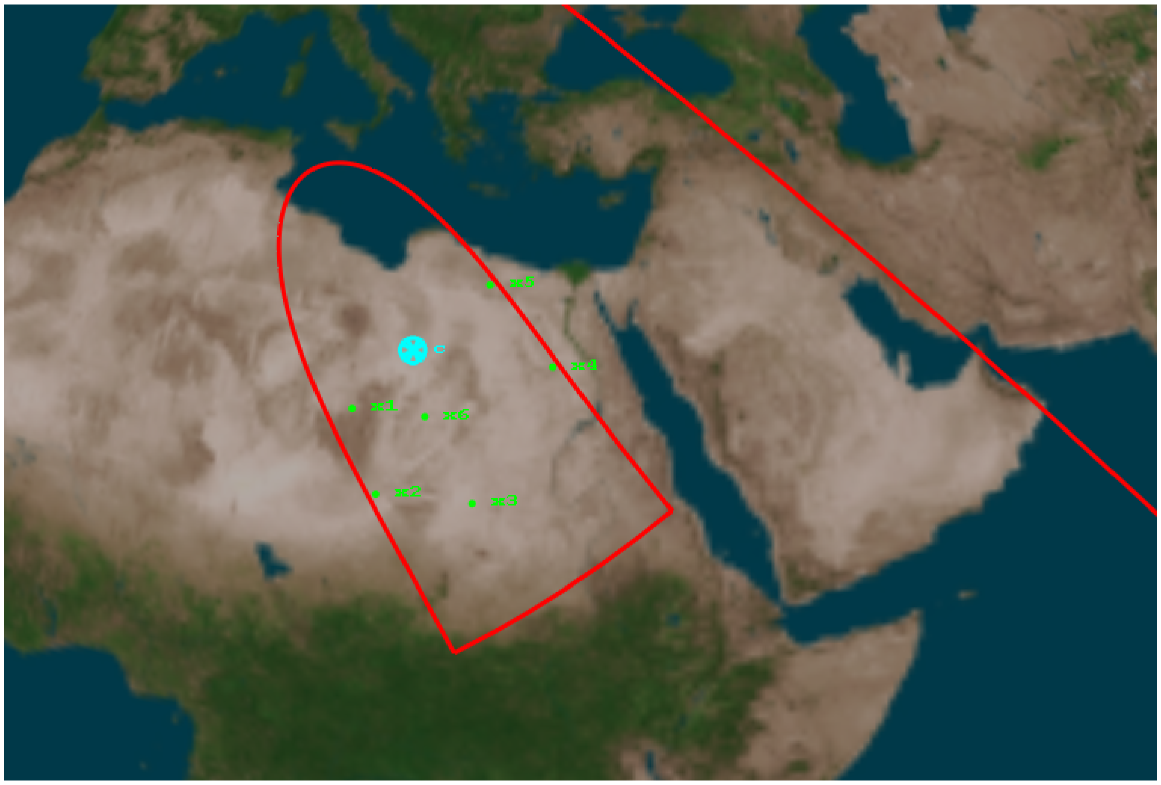


Figure 5.14: An additional visualization of optimal pointing results.

Chapter 6

Conclusion

The initial objective of the 2012 RIPS Aerospace Team was to investigate three regional coverage problems posed by The Aerospace Corporation: satellite to part of a region visibility, steerable sensor visibility, and region to satellite coverage with an oblate earth model. After this initial investigation, our team was to focus on at least one problem, delivering a solution in the forms of algorithms and software implementation.

Over the course of this research project, the RIPS team chose to focus efforts on the first two of these problems, and has developed algorithms and software for both. For the satellite to part of a region visibility problem, we have created a visibility function which is negative precisely during visibility periods, and have developed a corresponding root-finding scheme to locate these intervals. This algorithm is detailed in Chapter 4 and has been fully implemented in Matlab and tested over a range of orbits using calls to Astrolib. Furthermore, visibility results were compared with data generated by SOAP, and only minor discrepancies were found in most cases. Additionally, algorithms as well as partial solutions are provided for generalizing the method to nadir-pointing elliptical visibility cones.

