Software and Embedded Systems Engineering
Fakultät IV Elektrotechnik und Informatik
Technische Universität Berlin

Automated Test Generation for Satellite On-Board Image Processing

Master Thesis

for the degree of

Master of Science

Ulrike Witteck
Matr.-Nr. 376475

07. September 2018

Supervisor:

Prof. Dr. Sabine Glesner
Prof. Dr. Olaf Hellwich

Sworn Affidavit

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

Berlin, 07. September 2018

Signature

Abstract

On-board image processing technologies in the satellite domain are subject to extremely strict requirements with respect to reliability and accuracy in hard real-time. Due to their large input domain, it is infeasible to exhaustively execute all possible test cases. Furthermore, because of their complex computations, it is difficult to find specific test cases that provoke mission-critical behavior.

To overcome these problems, we first define a test approach that efficiently and systematically captures the input domain of satellite on-board image processing applications. We present a dedicated partitioning into equivalence classes for each input parameter. As a result, our approach systematically reduces the number of test cases. Moreover, we define novel multidimensional coverage criteria to assess a given test suite for its coverage on the input domain. We present a test generation algorithm that automatically inserts missing test cases into the given test suite based on our multidimensional coverage criteria. This results in a reasonably small test suite that covers the whole input domain of satellite on-board image processing applications.

Second, we define a test approach that automatically searches for test cases that are specifically tailored to provoke mission-critical behavior of satellite on-board image processing applications. For that, we present a novel genetic algorithm. We define a two-criteria fitness function that is based on the execution time and mathematical accuracy of the application under test. Therefore, our algorithm automatically selects test cases that provoke worse execution times and inaccurate results of the satellite on-board image processing application.

We investigate the efficiency of our approaches on the PLAnetary Transits and Oscillation of stars (PLATO) Fine Guidance System (FGS) algorithm. This is a satellite on-board algorithm that calculates the high-precision attitude of the spacecraft. The experimental results show that our first approach efficiently and systematically generates a test suite. This suite completely covers the input domain with respect to our multidimensional coverage criteria. This test suite has a higher error-detection capability than a randomly generated test suite. Therefore, our test approach increases the test efficiency and quality. Furthermore, our genetic algorithm automatically finds test cases that provoke longer execution times and less accurate results when using the generated test suite from the first approach as search space than using a randomly generated test suite. Hence, our genetic approach improves a given test suite to support robustness testing.

As a summary, the combination of our approaches increases the efficiency and effectiveness of the test process for satellite image processing applications.

Zusammenfassung

Bordeigene Bildverarbeitungsanwendungen im Bereich der Satellitentechnik unterliegen extrem hohen Anforderungen bezüglich Zuverlässigkeit, Genauigkeit und Echtzeitfähigkeit. Die Menge an Testfällen ist extrem groß. Daher ist es nicht möglich alle Testfälle auszuführen. Zudem ist es aufgrund der komplexen Berechnungen schwer, Testfälle zu finden, die missionskritisches Verhalten provozieren.

Als Lösung, präsentieren wir zuerst einen Testansatz, der die Eingabedomäne von Bildverarbeitungsanwendungen eines Satelliten effizient und systematisch erfasst. Dazu unterteilen wir jeden Eingangsparameter in speziell definierte Äquivalenzklassen. Auf diese Weise wird die Anzahl der Testfälle systematisch reduziert. Um eine bestimmte Testsuite hinsichtlich ihrer Abdeckung auf der Eingabedomäne zu bewerten, definieren wir neuartige multidimensionale Abdeckungskriterien. Zusätzlich präsentieren wir einen Algorithmus zur Testgenerierung, der fehlende Testfälle basierend auf unseren mehrdimensionalen Abdeckungskriterien automatisch in die gegebene Testsuite einfügt. Das ergibt eine reduzierte Testsuite, die die gesamte Eingabedomäne von Bildverarbeitungsanwendungen für Satelliten abdeckt.

Wir definieren einen zweiten Testansatz, der automatisch nach Testfällen sucht. Diese sind speziell auf das missionskritische Verhalten von Bildverarbeitungsanwendungen eines Satelliten zugeschnitten. Für diesen Zweck stellen wir einen neuartigen genetischen Algorithmus vor, der eine Fitnessfunktion mit zwei Kriterien nutzt. Diese Funktion basiert auf der Ausführungszeit und der mathematischen Genauigkeit der zu prüfenden Anwendung. Daher wählt unser Algorithmus automatisch Testfälle aus, die zu schlechteren Ausführungszeiten und ungenauen Ergebnissen der Bildverarbeitungsanwendung führen.

Wir untersuchen die Effizienz unserer Ansätze anhand des Algorithmus des PLATO FGS. Hierbei handelt es sich um einen bordeigenen Algorithmus, der die hochpräzise Lage des Satelliten berechnet. Die Versuchsergebnisse zeigen, dass unser erster Testansatz eine Testsuite effizient und systematisch generiert. Diese Testsuite deckt die Eingabedomäne in Bezug auf unsere multidimensionalen Abdeckungskriterien vollständig ab und verfügt daher über eine höhere Fehlererkennung als eine zufällig generierte Testsuite. Somit steigert unser Testansatz die Testeffizienz und -qualität. Wird die Testsuite vom ersten Ansatz als Suchraum für unseren genetischen Algorithmus verwendet, findet dieser Testfälle, die zu längeren Ausführungszeiten und weniger genauen Ergebnissen führen, als bei der Verwendung einer zufällig generierten Testsuite. Somit verbessert unser genetischer Ansatz eine gegebene Testsuite, um Robustheitstests zu unterstützen.

