Capstone Project Phase A

SATELLITES, DISTANCE APPROXIMATION, FEASIBILITY TEST, COLLISION DETECTION, SATELLITE HARDWARE, ALGORITHMS IMPLEMENTATION

24-2-D-29

Supervisor: Mr. Ilya Zeldner
Shir Cohen 316216340
Cohensh96@gmail.com
Yotam Aharon 208608448
yotamah14@gmail.com
Project Repository

BRAUDE
College of Engineering, karmiel

Table of Contents

Abstract

Our project is the continuation of project titled "Feasibility Analysis and Performance Testing of Collision Detection Algorithms for Satellites" **[10]**, which focused on developing an algorithm for calculating the minimal distance between two objects in space. In our project, we transition from the theoretical phase to the practical phase, optimizing and implementing SSA algorithms on actual hardware on satellite **On-Board Computer (OBC)**, following feasibility test results. The project has two goals, the first goal is to conduct a thorough investigation of various satellite OBCs and select the most suitable satellite OBC for algorithms implementation and future testing, the second goal is to perform simulations on the selected OBC, investigate and evaluate the performance of these algorithms on satellite OBC within a simulated space environment. The project is based on Dr. Elad Denenberg's research papers that introduced the algorithms **[2][3].**

Keywords

Satellite, Space debris, Minimal distance, Approximations algorithms, Feasibility test, Collision detection, Algorithm testing, Distance approximation, Orbiting objects, Satellite On-Board Computer (OBC), TS7553 on-board computer, Satellite hardware, Algorithms implementation.

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 **[15]** and the shift to cluster of smaller satellites instead of a single big one **[2]** 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 computationally cheap and fast enough to run on the satellite's own OBC.

At this stage of our project, we are actively conducting tests and implementing the algorithms on the selected satellite board. We have conducted extensive research to identify an optimal satellite board for evaluating these algorithms, even though the initial selection criteria did not specifically address space-related conditions. Our investigation has focused on potential board dimensions, particularly utilizing the **ISIS OBC** **[13]** satellite board data, selected by Dr. Elad Danenberg, with whom we are collaborating closely. The primary objective of this phase is to validate the feasibility of these algorithms by demonstrating their computational speed and memory efficiency in an environment that simulates the limited processing power and memory constraints typical of a real satellite system. Dr. Elad Danenberg, the developer of the **Conjunction Assessment Through Chebyshev Polynomials (CATCH)** algorithm **[2]** and the **SBO-ANCAS** algorithm **[3]**, is relying on our expertise to ensure that these systems can autonomously calculate potential collisions, necessitating the development of rapid TCA algorithms. To meet these stringent requirements, we are leveraging advanced approximations, including the CATCH **[2]**, SBO-ANCAS **[3]**, and **Alfano/Negron Close Approach Software (ANCAS)** algorithms **[1]**, all of which are rigorously tested to ensure suitability for deployment in autonomous and board-realized satellite operations The chosen satellite for performance and feasibility tests.

2. General Description

2.1. Background and Related Work

2.1.1. Space debris

There are around 27,000 **[15]** object in Earth orbit, the number of objects is predicted to grow even if we stop sending new satellite into space **[4]** 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 **[16][25]** 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 **[8]** by sharing relevant knowledge for free and tracking satellites, in a program called "Satellite Orbital Conjunction Reports Assessing Threatening Encounters in Space" (SOCRATES) **[18]** 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 **[11]**

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

2.1.2. 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 **[2]** 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.

2.1.3. 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) [5]**, 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 **[16][25]**. 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.

To our propagator has two tasks, the first is creating the data for each of the algorithms runs. Given a set of TLE from the user we can create a set of point in time for two satellites and run the algorithms with it. The second task is using a propagator as part of the SBO-ANCAS algorithm. SBO-ANCAS needs to sample new points as part of the algorithms so a propagator is needed.

2.1.4. The Algorithms

In this project, we utilize three algorithms that were previously implemented and tested by the prior team **[10]**. Each of these algorithms can be used to calculate the TCA. The following sections provide a detailed description of the methodologies and principles underlying these algorithms. By relying on these established algorithms, we ensure continuity in the computational approach and base on the preceding research efforts of the prior team. The following is a description of the algorithms.

2.1.4.1. ANCAS

The first algorithm, ANCAS **[1]** uses cubic polynomial as an approximation of a function over an interval. Given n points in time and the respective location and velocity

Input: $p[n], t[n]$
Output: $TCA + r_{TCA}$
$r_{TCA} = inf$
$t_{TCA} = inf$
for each set of 4 points **do:**
  Map the time points $t[1-4]$ to $\tau_0 =\leq \tau_1, \tau_2 \leq \tau_3 = 1$ on the interval $[0,1]$
  Calculate $\dot{f}(t), f_x, f_y, f_z$ using **[[1], Eq.2/5]** with the points $p[1-4]$
  Fit cubic polynomial to $\dot{f}(t)$ according to **[[1], Eq.1f-1j]** over $[0,1]$
  Find the cubic polynomial real roots in the interval $[0,1]$
  Fit cubic polynomials for f_x, f_y, f_z in the interval $[0,1]$
  for each root t^* **do:**
    calculate the distance $r(t^*)$ using **[[1], Eq.6]**
    if $r(t^*) < r_{TCA}$**:**
      $r_{TCA} = r(t^*)$
      $t_{TCA} = t^*$
    end
  end
end

vectors for 2 objects, we can find the TCA by:
Algorithm 1: ANCAS on n points, (the original algorithms description can be found at **[1]**)

The cubic polynomial coefficients calculations described in article **[1]**.
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 on a large time 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.

Variables and notation in ANCAS Algorithm:
To enhance the understanding of the ANCAS algorithm, here is a breakdown of the role of each variable, including the input, output, and other diffrenent notations used in the algorithm:

Input Variables:

- **p[n]:** This represents the position and velocity data of two orbiting objects at n distinct time points. Each element in p[n] is a vector that contains the 3D position and velocity information for the two objects involved in the calculation. This serves as the primary input data for the algorithm.

- **t[n]:** A corresponding time array that holds the timestamps associated with each data point in p[n]. This array represents the specific moments in time at which the positions and velocities of the two objects are known.These time points serve as the domain over which the polynomials are fitted. The algorithm works by fitting cubic polynomials to p[n] over the interval defined by t[n].

Output Variables:

- **t_TCA:** The time of closest approach (TCA), which is the time when the minimum distance between the two objects occurs. This is the primary result of the algorithm.
- **r_TCA:** The minimal distance between the two objects at the time of closest approach, calculated as the Euclidean distance between their positions at time t_TCA.

Other notations:

- The function $\dot{f}(t)$ represents the rate of change of the distance between the two objects over time.
- The components f_x, f_y, f_z represent the relative position between the objects in the 3D coordinate system, calculated using the data points.

2.1.4.2. SBO-ANCAS

The second algorithm, SBO-ANCAS **[1]** is based on the ANCAS algorithm, still using cubic polynomial as an approximation of a function over an interval. But uses additional points to get better results. Given an initial set of n points in time, the respective location and velocity vectors for 2 objects, tolerance in time and tolerance in distance we can find the TCA by:

Input: $p[n], t[n], TOL_d, TOL_t$
Output: $t_{TCA} + r_{TCA}$
$r_{TCA} = inf$
$t_{TCA} = inf$
for each set of 4 points **do:**
  $\mathrm{t_{new}} = \mathrm{t_1}, \mathrm{t_2}, \mathrm{t_3}, \mathrm{t_4}$
  Do
    Map the time points $t[1-4]$ to $\tau_0 =\leq \tau_1, \tau_2 \leq \tau_3 = 1$ on the interval $[0,1]$
    Calculate $\dot{f}(t), f_x, f_y, f_z$ using **[[1], Eq.2/5]** with the points $p[1-4]$
    Fit cubic polynomial to $\dot{f}(t)$ according to **[[1], Eq.1f-1j]** over $[0,1]$
    Find the cubic polynomial real roots in the interval $[0,1]$
    Fit cubic polynomials for f_x, f_y, f_z in the interval $[0,1]$
    for each root $t*$ **do:**
      calculate the distance r at $t*$ using **[[1], Eq.6]**
      if $r < r_{min}$ **:**
        $r_{min} = r$
        $t_{min} = t*$
      end
    end
    Sample r_d and $\dot{r}_d$ at t_{min} using a propagator
    $\mathrm{t_{new}} = \mathrm{t_{new}} \setminus \left(t_j \mid \max\left(\left|t_j - t_{min}\right|\right) \cup t_{min}\right.$

  While $| r_{min} - \| r_d(t_{min})\| | > TOL_d$ OR $\max\left(\left|t_j - t_{min}\right|\right) > TOL_t$
  $\mathrm{r_{tca}} = \| \mathrm{r_d}(t_{min})\|$
  $\mathrm{t_{tca}} = t_{min}$
end

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

The cubic polynomial coefficients calculations described in article **[1].**
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. In each iteration after finding the minimum point $\mathrm{t_{min}}$ we use the propagator to sample the location and velocity vectors at $\mathrm{t_{min}}$, we use the new and more accurate values and create a polynomial to find a more accurate minimum distance and so on until we reach the desired tolerance. This algorithm can give the best results, we can get the same results as checking every point with a small time-steps if we use small enough tolerance but with high cost

in run time. SBO-ANCAS have an additional loop in each iteration and sampling points with the propagator is an expensive operation.

Variables and notation in SBO-ANCAS Algorithm:
The SBO-ANCAS algorithm is an enhancement of ANCAS, incorporating iterative refinement and additional sampling to improve accuracy at the cost of longer runtime.To enhance the understanding of the SBO-ANCAS algorithm, here is a breakdown of the role of each variable, including the input, output, and other diffrenent notations used in the algorithm:
Input Variables:

- **p[n]:** The position and velocity vectors of two orbiting objects at n time points. This input provides the data needed to calculate the time and distance of closest approach between the objects.
- **t[n]:** The corresponding array of time values for each data point in p[n]. Each element represents the timestamp for the position and velocity of the objects.
- **Tolerance in time (TOL_t):** A small threshold that defines how close two time points must be for the algorithm to stop refining them. This determines the precision of the closest approach time.
- **Tolerance in distance (TOL_d):** A small threshold that specifies the acceptable error in the calculated minimal distance. It ensures the algorithm stops refining once the distance error is within this tolerance.

Output Variables:

- $\mathbf{t_{tca}}$: The time of closest approach (TCA) when the two objects are at their minimum distance.
- $\mathbf{r_{tca}}$: The smallest distance between the two objects during their encounter, calculated at $\mathbf{t_{tca}}$.

Other notations:

- $\mathbf{t_{new}}$: This variable represents the newly sampled time points introduced during each iteration of the SBO-ANCAS algorithm.
  In each iteration of the algorithm, once the cubic polynomial fitting and root finding are complete, t_{new} is sampled using the propagator based on the previous approximate solution, and the algorithm recalculates the positions and distances at these new points. This iterative refinement continues until the tolerance in time and distance is satisfied.
- $\boldsymbol{r_d}$: represents the relative distance between the two orbiting objects at a given time. It is the Euclidean distance between their position vectors in three-dimensional space. The goal of the algorithm is to find the TCA, which is the time when $\boldsymbol{r_d}$ is minimized. During each iteration of the algorithm, $\boldsymbol{r_d}$ is calculated for various candidate times (including the initial and newly sampled time points) to determine when the two objects are at their closest.
- $\boldsymbol{\dot{r}_d}$: represents the rate of change of the relative distance between the two objects with respect to time. In other words, it is the time derivative of the relative distance function $r_d(t)$.

- The algorithm uses $\dot{\boldsymbol{r}}_{\boldsymbol{d}}$ to help find candidate points for the closest approach. The critical points, where the distance between the objects could be minimal, occur when $\dot{\boldsymbol{r}}_{\boldsymbol{d}}(\boldsymbol{t}) = \boldsymbol{0}$. The algorithm fits a cubic polynomial to approximate $\dot{\boldsymbol{r}}_{\boldsymbol{d}}$, and then finds its roots (where the derivative is zero) to identify potential times for the TCA.

2.1.4.3. CATCH

The third algorithm, CATCH **[2]**, uses **Chebyshev Proxy Polynomial (CPP)** 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 3: CATCH the original algorithms description can be found at **[[2],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 **[[2], 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 **[[2], 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 **[[2],Eq.18]** and the eigen values of this matrix are the polynomial roots. Using the roots, we found and creating CPP for $p(t)_{f_x}, p(t)_{fy}, p(t)_{fz}$ 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 **[[2],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.

Variables and notation in CATCH Algorithm:
The CATCH algorithm uses Chebyshev Proxy Polynomials (CPP) for more precise approximations of the distance between orbiting objects. It is generally more computationally expensive than SBO-ANCAS, but it can deliver higher accuracy.
Here is a breakdown of the role of each variable, including the input, output, and other diffrenent notations used in the algorithm:
Input Variables:

- $\boldsymbol{p_1[]}$**:** This array contains the **position vectors** of the first object (e.g., satellite) at multiple time points. Each element in $\boldsymbol{p_1[]}$ represents the 3D position of the object at a specific time, giving its location in space.
- $\boldsymbol{p_2[]}$**:** This array contains the **position vectors** of the second object (e.g., another satellite or debris) at multiple time points. Like $\boldsymbol{p_1[]}$, each element represents the 3D position of this second object at specific times.
- $\boldsymbol{t_{max}}$**: Maximum time interval** for which the algorithm will search for the closest approach between the two objects. It defines the range of time over which the CATCH algorithm looks for the Time of Closest Approach (TCA). It bounds the period in which the algorithm calculates the minimum distance. The algorithm works over this time interval to fit the Chebyshev polynomials and find roots.
- **Γ:** Initial estimate or threshold for the minimal distance between the two objects. It can also represent the minimal acceptable distance for conjunctions in certain scenarios. The algorithm refines its search for the TCA by continuously updating the estimated distance between the two objects.
- **Degree of the Chebyshev polynomial (N)**: A parameter that controls the degree of the polynomial approximation used in the algorithm. Higher degrees provide more accurate results but at the cost of increased computational complexity.

Output Variables:

- $\mathbf{t_{tca}}$: The TCA when the two objects are at their minimum distance.
- $\mathbf{r_{tca}}$: The smallest distance between the two objects during their encounter, calculated at $\mathbf{t_{tca}}$.

Other notations:

- $\boldsymbol{f_x, f_y, f_z}$: The relative position components in the x, y, and z directions, used to calculate the relative distance between the two objects.
- $\boldsymbol{\dot{f}(t)}$:The derivative of the relative distance function, used to identify critical points (minima or maxima) where the distance between the objects might be minimized (i.e., the TCA).
- $\boldsymbol{p_{\dot{f}}}$: The Chebyshev Proxy Polynomial coefficients, which approximate the position or distance functions over time.

