TRACKING OF SPACE DEBRIS FROM PUBLICLY AVAILABLE DATA

Philip Hoddinott

Submitted in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Approved by:
Kurt Anderson, Chair
John Christian
Matthew Oehlschlaeger

Department of Mechanical, Aerospace, and Nuclear Engineering

Rensselaer Polytechnic Institute

Troy, New York

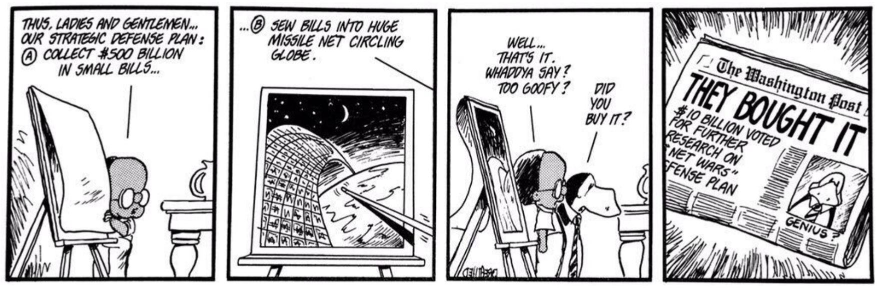
November 2018

October 24, 2018

Contents

But, Captain, I cannot change the laws of physics

-Lt. Commander Scott (Scotty)
USS *Enterprise*



List of Tables

List of Figures

0.1 Acknowledgments

This paper would never have been possible with out the help of so many people. I am indebted to my parents and my brother for their support.

John Benke for his assitance with Github and PHP. I am thankfull to Paul McKee for his assistance with Latex. Thank lots of people here

Mom, dad, David, P Anderson John B for git and PHP Paul McKee for latex

0.2 Abstract

The purpose of this project is to acquire information about earth orbiting debris, turn the information into a usable format, and develop a targeting algorithm for a debris removal satellite. The information is things such as orbital elements, debris size, and sping if i can find htis.

Should this be accomplished in a timely manner more work will be done on the orbital dynamics of OSCAR getting to said debris.

0.3 Introduction

Before the methods of this project are explored, the background of the problem must first be presented. Space debris and the threat they pose will be explained, as well as the proposed solution.

0.3.1 Space Debris

March 17th, 1958 the Vanguard 1 satellite was launched. It was the fourth man made satellite. Despite communication being lost in the 1960s, it is still in orbit to this day[6].

Figure 1: Artist's rendition of Vanguard 1 in orbit.

While spacecraft are supposed to be moved to a graveyard orbit, not all are. These large objects would be catastrophic if they collided with another object in orbit. However large objects do not pose the greatest threat. Their size makes them easy to track and there are comparatively little of them.

The debris that poses the greatest threat are the smaller pieces. Their size makes them harder to track and predict trajectories. And they are numerous. As of 2013 it is estimated

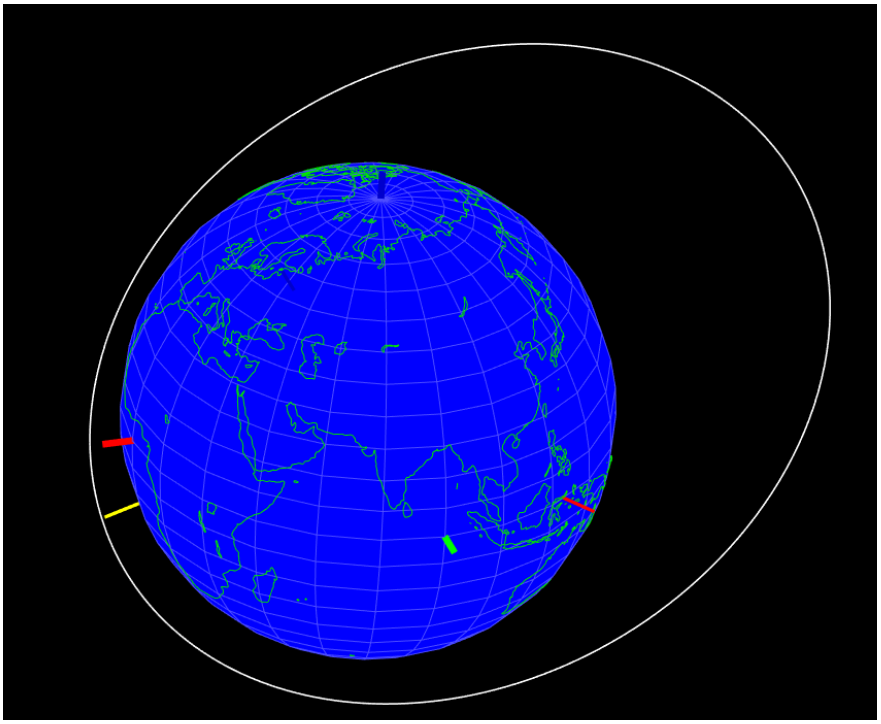


Figure 2: Vanguard's Orbit as of 10/17/2018 UPDATE THIS

that there are:

- 29,000 Objects larger than 10 cm.
- 670,000 Objects larger than 1 cm.
- Over 170,000,000 Objects larger than 1 mm.

These objects can do grievous harm to people or objects in space. A 10 cm object can cause the destruction of a satellite, a 1 cm object could penetrate the ISS's shields putting lives at risk and a 1 mm object could destroy critical subsystems.[7] The impact of one of these objects is shown in figure 3.

0.3.2 Kessler Syndrome

More than just harm to individual spacecraft or persons there is the threat of Kessler Syndrome. Also known as the Kessler effect or collision cascading, this is a scenario where there are enough objects in low earth orbit that a collision between objects will cause a chain reaction creating more and more space debris. This could result in space being inaccessible for