Hieraus ergibt sich, dass die Kombination unserer Ansätze die Effizienz und Effektivität des Testprozesses für Anwendungen der Satellitenbildverarbeitung erhöht.

Acknowledgements

I would like to thank my academic advisor Dr. Paula Herber from the Technical University (TU) Berlin for her encouragement and guidance as well as her continuous support. She gave me valuable advice regarding the structure and content of this thesis.

I also thank Dr. Denis Grießbach from the German Aerospace Center (DLR) for introducing me to the topic of the Fine Guidance System (FGS). He patiently answered my questions to the topic and gave me insightful comments and suggestions about my work.

In addition, I thank Gisbert Peter from the DLR for his many years of support and for initiating the topic of this thesis.

My thank also go to Bernd Ulmer from Ingenieurbüro Ulmer for his support with technical knowledge about satellite on-board applications, their design, and development. I thank Mr. Ulmer and Karsten Westerdorff from the DLR for their work to run the FGS algorithm on the target board.

I would like to thank Rainer Berlin from the DLR for his help on packet definitions for the communication between the test application and the application under test.

Finally, my thanks go to all named persons as well as to Claas Ziemke and Eilke Santjer from the DLR as proofreaders of this thesis. I am thankful for their valuable comments on this work.

Contents

1 Introduction

Various on-board image processing applications are subject to strict requirements with respect to reliability and accuracy in hard real-time. Due to the large input domain of such applications, testing the systems manually is error-prone and time-consuming. Hence, a test approach is needed that automatically and systematically generates test cases for testing such applications. However, the automated test generation for on-board image processing applications poses two major challenges: First, the large amount of input parameters and their possible combinations leads to a high number of test cases. Hence, the systematic and efficient coverage of the whole input domain is expensive. Second, many on-board image processing applications perform complex algorithmic computations. Therefore, it is difficult to find all test cases with a high probability that provoke mission-critical behavior. That means, scenarios where, for example, the real-time behavior or the delivered mathematical accuracy does not meet specified requirements. This may cause system failures, damages, or unexpected behaviors during mission lifetime.

Previous work [BK06][HP16][SBW01][MA00][VM14] presents diverse automated test approaches for several systems in various domains, for example for the automotive and the railway applications. The studies investigate systems with huge input domains and complex functional-behavior. However, these approaches do not define test requirements tailored to the specific domain of satellite on-board image processing applications, as considered in this thesis. Moreover, the presented approaches are not designed to search for test cases provoking real-time critical behavior and scenarios where the accuracy of the application gets critically low.

In this thesis, we present two novel test approaches for the specific domain of satellite on-board image processing applications: The first approach systematically selects test cases from the huge input domain of such applications. Our objective is to achieve a high coverage of the input domain while at the same time using reasonably small test suites. The second approach automatically generates test cases that provoke mission-critical behavior of the system. In this way, we aim for an improvement of a given test suite to support robustness testing.

For our first proposed approach, we adopt the equivalence class partition testing method. In general, this method partitions a given input domain or output domain into disjoint sub-domains called equivalence classes [VM14]. The use of some test values as representatives of each class reduces the number of required test cases [BQ15]. In our test approach, we specify a dedicated partitioning for each input parameter of the satellite on-board image processing application. At the same time, we present various concepts for the partitioning of image processing input data that can be transferred to other domains. Moreover, we define multidimensional coverage criteria for this specific application domain. The proposed criteria unite the individual coverage criteria of the input parameters. We present a test gen-

eration approach that uses our multidimensional criteria to automatically assess given test suites with respect to their coverage of input parameter combinations. In this way, our test approach enables efficient test case generation for the domain of such applications.

To reach the goal of our second approach, we define a genetic algorithm. It automatically searches for test cases that provoke mission-critical behavior with respect to run time and mathematical accuracy of the on-board image processing application. In general, a genetic algorithm solves search or optimization problems by applying evolutionary mechanisms. It evaluates solutions with respect to given criteria using a so-called fitness function. Then it improves the best solutions to satisfy these criteria [Moh05]. For our proposed test approach, we define a novel two-criteria fitness function based on the real-time behavior and mathematical accuracy provided by a satellite on-board image processing application. With that function, our genetic algorithm automatically steers the search of test cases that provoke long execution times and inaccurate results of the application.

To investigate the efficiency of our proposed test approaches, we used the Fine Guidance System (FGS) algorithm of the PLAnetary Transits and Oscillation of stars (PLATO) mission as a case study. It is a special satellite on-board image processing algorithm that calculates the high-precision attitude of the spacecraft by comparing tracked star positions in image frames taken on board with known star positions in a star catalog. The experimental results demonstrate the efficiency of the partitioning approach due to an increased error-detection capability of a given test suite. Furthermore, they show the efficiency of the genetic approach due to the automated search of specific test cases tailored for robustness testing.