- $\boldsymbol{a}$, $\boldsymbol{b}$: The time boundaries defining the interval over which the CATCH algorithm operates, typically from the start time $a = 0$ to the maximum time $b = t_{max}$.

2.2. 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 **[15]** 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 **[[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.

2.3. Algorithms Complexity

Time and space complexity analysis of the three algorithms that were previously implemented and tested by the prior team.

2.3.1. Algorithms Runtime Complexity

2.3.1.1. ANCAS Time Complexity

In ANCAS **[1],** 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 **[[1],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 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 **[[1], 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 $\left\lceil \frac{n-4}{3} \right\rceil + 1 = \left\lceil \frac{n-1}{3} \right\rceil \leq \frac{n}{3}$ iterations.

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

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

2.3.1.2. SBO-ANCAS Time Complexity

In SBO-ANCAS **[1]**, we are going over a set of n initial points, the outer loop 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 $\left\lceil \frac{n-4}{3} \right\rceil + 1 = \left\lceil \frac{n-1}{3} \right\rceil \leq \frac{n}{3}$ outer iterations.

In the inner loop we run until we reach the desire tolerance in distance and time, thus the number of inner iterations depends on the size of tolerance in distance, the size of tolerance in time, the error of the polynomial approximation, the change in relative distance in time between the 2 objects and the distance between the initial time points. For each inner iteration we use the propagator to sample a single point in time.

Let's start by looking at the tolerance in time condition for the inner loop, say we have 4 time points, $t_1 - t_4$, with the initial distance between 2 time points of $t_{distance}$, and tolerance in time TOL_t. To get to the desired tolerance we need the distance between t_m to the other points to be smaller than TOL_t. which means that at the last iteration we get:

$interval\ size_i = TOL_t \leq t_{4i} - t_{1i} \leq 2TOL_t$.

To find the worst-case scenario for the number of iterations we need to consider the smallest possible decrement in the total time interval per iteration. In the following example we can see that theoretically there is no limit to how many iterations we get.

We start with a set of 4 points, $t_1 - t_4$:

$t_{m1} = t_1 + \varepsilon \rightarrow t_{new} = \{t_1, t_{m1}, t_2, t_3\}$

$t_{m2} = t_3 - \varepsilon \rightarrow t_{new} = \{t_{m1}, t_2, t_{m2}, t_3\}$

$t_{m3} = t_{m1} + \varepsilon \rightarrow t_{new} = \{t_{m1}, t_{m3}, t_2, t_{m2}\}$

$t_{m4} = t_{m2} - \varepsilon \rightarrow t_{new} = \{t_{m3}, t_2, t_{m4}, t_{m2}\}$

And so on.

And if we look at the interval size in each step:

$interval\ size_0 = t_3 - t_1 = 2 * t_{distance}$

$interval\ size_1 = t_3 - t_1 - \varepsilon = 2 * t_{distance} - \varepsilon$

$interval\ size_2 = t_3 - t_1 - \varepsilon - \varepsilon = 2 * t_{distance} - 2 * \varepsilon$

...

$interval\ size_i = 2 * t_{distance} - i * \varepsilon$

And we continue until:

$interval\ size_k \leq TOL_t$

$2 * t_{distance} - k * \varepsilon \leq TOL_t$

$2 * t_{distance} - TOL_t \leq k * \varepsilon$

$$\frac{2 * t_{distance} - TOL_t}{\varepsilon} \leq k \rightarrow O\left(\frac{t_{distance} - TOL_t}{\varepsilon} * P\right)$$

where P = complexity of getting a single point from the propogator

We expect the distance between 2 points, $t_{distance}$, to be bigger than the tolerance and with a small enough $\varepsilon \rightarrow 0$ we get:

$$\lim_{\varepsilon \to \infty} \left(\frac{t_{distance} - TOL_t}{\varepsilon} * P\right) = \infty$$

Practically that not the case because there is a limit on how many small numbers we can fit between any set of 2 initial values, depending on the value of $t_{distance}$, the specific implementation and the precision of the variables. For example, if we use an IEEE 754 double-precision variable, the smallest possible value is about 5×10^{-324} so we will get a large but final number of iterations.

Let's look at the tolerance in distance, we compare two values of the distance in the same point in time. The first is the value we got from the polynomial approximation and the second is the value we got from the propagator. The different is because the error of the polynomial approximation. The closer the points in time will be, the smaller the change in distance will be and we can expect smaller errors. So, with a small enough time step we will reach the desired tolerance.

2.3.1.3. CATCH Time Complexity

In CATCH [2] 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 **[[2],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 **[2]**. 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 $\left(N_{deg}+1\right)\left(N_{deg}+1\right)$ 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: $O(n)$

2.3.2. Space complexity

The space complexity of the algorithms is the same. SBO-ANCAS*, 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 points for the input: $O(n+1) = O(n)$

*For SBO-ANCAS, we assume that the propogator uses a constant amount of memory, but this may not always be the case.

3. Expected Achievements

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

To achieve this goal, we expect to develop a complete testing system, as described in the Product section ahead, which will consist of two software components. One program, referred to as the Test Station, will run on a local computer, while the other, termed the Tested OBC, will operate on the satellite's OBC we will test with. The Test Station will be responsible for generating data sets, executing tests, and collecting performance data. The Tested OBC will receive the data, measure execution times, and execute the algorithms. The two programs need to be tested, proven and flexible enough to work with different algorithms implementations and data sets.

Upon completing the system implementation, we plan to conduct a series of experiments using diverse input sets and optimized versions of the algorithms. The objective of these experiments is to gather sufficient data to validate the feasibility of running TCA calculations on the satellite's OBC.

We want to finish the project with three key outcomes:

1.A fully functional testing system, comprising two components. The first part will run on the satellite's OBC, executing the algorithms, measuring run times, and transmitting the results. The second part will operate on a local computer, generating data for the algorithms, storing results, and analyzing the output from the first part. 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.A comprehensive set of experimental results, derived from testing a variety of input types, that will either confirm or refute the feasibility of executing TCA calculations on the satellite's OBC.

3.Optimized and reliable implementations of the CATCH algorithm, ready for integration into the satellite system when necessary.

4. Research / Engineering Process

4.1. Process

4.1.1. The project work flow

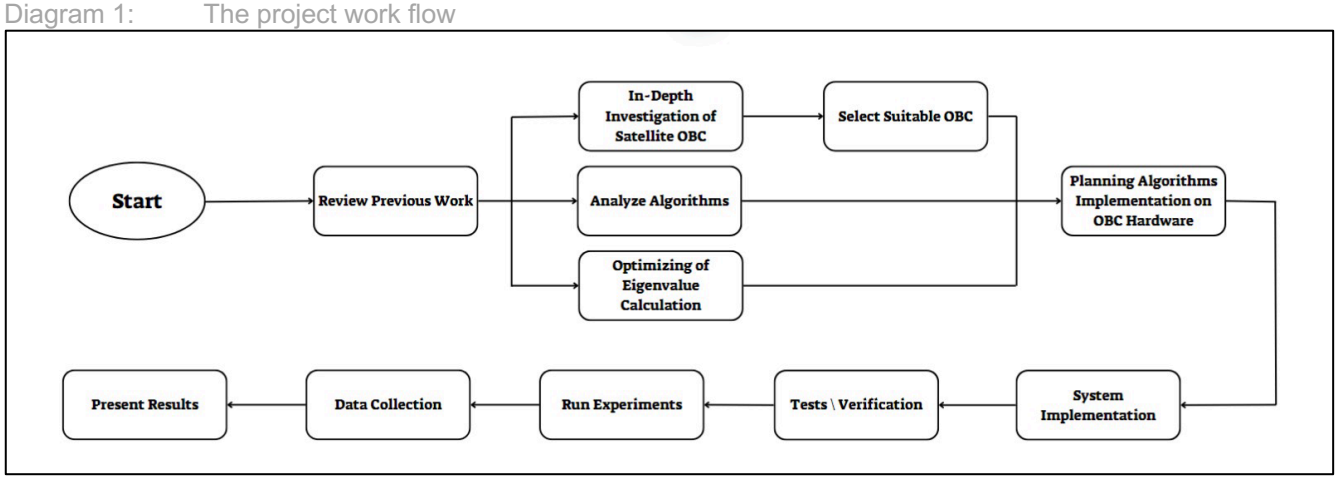
In the initial phase of the project, we began by thoroughly reviewing and analyzing the previous work titled "Feasibility Analysis and Performance Testing of Collision Detection Algorithms for Satellites" **[10]**, that forms the foundation of this project. We carefully studied the documentation and reports provided by the prior team and conducted an in-depth examination of the algorithms and their implementation. This process allowed us to develop a comprehensive understanding of the problem, the mathematical calculations involved, and the logic underpinning each algorithm, including the conclusions reached by the previous team members.
After building a solid understanding of the project, we moved on to the planning and execution of our tasks, which include the following:

- **In-Depth Investigation of Satellite OBC:** We conducted a detailed analysis to identify the most suitable satellite OBC for implementing and testing the algorithms. This investigation was guided by the specifications of the ISIS OBC satellite **[13]** presented by Dr. Elad Danenberg. Based on this foundation, we performed an extensive review of various satellite OBCs. Although the selected OBC does not need to meet space conditions at this stage, we conducted thorough research, including consultations with satellite manufacturers, and created a comparison file outlining the specifications of four relevant OBCs. Ultimately, we selected the OBC whose technical characteristics most closely aligned with those of the ISIS OBC. This OBC has been ordered, and in Phase B of the project, we will implement the algorithms on its hardware.
- **Optimization of Eigenvalue Calculation in the CATCH Algorithm:** The CATCH algorithm requires the calculation of eigenvalues after determining the CPP coefficients, which form a special NxN companion matrix **[[2], Eq. 18]**. The eigenvalues of this matrix correspond to the polynomial roots. A significant challenge in the CATCH algorithm is the computational cost of finding these roots, which involves solving for the eigenvalues of the NxN matrix. To address this, we conducted an in-depth study of the structure of the companion matrix **[[2], Eq. 20],** identifying key properties that could help reduce the overall number of operations. This optimization aims to enhance the convergence speed of the algorithm and improve its efficiency.
- **Planning Algorithm Implementation on Satellite Hardware:** We began developing a strategy for implementing the algorithms on the hardware of the selected satellite OBC. In addition, we researched various methods for establishing communication between the Tested OBC system (running on the satellite OBC) and the Test Station system (running on our local computer). This research included exploring wired communication options to assess system performance under different connectivity conditions.

In the next phase of the project, we will continue to optimize the eigenvalue calculation process and initiate the development of the testing system. This will involve a comprehensive study of the hardware of the selected satellite OBC, followed by the implementation of the algorithms on the selected OBC. We will use a propagator to generate the necessary data and create datasets that will be employed during testing.

Upon developing and validating the system, we will conduct a series of experiments to measure the algorithms runtime and memory usage on the selected satellite OBC. These experiments will include a variety of inputs and algorithm versions, along with optimizations aimed at improving performance. The data and results obtained from these tests will be integral to our final report. After completing the experiments, we will present our findings to our client Dr. Elad , to support his ongoing 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 implementing and developing some feature, creating unit tests for the relevant code and testing of the created code on the selected satellite OBC. 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. Additionally, as students balancing various academic responsibilities, including coursework, exams, and other commitments, we need a flexible development process. Agile allows us to effectively manage our time and accommodate the demands of the project alongside our other obligations.

4.1.3. Research and Implementation

Our project builds upon the previous work titled "Feasibility Analysis and Performance Testing of Collision Detection Algorithms for Satellites" **[10]**. We began by thoroughly reviewing the findings of the prior team, gaining a solid understanding of the algorithms, their mathematical foundations, and logical structures.
We also held several meetings with Dr. Elad to clarify project requirements and address any gaps in understanding. We conducted an extensive investigation to select the most suitable satellite OBC for testing, focusing on one similar to the ISIS OBC.
Additionally, we researched communication methods between the Tested OBC system and the Test Station that represents our local computer, examining wired options. We also considered potential challenges in transitioning from theoretical algorithms to practical implementation on the OBC.

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. After discussing the subject and reading about the performance of the different languages we decided to choose C++.
C++ is a compiled language, which allows for optimized performance that is crucial in systems with limited processing power, like OBCs.
C++ allows to interact directly with the hardware, such as memory management and direct control of I/O operations, which is important for embedded systems.
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), also the prior team found out implementation for SGP4 in C++ **[5]** in case we will need to implement using SGP4 directly.

4.1.3.2. Choosing the operation system

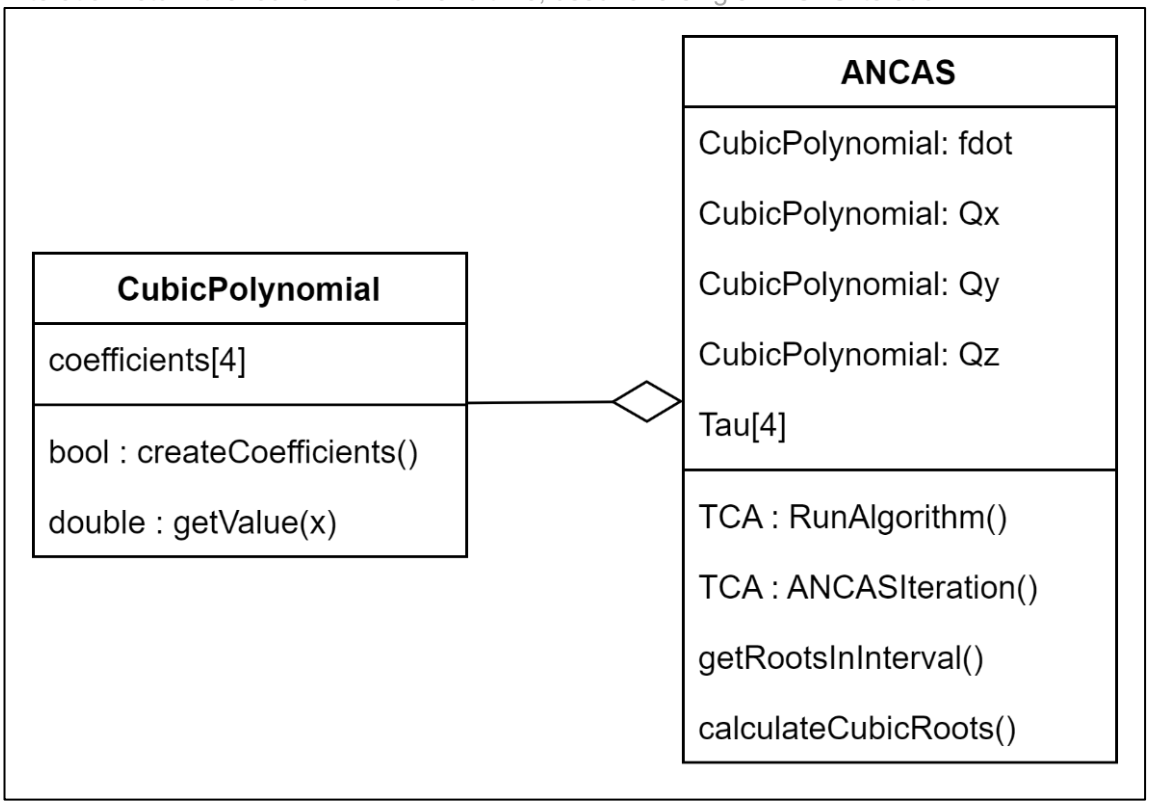
We have selected Linux as the operating system for both the TS-7553 OBC and the Test Station due to its real-time capabilities, flexibility, and efficiency. Linux, especially with real-time extensions, ensures precise scheduling and low-latency performance, crucial for algorithm testing on the OBC. Lightweight distributions optimize resource usage on the OBC, while Linux's broad hardware compatibility simplifies integration. Additionally, Linux offers robust support for various networking protocols, making it ideal for managing communication between the OBC and the Test Station. Whether using Ethernet, Wi-Fi, or other protocols, Linux ensures reliable real-time data transfer and system control, which is critical for evaluating our algorithms. This unified environment supports streamlined development, debugging, and efficient workflow, making Linux the best choice for our project.

