ABSTRACT

BJERKAAS, JAMES KEVIN. Modeling Ground Sensor Acquisitions of Low Earth Orbit Objects. (Under the direction of Professor Thom J. Hodgson).

The United States Strategic Command (STRATCOM) utilizes a sensor network to accomplish several of its missions. These missions include missile defense, missile warning, intelligence collection, and space surveillance. For the task of space surveillance, the locations of all man-made satellites, as well as debris formed by the collisions of these satellites, is of great interest to the United States.

Previous work has focused on assigning the various sensors in the sensor network to the separate tasks required. This thesis focuses on developing a simulation to learn more about the dynamic interactions between the sensors and the constantly moving orbiting field of satellites and debris.

The stochastic model developed shows over time what happens to the knowledge of the objects in Low Earth Orbit (LEO) when the sensors assigned to tracking their progress are changed. When sensors are reassigned from space surveillance to another task, the simulation exhibits a transient period where the state of the system adjusts to the new sensor coverage. Similarly, when a sensor is added to the task of space surveillance, a transient period occurs while the system adjusts to the new sensor.

This transient information can be used by decision makers as a tool in scheduling sensors to these various tasks, and can also be used to help refine heuristic methods developed previously in addressing this scheduling problem.

Modeling Ground Sensor Acquisitions of Low Earth Orbit Objects

by
James Kevin Bjerkaas

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Operations Research

Raleigh, North Carolina

2010

APPROVED BY:

_____	_____
Thom J. Hodgson Committee Chair	Russell E. King
_____	
James R. Wilson	

BIOGRAPHY

James Kevin Bjerkaas was born in Urbana, Illinois in 1970. He attended University of Maryland at College Park and graduated in May 1993 with a Bachelor of Science in Mathematics and a Bachelor of Science in Mathematics Education. After Graduation, he began Basic Combat Training at Fort Leonard Wood, Missouri, where he began his training as a Personnel Administration Assistant in the United States Army Reserve. After serving four years in the United States Army Reserves, he was commissioned as a Second Lieutenant in the United States Army in 1997 through Officer Candidate School (OCS) at Fort Benning, Georgia. He was branched Engineers and returned to Fort Leonard Wood, Missouri to begin his military officer education as a combat engineer.

His military assignments include serving as a Combat Engineer Battalion Intelligence Officer at Fort Hood Texas and deployed to Bosnia-Herzegovina, a Combat Engineer Platoon Leader at Fort Hood, Texas, an Executive Officer for a Combat Engineer Company in South Korea, as well as a Combat Engineer Battalion Assistant Operations Officer, a Staff Engineer Officer for the Coalition Joint Civil-Military Operations Task Force in Kabul, Afghanistan, a Budget Officer for a Combat Engineer Brigade, an Assistant Battalion Operations Officer, and a Combat Engineer Company Commander at Fort Lewis, Washington, a Combat Engineer Brigade Battle Captain in Tikrit, Iraq, and a Operations Research Analyst for the TRADOC Analysis Center at White Sands Missile Range, New Mexico. He has also earned a Master of Science Degree in Engineering Management from the University of Missouri at

Rolla and a Master of Science Degree in Industrial Engineering from New Mexico State University in Las Cruces, New Mexico.

James currently holds the rank of Major in the United States Army, and upon graduation, will serve as a mathematics instructor at the United States Military Academy at West Point, New York.

Jim and his wife Julie will celebrate their fifth wedding anniversary this June.

ACKNOWLEDGEMENTS

I would like to recognize and express my appreciation to the following people for their help and support to me while I worked on this thesis.

My wife Julie

My parents and brothers

Derek Tharaldson

Rajneesh

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Background

Two recent events have demonstrated the importance of tracking debris in orbit around the Earth. In January 2007 China conducted an anti-satellite test (Fengyn-1C), and in February 2009 two satellites, Iridium 33 and Cosmos 2251, collided 490 miles (790 kilometers) above the surface of the Earth. These two events created approximately 5000 objects of debris over 10 centimeters in diameter in Low Earth Orbit (LEO), increasing the number of objects tracked by the National Aeronautics and Space Administration (NASA) Space Debris program by about 50%. (Liou, 2010) This increase in debris makes it increasingly likely that man made satellites will suffer collisions during their orbit around the Earth. In addition, these objects pose risks to manned vehicles in orbit, including the International Space Station.

The United States Strategic Command (STRATCOM) is one of the unified commands under the Department of Defense. One of its missions is "to provide integrated surveillance and reconnaissance allocation recommendations to the Secretary of Defense." (U.S. Strategic Command Public Affairs Office, 2009) To accomplish this mission, STRATCOM considers the network of sensors available to the Department of Defense, including some of the sensors used by the NASA Space Debris program. This network is utilized to carry out several distinct tasks for the Department of Defense. One of these tasks is to maintain space situational awareness through space surveillance. This mission involves continuously

monitoring and collecting information on all man-made objects in orbit around Earth. From this information they produce a satellite catalog, "used by predictive orbital analysis tools to anticipate satellite threats and mission opportunities for friendly, adversary, and third party-assets." (3-14, 2009)

Previous work has been conducted in allocating sensors to this task, as well as other tasks such as missile defense, missile warning, and intelligence collection. Dulin developed a heuristic method for determining an optimal allocation of these sensors to their tasks. (Dulin, 2009) In his research, he treated space surveillance as a secondary task, and considered only the probability of success in these tasks. The interaction between sensors and objects in orbit was considered static, while in reality it is a dynamic system. Dulin identified the need to model this task of space surveillance as a dynamic system to achieve a greater level of precision in the overall model. Subsequent work will update Dulin's heuristic with a dynamic representation of this task of Space Surveillance. In support of this ongoing research, this thesis provides a simulation for modeling this task of Space Surveillance with greater fidelity and providing insight into the behavior of this dynamic system when sensors are removed or added to the network.

1.2 Problem

For this thesis, a simulation was developed to model man-made satellites in orbit and the sensors used to track those satellites by the United States. With estimates of over 15,000 distinct objects of size 10 cm and larger in LEO, the task of tracking these pieces, which

range in size from small bolts to large man-made satellites, is non-trivial. Of particular interest is an understanding of what happens to the knowledge of these objects when one or more sensors are reassigned from the task of monitoring the objects to another higher priority task. The questions of interest are: 1) What is the steady state of the system when all sensors are focused on tracking the objects in orbit, 2) What happens to this steady state when one or more sensors are removed. In addition, 2a) what is the new steady state, and 2b) How long does it take to reach this new state? The steady state of the system is defined as the long run average values of what is known about the objects in orbit. For example, with three sensors working to track objects in orbit, how many objects, on average, do they have data on from recent detections.

The simulation represents the movement of these objects in orbit in and out of the acquisition range of the sensors; the acquisition of these objects by the sensors when they are in range of the sensors; and the loss of individual sensors and the effect on the overall system.

1.3 Assumptions and Limitations

1.3.1.1 Perturbations to Orbits due to Non-Spherical Shape of the Earth

For the purposes of this model, perturbations to the orbits of the objects in LEO are ignored. These perturbations include changes in each successive orbit due to the non-spherical shape of the Earth and from the gravity of other physical objects in space other than the Earth. As an object orbits around the Earth, the shape of the Earth tends to pull the orbit westward, causing the path of the orbit to change at each rotation. In addition, the point of perigee,

where the object is closest to the Earth along its orbit, changes due to the shape of the Earth. For this simulation, the exact positions of objects are not required. The emphasis of the simulation is on the ability to track objects in LEO, not the exact locations of these objects. Ideally, the positions should be simulated as accurately as possible, however, because of run time considerations these perturbations are ignored. This topic is covered in more depth in Chapter 5 Future Research.

1.3.1.2 Perturbations to Orbits due to other Factors

Other perturbations to the orbits around the Earth are also ignored. These include the gravitational forces of other objects, such as the Moon and the Sun, and atmospheric conditions such as atmospheric drag and solar winds. These perturbations have less of an impact on the orbits than the perturbations due to the shape of the Earth. (Capderou, 2005)

1.3.1.3 Galilean Frame of Reference

The simulation assumes a Galilean frame of reference for the orbits in question. This means that the frame of reference is fixed with the origin at the Earth's center, the Z axis oriented along the line running from the North to South Pole, and the plane formed by the X and Y axes lies along the equatorial plane of the Earth, with the X axis oriented towards the Vernal Equinox.

1.3.1.4 Constant Velocity

The simulation assumes constant velocity for the orbits because the majority of the orbits are near-circular. This assumption also greatly simplifies computational complexity and

shortens run-time. For the sample set of 4880 object in orbit used for this thesis, 4803 have eccentricities less than 0.1, where an eccentricity of 0 represents a circular orbit.

1.3.1.5 Sensors Modeled

The sensors modeled are assumed to have similar characteristics to the Phased Array radar. Exact operating data was not available, so approximate ranges and failure rates were parameterized. In addition, the Space Fence was not modeled. The Space Fence is a set of transmitters and receivers along 33 degrees North latitude which detect any objects passing over the United States.

1.3.1.6 Sensors not Modeled

Long range sensors, such as optical telescopes, were not modeled. The focus of the model was on sensors focused on LEO, which are defined for purposes of this simulation as any orbit with an altitude of less than 2000 km from the surface of the Earth. In addition, passive receivers, which track transmissions from functioning man-made satellites, are not modeled.

1.3.1.7 Time Dependency of Object Lists

The behavior of the knowledge of objects in LEO over time is managed by lists of Known, Update, and Unknown objects. The simulation assumes that objects move from one list to another as a function of time. For example, if an object has not been detected for at least n hours, it will move from the Known list to the Update list. While other events may trigger movement of objects from one list to another in reality, such as a collision between objects in

orbit, or a maneuverable satellite changing course, this simulation deals with only changes due to elapsed time.

1.3.1.8 Classification of Parameters

In addition, some information, such as the reliability of the radars, and the schedules for which sensors are used when to track objects in orbit, was unavailable. Some of this information is sensitive, and not releasable to the public, while other information changes over time. For purposes of this thesis, parameters are used in place of this actual information. In particular, for the probability of detection equations, the actual probabilities of detection for a given phased array radar are estimated using the equation explained in Chapter 3. The purpose of these parameters is to allow the simulation to function as designed, but also allow for changes if actual data becomes available.

1.4 Overview

Chapter One includes background material, a statement of the problem, and a list of the main assumptions and limitations of this work. Chapter Two describes previous research in this area. Chapter Three describes the methodology used to address the problem, mainly explaining the model used to describe the behavior of the objects in orbit and the sensors tasked to track these objects. Chapter Four describes the preliminary experimentation conducted with the model, and describes the methods used to verify and validate the model. Chapter Five discusses areas for future research. These chapters are followed by references and appendices, including the code for the model and some examples of model output.

2 LITERATURE REVIEW

2.1 Current Models

There are currently two main types of models used by NASA to help in understanding space debris, engineering models and evolutionary models. (Stansbery, 2009) In addition, the North American Aerospace Defense Command (NORAD), uses several models, including SPG4, to predict positions of objects in orbit. (Hoots & Roehrich, 1980)

Engineering models, such as ORDEM2000 (Orbital Debris Engineering Model), are used to model the orbital debris environment in detail. This model is useful to spacecraft designers and mission planners in determining what type of protection the spacecraft will require. It is also useful to those interesting in sensing orbital debris in determining the best strategy for detecting debris. ORDEM2000 uses altitude, latitude and debris size to model the debris environment. (Liou, Matney, Anz-Meador, Kessler, Jansen, & Theall, 2002) ORDEM2000 does not model the behavior of sensors, however, and provides more detail than necessary for the problem.

Evolutionary models, such as LEGEND (LEO-to-GEO Environment Debris), model the long term behavior of space debris. This model, and other similar models, can help predict the future environment and assess the impact of satellite collisions and policy changes on the debris environment. These models take a longer view than would be of interest for this problem, as time horizons of 100 years or more are common. (Johnson, 2004)

NORAD uses a series of models to predict the locations of objects in orbit around the Earth. These models include SGP4 (Simplified General Perturbations), developed in 1970, for predicting positions of near Earth satellites, SDP4 (Simplified Deep-Space Perturbations), developed in 1979 for predicting deep space orbits, and SGP8, developed in 1980 which also predicts orbits near Earth. All of these models use the standard NORAD two-line data set, and take into account perturbations to individual orbits. These models are highly detailed, and take into account the methods by which the two line elements are generated by NORAD. (Hoots & Roehrich, 1980)

For the current problem, a model is needed that will simulate both the behavior of objects in orbit and the behavior of sensors detecting the objects in orbit. The models listed do not take sensors into account, but focus only on the objects in orbit. In addition, the level of detail used in the engineering models and the NORAD predictive models is not required for this problem. So a new simulation was developed to meet the specific goals of this thesis.

3 MODEL

This section describes the model used to represent the dynamic system of sensors and objects in LEO. First, the overall structure of the model is discussed, followed by an in-depth look at each of the main sub-structures involved. As each step is explained, the modeling of the orbit of the International Space Station (ISS) is presented as an example.

3.1 Overall Model Structure

The model, a stochastic simulation programmed in MATLAB, utilizes discrete time steps to analyze what happens to the knowledge of objects in space over time. This is accomplished by the following process. In Step One, the system is initialized. This involves four sub-steps. First, reference tables for the locations of the objects in LEO are determined. Second, reference tables for the locations of the sensors on the surface of the Earth are determined. Third, the lists which summarize the knowledge, or situational awareness, of the different objects in LEO are initialized, these lists are referred to as Object Lists. Lastly, time is set to 0. This completes Step One. In Step Two, time is incremented by a pre-determined step size. In Step Three, for each time increment, three actions are carried out by the model. First, the positions of the objects in orbit and the sensors on the surface of the Earth are updated using the reference tables generated in Step One. Next, calculations are performed to determine which objects would be acquired by sensors. Finally, the Object Lists are updated if necessary. In Step Four, the simulation then repeats Steps Two and Three until the designated end time is reached.

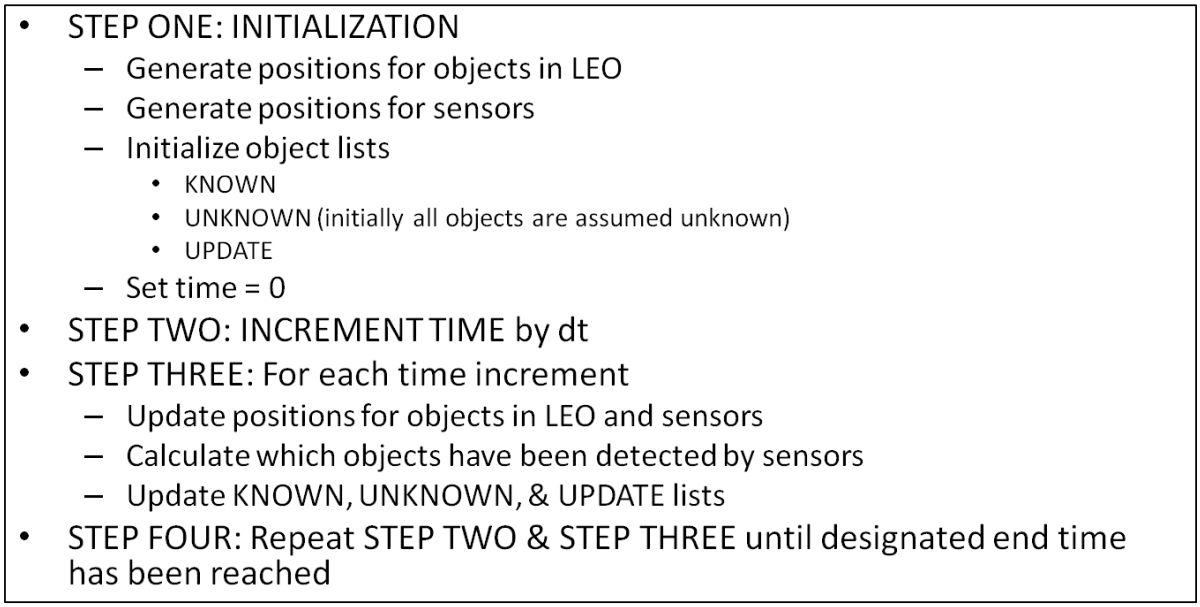

Figure 3.1. Simulation Structure

3.2 Step One: Initialization

The model requires a set of coordinates for each object indexed by time steps. As part of initialization, these arrays of XYZ coordinates and times are created. The simulation references this data when determining current positions at specific times. Because the time steps of the simulation are different than the time steps in the initial reference tables, the positions of objects are determined by interpolating between coordinates in the initial data which lie on either side of the time needed. This process is further explained within the discussion of Step Two. This section discusses how these reference tables of initial positions are created for both objects in orbit and sensors on the surface of the Earth. This section concludes with a description of the Object Lists describing the knowledge of objects in orbit over time.