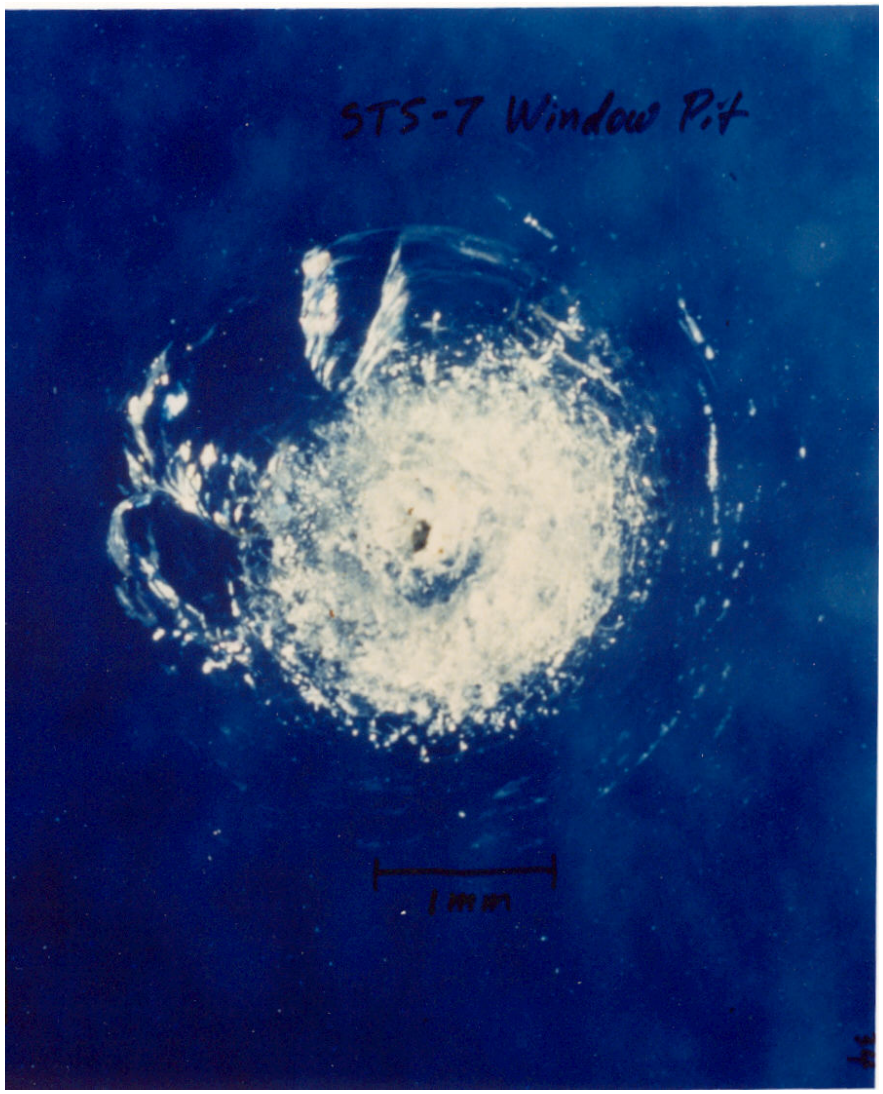


Figure 3: An impact crater on one of the windows of the Space Shuttle Challenger following a collision with a micrometeoroid[3]

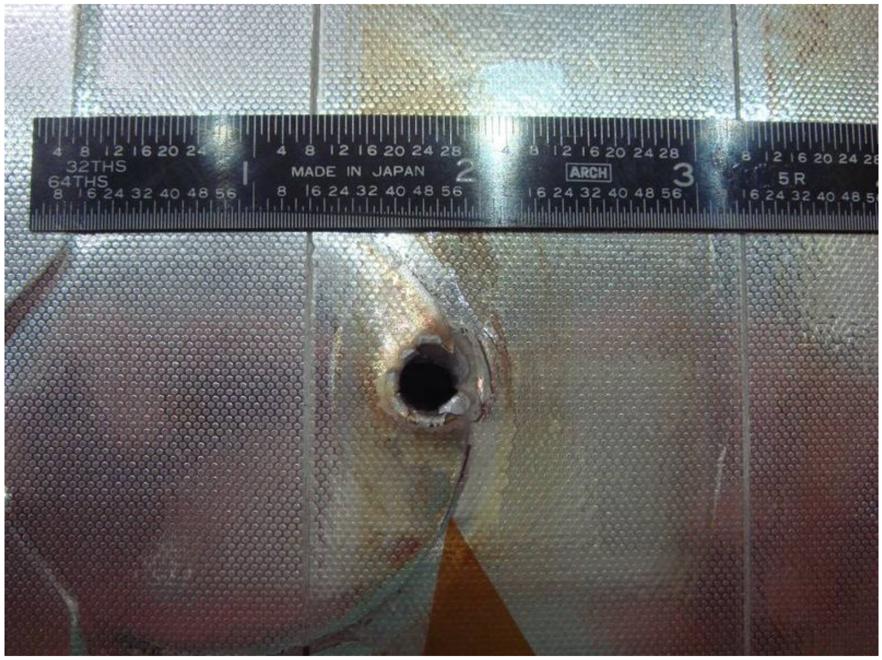


Figure 4: Debris impact on Space Shuttle Endeavorer's radiator panel[4]

years to come, as any object that left the earth's atmosphere would be immediately shredded by debris (thus adding even more debris).

This can be prevented by moving derelict spacecraft to graveyard orbits. However even spacecraft in graveyard orbits are not perfectly safe. Coolant tanks can puncture and coolant droplets freeze adding to the debris.

The alternative is to lower the orbit until it experiences atmospheric effects. But if the debris's orbit keeps it out of the atmosphere, then the debris must be moved.

0.3.3 CubeSats

For the task of moving debris, a CubeSat will be used. CubeSats are a type of miniaturized satellite. CubeSats get their name from their structure, they are made up of one or more $10 \times 10 \times 10$ cm units. These units have a maximum mass of 1.33 kilograms. First launched in 1998, as of August 2018, there have been 875 CubeSats launched[8].

CubeSats are used for projects that are too risky for a larger more expensive satellite, often demonstrating new technologies. They can take on larger risks due to their low cost.

As such CubeSats are common for experiential missions such as this one.

0.3.4 OSCAR

There is a CubeSat currently being designed by Rensselaer Polytechnic Institute (RPI). This CubeSat's goal is deorbit space debris by use of a magnetic tether.

This CubeSat is called O.S.C.A.R, which is short for Obsolete Satellite Capture and Recovery. It should be noted that this name is only temporary and will likely change in the near future. However for the purposes of this report, the RPI CubeSat will be referred to as OSCAR.

OSCAR will be a three unit $(10 \times 10 \times 30)$ CubeSat with a mass of four kilograms. It will be able to remove four pieces of space debris.[9]

For OSCAR to be able to deorbit space debris, it must know where the space debris is.

0.3.5 Orbital Elements

This thesis is concerned with the acquisition of current orbital data of space debris and it's transformation into useful formats.

First orbital elements should be defined. Orbital elements are the parameters that define an orbit. The traditional Keplerian elements found in many textbooks are as follows:

$$
\begin{array}{rc}
h & \text{specific angular momentum} \\
i & \text{inclination} \\
\Omega & \text{right ascension (RA) of the ascending node} \\
e & \text{eccentricity} \\
\omega & \text{argument of perigee} \\
\theta & \text{True anomaly}
\end{array}
\tag{1}
$$

They may be seen in figure 5.