4.1.3.3. Algorithms implementation

4.1.3.3.1. ANCAS Implementation

ANCAS **[1]** implementation is pretty straight forward, there are no inner loop or complicated algorithms in use here, we only need to find the roots of a cubic polynomial, and it can be done using a formula. We kept the implementation as simple and straight forward as possible, only taking out the code for each iteration logic, including fitting the polynomial and finding the roots, into a different function so we can reuse the code for SBO-ANCAS **[3]**. We created a class representing a cubic polynomial with functions for creating the coefficients and for getting a value at a point x. We created a function for finding the roots using the cubic polynomial formula and created unit tests that check the roots finding for a polynomial with 0 to 3 real roots in range.

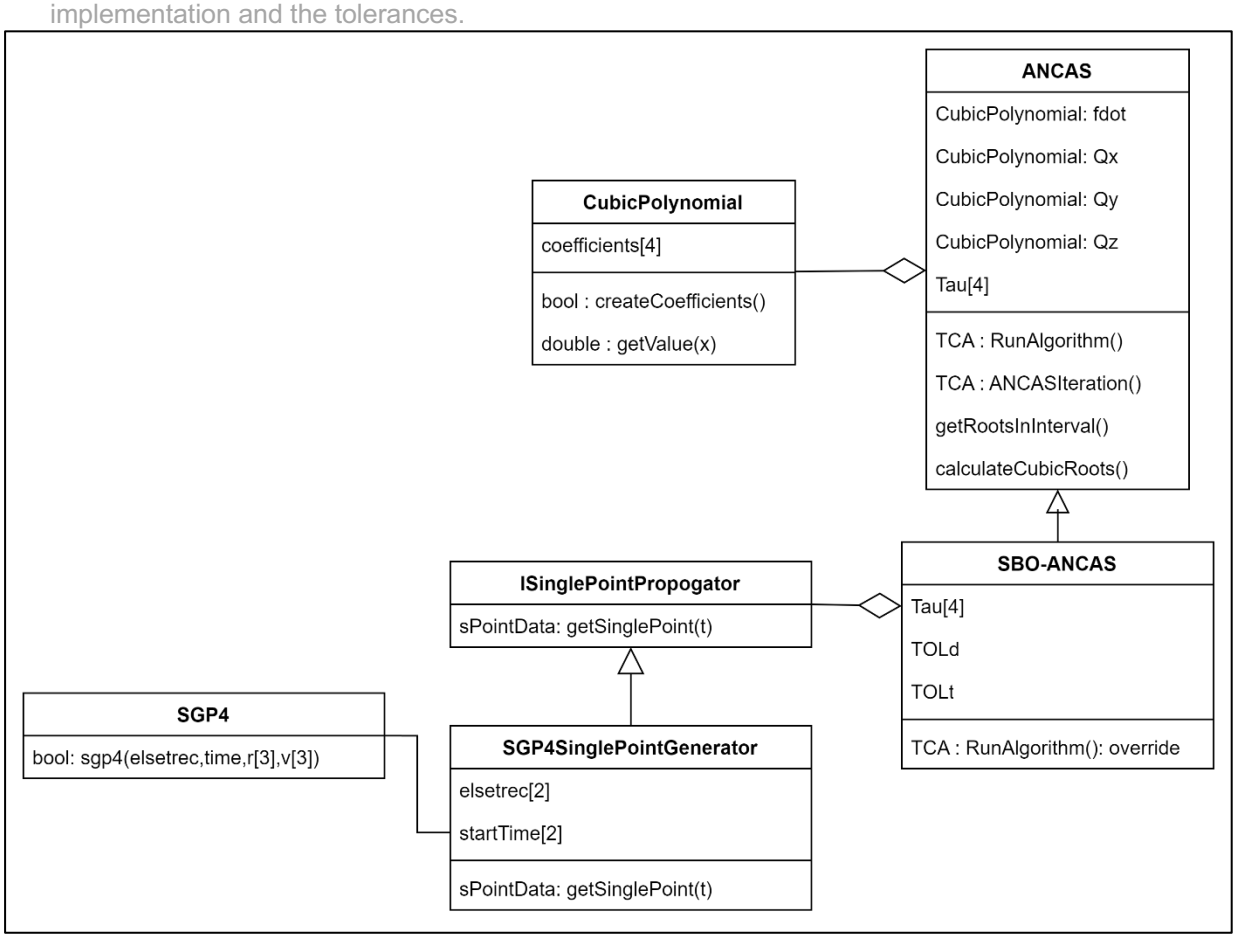
Diagram 2: Class diagram for ANCAS. Including a function for calculating the roots of a cubic polynomial used by a function for getting the roots in the interval 0,1. The function for ANCAS Iteration return the found minimum and time, used for a single ANCAS iteration.



4.1.3.3.2. SBO-ANCAS Implementation

SBO-ANCAS **[3]** acts similar to ANCAS **[1]** in every iteration, initialize the polynomials, finding the roots and so on. To avoid rewriting the same code we inherited ANCAS and only needed to override the RunAlgorithm function. We added an interface for the propagator SBO-ANCAS uses, because we only need a single point in time every time, we called it SinglePointPropogator. We implemented the interface using SGP4 and used it for our testing. Additionally, SBO-ANCAS needed the tolerances in both time and distance as input.

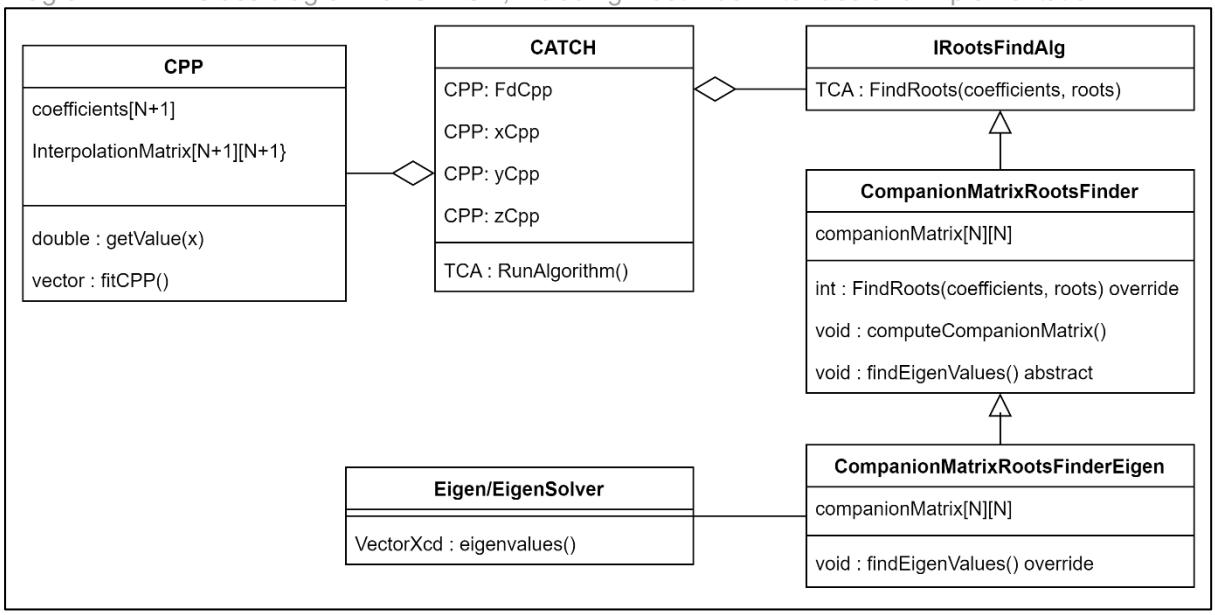
Diagram 3: Class diagram for SBO-ANCAS. Including the Propagator interface and implementation and the tolerances.



4.1.3.3.3. CATCH Implementation

CATCH implementation required implementing the Chebyshev Proxy Polynomials (CPP) class, with function for calculating the polynomial coefficients and to get the value at a point x. we needed the freedom to use different variations of the roots finding to check different libraries so we separated the root finding problem into a different interface. The CATCH class uses 4 CPP, for Fd,x,y,z, additionally it uses the Rootfinder interface to get the polynomial roots in each step. We implemented the CompanionMatrixRootFinder based on the algorithm described in the CATCH's article **[2]**, and we tried two libraries for finding the Eigenvalues of the Companion Matrix. We implemented using Eigen **[23]**.

Diagram 4: Class diagram for CATCH, including Rootfinder interface and implementation.



4.1.4. Comparative Analysis of Satellite Boards for Algorithm Testing

4.1.4.1. Background and Research Approach

As part of the continuation of the project "Feasibility Analysis and Performance Testing of Collision Detection Algorithms for Satellites", an in-depth study we conducted on the satellite **ISIS OBC [13]** provided by Dr. Elad Danenberg. This research involved gathering data through consultations with multiple companies, including SatSearch **[19]** , GOMSpace **[12]**, and ISILAUNCH **[14]**, and reviewing various datasheets **[26]** and other technical resources. The goal was to acquire a comprehensive understanding of the ISIS OBC's specifications, such as processor speed, RAM capacity, non-volatile storage, code storage, power consumption, and built-in Real-Time Clock (RTC). The goal was to find a cost-effective alternative that could support algorithm testing in a terrestrial environment without requiring space-grade durability.

4.1.4.2. Satellite ISIS OBC

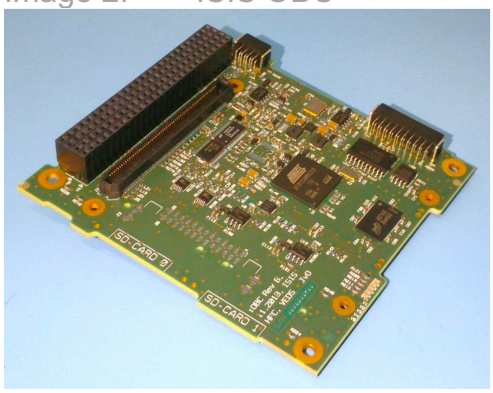
To understand the environment in which our project will be executed, it was essential to study the ISIS OBC satellite system. The ISIS OBC is a specialized satellite board developed by Innovative **Solutions In Space B.V. (ISIS) [13]**, a company that provides various systems for small satellites, such as CubeSats.
Key Features of the ISIS OBC:

1. **Processor**: The ISIS OBC is equipped with a 400 MHz ARM9 (AT91SAM9G20) processor. This processor is widely used in embedded systems due to its balance of power efficiency and performance, making it ideal for satellite operations where energy conservation is critical.
2. **Memory**: It comes with 32MB of SDRAM, which provides sufficient space for running multiple satellite applications simultaneously, including real-time data processing, communication, and algorithm execution.
3. **Non-Volatile Storage**: The OBC is equipped with two 2GB SD cards configured with a FAT32 file system. This allows the board to store important mission data, including telemetry, payload data, and algorithm results, even in the event of power loss. The redundancy in storage provides a safeguard against data corruption or failure in one of the storage systems.
4. **Real-Time Clock (RTC)**: A dual-redundant RTC ensures accurate and reliable timekeeping, which is critical for coordinating satellite operations and tasks such as collision detection. This feature ensures that the OBC can handle time-sensitive processes, especially those reliant on precise timing, such as navigation and communication.
5. **Power Consumption**: Operating at 380mW, the ISIS OBC is highly energy-efficient, which is essential for long-term satellite missions where power is often limited. This low power consumption also ensures minimal heat generation, preventing the system from overheating during extended operations.
6. **Operating System**: The OBC runs on FreeRTOS, a real-time operating system known for its efficient task scheduling and low resource usage. FreeRTOS is popular in space applications because it provides the necessary tools for real-time applications, such as collision detection algorithms, while maintaining a low memory and processing overhead.
7. **Modularity and Expandability**: The ISIS OBC is designed to be highly modular, meaning it can be customized to integrate with different payloads and subsystems required for specific satellite missions. This modularity makes it

versatile for a wide range of CubeSat missions, from research and educational missions to commercial space operations.

The ISIS OBC's combination of processing power, memory, real-time capabilities, and power efficiency makes it an ideal fit for our system. It ensures that the collision detection algorithms will run efficiently and reliably under the constraints typically encountered in satellite environments. The OBC's processing capacity, memory, and real-time accuracy align with the project's requirements for handling complex algorithms, ensuring that the test results will be both relevant and reliable for the final satellite application.

Image 2: ISIS OBC



The ISIS OBC serves as the reference system for our project because of its robust design and suitability for small satellite missions. Its balance of processing power, energy efficiency, and real-time capability make it an ideal candidate for testing collision detection algorithms in a constrained environment. By closely studying this system, we aim to find alternative OBCs that align with its specifications for cost-effective algorithm testing in terrestrial environments.

Understanding the ISIS OBC's architecture helps us ensure that any algorithm optimizations and tests we perform will be relevant when transferred to the actual satellite system. This background is crucial for selecting appropriate alternative boards that can simulate the ISIS OBC's performance in a laboratory setting.

4.1.4.3. Selection Criteria for Alternative Boards

Our goal was to identify an alternative board that closely matched the specifications of the ISIS OBC, ensuring comparable performance for algorithm testing. Another thing we needed 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. There are many options available for choosing our satellite OBC, and we needed to filter them by determining what is essential to us.
The selection criteria focused on the following key features:

- **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.

- **Processor Performance:** The processor is central to the board's ability to run complex algorithms efficiently. To ensure the algorithms will perform as expected, the alternative board must have a processor that closely matches as possible to the 400MHz ARM9 used in the ISIS OBC. This ensures that any performance results obtained during testing will be reliable and comparable when transferred to the ISIS OBC.

