

Capstone Project Phase B

Feasibility analysis and performance testing of collision detection algorithms for satellites

23-2-D-17

Supervisor: Mr. Ilya Zeldner

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

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

[Abstract 2](#_Toc165033153)

[Keywords 2](#_Toc165033154)

[1. Introduction 3](#_Toc165033155)

[2. General Description 4](#_Toc165033156)

[2.1. Background 4](#_Toc165033157)

[2.1.1. The Algorithms 4](#_Toc165033158)

[2.1.1.1 ANCAS 4](#_Toc165033159)

[2.1.2. The Propagator 7](#_Toc165033160)

[2.2. Goals 7](#_Toc165033161)

[2.2.1. Implementing the Algorithms 7](#_Toc165033162)

[2.2.2. Creating a Testing System 7](#_Toc165033163)

[2.2.3. Feasibility Testing and Analysis 8](#_Toc165033164)

[2.3. Users 8](#_Toc165033165)

[3. The solution 9](#_Toc165033166)

[3.1. Algorithms Analysis 9](#_Toc165033167)

[3.1.1. Algorithms Complexity 9](#_Toc165033168)

[3.2. Algorithms implementation 12](#_Toc165033169)

[3.2.1. ANCAS Implementation 12](#_Toc165033170)

[3.2.2. SBO-ANCAS Implementation 12](#_Toc165033171)

[3.2.3. CATCH Implementation 12](#_Toc165033172)

[3.3. Testing system 12](#_Toc165033173)

[3.3.1. Testing Station App 12](#_Toc165033174)

[3.3.2. Testing OBC App 12](#_Toc165033175)

[3.3.3. Feasibility Testing Environments 12](#_Toc165033176)

[3.4. Research and Development process 12](#_Toc165033177)

[3.5. Development tools 12](#_Toc165033178)

[3.6. Problems and solutions 13](#_Toc165033179)

[3.7. Feasibility Test Results 13](#_Toc165033180)

[3.8. Results and conclusion 13](#_Toc165033181)

[4. User guide 13](#_Toc165033182)

[5. Maintenance Guide 13](#_Toc165033183)

[6. REFERENCES: 14](#_Toc165033184)

# Abstract

The project we present is based on algorithms for finding the minimal distance between two objects in space, and the time of occurrence, which will have to work on satellite on-board computer (OBC). The project has two goals, the first goal is to implement the algorithms and deliver them ready to integrate and use for future testing and projects, and the second goal is to create a system for testing the algorithms performance on a satellite OBC or an emulated environment and conduct a feasibility study.

The project is based on Dr. Elad Denenberg's research papers that introduced the algorithms **[**[**1**](#Reference1)**][ ]**.

# Keywords

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

# Introduction

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

With the increasing number of objects in Earth orbit, around 27,000**[**[**3**](#Reference3)**]** and the shift to cluster of smaller satellites instead of a single big one **[**[**1**](#Reference1)**]** the cost of calculating the orbit of objects and finding the TCA for our satellite is only growing. To solve this problem a few cheaper algorithms were developed. The algorithms are supposed to be cheap and fast enough to run on the satellite’s own ***on-board computer (OBC)***. These algorithms have not been tested and we need to prove the task feasibility, to implement the algorithms and to show the calculation time and memory requirements in an environment with limited calculation power and memory simulating an actual satellite on-board computer. We are working with Dr. Elad Denenberg, who created the **Conjunction Assessment Through Chebyshev Polynomials (CATCH)** algorithm **[**[**1**](#Reference1)**]** and the **SBO-ANCAS** algorithm[ ]. Dr. Elad is working on creating an autonomous satellite and as part of his work he need our help. An autonomous satellite needs to calculate the possible collisions by itself and for doing that a fast algorithm for finding the TCA is needed, faster calculations are possible using approximations like the algorithms CATCH **[**[**1**](#Reference1)**],** SBO-ANCAS[ ] and **Alfano\Negron Close Approach Software (ANCAS)** **[**[**2**](#Reference2)**]**. These algorithms were never tested on an actual satellite on-board computer, and proved fitting to run on an autonomous satellite and this is we come in.

# General Description

## Background

### The Algorithms

In this project we implemented and tested three algorithms, all three can be used to calculate the TCA. The following is a description of the algorithms.

### ANCAS

The first algorithm, ANCAS **[**[**2**](#Reference2)**]** uses cubic polynomial as an approximation of a function over an interval. Given n points in time and the respective location and velocity vectors for 2 objects, we can find the TCA by:

Algorithm 1: ANCAS on n points, (the original algorithms description can be found at **[**[**2**](#Reference2)**]**)

**Input**:

**Output**:

**for** each set of 4 points **do:**

Map the time points to on the interval

Calculate using **Eq.([5](#eq5)/**[**2**](#eq2)**)** with the points

Fit cubic polynomial to according to **[**[**2**](#Reference2) **,Eq.1f-1j]** over

Find the cubic polynomial real roots in the interval

Fit cubic polynomials for in the interval

**for** each root **do:**

calculate the distance using **Eq.([6](#eq6))**

**if** **:**

**end**

**end**

**end**

The cubic polynomial coefficients calculations described in article **[**[**2**](#Reference2)**]**.