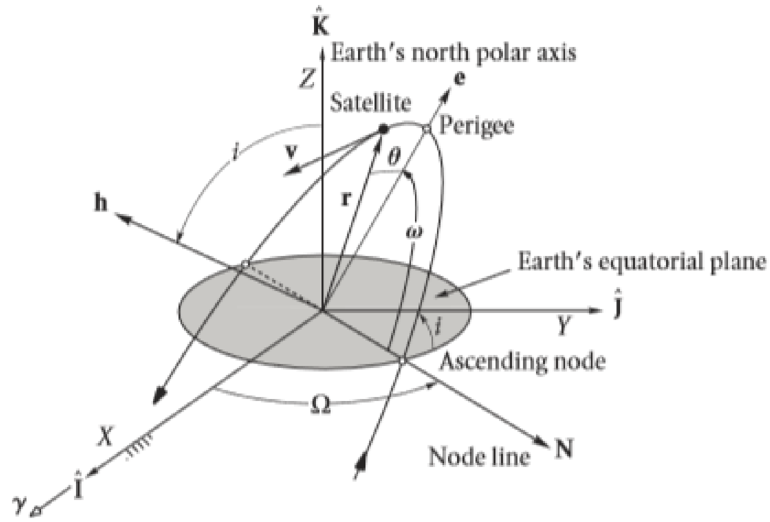


Figure 5: Geocentric equatorial frame and the orbital elements[5].

However for space debris the following set of orbital elements is slightly more useful.

$$\begin{array}{cc} e & \text{eccentricity} \\ a & \text{semimajor axis} \\ i & \text{inclination} \\ \Omega & \text{right ascension (RA) of the ascending node} \\ \omega & \text{argument of perigee} \\ \nu & \text{Mean anomaly} \end{array} \tag{2}$$

This is preferable as the semi major axis is used in conjunction with eccentricity with it's relation to perigee and apogee. However this is not a hard rule, and the code could be altered to use the first set of orbital elements. The orbital elements are better demonstrated in figure 6 and figure 7

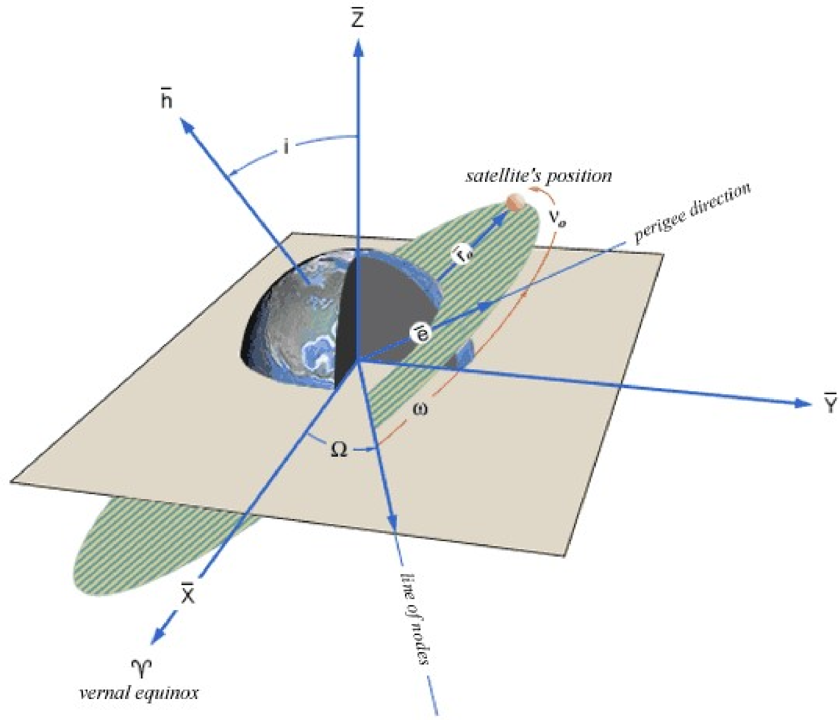


Figure 6: Diagram of Earth and Orbital Elements as well as Cartesian State Vectors[1]

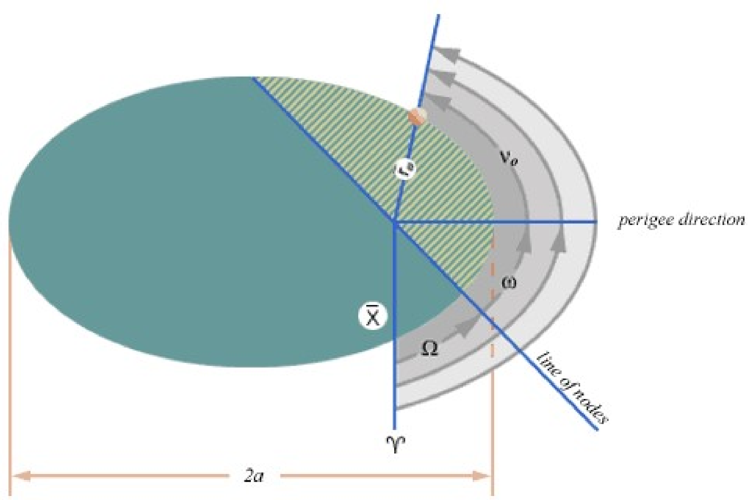


Figure 7: Diagram of Earth and Orbital Elements[1]

Table 1: Orbital Elements and their symbols[1]

Orbital Elements:		
Semi-major axis	a	Defines the size of the orbit.
Eccentricity	e	Defines the shape of the orbit.
Inclination	i	Defines the orientation of the orbit with respect to the Earth's equator.
Argument of Perigee	ω	Defines where the low point, perigee, of the orbit is with respect to the Earth's surface.
Right Ascension of the Ascending Node	Ω	Defines the location of the ascending and descending orbit locations with respect to the Earth's equatorial plane.
True/Mean Anomaly	ν	Defines where the satellite is within the orbit with respect to perigee.

0.4 Data Sources and Formats

First data sources for space debris must be found, and the data formats must be understood. The most common data format for orbital elements is the Two Line Element Set (TLE). Despite the large number of objects in space, real time data is surprisingly hard to find. While there are various websites and programs to track the location of objects such as the International Space Station, debris from old spacecraft is not quite as popular. The website space-track.org publishes TLEs for public use. First the TLE format will be explained then their acquisition will be discussed.

0.4.1 The Two Line Element Format

Orbital elements provide the means to determine a theoretical orbit. Since spacecraft are constantly experiencing forces such as atmospheric drag or solar wind their orbital elements are constantly changing.

A Two Line Element (TLE) is a data format that encodes a list of orbital elements for an Earth-orbiting object for a given point in time [Re do this]

An example shows how orbital information is derived from a Two Line Element [10].

1. **Name of Satellite:** NOAA 6

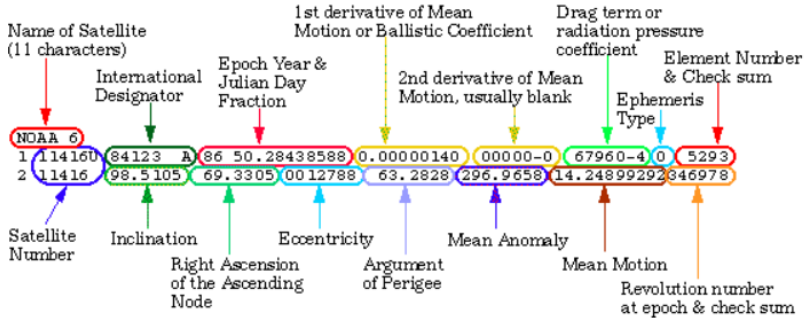


Figure 8: TLE with descriptions

This is the name of the satellite. In this example it is NOAA 6.