- **Memory:** Adequate RAM is essential for running collision detection algorithms, which can require substantial memory to process data in real time. The alternative board needs to have sufficient RAM to handle large data sets and computations efficiently. Additionally, ample storage capacity (both volatile and non-volatile) is necessary for storing the algorithms, input data, and output results during testing.

- **Non-volatile Storage:** Non-volatile storage retains data even when power is lost, making it critical for long-term data storage in real-time systems. The alternative board should offer robust non-volatile storage options to ensure data integrity during testing.

- **Real-Time Clock (RTC):** A built-in Real-Time Clock (RTC) is necessary for time-sensitive operations, especially for tasks like collision detection, which rely on precise timing. The ISIS OBC’s dual-redundant RTC ensures high accuracy and reliability, so the alternative board should have a similar feature to maintain consistency in timing operations. This way we can assess the performance of the algorithms in the real environment.

- **Power Consumption:** Power efficiency is crucial, particularly for extended testing scenarios where energy usage must be minimized to prevent overheating or power depletion. The alternative board should have power consumption comparable as possible to the ISIS OBC, which operates at 380mW, to support long-duration tests without frequent power interruptions.

- **Operating System Compatibility:** The board needs to be compatible with FreeRTOS or other real-time operating systems to ensure seamless integration with the software framework required for embedded applications. FreeRTOS is widely used for satellite systems because of its efficient real-time task

scheduling, so compatibility with this system will facilitate the development and testing of algorithms in a real-world context.

- **Cost:** While maintaining technical performance is important, the alternative board should also be cost-effective. It is essential to find a board that provides comparable performance and features to the ISIS OBC but at a lower price point, as this would reduce overall project costs without compromising the quality of testing.

4.1.4.4. Comparative Analysis

4.1.4.4.1. Test Setup

The comparison was structured around several performance metrics and tests designed to evaluate each board's suitability for algorithm testing. The tests focused on performance, memory handling, storage capabilities, and operational efficiency, excluding space-grade durability requirements. The goal was to ensure that the alternative board could adequately simulate the ISIS OBC in a laboratory environment.

4.1.4.4.2. Data Compilation

The following comparative table was developed to visually represent the performance differences between the ISIS OBC and the four alternative boards. The table extracted data from official datasheets **[26]**, internet research and creating communications with satellite companies through emails **[12][14][19].** This table is providing a structured overview of each board's specifications for side-by-side comparison.

Table 1: Comparative Analysis Table of Satellite Boards

Feature	ISIS-Innovative Solutions In Space B.V.	Atmel SAMA5D27-SOM1-EK1	SAM9-L9260	STM32F429I-DISC1	TS-7553
Processor	400MHz, 32-bit ARM9 (AT91SAM9G20)	ARM Cortex-A5 @ 500 MHz	ARM9 @ 180 MHz	ARM Cortex-M4 @ 180 MHz	ARM9 @ 250 MHz
RAM	32MB SDRAM	128MB DDR2	64MB SDRAM	256KB SRAM	64MB DDR2
Non-volatile data storage	2x2GB SD-Cards with FAT32 file system 256kB FRAM (high endurance and fast read/writes)	Requires MicroSD and FRAM	Requires MicroSD and FRAM	Requires external MicroSD and FRAM 64-Mbit (8MB) SDRAM	256MB XNAND Flash, needs MicroSD and FRAM
Code storage	1MB NOR-Flash	64Mb (8MB) QSPI Flash	512MB NAND Flash	2MB internal Flash	256MB XNAND Flash
Timing	2 redundant real-time Clocks	Single RTC	Single RTC	Single RTC	Single RTC
Average power consumption	380mW, typical usage @ 3.3V supply	300mW to 500mW	~1.5W to 2.5W	~100mW to 300mW	3.2W
Operating System	FreeRTOS	Linux, FreeRTOS	Linux, FreeRTOS	FreeRTOS, STM32CubeF4	Linux, FreeRTOS needs porting
Price ranges	€ 6000 EUR	$290 USD	€ 100 EUR	$29 USD	$180 USD

4.1.4.4.3. Detailed Board Comparisons

ISIS OBC [13][26]: The reference board, equipped with a 400MHz ARM9 processor, 32MB SDRAM, 2x2GB SD cards, 256kB FRAM, and a dual-redundant real-time clock (RTC), ensures stable and accurate timekeeping. It consumes 380mW of power and runs on FreeRTOS. It is a highly specialized board but expensive, with a cost of around 6000 EUR.

TS-7553 [24][26]:

- **Processor:** 250MHz ARM922T, slower than the ISIS OBC but with 64MB DDR2 RAM, double that of the ISIS OBC.
- **Power:** 3.2W, significantly higher than the ISIS OBC.
- **Cost:** $180 USD.
- **Modifications:** Requires additional MicroSD for storage ($5-$10) and FRAM ($10-$20). FreeRTOS needs to be ported from Linux (~$15-$30). It has a single real-time clock (RTC).

Atmel SAMA5D27-SOM1-EK1 [6][26]:

- **Processor:** 500MHz ARM Cortex-A5, faster than the ISIS OBC with 128MB DDR2 RAM.
- **Power:** 300mW to 500mW, similar to the ISIS OBC.
- **Cost:** $290 USD.
- **Modifications:** Requires additional MicroSD and FRAM for storage ($15-$30 total). It features a single real-time clock (RTC).

SAM9-L9260 [17][26]:

- **Processor:** 180MHz ARM9, significantly slower than the ISIS OBC.
- **Power:** 1.5W to 2.5W, much higher power consumption.
- **Cost:** 100 EUR.
- **Modifications:** Requires additional MicroSD and FRAM for storage, with substantial efforts required to port FreeRTOS. It comes with a built-in RTC.

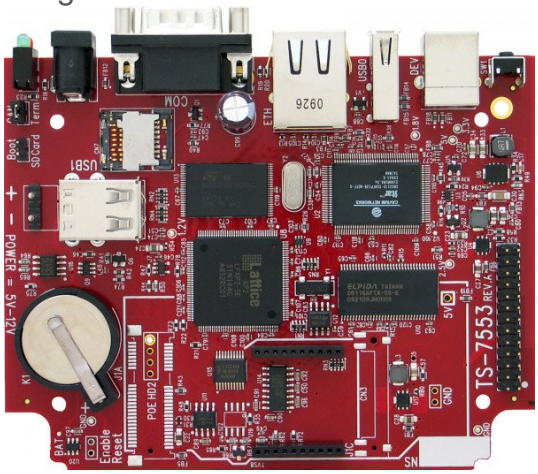
STM32F429I-DISC1 [21][26]:

- **Processor:** 180MHz ARM Cortex-M4, significantly less powerful than the ISIS OBC.
- **RAM:** 256KB SRAM, much less than the ISIS OBC's 32MB SDRAM.
- **Power:** 100mW to 300mW, the lowest power consumption but not sufficient for algorithm-heavy testing.
- **Cost:** $29 USD.
- **Modifications:** Requires external MicroSD and FRAM modules for storage (~$25-$45). It includes a single RTC for basic timekeeping.

4.1.4.4.4. Conclusion and Our Choice

After a thorough evaluation, the **TS-7553 [24]** has been identified as the closest match to the ISIS OBC **[13]** based on key specifications, including processor architecture, memory, and storage. Although it presents some drawbacks, such as higher power consumption and the absence of storage redundancy, its architectural similarity to the ISIS OBC makes it an optimal candidate for our algorithm testing. Importantly, the TS-7553 offers slightly lower performance, which aligns with our goal of testing in a constrained environment. This ensures that any optimizations we make will be applicable to the more advanced ISIS OBC, providing relevant and transferable results. Therefore, the TS-7553 OBC is our primary choice, as it ensures the algorithms will perform effectively in the final satellite system.

Image 3: **TS-7553 Onboard Computer Front view**



4.1.5. Investigation of Communication Protocol Methods Between the Tested OBC and the Test Station System

In our project, we explored different communication methods to establish a reliable and efficient connection between the **Tested OBC (the selected TS-7553 OBC [24])** and the **Test Station System**, which represents our local computer. Effective communication between these systems is essential to assess the performance of our algorithms in real-time and to simulate the operational conditions of the satellite system. We explored wired communication options to ensure flexibility and reliability in various testing environments. Furthermore, we considered various factors such as data transmission rates, reliability, and potential challenges in moving from theoretical algorithm design to practical implementation on the OBC.

4.1.5.1. Research on Communication Methods And Protocols

As part of our transition from the theoretical development of algorithms to their practical implementation on the TS-7553 OBC, we conducted extensive research on possible communication methods that would enable efficient, reliable and fast data exchange between the Test Station System and the TS-7553 OBC **[24]**. The communication methods explored are essential to ensure real-time performance testing, reliable data transfer, and alignment with the specific requirements of our project. Below is an overview of our investigation into wired communication methods:

We focused on **Ethernet** and **USB** communication for wired connections, as these are commonly used in embedded systems and can be easily implemented on the TS-7553 OBC. Both methods offer high data transfer rates and stability, which are critical for real-time algorithm performance monitoring.

1. **Ethernet** is one of the most widely used wired communication methods for local area networks (LANs). It offers high data transfer speeds and reliability, making it an excellent option for real-time data exchange.

- **Advantages:**
  - Provides high-speed data transfer (up to 1 Gbps or higher), enabling efficient handling of large datasets during algorithm testing.
  - Supports TCP/IP protocols, which ensure reliable data transmission with built-in error detection and correction, important for maintaining data integrity during testing.
  - Low latency and stable connection, essential for testing algorithms that require real-time feedback.
- **Implementation on TS-7553 [24]:** The **TS-7553 OBC** is equipped with an Ethernet port, which allows us to utilize this method without needing additional adapters or specialized drivers, making it straightforward to set up and use for direct communication with the Testing Station System.
- **Challenges:**
  - Requires physical cabling, which may reduce mobility or flexibility in testing setups, especially when the Testing Station System needs to be located far from the OBC.
  - Slightly higher power consumption compared to wireless options, though this is manageable in the testing environment.

2. **USB (Universal Serial Bus)** is wired communication method that provides plug-and-play simplicity and is widely supported by embedded systems. It offers fast data transfer rates, with speeds up to **5 Gbps** (USB 3.0), but is generally more suited for short-distance, simpler setups compared to Ethernet.

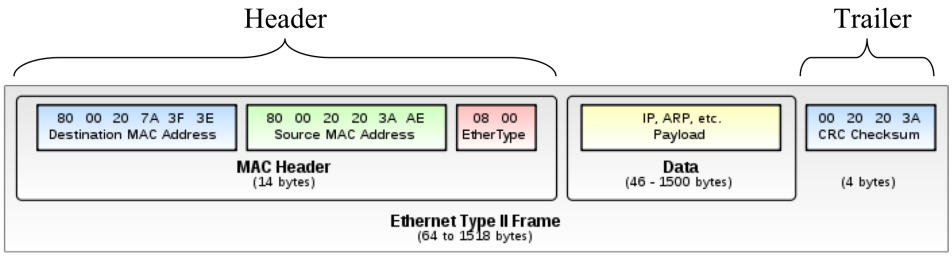
- **Advantages:**
  - Plug-and-play simplicity makes it an accessible option for quick data transfers between the Tested OBC and the Testing Station System.
  - Offers sufficient speed (up to 480 Mbps with USB 2.0 or 5 Gbps with USB 3.0) for the real-time monitoring and testing of algorithms.
  - USB can supply both data and power, simplifying the setup by reducing the number of necessary cables.
- **Implementation on TS-7553:** The TS-7553 OBC features USB ports, making it a viable option for communication. USB 2.0, for example, can offer speeds of up to 480 Mbps, which is more than sufficient for our needs. However, USB communication often requires additional configuration and drivers to ensure compatibility with real-time operating systems like FreeRTOS.
- **Challenges:**
  - Limited cable length (typically up to 5 meters) may restrict the flexibility of the test setup.

4.1.5.1.1. Selection of Optimal Wired Communication Protocol for Our System

To ensure clear communication between the **Tested OBC** (the TS-7553 OBC) and the **Test Station System**, we plan to implement **wired** communication method, testing each setup separately to determine performance and suitability. Our approach will allow us to run tests when the communication is wired, ensuring flexibility and comprehensive evaluation.

The **preferred wired communication method** for our project is **Ethernet**. This is because Ethernet offers several advantages for communication between the Tested OBC and the Test Station, making it the preferred choice for this project. It provides high data transfer rates of up to 1 Gbps, ensuring efficient communication and enabling the seamless transfer of large datasets during testing. The low latency of Ethernet is particularly crucial for real-time performance testing and accurate evaluation of algorithms, allowing for immediate feedback and minimizing delays. Additionally, Ethernet is more stable and reliable compared to USB, particularly for continuous, high-volume data transfers, making it well-suited for rigorous testing environments. Although USB is supported by the TS-7553 OBC and is easy to implement, it has limitations in cable length, making it better suited for simpler setups. Ethernet is preferable for its robustness and alignment with real-world satellite communication scenarios.

Image 4: Ethernet Header and Trailer



By implementing this communication method, we ensure the **TS-7553 OBC** is thoroughly evaluated under wired communication conditions, simulating real-world conditions and ensuring the successful implementation of the algorithms.

4.1.6. Challenges in Moving from Theoretical to Practical Algorithm Implementation

Moving from the theoretical design of the algorithms to their practical implementation on the **TS-7553 OBC [24]** poses several challenges. These challenges arise due to the constraints of hardware resources, differences in theoretical modeling versus real-world execution, and potential issues with communication between the Test Station System and the Tested OBC. Below, we outline these possible challenges and propose solutions to handle them effectively.

1. Limited Processing Power and Memory Constraints:

- **Challenge:** The TS-7553 OBC **[24]** has a 250 MHz CPU and 64 MB of RAM, which are limited compared to the higher processing power and memory typically available in a theoretical development environment. This constraint

may affect the performance and execution time of the algorithms, especially those that require intensive computational resources.

- **Solution:** We will optimize the algorithms to reduce computational complexity, ensuring they can operate efficiently within the TS-7553 OBC's hardware limits. Techniques such as code refactoring, removing unnecessary operations, and optimizing data structures will be employed. Additionally, we will monitor memory usage closely to prevent overflows or memory leaks, which could cause system crashes.

2. Real-Time Performance and Latency:

- **Challenge:** Theoretical algorithms assume near-instantaneous execution, but real-time performance on the TS-7553 OBC **[24]** may suffer due to processing delays or communication latency. This is particularly important in scenarios requiring real-time data transfer between the Tested OBC and the Test Station System, where delays could impact the accuracy of algorithm testing.
- **Solution:** To mitigate this, we have prioritized the use of Linux because of its real-time extensions and precise scheduling capabilities. Furthermore, we will prioritize the use of Ethernet for wired communication during real-time testing, as it offers low latency and reliable transmission. Linux's support for the Ethernet protocol ensures that we can achieve the low-latency, real-time data transfer needed for accurate and reliable algorithm testing, minimizing the potential impact of delays.