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.

#### SBO-ANCAS

The second algorithm, SBO-ANCAS **[**[**2**](#Reference2)**]** 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:

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

**Input**: ,,

**Output**:

**for** each set of 4 points **do:**

**Do**

Map the time points to on the interval

Calculate using **Eq.([5](#eq5)/**[**2**](#eq2)**)** with the points

Fit cubic polynomial to according to **[**[**2**](#Reference2) **,Eq.1f-1j]** over

Find the cubic polynomial real roots in the interval

Fit cubic polynomials for in the interval

**for** each root  **do:**

calculate the distance at  using **Eq.([6](#eq6))**

**if** **:**

**end**

**end**

Sample and at using a propagator

**While**  OR

**end**

The cubic polynomial coefficients calculations described in article **[**[**2**](#Reference2)**]**.

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 we use the propagator to sample the location and velocity vectors at , 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.

#### CATCH

The third algorithm, CATCH **[**[**1**](#Reference1)**]**, uses **Chebyshev Proxy Polynomial (CPP)[[1](#Reference1)]** 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 , each iteration searches the minimal distance in an interval with size . 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 **[**[**1**](#Reference1)**, algorithm 2]**

**Input**: 

**Output**: 









**While** **do**:

Fit CPP  with order N to  according to **[**[**1**](#Reference1) **, Eq.15]** over the interval 

Fit CPP with order N to  over the interval 

Find the roots of 

**for** each root  **do:**

calculate the distance  at  using Eq.([6](#eq6))

**if** **:**



**end**

**end**

****

**end**

The algorithm needs N+1 points in time in each Gamma interval in order to create CPP of order N. After calculating the CPP coefficients we can use them to create a special NxN matrix called the companion Matrix **[**[**1**](#Reference1)**,Eq.18]** and the eigen values of this matrix are the polynomial roots. Using the roots, we found and creating CPP for  we can calculate the minimal distance in each interval and eventually the TCA and respective distance in . The problem with CATCH is the cost of finding the roots, which is the cost of finding eigen values for an NxN matrix, to deal with it, Dr. Elad describe in his article**[[1](#Reference1),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.

### The Propagator

Using the current location and velocity of an object in orbit (for example our satellite) and a point in time (for example ten minutes from now) the Propagator can calculate the object location and velocity in the given point in time. The propagator uses a forces model to find the future orbit, this force model is different between different propagators and can affect the result’s accuracy and the calculation time. In this project we use a propagator called **Standard General Perturbations Satellite Orbit Model 4(SGP4)** **[**[**10**](#Reference10)**]**, SGP4 is old and famous propagator used for research that known for its fast run time and there are many available implementations we can use **[**[**4**](#Reference4)**]**. 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.

We used the propagator for two task, 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.

## Goals

In this project we have three main goals.

### Implementing the Algorithms

Our first goal is to implement the algorithms themselves and doing so while considering the environment the algorithm will have to work on. Until now SBO-ANCAS and CATCH were only implemented in MATLAB as part of the initial article and testing [ ][ ]. To work on a satellite OBC well the algorithms need to run efficiently on various systems and computer boards. We started implementing the algorithms in the first part of the project as a feasibility proof for our project and completed the implementation in this part.

### Creating a Testing System

To test the feasibility of running the algorithms on satellites OBC we needed a system fitting for running test and collecting data and results. We needed to run the algorithm on a dedicated system or emulator with a given set of data and parameters as input, to get the output and run time and to save the results and test related parameters in our data set in order to collect enough data on the algorithms expected run time and accuracy in different scenarios. The Testing System needed to be flexible enough to run the algorithms on different machine and environments and manage and collect the data well. We needed the system both for running the feasibility analysis ourself and for leaving it for DR. Elad to use for his research.

### Feasibility Testing and Analysis

The last part of our project is to conduct a feasibility analysis using our system. We needed to run our system with different types of inputs, different types of algorithms and different parameters for each algorithm. TBD

## Users

For our system we have two types of users, the first is DR. Elad and his associates wanting to test the algorithms against an emulator for now and a real satellite’s OBC in the future. The second type of user is us, wanting to run a feasibility test as part of our project, and any possible follow up team that will take part in the ongoing effort of creating and proving the feasibility of the autonomous satellites. (there are already other teams working in their projects toward this goal, that might need to use our system or expend it).

# The solution

## Algorithms Analysis

### Algorithms Complexity

#### ANCAS Time Complexity

In ANCAS **[**[**2**](#Reference2)**]**, for each set of 4 data points we need to create 4 cubic polynomials, one for the relative distance derivative and 3 for the relative distance X, Y, Z. each cubic polynomials required coefficients calculation which consist of 4 equations **[**[**2**](#Reference2)**,Eq 1f-1j]**, meaning the complexity of finding the polynomial coefficients is constant. To map the time points to the interval [0,1] we use a simple calculation for each point **[**[**2**](#Reference2)**]** 4 times, one for each point.