Toward the steerable sensor problem, the RIPS team provides two solutions to the crucial subproblem of optimal pointing in Chapter 5. An iterative scheme applying the smallest circle algorithm on a sequence of projective planes is detailed and implemented in Matlab, offering improvement over current fixed pointing capabilities. Because of potential convergence issues and lack of a truth standard to evaluate the accuracy of this algorithm, we develop a second, mathematically precise solution to locate the optimal pointing direction. We provide a first stage implementation of this non-iterative algorithm in Matlab that does not yet incorporate Astrolib or SOAP software. Our software implementation of the algorithms in Matlab should be considered unwarranted prototypes, and all software, test data, and corresponding documentation will be provided to The Aerospace Corporation.

6.0.3 Recommendations for Future Work

The RIPS Aerospace team finds the visibility function and root-finding approach to the first problem to be effective, as most results were within seconds of results generated by SOAP. At times, discrepancies of several minutes arise for certain orbit types, and it is not clear when these problematic inconsistencies occur. We recommend further investigation and testing over a wide range of orbit types to fully validate our approach and ensure robustness of the algorithm. The generality of the visibility function allows this same method to be applied to more complicated generalizations, such as non-nadir pointing models. Future work on these generalizations can thus be based in the same overall scheme. Additionally, we recommend a more intelligent heuristic interval chop to speed up the root-finding algorithm.

Regarding the steerable sensor problem, we recommend fully incorporating the non-iterative algorithm with Astrolib so that it can be propagated along an orbit. The algorithm can then be used to verify the accuracy of the iterative approach. Efforts can also be made to reduce computations required to calculate the solution by using more sophisticated minimization methods. Lastly, both solutions to the optimal pointing problem can be implemented within a larger root-finding scheme to solve for visibility periods by constructing a visibility function similar to our approach to the first problem.

Appendix A

Appendix

A.1 Algorithms and Calculations for Satellite to Region Problem

Here, we briefly explain our key algorithms and computations in terms of their functionality. We first display the algorithms then detail five key computations.

Algorithm 1 is used to calculate the visibility function at a given time by computing the distance function (Algorithm 2), calculating the footprint radius (Algorithm 5), and by determining if $\mathbf{g}$ is in the given region (Algorithm 4). Algorithm 2 calculates the distance from $\mathbf{g}$ to each boundary. To calculate the distance from $\mathbf{g}$ to a given boundary, Algorithm 3 checks if $\mathbf{g}$ is either closest to the interior of the boundary or to one of the boundary endpoints. If $\mathbf{g}$ is closest to one of the boundary endpoints, then the distance to the nearest endpoint is computed using `P2PDIST`. If $\mathbf{g}$ is closest to the interior of the boundary, then the distance is the perpendicular distance to the boundary. This is calculated using `P2SEGDIST`. Both Algorithm 4 and Algorithm 3 use the computation `VERTANGLE`, which computes the surface angles for a given vertex. Algorithm 5 uses `TANGANGLE` to determine if the visibility cone is larger than the earth and `SURFACERADIUS` calculates the radius of the footprint.

A.1.1 Algorithms

Figure A.1 shows examples of the goundtrack being closest to one of the boundary endpoints versus the groundtrack being closest to the interior of the boundary. The green dot represents the position of the groundtrack and the black line represents the boundary. The angles α and β are the angles between the boundary and the groundtrack. In Figure

Algorithm 1 Calculate $V(t)$ for a given t.

```
dist ← distance from $\mathbf{g}$ to region (Algorithm 2)
rad ← radius of footprint (Algorithm 5)
**if** $\mathbf{g} \in$ the region (Algorithm 4) **then**
    vis ← −dMin − rad
**else**
    vis ← dMin − rad
**end if**
**return** vis
```