3. Data Loss and Transmission Errors:

- **Challenge:** In wired communication, there is a risk of data loss or transmission errors. These issues could affect the accuracy of the data used in testing, leading to unreliable results.
- **Solution:** We will implement error detection and correction mechanisms, such as Cyclic Redundancy Check (CRC) and packet retransmission protocols such as Transmission Control Protocol (TCP), to ensure data integrity. This will help detect and correct transmission errors before they affect the test results.

4. Synchronization Issues:

- **Challenge:** Ensuring proper synchronization between the OBC and the Test Station System can be difficult, especially when there is variability in communication delays or processing times. This could lead to mismatches in the timing of data exchanges, affecting the accuracy of performance evaluations.
- **Solution:** We will implement synchronization protocols that align the clock of the OBC with the Test Station System, ensuring that all data transmissions are time-stamped. This will allow us to identify and adjust for any synchronization issues that may arise during testing.

By considering potential problems such as latency, data loss, and synchronization issues, we have ensured that the transition from theoretical algorithm development to practical implementation on the TS-7553 OBC will be as smooth as possible, leading to accurate and reliable results during testing.

4.1.7. Optimization of Eigenvalue Calculation in the CATCH Algorithm

The Chebyshev-based root-finding method, Conjunction Assessment Through Chebyshev Polynomials (CATCH), proposed in the paper **[2],** offers a novel approach to compute the TCA between orbiting objects. A crucial part of this method involves computing the eigenvalues of the companion matrix **[[2], Section 2.3]]**, whose structure can be exploited to optimize the calculation process and reduce computational cost. This section details our research on trying to optimize the calculation of eigenvalues in the CATCH algorithm **[[2], Section 2.3]]**, beginning with understanding the companion matrix's structure and progressing to identifying and implementing more efficient algorithms, with a focus on reducing the number of iterations and computational complexity without compromising accuracy.

4.1.7.1. Understanding the Companion Matrix in the CATCH Algorithm

The core of the eigenvalue computation in CATCH **[2]** lies in solving the companion matrix derived from the polynomial approximations of the relative positions between objects **[[2], Eq.20]]** . The companion matrix A is formed based on the coefficients of the CPP. This matrix is typically sparse, with specific non-zero entries located near the main diagonal. Given the structure of this matrix **[[2], Eq.20]]**, the roots of the CPP are the eigenvalues of this companion matrix, which map the roots of the polynomial to the desired interval. Traditionally, the eigenvalue computation requires the use of techniques like the QZ decomposition, which has a time complexity of $O(N^3)$ where N is the matrix size **[[2], Section 2.3.4]]**. As matrices grow in size, this becomes prohibitively expensive, particularly for onboard satellite systems with limited computational resources.

4.1.7.2. Structure of the Companion Matrix

In **Section 2.3.3** of the paper **[2]** the companion matrix is introduced to compute the roots of the CPP, which approximates the distance function between two orbiting objects. The matrix A, described in **[[2], Eq.20]]**, is defined as an N × N companion matrix where the entries are built from the CPP coefficients and structured using Kronecker delta functions:

Image 5: Kronecker delta functions[1]

$$A_{j,k} = \begin{cases} \delta_{2,k} & j = 1, k = 1 \dots N \\ \frac{1}{2}(\delta_{j,k+1} + \delta_{j,k-1}) & j = 2 \dots N-1, k = 1 \dots N \\ -\frac{a_{k-1}}{2a_N} + \frac{1}{2}\delta_{N-1,k} & j = N, k = 1 \dots N \end{cases}$$

The Companion matrix A, as outlined in **[[2], Eq.20]]**, of the paper **[2]** has the following properties:

- **Sparse Structure**: Most elements are zeros, except near the diagonals.

The matrix has many zero entries, making it sparse. This can significantly reduce storage and computational overhead when leveraging sparse matrix algorithms.

- **Tridiagonal Form**: Non-zero elements primarily exist on the first subdiagonal and superdiagonal.
- **Banded Matrix**: Most of the non-zero elements are located near the diagonal, specifically on the first subdiagonal and superdiagonal. This creates a **tridiagonal or near-tridiagonal structure** in the main part of the matrix.
- **Hessenberg-like Form**: The upper part of the matrix resembles a Hessenberg matrix, where all entries below the first subdiagonal are zeros except for some structured elements at the last row.

4.1.7.3. Algorithms for Eigenvalue Calculation

To optimize the eigenvalue calculation, we reviewed several algorithms, focusing on their time and space complexities, as well as their applicability to sparse and tridiagonal matrices. We began by analyzing the standard methods, followed by more efficient algorithms that take advantage of the companion matrix's structure.

4.1.7.3.1. Standard Eigenvalue Algorithms - QR Algorithm

The most basic algorithm used for eigenvalue calculation in dense matrices is the **QR algorithm**. This method involves decomposing a matrix into an orthogonal matrix (Q) and an upper triangular matrix (R), followed by iterative updates of the matrix until it converges to a diagonal form, where the diagonal elements are the eigenvalues.

QR Algorithm Steps:

1. Start with matrix $A_0 = A$.
2. Compute the QR decomposition: $A_K = Q_k R_k$.
3. Form the next matrix: $A_{k+1} = R_k Q_k$.
4. Repeat until A_K converges to an upper triangular matrix, whose diagonal elements are the eigenvalues.

The following table describing the number of operations at the k-th step of the QR algorithm for a square matrix of size n:

Operation	Number of Operations in the k-th Step
Multiplications	$4(n-k)^2$
Additions	$2(n-k)^2$
Orthogonalization	$n-k$ operations for vector norms and adjustments
QR Decompositions	Typically involves $(n-k)^3$ operations

In the QR algorithm, the operations count can vary significantly depending on the specific implementation details, such as whether Householder transformations or Givens rotations are used for the QR factorization, and the efficiency of these methods when applied to matrices of diminishing size as k increases. The orthogonalization process typically involves vector normalization and subtraction, which can involve a significant number of multiplicative operations and additions per step.

For Example the following table gives the number of operations in the *k*-th step of the QR-decomposition by the Householder transformation, assuming a square matrix with size *n*.

Operation	Number of Operations in the k-th Step
Multiplications	$2(n-k+1)^2$
Additions	$(n-k+1)^2+(n-k+1)(n-k)+2$
Divisions	1
Square root	1

Summing these numbers over the $n-1$ steps (for a square matrix of size *n*), the complexity of the algorithm (in terms of floating point multiplications) is given by $\frac{2}{3}n^3+n^2+\frac{1}{3}n-2=O(n^3)$

Time Complexity: $O(N^3)$.
While the QR algorithm is robust, its cubic time complexity makes it inefficient for large matrices, especially when applied to the companion matrix in CATCH.

4.1.7.4. Ongoing Optimization Efforts

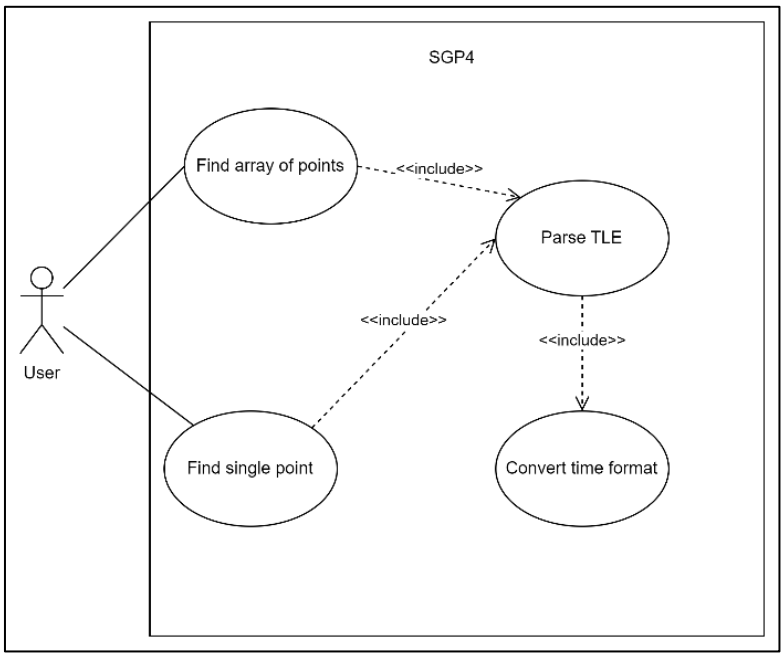
While our research is still in development, we have found that to achieve more efficient time complexity to the CATCH algorithm is both realistic and entirely achievable. This optimization promises to significantly enhance the algorithm's performance, particularly when handling larger datasets, without sacrificing accuracy or computational efficiency.
During our investigation, we discovered several key properties of the companion matrix, which are central to the eigenvalue calculations that form the core of the CATCH algorithm. The matrix exhibits a sparse and tridiagonal structure, where the non-zero elements are confined to a limited, constant number of values, primarily located near the diagonal. These characteristics make the matrix well-suited for specialized algorithms that exploit sparsity and tridiagonal.
Ongoing Investigation:
Although we have not yet obtained definitive proofs for the optimization, we continue to experiment with different approaches, focusing on reducing the number of operations while maintaining the accuracy of the eigenvalue calculations. Once we succeed, this optimization will mark a significant leap forward for the CATCH algorithm, enabling it to process large-scale data efficiently while ensuring precise results. This will be particularly impactful for real-time satellite conjunction assessments, where speed and accuracy are paramount.

4.2. Product

4.2.1. Finding the real minimal distance using SGP4

We need to use SGP4**[5]** for 2 tasks, the first is creating input data for ANCAS **[1]** and CATCH **[2]** 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 5: 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 4: Finding the TCA using SGP4

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

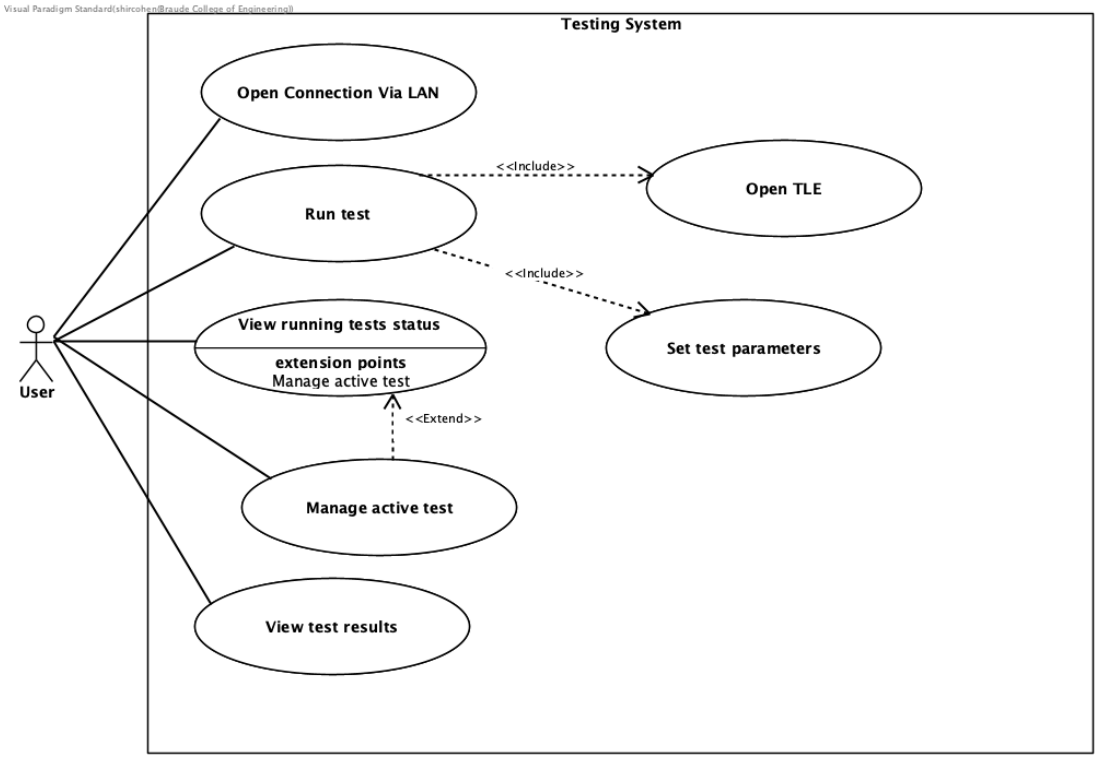
1. Initialize min_distance, step_size, min_time
2. Loop:
    a. Set counter to 2
    b. Calculate number of points and blocks
    c. For each block:
        i. Generate time points
        ii. Calculate jd, fr
        iii. Initialize satellite objects sat1, sat2
        iv. Assign jd, fr to sat1 and 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. The System

In order to test the algorithms on the TS-7553 OBC **[24]** 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 OBC, connected via **LAN using physical Ethernet cables** 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 6: System Use case diagram from the User point of view



4.2.2.1. Communication Protocol and Channels

In this section, we describe the communication methods and protocols used for data transmission between the Test Station and the Tested OBC. The purpose of the communication channel is to ensure accurate, real-time data exchange during the testing of satellite collision detection algorithms. Based on our project's requirements and hardware capabilities, we explored different communication methods and selected Ethernet as the primary communication protocol due to its reliability, speed, and low-latency properties.

4.2.2.1.1. The Communication Protocol

For the purpose of conducting tests on the Tested OBC and receiving results, two types of messages are exchanged between the Test Station and the Tested OBC via the Ethernet protocol:

1. **Message from the Test Station to the Tested OBC:**
   This message contains the test data necessary for the algorithm to run on the Tested OBC. It includes:
   - The **test recipe**: specifying which algorithm to run (CATCH **[2]** , ANCAS **[1]**, or SBO-ANCAS **[3]**), and any additional algorithm-specific parameters (e.g., the polynomial degree, tolerance levels, etc.).
   - The **test data**: consisting of an array of time points, position vectors, and velocity vectors for two objects in orbit.

   Once the Tested OBC receives this message, it proceeds to execute the appropriate algorithm based on the provided data.

2. **Message from the Tested OBC to the Test Station**:
   After executing the algorithm, the Tested OBC sends a message back to the Test Station. This message contains:

- The **calculated results**: including the TCA and the minimum distance between the two objects.
- **Performance metrics**: such as the total runtime for the algorithm, memory usage, and detailed breakdowns of the time taken for individual operations (e.g., polynomial fitting, root-finding, etc.).

These two types of messages ensure a structured and efficient data exchange between the Test Station and the Tested OBC, enabling seamless execution of tests and performance evaluation.