2. **International Designator:** 84123 A

   This number shows the last two digets of the launch year and what number launch of that year it was. In this example it is 84 123 A. The satellite was launched in the year 1984 and it was the 124th launch of the year. The A means that it was the first object that came from the launch. Check this is right

3. **Epoch Date and Julian Date Fraction:** 86 50.28438588

   The Epoch date shows the year this TLE was made and the Julian Date fraction is the number of days in said year.

4. **First Derivative of Mean Motion:** 0.00000140

   This is the daily rate of change of number of revolutions the object completes per day divided by two. This is also called the ballistic coefficient.[10] It's units are $\frac{rev}{day^2}$.

5. **Second Derivative of Mean Motion:** 00000-0

   This is the second derivative of mean motion divided by six, with units of $\frac{rev}{day^3}$.

6. **Drag term or B* Term:** 6970-4

   Of the terms given here, the B* term is the least heard of. It is a way of modeling drag on orbiting objects in propagation models. Aerodynamic drag is given by the following equation:

$$F_D = \frac{1}{2}\rho C_d A v^2 \tag{3}$$

Where A is the area, C_d is the drag coefficient, v the velocity, and ρ is the fluid density. From Newton's second law

$$F = m \times a \tag{4}$$

the accleration due to the force of drag is

$$a_D = \frac{F_D}{m} = \frac{\rho C_d A v^2}{2m} = \frac{C_d A}{m} \times \frac{\rho v^2}{2} \tag{5}$$

The ballistic coefficient is given by the following equation:

$$B = \frac{C_d A}{m} \tag{6}$$

The starred ballistic coefficient is then

$$B^* = \frac{\rho_0 B}{2} = \frac{\rho_0 C_d A}{2m} \tag{7}$$

This turns the equation for acceleration due to drag into [11][12]

$$a_D = \frac{\rho}{\rho_0} B^* v^2 \tag{8}$$

Possibly not needed maybe take out.

7. **Element Set Number and Check Sum**

8. **Satellite Number:** 11416U

This is the Satellite Catalog Number and designation of the object. A U means unclassified.

9. **Inclination (degrees):** 98.5105

   This is one of the orbital elements.

10. **Right Ascension of the Ascending Node (degrees):** 69.3305

    This is one of the orbital elements.

11. **Eccentricity:** 0012788

    This is one of the orbital elements.

12. **Argument of Perigee (degrees):** 63.2828

    This is one of the orbital elements.

13. **Mean Anomaly (degrees):** 296.9658

    This is one of the orbital elements.

14. **Mean Motion:** 14.24899292

    This is the orbits per day the object completes.

15. **Revolution Number and Check Sum:** 346978

A second example is given below. This time with references to the position of values, for extraction. The line under the dashes is the reference number line.

```
ISS (ZARYA)
1 25544U 98067A   04236.56031392  .00020137  00000-0  16538-3 0  9993
2 25544  51.6335 344.7760 0007976 126.2523 325.9359 15.70406856328906
----------------------------------------------------------------------
1234567890123456789012345678901234567890123456789012345678901234567890
1         2         3         4         5         6         7
```

Table 2 describes the second example TLE[13].

Table 2: Description of TLE

Line 0		
Columns	**Example**	**Description**
1-24	ISS (ZARYA)	The common name for the object based on information from the Satellite Catalog.
Line 1		
Columns	**Example**	**Description**
1	1	Line Number
3-7	25544	Satellite Catalog Number
8	U	Elset Classification
10-11	98	International Designator (Last two digits of launch year)
12-14	067	International Designator (Launch number of the year)
15-17	A	International Designator (Piece of the launch)
19-32	04	Epoch Year (last two digits of year)
21-32	236.56031392	Epoch (day of the year and fractional portion of the day)
34-43	.00020137	1st Derivative of the Mean Motion with respect to Time
45-52	00000-0	2nd Derivative of the Mean Motion with respect to Time (decimal point assumed)
54-61	16538-3	B* Drag Term
63	0	Element Set Type
65-68	999	Element Number
69	3	Checksum
Line 2		
Columns	**Example**	**Description**
1	2	Line Number
3-7	25544	Satellite Catalog Number
9-16	51.6335	Orbit Inclination (degrees)
18-25	344.7760	Right Ascension of Ascending Node (degrees)
27-33	0007976	Eccentricity (decimal point assumed)
35-42	126.2523	Argument of Perigee (degrees)
44-51	325.9359	Mean Anomaly (degrees)
53-63	15.70406856	Mean Motion (revolutions/day)
64-68	32890	Revolution Number at Epoch
69	6	Checksum

to do here, add more details on what the checksum and stuff like that is

0.5 NORAD Space-Track

Now we understand the format of data that is avalible (the Two-Line Element) and the format we want the data to be in (the orbital elements), the next thing to do is get the data. We want a source of TLEs for objects in space. Fortunately, there is one.

As well as tracking one S. Claus every December 24th[14] NORAD also tracks all objects currently in space. This is done through the Space Surveillance Network[15].

PUT IN THE STUFF YOU FOUND THIS MORNING

The process of TLE gathering and updating is somewhat shadowy. [16] What is known is that observations are collected multiple times per day at the Joint Space Operations Center (JSPOC) which is operated by the US Air Force Space Command (AFSPC). Then the unclassified TLEs are passed on for public release via the website space-track.org.

0.5.1 Space-Track.org

In their own words, Space-Track.org "promotes space flight safety, protection of the space environment and the peaceful use of space worldwide by sharing space situational awareness services and information with U.S. and international satellite owners/operators, academia and other entities."[17]

Information about objects in space may be found the website space-track.org. Space-track is the main source for orbital data, though some are also available from the website celestrak.com. Of the two space-track has far more information and better methods of access. "Space-Track.org is managed, maintained and administered by JFSCC"[18]. Space-track allows information to be downloaded manually from the TLE search [13] or the satellite catalog search [19]. While these are useful tools the code needs a way to automatically download data. Fortunately space-track also has an API that allows queries.

0.5.2 Space-Track Query

An application programming interface or API is a set of methods that allow simple communication between different codes. The Space-Track API allows for programs to send requests for data on objects in space. It is limited to less than 20 requests per minute and less than 200 requests per hour.

The API works by building an API similar to the following URL:

https://www.space-track.org/basicspacedata/query/class/boxscore/

Where:

- Base: URL: https://www.space-track.org/
- Request Controller: basicspacedata
- Request Action: query
- Predicate Value Pairs: class/boxscore/

The only publicly available Request Controller is basicspacedata. The expandedspacedata controller requires SSA Sharing Agreements [2]. The first two sections of the URL will never change. What will changes is the query, the class, and the boxscore. The request classes that are of interest to this thesis are in table 3.