This thesis is structured as follows: Section 2 gives a brief overview of equivalence class partition testing and describes the concept of genetic algorithms in general. Furthermore, it presents an introduction of the PLATO mission and the PLATO FGS algorithm. Section 3 outlines related work on equivalence class partition testing and on the use of genetic algorithms for test case generation in practice. Section 4 presents our definitions of the proposed equivalence class partitioning approach by means of the case study. First, it gives an introduction of the input parameters of the PLATO FGS algorithm and presents our equivalence class formulations of these specific parameters. Then, it shows our definition of multidimensional coverage criteria on the specified equivalence classes and sketches the overall test generation process. Section 5 presents our proposed genetic algorithm. First, it provides our description of the algorithm components. Then it gives an overview of our automated test generation approach. Our implementation of both approaches is represented in Section 6. After that, the section provides a summary of the practical results for both approaches. The thesis concludes with a summary of the main results and gives an overview of future work.

2 Background

This section gives an overview of equivalence class partition testing in general and describes the concept of genetic algorithms. After that, it outlines the PLATO mission and the mission-critical PLATO FGS algorithm.

2.1 Equivalence Class Partition Testing

To make testing more efficient and less time consuming, it is preferable to examine as many test cases as necessary to satisfy specified test criteria. However, the selection of a few test cases from a huge input domain is a major problem when testing an application. Test cases should be representatives of the entire input domain. At the same time, they should increase the number of detected errors in the code [Pet09].

Equivalence class partition testing offers a possible solution to this problem. It is a commonly used approach in practice. The technique partitions a given input domain or output domain into disjoint sub-domains, the equivalence classes. The method partitions the domain in such a way, that all elements in an equivalence class are expected to provoke the same system behavior according to a specification. The partitioning is based on one or multiple input parameters specified in the requirements. Equivalence classes represent subsets of parameter values that completely cover the input or output domain. For the purpose of software testing, it is therefore sufficient to test some representative values of each equivalence class. Broekman and Notenboom [BN03] explain the method on following small example: assume that a system behavior is subjected to a given temperature condition:

$$15.0 \leq \textit{system temperature} \leq 40.0 \tag{1}$$

Testing this system requires a huge number of test cases. However, applying the equivalence class partitioning approach leads to three equivalence classes:

- the system temperature is lower than 15.0
- the system temperature value is in the range from 15.0 to 40.0
- the system temperature is higher than 40.0

In this case, three test cases, are sufficient to cover the whole input domain of the parameter. Example test cases for each equivalence class are: 10.0 (invalid), 35.0 (valid), 75.0 (invalid).

The example demonstrates that the application of equivalence class partition testing leads to a limited number of required test cases. The selection of test cases from equivalence classes can be made according to various criteria: using border values, testing special values or randomly selecting test cases [BQ15, HP13, Pet09].

The increased partitioning effort is a drawback of using equivalence class partition testing compared to random testing. In many cases, several definitions of the domain partitioning are applicable. This is mainly because the tester assumes that test cases of the same equivalence class have the same system behavior. However, the approach removes redundant test cases but retains the completeness of the tests. Hence, the approach reduces the test effort compared to exhaustive testing [BQ15].

2.2 Genetic Algorithms

Manual generation of test cases by the tester for software application tests is error-prone and inefficient. Especially, if the number of input parameter combinations is considerable. Hence, an automated test approach is needed, designed to search for test cases specifically tailored to provoke erroneous behavior. In this case, the system violates given requirements. A promising approach is based on genetic algorithms. They transform the test case design into an optimization problem and automatically search for parameter combinations that satisfy given test criteria. In general, a genetic algorithm is a search-based method to solve complex optimization problems. The approach uses a cost function that evaluates automatically generated optimization parameters with respect to predefined test criteria. It is an effective method which rapidly delivers high-quality solutions to a problem [JT99, SPA16].

Genetic algorithms are inspired by the concept of biological evolution. The solutions to a problem experience evolutionary mechanisms like selection, mutation, and recombination. In terms of genetic algorithms, a solution to a problem is called an individual. It consists of a specified number of genes. The algorithm assigns each individual a value as a measure of their quality with respect to specified criteria by means of a cost function, called fitness function. The goal of applying the genetic algorithm is to create new individuals, so-called children, from previously created individuals, so-called parents, in each generation. This is done until a certain criterion is satisfied by a generation [SPA16, Moh05].

Figure 1 depicts the work-flow of a genetic algorithm. It shows that the algorithm first creates a start population from its search space. The population size indicates the number of individuals per generation. In the next step, the fitness value of each individual is computed and evaluated. This value is decisive for the survival probability of an individual in the selection process. The selection operator selects individuals with high fitness values in the current population to generate a new population from the old one. This corresponds to the principle: survival of the fittest. After that, the genetic algorithm applies the crossover operator to the new population. In this step, the operator swaps genes at random positions between two individuals at a predefined crossover probability p_c. The goal is to generate a better