Test Station to Tested OBC – Test Request Message

Table 2: Test Request Message Description

Bytes	Type	Field Name	Field Description	Expected Values
0-1	Unsigned Short	Opcode	Unique identifier for the message, used for sync and opcode, identifying the message start and type.	0x1234
2-5	Unsigned Int	Data Length	The size of the test data, can be different in every message.	$[0, 2^{32} - 1]$
6-9	Unsigned Int	CRC	4 Bytes CRC, to identify errors in the message without relying on the communication type.	Calculated on the message
10-209	Struct	Test Recipe	Struct containing all the required options for running the test, including the polynomial degree, number of points, tested algorithm and more.	can vary
210-N	Array	Points Data	The algorithms input, array with the data in each point in time, each entry containing the time at the point and 4 3d vectors, the location and velocity of 2 objects in this point in time.	can vary

Tested OBC to Test Station – Test Results Message

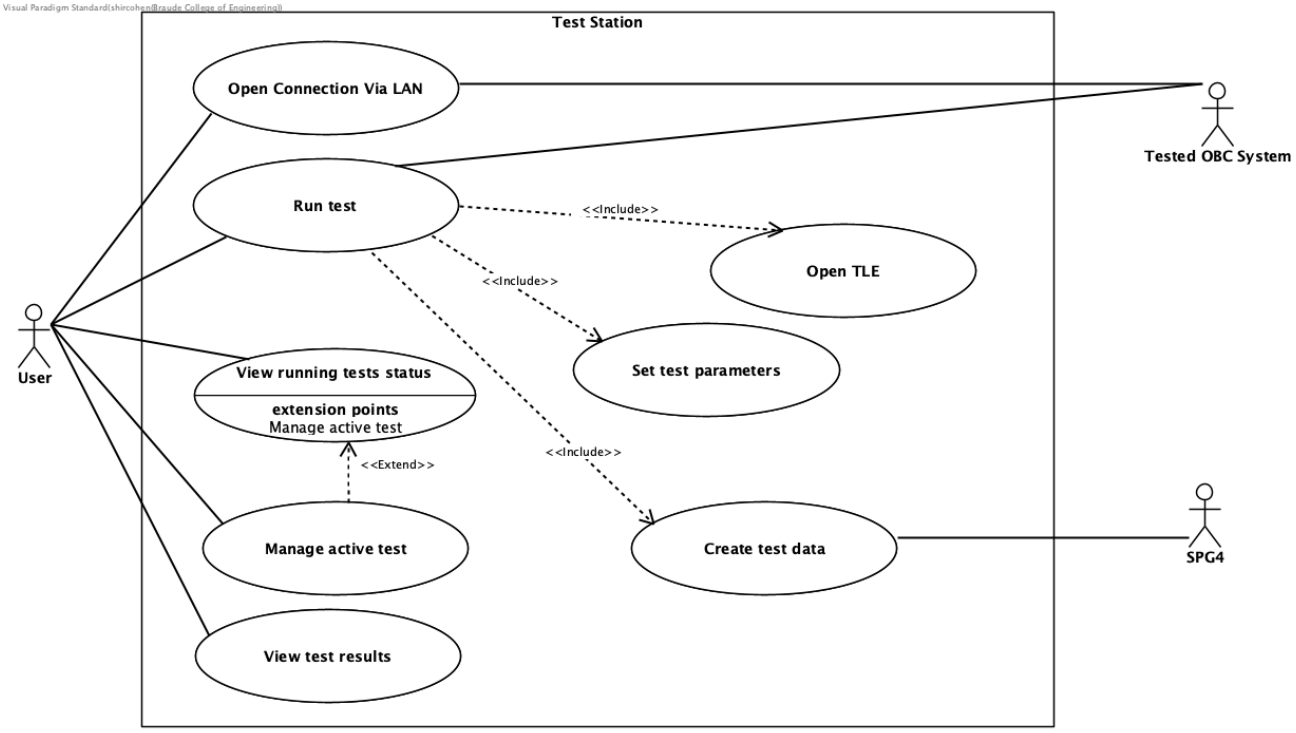
Table 3: Test Results Message Description.

Bytes	Type	Field Name	Field Description	Expected Values
0-1	Unsigned Short	Opcode	Unique identifier for the message, used for sync and opcode, identifying the message start and type.	0x4321
2-5	Unsigned Int	Data Length	The size of the test data, a constant size.	188
6-9	Unsigned Int	CRC	4 Bytes CRC, to identify errors in the message without relying on the communication type.	Calculated on the message
10-197	Struct	Test Results	Struct containing all the test results data, the found TCA and distance, total run time, average and minimal, memory usage, detailed breakdowns of the time taken for individual operations and more.	188

4.2.2.2. Test station system

We will use this types of messages for handling a large number of tests with different data and different expected result. This will help us to run large number of test variants using without the need to change the code to do so.
Using these types of messages, the testing software will start by creating fitting data for the tested OBC, sending the data to the tested OBC via our Ethernet communication protocol and while waiting for the result, the Test station will calculate the actual TCA. After finishing the calculation and getting the test result from the tested on-board computer, the Test station will analyze the result and save it into our result data set.

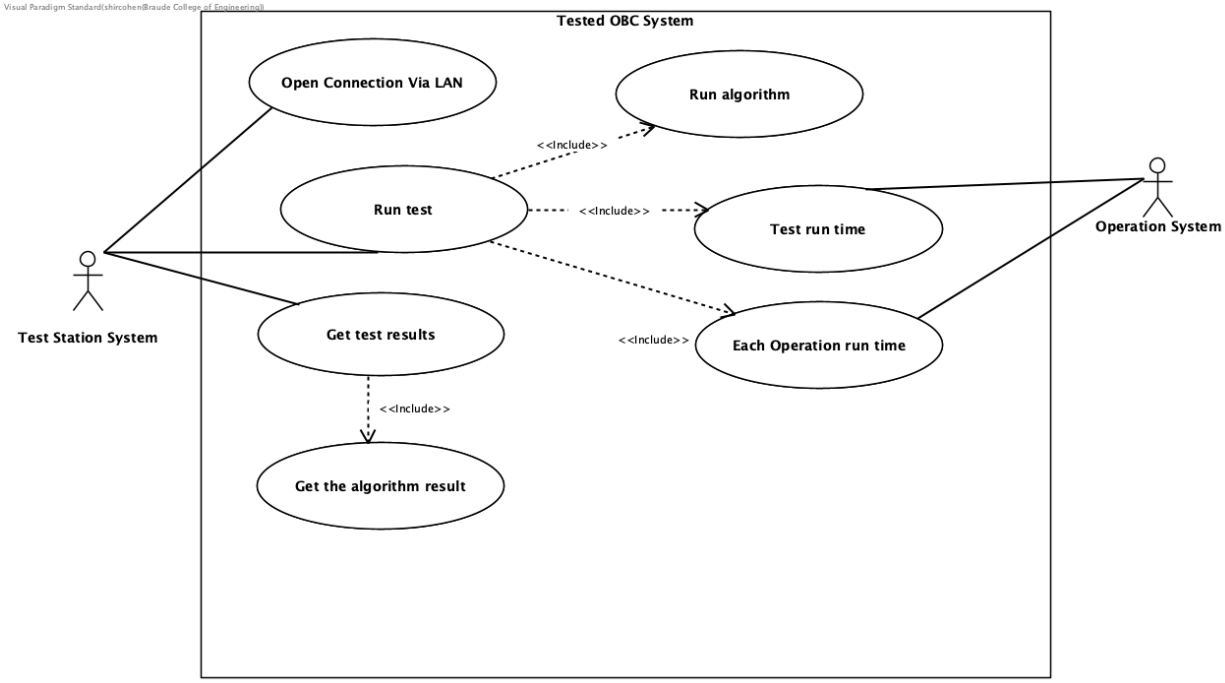
Diagram 7: System Use case diagram for the Test Station system



4.2.2.3. Tested OBC system

On the other side, the software running on the tested TS 7553 OBC **[24]** will be listening to the communication channel, after receiving a command from the Test station following by a data set the system will run the test. Starting the clock than running the algorithm with the given data set then stopping the clock and calculating the run time. Additionally, the system will measure and calculate the runtime for each individual operation in the algorithm. Once the test is completed, the results, including both the total runtime and the runtime for each operation, will be sent back to the testing station. The system will then wait for the next command.

Diagram 8: System Use case diagram for the Tested OBC system



4.2.2.4. Communication channel

To ensure clear communication between the Tested OBC (the TS-7553) and the Test Station System, we have chosen to implement a wired communication method using Ethernet exclusively. This approach provides a reliable and high-performance setup for data transmission, ensuring accurate and consistent testing results. By using Ethernet, we eliminate the complexities of wireless communication and focus on a stable, low-latency connection, which is crucial for the precise performance evaluation of the algorithms.

By relying on Ethernet for all data transmission, we ensure the TS-7553 OBC is thoroughly evaluated in a controlled and efficient environment, simulating real-world conditions where wired communication is often preferable for satellite systems. This approach simplifies the setup and guarantees the successful implementation of the algorithms.

4.2.2.5. Tested OBC system Implementation

The Tested OBC system is composed of three parts. the first is the algorithms we want to test, the versions that implemented already or any other variations we will create in the future. The second component is the Communication Manager, which facilitates communication with the test station via a Ethernet through physical cables. The communication channel implementation is OBC and operating system depended and we will have to implement specifically for the TS-7553 OBC. The third and last part is the part that manage it all, run tests and create the result. The Tested OBC works around messages from the Test Station System, we wait for an incoming message, get the Test Recipe and Test Data from the message, run the algorithm and return the results, the total runtime and the runtime for each operation. When not running a test the Tested OBC System waits for the next message. The Tested OBC system is repeatedly checking for incoming messages until a message arrived, then parsing the message and using the Test Recipe preparing and running the test, creating the result set and sending it back to the Test Station system.

Diagram 9: 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.

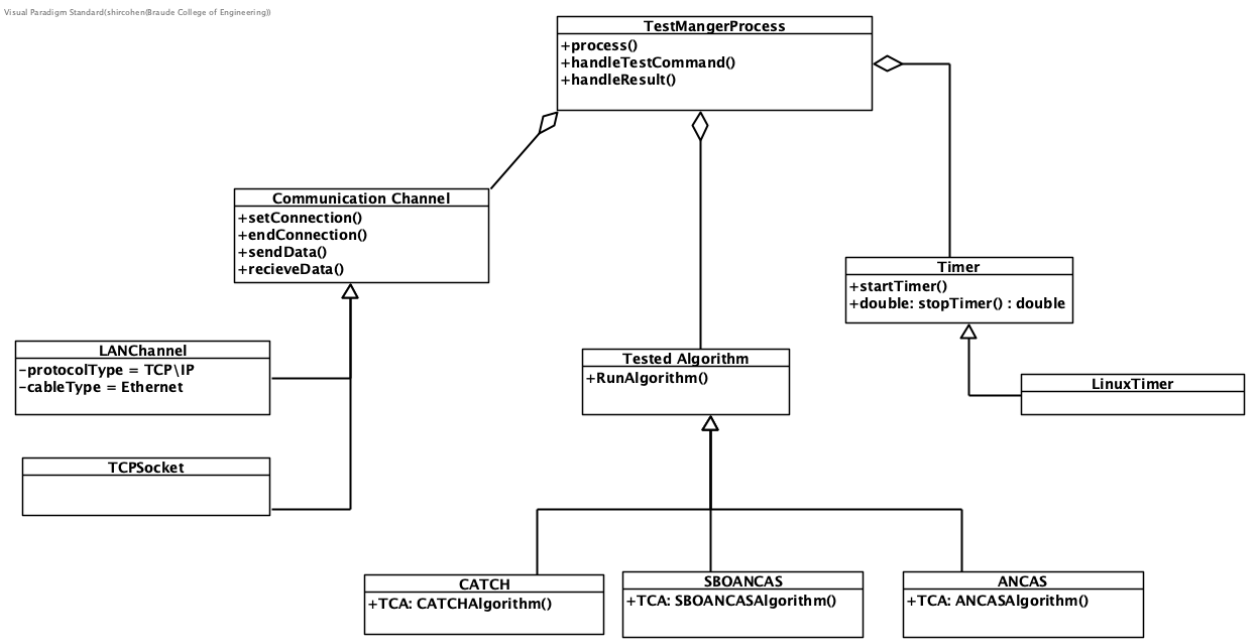
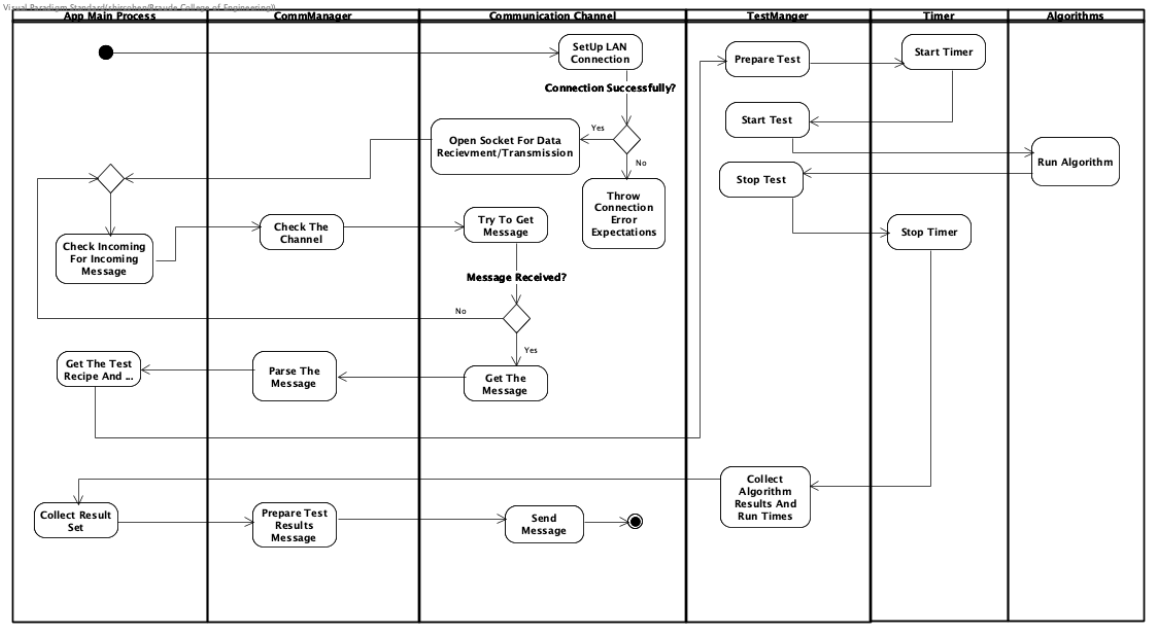


Diagram 10: Activity diagram of running a test on the Tested OBC System via Ethernet Communication Method and Protocol.