Finding the solution for a 3rd degree equations is quite simple, using a given formula with a constant run time we get between 1 to 3 real result.

For each of the roots we found, we calculate the distance using **Eq.([6](#eq6)),** 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):



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  iterations.

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



#### SBO-ANCAS Time Complexity

In SBO-ANCAS **[** **]**, we are going over a set of 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  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, , with the initial distance between 2 time points of , and tolerance in time . To get to the desired tolerance we need the distance between to the other points to be smaller than . which means that at the last iteration we get:

.

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, :

And so on.

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

And we continue until:

We expect the distance between 2 points, , to be bigger than the tolerance and with a small enough we get:

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 , 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 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.

#### CATCH Time Complexity

In CATCH**[**[**1**](#Reference1)**]** algorithm we iterate through the number of time points in our external loop, 

Where {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><msub><mi>t</mi><mrow><mi>m</mi><mi>a</mi><mi>x</mi></mrow></msub></mstyle></math>"} is the is end boundry in the time range where we're looking for mininal disdance, and equal to half of the smaller revolution time of the object **[**[**1**](#Reference1)**,part 4]**, The value of {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub></mstyle></math>"} is the order of the polynomial, while we can change the chosen value of N, it was determined that {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>N</mi><mo>=</mo><mn>16</mn></mstyle></math>"} give sufficient results.

Inside the loop we're doing the following steps:

1. Fit the CPP of order {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub></mstyle></math>"} to {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mover><mi>f</mi><mo>&#x2D9;</mo></mover><mfenced><mi>t</mi></mfenced><mo>,</mo><mo>&#xA0;</mo><msub><mi>p</mi><mi>x</mi></msub><mo>,</mo><mo>&#xA0;</mo><msub><mi>p</mi><mi>y</mi></msub><mo>,</mo><mo>&#xA0;</mo><msub><mi>p</mi><mi>z</mi></msub></mstyle></math>"} over each interval of points:

Assuming the arithmetic operations we use are basic operation done in time complexity of {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>O</mi><mfenced><mn>1</mn></mfenced></mstyle></math>"}, we calculate the Chebyshev polynomials**[[1](#Reference1)]**. We'll iterate through {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub><mo>+</mo><mn>1</mn></mstyle></math>"} 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 {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mfenced><mrow><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub><mo>+</mo><mn>1</mn></mrow></mfenced><mfenced><mrow><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub><mo>+</mo><mn>1</mn></mrow></mfenced></mstyle></math>"}which is also constant.

The complexity of this step is: {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>O</mi><mfenced><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub></mfenced><mo>&#xB7;</mo><mfenced><mrow><mi>O</mi><mfenced><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub></mfenced><mo>&#xB7;</mo><mfenced><mrow><mi>O</mi><mfenced><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub></mfenced><mo>&#xB7;</mo><mi>O</mi><mfenced><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub></mfenced></mrow></mfenced><mo>+</mo><mi>O</mi><mfenced><mn>1</mn></mfenced></mrow></mfenced><mo>&#xA0;</mo><mo>=</mo><mo>&#xA0;</mo><mi>O</mi><mfenced><mn>1</mn></mfenced></mstyle></math>"}

1. Finding the roots for {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><msub><mi>P</mi><mi>f</mi></msub></mstyle></math>"} will consist of calculating the companion matrix with a size of {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><msup><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub><mn>2</mn></msup></mstyle></math>"} and finding the eigen values, using the complexity of matrix multiplication for this step, the complexity will be {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>O</mi><mfenced><msup><msub><mi>N</mi><mrow><mi>d</mi><mi>e</mi><mi>g</mi></mrow></msub><mn>3</mn></msup></mfenced><mo>&#xA0;</mo><mo>=</mo><mo>&#xA0;</mo><mi>O</mi><mfenced><mn>1</mn></mfenced></mstyle></math>"}, rescaling each eigen value to the actual coefficient value also takes constant time.
2. 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:

#### 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:

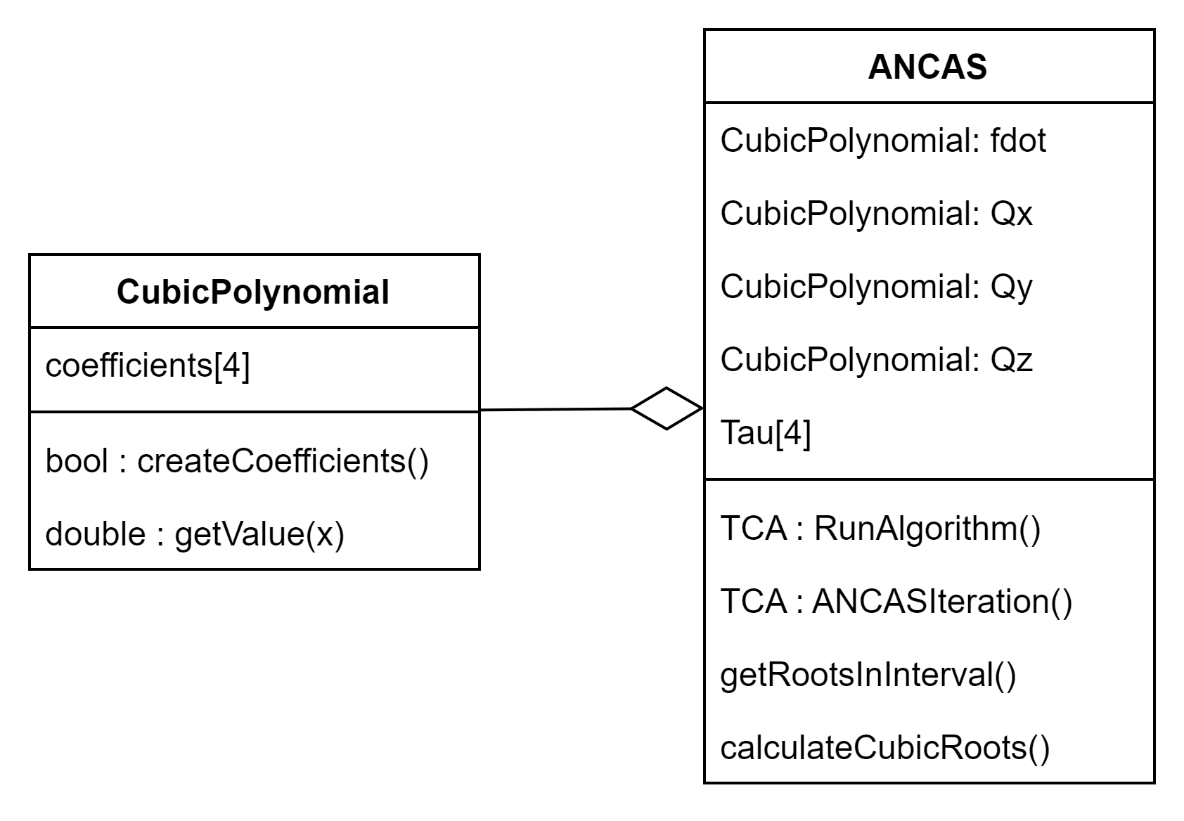
\*For SBO-ANCAS, we assume that the propogator uses a constant amount of memory, but this may not always be the case.

## Algorithms implementation

### ANCAS Implementation

ANCAS 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. 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 polynomials with 0 to 3 real roots in range.

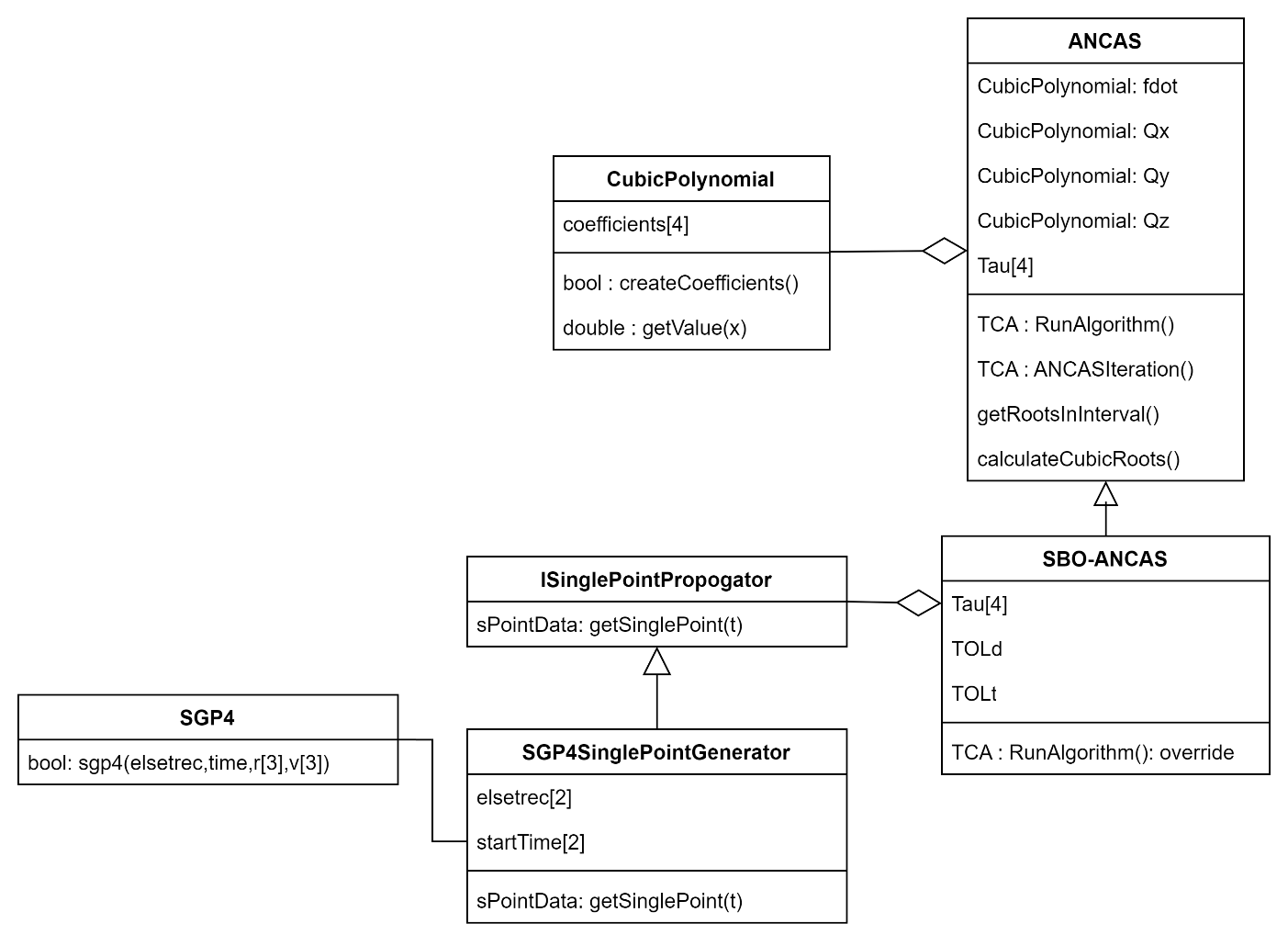
Diagram – Class diagram for ANCAS. Including a function for calculating the roots of a cubic polynomial used by a function for getting the roots in the interval 0,1. The function for ANCAS Iteration return the found minimum and time, used for a single ANCAS iteration.