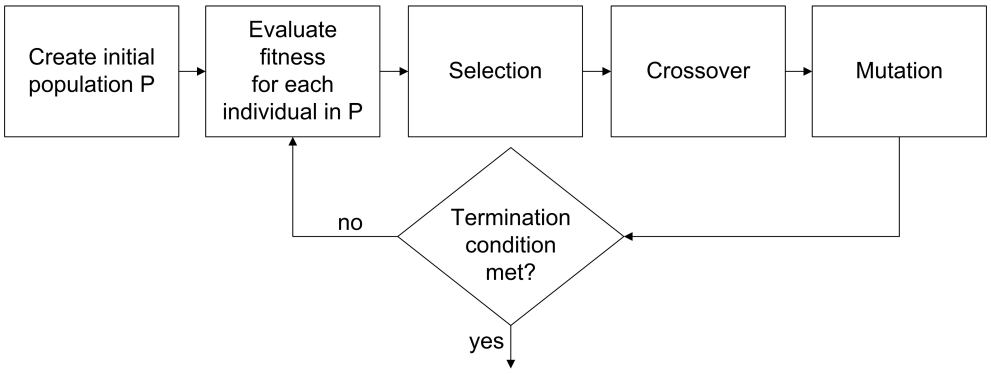

Figure 1: Genetic algorithm procedure

population by combining genes from fitter individuals. As seen in the picture, the next step of the genetic algorithm is the mutation process. The algorithm applies the mutation operator to the population to preserve the diversity of gene values. With a predefined mutation probability p_m, the process alters randomly selected gene values of individuals. As Figure 1 depicts, the genetic algorithm repeats the last four steps on the newly generated population if the current generation does not satisfy the predefined termination condition. This condition may define a maximum number of generations or a reasonably accurate solution [SPA16, Moh05, GKK04, pp. 36-41].

The selection strategy affects the convergence of the genetic algorithm. Too high convergence is a common problem. In that case, the algorithm delivers a locally optimal solution. On the other hand, solutions do not evolve if the convergence is too low. An advantage of genetic algorithms is the possibility to run on parallel processors. They can solve different complex, computation intensive problems, with many possible solutions in a wide search-space. Hence, it is possible to automatically search for optimal test data that provoke a specified behavior of the software application [JT99, SPA16, Moh05, GKK04, pp. 61-62].

2.3 Context: PLAnetary Transits and Oscillation of stars

PLATO is an European Space Agency (ESA) mission in the „cosmic vision“[1] program. The German Aerospace Center (DLR) manages the international consortium for developing the payload and scientific operation of the PLATO project [DLR17].

The main goal of the PLATO mission is the detection and characterization of Earth-like exoplanets orbiting in the habitable zone of solar-type stars. It achieves its scientific objectives by long uninterrupted ultra-high precision photometric monitoring of large samples of bright stars. This requires a very large Field of

[1]Cosmic vision is an ESA long-term space scientific program [ESA12].

View (FOV) as well as a low noise level. To achieve a high pupil size and the required FOV the instrument contains 26 telescopes for star observation. 24 normal cameras monitor stars fainter than magnitude 8 at a cycle of 25 s. Two fast cameras observe stars brighter than magnitude 8 at a cycle of 2.5 s. The size of the FOV of a fast camera is 38.7°×38.7°.

The cameras are equipped with four Charge Coupled Devices (CCDs) in the focal plane, each with 4510×4510 pixels. Figure 2 shows the CCD composition for the fast cameras. The blue areas represent the image area. The other half of the CCDs is the frame transfer storage area.

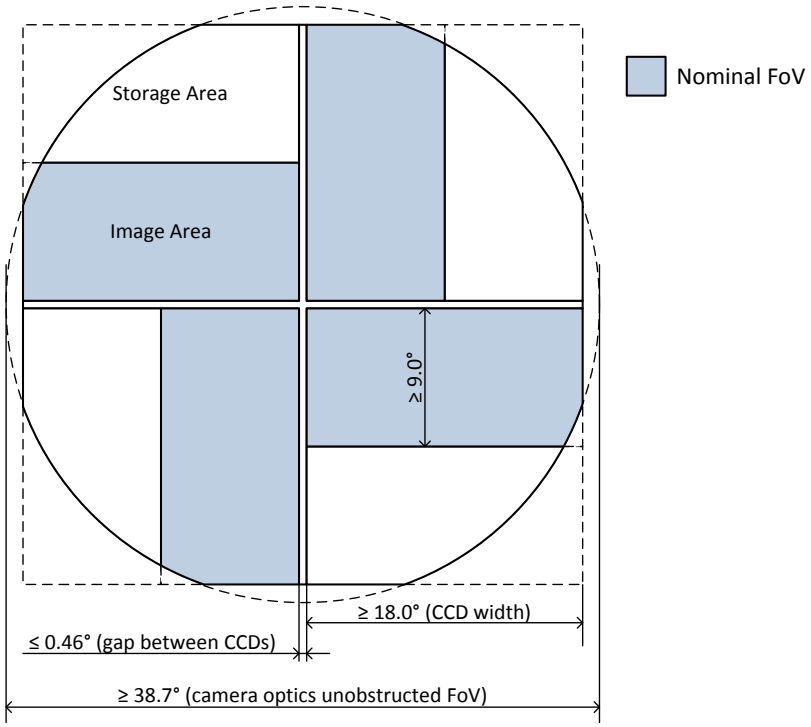

Figure 2: Camera Field of View [Gri17]

Each fast camera comes with a Fast camera Data Processing Unit (F-DPU). On this unit, the FGS algorithm is running. It calculates attitude data with an accuracy of milliarcseconds from the image data of the fast cameras. The F-DPU supplies the attitude data to the Attitude and Orbit Control System (AOCS). The FGS in connection with the AOCS is regarded as mission-critical component. Hence, it requires high reliablity [WP17].

Fine Guidance System Algorithm

Many spacecraft missions use a FGS to obtain accurate knowledge about the spacecraft orientation. We use the PLATO FGS algorithm as a case study to investigate the efficiency of the proposed test approaches. This section gives an overview of the PLATO FGS operation.

The attitude calculation of a telescope is based on estimated star centroids on the CCDs compared to their reference directions in a star catalog. Figure 3 presents an overview of the FGS algorithm [Gri17].