3.2.1 Generate Positions for Objects in Low Earth Orbit (LEO)

Before the model can determine the location of each object and sensor for a given time, the model generates a reference table for each object in the form of an array representing one complete orbit around the Earth. This array is composed of a set of *XYZ* coordinates referenced by a time *t*. As time is incremented in the simulation, each position is determined by interpolating between the appropriate times stored in this array. This section describes the construction of this array for each object in LEO.

The positions of man-made satellites in orbit around the Earth, as well as space debris, are cataloged according to their orbital parameters. Orbital parameters are a set of six elements which can uniquely determine the position of an object with respect to the Earth. The simulation, for ease of calculation, does not store the objects using these parameters. Instead, the objects are stored by their *XYZ* coordinates over time. The simulation represents each item in orbit by converting the object's orbital parameters into a three-dimensional Cartesian coordinate system. These coordinates are then indexed by a set of time steps breaking down one complete orbit into *n* distinct intervals. Using these specific time steps, the model determines where along the orbital path an object is at any given time. The following sections describe this conversion from orbital parameters to time indexed Cartesian coordinates.

3.2.1.1 Converting Orbital Parameters into Time-stepped Cartesian Coordinates

There are six orbital parameters necessary to describe an object's position along its orbital path in space. These orbital elements describe the object of interest's Keplerian motion in space and are used by NASA in tracking the positions of satellites and debris in orbit. (Grimaldi, 1997). The six elements can be divided into three main groups: the elements which describe the object's orbital ellipse, the elements which orient this ellipse within the frame of reference, and an element which fixes the position of the object at a set time. The next section explains in detail each of the orbital elements, discussing how these elements are used within the model, and demonstrating this process through an example: the International Space Station (ISS).

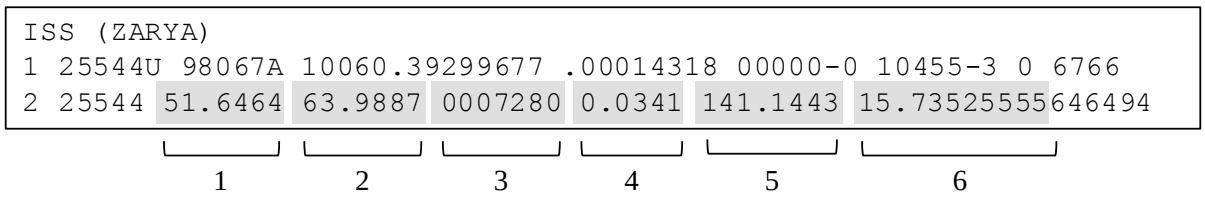

Figure 3.2. International Space Station Two-Line Elements

Figure 3.2 shows the orbital parameters for the International Space Station in the standard form of the data tracked for each known object in orbit. The shaded numbers represent the six orbital parameters, identified in Table 3.1 below.

Table 3.1. Orbital Parameters for International Space Station

Number	Element	Value for ISS
1	Inclination	51.6464°
2	Right Ascension of Ascending Node	63.9887°
3	Eccentricity	0.0007280
4	Argument of the Perigee (Periapsis)	51.6464°
5	Mean Anomaly	141.1443°
6	Revolutions per Day	15.73525555

3.2.1.2 The Characteristics of the Orbit's ellipse.

The elements which describe the characteristics of the object's orbital ellipse are the eccentricity and the length of the semi-major axis. NASA uses revolutions per day instead of the length of the semi-major axis. However, the length of the semi-major axis can be derived from the revolutions per day using the equation for the Keplerian Period. (Capderou, 2005)

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{3.1}$$

where μ represents the Earth's gravitational constant ≈ 398600.4418, T is the period of revolution (the inverse of revolutions per day), and a is one half the length of the semi-major axis of the ellipse. Once the semi-major axis and eccentricity are known, the movement of the object in the plane of its ellipse can be modeled. The semi-minor axis of the ellipse, b, can be found using the eccentricity and the semi-major axis.

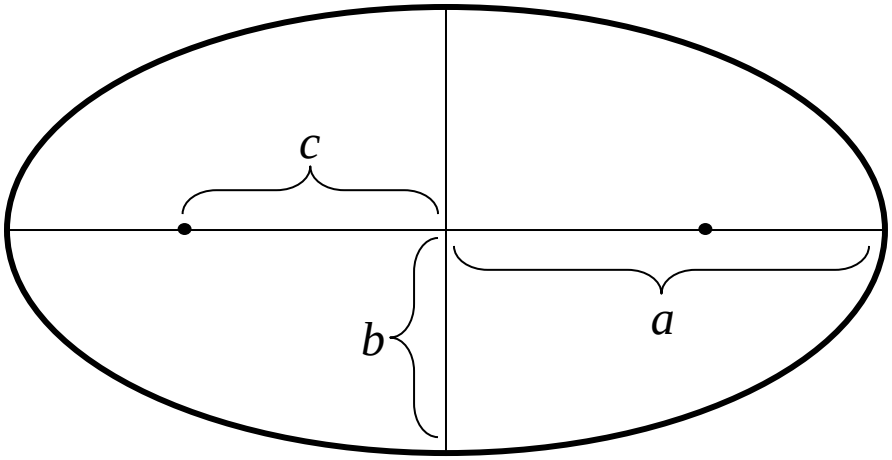

Figure 3.3. Elliptical Parameters

Then the distance from either foci to the center of the ellipse, *c*, can be found.

$$b = a\sqrt{1 - e^2} \tag{3.2}$$

$$c = \sqrt{a^2 - b^2} \tag{3.3}$$

For the ISS example, these equations yield the following values: a = 3.441994, b = 3.441993 and c = 0.002506. The semi-major and semi-minor axes are very close in length, as the orbit of the ISS is nearly circular (with eccentricity 0.000728). In terms of eccentricity a perfect circle has eccentricity 0, while an eccentricity of 1 describes a parabola, and values greater than 1 describe hyperbola. For our purposes, the orbits we are interested in are ellipses, so their eccentricities lie between 0 and 1.

Assuming constant velocity around the ellipse, an *X* and *Y* coordinate system (within the plane of the ellipse) is then derived, incrementing *t* from 0 to 360 degrees. Equations (3.4) and (3.5) demonstrate this procedure, using Cartesian coordinates in the plane of the orbit.

For the remainder of equations used throughout this section, either Cartesian or Polar coordinates are used for simplicity. In the simulation, whichever set of coordinates was simpler to manipulate was used, the equations for only the sets of coordinates used within the simulation are included in this section.

$$x(t) = a\cos(t) + c \tag{3.4}$$

$$y(t) = b\sin(t) \tag{3.5}$$

For the ISS example, incrementing *t* by 30 degrees, breaking the orbit into 12 steps, following *XY* coordinates are obtained. The dimensions for *x(t)* and *y(t)* are with respect to the center of the Earth.

Table 3.2. *XY* Coordinates for ISS within Orbital Plane

t	*x(t)*	*y(t)*
0	5830.6217	3363.4827
30	3368.3808	5825.7229
60	4.8972	6726.9654
90	-3358.5863	5825.7229
120	-5820.8272	3363.4827
150	-6722.0699	0.0000
180	-5820.8272	-3363.4827
210	-3358.5863	-5825.7229
240	4.8972	-6726.9654
270	3368.3808	-5825.7229
300	5830.6217	-3363.4827
330	6731.8644	0.0000
360	5830.6217	3363.4827

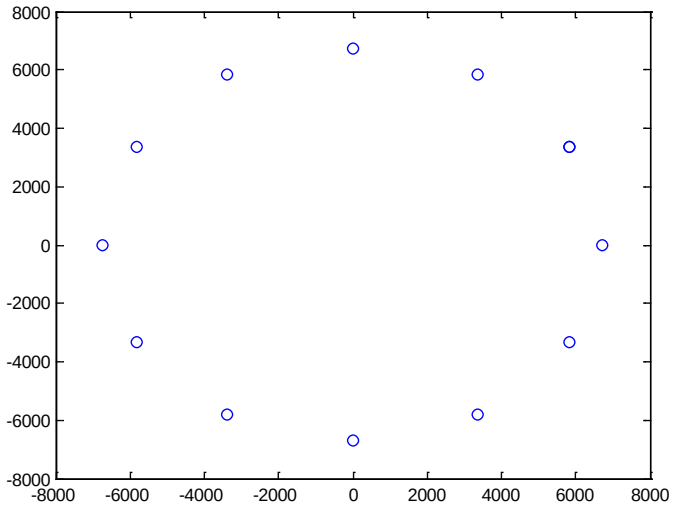

Figure 3.4. Graphical Representation of ISS Orbit within Orbital Plane

There is another orbital parameter, the Mean Anomaly, which describes the position of the object along the orbital path at a specified time. This parameter is not used due to the constant velocity assumption. The starting position of each object in orbit is determined randomly, thus eliminating the need for this parameter. In the ISS example, we would simply randomly select one of the 12 coordinate sets to start with at time 0.

3.2.1.3 The Orientation of the Orbital Plane with Respect to the Reference Plane

To orient the elliptical plane within the frame of reference, three angles are needed. They are the argument of the periapsis, the inclination, and the right ascension of the ascending node. These are sometimes referred to as Euler angles, or the angle of proper rotation, the angle of nutation, and the angle of precession, respectively. (Capderou, 2005)

3.2.1.4 The Argument of the Periapsis

The argument of the periapsis, ω, orients the ellipse within the orbital plane. The periapsis is the point of the orbit which passes closest to the body being orbited, for an object orbiting around the Earth, this is called the perigee. For an object orbiting the Sun, this is the perihelion. As a result, this orbital element is sometimes referred to as the argument of the perigee for Earth orbits. This element is an angle describing the difference between the primary axis of the orbital plane and the semi major axis of the orbit's ellipse. In Figure 3.5 the orbital plane is formed by (P_0, N, N'), where N and N' represent where in the orbital plane the path of the orbit crosses the equatorial plane of the Earth. The variable N is referred to as the Ascending Node, where N' is the Descending Node. The variable P_0 represents the direction of the perigee. So O (representing the center of the Earth) and P_0 lie along the semi-major axis of the elliptical orbit. (Capderou, 2005)

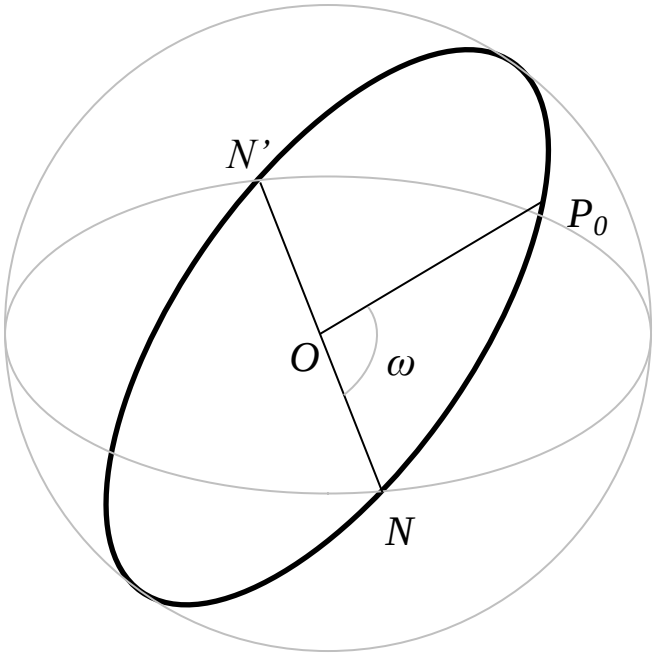

Figure 3.5. Argument of the Periapsis

In Polar coordinates, this is accomplished by the following equation. The variable θ represents the angle (in polar coordinates), of the current position along the orbital path of the object.

$$\theta = \theta + \omega \tag{3.6}$$

For the ISS, this rotates the *XY* coordinates to the following values shown in Table 3.3 and Figure 3.6. In the figure and the table, red represents the coordinates before the transformation, and blue represents the transformed coordinates. This color coding is also followed for the remainder of the figures in this section.

Table 3.3. *XY* Coordinates for ISS within Orbital Plane adjusted for Argument of the Perigee

t	old *x(t)*	new *x(t)*	old *y(t)*	new *y(t)*
0	5830.6217	980.3456	3363.4827	6659.4360
30	3368.3808	-2478.3861	5825.7229	6256.4078
60	4.8972	-5272.2221	6726.9654	4178.0093
90	-3358.5863	-6652.5561	5825.7229	981.1456
120	-5820.8272	-6249.5289	3363.4827	-2477.5863
150	-6722.0699	-4171.1312	0.0000	-5271.4219
180	-5820.8272	-974.2680	-3363.4827	-6651.7552
210	-3358.5863	2484.4637	-5825.7229	-6248.7270
240	4.8972	5278.2997	-6726.9654	-4170.3285
270	3368.3808	6658.6337	-5825.7229	-973.4648
300	5830.6217	6255.6065	-3363.4827	2485.2671
330	6731.8644	4177.2088	0.0000	5279.1027
360	5830.6217	980.3456	3363.4827	6659.4360

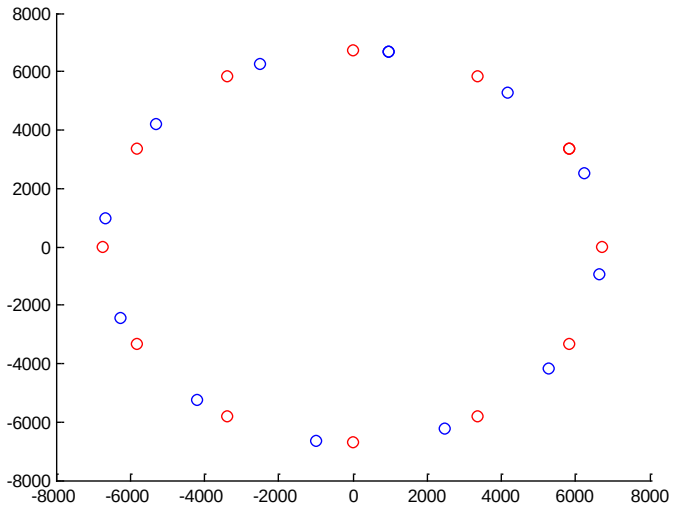

Figure 3.6. Graphical Representation of ISS Orbit within Orbital Plane with Adjustment for Argument of the Perigee

3.2.1.5 The Inclination

The inclination, *i*, describes the angle between the orbital plane and the equatorial plane of the Earth. In Figure 3.7, the inclination describes the angle between the orbital plane (*N*, *O*, P_0) and the equatorial plane (*N*, *O*, *y*).

