Capstone Project Phase A

FEASIBILITY ANALYSIS AND PERFORMANCE TESTING OF COLLISION DETECTION ALGORITHMS FOR SATELLITES

23-2-D-17

Supervisor: Mr. Ilya Zeldner

Hallel Weil 316484153 Hallel.Weil@e.braude.ac.il

Omer Shamir 314763970 Omer.Shamir@e.braude.ac.il

Abstract

The project we present is based on implementing a solution to find the minimal distance between two objects in space, and the time of occurrence, which will have to work on satellite on-board computer (OBC). The project has two goals, the first one is to conduct a feasibility study, and create a proof of concept to a newly introduced algorithm by creating an actual implementation of the algorithm, and doing so while considering the environment the algorithm will have to work on. The second goal is to assess the solution in the required environment, starting with an emulated one.

The project is based on Dr. Elad Denenberg's research paper that introduced the algorithm [1].

Keywords

Satellite, Space debris, Minimal distance, Approximations algorithms, Feasibility test, Collision detection, Algorithm testing, Space mission, Distance approximation, Orbiting objects, Satellite on-board computer.

1. Introduction

One of the things that concerns satellite operators during a mission, is the risk of colliding into other objects. There is a significant amount of space debris orbiting Earth, including decommissioned satellites or parts of them, rockets, and other human-made objects, and there are also natural celestial bodies in space we should be aware of like asteroids. In order to avoid these threats, we start by keeping track of them, then we identify possible collisions and recalculate our path. Doing so is done by calculating the future orbit of 2 object and finding the point in time where the distance between them is the smallest, this time is called **Time of Closest Approach (TCA)** and the TCA and the respective distance is the values we are looking for.

With the increasing number of objects in Earth orbit, around 27,000**[3]** and the shift to cluster of smaller satellites instead of a single big one **[1]** the cost of calculating the orbit of objects and finding the TCA for our satellite is only growing. To solve this problem a few cheaper algorithms were developed. The algorithms are supposed to be cheap and fast enough to run on the satellite's own ***on-board computer (OBC)***. These algorithms have not been tested and we need to prove the task feasibility, to implement the algorithms and to show the calculation time and memory requirements in an environment with limited calculation power and memory simulating an actual satellite on-board computer.

We are working with Dr. Elad Denenberg, who created the **Conjunction Assessment Through Chebyshev Polynomials (CATCH)** algorithm **[1]**. Dr. Elad is working on creating an autonomous satellite and as part of his work he need our help. An autonomous satellite needs to calculate the possible collisions by itself and for doing that a fast algorithm for finding the TCA is needed, faster calculations are possible using approximations like the algorithms CATCH **[1]** and **Alfano\Negron Close Approach Software (ANCAS) [2]**. These algorithms were never tested on an actual satellite on-board computer, and proved fitting to run on an autonomous satellite and this is we come in.

In this project we will prove the feasibility of the task, finding TCA and the respective distance in an environment similar to a satellite on-board computer. To do so we will implement the algorithms, test their run times with different inputs, check the memory required and compare the result to the actual result (The real TCA) to find the size of the error. We will prove the task feasibility, if possible, else we'll show why it is not and give the result to Dr. Elad to help progress this research.

2. Background and Related Work

2.1. Space debris

There are around 27,000[3] object in Earth orbit, the number of objects is predicted to grow even if we stop sending new satellite into space [9] because the more object there are, the more collisions will happen and the result of each collision is more smaller objects. To deal with the increasing number of objects the international space agencies create guidelines [5][6] such as removing satellites from orbit at the end of their mission to prevent the creation of additional manmade debris. Another solution is done at CelesTrak [7] by sharing relevant knowledge for free and tracking satellites, in a program called “Satellite Orbital Conjunction Reports Assessing Threatening Encounters in Space” (**SOCRATES**) [8] they publish future possible collisions between satellites and inform the relevant satellites operators about the risk, to make sure they can avoid it.

Image 1, space debris damage, taken from Freethink’s article [14]

The damaged done to a block of aluminum by a 15 grams object moving at a speed of around 25,000[km/h]

2.2. Finding the possible collisions

Because the increasing number of active satellites and the shift to cluster of smaller satellites instead of a single big one [1] because they’re cheaper to purchase and maintain, the task of calculating the possible collisions for all the satellites is becoming more computationally expensive and complex, and running clusters of satellites is a complex task by itself for the satellite operator. Because of that the idea of an autonomous satellite become more and more relevant. An autonomous satellite needs the ability to handle its own orbit and for that identifying the possible collisions is a necessary component.

2.3. The Propagator

For most of the existing satellite, and for each future autonomous satellite, calculating its future orbit is already an integrated part in the satellite task, the satellite uses this calculation to do minor changes to its orbit to fit the desired orbit. This future orbit calculation done on the satellite system in a software called Propagator. Using the current location and velocity of an object in orbit (for example our satellite) and a point in time (for example ten minutes from now) the Propagator can calculate the object location and velocity in the given point in time. The propagator uses a forces model to find the future orbit, this force model is different between different propagators and can affect the result's accuracy and the calculation time. In this project we use a propagator called **Standard General Perturbations Satellite Orbit Model 4(SGP4) [10]**, SGP4 is old and famous propagator used for research that known for its fast run time and there are many available implementations we can use **[4]**. SGP4 get the object data using format called **Two Lines Elements (TLE)** which consist of two lines of data, including the object location, velocity, the corresponding time, the average number of revolutions per day (Mean Motion) and more.

2.4. Finding the minimal distance

Using the propagator we can calculate the location in time of two object with a small-time step, and in each point of time we can calculate the relative distance between them using the equation:

$$dist = \sqrt{(r_{1x} - r_{2x})^2 + (r_{1y} - r_{2y})^2 + (r_{1z} - r_{2z})^2} \quad \textbf{(Eq.1)}$$

and finding the minimal distance give us the TCA and respective distance. This solution is accurate but slow, using SGP4 for a large number of points is computationally expensive. To get faster calculation we can use approximation.

2.5. Finding the TCA using polynomials approximations

We will start by calculating the relative location vector:

$r_{relative} = r_1 - r_2$ **(Eq.2)**

and relative velocity vector:

$v_{relative} = v_1 - v_2$ **(Eq.3)**

using $r_{relative}, v_{relative}$ we can calculate the relative distance function:

$f(t) = r_{relative} \cdot r_{relative}$ **(Eq.4)**

and its derivative:

$$\dot{f}(t) = 2\dot{r}_{relative} \cdot r_{relative} = 2v_{relative} \cdot r_{relative} \quad \textbf{(Eq.5)}$$

Given the derivative $\dot{f}(t)$, every point in time $t*$ where $\dot{f}(t*) = 0$ we have an extrema point, that can be a local minimum or maximum of the relative distance function $f(t)$, by finding all the extrema points and find the values of $f(t)$ in each

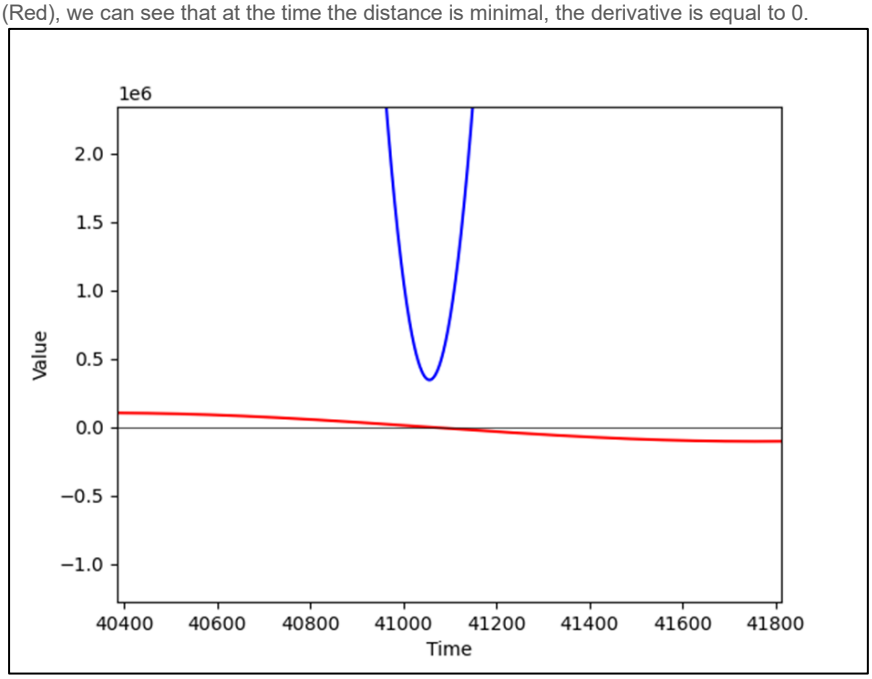
of the points, one of the points will be the minimal distance. In the next algorithms we use polynomials to approximate the relative distance function $f(t)$ and the derivative $\dot{f}(t)$ using only a finite number of points in time consisting of the location and velocity vectors of two objects. After creating a polynomial $p_{\dot{f}(t)}$ representing the derivative $\dot{f}(t)$ in an interval of time, we find the real roots of the polynomial in the interval, each of these roots is an extrema of the relative distance function $f(t)$, by finding the distance in each of the extrema points using the polynomials $p(t)_{f_x}, p(t)_{fy}, p(t)_{fz}$ to approximate the x, y, z values of the relative distance vector we can calculate the distance:

$$r(t) = \sqrt{p(t)_{f_x}{}^2 + p(t)_{fy}{}^2 + p(t)_{fz}{}^2}$$ **(Eq.6)**

and after finding the distances for all the possible points in time (the polynomial roots) we can take the minimal value and the corresponding root (the point in time) and we got the absolute minimal distance in the interval – the TCA. There are 2 algorithms using this king of approximation.

Graph 1, relative distance and derivative over time

In this graph we can see the relative distance function $f(t)$ (Blue) at some point in time and the derivative $\dot{f}(t)$ (Red), we can see that at the time the distance is minimal, the derivative is equal to 0.



2.5.1. ANCAS

The first algorithm, ANCAS **[2]** uses cubic polynomial as an approximation of a function over an interval. Given 4 points in time and the respective location and velocity vectors for 2 objects, we can find the TCA by:

Algorithm 1: ANCAS on 4 points, (the original algorithms description can be found at [2])

Input: $p[4], t[4]$

Output: $TCA + r_{TCA}$

$r_{TCA} = \inf$

$t_{TCA} = \inf$

Map the time points $t[1-4]$ to $\tau_0 = 0 \le \tau_1, \tau_2 \le \tau_3 = 1$ on the interval $[0,1]$

Calculate $\dot{f}(t)$ f_x, f_y, f_z using **Eq.(5/2)** with the points $p_i[0...3]$

Fit cubic polynomial to $\dot{f}(t)$ according to **[2 ,Eq.1f-1j]** over $[0,1]$

Find the cubic polynomial real roots

Fit cubic polynomial for f_x, f_y, f_z in the interval $[0,1]$

for each root t^* **do:**

calculate the distance r at t^* using **Eq.(6)**

if $r < r_{TCA}$ **:**

$r_{TCA} = r$

$t_{TCA} = t^*$

end

end

The cubic polynomial coefficients calculations described in article **[2]**.
Finding the roots of a cubic polynomials can be done by solving the 3rd degree equation and we will find between 1 to 3 real solutions. There is a problem with the algorithm, the point in time must be relatively close because the algorithm can only find up to 3 extrema points, so working with a large interval means we can miss possible points and even miss the actual point of the TCA. Because the root finding can be done fast using the 3rd degree equation the algorithm run relatively fast but the result can be inaccurate.

