HIGH SCHOOL EDITION
Journal of Student Research

Volume 12 Issue 3 (2023)

Implementing Automatic Debris Avoidance Program

Akshan Mohaney

Pace Junior Science College, India

ABSTRACT

The problem of space debris from human sources is increasing exponentially and satellites are at a constant risk of encountering them while moving in orbit. The current process of debris classification and avoidance is tedious and is done manually and soon with the ever-increasing debris problem, it is not going to take us a long way. Thus, the main question now is how satellites can use computer programming efficiently to classify incoming debris (as potentially harmful or not) and change/move their orbit to avoid incoming debris in space without any manual effort? To start thinking on this topic, reading a few sources (listed in reference) already published on this topic was necessary. Using the gathered information, I framed an algorithm on how the debris can be classified as potentially harmful or not and actions to be taken depending on the situation. I also framed a program as a kind of a blueprint to explain the logic on which this program can be written and built upon in the future. To do this I enriched my knowledge of 3D geometry by reading various articles. Numerous tests were done on the program and based on those I kept on correcting the main program and adding more detailed cases thus, leading to the final program.

Introduction

There has been a vast development in the field of satellites and space ever since *Sputnik - 1*, the first satellite to be ever launched, was put into orbit. Rapid development in the space field gave rise to an increased number of rockets launches and satellites and other manmade objects in space which thus, resulted in increased space waste but what is this space waste and why does it necessarily concern us? The debris left in space during missions like rocket boosters, explosive bolts which fragment into pieces (which are one time usable), retired satellites, paint chips, cameras, screws etc. Which to revolve around the earth in orbits are called space debris

Orbits slowly decay overtime, and the debris reaches the Earth but this process takes a lot of time. According to NASA **[3]**, debris left in orbits below 370 miles (600 km) normally fall back to Earth within several years. At altitudes of 500 miles (800 km), the time for orbital decay is often measured in decades. Above 620 miles (1,000 km), orbital debris normally will continue circling Earth for a century or more and with the increase in the number of debris, efficient avoidance of debris is extremely important for satellites moving in space. There is a high chance that these debris collide with satellite and damage expensive equipment. They can tear through solar panels of the satellites or can even damage computers onboard or maybe a large piece can even destroy a satellite completely if not avoided. As of 2021, the **United States Space Surveillance Network** was tracking more than 15,000 pieces of space debris larger than 10 cm (4 inches) across. It is estimated that there are about 200,000 pieces between 1 and 10 cm (0.4 and 4 inches) across and that there could be millions of pieces smaller than 1 cm **[5]**. Currently the process of debris avoidance is extremely stressful and constant monitoring of the satellite and incoming debris has to be done and manual orbit adjustment has to be done. The objects larger than 10 cm can be easily tracked and can be avoided but the small ones are difficult to track(however). Soon, with the increasing number of debris, keeping track of these debris will become difficult manually and the satellite will somehow have to manage dodging the debris by themselves. Space debris can be added due to accidents as well. For example, on 10th February 2009 , an active commercial satellite *Iridium-33* and *derelict Russian military Kosmos 2251* collided with each other causing 1000s of debris to be scattered in space**[6]**. Space debris have caused damage in the past and if we don't avoid them, another collision is likely to occur soon.

In this research paper, I have tried to explain a basic program which I have framed that satellites can use to avoid single incoming debris automatically. I have explained the design of the program which consists of the algorithm on which I coded it, each case of the program with its respective outputs and different assumptions to be considered while using this program and their derivations. Some articles have also been cited which helped me with the programming. Along with that I have also used graphs from **GeoGebra.com** to explain how the different cases would look like in 3D space and to check the validity of the cases.

Design

❖ Assumptions/Approximations to be considered while using the program:

1. We assume that the satellite's position, velocity, and orbit equation is known and it can measure the relative position and velocity of incoming debris using onboard equipment available.
   For relative velocity, we consider the satellite to be of fixed origin. The x axis is sideways, y axis upwards and z axis forward.

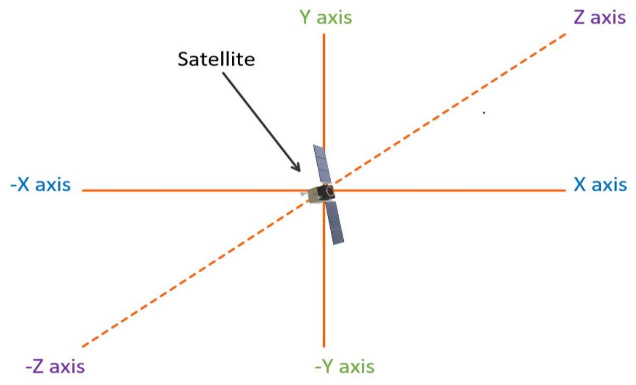


Fig 1: Displays the axes for the satellite's relative coordinates.

2. We assume the center of earth as the center of the absolute coordinate system since the center doesn't rotate.
3. We consider all objects detected to fall within measurable parameters **(i.e., more than 10cm wide)**
4. We assume a single debris-satellite system. **(Most common)**
5. We assume the satellite to be moving in a straight-line path for a few kilometers. We consider local linearization till the point the error between the straight line assumed and the elliptical path becomes 5% beyond which we will have to switch to a different line. The linearization problem mainly arises at the curved path of the ellipse where chances of curving become more. Let us thus, calculate what is the minimum distance for which local linearization is valid and can be safely assumed.

Let us consider a ***Molniya orbit*** in a particular plane (it can be assumed 2D and let us consider the x-y coordinate system:

- ***A Molniya orbit is an orbit followed by Soviet Satellites which have maximum eccentricity for an Earth orbiting satellite [6]***.

Let us consider a vertical tangent point on the positive x-axis. Let c be the distance between satellite and vertical tangent at the point.

Point P≡ ($a\cos\theta$,$b\sin\theta$)according to the parametric form. Thus, $c = a - a\cos\theta$.