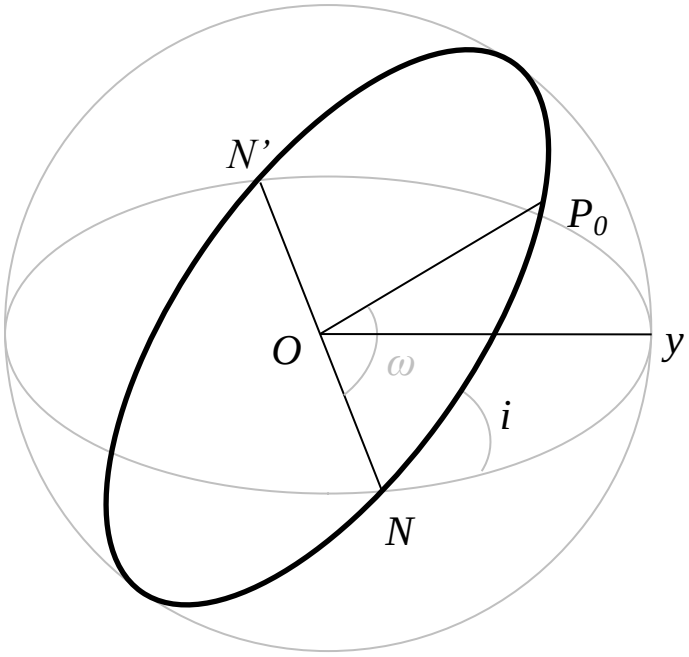

Figure 3.7. Inclination

In Cartesian coordinates, this translation is accomplished by the following equations.

$$x = x \tag{3.7}$$

$$y = y\cos(i) \tag{3.8}$$

$$z = z\sin(i) \tag{3.9}$$

For the International Space Station, the following table and figure demonstrate this translation, with an inclination of approximately 51 degrees, the orbit is inclined 51 degrees from the equator.

Table 3.4. *XYZ* Coordinates for ISS with Adjustment for Inclination

t	*x(t)*	*y(t)*	*z(t)*
0	980.3456	4132.2660	5222.3046
30	-2478.3861	3882.1819	4906.2514
60	-5272.2221	2592.5087	3276.3791
90	-6652.5561	608.8135	769.4107
120	-6249.5289	-1537.3743	-1942.9138
150	-4171.1312	-3270.9854	-4133.8291
180	-974.2680	-4127.5000	-5216.2813
210	2484.4637	-3877.4158	-4900.2282
240	5278.2997	-2587.7427	-3270.3559
270	6658.6337	-604.0475	-763.3874
300	6255.6065	1542.1403	1948.9371
330	4177.2088	3275.7514	4139.8524
360	980.3456	4132.2660	5222.3046

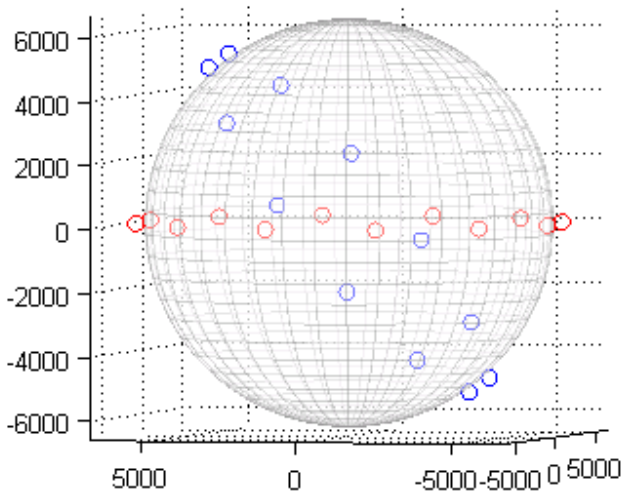

Figure 3.8. Graphical Representation of ISS Orbit with Adjustment for Inclination

3.2.1.6 The Right Ascension of the Ascending Node

The last angle is the right ascension of the ascending node, *Ω*, which describes the angle describing the point where the orbit's path crosses the equatorial plane. In Figure 3.9, *Ω* is the angle *xON*, where *N* is the point where the path of the orbit crosses the Earth's equatorial plane, from below the equator to above the equator, hence 'ascending,' *O* is the Earth's center, and *x* is chosen so that *Ox* points to a distant star to fix the reference frame.

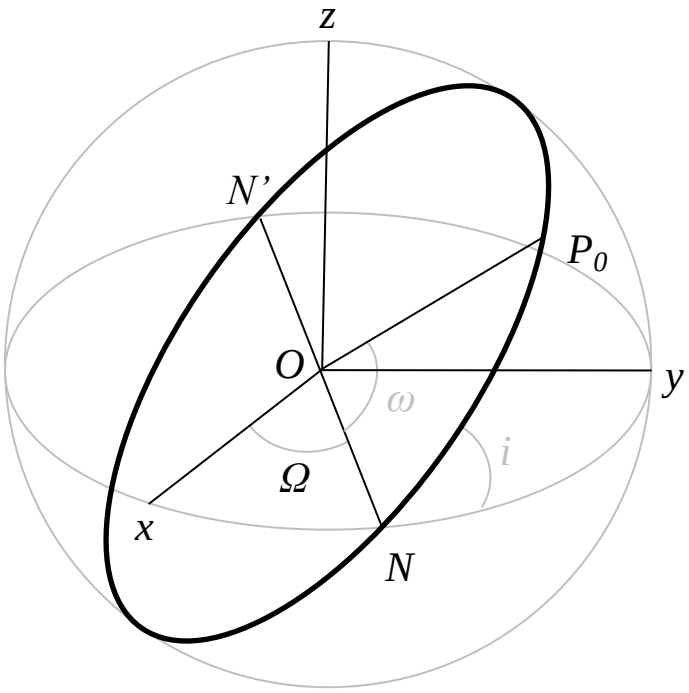

Figure 3.9. Right Ascension of the Ascending Node

This translation is accomplished by the following equation, in cylindrical coordinates.

$$\theta = \theta + \Omega \tag{3.10}$$

Note that Equation (3.10) looks very similar to Equation (3.6), however, because of the order in which the translations are applied, they do not perform the same action. When the argument of the periapsis was applied in Equation (3.6), it was still a two-dimensional problem within the orbital plane. In Equation (3.10) this is no longer the case, and the *z* coordinate is kept fixed, while orientation of the *x* and *y* coordinate's in the Earth's equatorial plane is adjusted.

The following figure and table portray the completed orbit for the International Space Station. In its current form, the table of *XYZ* values is indexed by an angle *t* which ranges from 0 through 360 degrees. The final step for determining the reference points for the ISS is to index these *XYZ* coordinates by time.

Table 3.5. *XYZ* Coordinates for ISS with Adjustment for Right Ascension of the Ascending Node

t	*x(t)*	*y(t)*	*z(t)*
0	-3283.7698	2693.2427	5222.3046
30	-4575.8385	-524.8197	4906.2514
60	-4642.0318	-3601.2453	3276.3791
90	-3464.6132	-5711.7084	769.4107
120	-1359.0710	-6290.7120	-1942.9138
150	1110.4164	-5183.1126	-4133.8291
180	3282.1518	-2685.6906	-5216.2813
210	4574.2206	532.3718	-4900.2282
240	4640.4138	3608.7974	-3270.3559
270	3462.9952	5719.2605	-763.3874
300	1357.4530	6298.2641	1948.9371
330	-1112.0343	5190.6647	4139.8524
360	-3283.7698	2693.2427	5222.3046

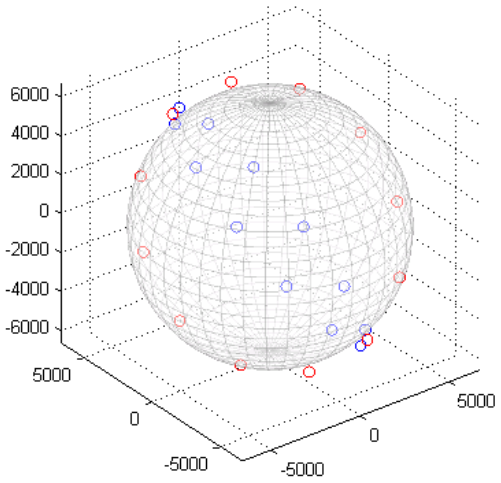

Figure 3.10. Graphical Representation of ISS Orbit with Adjustment for Right Ascension of the Ascending Node

3.2.1.7 Time

The last step is to assign a time value for each set of *XYZ* coordinates. This is done by multiplying the period *T* of the orbit by the fractional revolution of the orbit.

$$t = \frac{T\theta}{360} \tag{3.11}$$

The following table shows the final result, using the orbital period of the ISS, approximately 91 minutes. Note, for simplicity only 12 points were used in defining the orbit. This number is generally higher in actual practice, as is explained in Chapter 4.

Table 3.6. *XYZ* Coordinates for ISS with time indexing

***t* (min)**	***x(t)***	***y(t)***	***z(t)***
0.00	-3283.7698	2693.2427	5222.3046
7.63	-4575.8385	-524.8197	4906.2514
15.25	-4642.0318	-3601.2453	3276.3791
22.88	-3464.6132	-5711.7084	769.4107
30.50	-1359.0710	-6290.7120	-1942.9138
38.13	1110.4164	-5183.1126	-4133.8291
45.76	3282.1518	-2685.6906	-5216.2813
53.38	4574.2206	532.3718	-4900.2282
61.01	4640.4138	3608.7974	-3270.3559
68.64	3462.9952	5719.2605	-763.3874
76.26	1357.4530	6298.2641	1948.9371
83.89	-1112.0343	5190.6647	4139.8524
91.51	-3283.7698	2693.2427	5222.3046

3.2.1.8 MATLAB function for Initial Object Locations

The MATLAB code for the process of defining the initial positions of each object in low Earth orbit is included in the appendix, the function name is ORBIT.

3.2.2 Generate Positions for Sensors

There are two main types of sensors used to track space debris in low Earth orbit from the surface of the Earth, Phased Array Radar and a collection of transmitters and receivers known as the Space Fence. For this simulation, the sensors on the surface of the Earth, referred to as Earth-based sensors, are assumed to be Phased Array Radars. The Space Fence is planned as a future modification to this simulation. Other types of sensors, such as optical telescopes and orbital platforms, are not considered, but could easily be adapted to fit into the simulation. The optical telescopes are utilized mainly for objects farther out than low Earth orbit. And orbital sensors are not currently being used as a persistent source for debris detection.

The initial positions for Earth-based sensors are determined through a similar method to objects in orbit. However, a simpler method for sensors can be used, since they are on the surface of the Earth. The Latitude and Longitude of the sensors position determine where the sensor orbits the Earth, as well as the sensor's starting position. The starting position is important, since it also fixes the sensors' positions relative to each other.

3.2.2.1 The Characteristic's of the Orbit's ellipse.

The sensor's rotation is modeled as an orbit along the surface of the Earth with a period of rotation of one day. The orbital plane of this orbit then becomes a circle – with eccentricity of 0. To determine the length of the semi-major axis, or radius for a circle, the Latitude is used. The Latitude measures approximately the angle between a line running from the center of the Earth to the sensor's location, and a line running from the center of the Earth to a point on the Equator along the same Longitude as the sensor. In reality, this number is slightly different due to the non-spherical nature of the Earth. However, for this simulation, the Earth is assumed to be spherical, therefore the Latitude is used for this simulation. To determine the radius of the circular path of the sensor, the following equation is used.

$$a = \cos(Lat)ER \tag{3.12}$$

Here *Lat* represents the Latitude of the sensor and *ER* is the radius of the Earth, for which we use 6371 km, the Earth's mean radius. To determine the circular path, equations (3.4) and (3.5) are used to determine the *XY* coordinates within the plane of the sensor's rotation. However, since the sensor is assumed to move in a circle, the equation simplifies. In a circle,

the semi-major and semi-minor axes are both equal to the radius, and there is only one focus, the center of the circle. In algebraic notation, $a = b$ and $c = 0$. The equations then become:

$$x = a\cos(t) \tag{3.13}$$

$$y = a\sin(t) \tag{3.14}$$

3.2.2.2 The Orientation of the Orbital Plane with Respect to the Reference Plane

This orientation also simplifies for a circular orbit. Because the sensor moves in a circle parallel to the equator, the orbit never crosses the equatorial plane. Therefore, the Right Ascension of the Ascending Node is not relevant. Similarly, the inclination of this circular path to the orbital plane is 0 degrees. The only parameter that is relevant for the sensors circular orbit is the Argument of the Periapsis (or Perigee). This parameter is replaced by the Longitude of the sensor's position. The Longitude is similar to the Latitude in that it measures an angle formed by two lines meeting at the center of the Earth, with one line moving to the position of the sensor. For Longitude, the base reference is not the Equator, but the Prime Meridian. So Longitude measures the angle between a line running from the center of the Earth to the sensor's location, and a line running from the center of the Earth to a point on the Prime Meridian along the same Latitude as the sensor. To adjust the sensor's position for Latitude, Equation (3.6) is used, modified by replacing ω (the Argument of the Periapsis) with Longitude (Long)

$$\theta = \theta + Long \tag{3.15}$$

3.2.2.3 Time

The last step is to assign a time value for each set of *XYZ* coordinates. This is done exactly the same as with objects in low Earth orbit, only using a period of 1 day. Equation (3.11) then becomes:

$$t = \frac{\theta}{360} \tag{3.16}$$

where θ represents the portion of 360 degrees represented by a given set of *XYZ* coordinates. The following figure and table show the results of these calculations for a sensor with Latitude of 30 degrees.

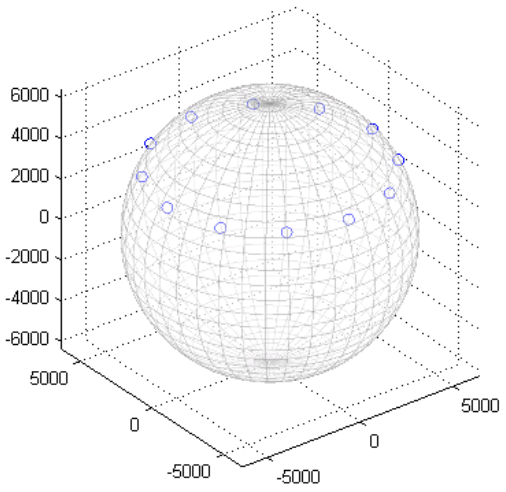

Figure 3.11. Sensor Positions with a Latitude of 30 degrees

Table 3.7. Sensor Positions with a Latitude of 30 degrees

***t* (min)**	***x(t)***	***y(t)***	***z(t)***
0.0000	2758.7239	4778.2500	3185.5000
0.0833	0.0000	5517.4478	3185.5000
0.1667	-2758.7239	4778.2500	3185.5000
0.2500	-4778.2500	2758.7239	3185.5000
0.3333	-5517.4478	0.0000	3185.5000
0.4167	-4778.2500	-2758.7239	3185.5000
0.5000	-2758.7239	-4778.2500	3185.5000
0.5833	0.0000	-5517.4478	3185.5000
0.6667	2758.7239	-4778.2500	3185.5000
0.7500	4778.2500	-2758.7239	3185.5000
0.8333	5517.4478	0.0000	3185.5000
0.9167	4778.2500	2758.7239	3185.5000
1.0000	2758.7239	4778.2500	3185.5000

3.2.2.4 MATLAB function for Initial Sensor Locations

The MATLAB code for the process of defining the initial positions of each sensor is included in the appendix, the function name is SENSORPOSITION.

3.2.3 Initialize Object Lists

As mentioned in the problem summary in Chapter 1, three lists are kept to summarize the knowledge of objects in low Earth orbit. The first list consists of those objects whose locations are known. This list is initially set to 0. The second list consists of those objects whose positions are unknown; all objects in low Earth orbit are initially added to this list. The final list includes objects whose position needs to be updated. Each list is stored as a number of elements for a given time within the simulation. The initial value of each list at time zero would be 0 known objects, *N* unknown objects, and 0 update objects, where *N* is

the total number of objects simulated in low Earth orbit. The transitions between these lists are discussed in Section 3.4.

3.3 Step Two: Increment Time

After all positions of objects in low Earth orbit and sensors on the Earth are initialized, and the initial object lists are set, time is incremented by a set step size, *dt*.

3.4 Step Three: Actions at each Time Step

After the time has been incremented, the simulation performs three actions. It updates the locations of all objects and sensors in the simulation, it calculates which objects have been detected, and it updates the object lists. All of these steps are explained in detail in the following sections.

3.4.1 Update Positions for Objects in Orbit and Sensors

At each time increment in the simulation, all of the positions of the objects in orbit and the sensor locations are updated. The simulation accomplishes this task by linearly interpolating between points in the initial position locations. Because there are only a finite set of reference locations for each object, and the periods of rotation differ for most of the objects in low Earth orbit, the time indexes for each set of *XYZ* coordinates will be different. So whatever time increment *dt* is chosen, it will not match the coordinate time references. The interpolation procedure then works in a two step process. First, it finds which two sets of *XYZ* coordinates it should use for interpolating a given object for a given time, and second, it determines the desired position by linear interpolation.