2.5.2. CATCH

The second algorithm, CATCH **[1]**, uses **Chebyshev Proxy Polynomial (CPP)[1]** to approximate the functions. The CPP can give more accurate result, depending on the degree of polynomial we want to use. We can choose high enough degree to get the size of error we want. The algorithm work on time interval from 0 to $t_{\max}$, each iteration searches the minimal distance in an interval with size Γ. The degree of the CPP is part of the algorithm input and appear as N.

Algorithm 2: CATCH the original algorithms description can be found at **[1, algorithm 2]**

Input: $p_1[], p_2[], t_{\max}, \Gamma, N$

Output: $TCA + r_{TCA}$

$r_{TCA} = \inf$

$t_{TCA} = \inf$

$a = 0$

$b = \Gamma$

While $b < t_{\max}$ do:

Fit CPP $p_{\dot{f}}$ with order N to $\dot{f}(t)$ according to **[1 , Eq.15]** over the interval $[a,b]$

Fit CPP with order N to f_x, f_y, f_z over the interval $[a,b]$

Find the roots of $p_{\dot{f}}$

for each root $t*$ **do:**

calculate the distance r at $t*$ using Eq.(6)

if $r < r_{TCA}$ **:**

$r_{TCA} = r$

$t_{TCA} = t*$

end

end

$a = a + b$

$b = b + \Gamma$

end

The algorithm needs N+1 points in time in each Gamma interval in order to create CPP of order N. After calculating the CPP coefficients we can use them to create a special NxN matrix called the companion Matrix **[1,Eq.18]** and the eigen values of this matrix are the polynomial roots. Using the roots, we found and creating CPP for $p(t)_{f_x}, p(t)_{f_y}, p(t)_{f_z}$ we can calculate the minimal distance in each interval and eventually the TCA and respective distance in $[0, t_{\max}]$. The problem with CATCH is the cost of finding the roots, which is the cost of finding eigen values for an NxN matrix, to deal with it, Dr. Elad describe in his article**[1,part 4]** that we can get sufficient results for both runtime and error size, using degree of 16 for the polynomial. Using a constant degree give us deterministic run times and the size of the error is small enough for the required result.

2.6. Testing for possible collisions

A satellite operator needs to find if there are possible collision with the satellite, to do so the operator need to find the TCA between his satellite to every other object in Earth orbit. The objects in Earth orbit are tracked constantly, the data set of the objects in Earth orbit is called the Catalog. There are around 27,000 **[3]** in the Catalog for now, and checking each one of them can be expensive. In order

to minimize the number of objects and remove object that we don't have any change to collide with, the operator will use a set of filters on the Catalog **[1,part 2.2]**, each filter can remove irrelevant objects. As for today, because of ANCAS high speed but low accuracy, it was only used as a filter for the Catalog, and all the calculations at the end were done by using a Propagator with a small time-step. By proving CATCH feasibility and quality of result we can shift the final calculation to CATCH and save a significant amount of time.

2.7. Algorithms Complexity

2.7.1. ANCAS Time Complexity

In ANCAS **[2]**, for each set of 4 data points we need to create 4 cubic polynomials, one for the relative distance derivative and 3 for the relative distance X, Y, Z. each cubic polynomials required coefficients calculation which consist of 4 equations **[2,Eq 1f-1j]**, meaning the complexity of finding the polynomial coefficients is constant. To map the time points to the interval [0,1] we use a simple calculation for each point **[2]** 4 times, one for each point. Finding the solution for a 3rd degree equations is quite simple, using a given formula with a constant run time we get between 1 to 3 real result.
For each of the roots we found, we calculate the distance using **Eq.(6),** and check if we found a smaller distance. In the worst case we check 3 times. Meaning for each set of 4 data points the complexity is (where k is a constant number):

$$O(4*k_{coefficients}+k_{roots}+3*k_{dist})=O(1)$$

Calculating the complexity for finding the TCA over **n** data points means we check the first 4 points and for each iteration after that we use the last point from the previous iteration as the first points meaning we need 3 new points, so we need to do $\left\lceil \frac{n-4}{3} \right\rceil +1=\left\lceil \frac{n-1}{3} \right\rceil \le \frac{n}{3}$ iterations.

The complexity of running ANCAS on **n** data points is:

$$O\left(\frac{n}{3}*k_{ancasIteration}\right)=O(n)$$

2.7.2. CATCH Time Complexity

In CATCH**[1]** algorithm we iterate through the number of time points in our external loop, $n=\frac{t_{max}}{\Gamma}\cdot N_{deg}=number\ of\ \ points$

Where t_{max} is the is end boundry in the time range where we're looking for mininal disdance, and Γ equal to half of the smaller revolution time of the object **[1,part 4]**, The value of N_{deg} is the order of the polynomial, while we can change the chosen value of N, it was determined that $N=16$ give sufficient results.

Inside the loop we're doing the following steps:

1. Fit the CPP of order N_{deg} to $\dot{f}(t)$, p_x, p_y, p_z over each interval of points: Assuming the arithmetic operations we use are basic operation done in time complexity of $O(1)$, we calculate the Chebyshev polynomials**[1]**. We'll iterate through $N_{deg}+1$ points, which is a constant in our case, meaning that the time complexity will also be constant. Each iteration requires us to sample a new time point which will be our input parameter x, calculating the interpolation matrix with size of $(N_{deg}+1)(N_{deg}+1)$ which is also constant.
   The complexity of this step is: $O(N_{deg})\cdot\left(O(N_{deg})\cdot\left(O(N_{deg})\cdot O(N_{deg})\right)+O(1)\right)=O(1)$

2. Finding the roots for P_f will consist of calculating the companion matrix with a size of ${N_{deg}}^2$ and finding the eigen values, using the complexity of matrix multiplication for this step, the complexity will be $O\left({N_{deg}}^3\right)=O(1)$, rescaling each eigen value to the actual coefficient value also takes constant time.

3. For each time point we'll calculate in our interval we'll check if we found a new minimal distance, if we did, we'll update the minimum distance and the time of occurrence. This step also has a constant time complexity.

It means that the only inputs that determines our time complexity are the values of how long each interval time, and how long in the future we want to look it,
meaning the complexity equals the number of different time-points we measure, which is: $O(n)$

2.7.3. Space complexity

The space complexity of the algorithms is the same. Both ANCAS and CATCH uses a constant number of internal variables to help with the calculations. Because our task is finding a minimum, we only need one variable to store the current minimum without any dependency for the input size. We also use some internal variables representing the polynomial and other related logics. The only memory that is related to the size of the input is the input itself. The input consists of 2 location vectors, 2 velocity vectors and the time point value for each time point in our data set, so we can see that the size of memory the input uses is linear to the number of points we need to test. We get constant space complexity for the algorithms themselves and linear to the number of time point for the input: $O(n+1)=O(n)$

3. Expected Achievements

In this project we expect to finish with concrete and comprehensive proof of feasibility or sufficient proof for the infeasibility of the task of running the TCA calculations on the satellite on-board computer.

To achieve this goal, we expect to create the full test system as described in the Product section ahead, including two programs, one running on our own computer and referred to as **Testing Station** and the other program running on the satellite on-board computer (OBC) we test with, and referred to as **Tested OBC**. The Testing Station is creating data sets, running test and collecting data and the Tested OBC getting the data, timing the run times and running the algorithms. The two programs need to be tested, proven and flexible enough to work with different algorithms implementations and data sets.

After finishing the implementations of the programs, we want to run experiments with different inputs and different optimized version of the algorithms and collect enough data to prove the feasibility of the task.

We want to finish the project with three main products:

1. The full testing system, including 2 parts, the first part working on a satellite computer we are testing and running the algorithms, testing the run time and sending the result to the second part. The second part, working on our own PC, creating the data for the algorithms, saving and analyzing the result the first part return. The system should be finished, tested and available for future use if needed, generic enough to be used on other types of algorithms, with minimal changes needed.

2. Concrete result of the experiments, with the variety of possible input types tested. Proving or disproving the feasibility of running the TCA calculations on a satellite on-board computer.

3. Optimized and reliable implementation of the algorithms, CATCH and ANCAS, ready for integrating into the satellite system when needed.

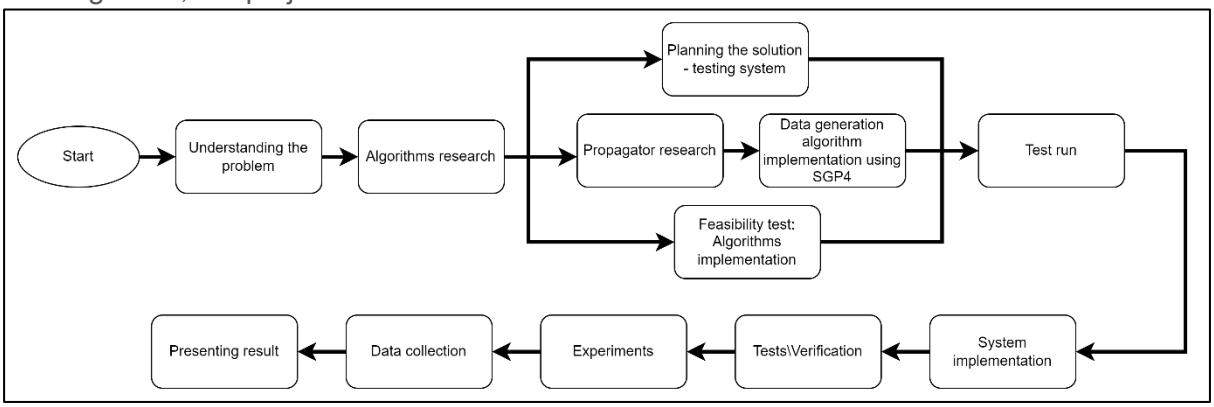
4. Research / Engineering Process

4.1. Process

4.1.1. The project work flow

At part one, we started with understanding the problem we need to solve, we moved to researching the algorithms and then started planning the solution, implementing the algorithms and researching and preparing the means to create our data – the propagator. We created a data set and used it as an input for a test run. We analyzed the result to understand the kind of result we will get in the next part. In part 2 we are planning to start with the development process of the testing system, after developing and verifying our system we can start working with the system to test the algorithms run time and memory use, by running variety of experiments of different data set as inputs, trying and implementing different variants and optimization of the algorithms to see the effect on the algorithms run time and result and collecting the data needed for our final report. After finishing our experiments, we will present the result to our client, Dr. Elad, to use in his research.

Diagram 1, the project work flow



4.1.2. Software development methodology

In this project we decided to work in an agile development process. Each iteration will consist with developing some feature, creating unit tests for the relevant code and testing of the created code on our own computer. By working with the iterative process, we can more easily adapt to changes, be it change in the dedicated system we want to test the algorithms on, changes in requirements or new requirement from our client Dr. Elad or handling a busy week during the semester. As students actively working in the industry we need to attend our jobs, do our chores and homework and work on the project at the same time. We need the development process to be as flexible as possible to allow us to combine the work on the project with the rest of our responsibilities.

4.1.3. Research and Implementation

We started our project by learning about the subject, there is a lot of information available on space debris, we learned about the problem and risks

to understand the need for our project. We met Dr. Elad and understood his requirements and studied the articles he gave us.
We implemented the algorithms, both to show the feasibility of our solution and to understand the algorithms better and find possible problems in moving from the abstract theoretical algorithm to the practical implementation of the algorithm.