For Molniya orbit,
$a\ (semi\ major\ axis\ length) = 2R_e$
$b\ (semi\ minor\ axis\ length) = R_e \quad (\boldsymbol{R_e = radius\ of\ Earth \approx 6000\ km})$

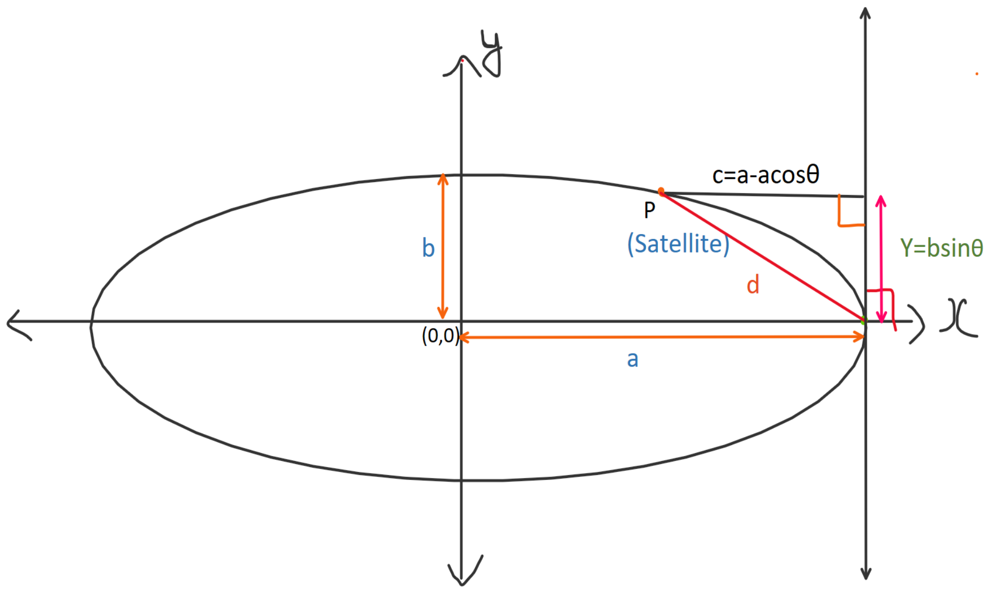


Fig 2: Elliptical coordinates with relevant labels for calculating linear approximation validity.

If we must assume **5%** error,

$$\frac{\boldsymbol{c}}{\boldsymbol{a}} = \frac{\boldsymbol{a - a\cos\theta}}{\boldsymbol{a}} = 0.05$$

So, from here we get $\theta = \mathbf{17°}$

$c = a - a\cos\theta = a(1 - \cos\theta) = 2\mathrm{R_e} * 0.05$

$Y = b\sin\theta = R_e\sin 17° = 0.29R_e$

Distance wise,
d= $\sqrt{c^2 + Y^2}$ ≈**1840 km**

Therefore, the minimum distance over which linearization is valid is **1840 km**. This distance is sufficient to linearize any incoming positional data relayed by sensors onboard the satellite.

❖ Structure of program/algorithm on which program is based:

The program tries to find out whether the debris and satellite intersect at a point or not as a first criterion and then recommends firing in a for a particular time to avoid debris. Step by step algorithm is:

1. Take inputs which include position and velocity of satellite and relative position and velocity of debris with respect to satellite and satellite size
2. Convert everything to absolute coordinate system
3. Find unit vectors of satellite and debris velocity and their cross product (whether the cross-product result is valid or not can be through different conditions)
4. Use a parametric equation of line of the debris and the satellite and we equate them and find parametric simultaneous equations. (**The parametric variables in this case indicate time so they must always be positive**). We solve the parametric equation in each coordinate differently.
5. Depending on the three simultaneous equations and the parametric variables on solving those equations, we can create different cases **[1]** according to which we decide if they intersect or not.

6. If they do, we apply firing in the cross-product direction for a particular time or if they don't, we terminate the operation.

Actual Program:

➔ This is the actual program I framed ***in Python*** (**Note: This program is just a blueprint or logic on basis of which the algorithm can be implemented):**

```
from scipy import linalg
import numpy as np
import math
psx= float(input("Position x coordinate of satellite:"))
psy = float(input("Position y coordinate of satellite:")) #line 5
psz = float(input("Position z coordinate of satellite:"))
vsx = float(input("Velocity x coordinate of satellite:"))
vsy = float(input("Velocity y coordinate of satellite:"))
vsz = float(input("Velocity z coordinate of satellite:"))
rpdx = float(input("Enter x position  of debris relative to satellite:")) #line 10
rpdy = float(input("Enter y position of debris relative to satellite:"))
rpdz = float(input("Enter z position of debris relative to satellite:"))
rvdx = float(input("Enter x velocity of debris relative to satellite:"))
rvdy = float(input("Enter y velocity of debris relative to satellite:"))
rvdz = float(input("Enter z velocity of debris relative to satellite:")) #line 15
satsize = float(input("Enter size of satellite:"))
rpd = np.array([rpdx,rpdy,rpdz])
rvd = np.array([rvdx,rvdy,rvdz])
ps = np.array([psx,psy,psz])
vs = np.array([vsx,vsy,vsz]) #line 20
magvs=(math.sqrt(pow(vs[0],2)+pow(vs[1],2)+pow(vs[2],2)))
pd = rpd + ps
```

```
vd = rvd + vs
magvd = (math.sqrt(pow(vd[0],2)+pow(vd[1],2)+pow(vd[2],2)))
vdx = vd[0] #line 25
vdy = vd[1]
vdz = vd[2]
pdx=pd[0]
pdy=pd[1]
pdz=pd[2] #line 30
unitvs=vs/magvs
unitvd=vd/magvd
resdir1 = np.cross(unitvd,unitvs)
resdir2 = np.cross(unitvs,unitvd)
c1= pdx-psx #line 35
c2 = pdy-psy
c3= pdz-psz
print("x velocity of debris",vdx,"y velocity of debris",vdy) #just printing velocities
A=np.array([[vsx,-vdx],[vsy,-vdy]])
B=np.array([[c1],[c2]]) #line 40
if np.linalg.det(A)==0.0:
 solve1=np.array([[vsx,-vdx],[vsz,-vdz]])
 solve2=np.array([[c1],[c3]])
 if linalg.det(solve1)==0.0:
  if c1==c2: #line 45
   if c1==c3:
    print("They are coinciding")
    if magvd==magvs:
     print("Both moving together.No need to do anything.")
     exit()#line 50
    elif magvd>magvs:
     print("Debris is faster than satellite.If debris is behind satellite then fire in perpendicular direction or else all ok.")
     exit()
    elif magvd<magvs:
     print("Debris is slower than satellite.If debris is ahead satellite then fire in perpendicular direction or else all ok.") #line 55
     exit()
   else:
    print("Both are parallel all ok.")
    exit()
  else: #line 60
   print("They are parallel")
   exit()
 else:
  solve3=linalg.solve(solve1,solve2)
  t1= solve3[0,0] #line 65
  u1=solve3[1,0]
```

```
    if t1>0 and u1>0:
      if t1==u1 or abs(t1-u1)<=1:
        print("Object intersect at one point.")
        print("Fire in ",resdir1," or " , resdir2) #line 70
        exit()
      else:
        print("Object will not meet at same time")
        exit()
    else: #line 75
      print("All ok. False alarm.")
      exit()
else:
  C=linalg.solve(A,B)
  t=C[0,0]  #line 80
  u=C[1,0]
  print("t=",t,"u=",u)
  if t>0 and u>0:
     print("t=",t)
     print("u=",u) #line 85
     if (vsz*t)-(vdz*u)==pdz-psz:
      print("The path of the objects have a common intersection point")
      if t==u or abs(t-u)<=1:
       print("Objects will meet at same time at same place")
       print("Fire in direction",resdir1 , "or", resdir2) #line 90
       exit()
      elif abs(t-u)>=1:
       print("Objects will not meet at same time")
       exit()
     else: #line 95
      print("The two orbits are skew lines")
      s=np.cross(vs,vd)
      diff=pd-ps
      mag =np.linalg.norm(s)
      q = s/mag #line 100
      sd= abs(np.dot(diff,q))
      print(sd , " is the shortest distance between satellite and debris.")
      if sd<=(3*satsize) :
        if t==u  or abs(t-u)<=1:
         print("Satellites too close. Change path by firing in",resdir1,"or",resdir2) #line 105
        else:
          print("All oK")
      else:
        print("ALL ok.")
  else:  #line 110
    print("All ok")
```