### SBO-ANCAS Implementation

SBO-ANCAS acts similar to ANCAS 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 get 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.

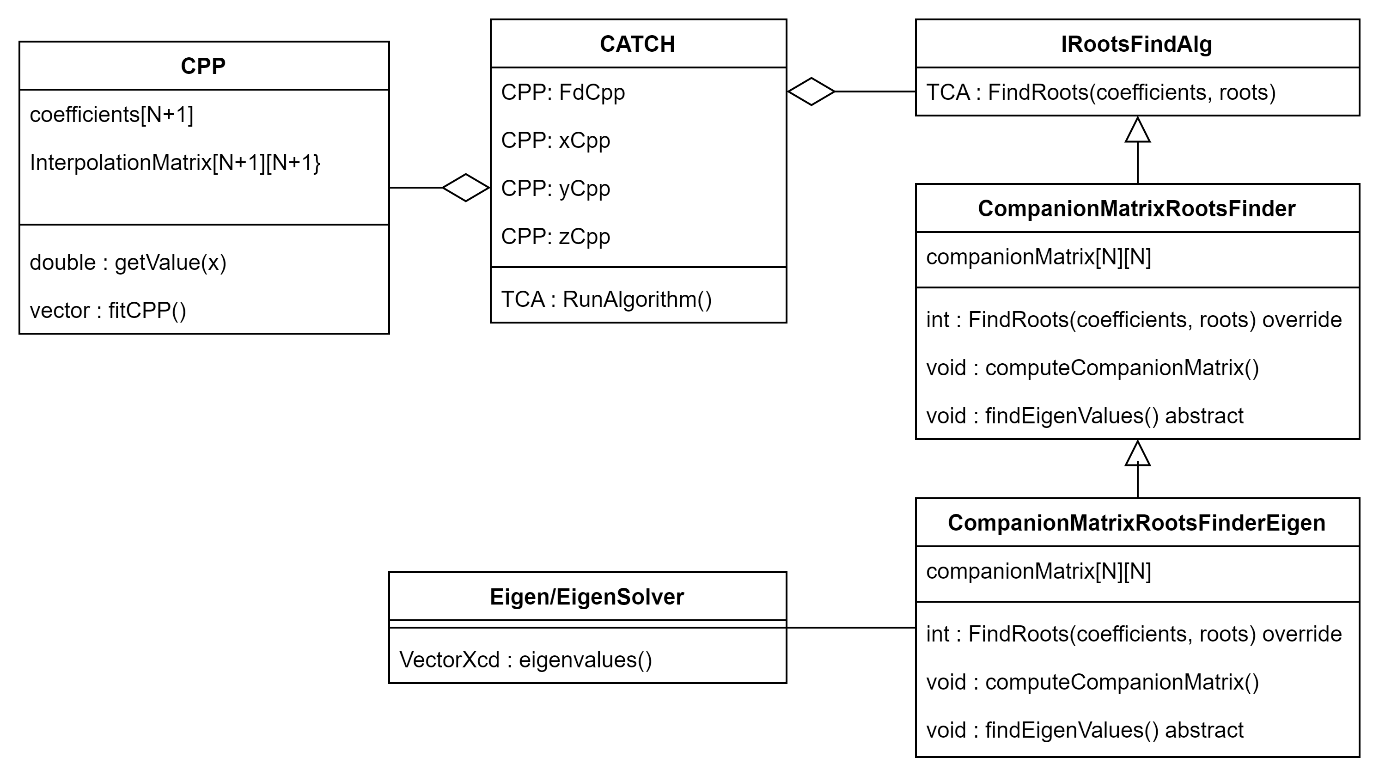
Diagram – Class diagram for SBO-ANCAS. Including the Propagator interface and implementation and the tolerances.