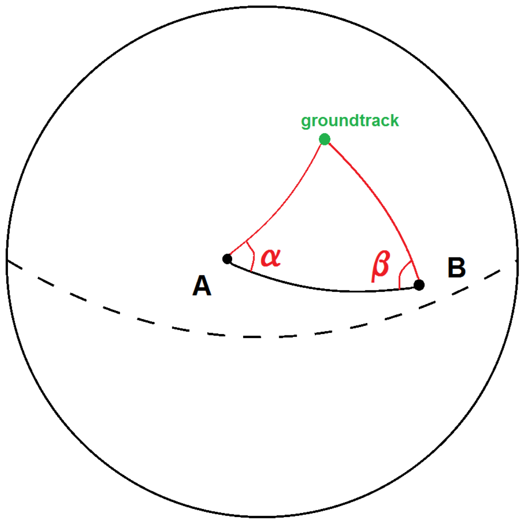
Algorithm 2 Calculate $d(t)$, the distance from $\mathbf{g}$ to the region at time t.

```
for all $i \in \{1, \ldots, N\}$ do
    if $\mathbf{g}$ is closest to the interior of the $i$th boundary (Algorithm 3) then
        dTemp $\leftarrow$ distance from $\mathbf{g}$ to interior of $i$-th boundary P2SEG($\mathbf{b}_i, \mathbf{b}_{i+1}$)
    else if $\mathbf{g}$ is closest to boundary end point $i$ (Algorithm 3) then
        dTemp $\leftarrow$ P2PDIST($\mathbf{g}$, $\mathbf{b}_i$)
    else
        dTemp $\leftarrow$ P2PDIST($\mathbf{g}$, $\mathbf{b}_{i+1}$)
    end if
    if dTemp $\leq$ dMin then
        dMin $\leftarrow$ dTemp
    end if
end for
return dMin
```

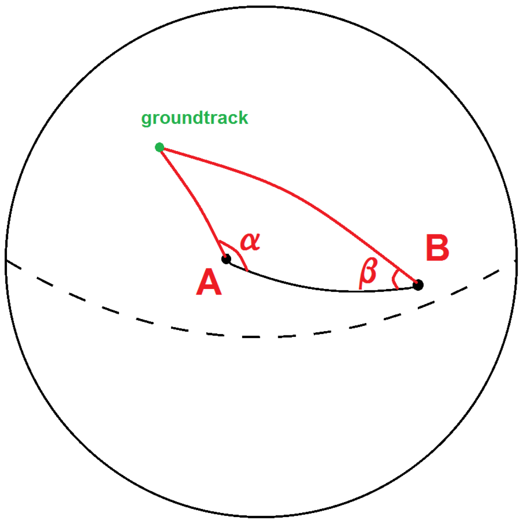
Algorithm 3 Test of $\mathbf{g}$ is closest to the interior of the boundary connecting $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$.

```
if VERTANGLE($\mathbf{g}, \mathbf{b}_i, \mathbf{b}_{i+1}$) $> 90°$ then
    return $\mathbf{g}$ is closest to boundary end point $\mathbf{b}_i$
else if VERTANGLE($\mathbf{g}, \mathbf{b}_{i+1}, \mathbf{b}_i$) $> 90°$ then
    return $\mathbf{g}$ is closest to boundary end point $\mathbf{b}_{i+1}$
else
    return $\mathbf{g}$ is closest to the interior of boundary $B_i$
end if
```

A.1(b), the angle β is less than $90°$ but angle α is greater than $90°$. This means that $\mathbf{g}$ is closest to the endpoint.