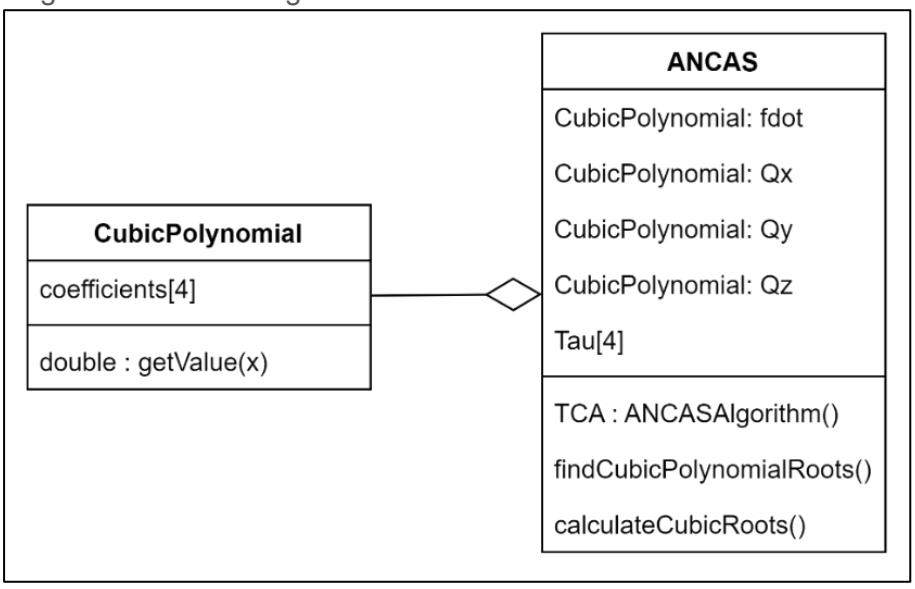
4.1.3.1. Choosing the language

The requirements: we needed a programing language that can be fast, easy to use and we wanted OOP to fit our idea of a generic, modular and easy to change implementation. We decided to choose a programming language we are familiar with and can be used easily. After discussing the subject and reading about the performance of the different languages we decided to choose C++. Another reasons we considered were the fact that there is a big variety of libraries we can use already available for C++ (eigen\boost for calculating eigen values), we also found out implementation for SGP4 in C++ in case we need to implement using SGP4 directly.

4.1.3.2. Implementing the algorithms – ANCAS

There are 2 versions of ANCAS **[2]**, the first use 2 data points but required the acceleration in addition to location and velocity, our Propagator, SGP4 only generate the location and velocity, for this reason we use the second version of ANCAS, which has 4 data points. The 4 data points version require the velocity and location vectors of 2 objects in 4 points in time, spread uniformly **[2]**. The ANCAS algorithm is described for a single set of 4 data points, while the implementation for it is iterative and run on a set of **n** points, running each iteration on a set of 4 points, to check the full time-interval each iteration uses last iteration's last point as its first point.

Diagram 2 – Class diagram ANCAS



Algorithm 3, iterative ANCAS over n points (the original algorithms description can be found at **[2]**)

n data points for 2 satellites, n time points

Input: $p[n], t[n]$

Output: $TCA + r_{TCA}$

$r_{TCA} = \inf$

$t_{TCA} = \inf$

$Offset = 0$

$lastIndex = 4$

While $lastIndex \le n$ **do:**

Map the time points $t[Offset+1 \ldots Offset+4]$ to $\tau_0 = 0 \le \tau_1, \tau_2 \le \tau_3 = 1$ on the interval $[0,1]$

Calculate $\dot{f}(t)$ f_x, f_y, f_z using **Eq.(5/2)** with the points $p_i[offset \ldots offset + N - 1]$

Fit cubic polynomial to $\dot{f}(t)$ according to **[2 ,Eq.1f-1j]** over $[0,1]$

Find the cubic polynomial real roots

Fit cubic polynomial for f_x, f_y, f_z in the interval $[0,1]$

for each root t **do:**

calculate the distance r at t using **Eq.(6)**

if $r < r_{TCA}$**:**

$r_{TCA} = r$

$t_{TCA} = t*$

end

end

$Offset = Offset + 3$

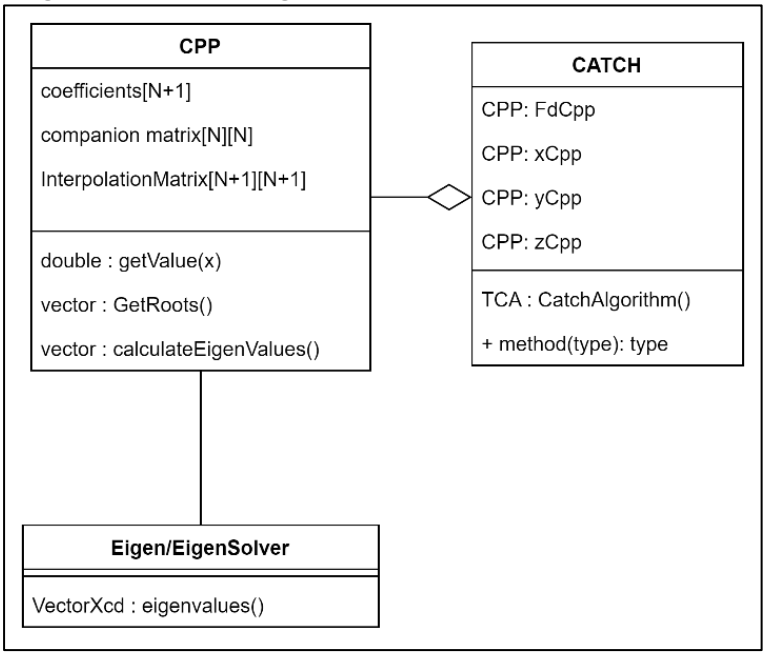
$lastIndex = lastIndex + 3$

.

4.1.3.3. Implementing the algorithms – CATCH

CATCH**[1]** is already iterative, running over the time interval $[0, t_{\max}]$, we needed to fit the algorithm to C++. We found out that the algorithm needs a specific time points, we need $N+1$ points for each interval $[a,b]$, where each interval first points is the previous interval's last point. The formula for finding the time points we need, called the "**Chebyshev Gauss Lobatto nodes**" **[1]** is described in the article**[1**,**Eq.11]**, as $t_j = \frac{b-a}{2}\cos(\frac{\pi j}{N}) + \frac{b+a}{2}$, the points $t_{0\ldots N}$ within the range of a to b where $t_0 = b, t_N = a$, and to fit this, because the points in our data are from a to b, we read $N+1$ points in each iteration from the last to the first, to read t_j we take the point in index $N-j$ from the data array. After calculating the coefficients using these points, we need to find the roots, by calculating the eigen values of the companion matrix. In our implementation we used the Eigen library, we decided to use an existing implementation at this point to show our implementation as a feasibility of our project. In the future we will test different implementations for finding eigen value to find the most efficient one.

Diagram 3 : Class diagram CATCH



Algorithm 4, CATCH implementation (the original algorithms description can be found at [1])

Input: $p_1[n], p_2[n], t_{\max}, \Gamma, N$

Output: $TCA + r_{TCA}$

$r_{TCA} = \inf$

$t_{TCA} = \inf$

$a = 0$

$b = \Gamma$

$Offset = 0$

$lastIndex = N$

While $lastIndex \le n$ **do:**

Calculate $\dot{f}(t)$ f_x, f_y, f_z using **Eq.(5/2)** with the points $p_i[offset...offset + N - 1]$

Fit CPP $p_{\dot{f}}$ with order N to $\dot{f}(t)$ according to **[1,Eq.15]** over the interval $[a, b]$

Fit CPP with order N to f_x, f_y, f_z over the interval $[a, b]$

Find the roots of $p_{\dot{f}}$

for each root t **do:**

calculate the distance r at t using **Eq.(6)**

if $r < r_{TCA}$ **:**

$r_{TCA} = r$

$t_{TCA} = t*$

end

end

$a = a + b$

$b = b + \Gamma$

$Offset = Offset + N - 1$

$lastIndex = lastIndex + N - 1$

end

4.1.4. Test run

After implementing the algorithms, we ran them on our personal computer, to obtain the result, run-time, and better understanding of what to expect in part 2. Using the implementation of SGP4 for python **[4]**, which uses the C++ implementation (run fast) but easy and fast to write scripts to(python) we created 2 fitting data sets, for a time period of 2 weeks, one for ANCAS with time-points in a fixed distance from one another, and another data set for CATCH with time-points fitting to the algorithm, we created the data for 2 satellites, **COSMOS** and **LEMUR2** based on 2 TLE we got from Dr. Elad. We tested the run time of each algorithm, running with the same number of points, on the same time period, and saved the result.

Image 2, test run result

testName	algName	numberOfPoints	runTime(sec)	runTime(microSec)	TCA distance	TCA time
LEMUR2_COSMOS	CATCH	6452	0.052032	52032	0.0665403	177106
LEMUR2_COSMOS	ANCAS	6452	0.00068	680	54.9975	174306

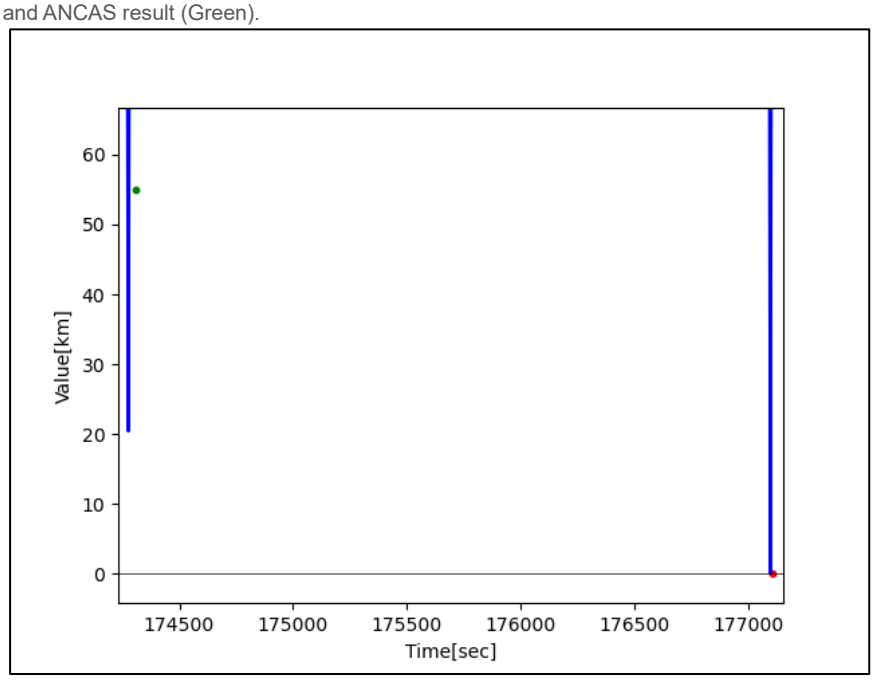
we used SGP4 with small time-steps of $10^{-5}sec$ to get actual result and got minimal distance of 0.084[KM] at the time point 177095.76[Sec] and compared them to the result we got here.

	distance [KM]	deviation [KM]	time [Sec]	deviation [Sec]
real result	0.084	-	177095.76	-
CATCH	0.067	0.017	177106	10.24
ANCAS	54.9975	54.9135	174306	2789.76

ANCAS run a lot faster compared to CATCH but with less accurate result. The result can be seen on the following graphs.