➔ Program Explanation:

1. *From start till cases:*

- Of course, the first three lines correspond to importing different libraries required to write the program. I have used the linalg function from scipy library, imported numpy(as np) and imported math library.

- From **line 4 - line 15,** the variables which are to be inputted are: psx,psy and psz correspond to the absolute position of the satellite in x,y and z direction. vsx,vsy and vsz correspond to the absolute velocity of the satellite in x,y and z direction. rpdx,rpdy and rpdz correspond to the relative position of debris with respect to satellite. rvdx, rvdy and rvdz correspond to the relative velocity of the debris in x,y and z direction. function

- According to **point 1 of Assumptions**, we get the position of the satellite with respect to the center of Earth but for debris we will have to find using relative concept. We have to find these relative quantities using the coordinate system I have drawn before.

- In **line 16**, we also input satsize which is the size of the satellite required for setting the closest approach limit.

- Next in **line 17**, we write rpdx,rpdy and rpdz combined together as rpd(relative position of debris) vector(rpd=np.array[rpdx,rpdy , rpdz]).
  - ➢ **np represents a numpy library imported as np and an array function from numpy is used to represent a vector as a 1D array.**

  The rpd vector corresponds to a random position recorded at a particular point in time. Similarly rvdx , rvdy and rvdz are combined together as rvd(relative velocity of debris vector) in **line 18.**

- Similarly on combining psx,psy and psz , we get the ps(position of satellite) vector in **line 19**. The position vector just refers to a random recorded position of the satellite at a point in time in its orbit(the time at which we consider the position vector doesn't matter).On combining vsx,vsy and vsz we get vs(velocity of satellite) vector in **line 20**.

- In **line 21**, we also define a new variable magvs which is used to find the magnitude of vector vs.
  The magnitude of a vector $\overrightarrow{A} = (A_x, A_y, A_z)$ is given by
  $|\overrightarrow{A}| = \sqrt{{A_x}^2 + {A_y}^2 + {A_z}^2}$ *(where A_x , A_y, A_z are x , y and z components of vectors but if we take our case where the vector's tail is at origin , they are equal to the x,y and z coordinate of the vectors)*

  - ➢ **pow() function is used to raise a variable to a particular power(i.e., 2 in this case) and the sqrt() function of the math library is used to take the square root of a quantity.**

- Now, according to relative motion concept,
  Absolute debris position vector(pd) = Relative debris position vector with respect to satellite + Absolute position vector of satellite
  ∴ pd = rpd + ps **(line 23)**
  Similarly for velocity,
  vd = rvd + vs **(line 24)**

- We also define one more variable magvd in **line 25** which is the magnitude of the vd vector.

- Now from **line 25- line 27**, using indexes used to find array elements, we find vdx(x coordinate of vd vector) , vdy(y coordinate of vd vector) and vdz(z coordinate of vd vector). Similarly we find pdx , pdy , pdz of the pd vector from **line 28-line 30.**

- We also find the unit vector for the vd vector.
  Let $\vec{A}$ be a vector and let $\hat{A}$ be its unit vector
  $|\vec{A}|$ is magnitude of $\vec{A}$.
  $$\hat{A}= \frac{\vec{A}}{|\vec{A}|}$$

Therefore, in case of vs vector,
unitvs = $\frac{vs}{magvs}$ **(line 31)**
Similarly,
unitvd = $\frac{vd}{magvd}$ **(line 32)**

- Now, in **line 33 and line 34**, to fire thrusters, the best direction (unit vector I mean) to fire them is perpendicular to the plane i.e., perpendicular to the unit vector of velocity (I mean absolute when I don't mention anything) of both the debris and satellite. Thus, we use the
  - ➢ **np.cross() function to find the cross product of unitvs and unitvd(whether the cross product will actually be used as a direction to fire or not depends on the different cases which will come ahead).**

  Also, we know that both $\vec{A}$ x $\vec{B}$ and $\vec{B}$ x $\vec{A}$ are perpendicular to the plane of $\vec{A}$ and $\vec{B}$.Thus, there are two directions possible perpendicular to the plane which are given by resdir1 = cross product of unitvd and unitvs taken in order **(line 33)** and
  resdir2=cross product of unitvs and unitvd taken in order **(line 34)**.
- Now, we must find out the parametric variables (t and u as given in the program). Here the parametric variables correspond to time.