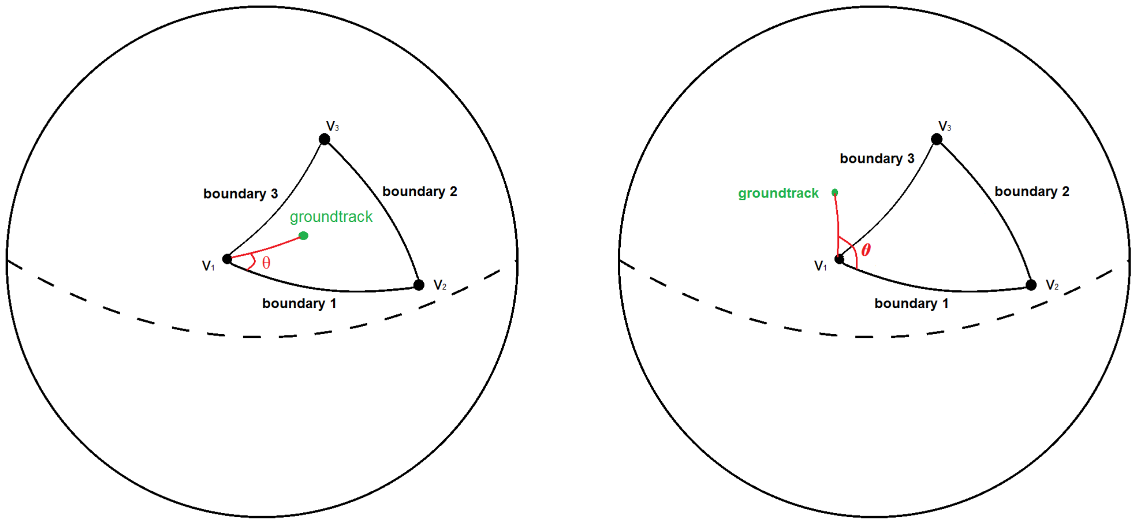


(a) α amd $\beta < 90°$.



(b) $\beta < 90^o$, but $\alpha > 90°$.

Figure A.1: The closest point along a geodesic from the groundtrack.



(a) $\theta < \angle \mathbf{v}_3\mathbf{v}_1\mathbf{v}_2$. (b) $\theta > \angle \mathbf{v}_3\mathbf{v}_1\mathbf{v}_2$.

Figure A.2: Possible positions of a groundtrack relative to a target region.

The point A is the closest point along the geodesic connecting A and B. In Figure A.1(a), both of the angles β and α are less than 90° meaning $\mathbf{g}$ is closest to the interior of the boundary segment. This is because neither of the endpoints of the geodesic segment are the closest points along the geodesic with respect to the groundtrack.

Figure A.2 shows how Algorithm 4 determines if the groundtrack position is inside of the convex hull defined by $\{\mathbf{v}_i\}$. The groundtrack is the green dot and the convex hull is shown in black. In Figure A.2(a), the groundtrack is inside the region. Then angle between the groundtrack and the first boundary, θ is less than the first vertex angle, ϕ. In Figure A.2(b), the groundtrack is outside of the region. The angle between the groundtrack and the first boundary, θ is larger than the first vertex angle, ϕ. In the actual algorithm each vertex angle is checked.

A.1.2 Calculations

P2PDIST Computes the surface distance between $\mathbf{v}_1$ and $\mathbf{v}_2$ on the earth. Convert $\mathbf{v}_1, \mathbf{v}_2$ to spherical coordinates and then use the Haversine formula to calculate the surface

Algorithm 4 Test if $\mathbf{g}$ is contained in convex hull defined by $\{\mathbf{v}_i\}$.

```
in ← TRUE
for all $i \in \{1, \ldots, N\}$ do
    $\theta \leftarrow$ angle between $\mathbf{g}$ and the $i$-th boundary VERTANGLE($\mathbf{g}, \mathbf{b}_i, \mathbf{b}_{i+1}$)
    $\phi \leftarrow$ $i$-th vertex angle VERTANGLE($\mathbf{b}_{i-1}, \mathbf{b}_i, \mathbf{b}_{i+1}$)
    if $\theta > \phi$ then
        in ← FALSE
    end if
end for
return in
```

Algorithm 5 Calculate the radius of the footprint $r(t)$ when the satellite is an altitude h and the semi-vertex angle of the visibility cone is β.

```
$\beta_{tan}$ = TANGANGLE($h$)
if $\beta_{tan} > \beta$ then
    $\theta$ = SURFACERADIUS($\beta_{\text{tan}}$)
else
    $\theta$ = SURFACERADIUS($\beta$)
end if
radius $\leftarrow R_{earth} \cdot \theta$
return radius
```

distance:

$$\mathbf{v}_1 = \begin{pmatrix} \theta_1 \\ \phi_1 \\ R_{earth} \end{pmatrix},$$

$$\mathbf{v}_2 = \begin{pmatrix} \theta_2 \\ \phi_2 \\ R_{earth} \end{pmatrix},$$

$$d = 2R_{earth} \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_1 - \phi_2}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\theta_1 - \theta_2}{2}\right)}\right).$$

P2SEGDIST Calculates the distance between $\mathbf{g}$ and the geodesic passing through $\mathbf{v}_1$ and $\mathbf{v}_2$. The distance is measured along the geodesic perpendicular to the original geodesic.

$$v = \mathbf{v}_1 \times \mathbf{v}_2,$$

$$d = R_{earth} \left| \frac{\pi}{2} - \arccos\left(\frac{\mathbf{g} \cdot \mathbf{v}}{\|\mathbf{g}\| \|\mathbf{v}\|}\right) \right|.$$

Figure A.3 shows the distance between the groundtrack and the interior of the geodesic segment connecting $\mathbf{v}_1$ and $\mathbf{v}_2$.