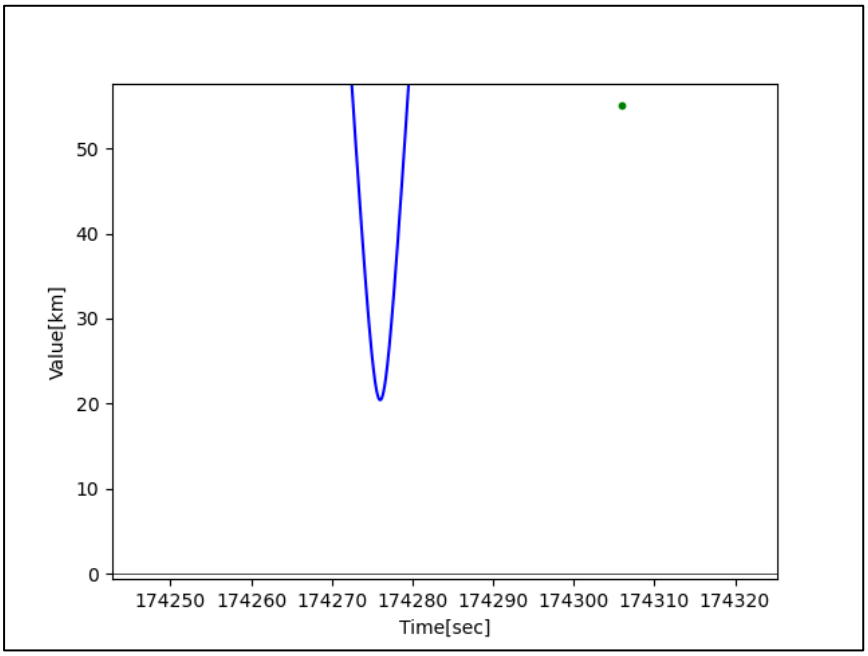
Graph 2, Algorithms result over the distance in time

In this graph we can see the distance between COSMOS and LEMUR2(Blue) over time, CATCH result (Red) and ANCAS result (Green).



Graph 3, ANCAS result over the distance in time

In this graph we can see the distance between COSMOS and LEMUR2(Blue) over time, ANCAS result (Green), we can see ANCAS result are quit far from the actual TCA but sit close to the previous point it time the satellites were close.



Graph 4, CATCH result over the distance in time

In this graph we can see the distance between COSMOS and LEMUR2(Blue) over time, CATCH result (Red), we can see relatively how close the result are both in time and the minimal distance.

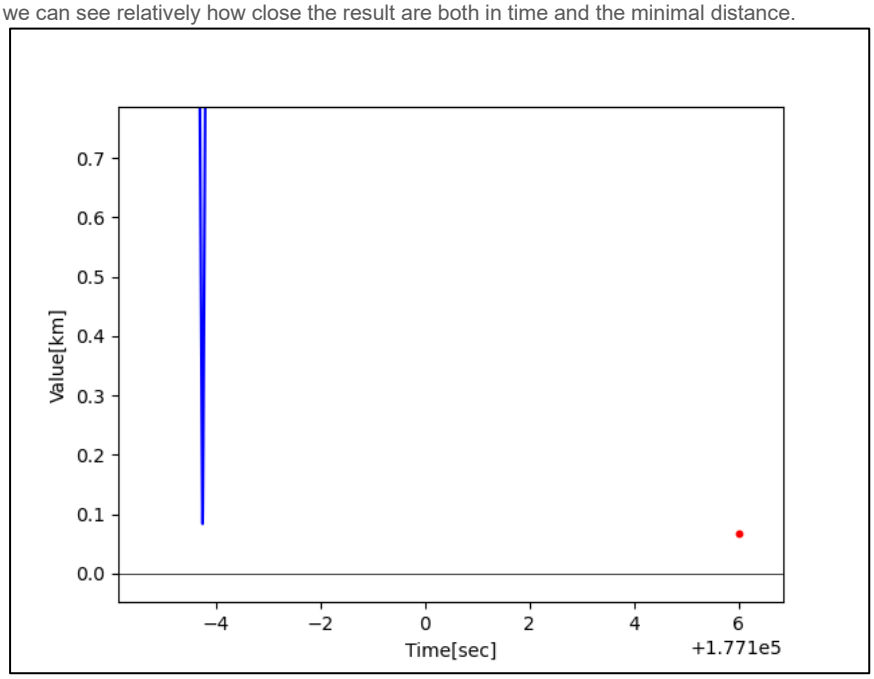


Image 3, test run result, ANCAS running with twice the number of points over the same time interval.

In this test we gave ANCAS twice the number of points(smaller distance between points), and the TCA result are better – giving the same time as CATCH but the distance is still far from close.

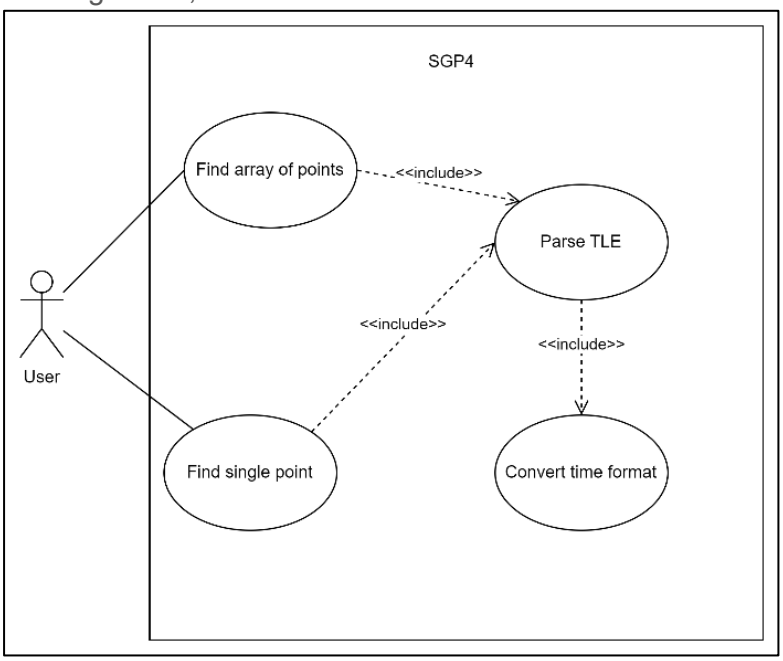
testName	algName	numberOfPoints	runTime(sec)	runTime(microSec)	TCA distance	TCA time
LEMUR2_COSMOS	CATCH	6452	0.051939	51939	0.0665403	177106
LEMUR2_COSMOS	ANCAS	12902	0.001264	1264	9.12635	177106

4.2. Product

4.2.1. Finding the real minimal distance using SGP4

We need to use SGP4 for 2 tasks, the first is creating input data for ANCAS and CATCH the same way an actual satellite will do, and the second is to find the real TCA and respective distance in order to evaluate the algorithms result and the size of the error. In order to find the TCA we need to run SGP4 over the time interval we want to test with small time steps. We needed to find optimal time step for good result and the fastest way to find it. We used SGP4 features to do so.

Diagram 4 , SGP4 use case



We can use the SGP4 array function, by giving the function array of time points as input we will get array of results faster than calling the function on a single point at a time. Because we are working with a large number of points, finding the TCA in a 2 weeks interval and calling SGP4 with a really small time step(less than a second) we needed to make sure we don't cause our software to crash because of a memory problem while trying to work with large arrays, we also wanted our algorithm to find the most accurate solution, without the need to try for many different time steps ourself. in order to do so we created a simple algorithm.

Algorithm description:

For each iteration we'll do the following:

1. Calculate times with a given step-size
2. Translate the time point to Julian day date format required by SGP4
3. Call SGP4 on array of object_1 and object_2 and get location vectors of the objects
4. For each time point, we'll calculate the relative distance between the objects

5. If a new minimal distance is received, save the distance and the time it happened, and save the vectors in a file.
6. If minimal step-size is reached, or the minimum we received is no better than the previous 2, get out of the loop. Else, reduce the step-size and repeat 1 to 6.

Improvement for the Algorithm:

Working with large size of data can make errors, such as memory errors, it is easy to deal with if we limit the number of time-point we measure to blocks with fixed size.

A. Calculate times with a given step-size
B. divide the times received into blocks
C. for each block do steps 3 to 5 in the previous algorithm
D. If minimal step-size is reached, or the minimum we received is no better than the previous 2, get out of the loop. Else, reduce the step-size and repeat A to D.

Algorithm 5, finding the TCA using SGP4

Pseudo code:
Input: tle1, tle2, t_end, st_min, p_max, factor, init_step_size, name

1. Initialize min_distance, step_size, min_time
2. Loop:
    a. Set counter to 2
    b. Calculate number of points and blocks
    c. For each block:
        i. Generate time points
        ii. Calculate jd, fr
        iii. Initialize satellite objects sat1, sat2
        iv. Assign jd, fr to sat1 and sat2
        v. Calculate r1, r2 for sat1 and sat2
        vi. Calculate distance between the objects
        vii. Save location vectors and times to a file
        viii. Find the minimum distance between the objects in the current block
        ix. If a new minimum distance is found, update min_distance and min_time
    d. If no new minimum is found in any block, decrement the counter

    while (step size is greater than st_min and counter is greater than 0)
3. Return min_distance and min_time

4.2.2. Satellite on-board computer

To understand our project environment, we need to learn about satellites systems. We need to estimate the amount of memory and CPU time our algorithm can work with. Obviously, there are a lot of satellites and each one have a different system with different on-board computer and different possibilities. Because in our project we are working with Dr. Elad, a researcher, we needed to look at the kind of system a research satellite will use. Research satellites are usually small satellite that accompany a bigger satellite's launch, catching a ride beside it to save the launch cost. Another thing we need to take into account is that the satellite needs most of its computation power to be able to perform the task it was created for, research,

communication or any other task, any calculation we need that take away from its limited computation power are essentially stopping the satellite from performing its task.

4.2.2.1. Requirements

There are many options available for choosing our satellite **on-board computer (OBC)**, and we need to filter them by determining what is essential to us.

Accurate clock:
Because our purpose is research, we'll need a computer with the ability to measure time accurately, this way we can assess the performance of the algorithms in the real environment.

Modest computing resources:
By choosing a weaker satellite on-board computer we can achieve the following goals:

1. Lower computing capabilities will cost less money, we don't have to use high-end satellite board with capabilities we don’t need.
2. To assess the performance of the algorithm, we want our computing capabilities to be limited, to check it's functionally even in environment with tight resources.
3. By testing the algorithms in a low resource environment, we prove its feasibility in any other environment with more resources.
4. The algorithms we are testing are meant to be a part of a research satellite which usually is a smaller satellite with a small and efficient on-board computer.

Memory and storage:
We do want to have some minimal requirement, for instance, to have enough storage and memory, but with the size of the compiled program being negligible and a simple OS it shouldn't be a problem even for the low-end ones.

Operation system capabilities:

1. The operating system needs to support the system calls we plan to use, like getting the current time for our timer.
2. Another option is to install our own operating system on the computer.
3. We want the OS to be as light as possible, to have only the services we need, the reason behind it is that this way we'll get more accurate results from testing our algorithm, by reducing the context switches and avoid handling unnecessary tasks and errors.