  If we assume constant velocity (as stated before) over a short distance on a linear path **[2]**
$$\vec{v} = \frac{\Delta\,\vec{r}}{t}$$
where $\Delta\,\vec{r}$ is change in position and t is time over which change occurs.
$$\therefore \quad \vec{v}\cdot t = \Delta\,\vec{r}$$
We can write $\vec{v}$ as $(v_x\,,\,v_y\,,\,v_z)$
Let $\vec{r_0}$ be initial position vector (the position vector inputted) and $\vec{r}$ be a general vector.
$\Delta\,\vec{r}\ = \vec{r}$- $\vec{r_0}$
$\therefore\ \vec{v}\cdot t = \vec{r_1}$- $\vec{r_0}$
Therefore, $\vec{r}$= $\vec{r_0}$ + $\vec{v}$.t
We can write $\vec{r_0}$ as $((\vec{r_0})_x,(\vec{r_0})_y,(\vec{r_0})_z)$
Similarly, $\vec{v}$ can be written as $(\vec{v}_x\,,\,\vec{v}_y\,,\,\vec{v}_z)$

$\vec{r}$ = $((\vec{r_0})_x\,,\,(\vec{r_0})_y,\,(\vec{r_0})_z\,)$ + t $(\vec{v}_x\,,\,\vec{v}_y\,,\,\vec{v}_z\,)$
Therefore, individually the equations can be written as
$\vec{r_x}$ = $(\vec{r_0})_x$ + $\vec{v}_x$.t
$\vec{r_y}$ = $(\vec{r_0})_y$ + $\vec{v}_y$.t
$\vec{r_z}$ = $(\vec{r_0})_z$ + $\vec{v}_z$.t
Where $\vec{r_x}$,$\vec{r_y}$,$\vec{r_z}$ are the component vectors of the general position vector $\vec{r}$.

For satellite **(parametric variable is t in this case) ,**

$$\vec{\boldsymbol{r}}_{\boldsymbol{s}} = (\boldsymbol{psx}, \boldsymbol{psy}, \boldsymbol{psz}) + \boldsymbol{t}(\boldsymbol{vsx}, \boldsymbol{vsy}, \boldsymbol{vsz})$$

<u>Note</u>: *We are assuming the tail of the position vector from origin, therefore, the three component vectors of the position vector $\vec{r}_s$ will be equal to the respective coordinates of the point*

Thus, for satellite let us assume the three-component vector to be cs_x , cs_y , cs_z .therefore, the three component vectors will be:

$$cs_x = psx + vsx.t$$
$$cs_y = psy + vsy \cdot t$$
$$cs_z = psz + vsz \cdot t$$

For debris (the parametric variable in this case is u) let us assume the three component vectors to be cd_x , cd_y , cd_z.Therefore, the three equations will be

$$cd_x = pdx + vdx \cdot u$$
$$cd_y = pdy + vdy \cdot u$$
$$cd_z = pdz + vdz \cdot u$$

Now, for the two objects to intersect at a point,

$cs_x = cd_x$ **(for x intersection)**
$cs_y = cd_y$ **(for y intersection)**
$cs_z = cd_z$ **(for z intersection)**

Therefore,

$psx + vsx.t = pdx + vdx.u$
$\therefore\ (vsx \cdot t) - (vdx \cdot u) = pdx - psx$
$Let\ pdx - psx = \boldsymbol{c_1}(line\ 35)$
$\therefore\ (vsx \cdot t) - (vdx \cdot u) = \boldsymbol{c_1}\ (\boldsymbol{equation\ 1})$

$psy + vsy \cdot t = pdy + vdy \cdot u$
$\therefore\ (vsy \cdot t) - (vdy \cdot u) = pdy - psy$
$Let\ pdy - psy = \boldsymbol{c_2}\ (line\ 36)$
$\therefore\ (vsy \cdot t) - (vdy \cdot u) = \boldsymbol{c_2}\ (\boldsymbol{equation\ 2})$

$psz + vsz.t = pdz + vdz.u$
$\therefore\ (vsz \cdot t) - (vdz \cdot u) = pdz - psz$
$Let\ pdz - psz = \boldsymbol{c_3}(line\ 37)$
$\therefore\ (vsz \cdot t) - (vdz \cdot u) = \boldsymbol{c_3}\ (\boldsymbol{equation\ 3})$

Let us consider these three equations to be three lines.
Now, let us consider equation 1 and equation 2
$(vsx \cdot t) - (vdx \cdot u) = \boldsymbol{c_1}$
$(vsy \cdot t) - (vdy \cdot u) = \boldsymbol{c_2}$

<u>Note:</u> These two equations and the properties of the lines of these two equations tell us if the lines along which the debris and satellite move intersect in the x-y coordinate system or not.

We can solve these two equations with the help of matrices.
Then t and u can be obtained by solving,

$$\begin{bmatrix} vsx & -vdx \\ vsy & -vdy \end{bmatrix} \cdot \begin{bmatrix} t \\ u \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

Let:

$A = \begin{bmatrix} vsx & -vdx \\ vsy & -vdy \end{bmatrix}$(Line 39)

$B = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ $(line\ 40)$

Point i.]: Now, let us consider a matrix

$H = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

The determinant of this matrix (det H) = $ad - bc$

Thus, if det H=0,

$ad - bc = 0$

$ad = bc$

$\frac{a}{b} = \frac{c}{d}$

Point ii.]: Let us consider two lines having equations:

$a_1x + b_1y + c_1 = 0$

$a_2x + b_2y + c_2 = 0$

For these two lines to be parallel,

$\frac{a_1}{b_1} = \frac{a_2}{b_2} \neq \frac{c_1}{c_2}$

For these two lines to be coinciding/same,

$\frac{a_1}{b_1} = \frac{a_2}{b_2} = \frac{c_1}{c_2}$

2. *Cases:*

Note: I might use different parametric variables in some hypothetical examples. However, in real examples corresponding to the input/output of the program I have used the parametric variables given in the program. (In case 1 , I have used t1 and u1 as parametric variables but in case 2 I have used t and u).

Case 1: If det A = 0(line 41-line 77)

Let us consider:

$A = \begin{bmatrix} vsx & -vdx \\ vsy & -vdy \end{bmatrix}$

If det A = 0,
This means that,

$$\frac{vsx}{vsy} = \frac{-vdx}{-vdy} = \frac{vdx}{vdy}$$

This means that equation 1 and equation 2 are parallel or coinciding.
Let us keep this aside for the moment.
Let us consider equation 1 and equation 3 now.

$$(vsx.t) - (vdx.u) = c1$$
$$(vsz.t) - (vdz.u) = c3$$

To solve these two equations let us define two matrices.

$$solve1 = \begin{bmatrix} vsx & -vdx \\ vsz & -vdz \end{bmatrix}$$
$$solve2 = \begin{bmatrix} c_1 \\ c_3 \end{bmatrix}$$

Let us now consider two subcases:

- **Subcase 1: If det solve1=0(line 44-line 62):**

As explained before this means that

$$\frac{vsx}{vsz} = \frac{-vdx}{-vdz} = \frac{vdx}{vdz}$$

Thus, taking equation 1 and equation 3 into consideration,
This means that either equation 1 and equation 3 are parallel or coinciding **(Point ii.)**.
Now, let us revert to equation 1 and equation 2.
We already proved that either equation 1 and equation 2 are parallel or coinciding **(Point ii.).** This depends on whether constants c_1 and c_2 are equal or not. Thus, let us consider two more subcases:

- **Subcase A: If $c_1 = c_2$ (line 45-line 60)**

This indicates that the two lines are coinciding in nature i.e., the two lines are one over the other.
Now again let us revert to equation 1 and equation 3. Similarly, for them to be parallel or coinciding depends on whether constants c_1 and c_3 are equal or not. Thus, let us consider two more subcases:

- **Subcase (i): $c_1 = c_3$ (line 46- line 56)**

This means that equation 1 and equation 3 are coinciding i.e., one over the other. We have already proved in subcase A that equation 1 and equation 2 are coinciding and we have also proved that equation 1 and equation 3 are coinciding. This brings us to the combined result that the lines along which debris and satellite are moving are the same/coinciding.
For example, let us consider two lines in 3D.

$$L_1 = (2,2,2) + \lambda_1 \cdot (2,2,2)$$

$$L_2 = (4,4,4) + \lambda_2 \cdot (4,4,4)$$

If we split it into three equations, we get:

For L_1	For L_2
$2+2\lambda_1(for\ x)$ $2+2\lambda_1(for\ y)$ $2+2\lambda_1(for\ z)$	$4+4\lambda_2(for\ x)$ $4+4\lambda_2(for\ y)$ $4+4\lambda_2(for\ z)$

Solving the two systems individually for x, y and z we get
$2\lambda_1 - 4\lambda_2 = 2$
i.e., $\lambda_1 - 2\lambda_2 = 1$(for all x,y as well as z)

We can thus, see that for equation 1 and 2,
$\frac{1}{1} = \frac{-2}{-2}$ and $c_1 = c_2 = 1$
Thus, equation 1 and equation 2 are coinciding.

We can thus, see that for equation1 and 3,
$\frac{1}{1} = \frac{-2}{-2}$ and $c_1 = c_3 = 1$
Thus, equation 1 and 3 are coinciding
Thus, equation 1,2 as well as 3 are coinciding.

This can also be proved by plotting it in a 3D coordinate system (I have used GeoGebra here. In this diagram the two lines appear one(coincident)

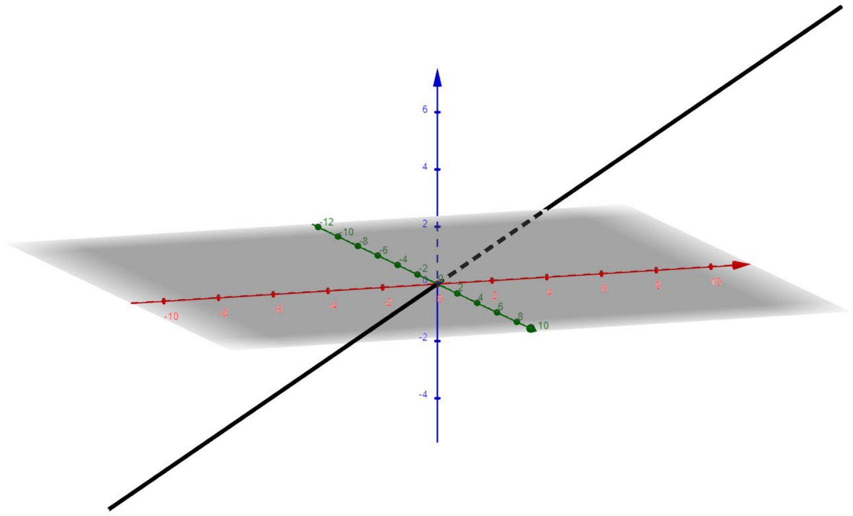


Fig 3: Lines L_1 and L_2(for this case) plotted in GeoGebra 3D

Now whether the debris and satellite collide or not depends on relative velocities of the debris and satellite. So let us consider three further subcases:

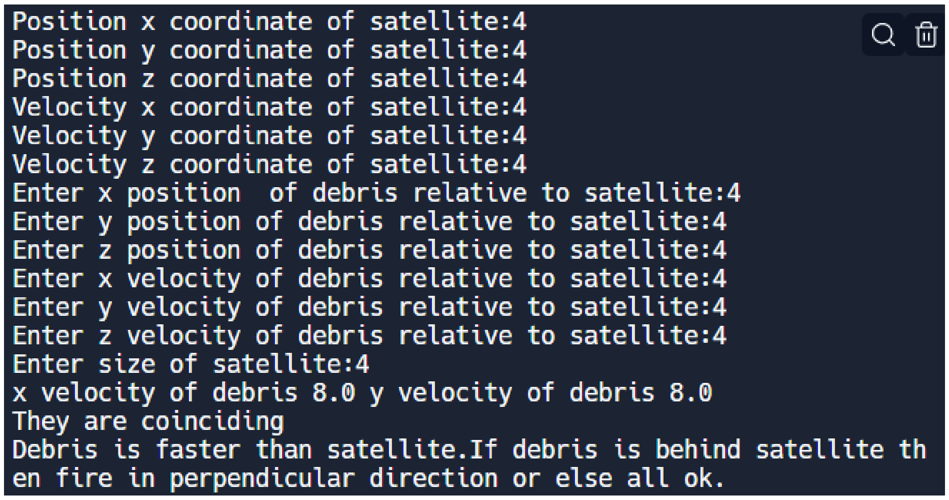
- **Subcases a.): magvd=magvs (line 48-line 50)**

This basically means that the magnitude of relative velocity (since direction of velocity is same for coincident paths) between debris and satellite is 0. Thus, we can say that we do not need to fire the thrusters of the satellite.

- **Subcase b.) : magvd> magvs (line 51-line 53)**

In this case it completely depends on whether the debris is behind or ahead of the satellite. Thus, in this case the program will print that if the debris faster than satellite and is behind it we can fire in any perpendicular direction to avoid the debris since it is approaching the satellite and if the debris is ahead of the satellite, we don't need to do anything (all ok) since debris is moving away.