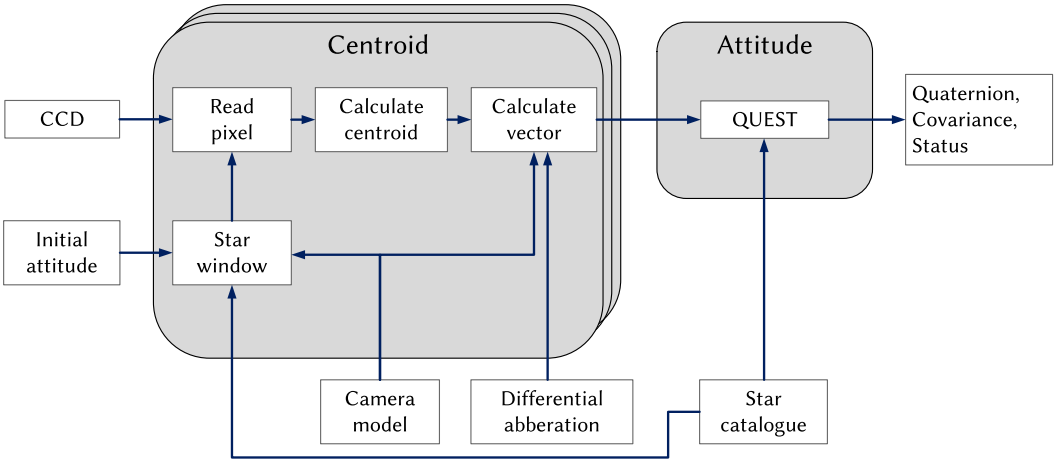

Figure 3: Overview of the Fine Guidance System algorithm [Gri17]

As Figure 3 shows, the FGS gets an initial attitude with an accuracy of 2 arcminutes as input. It uses the value for the initialization of the autonomous attitude tracking. Every 2.5 s, the FGS reads 7×7 pixel sub-windows from a full CCD-image. It determines the sub-windows by means of the initial attitude, preselected guide stars and the camera model. The latter, also called pinhole model, includes equations to transform the centroid coordinates $(u_c, v_c)^T$ on the CCD into unit vectors $(x, y, z)^T$ in the camera coordinate system [Lie02, Gri17].

Each sub-window contains one guide star. Guide stars are predefined stars in a star catalog that satisfy given criteria. For example, the magnitude of the star is within a certain range, the star has very low contamination, etc. [Gri17].

The FGS algorithm calculates centroids after reading sub-images, as Figure 3 depicts. A linear center of mass calculation estimates the centroid position. To get a more precise solution, the algorithm separately estimates each centroid using a Gaussian-Point Spread Function (PSF) observation model. The PSF describes the distribution of the star light over the CCD pixels [SW15]. In the case of a Gaussian-

PSF there is a Gaussian distribution. Otherwise, we call it a non-Gaussian-PSF. Equation 2 presents a Gaussian-PSF observation $h(i, j)$ of a single pixel [Gri17].

$$h(i,j) = I_m \cdot \frac{1}{2\pi\sigma^2} \int_i^{i+1} e^{-\frac{(u-u_c)^2}{2\sigma^2}} \, \mathrm{d}u \int_j^{j+1} e^{-\frac{(v-v_c)^2}{2\sigma^2}} \, \mathrm{d}v + D + \xi \tag{2}$$

The FGS algorithm uses the pixel observation to determine the centroid position $(u_c, v_c)^T$, intensity I_m, image background D and PSF width σ. A non-linear least square fitting method iteratively refines the parameters of the PSF model. The PLATO FGS algorithm calculates the correction by means of the QR-decomposition [Gri17].

In the next step, shown in Figure 3, the FGS algorithm transforms the pixel coordinates of the calculated centroid position into a star direction vector in the camera boresight[2] reference frame. Figure 3 shows the differential aberration as an input of the vector calculation. The aberration is the difference between an apparent position change of a star to an average aberration due to movement of the telescope [MC14]. The aberration value is an optional input and corrects the transformed star direction. We do not consider the value in the following considerations.

In the last step, the FGS algorithm calculates an attitude quaternion from at least two star directions in the boresight reference frame and the corresponding reference vectors from a star catalog by means of the QUaternion ESTimator (QUEST) algorithm. In addition, it delivers an attitude covariance matrix. Within the QUEST algorithm, a TASTE-test measures the validity of the data. The TASTE-test is a simple scalar test that validates the input data for the QUEST-algorithm. The fast validation algorithm uses a variable, called TASTE, which gives the name to the test. If an input star is misidentified, then the TASTE value is high. Therefore, we use the value as a qualitative measure of the accuracy of the FGS algorithm. In the following sections, we denote it as a quality index[Gri17, Shu08].

[2]The boresight is a line passing the geometric center of the focal plane and the pinhole of the camera system [Lie02].

3 Related Work

Equivalence class partition testing „is probably the most widely described, and one of the most widely practiced, software testing techniques“ [Kan04]. This section presents a selection of published work on equivalence class partition testing. Moreover, it represents some previously published work on genetic algorithms used for software testing. The first genetic algorithm was developed by John Holland at the University of Michigan in 1975 [JT99].

Equivalence Class Partition Testing

Various studies investigated equivalence class partition testing strategies for different domains, for example, railway, automotive, avionics, etc. [HP16].