VERTANGLE Computes surface angle A created by vectors $\mathbf{y}, \mathbf{x},$ and $\mathbf{z}$ shown in Figure A.4 by using the spherical law of cosines.

$$a = \arccos\left(\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}\right),$$

$$b = \arccos\left(\frac{\mathbf{x} \cdot \mathbf{z}}{\|\mathbf{x}\| \|\mathbf{z}\|}\right),$$

$$c = \arccos\left(\frac{\mathbf{y} \cdot \mathbf{z}}{\|\mathbf{y}\| \|\mathbf{z}\|}\right),$$

$$A = \arccos\left(\frac{\cos c - \cos a \cos b}{\sin a \sin b}\right).$$

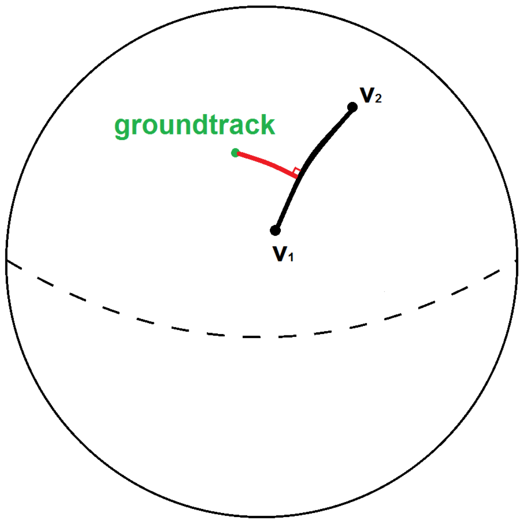


Figure A.3: Surface distance between groundtrack and a geodesic curve.

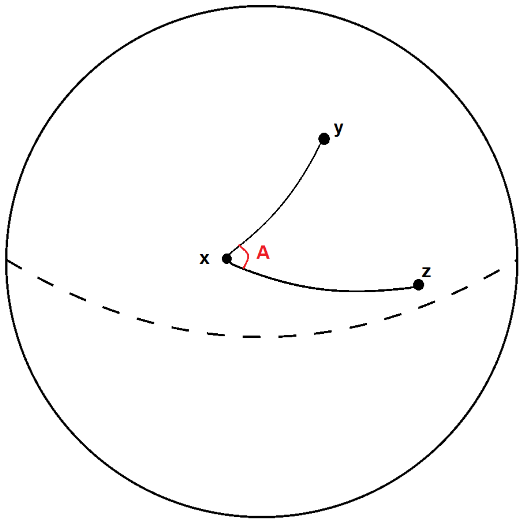


Figure A.4: A vertex angle on the surface of the earth.

SURFACERADIUS Calculates the central angle θ of the surface radius of the footprint made by a nadir pointing circular visibility cone with angle β and altitude h.

$$b = \sin\beta\cos\beta\cdot(R_{earth}+h) - \sin\beta\sqrt{R_{earth}^2 - (R_{earth}+h)^2\sin^2\beta},$$

$$\theta = \arcsin\left(\frac{b}{R_{earth}}\right).$$

Figure A.5 shows a cross section of the green visibility cone with semi-vertex angle β intersecting the earth. The purple arc represents the footprint radius. To calculate the footprint radius calculate distance b, denoted by the blue line, then calculate the central angle θ.

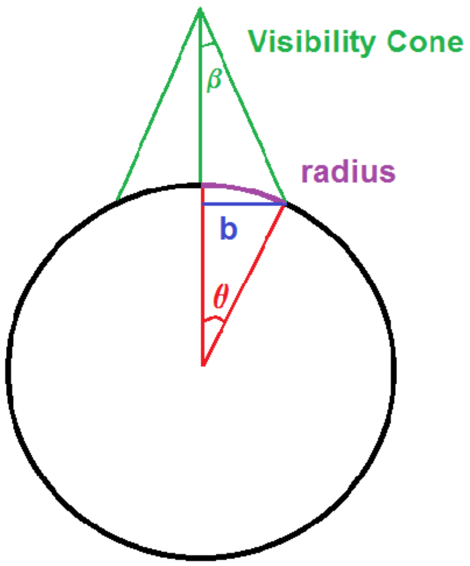


Figure A.5: A side view of a circular visibility cone intersecting the earth.