For example, let us consider the following inputs:



```
Position x coordinate of satellite:4
Position y coordinate of satellite:4
Position z coordinate of satellite:4
Velocity x coordinate of satellite:4
Velocity y coordinate of satellite:4
Velocity z coordinate of satellite:4
Enter x position  of debris relative to satellite:4
Enter y position of debris relative to satellite:4
Enter z position of debris relative to satellite:4
Enter x velocity of debris relative to satellite:4
Enter y velocity of debris relative to satellite:4
Enter z velocity of debris relative to satellite:4
Enter size of satellite:4
x velocity of debris 8.0 y velocity of debris 8.0
They are coinciding
Debris is faster than satellite.If debris is behind satellite th
en fire in perpendicular direction or else all ok.
```

Fig 4: Screenshot of an input into the program

In these inputs, $rvd = (4,4,4)$ and $rpd = (4,4,4)$; $vs = (4,4,4)$ and $ps = (4,4,4)$
$pd = rpd + ps = (4,4,4) + (4,4,4) = (8,8,8)$
$vd = rvd + vs = (8,8,8)$

Thus,
$L_1(same\ as\ \vec{r_s}): (4,4,4) + \lambda_1 \cdot (4,4,4)$
$L_2(same\ as\ \vec{r_d}): (8,8,8) + \lambda_2 \cdot (8,8,8)$
Thus, on splitting and solving L_1 and L_2 for x,y and z directions , we get
$4\lambda_1 - 8\lambda_2 = 4$
$\lambda_1 - 2\lambda_2 = 1$ (for x,y as well as z)
Thus, on using the conditions listed assumed before in the cases, combined we can say that lines along which satellite and debris move are coinciding.

This can again be proved using GeoGebra.

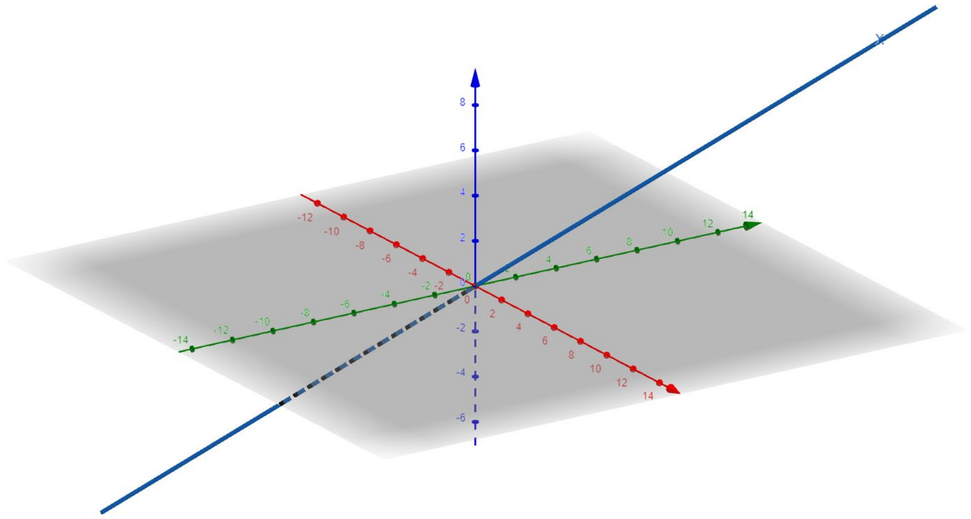


Fig 5: Lines L_1 and L_2(in this case) plotted in GeoGebra 3D

Thus, we can see that the lines L_1 (represented in black) and L_2 (represented in blue are coinciding).

- ○ **Subcase c.): magvd<magvs (line 54-line 56)**

In this case it also depends on whether the debris is behind or ahead of the satellite. Thus, in this case the program will print that if the debris is ahead of us we can fire in any perpendicular direction to avoid it since the satellite is approaching it and if the debris is behind the satellite we don't need to do anything (all ok) since the satellite is moving away.

- ▪ **Subcase (ii): $c_1 \neq c_3$ (else) [line 57-line 59]**

In this case, since $\boldsymbol{c_1} \neq \boldsymbol{c_3}$, this means that equation 1 and equation 2 are coinciding since ($\boldsymbol{c_1} = \boldsymbol{c_2}$) but equations 1 and 3 are not coinciding but parallel. So equation 2 and equation 3 are also not coinciding but parallel. This means combined that the lines along which debris and satellite are traveling intersect are parallel to each other so no firing needed.

- ➢ **Subcase B: _$c_1 \neq c_2$ (else) [line 60-line 62]**

In this case, if $\boldsymbol{c_1} \neq \boldsymbol{c_2}$, it means equation 1 and equation 2 do not coincide but are parallel. We do not know anything about c_1 and c_3 but it really doesn't matter since irrespective of that, the combined path of debris and satellite is parallel.

- ▪ **Subcase 2: if det solve1 ≠ 0 (else)[line 63-line 77]**

In this case the lines of equation 1 and equation 2 are parallel or coincident but the lines of equation 1 and equation 3 are intersecting and neither parallel nor coincident. So combined, we can say that the path of debris and satellite intersect in z.
Thus, in this case we will solve for the two parametric variables (previously I named them $\lambda_1$and λ_2 but now I am naming them t1 and u1) using equation 1 and equation 3 using matrices.
We will solve the equation:

$$solve1 \cdot \begin{bmatrix} t1 \\ u1 \end{bmatrix} = solve2$$

solve is function in linalg method in scipy library which helps in solving the above equation. After solving for t1 and u1, there are two possible subcases,

➢ **Subcase A: if t1>0 and u1>0(line 67-line 77):**

The parametric variables measure time in this case and time must always be positive. Negative time does not exist. Let us consider two subcases

▪ **Subcase (i): if t1=u1 or $|t1 - u1|$<=1(line 68-line 71)**

When t1=u1 or if $|t1 - u1|$<=1(I have kept this condition since it is possible that we might have some error in linearizing and so to keep a safety factor in mind, given a max one second time gap.), the two objects can be considered intersecting at a point and so the satellite should fire in resdir1 or resdir2 (given before)

▪ **Subcase (ii): if $|t1 - u1|$ >= 1(else) [line 72-line 74]**

The satellite and debris will reach the intersection point at different times so all ok no need to fire.

➢ **Subcase B: If either t1 or u1 is negative or both are negative(else)[line 75-line 77]**

If either t1 or u1 is negative or both are negative, then debris and satellite will not intersect (time cannot be negative).

Case 2: if det A≠0(else)[line 78-line 111]

If det A is not zero then it means that equation 1 and equation 2 are intersecting (In this case I am assuming parametric variables as t and u)

So we solve for t and u by using matrices by solving the equation (using linalg.solve() function)

$$A \cdot \begin{bmatrix} t \\ u \end{bmatrix} = B$$

We then get the values of t and u and check if it is positive or not (explained before). Thus, there are two subcases.