In the automotive domain, DaimlerChrysler Research developed a test approach, called Time Partition Testing (TPT), to test the continuous behavior of control systems. Systems with continuous behavior operate on steadily changing input and output signals. Various projects for production-vehicle development used this approach. Bringmann and Krämer [BK06] explained the principle of the TPT approach using an exterior headlight controller as an example. In most cases, automotive embedded control systems based on complex functional behavior and a large input domain. To increase the test efficiency the TPT approach systematically selects test cases revealing redundant or missing test scenarios in a test set. Using a graphical, formal state machine notation, the TPT approach partitions a test scenario into stream-processing components. Each component defines the behavior of output variables depending on the behavior of input variables up to a certain point in time, specified by a temporal predicate. Test cases define variations in the state machine to test different functional aspects of the system under test.

The study shows that state machines are suitable to partition the temporal behavior of input and output variables in order to model, compare and select test cases. The modeled test cases test the complex functional requirements of control systems. A huge input domain and complex functional behavior are also characteristics of the system class we investigate in this thesis. However, the behavior of systems from this class is not dependent on the arrival time of the input values. Hence, the TPT approach is not applicable to the system class that we consider.

Huang and Peleska [HP16] developed a model-based black-box equivalence class partition testing strategy used in the railway domain. The approach automatically generates finite and complete test suites for safety-critical reactive systems in relation to fault models. Huang and Peleska investigated the approach using the Ceiling Speed Monitor of the European Train Control System as an example for systems with potentially infinite input domain but finite output domain and internal variables. Their approach models the reactive behavior of such systems by

means of deterministic state transition systems. Moreover, it partitions the state space into a finite number of equivalence classes such that all states in a class provide the same output traces for the same non-empty input trace.

The study shows that the proposed approach partitions a large input domain of a complex system into finite equivalence classes. Based on these classes, the approach generates a complete test suite. In the study, it is considered as complete, if the following conditions hold: First, at least one test in this suite fails when testing an application that violates a given specification. Second, each test in the suite passes for all applications that satisfy the specification. Huang and Peleska investigated models whose behavior can be represented by state transition systems. However, we have no state transition system description of the considered satellite on-board image processing application. Hence, we develop an approach that does not need such a description.

Genetic Algorithms

Various papers present genetic algorithms employed as automated software testing method. The algorithms are used to automatically generate test data for example

- for structural-oriented tests, like control flow testing, data flow testing [SK09] [VM14] [SBW01]
- fault-based testing [JES98]
- function-oriented tests, for example examining the temporal behavior of an application [WM01] [AM99] [SBW01] or detecting errors in the software under test [MA00].

Mantere and Alander [MA00] used a genetic algorithm to automatically generate test images that detect errors in an image processing software. These test images improve a commonly used, limited set of test cases. Mantere and Alander investigated various fitness functions considering the difference between the input test image and the processed output image of the application under test.

The study shows that genetic algorithms are useful to automatically generate test data that detect errors in image processing software. In contrast to the application presented in the study, the class of on-board image processing algorithms considered in this thesis does not manipulate the input data. Therefore, the difference between input and output image is not available for the fitness function.

Varshney and Mehrotra [VM14] used genetic algorithms for structural testing. Their algorithm uses the data flow dependencies of a program to automatically optimize test data. The data flow analysis focuses on the interaction of variable definitions and references in a program using a control flow graph. The genetic algorithm automatically generates test data that satisfies specified path criteria.

Varshney and Mehrotra applied the test approach to sample programs, for example, the triangle classification problem.

The study shows that genetic algorithms are feasible to generate test data that achieve high coverage of variable definition and reference paths in the program code. Moreover, the study compares the proposed genetic algorithm with random testing. It shows that the data generated by the genetic algorithm achieves higher coverage of the program flow graph in fewer generations than data generated by a random testing approach. Therefore, the genetic approach is more effective. However, in contrast to Varshney and Mehrotra, we develop an approach that does not depend on the internal structure of the system.

In the first part of their work, Sthamer, Baresel, and Wegener [SBW01] present like Varshney and Mehrotra the use of an evolutionary algorithm that automatically generates test cases for structural testing. In the second part, Sthamer, Baresel, and Wegener used an evolutionary approach to investigate the temporal behavior of embedded systems. The evolutionary approach transforms the test case generation in an optimization problem. It automatically searches for input situations such that the system under test violates specified timing constraints. Hence, the execution time determines the fitness value of individuals. Sthamer, Baresel, and Wegener used an engine control system as a case study. Such systems depend on a large number of input parameters and specified timing constraints. The experiments illustrate that the evolutionary approach generates test data that detect errors in the timing behavior of systems. Thus, the effectiveness and efficiency of the test process increase and development costs decrease. The study shows that the evolutionary approach is applicable to different test goals as well as for testing systems of various application fields. However, we investigate, besides the temporal behavior of image processing applications, the accuracy of the application for various input values. Therefore, we define a fitness function that considers additional metrics to evaluate the individuals.

All of these proposed approaches show that both methods, equivalence class partition testing and genetic algorithms, improve the software test efficiency. The methods are applicable to test various systems with large input space in different domains. According to the results, equivalence class partition testing seems to be capable to efficiently reduce an existing set of test cases by detecting redundant test cases. In addition, the method generates a complete set of test cases by adding missing relevant test cases to the set. The studies confirm that genetic algorithms are applicable to automatically generate test cases that satisfy special test criteria. However, the partitioning of the input domain for equivalence class partition testing and the definition of fitness functions for the utilization of genetic algorithms must be adapted to the specific problem.