3.4.1.1 Determining Upper and Lower Time Bounds

For a given time and a given object, the simulation must first determine how far along in time the object has moved from its initial position. The simulation divides the current time by the period of rotation of the object of interest. It then uses the remainder (or modulo) to then find the desired sets of coordinates. The simulation finds the first time index greater than this remainder. It then selects the time index below, or the last time index less than the remainder. These two sets of coordinates are then used for interpolation.

3.4.1.2 Linearly Interpolating to Determine Current Location

Linear interpolation is used to find the desired location. Given two time referenced sets of coordinates, (x_1,y_1,z_1,t_1) and (x_2,y_2,z_2,t_2) and the current time, t, the following equations are used.

$$x = x_2 - \frac{t_2 - t}{t_2 - t_1}(x_2 - x_1) \qquad (3.17)$$

$$y = y_2 - \frac{t_2 - t}{t_2 - t_1}(y_2 - y_1) \qquad (3.18)$$

$$z = z_2 - \frac{t_2 - t}{t_2 - t_1}(z_2 - z_1) \qquad (3.19)$$

3.4.1.3 MATLAB function for Interpolating Object Positions

The MATLAB code for the process of interpolating object positions is included in the appendix, the function name is INTERPORBIT.

3.4.2 Calculate Detections of Objects in Orbit by Sensors

To determine which objects have been acquired by sensors in the given time period, the simulation calculates which objects have passed within the field of view of each sensor, and then determines if they have been detected based upon probability of detection.

3.4.2.1 Determining Objects within Field of View of Sensors

To determine which objects are in a given sensors field of view, the simulation calculates the angle between a line running from the center of the Earth to the sensor's location, and a line running from the sensor's location to the location of the object in low Earth orbit.

$$\theta = asin\left(\frac{u \cdot v}{\|u\|\|v\|}\right) \tag{3.20}$$

After determining this angle, the simulation checks to see if this angle is less than the field of view of the sensor. If it is, it moves on to the next step.

The simulation determines if the object is within the sensors field of view by calculating the azimuth and elevation from the sensor to the object in orbit. The elevation is determined from the angle θ calculated previously.

$$Elevation = 90 - \theta \tag{3.21}$$

The azimuth is determined using the latitudes and longitudes of the sensor and the object in orbit. The longitude is determined relative to the *X* axis, and not the Prime Meridian, this takes into account how far the Earth has rotated from the Prime Meridian at its current position.

$$Azimuth = atan\left(\frac{sin(Long\ A - Long\ B)cos(Lat\ B)}{\cos(Lat\ A)\sin(Lat\ B) - \sin(Lat\ A)\cos(Lat\ B)\cos\left(Long\ A - Long\ B\right)}\right) \quad (3.22)$$

In equation (3.22) *A* represents the Sensor while *B* represents the object in LEO. This azimuth lies between -90 and 90 degrees. To convert to an azimuth between 0 and 360 degrees, the simulation determines which quadrant the resulting azimuth belongs to, adding 180 degrees if necessary.

3.4.2.2 Checking for Detection within Field of View of Sensors

If an object falls within the field of view of a sensor, the simulation generates a random number between 0 and 1 and compares the result to the probability of detection for the given sensor. The test runs for the simulation utilized both of these methods for this probability, as the actual data was unavailable. The first method was to simply set a probability of detection for an object within a sensor's field of view.

$$P[Object\ Detected | Object\ within\ Field\ of\ View] = p \quad (3.23)$$

The second method was to use a probability dependent on the angle of detection, where *E* is the field of view of the sensor in terms of elevation, *M* is the midpoint angle of that field of view, and θ is the angle determined in the previous step. This method increases the probability of detection for an object within the central portions of a field of view, and decreases the likelihood of detection at the limits of the field of view.

$$P[Object\ Detected | Object\ within\ Field\ of\ View] = p\left(1 - \frac{\theta - \mathrm{M}}{E/2}\right)^2 \quad (3.24)$$

In either case, if the randomly generated number is less than the calculated probability, the object has been detected.

3.4.2.3 MATLAB function for Checking for Detections

The MATLAB code for the process of checking for detection within a sensor's field of view is included in the appendix, the function name is FINDANGLE. In addition, the function TESTHORIZON is also used for this process, calculating the azimuth and elevation for a given object in orbit with respect to a given sensor.

3.4.3 Update Known, Unknown, and Update Lists

There are four possible transitions between the known, unknown and update lists.

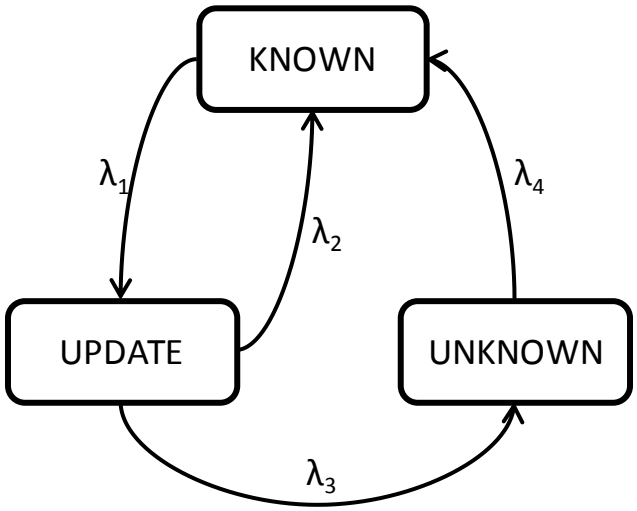

Figure 3.12. Object Lists State Diagram

If an object is detected for the first time, it moves from the unknown list to the known list. If an object is detected while it is on the update list, it moves to the known list. If an object is

detected while on the known list, it remains on the known list. As an object is detected, the last time it was detected is stored. The simulation uses this time to determine when an object moves from the Known or Update lists. If an object is on the Known list, and has not been detected in a certain amount of time, t^*, the object moves to the Update lists. If an object is on the Update list, and has not been detected in a certain amount of time, s^*, the object moves to the Unknown list. In this manner the lists are maintained. The rates, λ_i, from one object list to another, can then be determined based on the behavior of the object lists over time.

The MATLAB code for updating the object lists is included in the main program file, which is included in the appendix, the function name is SIMORBIT.

3.5 Simulation Input

The simulation uses three main sources of data. First, data for the orbital elements of the objects simulated in LEO, which are obtained by observed data. Second, data for the sensors on the surface of the Earth, which are based on the actual geographical locations and limitations of the radars simulated. Third, various parameters within the simulation are set for each run. The following sections describe each of these inputs.

3.5.1 Orbiting Objects Data

There are two main methods used to populate the objects in the simulation. The first method involves randomly generating objects. This method was primarily used to test the mechanics of the simulation for purposes of validation. The second source was a database of objects in

North American Aerospace Defense Command's (NORAD) two line format, which included approximately 5,000 objects of the 15,000 objects currently catalogued. From each of these two line elements, the six orbital parameters of interest were obtained. The entire set of 15,000 objects, or a random subset of these objects, can be used to simulate the effect of changing sensors over time. The original sets of two line elements for each object were obtained from the CelesTrak website, www.celestrak.com, associated with the Center for Space Standards and Innovation. (Kelso, 2010) Sample input data is included in the Appendices.

3.5.2 Sensor Data

For the sensors in the simulation, the locations of eight Phased Array Radar are used. These radar are part of the Space Surveillance Network utilized by the United States Department of Defense. The coordinates for each radar, as well as the azimuth and elevation limits for their fields of view are summarized in the following table, from a 2001 list furnished by Dr. Nicholas Johnson, Chief Scientist for Orbital Debris, NASA Johnson Space Center.

Table 3.8. Phased Array Radar Locations Used

Location	Latitude	Longitude	Azimuth Limits	Elevation Limits
Eglin AFB, FL	30.57° N	273.79° E	120°-240°	1°-105°
Thule AFB, Greenland	76.57° N	291.70° E	297°-177°	3°-80°
RAF Fylingdales, UK	54.37° N	359.33° E	0°-360°	4°-70°
Clear AS, AK	64.29° N	210.81° E	170°-110°	1.5°-90°
Cavalier AS, ND	48.72° N	262.10° E	298°-78°	1.9°-95°
Cape Cod AS, MA	41.75° N	289.46° E	347°-227°	3°-80°
Beale AFB, CA	39.14° N	238.65° E	126°-6°	3°-80°
Eareckson AFB, AK	52.74° N	174.09° E	259°-19°	0.6°-80°

The MATLAB code for loading the data for the actual sensors is located in the appendix, the function name is GETREALSENSORS.

3.5.3 Simulation Parameters

Throughout the development of the simulation, values for several key parameters were chosen. These parameters include the number of steps to partition each object's orbit into, the time step used to increment the simulation, the probability of detection given an object is within a sensor's field of view, and the time threshold for moving an object from the known list to the update list, and from the update list to the unknown list. The values chosen for these parameters are discussed in Chapter 4.

3.6 Simulation Output

As output, the simulation generates the object lists referenced by time. This information demonstrates the dynamic behavior of the system of sensors and orbiting objects as sensors are removed or added over time. The following figure shows sample output for a run of ten days with 25 objects and 2 sensors. The update rate for movement from Known to Update and Update to Unknown was 2.5 for this sample run. From this output, the behavior of the Known, Unknown and Update lists can be examined over time. Sensors can be added or subtracted to simulate changes in mission for individual sensors. The effects of these changes on the Object Lists will then demonstrate the effect the removal or addition of the sensor has on the ongoing task of Space Surveillance. Section 4 includes larger runs demonstrating long run behavior of these Object lists.

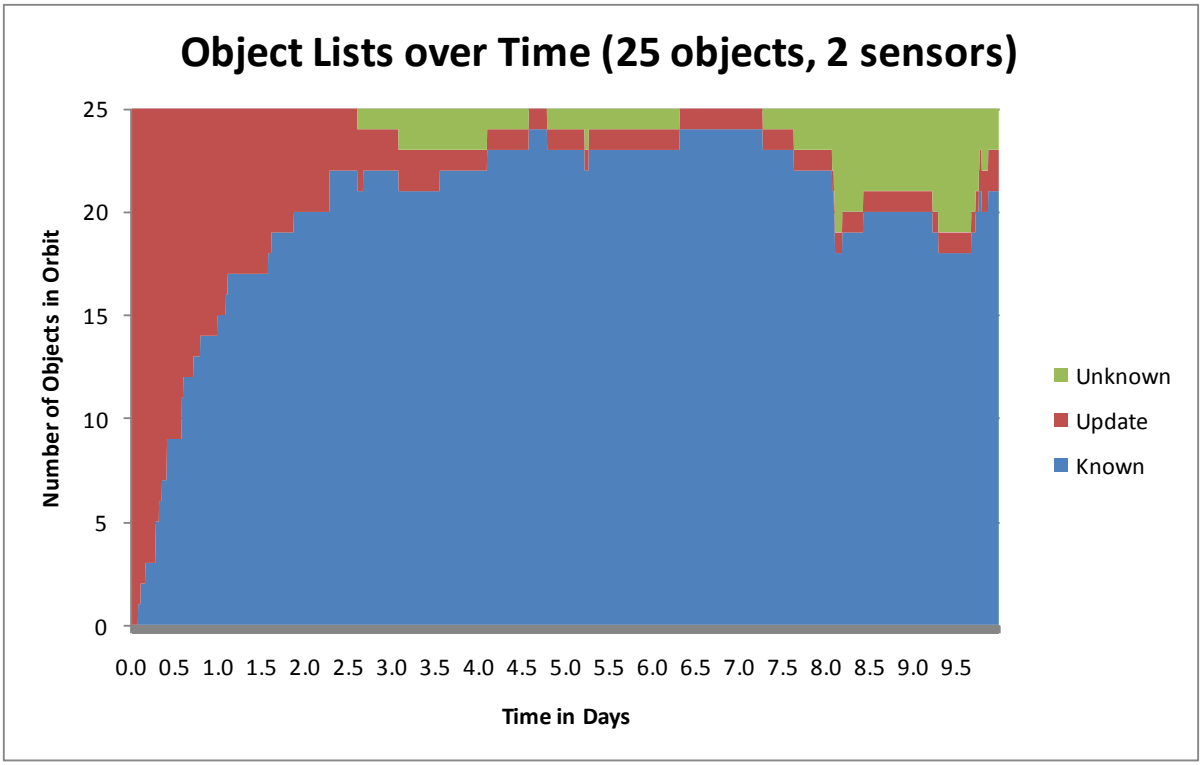

Figure 3.13. Sample Simulation Output

4 EXPERIMENTATION

This section summarizes the experimentation conducted using the simulation. This includes experimentation involved in the selection of key parameters necessary for the simulation to function. In addition, this section discusses some test runs conducted to demonstrate the simulation's ability to model the dynamic system of sensors and orbiting objects.

4.1 Parameter Selection

4.1.1 Number of Partitions for Orbiting Objects

As mentioned previously in Chapter 3, the simulation generates a reference data set for each object in low Earth orbit. This data set consists of a time referenced set of *XYZ* coordinates. One important parameter for the simulation is how many time indexes, or partitions, should be used to generate this set. In the International Space Station (ISS) example illustrated previously, 12 partitions were used. This number is a little low, as the distance between each point in the data set for 12 partitions is over 3,000 kilometers.

To assess the impact of different size partitions on model performance, some small test runs were made varying the number of partitions for each orbit. Figure 4.1 shows the results of these test runs. The runs consisted of 100 orbiting objects and 6 sensors and ran for 1.25 simulated days. The 100 orbiting objects were randomly selected from the CelesTrak list, and the same 100 objects were used for all of the described iterations.

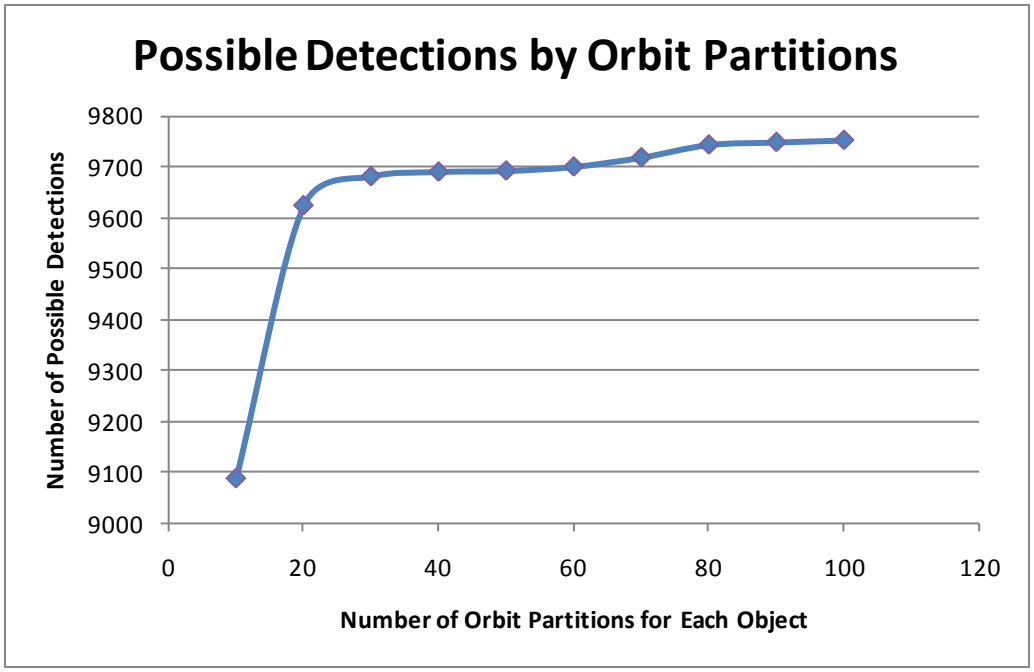

Figure 4.1. Possible Detections by Orbit Partitions

Further runs were made varying the partitions of the sensor position while maintaining the orbit partitions constant. Figure 4.2 shows the results when orbit partitions were held constant at 60 while the number of partitions for sensor positions was varied from 10 to 120. Again, these runs consisted of the same 100 objects and 6 sensors and ran for 1.25 days simulated time. The results of this latter experiment shows the increase in detections when the number of partitions is small. This is a result of the exaggerated field of view of the sensor when only a small number of partitions are used. For the remainder of the simulation runs referenced in this section, we used 60 partitions for both objects in orbit and sensors on the ground.