▪ **Subcase 1: t>0 and u>0(line 83-line 109)**

Now there are two subcases depending on whether t and u satisfy equation 3 or not

➢ **Subcase A: (vsz*t) -(vdz*u) =pdz-psz(line 86-line 94)**

If this is true then the two paths of the satellite and debris do intersect at a point but now at what time they intersect matters. So further there are two subcases:

▪ **Subcase (i): t=u or $|t - u|$<=1(line 88-line 91)**

In this case, the satellite and debris intersect at a point at the same or roughly the same time. So to avoid this from happening we fire in resdir1 or resdir2

▪ **Subcase (ii): $|t - u|$>=1(else)[line 92-line 94]**

In this case, the satellite and debris cross a point but not at the same time and thus, we do not need to change direction i.e., fire thrusters.

➢ **Subcase B: if equation 3 not satisfying(else)[line 95-line 109]**

In that case the paths of the satellites and debris will intersect in the 2D(x-y coordinate) system but they will not meet in the z coordinates (in 3D coordinate system)since equation 3 is not satisfying. These paths will then be considered as skew lines i.e., the lines that are not parallel but are not intersecting as well. In this case we can find the shortest distance between two skew lines.

Let us assume two skew lines **[4]**

$$\boldsymbol{\overrightarrow{r_1} = \overrightarrow{a_1} + \lambda \cdot \overrightarrow{b_1}}$$
$$\boldsymbol{\overrightarrow{r_2} = \overrightarrow{a_2} + \lambda \cdot \overrightarrow{b_2}}$$

The shortest distance between these two skew lines(d) is given by

$$d = \frac{\left|(\vec{a}_2 - \vec{a}_1) \cdot \left(\vec{b}_1 \times \vec{b}_2\right)\right|}{\left|\vec{b}_1 \times \vec{b}_2\right|} = \left|(\vec{a}_2 - \vec{a}_1). \left(\vec{b}_1 \hat{\times} \vec{b}_2\right)\right|$$

Fig 6: The shortest distance equation

So, in our cases for lines
$L_1 = ps + t \cdot vs$
$L_2 = pd + t \cdot vd$

For satellite and debris respectively.
Let vector $s = vs \times vd$ (line 97)

Note: np.cross() is used to find cross products.

Let $\boldsymbol{diff = pd - ps(line\ 98)}$
Let $\boldsymbol{mag = |s|}$

***Note: np.linalg.norm()* is used to find magnitude of vector s**

Let $\boldsymbol{q(unit\ vector\ of\ s) = s/mag. (line\ 100)}$
Therefore $\boldsymbol{sd(shortest\ distance) = |diff.s|}$

Note: abs() is an absolute function and np.dot() is used to find dot products[line 101]

Let us consider an example (note this might not be a real case)
$L_1: (4,2,7) + \lambda_1 \cdot (-1,1,-1)$

$L_2: (1,1,1) + \lambda_2 \cdot (1,1,2)$

If we take three different equations of the line and equate each of them in x , y and z we get
$\lambda_1 + \lambda_2 = 3(for\ x)$
$\lambda_2 - \lambda_1 = 1(for\ y)$
$\lambda_1 + 2\lambda_2 = 6(for\ z)$

From equation a and b we get λ_1=1 and λ_2=2 but these values don't satisfy equation c. In GeoGebra, it looks like :

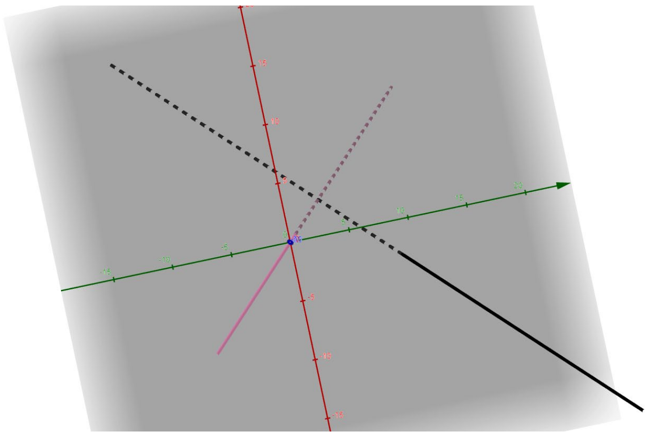


Fig 7: Lines L_1 and L_2(for this case) plotted in GeoGebra 3D(Bottom view ignoring z - axis)

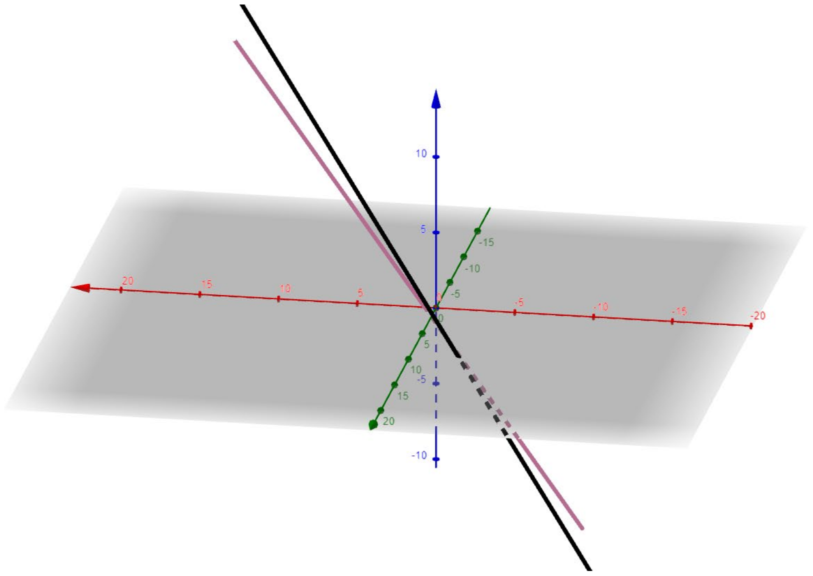


Fig 8: Lines L_1 and L_2(for this case) plotted in GeoGebra 3D (in this view, all the axis are seen)