TANGANGLE Computes the angle semi vertex angle β_{tan} necessary so that a circular visibility cone with altitude h is tangent to the earth.

$$\beta_{tan} = \arcsin\left(\frac{R_{earth}}{R_{earth} + h}\right).$$

Appendix B

Glossary

Ascending node. The point where the satellite crosses through the equatorial plane in a northerly direction.

Earth-centered inertial frame. A frame of reference whose origin is the center of the earth and which does not rotate with respect to inertial space.

Earth-centered rotating frame. A frame of reference whose origin is the center of the earth but which rotates with the earth.

Footprint. The intersection of a visibility cone with the surface of the earth.

Great circle of arc. The shortest path between two points on the surface of the earth.

Groundtrack.The location of the center of a visibility cone footprint on the surface of the earth.

Inclination. The angle between the normal to the orbit plane and the normal to the equatorial plane.

LEO. An orbit with an altitude approximately below 2,000 km.

Molniya orbit. A highly elliptical orbit with an orbital period of half a day.

Projection distance. The distance between the center of the visibility cone footprint and a point of interest projected onto the plane orthogonal to the vector defining the visibility cone center and tangent to the earth surface.

Right ascension of the ascending node. The angle between the unit vector **X** and the point where the satellite crosses the ascending node, measured counterclockwise when viewed from the north side of the equatorial plane.

Appendix C

Abbreviations

ECI.	Earth Centered Inertial frame
ECR.	Earth Centered Rotating frame
LEO.	Low Earth Orbit
RAAN.	Right Ascension of the Ascending Node
SOAP.	Satellite Orbit Analysis Program

Selected Bibliography Including Cited Works

[1] Roger R. Bate, Donald D. Mueller, and Jeremy E. While. *Fundamentals of Astrodynamics*. Dover, 1971.

A standard textbook on astrodynamics. It provided a reference for orbital mechanics and satellite propagation.

[2] Ingrid Carlbom and Joseph Paciorek. Planar Geometric Projections and Viewing Transformations. *Computing Surveys*, 1978.

Gives a thorough background to projective geometry and vertical perspective projection. This includes details about calculating projections using homogeneous coordinates and projection matrices.

[3] John M. Coggi and David Y. Stodden. *Satellite Orbit Analysis Program*. The Aerospace Corporation, May 2012.

Documents important visualization capabilities for regional coverage problems available through the Soap Orbit Analysis Program (SOAP) used at The Aerospace Corporation.

[4] Gelfand and Fomin. *Calculus of Variations*. Prentice-Hall, 1963.

Discusses the essential principle of variational method for optimal path problems.

[5] Gary Green. Region Visibility Theory. *The Aerospace Corporation*, 2012.

Gives an overview of regional coverage analysis. The author outlines the current mathematical approach that The Aerospace Corporation uses to model regional coverage. In addition, the author introduces problems of interest in the field for which The Aerospace Corporation has no current capabilities.

[6] Elizabeth A. Howard. *Astrolib C User's Guide*. Air Force Space Command, 3.3 edition, February 2011.

Details the main functions of Astrolib. Astrolib is a software library used by The Aerospace Corporation for regional coverage analysis. This manual allowed us to incorporate our code with the Astrolib functions made available to our group.

[7] Jacob Kogan. *Introduction to Clustering Large and High-Dimensional Data*. Cambridge, 2007.

Focuses on a few of the most important clustering algorithms, providing also some useful optimization techniques for high-dimensional objective functions.

[8] David A. Vallado. *Fundamentals of Astrodynamics and Applications.* Space Technology, 2007.

A professional astrodynamics reference. It emphasizes the practical use of astrodynamics in space missions.

[9] Emo Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). *New Results and New Trends in Computer Science*, 1991.

Outlines a smallest circle algorithm that runs in linear time using recursion.