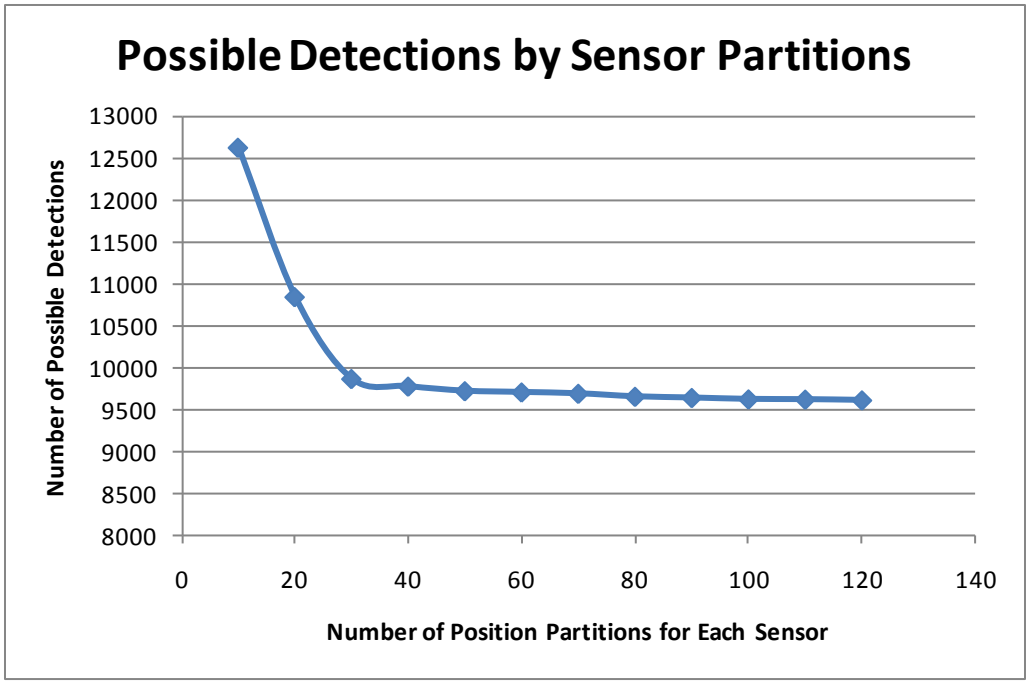

Figure 4.2. Possible Detections by Sensor Partitions

To determine if this size partition is appropriate, consider the resulting error in determining the position of each object in orbit by interpolation. The greater the number of partitions used, the greater the distance is between any two points, and so the interpolation between those two points will result in a greater degree of error. Using 60 partitions, the greatest magnitude of error due to interpolation would be approximately 10 km at an altitude of 1000 km above the surface of the Earth. To determine the approximate error, assume the orbit is near circular between the two points used to interpolate. In Figure 4.3, t_i and t_{i+1} represent the two interpolation points, R_S represents the radius of the orbit at either point, R_I represents the interpolated radius, D represents the straight line distance between t_i and t_{i+1}, θ represents the angle between t_i and t_{i+1} passing through the center of the Earth, and E, the error. The

coordinates of the two points, R_S, D and θ are known. The Error is then calculated using the Pythagorean Theorem. For an object in LEO with an altitude of 1000 km, this error is approximately 10.10 km.

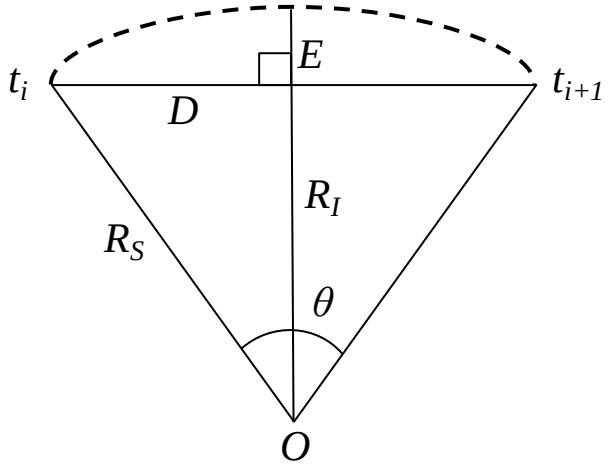

Figure 4.3. Interpolation Error

Table 4.1 shows the resulting approximate error (in kilometers) for varying numbers of partitions for objects in orbit altitudes of 1000 km and 2000 km. For the purposes of our test runs, 60 partitions is satisfactory. If a greater level of accuracy in altitude is required, the number of partitions can be increased, with a penalty to run time.

Table 4.1. Interpolation Error by Number of Partitions

Number of Partitions	Error for an Orbit at 2000km	Error for an Orbit at 1000km
10	410	361
20	103	91
30	46	40
40	26	23
50	17	15
60	11	10
70	8	7
80	6	6
90	5	4
100	4	4

4.1.2 Time Step

The size for the time step used to increment the model has a great impact on both run time and model performance. As expected, the smaller the time step, the greater accuracy within the simulation, but the slower the run time. Figure 4.4 shows the results of a series of experiments to determine this relationship. Using the same 100 objects in orbit, 6 sensors and 1.25 days in simulated time, the simulation was run for various sized time steps. Using 2 minute time steps (900 total steps for 1.25 days), all 100 objects in orbit were detected at least once over the simulation run. However, using 12 minute time steps (150 total steps for 1.25 days), only 92 objects in orbit were detected. Ideally, 2 minute time steps would be chosen, however the run time is over 4 times as long as that for 9 minute time steps, the number used for most of our test runs during simulation development. For all subsequent runs discussed in this section, a time step of 6 minutes was used, a compromise between speed and performance.

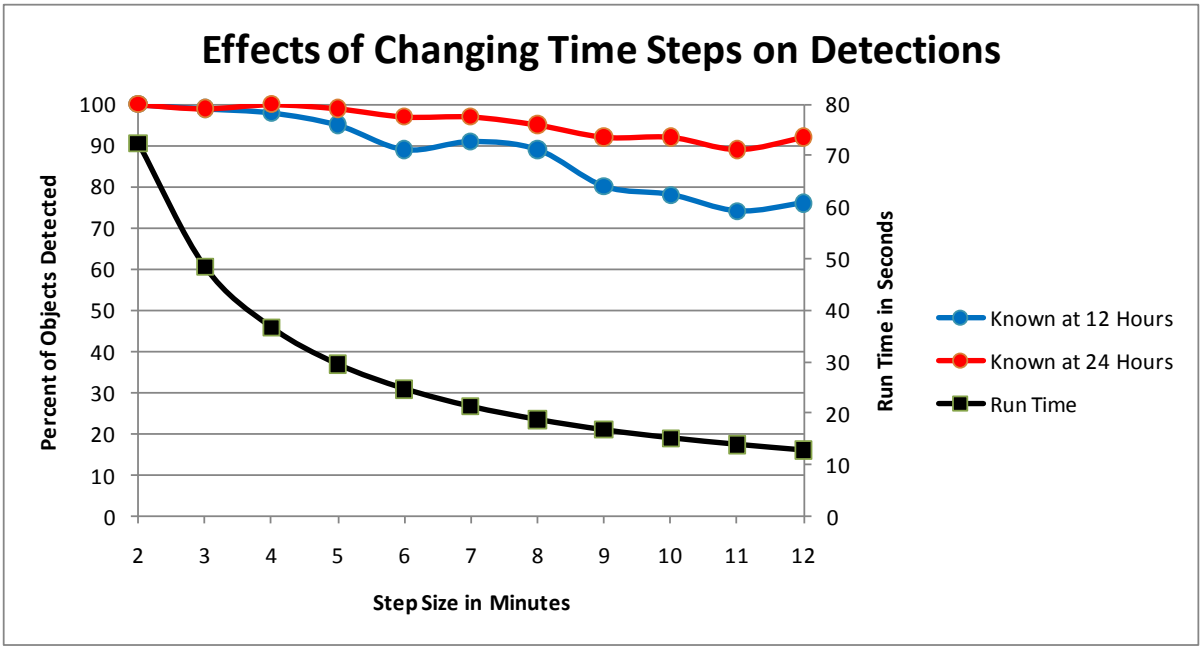

Figure 4.4. Effects of Changing Time Steps on Detections

The impact of this compromise is on how much time passes between each step within the simulation. The greater this time, the more likely it is that a possible detection will not be calculated, degrading the accuracy of the simulation. For a sensor with a 120 degree field of view, the diameter of what this sensor sees at 1000 km is approximately 1700 km using a simple law of cosines approximation. According to the NASA Orbital Debris Program Office, debris travels at 7 to 8 km/s. (Stansbery, 2009) This translates to roughly 420 to 480 km/min. For any time increment greater than 4 minutes, the likelihood of missing possible detections increases, as we saw in out small experiments. Again, for a desired level of accuracy, a smaller time step can be used.

4.1.3 Probability of Detection for an Object within a Radar's Field of View

As mentioned in Chapter 3, two different methods for determining probability of detection were considered for use within the simulation. The first method involved a simple Bernoulli distribution, with a set probability of detection given the object entered the radar's field of view. The second method also involved a Bernoulli distribution for the probability of detection, but also used the angle of detection to adjust this probability, as shown in Equation (4.1). The effect of this adjustment was to cause the probability of detection to increase the closer the object to be detected is to the center of the radar's field of view. Data sets were not available to use in validating this choice of probability, and so a value of 0.99 was selected as what is called the probability factor, which adjusts the calculated probability based on the angle of elevation. This value can be set at different levels to calibrate the model to actual probabilities if they are known.

$$P[Object\ Detected | Object\ within\ Field\ of\ View] = p\left(1 - \frac{\theta - \mathrm{M}}{E/2}\right)^2 \tag{4.1}$$

where p is the probability factor, θ is the angle of elevation of the object in orbit from the given sensor, M is the midpoint of that sensor's elevation range, and E is the sensor's elevation range. In Figure 4.5, the results of altering the probability factor p are shown over runs using 1000 objects in orbit, and 1 sensor, over a period of 10 days. The average size of the Known lists, from 5 to 10 days, are shown, as well as the overall percentage of detections. This percentage is the total number of successful detections divided by the total number of objects which passed through the azimuth and elevation of the sensor during the

course of the simulation run. Little variation occurred as a result of this change in probability factor, as most of the probability of detection is a function of the angle of elevation.

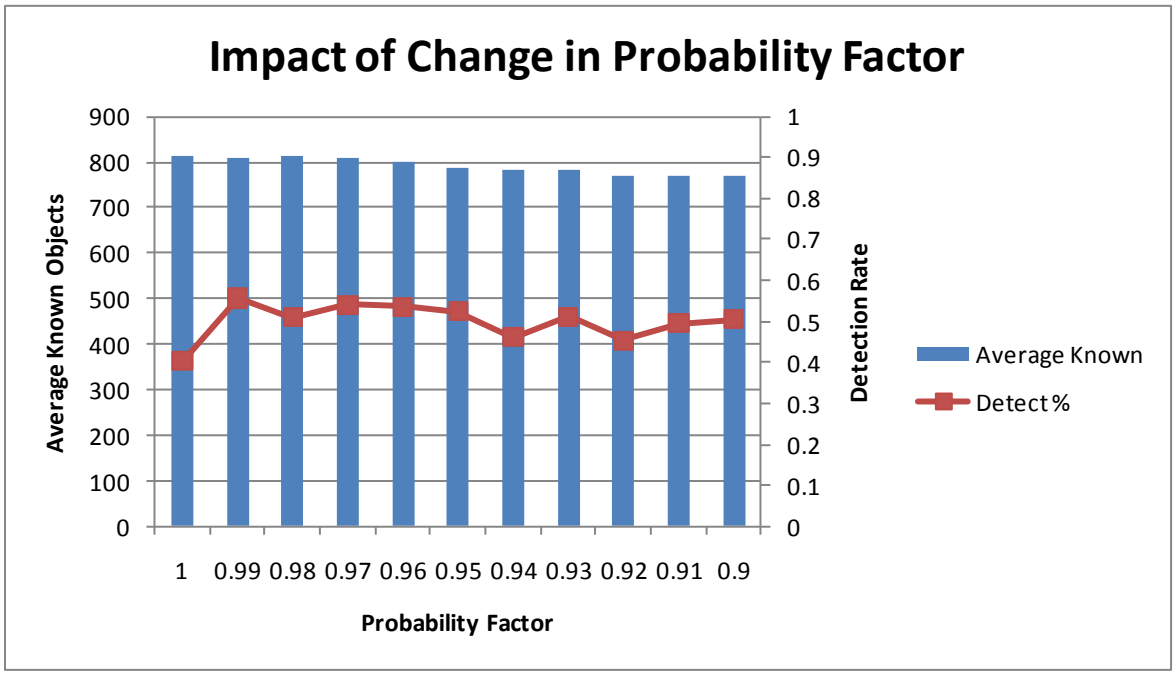

Figure 4.5. Impact of Change in Probability Factor

4.1.4 Movement Times between Object Lists

In this simulation, an object moves between the Known List, the Update List, and the Unknown List as a function of time. This time period was set at 0.5 days, but the actual mechanism that changes objects is unknown. The value of 0.5 days was chosen by taking the average time between detections and adding two standard deviations to this value. This would allow most detections to occur within a lapse time interval. For one sensor, Eglin AFB, the average time between detections was approximately 0.1 days, or 2.4 hours. The standard deviation of the time between detections was 0.18 days, or 4.5 hours. This resulted

in a value of 0.46 days for a lapse time, which was rounded to 0.5 days, or 12 hours, for simplicity. This period of time coincides with the observation that most objects are detected several times a day. (Johnson, 2004)

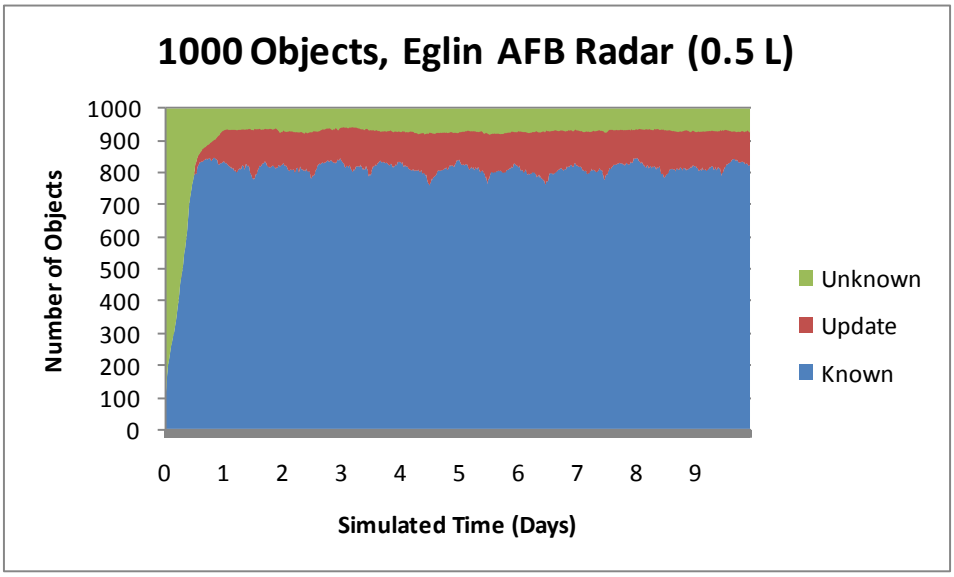

Figure 4.6. Simulation Results for a Lapse Time of 0.5 Days

Figure 4.6 shows the results of a simulation run with 1000 objects and 1 sensor (Eglin AFB) over a simulated time period of 10 days. Steady state behavior seems to emerge at about 1 day, or two lapse time periods. This outcome occurred in all simulated runs made during the development of the model. The steady state could not emerge prior to this time, as no object could have moved from the Known list to the Unknown list in any time less than two lapse periods.