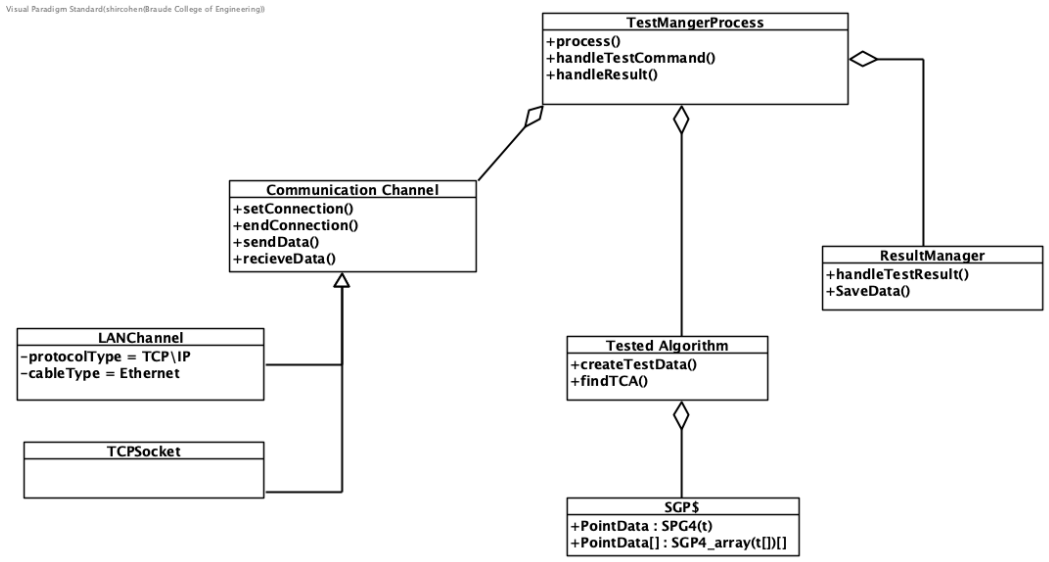


4.2.2.6. Test station system implementation

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. 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. For communication, the system exclusively uses Ethernet. The test station first establishes a wired connection to the Tested OBC by setting up Ethernet communication. The system checks for a successful connection, configures the necessary sockets for data transmission, and ensures robust, low-latency communication. Once the connection is established, the system transmits the test data and receives the results. This approach provides stable and high-speed data transmission, which is crucial for the accurate performance evaluation of the algorithms.

Diagram 11: Test station system class diagram

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

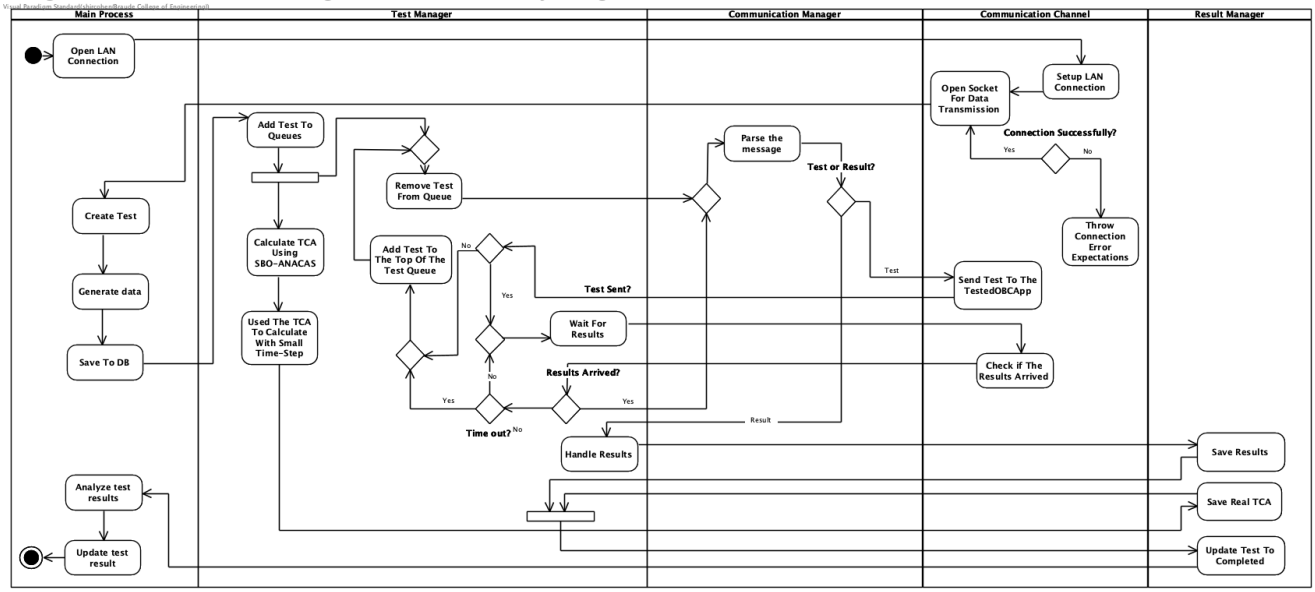


4.2.2.6.1. Running A Test - Activity Diagram

Initially, the test station system create LAN communication to establish how tests will be transmitted and results received. For LAN communication, the system sets up a connection using physical Ethernet cables, checks for successful connection, and configures sockets for data transmission.

After setting up the communication,the Main process create the test, it places it into 2 queues in the Test Manager. Inside the Test manager we have 2 independent thread each with his own task, and each with his own incoming tests queue. The first get the test from the queue and send it to the Tested OBC App, waits for the response and update the Results Manager with the test results. The second get the test, calculate the real TCA and update it using the Results Manager. Only after both of them finished with the test we can update its sate to Completed and display the results.

Diagram 12: Test Manager Process activity diagram via Ethernet Communication Method.



4.2.2.7. Full system

Looking at the full system, the process of creating and running a test starts from the user, the user click on the open LAN connection to establish LAN communication.then the user goes to the Test Creation page and create a test, filling all the necessary fields, after that the GUI managers collect the input and call the Main Process to create a test, the Main Process generate the data, save the test and give it to the test manager who forward it to the Tested OBC via the communication channels. The Tested OBC take the input, run the algorithm and send back the results who go all the way back to the tests results page and displayed to the user.

Diagram 13: Top view Use Case diagram for running a test, with the full 2 systems

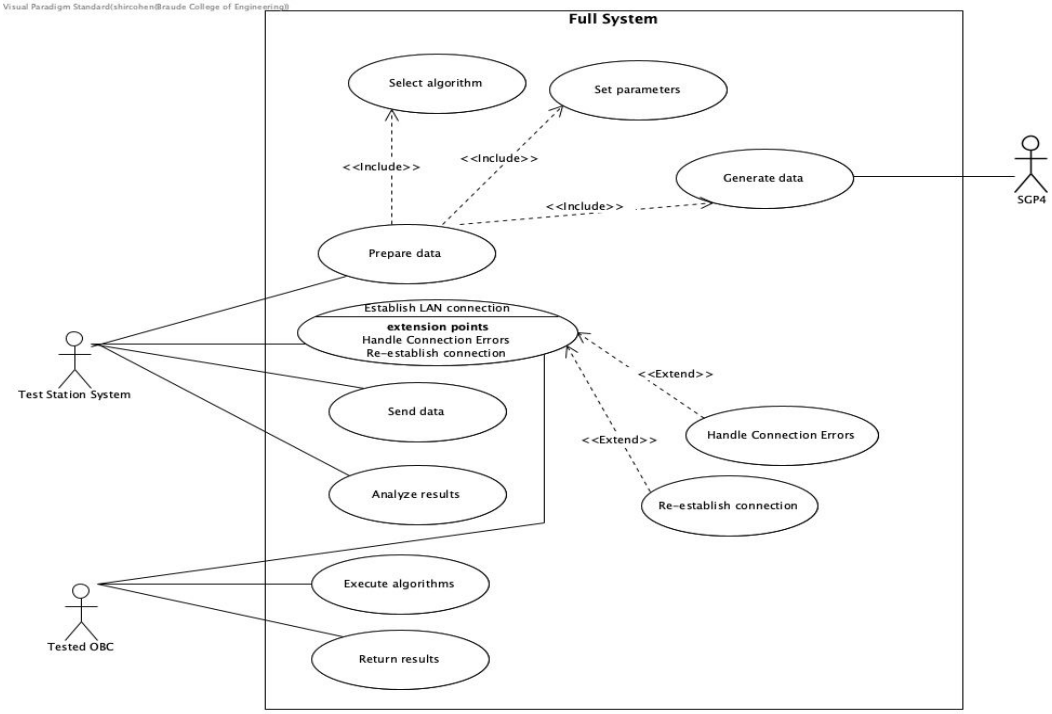
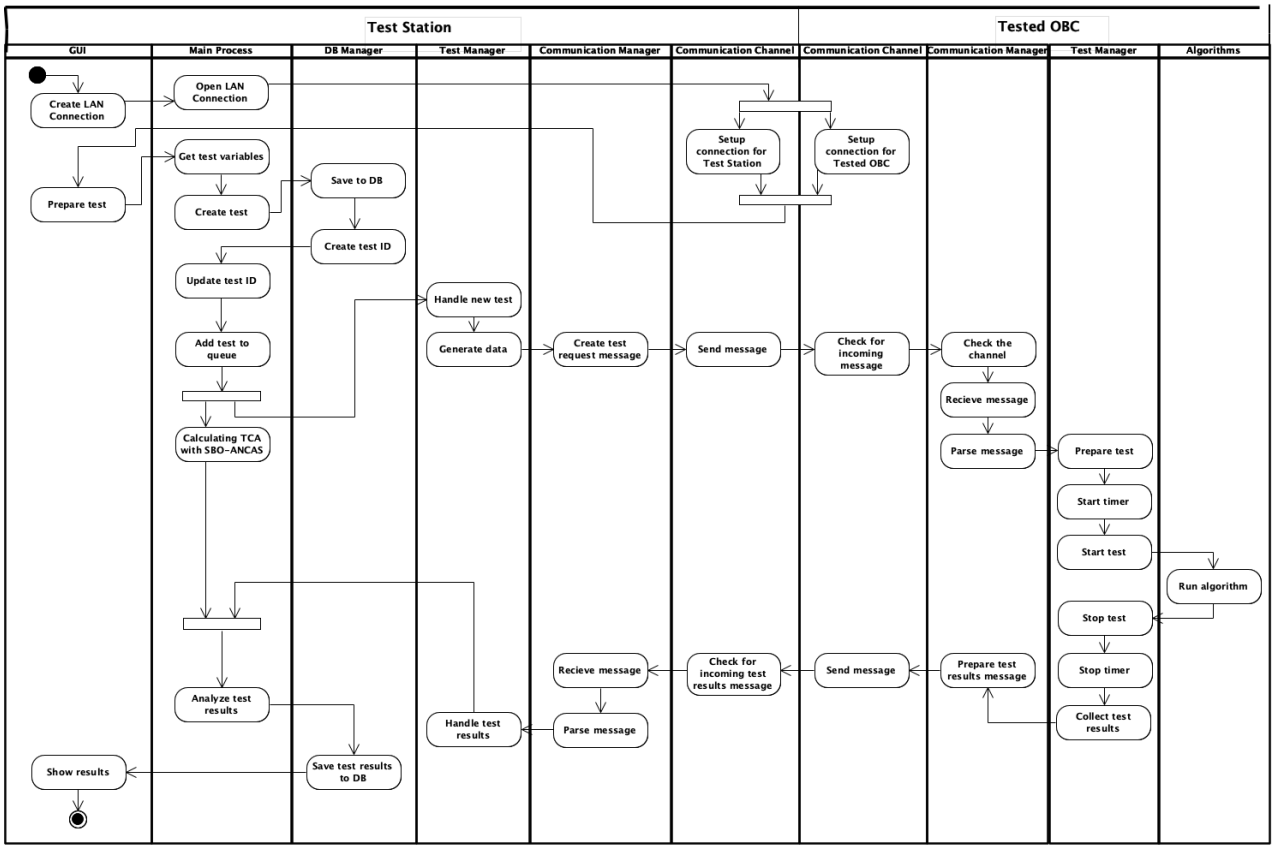


Diagram 14: Top view activity diagram for running a test, with the full 2 systems

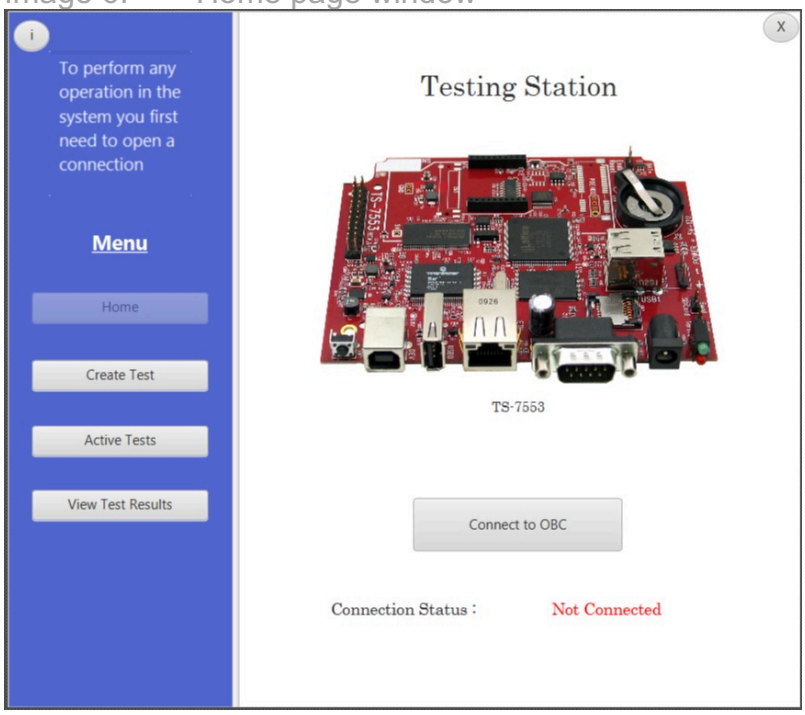


4.2.3. User interface

For the Test 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.