4 Multidimensional Coverage Criteria for Automated Testing of Image Processing Algorithms

Many image processing applications require various input parameters such as the position of an object in the image, the magnitude of an object, the position of an object in a pixel, a pattern to distinguish different objects, an observation model, etc. This leads to a huge input domain, which makes testing expensive. Testing such applications manually is error-prone and time-consuming. Therefore, automated test systems are needed. However, automated test generation for satellite on-board image processing applications poses a challenge: The number of possible input parameter combinations in the test image are very large. That means an enormous amount of test cases is possible. This makes it hard to efficiently capture the huge input domain and makes sure that standard inputs, as well as corner cases, are methodically covered.

To overcome that problem, we define a partitioning approach that systematically selects test cases from the huge input domain of satellite on-board image processing applications and enhances a given test suite. To evaluate the efficiency of our proposed test approach, we investigate a case study, namely the PLATO FGS algorithm (Section 2.3). Satellite on-board image processing algorithms are subject to strict requirements with respect to their reliability and accuracy in hard real-time. Therefore, such systems require extensive testing.

Figure 4 depicts an overview of our proposed partitioning approach. Our key idea is to define equivalence classes on input parameters typically used by satellite on-board image processing applications. Section 4.3 presents our concepts to partition the individual input parameters into equivalence classes. Furthermore, we define multidimensional coverage criteria, described in Section 4.4.

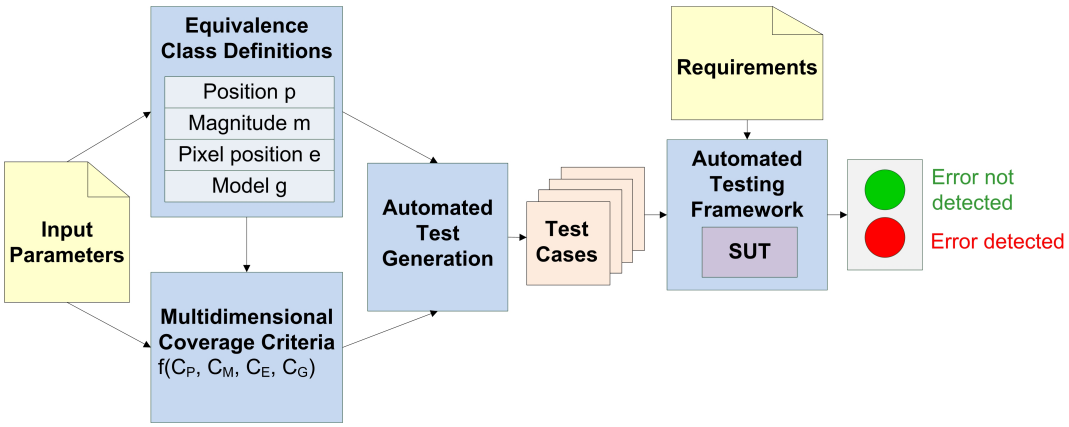

Figure 4: Overview of the partitioning approach

These criteria are based on the equivalence class definitions and measure the number of covered input combinations of a given test suite. As shown in Figure 4, our equivalence classes, as well as coverage criteria, are inputs of our implemented test generation algorithm (see Section 4.5). The algorithm selects for each equivalence class combination a test case from a given test suite as representatives. Our idea is to reduce the number of redundant test cases. Furthermore, our algorithm generates new test cases for missing combinations to reach complete coverage of the input domain. As a result, the algorithm generates a reasonably small test suite that covers the whole input domain of on-board image processing application. The selected test cases serve as input for our automated testing framework. Moreover, we insert requirements for the automated evaluation of the results of the on-board image processing application. If the results do not meet the requirements, the test detects an error. The goal of the test is to automatically detect errors in the on-board image processing application code.

The following sections describe the mentioned steps of the partitioning approach in more detail using the PLATO FGS as a case study.

4.1 Assumptions and Limitations

This section states the assumptions and limitations of our proposed partitioning approach and discuss its applicability to the PLATO FGS algorithm.

In the following, we consider systems with objects in a 2D camera image as inputs. In the case study, the observed objects are stars with magnitudes between 5.5 to 7.0, uniformly distributed in the image.

We consider four star parameters that affect the mathematical accuracy of the centroid estimation of the FGS algorithm: the position in the image, the magnitude, the sub-pixel, and the PSF shape. The input of the centroid calculation is a single star. Hence, a test case for the automated test generation is a test star. The test evaluation is based on the centroid position calculated by the centroid algorithm of the FGS algorithm. In the following sections, we denote a test suite as complete if it reaches full coverage on the input domain with respect to our defined multidimensional coverage criteria.

4.2 Running Example: Input Parameters

One of the main contributions of this thesis is our definition of multidimensional coverage criteria on the input domain of satellite on-board image processing applications. These criteria are based on the combination of equivalence classes of the PLATO FGS input star parameters: position on the Focal Plane Assembly (FPA), magnitude, sub-pixel position, and PSF shape. The accuracy of the centroid estimation mainly depends on the combination of these input parameters.

This section describes how the parameters affect the quality of the centroid calculation of the PLATO FGS algorithm.