### CATCH Implementation

CATCH implementation required implementing the Chebyshev Proxy Polynomials(CPP) class, with function for calculating the polynomial coefficients, 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 [ ] , and we tried two libraries for finding the Eigenvalues of the Companion Matrix. We implemented using Eigen and Armadillo. Unfortunately, the Armadillo library is quite heavy (while using the library the code uses around 400MB) and its too much for the satellite’s OBC so we removed the Armadillo implementation.

Diagram – Class diagram for CATCH, including Rootfinder interface and implementations.

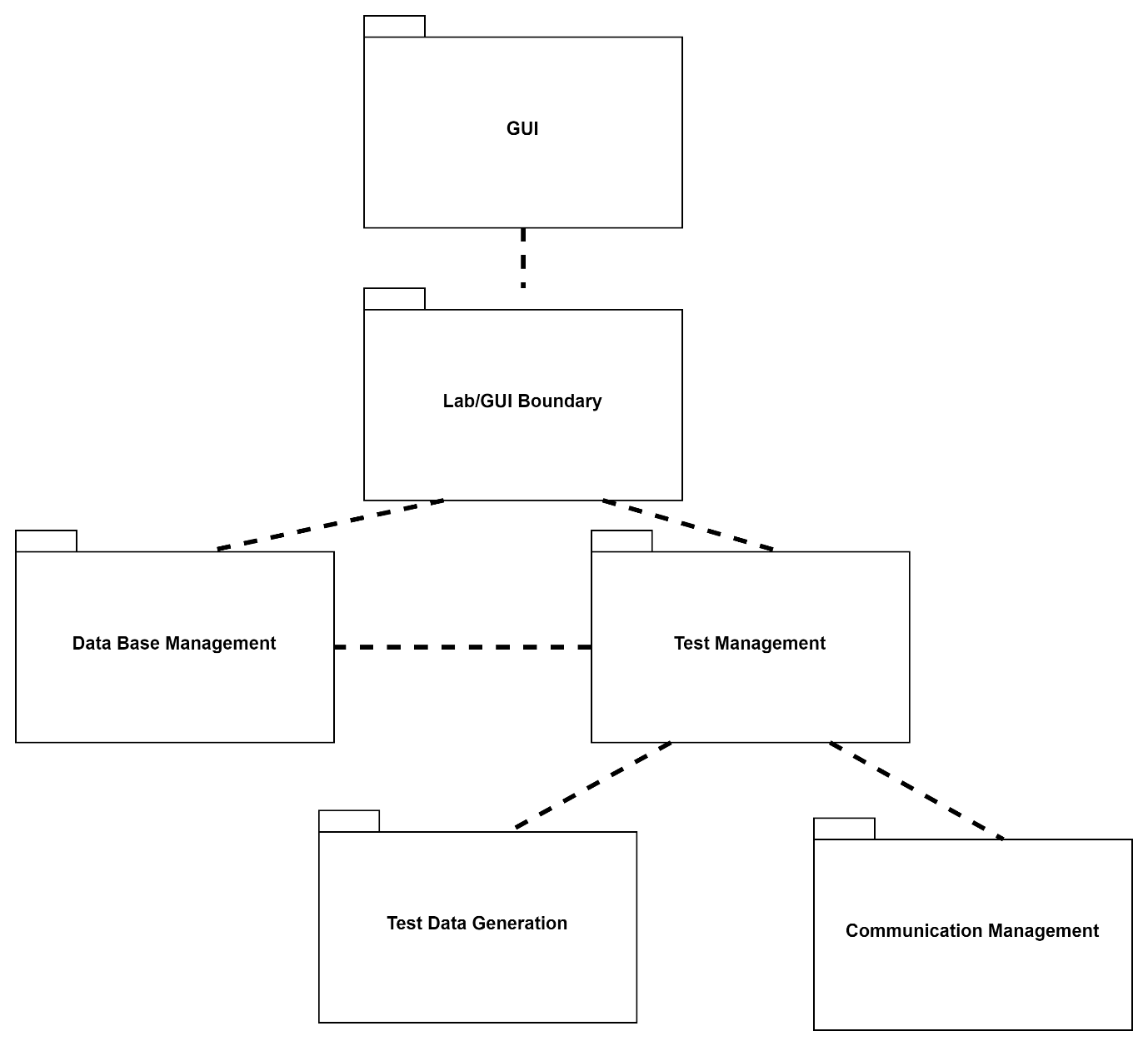


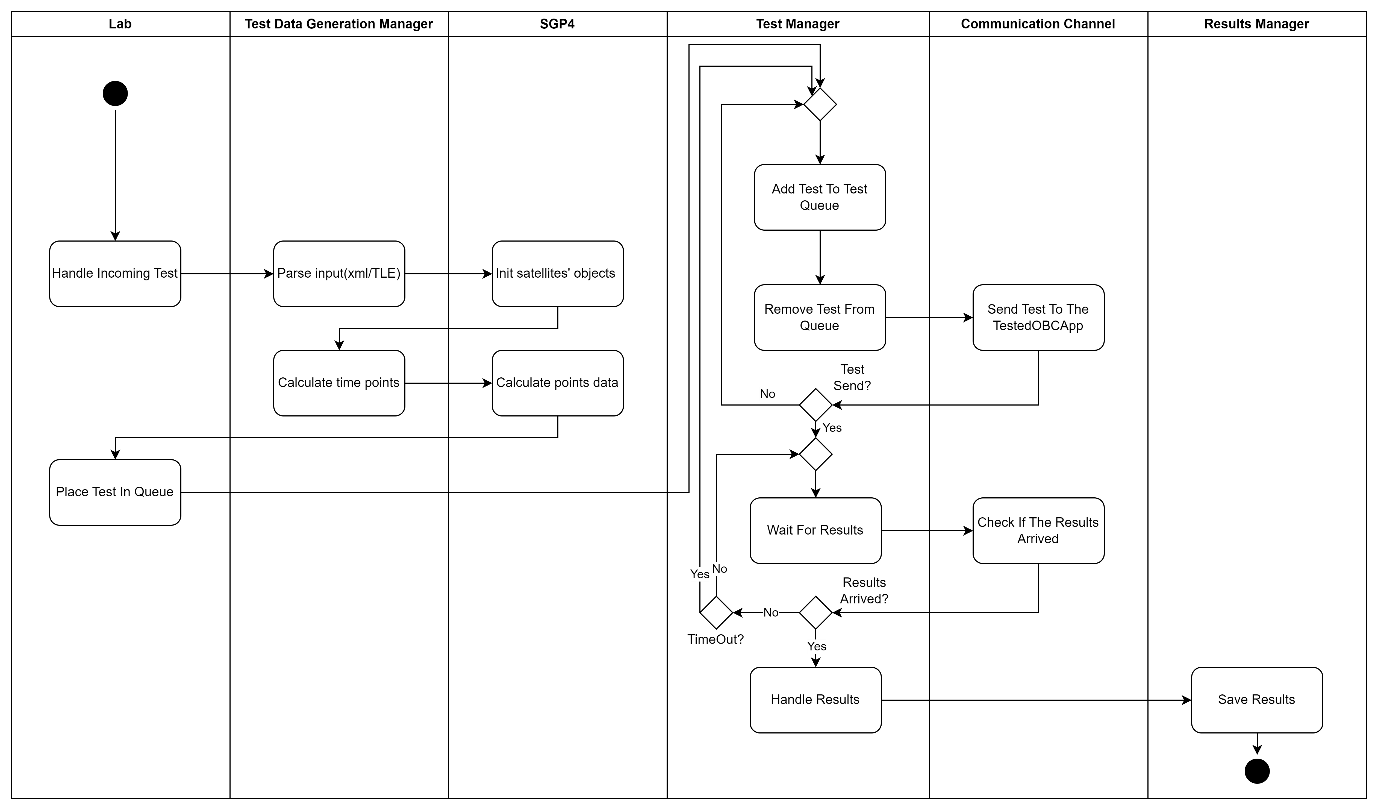
## Testing system

Top view activity diagram for running a test, with the full 2Apps system

### Testing Station App

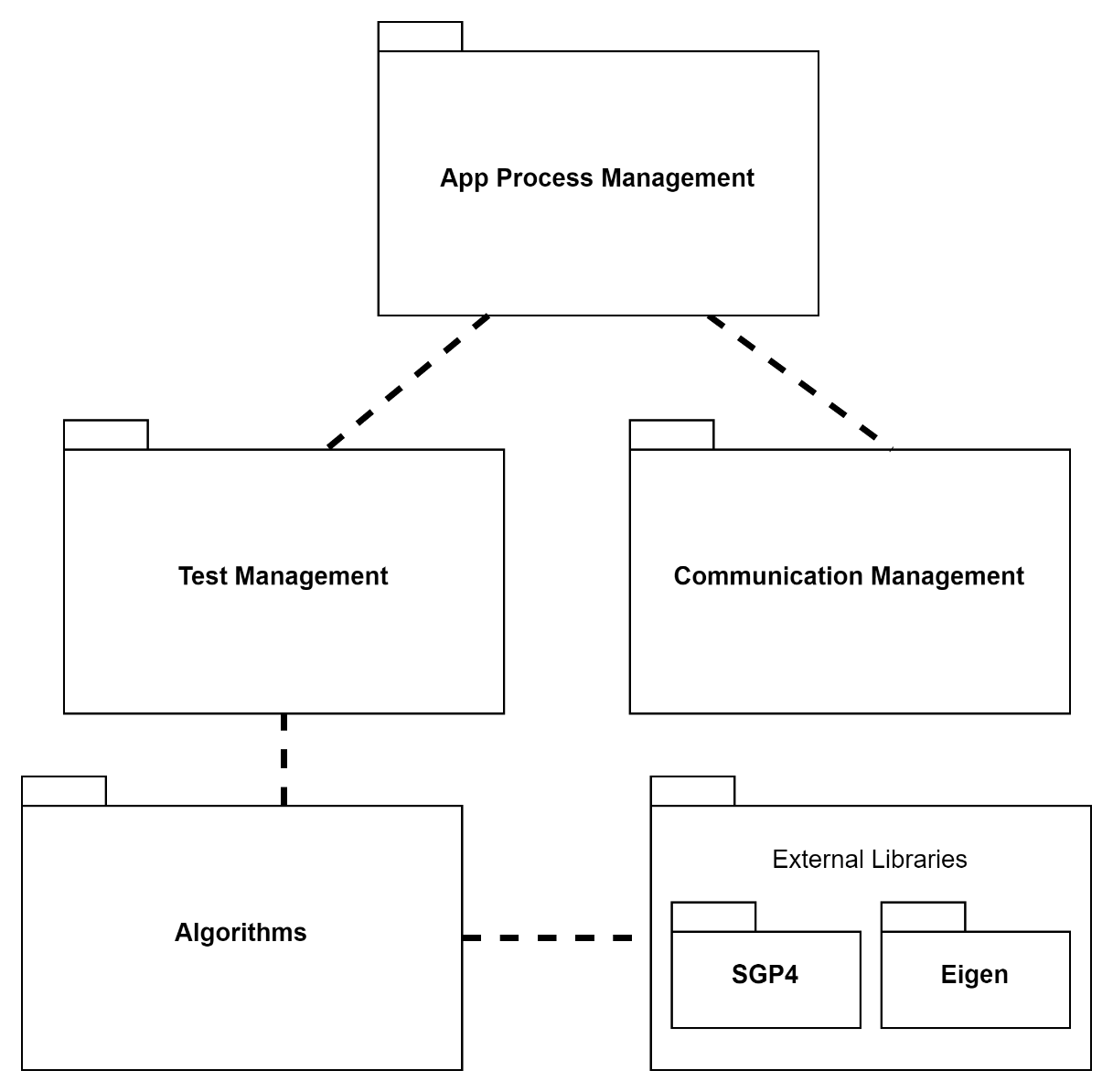
Package diagram and top down explanation of the system, class diagram for each package with explanations. Activity for running a test





### Testing OBC App

Package diagram and top down explanation of the system, class diagram for each package with explanations. Activity for running a test