Table 3: Request Classes of Interest[2]

Class Name	Class Description
tle	Historical record of orbital element sets (Two-Line Elements). Additional metadata columns are available so that users can filter their queries by these values, downloading only the data they need. Shown in html, csv, and xml formats, these additional columns include: object_name, object_type, semi-major axis, period, apogee, perigee, and file. NOTE: There are over 97 million TLEs in the database. The user is strongly advised to limit their API queries by OBJECT_NUMBER / NORAD_CAT_ID AND an epoch range such as ">now-30" to avoid "Query range out of bounds" errors.
satcat	Satellite Catalog Information. The "CURRENT" predicate indicates the most current catalog record with a 'Y'. All older records for that object will have an 'N'.
omm	Orbit Mean-Elements Message (OMM)Keplerian element format that complies with CCSDS Recommended Standard 502.0-B-2

Question for OGE/Professor Anderson: I am citing these tables word for word, what is the proper way to do that for a table?

The Orbit Mean-Elements Message was not used in this thesis because it could not handle large groups of requests. However because it directly returns the Keplerian elements, it may be useful if the API functionality is improved in the future. As a result the TLE and satcat classes are the only classes used in this code. The predicate values pairs may be specified by using REST Operators. The REST Operators that can be used are seen in table 4

Table 4: REST Operators[2]

Operator	Description
$>$	Greater Than (alternate is %3E)
$<$	Less Than (alternate is %3C)
$<>$	Not Equal (alternate is %3C%3E)
,	Comma Delimited 'OR' (ex. 1,2,3)
–	Inclusive Range (ex. 1–100 returns 1 and 100 and everything in between)Date ranges are expressed as YYYY-MM-DD%20HH:MM:SS–YYYY-MM-DD%20HH:MM:SS or YYYY-MM-DD–YYYY-MM-DD
null-val	Value for 'NULL', can only be used with Not Equal ($<>$) or by itself.
$\sim\sim$	"Like" or Wildcard search. You may put the $\sim\sim$before or after the text; wildcard is evaluated regardless of location of $\sim\sim$in the URL.For example, $\sim\sim$OB will return 'OBJECT 1', 'GLOBALSTAR', 'PROBA 1', etc.
ˆ	Wildcard after value with a minimum of two characters. (alternate is %5E) The wildcard is evaluated after the text regardless of location of ˆin the URL. For example, ˆOB will return 'OBJECT 1', 'OBJECT 2', etc. but not 'GLOBALSTAR'
now	Variable that contains the current system date and time. Add or subtract days (or fractions thereof) after 'now' to modify the date/time, e.g. now-7, now+14, now-6.5, now+2.3. Use $<$,$>$,and – to get a range of dates; e.g. $>$now-7, now-14–now

The REST Predicates are seen in table 5

Table 5: REST Predicates[2]

REST Predicates	REST Description
class/classname	Class of data message requested from the API. Required for all operations.
predicates/p1, p2,...	Comma-separated list of predicates to be returned as the result. Can also be set to 'all' or omitted entirely for default behavior of returning all predicates.
metadata/true	Includes data about the request (total # of records, request time, etc); ignored when used with tle, 3le, and csv formats.
limit/x(,x?)	Specifies the number of records to return. Takes an integer for an argument, with an optional second argument to specify offset. For example, /limit/10/ will return the first 10 records; /limit/10,5/ will show 10 records starting after record #5 (rows 6 through 15)
orderby/predicate (asc?—desc?)	Allows ordering by a predicate (column), either ascending or descending, with asc/desc separated from the predicate by a space. Multiple Orderings are supported, separating the pairs of predicate and asc/desc by commas (e.g. http://.../DECAY desc,APOGEE desc/...)
distinct/true	Removes duplicate rows.
format/xxxx	Specifies return format, can be: json, xml, html, csv, tle, 3le, kvn, or stream. See below for additional information. If no format is specified, the default is JSON
emptyresult/show	Queries that return no data will show 'NO RESULTS RETURNED' instead of the default blank page

Finally the data returned may be in the formats seen in table 6

Table 6: Available Data Formats[2]

Format	Request Rules
eXtensible Markup Language (xml)	No special request rules. For CDM request class, shows CCSDS-compliant format.
JavaScript Object Notation (json)	Preferred format, no special request rules.
HyperText Markup Language (html)	Not recommended for machine parsing. No special request rules.
Comma-Separated Values (csv)	Due to the limitations of the CSV specification, requesting CSV formatted data does not allow for transmission of metadata; metadata concerning the request will need to be gathered through one of the other formats prior to requesting CSV.
Two-Line Element Set (tle)	To get traditionally-formatted TLE data, request the 'tle', 'tle_latest', or 'tle_publish' class. We recommend that you omit predicates from URLs that use tle format. If you do include predicates, it must include line 1 & 2 like this: "/predicates/-TLE_LINE1,TLE_LINE2/". Like CSV, tle format ignores /metadata/true/ REST Predicate.
Three-Line Element Set (3le)	The format adds TLE_LINE0 or the "Title line", a twenty-four character name, before the traditional Two Line Element format. The 3le format defaults to include a leading zero so that you can easily find the object name via scripts. If you do not want the leading zero, you should include this in your URL to show the OBJECT_NAME instead of TLE_LINE0: "/predicates/OB-JECT_NAME,TLE_LINE1,TLE_LINE2/". Like CSV and tle, 3le format ignores /metadata/true/ REST Predicate.
Key=Value Notation (kvn)	Key=Value Notation format, currently for use exclusively with CDM Class. Incompatible with /metadata/true/.
File Stream (stream)	File Stream format, currently for use exclusively with download class.

0.5.3 Code Queries

Using the following rules, the code generates URLs that query space-track for the relevent information.

The first url that is generated is:

https://www.space-track.org/basicspacedata/query/class/satcat/OBJECT_TYPE/debris /RCS_SIZE/small/LAUNCH_YEAR/>**launchYear**/orderby/DECAY asc/format/csv/metadata/false

Where **launchYear** is replaced by the year the debris was launched in. This is the best way of controlling how much debris the user gets. Setting a launch year of 1960 will obviously included all small debris (debris that is less than 10 cm^2 cross section), while setting a launch year of 2018 would return very little debris. It is advisable to run test cases with small debris before performing operations on all available debris.

This url will return the satcat Ids as seen by the "class/satcat". It will return them in a csv format as seen by the "format/csv" and it will only select small debris, as seen by the "RCS_SIZE/small".

The second url is later used to return the TLEs from given satcat Ids.

0.6 Code Overview

The code obtains orbital data for space debris depending on what variables the user has inputted, such as the launch year of the debris or how recent the data should be. First the code ascertains what data is already available. It checks to see if there is a .mat file for the SATCAT. This is simply a file that contains the catalog ids of the debris. If this file does not exist then the code has not been run for the current parameters, and the code will be run.

The code will also check to see if there is a TLE .mat file. If this file exists the code then checks when the file was created. Depending on user variables, the code may consider the information to be out of date, will rerun everything. However if the TLE .mat file is recent, then the code will load the current debris data into the workspace.