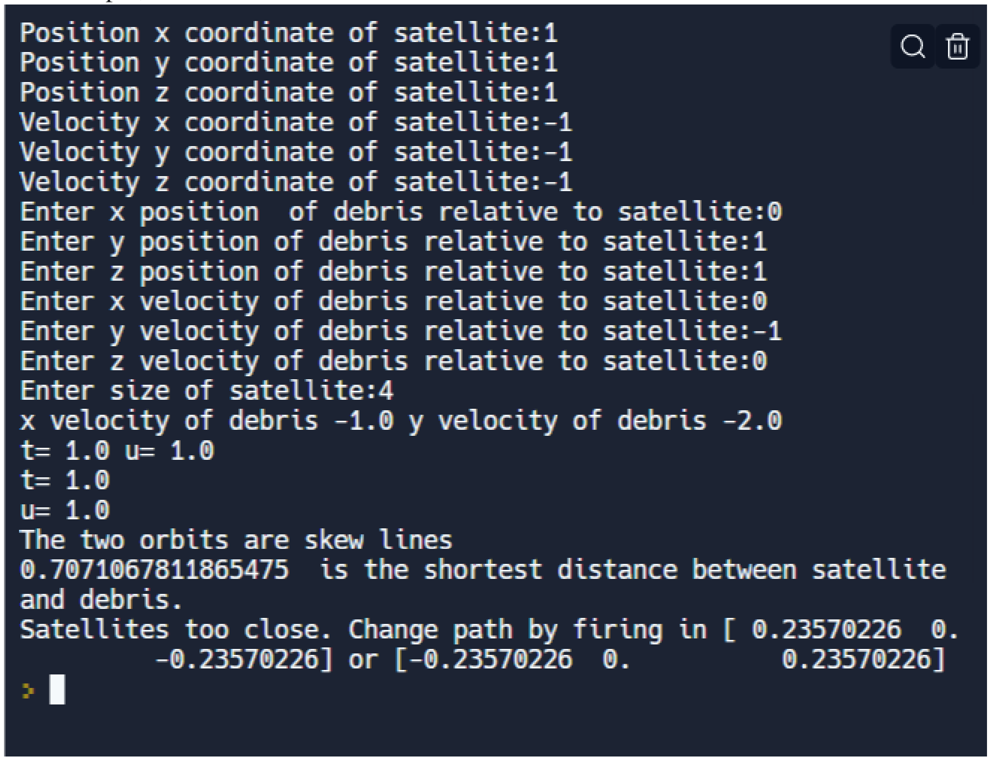
If we think carefully, the point where the two skew lines are closest is the point where they seem to intersect in the 2D x-y coordinate system. Thus, the point when they are closest depends on values of t and u.Thus, there are again two subcases.

- **Subcase (i): sd<=(3*satsize) [line 103-line 107]**

If the closest distance between the path of the debris and satellite is less than 3*satsize (safety factor assumed by me) then we can consider the two lines to be intersecting. But it also depends on the time (t and u). Thus, there are further two subcases:

- **Subcase a.): if t=u or $|t-u|$<=1[line 104 and line 105]**

In this case we can say that the satellite and debris are approaching the shortest distance (which is less than 3*satsize which I consider too close) at the same time or roughly the same time(collision course) then the satellite must fire in resdir1 or resdir2 to avoid debris.
For example,



```
Position x coordinate of satellite:1
Position y coordinate of satellite:1
Position z coordinate of satellite:1
Velocity x coordinate of satellite:-1
Velocity y coordinate of satellite:-1
Velocity z coordinate of satellite:-1
Enter x position  of debris relative to satellite:0
Enter y position of debris relative to satellite:1
Enter z position of debris relative to satellite:1
Enter x velocity of debris relative to satellite:0
Enter y velocity of debris relative to satellite:-1
Enter z velocity of debris relative to satellite:0
Enter size of satellite:4
x velocity of debris -1.0 y velocity of debris -2.0
t= 1.0 u= 1.0
t= 1.0
u= 1.0
The two orbits are skew lines
0.7071067811865475  is the shortest distance between satellite
and debris.
Satellites too close. Change path by firing in [ 0.23570226  0.
         -0.23570226] or [-0.23570226  0.          0.23570226]
> 
```

Fig 9: Screenshot of an input into the program

- **Subcase (b): if $|t-u|$>1(else) [line 106 and line 107]**

In this case though the paths of satellite and debris are very close but the two approach the closest point at different times (**$|t-u|$>1** what I consider different**).** So, no need for the satellite to fire the thrusters.
subcase 2

- **Subcase (ii): if sd>(3*satsize) [else] {line 108 and line 109}**

There is significant distance between the satellite and debris (I consider it significant) so no need to worry.

- **Subcase 2: If either t or u or both less than 0(else)[line 110 and line 111]**

If either t or u or both are negative then the debris and satellite will not intersect.

***Note:** If we want to find out how much to fire to get away from the debris, we must keep reiterating the program continuously. After the first iteration, it will show that the two paths have become skew lines since the satellite is moving away perpendicular to the debris path. When the shortest distance between the skew lines becomes more than 4 times satellite size(satsize) [according to the assumption] we stop firing.*

Conclusion

In this paper, I talked about the problem of space debris and how its active avoidance is necessary. I proposed and explained in detail a program which can be used for active avoidance of a single debris by a satellite. I know that the program is just a starting stage to something big. In the future, it may be possible to build upon this program and expand it to multi debris systems as well or maybe more cases might also be included. This program can have other applications as well. It can be used to avoid collisions between two satellites, collision of meteorites/asteroids with satellites in interplanetary space and many other applications as well.

References

1.) Hlas, M., & Straub, J. (2016, March). An autonomous satellite debris avoidance system. In 2016 IEEE Aerospace Conference (pp. 1-5). IEEE.

2.) Allain, R. (2021, October 4). Where do two lines intersect in 3 dimensions? Medium. Retrieved March 16, 2023, from https://rjallain.medium.com/where-do-two-lines-intersect-in-3-dimensions-d28f738de36a

3.) Garcia, M. (2015, April 14). Space debris and human spacecraft. NASA. Retrieved March 16, 2023, from https://www.nasa.gov/mission_pages/station/news/orbital_debris.html

4.) *Find the shortest distance between the skew lines R = (6i - toppr*. (n.d.). Retrieved March 16, 2023, from *https://www.toppr.com/ask/question/find-the-shortest-distance-between-the-skew-lines-r6i2j2kti2j2k-and-f4iks3i2j2k-where-st-are-scalars/)*

5.) Gregerson, E. (2023, February 16). Space debris. Encyclopedia Britannica. Retrieved March 16, 2023, from https://www.britannica.com/technology/space-debris

6.) Kolyuka, Y. F., Ivanov, N. M., Afanasieva, T. I., & Gridchina, T. A. (2009, September). Examination of the lifetime, evolution and re-entry features for the “Molniya” type orbits. In Proceedings of the 21st International Symposium on Space Flight Dynamics—21st ISSFD, Toulouse, France (Vol. 650, pp. 30-110).