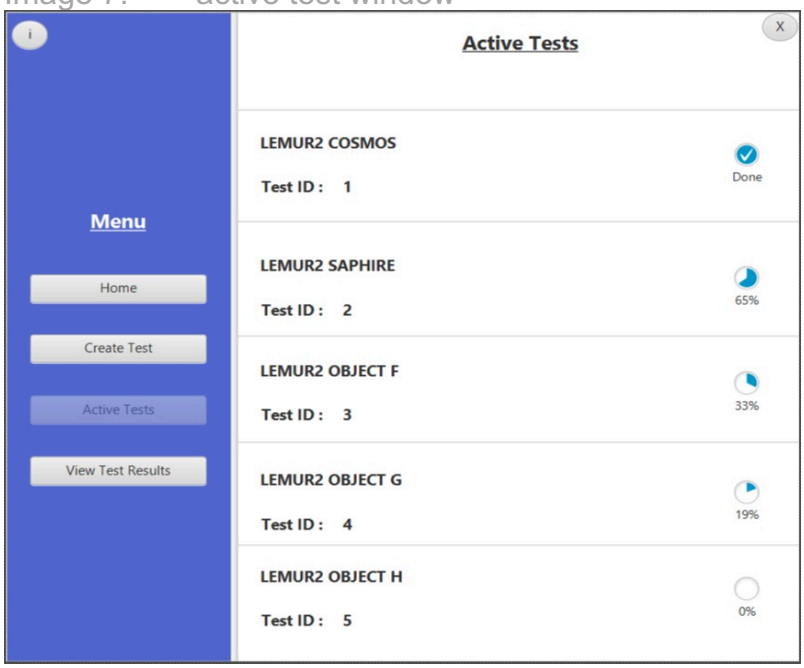
4.2.3.1. Home page view

Image 6: Home page window



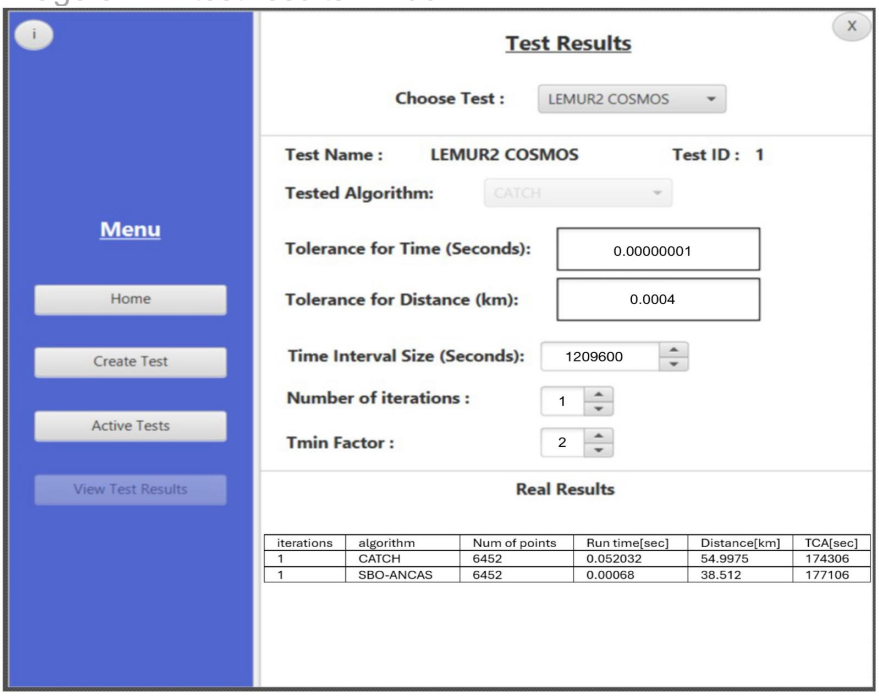
4.2.3.2. Active tests view

Image 7: active test window



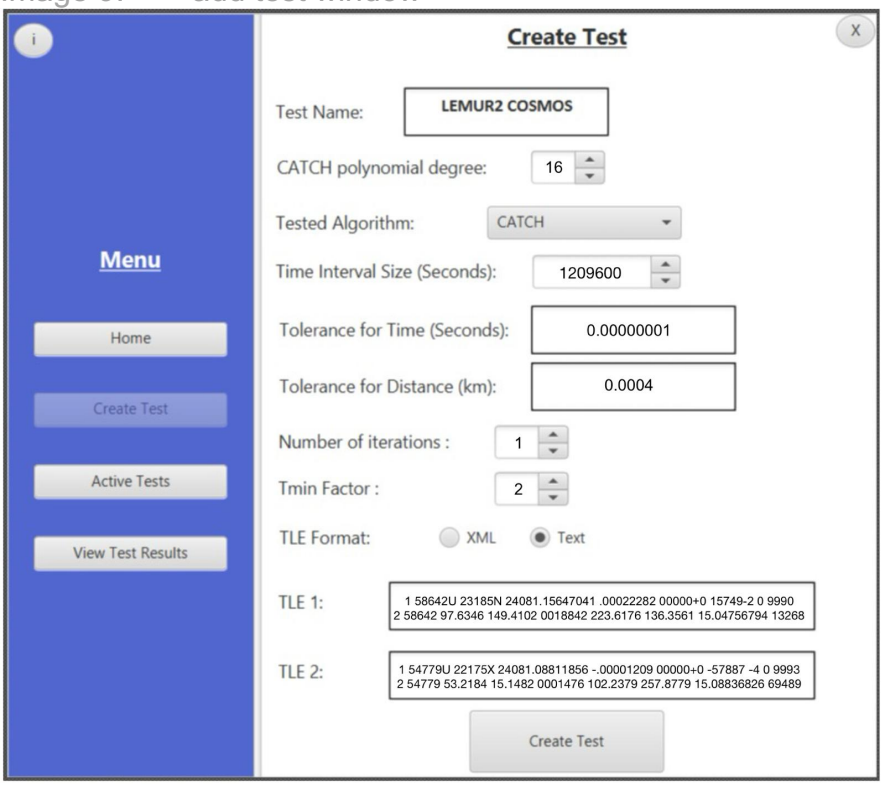
4.2.3.3. Test result window

Image 8: test results window



4.2.3.4. Add test window

Image 9: add test window



4.2.4. System Requirements

4.2.4.1. Functional Requirements

Table 4: Functional Requirements.

1	The system allows to execute CATCH, ANCAS, and SBO-ANCAS algorithms on the TS-7553 OBC to calculate the TCA and distance between two space objects.
2	The system allows the user to select which algorithm (CATCH, ANCAS, or SBO-ANCAS) to run on the OBC.
3	The system allows to generate, store, and manage test data derived from simulations.
4	The system allows to establish wired LAN communication via Ethernet between the TS-7553 OBC and the testing station.
5	The system allows to open a socket connection between the TS-7553 OBC and the test station, allowing data transfer for LAN communication type.
6	The System allows to ensure data transmission integrity and security over the network.
7	The system allows to create a Test Receipt (structure containing parameters such as the polynomial degree, number of points, and the algorithm to be tested).
8	The system manages and sends Point Data (which includes time, position, and velocity vectors of two objects for each time point).
9	The system allows to perform tests on both the OBC and the Test Station.
10	The system allows to calculating the real TCA using SBO-ANCAS with small time steps.
11	The system provides a way to compare test results against a calculated real TCA using the SBO-ANCAS algorithm.
12	The system allows to measure and report the runtime and memory usage of the selected algorithm on the TS-7553 OBC.
13	The system can measure the running times of different operations in the algorithm
14	The system allows error detection during data transmission and notifies the user in case of communication issues.
15	The system saves test results, including runtime and accuracy data, in a database for future analysis.
16	The system allows the user to view test results.

4.2.4.2. Non Functional Requirements

Table 5: Non Functional Requirements.

1	The algorithms implemented must efficiently perform within the TS-7553 OBC's constraints (250 MHz CPU and 64 MB of RAM).
2	The system must provide accurate TCA and distance calculations with minimal error.
3	The system must be able to handle various amounts of input data, allowing for scalability in the number of tracked objects
4	The system must operate efficiently under both wired and wireless communication protocols, ensuring minimal latency.
5	The system must be compatible with the ARM9 architecture of the TS-7553 OBC
6	The system must ensure reliable algorithm execution on the TS-7553 OBC, with error handling mechanisms for computational or hardware failures.
7	The system must be able to detect and correct errors in data transmission between the OBC and the testing station.
8	The system should to ensure compatibility with LAN communication.
9	The system must minimize latency during algorithm execution to support near real-time TCA and distance computations in critical scenarios.
10	The system should ensure that real-time performance is maintained, particularly in ensuring time synchronization between the OBC and the Test Station during data transfers and algorithm execution.
11	The system must be maintainable, allowing easy updates or modifications to the algorithm implementations.
12	The system must support a flexible configuration, allowing the user to adjust test parameters like time intervals and polynomial degrees.
13	The system should optimize the CATCH algorithm to efficiently find eigenvalues on the TS-7553 OBC to reduce overall computational cost.
14	The communication system must be able to handle large data transfers with minimal delay and ensure that data is reliably transferred.
15	The system should be scalable to support additional or updated algorithms and communication methods in future projects, ensuring it can be easily extended with minimal changes to core functionalities.
16	Error detection mechanisms (e.g., CRC checks) should be implemented for both the communication and the testing process to ensure data integrity.
17	The system need to optimize memory usage during algorithm execution to prevent the TS-7553 OBC from running out of available resources, such as RAM, while performing TCA and distance calculations.
18	The system must ensure that the CATCH, ANCAS, and SBO-ANCAS algorithms execute within a reasonable timeframe, particularly on the

	resource-constrained TS-7553 OBC, without significant delays in finding the TCA.

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. Test cases

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

5.2.1. Test-Station test cases

Table 6: 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.

5.2.2. Tested-OBC test cases

Table 7: 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.

5.3. Acceptance Tests

Table 8: Acceptance tests

Test Case ID	**Requirement ID**	**Test Description**	**Expected Outcome**	**Pass/Fail Criteria**
TC01	FR1	Execute CATCH, ANCAS, SBO-ANCAS on TS-7553 OBC	Algorithm correctly calculates TCA and distance	Pass if TCA and distance are accurately computed
TC02	FR2	Select and execute an algorithm on the OBC	Selected algorithm runs successfully	Pass if the selected algorithm executes without errors
TC03	FR3	Generate, store, manage test data	Data is correctly generated, stored, and managed	Pass if data integrity and availability are maintained
TC04	FR4	Establish wired LAN communication	Stable Ethernet connection established	Pass if connection is stable and data transfer is successful
TC05	FR13	Measure the runtime of the eigenvalue calculation in the CATCH algorithm on the TS-7553 OBC.	The algorithm's runtime is within the expected performance limits.	Pass if the runtime is within the acceptable bounds as defined in the performance specifications.
TC06	FR5	Open socket connection for data transfer	Data is successfully transferred via sockets	Pass if data is sent and received without errors

TC07	FR6	Ensure data transmission integrity and security	Data transmitted is secure and intact	Pass if data is received as sent without corruption
TC08	FR7	Create a Test Receipt	Test Receipt is correctly created with all parameters	Pass if receipt contains correct details as per input
TC09	FR8	Manage and send Point Data	Point Data is accurately managed and sent	Pass if data matches expected format and content
TC10	FR9	Perform tests on OBC and test station	Tests run correctly on both setups	Pass if both platforms yield correct and consistent results
TC11	FR11	Compare test results with real TCA calculation	Test results match with real TCA calculation	Pass if computed results are within acceptable error range
TC12	FR12	Measure runtime of algorithm on the TS-7553 OBC	Runtime is within expected limits	Pass if runtime is as per specification
TC13	FR14	Detect and display communication errors	Appropriate error messages are displayed	Pass if errors are correctly detected and displayed
TC14	FR15	Ensure that test results, including the runtime and accuracy data, are saved and can be queried from the database.	Test results are stored in the database and can be retrieved correctly for analysis.	Pass if the data is stored without error and can be accessed accurately for future use.
TC15	FR12	Compare algorithm performance on OBC and Test station	Performance metrics are comparable	Pass if performance is consistent across platforms
TC16	FR16	View and analyze test results	Results are viewable and analyzable	Pass if results display is clear and informative
TC17	FR12	Measure runtime and memory usage	Runtime and memory usage are within specified limits	Pass if both metrics meet system requirements
TC18	FR14	Detect errors during data transmission	Errors are detected and reported	Pass if all transmission errors are caught and reported

6. REFERENCES:

[1] Alfano, S. (1994). Determining Satellite Close Approaches-Part II. ADVANCES IN THE ASTRONAUTICAL SCIENCES, 87, 233 -233.

[2] Denenberg, E. (2020). Satellite closest approach calculation through chebyshev proxy polynomials. Acta Astronautica, 170, 55-65.

[3] Denenberg, E., & Gurfil, P. (2016). Improvements to time of closest approach calculation. Journal of Guidance, Control, and Dynamics, 39(9), 1967-1979.

[4] Kessler, D. J., Johnson, N. L., Liou, J. C., & Matney, M. (2010). The kessler syndrome: implications to future space operations. Advances in the Astronautical Sciences, 137(8), 2010.

[5] Vallado, D. A., Crawford, P., Hujsak, R., & Kelso, T. S. (2006, August). Revisiting spacetrack report# 3. AIAA-2006-6753. AIAA Astrodynamics Specialists Conference and Exhibit. Keystone, CO: American Institute of Aeronautics and Astronautics.

[6] Atmel SAMA5D27-SOM1-EK1 OBC's data

[7] C++ Coding Conventions

[8] CelesTrak

[9] CelesTrak Current Data

[10] Feasibility Analysis and Performance Testing of Collision Detection Algorithms for Satellites Project's GitHub repository

[11] Freethink article about space debris

[12] GOMSpace's website

[13] ISIS On Board Computer

[14] ISILAUNCH's website

[15] NASA on orbital debris

[16] NASA Procedural Requirements for Limiting Orbital Debris

[17] SAM9-L9260 OBC's data

[18] Satellite Orbital Conjunction Reports Assessing Threatening Encounters in Space

[19] SatSearch's website

[20] SGP4 python library

[21] STM32F429I-DISC1 OBC's data

[22] The Armadillo Library

[23] The Eigen Library

[24] TS-7553 OBC's data

[25] UN Space Debris Mitigation Guidelines

[26] OBC's Datasheets, GitHub repository

[27] Our GitHub repository

7. AI Prompts

During the research phase of our project, we used AI tools such as ChatGPT to validate our technology choices, improve grammar, and explore new feature ideas for our application. The following are examples of the prompts we used and how the AI's feedback influenced our research and development decisions.

[28] Write me short literature review on the Propagator SGP4. Remember to include references.
https://chatgpt.com/share/66e9f25b-5580-8006-873e-73a4bb0a32d4

[29] Create for me a two page literature review about [2] Denenberg, E. (2020). Satellite closest approach calculation through chebyshev proxy polynomials. Acta Astronautica, 170, 55-65. Remember to include references.
https://chatgpt.com/share/66e9f307-3c88-8006-8ab4-ed7fb3743687

[30] Create for me a two page literature review about Alfano, S. (1994). Determining Satellite Close Approaches-Part II. ADVANCES IN THE ASTRONAUTICAL SCIENCES, 87, 233-233. Remember to include references.
https://chatgpt.com/share/66e9f437-28d4-8006-985f-e0e22ec792b6

[31] Create for me a two page literature review about [3] Denenberg, E., & Gurfil, P. (2016). Improvements to time of closest approach calculation. Journal of Guidance, Control, and Dynamics, 39(9), 1967-1979. Remember to include references.
https://chatgpt.com/share/66e9f4ad-e3ec-8006-afd4-0fbe2e85b792

[32] Sort the reference articles in alphabetical order according to the first author of the article
https://chatgpt.com/share/66e9f684-d008-8006-887b-7a04f5d937e8