Assuming that the code detects that it has no data, or the data is out of date the code will work in the following steps

1. The desired NORAD satellite catalog ids will be collected with the get_SATCAT.m file. These ids are determined by user variables such as launch date. The code downloads them as a csv, strips things like quotation marks away, and sorts the ids in order.
   - if a SATCAT .mat file exists and is recent this step is skipped.
2. The code then begins to loop through the array of Ids. It enters an error catch to handle url time outs.
   - Within this loop the code then downloads the TLEs from the satellite ids. The TLEs are stored in a string. It does this in groups that have their size determined by the user. A large group size may risk url timeouts, but a small group size will result in a long run time.
   - The TLEs are then parsed, the TLEs are turned into usable orbital information, and saved as an array in MATLAB.
   - When the loop has run through all the Ids, the array of data is saved as a .mat file wit the date of it's creation.
3. Now the array of debris data is ready. A copy is saved with a header listing what each collum represents. Another is sent to an orbit visualization file.
4. It should be noted that not all TLEs requested will be provided. So if the code requests TLEs of objects with a id from 1000 to 1100 and expects 100 TLEs, but some are classified, then duplicated TLEs less than 100 TLEs will be returned. The user now has a usable list of space debris and their orbital elements.

What now follows is a more in depth explanation of each file.

0.6.1 VarStore.m

This is a file where a few important variables are stored. It allows the user to only have to edit one file to change the code's operation.

0.6.2 UserPass.m

This file is where the username and password for space-track.org are stored. This is kept separate from the other code to ensure privacy of usermane and password.

0.6.3 MASTER_TLE.m

This file is the master file for the Two Line Element MATLAB files. Running it will run all of the associated MATLAB files. These MATLAB files take some time to run, so it may be convenient to alter the range of data operated on by MASTER_TLE.m in VarStore.m

0.6.4 get_SATCAT.m

Get_SATCAT.m is the MATLAB file that gets the satellite catalog numbers of all orbital debris launched after a given year and with the RCS_SIZE value equal to SMALL. The Launch Year can be set by the values given by the user in VarStore.m The default launch year is set to 1990. Note that an earlier launch year will provide more data, and thus it will take more time to process. This may cause a time out error. Should this happen the timeOutVal in VarStore.m should be adjusted to be longer.

Once the SATCAT csv file has been acquired the file then formats it. The quotation marks that are around every entry are removed. The debris that have already deorbited are also removed. Finally the debris is sorted by NORAD Catoluge Id, and saved as a .mat file. The post request in the get_SATCAT.m file was originaly developed by Joao Encarnacao[20]

0.6.5 get_TLE_from_ID_Manager.m

The purpose of get_TLE_from_ID_Manager.m is to act as a handler for function get_TLE_from_NorID.m. The code first checks to see if there exists a folder to store the data in, if there is not a folder is created. Then the code begins to run through the list of NORAD Ids. It sends these Ids to get_TLE_from_NorID.m, receives the orbital element data from those Ids.

The code also contains an error handler. The most common problem with downloading large numbers of TLEs is a url timeout may occur. Timeouts are easy for a human to deal with, they may be fixed by simply refreshing the web page. However for MATLAB, it is more difficult. The code deals with a url timeout as follows:

First the code detects the specific error thrown for url timeouts. This error is caught, so the code does not stop running. Then the code gets the current location in the Id array the time out occurred at, and restarts at that spot. The effect of this is the same as refreshing a webpage that is not loading.

Once all the data has been collected for the given Ids, it is saved as a .mat file with the time that it was created.

0.6.6 get_TLE_from_NorID.m

The purpose of get_TLE_from_NorID.m is to download the TLEs from NORAD given Ids, and to save the TLEs as useful orbital elements. This function may be divided into two parts: the web interface for TLEs and the TLE to orbital element section.

The first part of the code creates a url from the current set of Ids. This url is then used to download the TLEs. The TLEs are saved as a string within MATLAB.

In the next part of the code a loop goes through the TLE string. The information from these TLEs is saved as a matrix. Earlier vesrions of this projects code first downloaded all TLEs as .txt files and then parsed through the .txt files to get the orbital elements. However that was unnecessarily slow as MATLAB had to open each .txt file and read it's contents as opposed to reading the contents of a string already in the workspace. By constantly saving the results as a .mat file through out the operation of the code, there is no threat of data loss.

From each TLE the inclination (i), RA of ascending node (Ω), eccentricity (e), argument of perigee (ω), mean anomaly, and mean motion (n) may be acquired directly. From these values the other orbital elements may be calculated.

Period of rev:

$$T = \frac{24 \times 60 \times 60}{n} \tag{9}$$

Semi-major axis:

$$a = \left(\left(\frac{T}{2\pi}\right)^2 \times \mu_{\text{earth}}\right)^{(1/3)} \tag{10}$$

where μ_{earth} is the standard gravitational parameter of earth: 3.986×10^{14}.

Semi-minor axis:

$$b = a \times \sqrt{1 - e^2} \tag{11}$$

Perigee and Apogee:

$$r_{\text{per}} = (1 - e) \times a \tag{12}$$

$$r_{\text{ap}} = (1 + e) \times a \tag{13}$$

The code to parse the elements from the TLE was based off of Brett Pantalone's parse TLE code[21], modified to handle multiple TLEs.

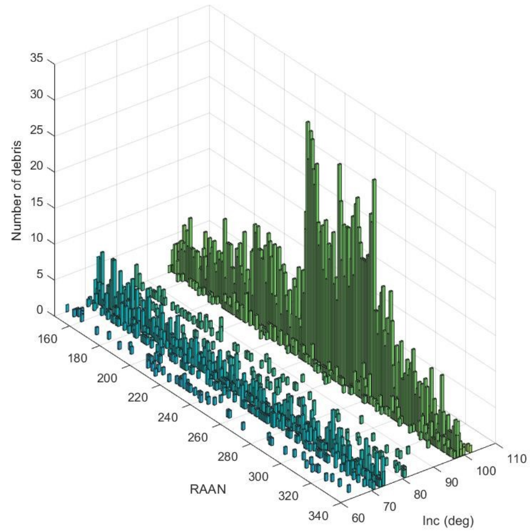


Figure 9: Debis loc

0.7 Targeting

Once the data has been collected, the next step is to target possible orbits. Since OSCAR has a delta V of Find delta V inclanation and plane changes are out of the question. What is desired is a number of debris in a plane with simialr inclanations.

0.8 Conclusion

The goal of this thesis was to create a program that can download the latest orbital elements needed to calculate the real time locations of space debris, given user inputted parameters. The information would be usable to someone who has taken a basic spaceflight mechanics class.

Bibliography

[1] NASA, 2012.

[2] Space Track. Introduction to the space track api, 2013.

[3] NASA. Space debris impact on space shuttle window.

[4] NASA. Image of the entry hole created on space shuttle endeavour's radiator panel by the impact of unknown space debris.