### Feasibility Testing Environments

Emulator, RP4, Windows. The different system we can work with

## Research and Development process

What we did, development process and so on

Testing, unit tests, testing the system.

## Development tools

Development tools we used – VS, CMAKE, GIT, GTest

## Problems and solutions

Cross platform communication, synchronizing stuff, database management

## Feasibility Test Results

Running and analyzing the results!! important

## Results and conclusion

Conclusion. Screen shots of the app and so on.

# User guide

How to run the system with screenshots and examples.

# Maintenance Guide

Starting with installation (scripts, requirements) for each system.

Requirement for compiling after changes, used libraries, how to add stuff(catch roots finding algorithms for example)

# REFERENCES:

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

[2] Alfano, S. "Determining Satellite Close Approaches-Part II."

<https://www.researchgate.net/publication/252559100_Determining_satellite_close_approaches_part_II>

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

[4] SGP4 python library :

<https://pypi.org/project/sgp4/>

[5] NASA Procedural Requirements for Limiting Orbital Debris:

<https://www.iadc-home.org/references/pdfview/id/74>

[6] UN Space Debris Mitigation Guidelines:

<https://www.iadc-home.org/references/pdfview/id/75>

[7] CelesTrak:

<https://celestrak.org/>

[8] Satellite Orbital Conjunction Reports Assessing Threatening Encounters in Space:

<https://celestrak.org/SOCRATES/>

[9] The Kessler Syndrome: Implications to Future Space operations:

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=227655e022441d1379dfdc395173ed2e776d54ee>

[10] SGP4: Revisiting Spacetrack Report :

<https://celestrak.org/publications/AIAA/2006-6753/AIAA-2006-6753.pdf>

[11]Google test framework

<https://github.com/google/googletest>

[12] EnduroSat’s Onboard Computer Type I:

<https://www.endurosat.com/cubesat-store/cubesat-obc/onboard-computer-obc/>

[13] Virtual COM Port Driver:

<https://www.virtual-serial-port.org/>

[14] Freethink article about space debris:

<https://www.freethink.com/space/space-debris-15000mph>