Interface:
For out testing, we'll need a communication interface that will enable us to feed data to the on-board computer and receive the results back. There are

many possible communications types we can use, our main preference is a communication type we can connect to our PC without the need for special adaptors and drivers, like Ethernet or USB.

4.2.2.2. Our choice

One of the difficulties we encountered was the lack in available data on prices for the satellite components, because the small market it is, and it made comparing different on-board computers a much harder task.
Eventually we came up with the **EnduroSat's Onboard Computer Type 1[12]**,
with the specifications:

Processor:	ARM Cortex M7
SRAM:	1MB
FRAM:	8Mbit
Clock type:	Real Time Clock
Secure onboard memory:	8GB
Interfaces:	4x RS-485, 2x RS-422, 3x UART, 2x I2C, SPI, USB, CAN

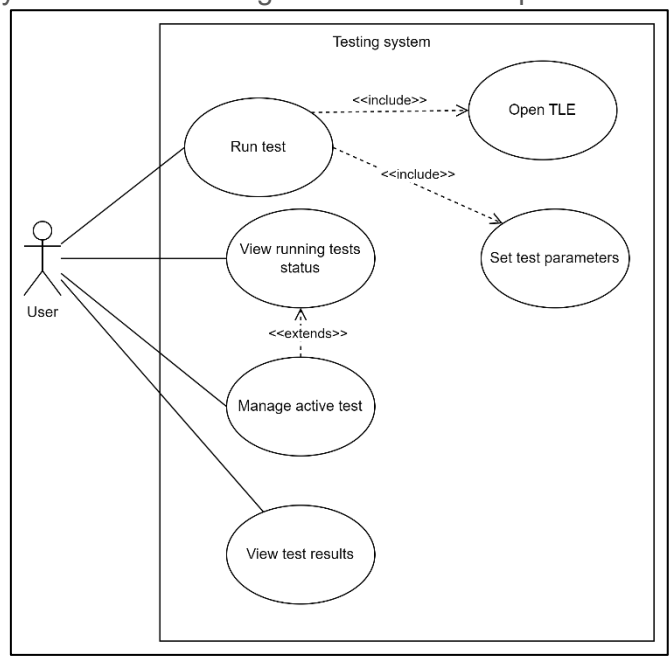
The cost of this ranges between 5,100 dollars to 11,700 dollars, depends if we already have their SDK license or not.

Image 4 **EnduroSat's Onboard Computer Type 1[12]**

4.2.3. The System

In order to test the algorithms on a dedicated on-board computer we needed to create a system fitting to run such tests, for each test we need to start by creating the data set for the test, the data set is composed of 3 parts, we need data in the points of time for ANCAS, CATCH and lastly, we want to find out what the actual TCA is to compare the algorithms results to. In order to find out the actual TCA we need to run the propagator with small time steps using the algorithm we described previously **[4.2.1.]**. The algorithm we described is not a good fit for the satellite on-board computer because it required a lot of memory and is computationally expensive, because of that, creating the data will be done on our own personal computer, and the testing equipment will be composed of 2 parts, the **Test Station**, our own computer, creating the data for each test and handling the test result. The second part is the Tested on-board computer, **Tested OBC**, connected via one of the possible communication methods, receiving the data from the test station, running the algorithms, checking the run time and memory used and sending the result back to the test station.

Diagram 5, System Use case diagram from the User point of view



4.2.3.1. Testing station system

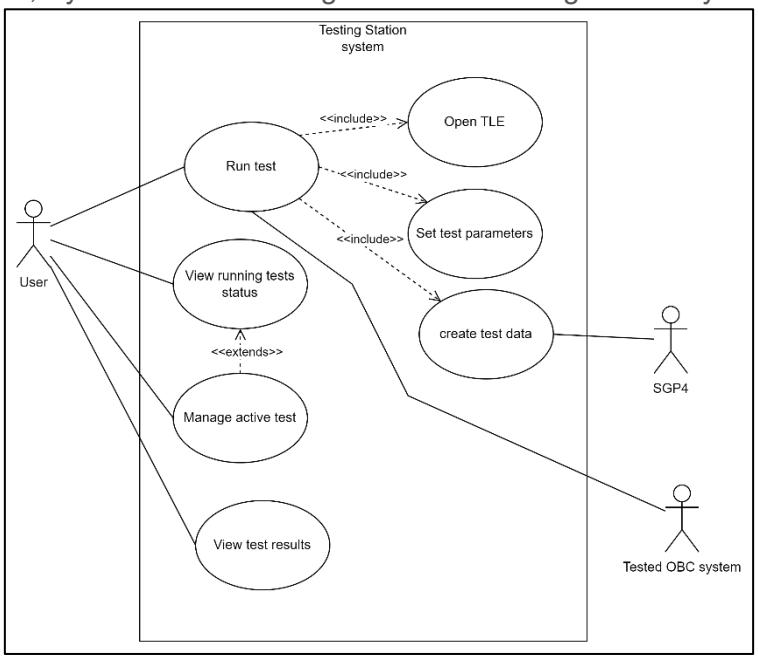
For handling a large number of tests with different data and different expected result, we will create test parameters to help us run large number of test variants using our framework without the need to change the code to do so.
Each test will be created using the following variables:

Variable name	Description	Possible values
Test data	2 TLE required for creating the data for the test	Any possible set of 2 TLE objects

Time interval size	The size of the time interval we want to test, in second. For example, to get a time interval of a single day we need to put 24*60*60.	Any positive integer, at least big enough for 1 iteration of the algorithms.
Number of iterations	To get more deterministic result we can run the same test a few times in a row and compare the result.	At least 1
Algorithm specific parameters	A set of parameters for the algorithms, for now we only need to set the value N for CATCH.	

Using these variables, the testing software will start by creating fitting data for the tested on-board computer, sending the data to the computer with the appropriate command and while waiting for the result, the testing station will calculate the actual TCA. After finishing the calculation and getting the test result from the tested on-board computer, the station will analyze the result and save it into our result data set. We will add the ability to configure sequence of test and leave the station to run all of them.

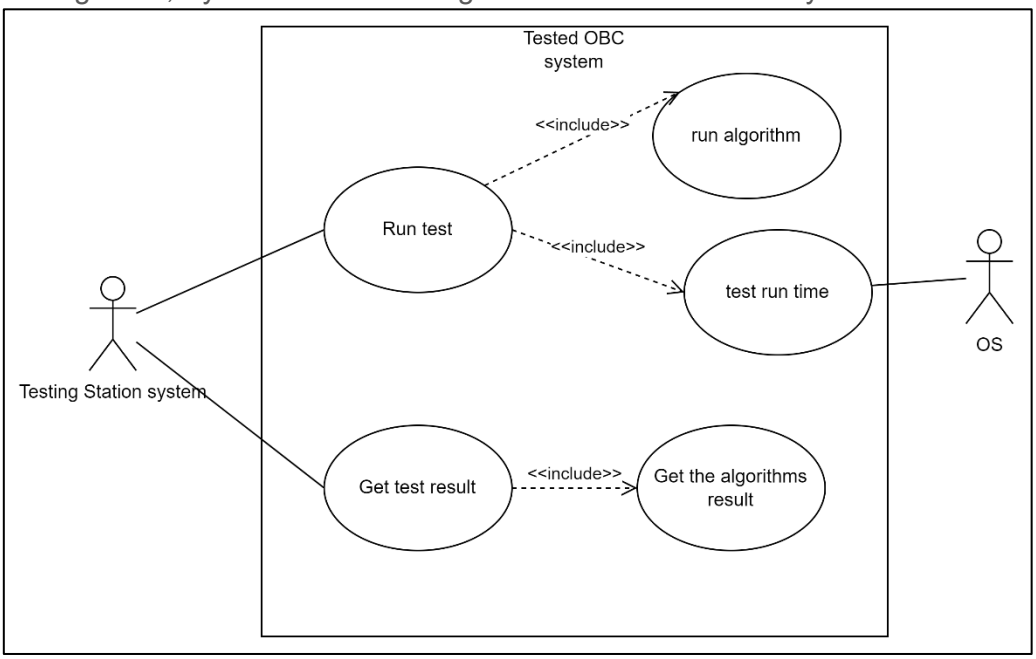
Diagram 6, System Use case diagram for the Testing Station system



4.2.3.2. Tested on-board computer system

On the other side, the software running on the tested on-board computer (OBC) will be listening to the communication channel, after receiving a command from the testing station following by a data set the system will run the test. For each algorithm starting the clock than running the algorithm with the given data set then stopping the clock and calculating the run time, repeating the test the number of times we got in the command from the test station, and for each test saving the result. After we finish running all the tests, we send back the result and wait for the next command.

Diagram 7, System Use case diagram for the Tested OBC system



4.2.3.3. Communication channel

The communication type is depended on the OBC type, each OBC can come with different communication ports and we can choose to use most of them. Our system implementation won't rely on the communication type. By writing in C++, we can use Abstraction and Inheritance to implement the system without relying on the communication type and being able to support a variety of different communication types easily. By creating a general interface between the tested OBC and the testing system, creating agreed on messaged for sending Test Command with Data from the station to the OBC, and replying with Test Result, we can create the communication protocol we need without relying on the communication type.

The communication type we would hope to use is USB 3.0, providing fast data transfer and can be found on many OBCs and on any personal computer. If it won't be possible, we can use other communication types such as Ethernet or even some of the serial protocols (RS485 and so on). We prefer to use the connections already available on our computer to save the need to get specialize adapters and drivers.

4.2.3.4. Communication protocol

For our communication protocol we created simple message fitting to use in our system, we need a Command message from the test station to the tested OBC, a Result message from the tested OBC to the testing system and an Error message. Each message is with the same following structure:

Byte number	Field name
0-1	Op Code

2-5	Data size
6-…	Data

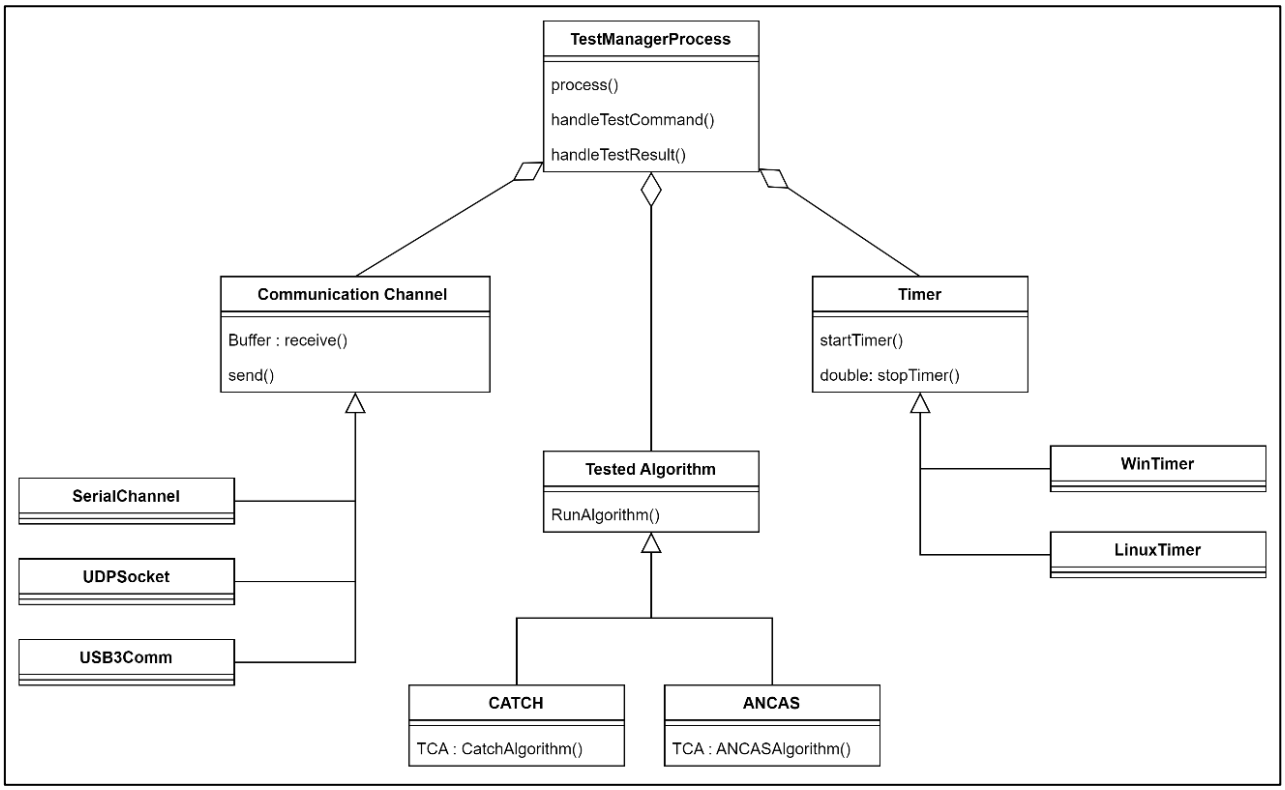
Each message will send a different type of data, the Command will start with the test variables following with the test data, the result will be a table of result, one row for each test\test iteration and the error will contain an error code.