[5] HOWARD D. CURTIS. *ORBITAL MECHANICS FOR ENGINEERING STUDENTS*. Elsevier Butterworth Heinemann, 2013.

[6] Usnrl. Vanguard i celebrates 50 years in space, Mar 2008.

[7] How many space debris objects are currently in orbit?, Jul 2014.

[8] Erik Kulu. Nanosatellite and cubesat database.

[9] Paul McKee. Cubesat dynamics and attitude control: Kanes method, lqr, and kalman filtering. Master's thesis, Rensselaer Polytechnic Institute, Troy, NY, 2018.

[10] Amiko Kauderer Kim Dismukes. Definition of two-line element set coordinate system, 2011.

[11] By Dr. T.S. Kelso. Frequently asked questions: Two-line element set format, 1998.

[12] Canadian Satellite Tracking and Orbit Research (CASTOR). B-star drag term, 2010.

[13] Space Track. Basic description of the two line element (tle) format, 2013.

[14] NORAD. Norad tracks santa.

[15] Dr.T.S. Kelso. Space surveillance, 1997.

[16] David A Vallado and Paul J Cefola. Two-line element sets–practice and use. In *63rd International Astronautical Congress, Naples, Italy*, 2012.

[17] Space Track. Space-track.org, 2018.

[18] Space Track. Space track legend, 2018.

[19] Space Track. Space-track satellite catalog search, 2018.

[20] Joao Encarnacao. matlab-sgp4. `https://github.com/jgte/matlab-sgp4`, 2015.

[21] Brett Pantalone. Read two-line element ephemeris files. `https://www.mathworks.com/matlabcentral/fileexchange/56904-read-two-line-element-ephemeris-files`, 2016.

Appendix 1 - MATLAB code

0.8.1 Master_TLE.m

```matlab
%% Master Program
% By Philip Hoddinott
% This is the Master program to aquire TLEs and turn them into usable
% keplerian elements.
%% Setup
close all; clear all; clc; % clear workspace
%% get data
VarStore % run var store for stored variables, ugly but it works


%% check for existing data
strNam = ['mat_files/TLE_',num2str(launchYear),'.mat']; % get strNam
strNam_SC = ['mat_files/SATCAT_',num2str(launchYear),'.mat']; % get strNam

try % check the satcat tog;
    load(strNam_SC);
    tog_SC=0;

    try % try for TLE file
        load(strNam, 'tle_final','dateCreated'); % load in file
        cTime =datetime;
        if cTime>(dateCreated+calweeks(1)) % file out of date, rerun not SATCAT
            fprintf('File is one week out of date, auto running program\n');
            tog_All=1;
            tog_SC=0;
        else % file recent, no need to run any
            fprintf('The file %s, was created within the last week\n',strNam);
            tog_All=0;
            tog_SC=0;
        end
    catch ME % if tle not found or no date run all
        switch ME.identifier
            case 'MATLAB:load:couldNotReadFile'
                warning('TLE File does not exist, auto running program');
```

```
            case 'MATLAB:UndefinedFunction'
                warning('dateCreated not found, auto running program');
        end
        tog_All=1;
        tog_SC=0;
    end

catch ME
    switch ME.identifier
        case 'MATLAB:load:couldNotReadFile'
            warning('SATCAT File does not exist, auto running program');
            tog_SC=1;
            tog_All=1;
    end
end
% assign values
get_SATCAT_tog =tog_SC; % toggle for get_SATCAT 1 = run, 0 = don't run
get_Multiple_TLE_from_Id_tog =tog_All;
readTLE_txt_tog=tog_All;
check_TLE_Edit_TLE_tog=tog_All;



%% func get_SATCAT
% Function to get a .mat file from the SATCAT

if get_SATCAT_tog==1 % if the satcat file is out of date, run this
    get_SATCAT % get SATCAT, comment out if already run
    fprintf('get_SATCAT.m has finished running\n'); % output SATCAT has run
else % if the satcat file is recent then no nead to run get_SATCAT
    load(strNam_SC,'all_TLE','decayEnd'); % load mat of SATCAT
    fprintf('get_SATCAT.m was not run\n'); % output SATCAT was not run
end

relDeb=str2num(char(all_TLE(2:decayEnd,2))); % get NORAD CAT ID

%% func get_Multiple_TLE_from_Id
% Function to get tle txt files from the norat_cat_ids in the SATCAT
VarStore % run var store for stored variables, ugly but it works
```

```
if get_Multiple_TLE_from_Id_tog==1
    get_TLE_from_ID_Manager
    fprintf('get_Multiple_TLE_from_Id.m has finished running\n');
else
     fprintf('get_Multiple_TLE_from_Id.m was not run\n');
end

%close all; clear all; % clear out everything
VarStore % run var store for stored variables, ugly but it works
strNam = ['mat_files/TLE_',num2str(launchYear),'.mat']; % get strNam

load(strNam, 'tle_final')

%tle_low=sortrows(tle_final(:,:),11);
%save('Orbits_MOD_1/tle_low2high.mat','tle_low');
%{
tle_high=sortrows(tle_final(:,:),11,'descend');
save('Orbits_MOD_1/tle_high2low.mat','tle_high');
% Note that these files could be made functions in MATLAB. For debuggin
% purposes they currently are not

tle_high=sortrows(tle_final(:,:),4,'descend');
save('Orbits_MOD_1/tle_RANN.mat','tle_high');

tle_INC=sortrows(tle_final(:,:),3,'descend');
save('Orbits_MOD_1/tle_INC.mat','tle_INC');
%}
tle_view=tle_final;
tle_view_temp=["norad_cat_id","Epoch time","Inclination (deg)","RAAN (deg)","Eccentricity (deg)","Arg of perigee(deg)","Mean anomaly (deg)","Mean motion (rev/day)","Period of rev (s/rev)","Semi-major axis (meter)","Semi-minor axis (meter)"];

tle_veiw = [tle_view_temp;tle_view]; % useful for looking at numbers
```