Figures 4.7 and 4.8 show the results of lowering and raising the lapse time, respectively. With a lapse time less than 0.5 days, the behavior of the Object lists over time varies much,

more, with the size of the Known list ranging from approximately 400 objects to 550 objects, a range of 150 objects. This range was approximately 50 objects with a lapse time of 0.5 days. In addition, the average value of the Known objects is approximately 450 with a lapse time of 0.25 versus an average of approximately 800 objects with a lapse time of 0.5 days. Similar results are obtained by increasing the lapse time to 0.75 hours. The longer the lapse time period, the greater chance that objects will remain in the Known list, so the average known objects increases.

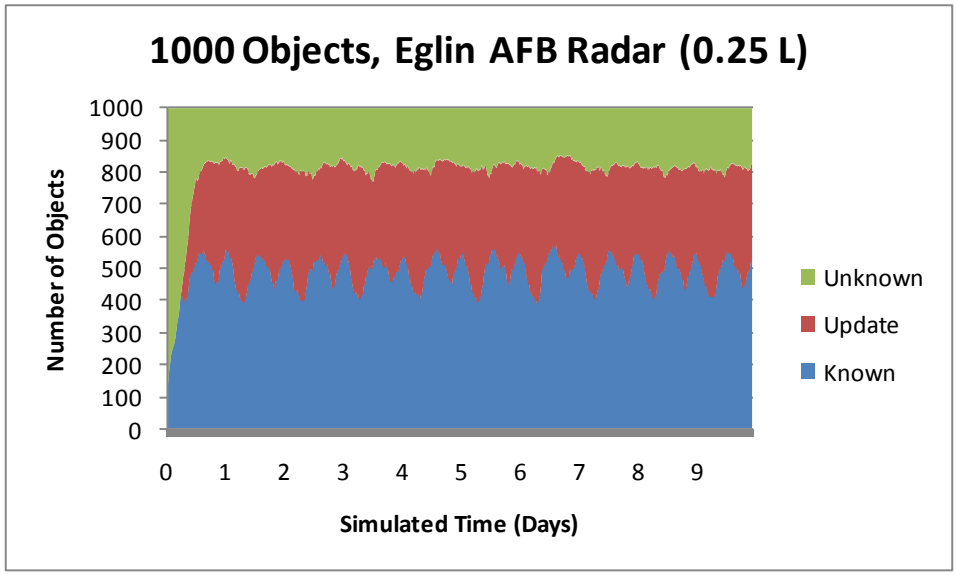

Figure 4.7. Simulation Results for a Lapse Time of 0.25 Days

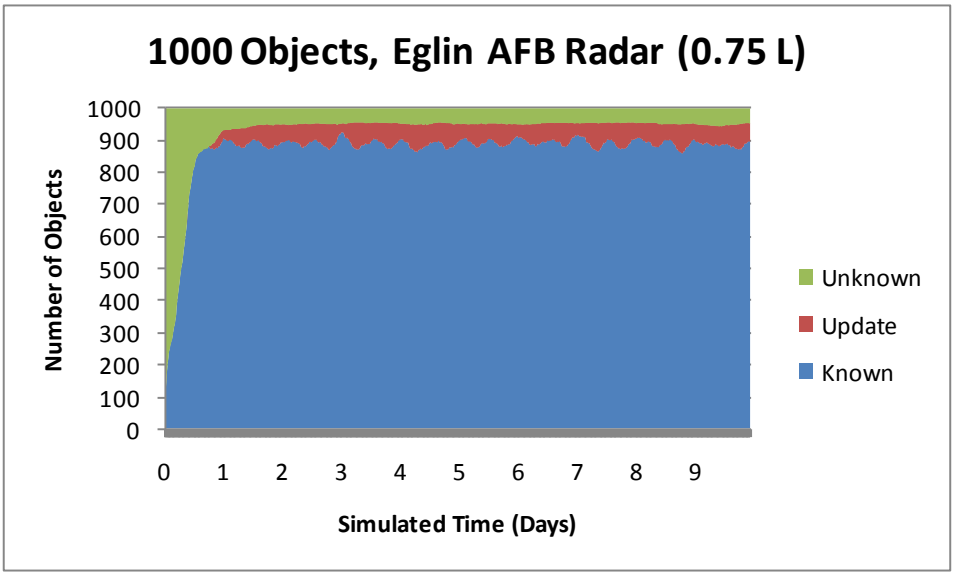

Figure 4.8. Simulation Results for a Lapse Time of 0.75 Days

As mentioned in the assumptions, there are other events which could potentially move an object from the Known list to the Update or even Unknown list. For a man-made satellite, these could include navigation changes triggered remotely or automatically based on certain criteria. For foreign satellites, this information would not always be known, and so satellite positions could change significantly. In addition, collisions between satellites and debris in orbit could alter the path of the satellites orbit significantly. These effects are not currently modeled within the simulation.

4.2 Test Runs

The following section discusses test runs demonstrating the ability of the simulation to address the given problem.

4.2.1 Demonstration of Steady State Behavior

In order for the simulation to help detect changes in performance due to the addition or subtraction of a sensor, the simulation must be able to define the state of the system at a given point in time. This state is defined by the Object Lists for a given time. The following output shows that a steady state is reached after an initial warm-up period. Because the Object Lists begin at 0 known and all unknown, objects must be detected to populate the lists. In this sample run, 5000 random objects and 5 random sensors were generated; these sensors had azimuth ranges of 0 to 360 degrees, with elevation of 0 to 90 degrees. The simulation was run for 60 days with an update rate of 2.5 days. After about 5 days in simulated time, the simulation reached an apparent steady state, where the number of objects in each of the Known, Update, and Unknown lists appear to remain close to their long run values for this system. As mentioned earlier, this behavior exhibited itself in all runs made, with different degrees of variation in each lists size over time. Fewer sensors tended to exhibit a higher level of variation, while more sensors lessened the variation of the list sizes over time.

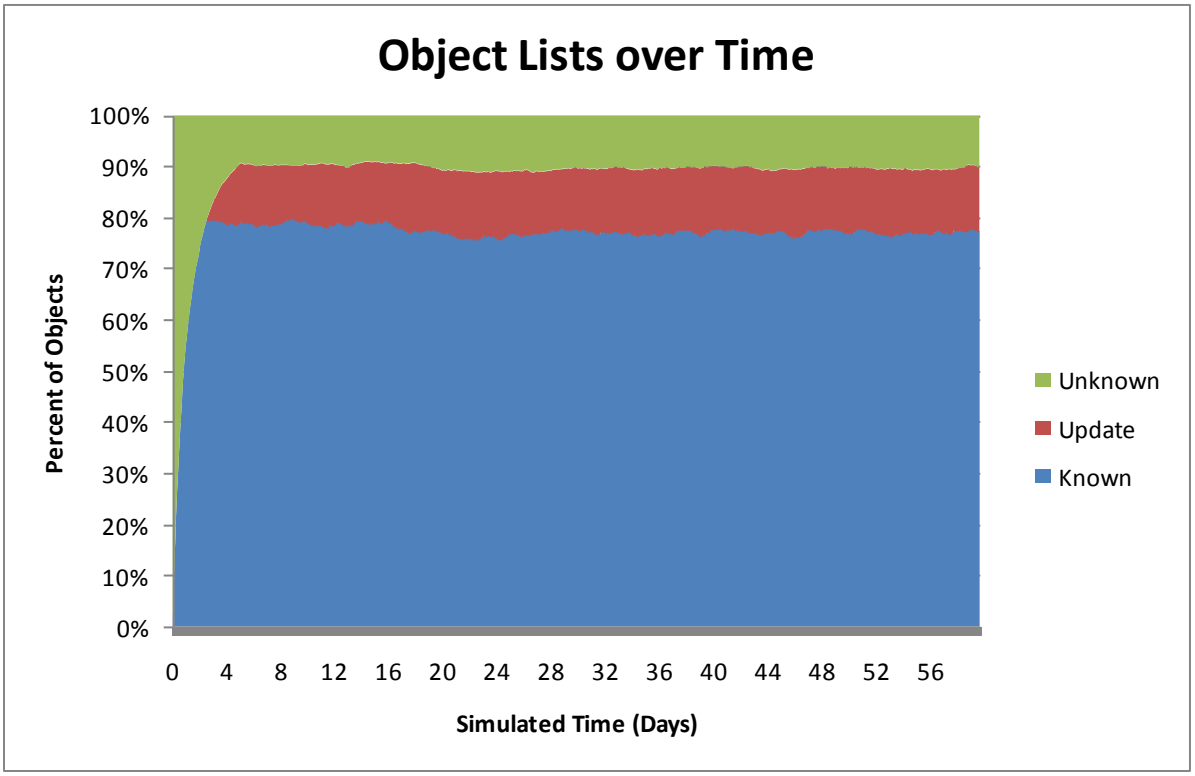

Figure 4.9. Object Lists Over Time

4.2.2 Demonstration of Sensor Changes

The next run subtracted one sensor after 10 simulated days. Then, after 15 simulated days, the sensor was again put back into the simulation. The effect of this change can be seen in the following figure. Because the lists are not updated until the lapse time has passed, it takes two lapse time periods (or 1 days for this run), until the changes in the steady states take effect. In this example, 5000 objects in orbit and 3 sensors were used, and the model run for 20 simulated days with an update rate (lapse time) of 0.5 days. Once the sensor was replaced, there is again a transitional period between the 2 sensor state and a return to the initial 3 sensor state. This period appears to also take two lapse periods, or 1 day to settle

down, although the initial change in the Known lists takes less time as those objects detected are immediately moved to the Known list. There is a cyclical behavior present in the 2 sensor state, as the Eglin radar and the Thule radar remain in coverage. Once the Fylingdales radar is returned to coverage, this variation disappears and returns to the previous steady state.

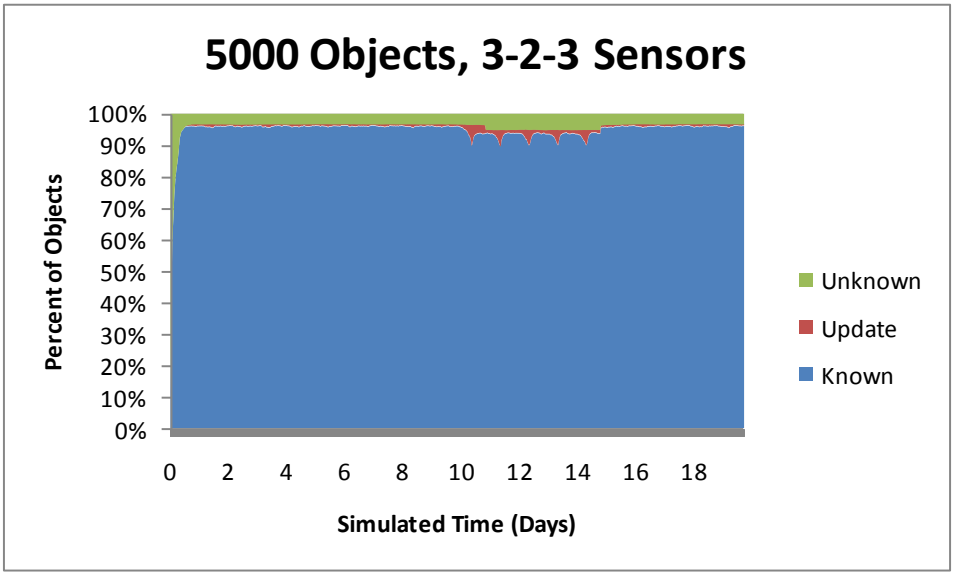

Figure 4.10. Object Lists over Time with Subtraction of Sensor

4.2.3 Demonstration of Sensor Changes with Actual Data for Objects

The following figure shows the results of running the simulation with 4880 actual objects, one to three sensors (Eglin, Thule and Fylingdales) run for 30 simulated days. At the tenth day, 2 additional sensors are added (Thule and Fylingdales), and then at the fifteenth day, 1 sensor (Fylingdales), is removed.

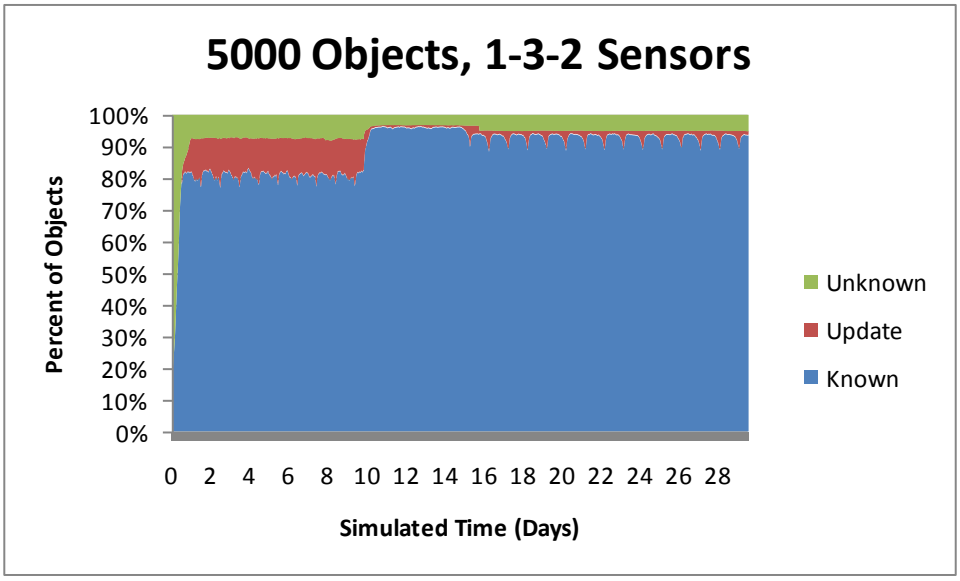

Figure 4.11. Results with Addition then Subtraction of Sensors

For each transition period, the transient time appears to take two lapse time periods, or 1 day. In Table 4.2 the numerical results of the average size of each Object List, as well as the standard deviation for each list, is displayed. The table takes these averages two days after the sensor change to avoid including the transient period in the results. The impact of the addition of two sensors can be seen in an increase from an average of 81% of objects known, to an average of 96% of objects known. In addition, the lists are less sensitive to the orbital position of the sensor. With only one sensor, there is greater variation in the size of the Known list than with three sensors. One sensor exhibits a standard deviation of 1.3% of the Known list, while three sensors exhibit a standard deviation of only 0.2% of the Known list.

Table 4.2. Results with Addition then Subtraction of Sensors

		Known	Update	Unknown
1 Sensor	Mean	80.88%	11.76%	7.36%
2 to 10	Std Dev	1.31%	1.28%	0.14%
3 Sensors	Mean	96.03%	0.66%	3.31%
12 to 15	Std Dev	0.17%	0.17%	0.04%
2 Sensors	Mean	92.97%	1.95%	5.08%
17 to 30	Std Dev	1.31%	1.31%	0.04%

5 FUTURE RESEARCH

Several of the assumptions used during the development of this simulation could be revisited for future improvements to this simulation. In addition, the simulation could be used as a tool in examining additional problems. Some of these assumptions include constant velocity, type of sensor utilized, and orbit perturbations. Additionally, the simulation could be useful in determining geographic locations for future and/or mobile sensors.

5.1 Constant Velocity

For this thesis, constant velocity was assumed. However, this is not the case for elliptical orbits, near circular or not. Further work could be conducted to determine the impact on the performance and accuracy of this model by using variable velocity. In addition to more accurately determining the positions of each object in LEO, this could also be used to increase the fidelity of the detection calculations. The position and speed of the object would be used to determine whether or not an object is detected by a given radar.

5.2 Additional Sensor Types

For this thesis, a phased-array radar was used as the basis of all sensors used. In reality, several different types of sensors, with different capabilities and reliabilities, are used as part of the Space Surveillance Network. In addition to other types of radar, some optical telescopes, as well as the Space Fence, are key contributors to space surveillance. While optical telescopes mainly concentrate on Deep Space and Geosynchronous Orbits, they would add further fidelity to the simulation. In addition, the altitude and azimuth for each sensor could be more accurately modeled, and the resulting impact on the simulation assessed.