4.2.3.5. OBC system Implementation

The OBC test system is composed of 3 parts. the first is the algorithms we want to test, the versions we implemented already or any other variations we will create in the future. The second part is the communication with the test station. The communication channel implementation is OBC and operating system depended and we will have to implement specifically for the OBC we want to test. The third and last part is the part that manage it all, run tests and create the result.

Diagram 8, the Tested OBC system class diagram

The communication channel is depended on both the operating system and the communication type and protocol we use. The timer and reading the clock in order to create a timer is depended on the operating system.



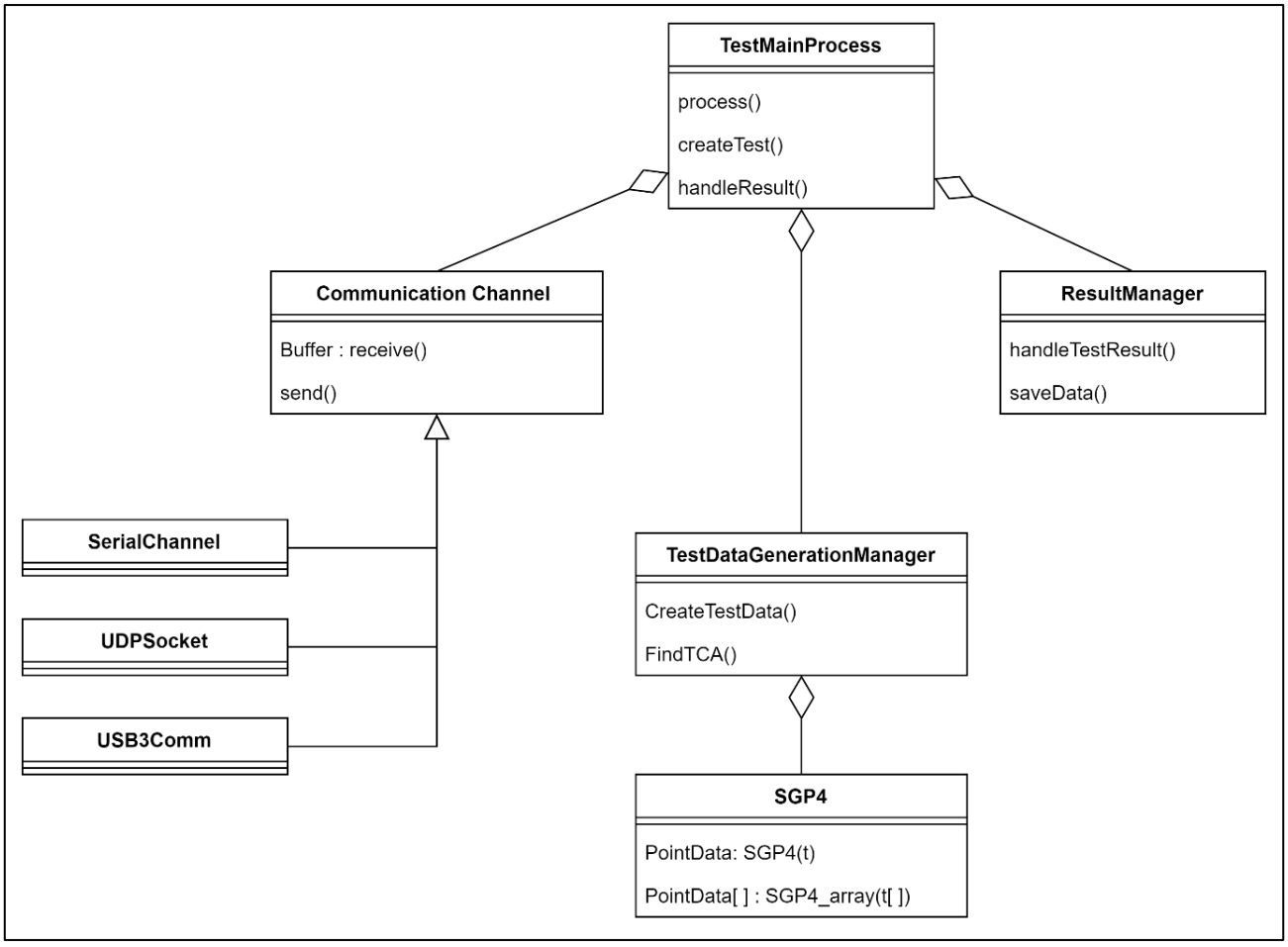
4.2.3.6. Test station system implementation

The test station system process is starting by loading a specific test variables and data, the test initial data is just two orbiting objects TLE data sets. After loading

the data the system need to create the actual test data, send the test data and the relevant command and start the calculation for the actual results using SGP4 with small time-steps simultaneously. The test station is implemented on a personal computer, and we are planning to implement it on the windows OS. We can divide the system to 4 parts, starting with the main process, managing the test and result and controlling the system. The second part is the test data generation, using the propagator we create data set for the algorithms and calculating the actual minimal distance for comparing the algorithm's result. The third part is our own data handling, we need to save each test result, calculate the size of the error and so on. The last part is the communication channel, communicating with the tested OBC system, sending data and receiving tests results.

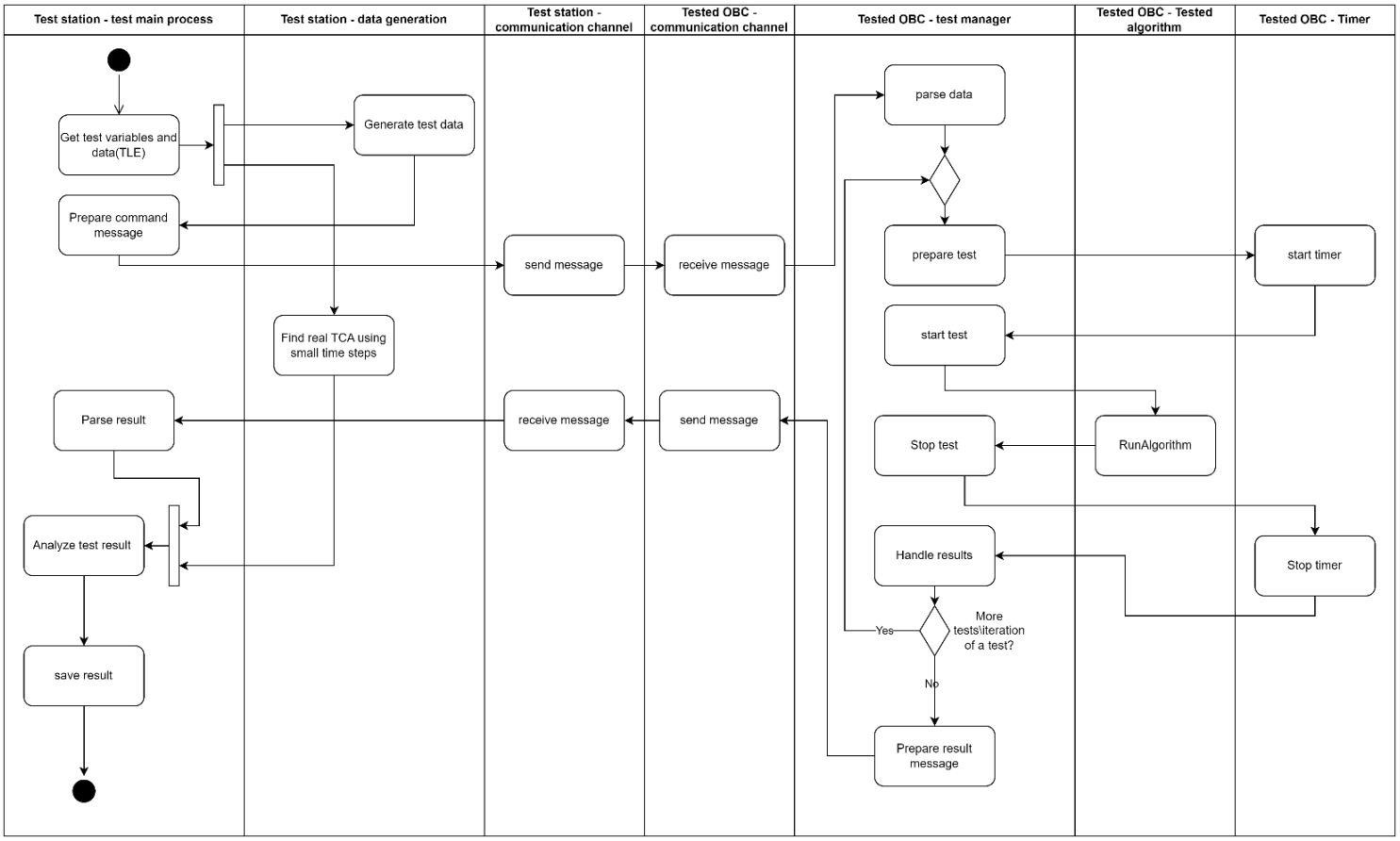
Diagram 9, Test station system class diagram

The communication channel is depended only on the communication type and protocol we use.