0.8.2 get_SATCAT.m

```
%% get_SATCAT.m
% By Philip Hoddinott
% This code gets SATCAT from NORARD Query via post requests.
% This code was based off of the following code: https://github.
```

```
com/jgte/matlab-sgp4/blob/master/get_tle.m

load('UserPass.mat') % load in username and password

URL='https://www.space-track.org/ajaxauth/login';

post={... % Create Post (rember to referance the github)
  'identity',username, 'password',password,...
  'query',[...
    'https://www.space-track.org/basicspacedata/query/class/satcat
       /OBJECT_TYPE/debris/',...
    'RCS_SIZE/small/LAUNCH_YEAR/>',num2str(launchYear),'/'...
    'orderby/DECAY asc/format/csv/metadata/false']...
};


out=urlread(URL,'Post',post,'Timeout',timeOutVal); % gets the
   output
outStr=convertCharsToStrings(out); % coverts output to string


newStr = strsplit(outStr,["\n"]); % split string by line break
for i=1:length(newStr) % split string by commas
    all_TLE(i,:) = strsplit(newStr(i),',');
end


[m,n] = size(all_TLE); % get size of matrix
%% Remove " marks
for i=1:m % remove the " marks
    for j=1:n
        all_TLE(i,j)=strip(all_TLE(i,j),'left','"');
        all_TLE(i,j)=strip(all_TLE(i,j),'right','"');
    end
end

%% find where the last decayed item is and remove it
decay_loc=8;

for i=1:length(all_TLE(1,:)) % find the columm decay is located in
   , it should be in 8
    if all_TLE(1,i)=="DECAY"
        decay_loc = i; % save decay loc
    end
end
```

```


decayEnd=m+10;
for i=2:length(newStr) % find the columm decay is located in, it
    should be in 8
    if all_TLE(i,decay_loc)~=[""] && decayEnd==m+10
        decayEnd=i-1; % get the last location where there is no
            yet decay
    end
end
all_TLE=all_TLE(1:decayEnd,:); % trim to only have debris that has
    not yet decayed


all_TLES=sortrows(all_TLE(2:end,:),2); % sort rows by NORAD ID
all_TLE=[all_TLE(1,:);all_TLES]; % save the sorted row by NORAD ID

%% save to a file
strNam = ['mat_files/SATCAT_',num2str(launchYear),'.mat'];
save(strNam,'all_TLE','decayEnd');
```

0.8.3 get_TLE_from_ID_Manager.m

```
%% get_TLE_from_ID_Manager
% By Philip Hoddinott
% This code handles getMultiple_TLE_from_ID and deals with url
    read time outs. urlread is an older function, however it works
    best with post requets.


load('UserPass.mat') % load in username and password
if exist(tle_folder)~=7 % if the folder does not exist it will
    make it
    mkdir(tle_folder)% tle_text_files % note this will give a
        warning if folder already exists
end
tle_final=[];
strNam = ['mat_files/New_TLE_',num2str(launchYear),'.mat']; % save
    the TLE as a .mat
save(strNam,'tle_final');
tleA = 1:tle_inc:decayEnd;
tleA=[tleA,decayEnd];

jStart =1; % starting value, will get reset if the connection
    times out
```

```
jEnd=length(tleA)-1; % end val

j_GMTFI=jStart;

while j_GMTFI~=jEnd % ensures that try catch keeps running untill
    it has gone all the way though
    fprintf('While loop, j = %d, jE = %d\n',j_GMTFI,jEnd);
    try
        for j_GMTFI = jStart:jEnd % try normal loop
            fprintf('Try for, j = %d, jE = %d\n',j_GMTFI,jEnd);
            load(strNam,'tle_final');
            tle_current=tle_final;
            get_TLE_from_NorID
            tle_final=[tle_current;tle_final];
            save(strNam,'tle_final'); % save the tle every loop
                through
        end

    catch ME      % time out catch
        fprintf('catch ME, j = %d, jE = %d\n',j_GMTFI,jEnd);
       switch ME.identifier
            case 'MATLAB:urlread:Timeout'
                warning('connection timed out at j = %d, trying
                    again',j_GMTFI);
                jStart=j_GMTFI;
       end
    end

end
load(strNam,'tle_final');
[C,ia,ic]=unique(tle_final(:,1),'rows'); % sort by row by norard
    cat id
tle_final_N = tle_final(ia,:);
dateCreated=datetime;
save(strNam,'tle_final_N','dateCreated');
```

0.8.4 get_TLE_from_NorID.m

```
%% get_TLE_from_NorID.m
% By Philip Hoddinott
% This code gets the TLEs from NORARD IDs and then extracts the
    keplerian elements from the TLEs. This code is orginally based
    off of this code: https://github.com/jgte/matlab-sgp4/blob/
    master/get_tle.m

```

```
%% Create string for post request
strTLE='/';
for i = tleA(j_GMTFI):tleA(j_GMTFI+1)-2
    strTLE=[strTLE, num2str(relDeb(i)),','];
end
if tleA(j_GMTFI+1)==decayEnd
    strTLE=[strTLE, num2str(relDeb(tleA(j_GMTFI+1)-1)),num2str(relDeb(tleA(j_GMTFI))),'/'];
    fnName = [tle_folder,'/tle_',num2str(j_GMTFI),'_',num2str(tleA(j_GMTFI)),'_',num2str(tleA(j_GMTFI+1)),'.txt'];
else
    strTLE=[strTLE, num2str(relDeb(tleA(j_GMTFI+1)-1)),'/'];
    fnName = [tle_folder,'/tle_',num2str(j_GMTFI),'_',num2str(tleA(j_GMTFI)),'_',num2str(tleA(j_GMTFI+1)-1),'.txt'];
end
fprintf('j = %d, tle %d to tle %d',j_GMTFI,tleA(j_GMTFI), tleA(j_GMTFI+1));
c=clock;
fprintf(' H = %d, Min = %d, Sec = %.1f\n',c(4),c(5),c(6));
%fprintf(strTLE);fprintf('\n'); %uncomment if you need to see %NORAD_CAT_ID
strTLE=[strTLE,'orderby/ORDINAL%20asc/limit/',num2str(tle_inc),'/format/tle/metadata/false'];


URL='https://www.space-track.org/ajaxauth/login';

post={'identity',username,'password',password,...
   'query',[...
     'https://www.space-track.org/basicspacedata/query/class/tle_latest/NORAD_CAT_ID',...
     strTLE
   ]...
};

out3TLE=urlread(URL,'Post',post,'Timeout',timeOutVal); % gets the output
outStr=convertCharsToStrings(out3TLE); % coverts output to string


j=1;
k=1;
C = splitlines(outStr);
while j<length(C)-1
    LineOne=convertStringsToChars(C(j,1)); % get first line
```

```
    LineTwo=convertStringsToChars(C(j+1,1)); % get second line
    satnum = str2num(LineOne(3:7)); % get sat num
    Incl = str2num(LineTwo(9:16)); % get inc
    Omega = str2num(LineTwo(18:25)); % get
    ecc = str2num(['.' LineTwo(27:33)]);
    w = str2num(LineTwo(35:42));
    M = str2num(LineTwo(44:51));
    n = str2num(LineTwo(53:63));
    T = 86400/n;
    a = ((T/(2*pi))^2*398.6e12)^(1/3);
    b = a*sqrt(1-ecc^2);
    % store in array
    tle_stor(k,1)=satnum;
    tle_stor(k,2)=str2num(LineOne(19:32));
    tle_stor(k,3)=Incl;
    tle_stor(k,4)=Omega;
    tle_stor(k,5)=ecc;
    tle_stor(k,6)=w;
    tle_stor(k,7)=M;
    tle_stor(k,8)=n;
    tle_stor(k,9)=T;
    tle_stor(k,10)=a;
    tle_stor(k,11)=b;

    j=j+2;
    k=k+1;
end
tle_final=tle_stor;
```