5.3 Orbital Perturbations

As mentioned in section 1, the non-spherical shape of the Earth has an impact on all objects in orbit around it. In addition, the gravity of the moon and other objects in space, as well as atmospheric drag, all work to perturb the orbits. Work could be done to assess the value of adding these perturbations into the simulation.

5.3.1 Orbital Perturbations due to the Non-Spherical Shape of the Earth

The non-spherical shape of the Earth causes a precessional effect on the orbit of an object around the Earth. The two orbital elements affected by this precessional effect are the Right Ascension of the Ascending Node and the Argument of the Periapsis. Both of these changes vary as a function of the inclination and eccentricity of the given orbit.

$$\dot{\Omega} = -\frac{Jn\ cos\ i}{a^2(1-e^2)^2} \tag{5.1}$$

$$\dot{\omega} = -\frac{Jn\left(2-\frac{5}{2}sin^2 i\right)}{a^2(1-e^2)^2} \tag{5.2}$$

In Equations (5.1) and (5.2), *Jn* represents the harmonic coefficient for the geopotential, a quantity which is approximately 1.624 x 10^{-3} for the Earth. (Haymes, 1971), *i* represents the orbit's inclination, *a* the length of the orbit's semi-major axis, and *e* the orbit's eccentricity. For the change in the Right Ascension, Ω, the effect is that as the orbit moves around the Earth, it is pulled Westward, and so crosses the Equator (the Right Ascension), further and further from its 'original' crossing point. The variables *i* and *e* refer to the orbit's inclination and eccentricity respectively. This effect is not realized for a polar orbit, with and inclination of 90 degrees, as cos(90)=0. The second element effected is the Argument of the Periapsis, *ω*. As the object moves around the Earth, its perigee will shift at each subsequent orbit. An orbit around the Earth that is near equatorial (with 0 degrees inclination), will exhibit little or no change in the Argument of the Periapsis, as sin(0)=0.

For all other orbits, these changes have significant impact on the orbits. As an example, for an orbit with an inclination of 45 degrees, the Right Ascension will change by -3.37º per day, and the Argument of the Periapsis by 3.58 º per day. (Haymes, 1971)

The implications for this on the simulation are great. The simulation currently uses the Right Ascension and Argument of the Periapsis during the initialization phase of the run. These values are used to generate the reference tables from which the locations of the objects in

orbit are updated throughout the run. But the values of the Right Ascension and the Argument of the Perigee are constantly changing, so these reference tables would need to be constantly changed. One possible solution would be to generate a reference table covering the entire desired run length of the simulation. This would only have to be generated once. Then subsequent runs could interpolate positions from these pre-generated numbers that include perturbative effects.

5.4 Geographical Locations for Additional Sensors

This simulation could be adapted as a decision tool for locating new sensors or mobile sensors to maximize effectiveness. Given certain capabilities of a sensor, requirements for detection coverage, and geographical and political constraints, possible locations could be evaluated using this simulation.

REFERENCES

3-14, J. P. (2009, January 9). Space Operations. United States Strategic Command.

Arney, K. (2008). *Global Sensor Management: Allocation of Military Surveillance Assets.* Raleigh, NC: North Carolina State University.

Capderou, M. (2005). *Satellites: Orbits and Missions.* Paris: Springer.

Dulin, J. L. (2009). *Global Sensor Management: Real-Time Reallocation of Military Assets among Competing Tasks and Functions* . Raleigh, NC: North Carolina State University.

Grimaldi, B. (1997, June 19). *NASA/NORAD 2-Line Elements*. Retrieved February 24, 2010, from Science at NASA: http://science.nasa.gov/Realtime/rocket_sci/orbmech/state/2line.html

Haymes, R. C. (1971). *Introduction to Space Science.* New York: John Wiley & Sons, Inc.

Hoots, F. R., & Roehrich, R. L. (1980). *Spacetrack Report No. 3.* Peterson AFB, CO: Aerospace Defense Center.

Johnson, N. L. (2004). The world state of orbital debris measurements and modeling. *Acta Astronautica 54* , 267-272.

Kelso, T. (2010, February 22). *CelesTrak*. Retrieved March 14, 2010, from CelesTrak: http://celestrak.com/

Liou, J. (2010, January). An Updated Assessment of the Orbital Debris Environment in LEO. *NASA Orbital Debris Quarterly News* , pp. 7-8.

Liou, J.-C., Matney, M. J., Anz-Meador, P. D., Kessler, D., Jansen, M., & Theall, J. R. (2002). *The New NASA Orbital Debris Engineering Model ORDEM2000.* Houston, Texas: National Aeronautics and Space Administration.

Stansbery, E. (2009, July 22). *NASA Orbital Debris Program Office*. Retrieved March 13, 2010, from NASA: http://www.orbitaldebris.jsc.nasa.gov/index.html

U.S. Strategic Command Public Affairs Office. (2009, March). *United States Strategic Command Fact Sheet*. Retrieved March 3, 2010, from United States Strategic Command: http://www.stratcom.mil/factsheets/snapshot/

United Nations Committee on the Peaceful uses of Outer Space. (1999). Technical Report on Space Debris. *Scientific and Technical Subcommittee* (pp. 1-41). New York, NY: United Nations.

APPENDICES

SIMORBIT (Main Program):

```
function
[times,known,unknown,update,Object,Sensor,period,sensorperiod,diffTimes] =
simorbit(numorbits,numsensors,userealsensors,lapsetime,usesatdata,SD,simti
me)
% THESIS VERSION NOTES

% Simulates detection of objects in Low Earth Orbit (LEO) by ground based
% sensors.  Outputs lists of Known, Update, and Unknown sensors over time.
% Either randomly generates objects and sensors, or user provided input
% data.  Assumes no perturbations to orbits due to shape of Earth, or
other
% factors.

% DATE LAST MODIFIED: 19 March 2010

% INPUT PARAMETERS
% numorbits=1;          % number of objects to simulate in orbit
% numsensors=1;         % number of sensors to simulate
% userealsensors;       % 1 if real sensors used, 0 o/w
% lapsetime = 2.5;      % time since last detection for satellite to
change lists
% usesatdata = 0;       % 1 if sat data used, 0 o/w generates random data
% SD;                   % database of orbital parameters of satdata
% simtime=10;           % time to be simulated (in days)

% OUTPUT
% times: vector of size n=numtimesteps with simtime at each step
% known: vector of size n with values of known list at each time step
% unknown: vector of size n with values of unknown list at each time step
% update: vector of size n with values of update list at each time step
% Object: reference table (array) of orbiting values used for run
% Sensor: reference table (array) of sensors used for run
% period: vector with object periods
% sensorperiod: vector with sensor periods
% diffTimes: test vector used to calculate avg time between unique time
% acquisitions

tic

% SET RUN PARAMETERS
graphics = 0;               % 0 graphics off, 1 graphics on
tindex=0.00;                % initial starting time
tstep=0.00625;              % 9 minutes with 0.00625
numtimeint=round(simtime/tstep);   % number of time steps used to reach
sim time
n=60;                       % number of points used to estimate each orbit
diffTimes=[];
k=1;
```

```
fprintf('----------------------START----------------------');

% PLOT 3D REPRESENTATION OF EARTH
if (graphics==1)
    hold off;
    drawearth;
end

% GENERATE ORBITS (for numorbits of objects in Low Earth Orbit)
classification=zeros(1,numorbits);
LTA=zeros(1,numorbits);
badorbitcount=0;

Object=zeros(4,n+1,numorbits); %1 = X, 2 = Y, 3 = Z, 4 = time
period=zeros(1,numorbits);

for i=1:numorbits

    if (usesatdata==1)   %use external data SD
        e=SD(i,3);
        p=SD(i,4);
        inc=SD(i,1);
        L=SD(i,2);
        RPD=SD(i,6);
        M=SD(i,5);
    else                 %use random data
        e=unifrnd(0.0,0.12);
        p=unifrnd(0,180);
        inc=unifrnd(0,90);
        L=unifrnd(0,360);
        RPD=unifrnd(12.5,16.5);
        M=unifrnd(0,360);
    end

    [t,A,B,C,badorbit]=orbit(e,p,inc,L,RPD,M,n); %generates orbit vectors
from orbital parameters
    if (badorbit==1) %if orbit identified bad orbit == orbit with < 160km
altitude
        badorbitcount=badorbitcount+1;
    end

    while (badorbit==1) %if badorbit detected, alter eccentricity and
period until goodorbit achieved
        e=unifrnd(0.0,0.12);
        RPD=unifrnd(12.5,16.5);
        [t,A,B,C,badorbit]=orbit(e,p,inc,L,RPD,M,n);
    end

    Object(1,:,i)=A;
    Object(2,:,i)=B;
    Object(3,:,i)=C;
```

```
    Object(4,:,i)=t;

    period(i)=1/RPD;
    classification(i)=0;% 0 = unknown, 1 = known, 2 = needs updating
    LTA(i)=0; % Last Time Acquired

    % OPTIONAL GRAPH INSTRUCTIONS
    % plot3(A(1),B(1),C(1),'o');
    % grid on;
    % axis square;
    % hold on;

end

badorbitcount

% TEST ORBITS FOR CRASHES

numcrash=testorbitsforcrash(Object,numorbits,n) % s/b 0!

% GENERATE SENSORS (for numsensors located on the surface of the Earth)

Sensor=zeros(4,n+1,numsensors); %1 = X, 2 = Y, 3 = Z, 4 = time
sensorperiod = zeros(1,numsensors);
sensorLat = zeros(1,numsensors);
sensorLong = zeros(1,numsensors);
LAzimuth1 = zeros(1,numsensors);
RAzimuth1 = zeros(1,numsensors);
LAzimuth2 = zeros(1,numsensors);
RAzimuth2 = zeros(1,numsensors);
LElevation = zeros(1,numsensors);
UElevation = zeros(1,numsensors);
MElevation = zeros(1,numsensors);

numsensorstogenerate=numsensors+2;  %to allow for adding sensors to sensor
network during run
if (userealsensors==1)
% Generate actual sensors

[Sensor,sensorperiod,LAzimuth1,LAzimuth2,RAzimuth1,RAzimuth2,LElevation,UE
levation,MElevation]=genrealsensors(n,numsensors);
    numsensorstogenerate=8;
else
    % Generate random sensors
    for k=1:numsensorstogenerate
        lat=unifrnd(-90.,90.);
        long=unifrnd(-180.,180.);
        [st,sA,sB,sC]=sensorposition(lat,long,n);

        Sensor(1,:,k)=sA;
        Sensor(2,:,k)=sB;
```

```
        Sensor(3,:,k)=sC;
        Sensor(4,:,k)=st;

        LAzimuth1(k) = 0.;
        RAzimuth1(k) = 360.;

        LAzimuth2(k) = 0.;
        RAzimuth2(k) = 0.;

        LElevation(k) = 0.;
        UElevation(k) = 90.;
        MElevation(k) = 0.5*(UElevation(k)-LElevation(k));

        sensorLat(k)=lat;
        sensorLong(k)=long;

        sensorperiod(k)=1;
    end
end


% MAIN SIMULATION -
% - increments time by tstep for numtimeint
% - for each discrete time event, calculates position of objects in orbit
%   and sensors on the earth, and then determines whether or not objects
%   have been detected
% - updates respective lists of objects in orbit (known, update, unknown)

known=zeros(1,numtimeint);
unknown=zeros(1,numtimeint);
update=zeros(1,numtimeint);
times=zeros(1,numtimeint);

known(1)=0;
unknown(1)=numorbits;
update(1)=0;
times(1)=tindex;

printtime=160;

numpossdetect=0;
numdetect=0;
testvector=zeros(3,numorbits);
sensor1vector=zeros(3,numsensors);

for j=1:numtimeint

    if (j==2400) % PROGRAM subtraction of sensor
        numsensors=numsensors-1
        tindex
    end
```

```
     if (j==4800)   % PROGRAM addition of sensor
        numsensors=numsensors+1
        tindex
    end


    if (j==printtime)
        printtime
        printtime=printtime + 160;
    end

    if j > 1
        times(j)=tindex;
        known(j)=known(j-1);
        unknown(j)=unknown(j-1);
        update(j)=update(j-1);
    end

    for i=1:numorbits
        tremindex=rem(tindex,period(i));


[testvector(1,i),testvector(2,i),testvector(3,i)]=interporbit(Object(:,:,i
),tremindex,'mo','m',graphics);

        for m=1:numsensors

            sensortime=rem(tindex,sensorperiod(m));

[sensor1vector(1,m),sensor1vector(2,m),sensor1vector(3,m)]=interporbit(Sen
sor(:,:,m),sensortime,'ro','r',graphics);


[testangle,azelevcheck]=findangle(sensor1vector(:,m),testvector(:,i),graph
ics, LAzimuth1(m), LAzimuth2(m),RAzimuth1(m),
RAzimuth2(m),LElevation(m),UElevation(m));

            objdetect=0;

            % determine probability of detection
            if testangle > 90   %object above sensor's local horizon
                numpossdetect=numpossdetect+1;
                probdetect=unifrnd(0,1);
                % Probability of 1 in the middle of the average range at
                % 45 degrees of elevation, and 0 at extremes (0 and 90
                % elevation)
                if (probdetect < (0.99*(1-
(((90+LElevation(m)+MElevation(m)-testangle)/MElevation(m))^2))))
                    objdetect=1;
                end
```

```
            end

            timecheck=lapsetime+LTA(i);
            timecheck2=2*lapsetime+LTA(i);

            if (objdetect==1) % satellite is detected
                numdetect=numdetect+1;
                if classification(i)==2 % the satellite was on update list
                    update(j)=update(j) - 1;
                    known(j)=known(j) + 1;
                else if classification(i)==0 % the satellite was on the
unknown list
                       unknown(j)=unknown(j) - 1;
                       known(j)=known(j) + 1;
                    end
                end

                classification(i)=1;
                if ((LTA(i) ~= tindex) && (LTA(i)>0)) %add interval
between acquisitions to diffTimes vector
                    diffTimes(k)=tindex-LTA(i);
                    k=k+1;
                end

                LTA(i)=tindex;

            else if (tindex > timecheck) && (classification(i)==1)
                    classification(i)=2;
                    known(j)=known(j) - 1;
                    update(j)=update(j) + 1;
                else if (tindex > timecheck2) && (classification(i)==2)
                    classification(i)=0;
                    unknown(j)=unknown(j) + 1;
                    update(j)=update(j) - 1;
                    end
                end

            end


        end
    end


    tindex=tindex+tstep;
end

numpossdetect
numdetect
toc
```

```
return
```

GETREALSENSORS (generates sensor data for actual sensor locations):

```
function[Sensor,sensorperiod,Laz1,Laz2,Raz1,Raz2,Lel,Uel,Mel]=genrealsenso
rs(n,numsensors)
% generates sensors using actual LAT/LONG (or close approximations
thereof)

% Eglin AFB (Florida) Phased array radar 30.57N, 273.79

% Thule AFB (Greenland) Phased array radar 76.57N, 291.70E
% RAF Fylingdales (Great Britain) Phased array radar 54.37N, 359.33E
% Clear AS (Alaska) Phased array radar 64.29N, 210.81 E
% Cavalier AS (North Dakota) Phased array radar 48.72N, 262.10E
% Cape Cod AS (Massachusetts) Phased array radar 41.75N, 289.46E
% Beale AFB (California) Phased array radar 39.14N, 238.65E

% Eareckson AFB (Alaska) Phased array radar 52.74N, 174.09E

Lat = [30.57,76.57,54.37, 64.29, 48.72, 41.75, 39.14, 52.74];
Long = [-86.21,-68.30,-0.67, -149.19, -97.90, -70.54, -121.35,-185.91];

LAzimuth1 = [120,297,0,170,298,347,126,259];
RAzimuth1 = [240,360,360,360,360,360,360,360];
LAzimuth2 = [0,0,0,0,0,0,0,0];
RAzimuth2 = [0,177,0,110,78,227,6,19];
LElevation = [1,3,4,1.5,1.9,3,3,0.6];
UElevation = [105,80,70,90,95,80,80,80];

for i=1:numsensors
    senslat=Lat(i);
    senslong=Long(i);
    [st,sA,sB,sC]=sensorposition(senslat,senslong,n);

    Sensor(1,:,i)=sA;
    Sensor(2,:,i)=sB;
    Sensor(3,:,i)=sC;
    Sensor(4,:,i)=st;

    sensorperiod(i)=1;
    Laz1(i)=LAzimuth1(i);
    Raz1(i)=RAzimuth1(i);

    Laz2(i)=LAzimuth2(i);
    Raz2(i)=RAzimuth2(i);

    Lel(i)=LElevation(i);
    Uel(i)=UElevation(i);
    Mel(i)=0.5*(UElevation(i)-LElevation(i));

end
```

```
return
```