4.2.3.7. Full system activity diagram

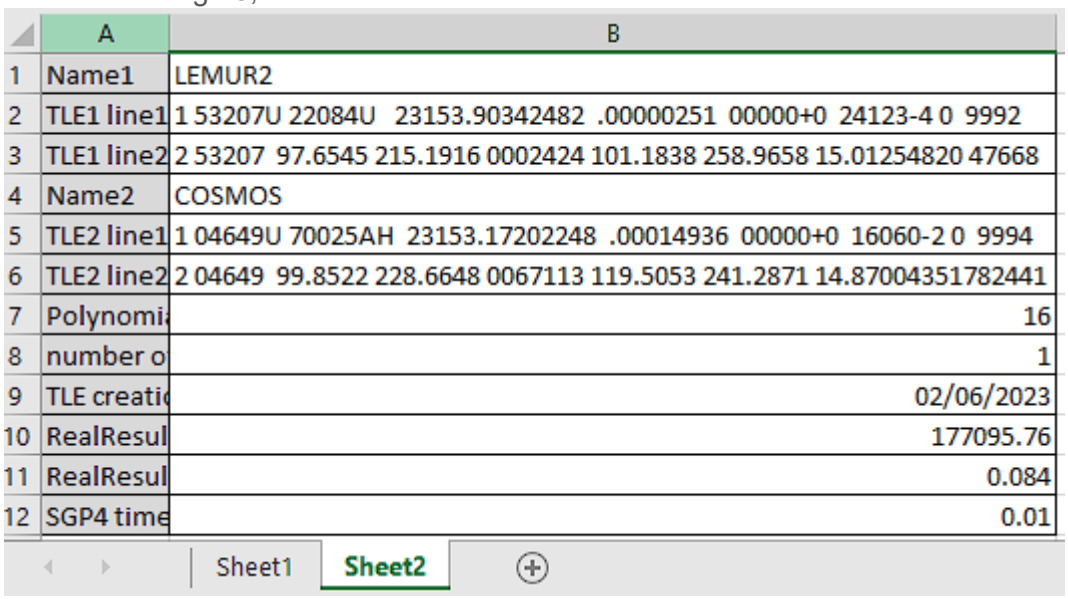
Diagram 10, full system activity diagram



4.2.4. Result data set

For each test we run on the system, a set of result is created, analyzing the result and displaying the data appropriately is an important part of our project, displaying the result as part of our GUI is described in the following paragraph. Each set of result start with the test data, including 2 TLEs, the names of the satellites, CATCH polynomial degree, number of iterations, creation date for the TLEs, the SGP4 time step size for finding the real TCA (the accuracy), the real result and the tests result. The test result for each iteration includes ANCAS result and CATCH result with both run time and the number of points tested. The data will be saved in an CSV format in a result folder and can be opened manually or via the GUI.

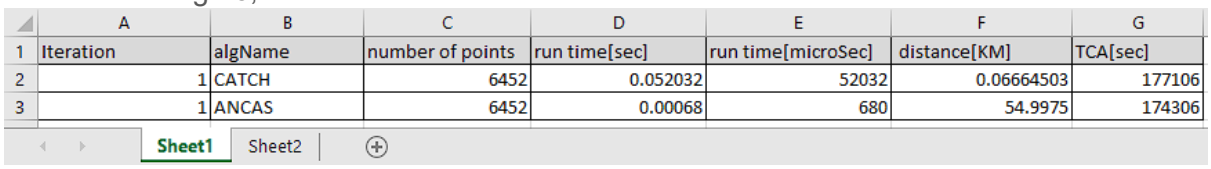
Image 5, test data table



	A	B
1	Name1	LEMUR2
2	TLE1 line1	1 53207U 22084U 23153.90342482 .00000251 00000+0 24123-4 0 9992
3	TLE1 line2	2 53207 97.6545 215.1916 0002424 101.1838 258.9658 15.01254820 47668
4	Name2	COSMOS
5	TLE2 line1	1 04649U 70025AH 23153.17202248 .00014936 00000+0 16060-2 0 9994
6	TLE2 line2	2 04649 99.8522 228.6648 0067113 119.5053 241.2871 14.87004351782441
7	Polynomi	16
8	number o	1
9	TLE creati	02/06/2023
10	RealResul	177095.76
11	RealResul	0.084
12	SGP4 time	0.01

Sheet1 Sheet2

Image 6, test result table



	A	B	C	D	E	F	G
1	Iteration	algName	number of points	run time[sec]	run time[microSec]	distance[KM]	TCA[sec]
2	1	CATCH	6452	0.052032	52032	0.06664503	177106
3	1	ANCAS	6452	0.00068	680	54.9975	174306

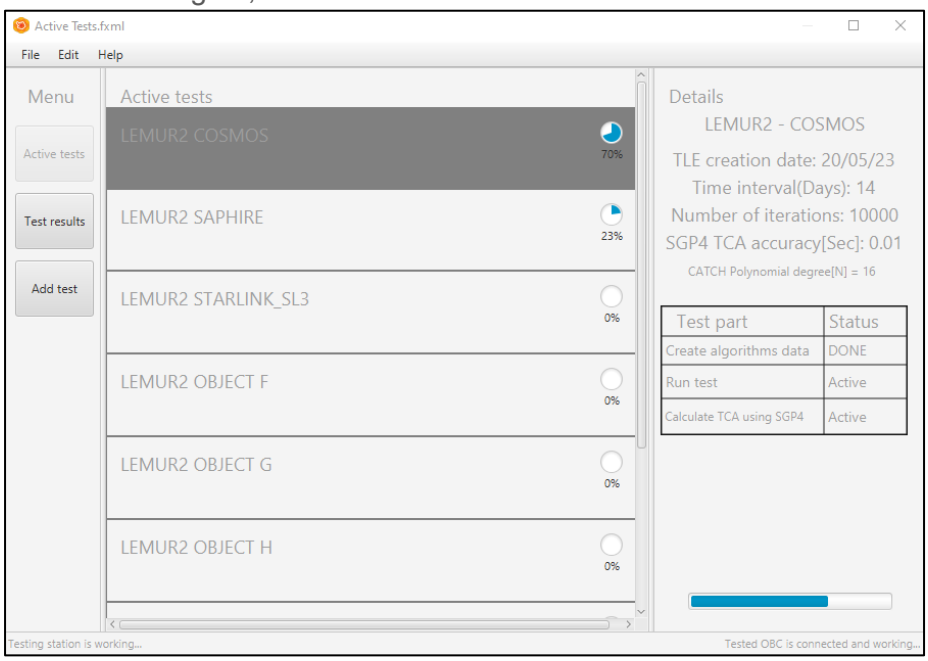
Sheet1 Sheet2

4.2.5. User interface

For the Testing Station, running on the user computer, there is a simple user interface including windows for viewing the active tests, one for creatin a new test, one for viewing the finished tests that can lead to the test result view.

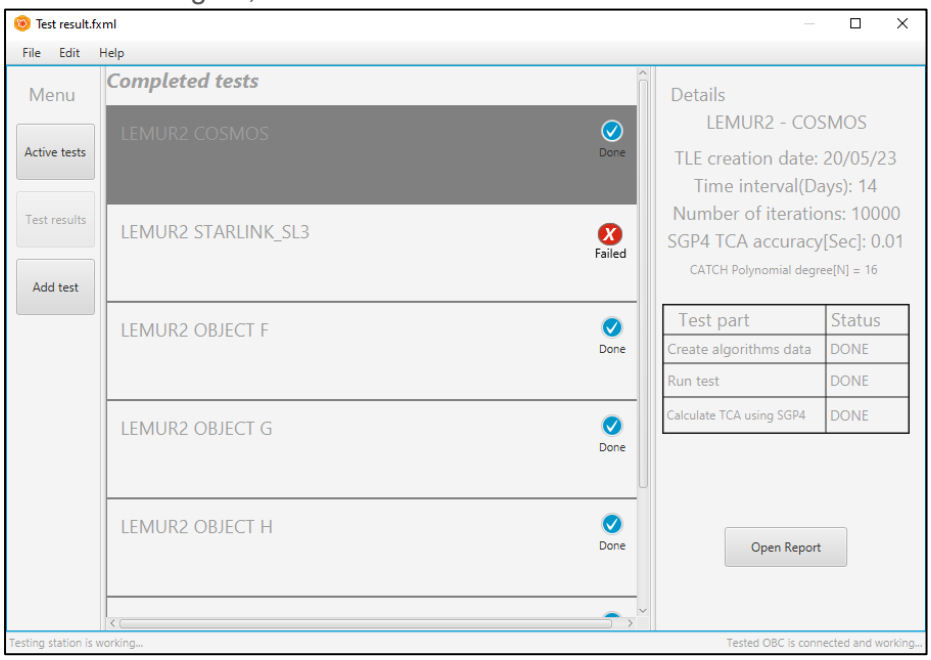
4.2.5.1. Active tests view

Image 7, active test window



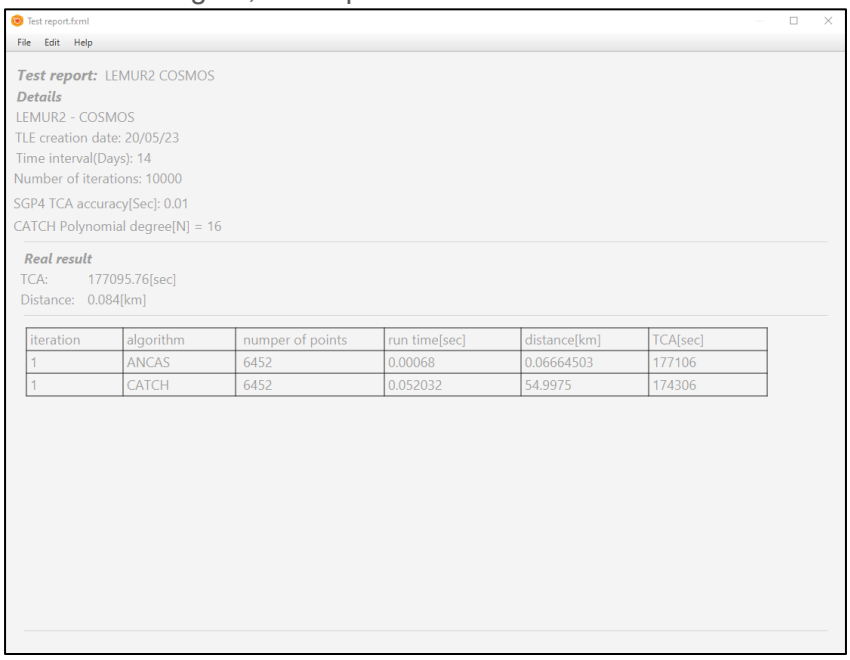
4.2.5.2. Test result window

Image 8, test results window



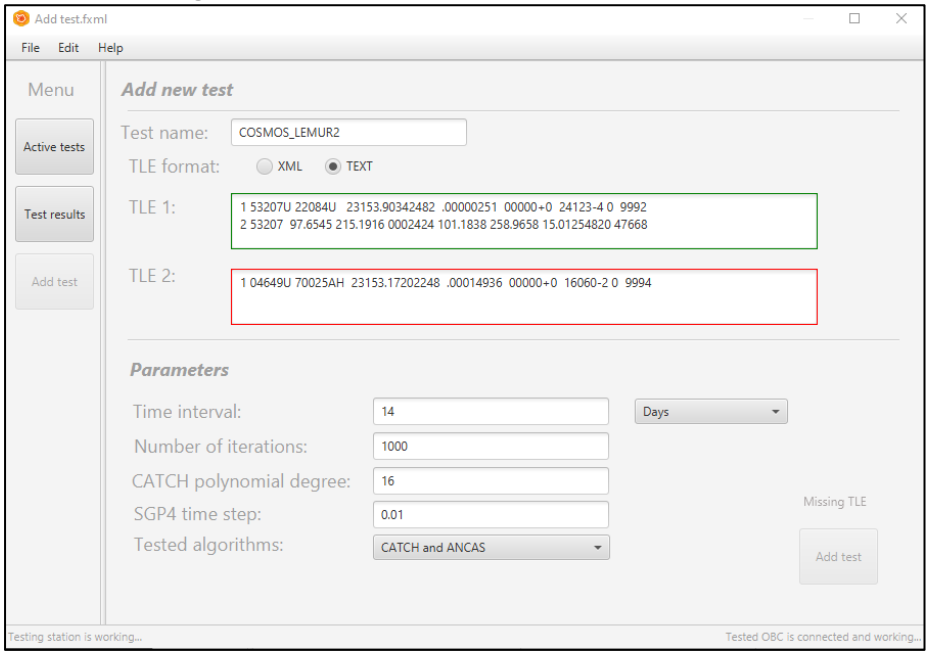
4.2.5.3. Test result report

Image 9, test report window



4.2.5.4. Add test window

Image 10, add test window



5. Evaluation/Verification Plan

5.1. Verification plan

For the verification plan we decided to go with a bottom-up approach, for each feature or part of the system we develop, we will start with creating a fitting unit tests. Then we can test our system locally simulating the inputs and outputs, and finally creating a new version and running it on the OBC we are working with.