FINDANGLE (determines azimuth and elevation angles between sensor and object):

```
function [angle,azelevcheck]=
findangle(vectorA,vectorB,graphics,Laz1,Laz2,Raz1,Raz2,Lel,Uel)
% function determines azimuth and elevation between sensor and orbiting
% object
% Vector A (v1) is the sensor's position, Vector B (v2) is the orbiting
object's
% position

azelevcheck=0;
v1=[vectorA(1),vectorA(2),vectorA(3)];
v2=[vectorB(1),vectorB(2),vectorB(3)];
v3=[v1(1)-v2(1),v1(2)-v2(2),v1(3)-v2(3)];

% Determine angle between v1 and v3
C = dot(v1,v3);
A = norm(v1);
B = norm(v3);

angle=acosd(C/(A*B));

% Determine azimuth and elevation between v1 and v2;
[az,elev1]=testhorizon(v1,v2);

    azimuth=az;

    elevation=angle-90.;

if (angle > 90.)

        if (((Laz1 < azimuth) && (azimuth < Raz1)) || ((Laz2 < azimuth) &&
(azimuth < Raz2)))

            if ((Lel < elevation) && (elevation < Uel))

                if (graphics==1)
                    plot3(v2(1),v2(2),v2(3),'bo');

line([v2(1),v1(1),0],[v2(2),v1(2),0],[v2(3),v1(3),0],'Color','r');
                    hold on
                end
                azelevcheck=1; % object within both elevation and azimuth
            else
                if (graphics==1)
                    plot3(v2(1),v2(2),v2(3),'yo');  %object within azimuth
but not elevation
                end
            end
        else
            if (graphics==1)
```

```
                plot3(v2(1),v2(2),v2(3),'go');    %object not within
azimuth
            end
        end
end

return
```

INTERPORBIT (interpolates to find desired object position given time):

```
function
[targetA,targetB,targetC]=interporbit(object,tindex,color,color1,graphics)
% function interpolates object for a given time (tindex)

[row,timeindex]=find(object(4,:)>tindex,1);
lowertimeindex=timeindex-1;

targetA=object(1,timeindex)-(object(4,timeindex)-
tindex)/(object(4,timeindex)-
object(4,lowertimeindex))*(object(1,timeindex)-object(1,lowertimeindex));
targetB=object(2,timeindex)-(object(4,timeindex)-
tindex)/(object(4,timeindex)-
object(4,lowertimeindex))*(object(2,timeindex)-object(2,lowertimeindex));
targetC=object(3,timeindex)-(object(4,timeindex)-
tindex)/(object(4,timeindex)-
object(4,lowertimeindex))*(object(3,timeindex)-object(3,lowertimeindex));

% graphs output
if (graphics==1)
    % uncomment next line for additional display options
%plot3(object(1,:),object(2,:),object(3,:),color1);
%plot3(object(1,timeindex),object(2,timeindex),object(3,timeindex),'o')
%plot3(object(1,lowertimeindex),object(2,lowertimeindex),object(3,lowertim
eindex),'o')
plot3(targetA,targetB,targetC,color)
hold on;

end

return
```

ORBIT (generates Cartesian coordinate reference table given orbital elements):

```
function [tm,x,y,z,badorbit] = orbit(e,p,inc,L,RPD,M,n)
% Orbital Elements
% Shape, Size and Orientation of Ellipse
%   Eccentricity (e)
%   Semi-major axis (a) calculated from Period (T) in days, data is in
%       revolutions per day (RPD)
%   Argument of the periapsis (p)
% Orientation of orbital plane
%   Inclination (inc)
%   Longitude of the ascending node (L)
% Position of orbiting body
%   Mean anomaly at epoch (M)not yet!!!!!!
%
% n is number of points along orbit
% Orbital State Vectos
%  Position (x,y,z)
%  Velocity (x,y,z) don't use this yet!
%  theta = originally theta, working towards time (seconds)

% Generate equation of ellipse (in two dimensions)
% eccentricity of an ellipse = sqrt(a^2 - b^2)/a where a is semi-major
% axis, b is semi-minor axis

% Derive semi-major axis using formula T = 2Pi*sqrt(a^3/mu)
ER = 6371;          % estimated radius of the EARTH in km
mu = 398600.4418;   % geocentric gravitational constant (km^3/s^2) +/-
0.0008
T = 1/RPD;
a = ((((T*24*60*60)/(2*pi))^2)*mu)^(1/3);
b = sqrt((a^2)*(1-e^2));
c = sqrt((a^2)-(b^2));
badorbit=0;

if ((a-c) < (ER + 160))
    badorbit=1;
    tm=0;
    x=0;
    y=0;
    z=0;
    return;
end


x1 = [];
y1 = [];
z1 = [];
theta = [];
for i = 1:n+1   %uses degrees, Cartesian coordinates
    t = (360*i)/n;
```

```
    x1(i) = a*cosd(t)+c;
    y1(i) = b*sind(t);
    theta(i)=i;

end

[theta1,r1]=cart2pol(x1,y1);

% orients orbit in plane (Argument of the periapsis)
for i=1:n+1
    r2(i)=r1(i);
    theta2(i)=theta1(i)+pi*p/180;
end

z1 = zeros(n+1);

% inclination, uses degrees, Cartesian coordinates
[x2,y2]=pol2cart(theta2,r2);
z2=z1;

for i=1:n+1
    x3(i)=x2(i);
    y3(i)=y2(i)*cosd(inc);
    z3(i)=y2(i)*sind(inc);
end

[theta3,r3]=cart2pol(x3,y3);

%Longitude of the ascending node, uses Polar, radians
for i=1:n+1
    theta4(i)=theta3(i)+pi*L/180;
    r4(i)=r3(i);
    z4(i)=z3(i);
end

[x4,y4]=pol2cart(theta4,r4);

x=x4;
y=y4;
z=z4;
tm=(theta-1)*T/n;

return
```

SENSORPOSITION (generates Cartesian coordinate reference table given sensor LAT LONG):

```
function [tm,x,y,z] = sensorposition(Lat,Long,n)
% MOD 18 MAR 10, changed i to i-1, line 49 to i

% given latitude, longitude of sensor, time T, number of points n
% output would be x,y,z coordinates for given time t

% center of earth assumed to be (0,0,0)

% radius = cos(lat)*radius of earth
% equals a, eccentricity 0

% z = sin(lat)*radius
% z is north, south displacement from equator (z=0)
ER = 6371;      % Earth's radius (km)

x1 = [];
y1 = [];
z = [];
theta = [];
a=cosd(Lat)*ER;
z0=sind(Lat)*ER;
for i = 1:n+1   %uses degrees, Cartesian coordinates
    t = (360*(i-1))/n;
    x1(i) = a*cosd(t);
    y1(i) = a*sind(t);
    z1(i)=z0;
    theta(i)=i-1;
end

[theta1,r1]=cart2pol(x1,y1);
% orients sensor wrt longitude
for i=1:n+1
    t = (2*pi*(i-1))/n;
    r2(i)=r1(i);
    theta2(i)=theta1(i)+pi*Long/180;
end

[x2,y2]=pol2cart(theta2,r2);
z2=z1;

x=x2;
y=y2;
z=z2;

tm=(theta)/n;

return
```

TESTHORIZON (determines azimuth and elevation angles between two objects):

```
function [Az, El] = testhorizon(v1,v2)
ER = 6371.;
% v1 sensor
% v2 satellite

[Lat1,Long1,r1] = XYZtoLatLong(v1(1),v1(2),v1(3));
[Lat2,Long2,r2] = XYZtoLatLong(v2(1),v2(2),v2(3));

% DETERMINE AZIMUTH BETWEEN POINTS

    Az1 = -atand( (sind(Long1-Long2)*cosd(Lat2)) / (cosd(Lat1)*sind(Lat2)-
sind(Lat1)*cosd(Lat2)*cosd(Long1-Long2)));
    %origAz=Az1;

    if (( (Long1-Long2 < 0.)&&(Az1<0.)) || ((Long1-Long2 >=0.) &&
(Az1>=0.)))
        Az2=Az1+180.;
    else
        Az2=Az1;
    end

if (Az2<0.)
    Az=Az2+360.;
else
    if (Az2>360.)
        Az=Az2-360.;
    else
        Az=Az2;
    end
end


% DETERMINE ALTITUDE BETWEEN POINTS
% Express altitude of satellite in terms of Earth radii
nu = r2/ER;

C = dot(v1,v2);
A = norm(v1);
B = norm(v2);

angle=acosd(C/(A*B));
Zeta = atand(sind(angle)/(cosd(angle)-(1/nu)));

El=90.-Zeta;

return
```

TESTORBITSFORCRASH (determines if an orbital path intersects with the Earth):

```
function[numcrash]= testorbitsforcrash(orbit,numorbits,n)

ER = 6371;
numcrash = 0;

for j=1:numorbits

    numcrashpts = 0;

    for i=1:n
        [theta,phi,r]=cart2sph(orbit(1,i,j), orbit(2,i,j), orbit(3,i,j));
        if (r < ER)
            numcrashpts=numcrashpts+1;
        end
    end

    if (numcrashpts > 0)
        numcrash = numcrash + 1;
    end
end

return
```

XYZtoLatLong (Transforms Cartesian coordinates to LAT LONG):

```
function [Lat,Long,r] = XYZtoLatLong(x,y,z)

[u,w,r]=cart2sph(x,y,z);

Lat=w*180/pi;
Long=u*180/pi;

return
```

SAMPLE OBJECT DATA

Type	Name	Catalog No	Inclination	Right Ascension of Ascending Node	Eccentricity with assumed leading decimal	Argument of the Perigee	Mean Anomaly	Revolutions per Day (Mean Motion)
science	ALOUETTE 1 (S-27)	62049A	80.4648	251.8838	0.0023025	349.5556	10.5107	13.68715984
visual	ATLAS CENTAUR 2	63047A	30.3596	178.3192	0.0617140	232.2213	122.0908	13.95019150
visual	THOR AGENA D R/B	64002A	99.1179	213.9160	0.0034184	329.0696	30.8477	14.31262047
visual	SL-3 R/B	64053B	65.0783	114.7148	0.0067687	182.9541	177.1170	14.58087698
radar	CALSPHERE 1	64063C	90.1618	327.4639	0.0029040	159.9719	200.2584	13.70471856
radar	CALSPHERE 2	64063E	90.1601	330.4444	0.0017390	174.6886	185.4445	13.52465796
amateur	OSCAR 3 (OSCAR III)	65016F	70.0729	120.2392	0.0017634	144.3028	215.9259	14.04716330
radar	LCS 1	65034C	32.1390	105.2034	0.0006389	67.7772	292.3399	9.89276733
radar	TEMPSAT 1	65065E	89.8120	299.9671	0.0069665	338.5007	21.3201	13.33258554
radar	CALSPHERE 4(A)	65065H	90.1870	83.6638	0.0068244	264.5683	94.7659	13.35316532
engineering	ATS 1	66110A	7.8760	310.5213	0.0004022	194.0313	166.1092	1.00287763
visual	SL-8 R/B	67045B	74.0096	157.3332	0.0068430	188.9104	171.0837	14.42461491
nnss	TRANSIT 16	67048A	89.6518	202.8668	0.0019166	167.5588	192.6038	13.49715187
radar	OPS 5712 (P/L 160)	67053A	69.9333	114.5810	0.0005473	122.7943	237.3730	14.38270313
radar	OPS 5712 (P/L 153)	67053H	69.9725	42.6828	0.0009334	293.9270	66.0858	13.95924200
radar	SURCAL 150B	67053J	69.9562	290.1594	0.0005144	288.6539	71.4032	14.46484256
nnss	TRANSIT 17	67092A	89.2601	121.1666	0.0049501	6.4027	353.7746	13.52425083
engineering	ATS 3	67111A	9.4010	317.9042	0.0014372	41.1997	319.0219	1.00272521
nnss	TRANSIT 18	68012A	89.9811	286.6698	0.0073208	258.1256	101.1673	13.50527125
visual	SL-8 R/B	68040B	74.0371	254.7099	0.0034916	255.7323	103.9981	14.84217194
visual	OAO 2	68110A	34.9967	319.3812	0.0004864	106.3304	253.7916	14.44867250
visual	ISIS 1	69009A	88.4306	109.3396	0.1714745	116.8385	261.8099	11.28858762
visual	METEOR 1-1	69029A	81.1640	18.4029	0.0008460	142.3132	217.8713	15.48132716
amateur	OSCAR 5 (AO-5)	70008B	102.1298	203.1031	0.0027614	296.4760	63.3475	12.52156866
visual	SERT 2	70009A	99.2111	63.5865	0.0004101	262.3381	97.7293	13.58227505
nnss	NNSS 19	70067A	89.8661	311.4022	0.0173749	17.3789	343.3202	13.50136236
noaa	NOAA 1 [-]	70106A	102.0767	182.6714	0.0032254	32.8711	327.4357	12.53929774
visual	SL-3 R/B	70113B	81.1481	196.2695	0.0041901	128.2787	232.2263	15.13601856
visual	SL-3 R/B	71028B	81.2357	210.1171	0.0049862	137.2753	223.2354	15.02304389
radar	RIGIDSPHERE 2 (LCS 4)	71067E	87.6210	243.0759	0.0068264	98.6101	262.2848	14.31160538
visual	ASTEX 1	71089A	92.7088	350.8867	0.0018584	39.6318	320.6234	14.46204194
visual	SL-8 R/B	71119B	73.9034	113.7181	0.0805298	196.1176	161.2908	13.76817803
visual	COSMOS 482 DESCENT CRAF	72023E	52.0992	348.7405	0.2209595	158.3818	212.7216	11.17121809
visual	OAO 3 (COPERNICUS)	72065A	35.0093	290.9770	0.0006974	227.7892	132.2219	14.56569021
visual	ATLAS CENTAUR R/B	72065B	35.0071	331.7782	0.0040162	64.5278	295.9586	14.67976629
radar	RADCAT	72076A	98.5934	203.0972	0.0001841	276.6044	83.4994	15.28915313
noaa	NOAA 2 [-]	72082A	101.4023	208.8402	0.0003289	237.7502	122.3252	12.53003216
amateur	OSCAR 6 (AO-6)	72082B	101.3948	203.3712	0.0003671	203.4751	156.6155	12.53077661
nnss	NNSS O-20	73081A	89.8572	357.7087	0.0160190	39.0686	322.1914	13.69753461
noaa	NOAA 3 [-]	73086A	101.7248	213.3275	0.0006657	125.5673	234.6013	12.40298054
visual	SL-8 R/B	73107B	73.9559	205.1120	0.0428516	63.4670	300.9743	14.68035739
visual	SL-8 R/B	74044B	82.8815	58.6073	0.0218975	356.1483	3.8029	15.24923796
noaa	NOAA 4 [-]	74089A	101.4400	215.4384	0.0008712	271.3318	88.6757	12.53052965
amateur	OSCAR 7 (AO-7)	74089B	101.4247	215.2923	0.0011806	221.0439	138.9749	12.53575894
geodetic	STARLETTE	75010A	49.8248	37.3686	0.0205926	151.0617	210.1880	13.82271245
visual	DELTA 1 R/B	75072B	89.1518	336.0515	0.0946567	192.1329	165.5581	13.68710298
goes	GOES 1 [-]	75100A	14.3596	347.6531	0.0002587	247.0011	113.0042	1.00286170
visual	SL-8 R/B	75112B	74.0606	28.2981	0.0016716	206.0398	153.9926	14.35884543
geo	LES 9	76023B	10.9221	147.7958	0.0023770	324.8980	34.7519	1.00267179