5.2. Unit testing

Implementing unit testing using Google Test **[11]**. The GTest and GMock libraries allow us to create unit tests for C++. We are planning to create unit tests for each software component, using mock objects to control the input and outputs, and create test cases for the different possible input types and values. If needed, refactor the code, fix bugs and create save code.

For each software unit, perform the following general tests (and more if needed):

#	Test description	Expected result
1	Error in the input data, wrong format, wrong size, missing data (depended on the type of input).	Handle the error. Return error code or throw an exception we expect to get.
2	Null pointers/data. For each object that can be Null, create an appropriate test	Handle the error. Return error code or throw an exception we expect to get.
3	MSS (main success scenario), run the unit with correct input values and check the result.	Correct output.
4	Variations. Test the possible input variations. (Depending on the type of input)	Using code coverage tools, all the tested unit/ function should be covered.

5.3. Testing the system locally, simulations

For each part of our system, the testing station system and the tested OBC system, we can implement simple simulation by replacing the parts managing the communication with a local implementation and running the program independently from the other part.

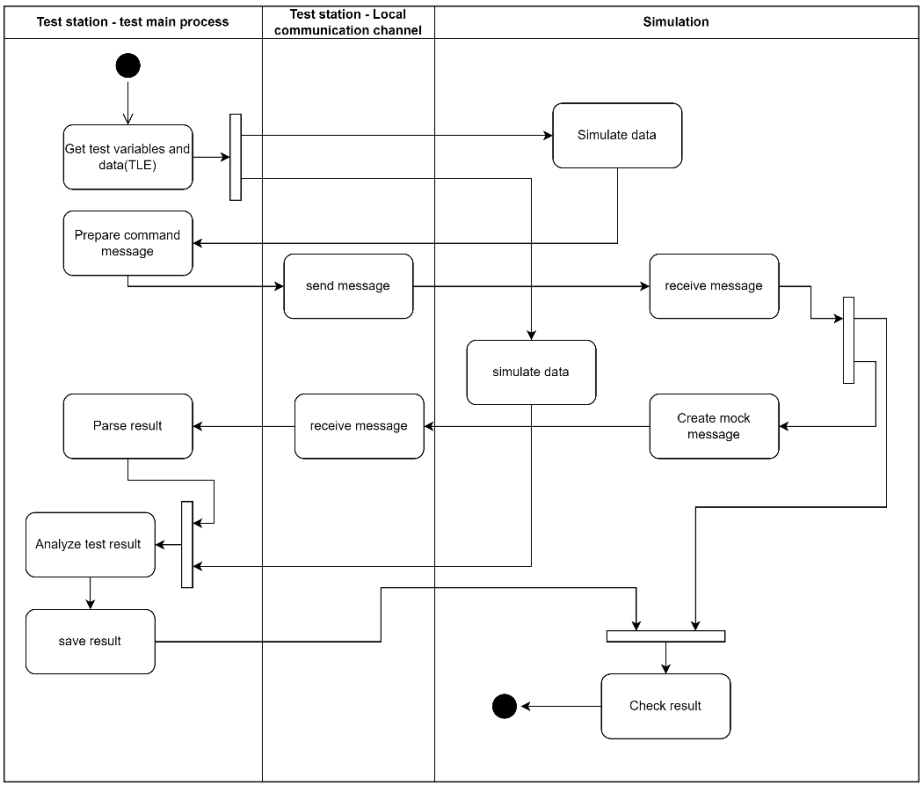
5.3.1. Simulating communication channels locally

There are many ways to simulate the communication channel locally between 2 programs on the same computers and we can use different way for different communication types. For ethernet connection, using TCP/UDP, we can simulate the connection using a **loopback** connection using tools available with Windows, for serial communication we can use some of the available simulator like **Virtual COM Port Driver [13]**.

5.3.2. Test-Station local simulation

For simulating the environment of the test station program, we will create a simple simulation, generating a mock data set instead of SGP4, simulating the communication locally and handling the output we get from the program to see the results. By running the program, we want to test that given the mock data set and the test variable we put in, the program creates the correct message and try to send them via the communication channel, handle the returning message our simulation will mock and send and eventually save the correct output. Doing so we can check the full capabilities of the test station program.

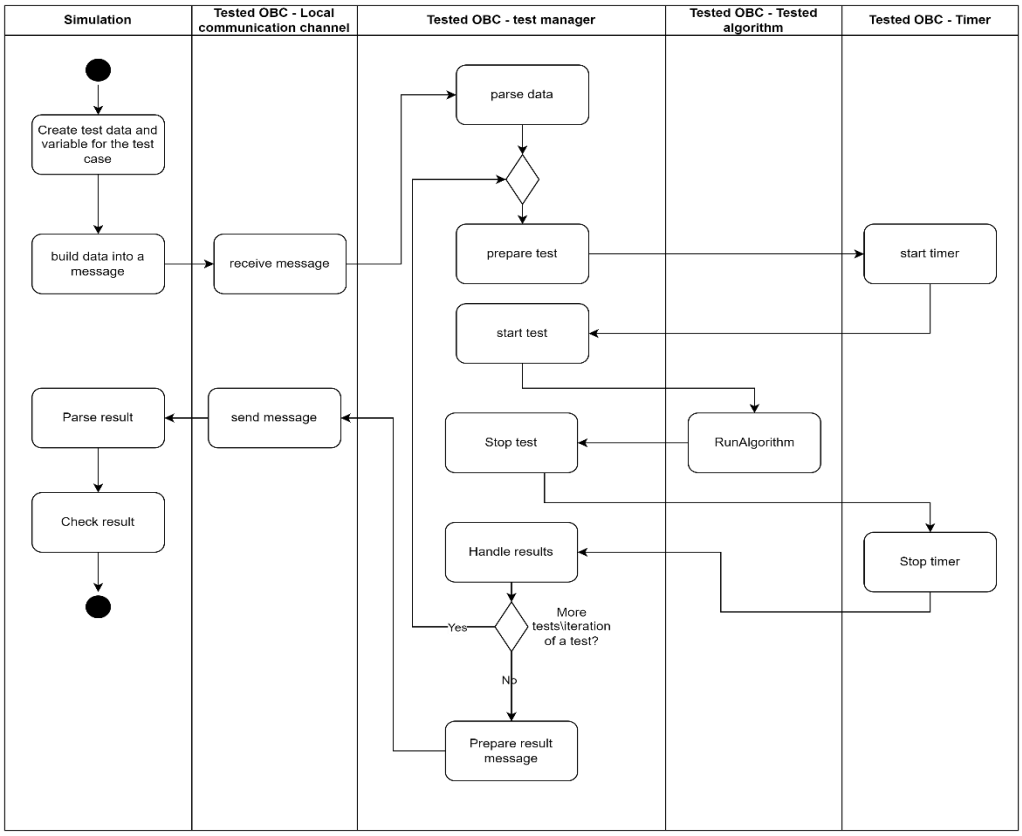
Diagram 11, test-station local simulation activity diagram



5.3.3. Tested OBC local simulation

For this part, we want to test our system logic and communication protocols without relying on the specific system we work with eventually. We want to simulate the environment for testing proposes on our personal computer. Because we use abstraction, we can create a different implementation of the communication channel and timer fitting for our system and simulate receiving and sending data via a local communication solution to simulate the full environment and test the program logic. We start the test by sending a mock message and data we create, letting the program run and testing the result we get back, seeing if it's the expected result.

Diagram 12, tested-OBC local simulation activity diagram



5.4. Test cases

We run the following test cases for both the Testing Station and the Tested OBC on the local simulation and on the actual system.

5.4.1. Test-Station test cases

#	Test description	Expected result
1	Error in the input data: wrong input data (TLE, either one of the 2 required TLE or both)	No experiment is initiated, error message to the user.
2	Error in the input data: Missing test variable (Number of iterations, CATCH polynomial degree)	No experiment is initiated, error message to the user.
3	MSS (main success scenario), correct input values (TLE, variables).	Send correct message and data, save the correct result received from the simulation.
4	Communication error: receive error message from the Tested OBC (simulated).	Try to send the data again, display error message to the user on repetitive errors.
5	Testing the User interface: save active tests on closing the window, continue test on the next activation.	

6	Testing the User interface: testing the displayed system status.	
7	Communication error: no communication with the Tested OBC.	Display error message to the user.

5.4.2. Tested-OBC test cases

#	Test description	Expected result
1	Error in the input data: missing\out of bound variables.	Send back error message.
2	Error in the input data: missing data points (missing some of the values for a points in time data composed from r1,v1,r2,v2,t)	Send back error message.
3	MSS (main success scenario) variations (run ANCAS/CATCH), correct input values (test variables, test data set).	Send back correct result.
4	Communication error: receive error message from the Testing station.	Try to send the result message again, give up after repetitive errors.
5	Communication error: no communication from the Testing station	Wait for the station response.

6. REFERENCES:

[1] Denenberg, Elad. "Satellite closest approach calculation through Chebyshev Proxy Polynomials."
https://www.researchgate.net/publication/338609111_Satellite_Closest_Approach_Calculation_Through_Chebyshev_Proxy_Polynomials

[2] Alfano, S. "Determining Satellite Close Approaches-Part II."
https://www.researchgate.net/publication/252559100_Determining_satellite_close_approaches_part_II

[3] NASA on orbital debris:
https://www.nasa.gov/mission_pages/station/news/orbital_debris.html

[4] SGP4 python library :
https://pypi.org/project/sgp4/

[5] NASA Procedural Requirements for Limiting Orbital Debris:
https://www.iadc-home.org/references/pdfview/id/74

[6] UN Space Debris Mitigation Guidelines:
https://www.iadc-home.org/references/pdfview/id/75

[7] CelesTrak:
https://celestrak.org/

[8] Satellite Orbital Conjunction Reports Assessing Threatening Encounters in Space:
https://celestrak.org/SOCRATES/

[9] The Kessler Syndrome: Implications to Future Space operations:
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=227655e022441d1379dfdc395173ed2e776d54ee

[10] SGP4: Revisiting Spacetrack Report :
https://celestrak.org/publications/AIAA/2006-6753/AIAA-2006-6753.pdf

[11]Google test framework
https://github.com/google/googletest

[12] EnduroSat's Onboard Computer Type I:
https://www.endurosat.com/cubesat-store/cubesat-obc/onboard-computer-obc/

[13] Virtual COM Port Driver:
https://www.virtual-serial-port.org/

[14] Freethink article about space debris:
https://www.freethink.com/space/space-debris-15000mph