The star signal is spread over all pixels in the sub-image. Hence each pixel includes information about the star. However, 90% of the energy is within 2×2 pixel around the centroid. Each pixel also contains noise. We call the information usable if the star signal is less interfered by noise and the Signal-to-Noise Ratio (SNR) is high. If the SNR in the pixel is sufficient, a linear independent equation exists for this pixel. The centroid calculation needs at least 5 linear independent equations to estimate the 5 unknown parameters of the pixel observation (cf. Section 2.3 Equation 2).

The distribution of the star signal depends on the star position on the FPA and the position in a pixel. Due to optical aberrations of the telescope, the PSF shape of the star is wider in the FPA corner than close to the FPA center. Assume that the other input parameters contain reasonably good, constant values. Then a small PSF leads to a low number of pixels with a high SNR and a low number of linear independent equations. In case of a wide PSF the SNR is low but many linear independent equations exist. Both cases can be sufficient for an accurate parameter estimation [Gri17].

The SNR in a pixel also depend on the centroid sub-pixel position. Suppose the other parameters have adequate, constant values. If the centroid is positioned in the pixel center, most star flux is accumulated in a few pixels. Then these pixels have a high SNR compared to the neighboring pixel. In contrast, more pixels have a sufficient SNR if the centroid is on the pixel border or corner. In this case, the star information is distributed more evenly over several pixels. The other pixels have a low SNR. Due to movement, the centroid may move to neighbor pixels. This leads to variations in the pixel illumination and the apparent centroid position [Gri17].

The star magnitude affects the measured flux (photoelectrons per second) of the star. The accumulated number of photoelectrons per pixel denotes the illumination of the pixel. Equation 3 shows the relation between the magnitude m and the corresponding flux F_m in e^-/s.

$$F_m = F_0 TQA * 10^{-0.4*m} \tag{3}$$

with magnitude m of a star, reference flux F_0 of star with m = 0, transmission efficiency T of the optical system, quantum efficiency Q of the detector, and effective light-collecting area A. As the equation shows, the star flux is non-linear to the magnitude of the star. Assume that the other input parameters have sufficient good, constant values. Then, a low magnitude corresponds to a high number of photoelectrons. This means the SNR per pixel is high. More information is usable than in case of a high magnitude [Gri17].

In addition, the accuracy of the centroid calculation depends on the PSF shape. In the best case scenario, the shape is a symmetric Gaussian-PSF. Then, the observation model perfectly fits the star. Therefore, the accuracy of the centroid

calculation is high. But in reality, the PSF shape is non-Gaussian. In that case, the observation model is less accurate. Besides, movements lead to stronger variations in the expected centroid positions [Gri17].

While individual parameter values provide a good centroid estimation, a combination of these parameters leads to changes in the quality of the results. For illustration, Figure 5 shows some example stars with non-Gaussian-PSF shape that are less suitable as guide stars. They all lead to inaccurate estimation results.

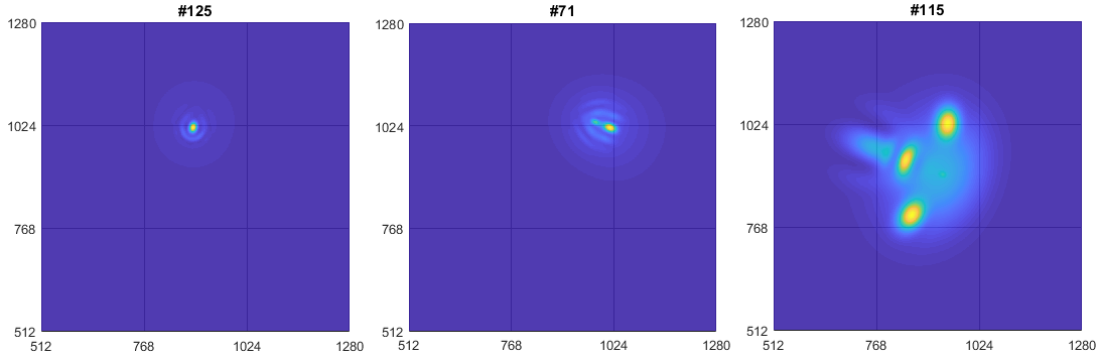

Figure 5: Examples of different low quality stars [Gri17]

What the three stars have in common is that their intensity is concentrated on the pixel edge. For all stars, the magnitude and the FPA position are sufficient good. However, a small variation due to movement may lead to a big change of the illumination of the pixels. Because the Gaussian-PSF observation model does not fit the star PSF shape, the centroid estimation is less accurate and the FGS assumes a big movement of the star.

This section demonstrated the affection of the four most important input parameters on the accuracy of the centroid calculation: the star position on the FPA, the star magnitude, the star position in a pixel and the star PSF shape.

4.3 Equivalence Class Definitions

The quality of centroid calculation of the PLATO FGS algorithms depends on various parameters as described in Section 4.2. For this reason, we define the input domain as a set of input parameters I. The set includes the star position on the FPA $\mathcal{P}$, the star magnitude $\mathcal{M}$, the star position in a pixel $\mathcal{E}$ and the star PSF shape $\mathcal{G}$. This section presents our concepts for partitioning the input parameters $\mathcal{P}$, $\mathcal{M}$, $\mathcal{E}$ and $\mathcal{G}$ into equivalence classes.

Star position on the FPA

Section 4.2 clarifies that the size and shape of the PSF depend on the star position on the FPA. Hence, our idea is to partition the FPA into equally sized, circular areas, as shown in Figure 6. The tester specifies the initial radius r_0.

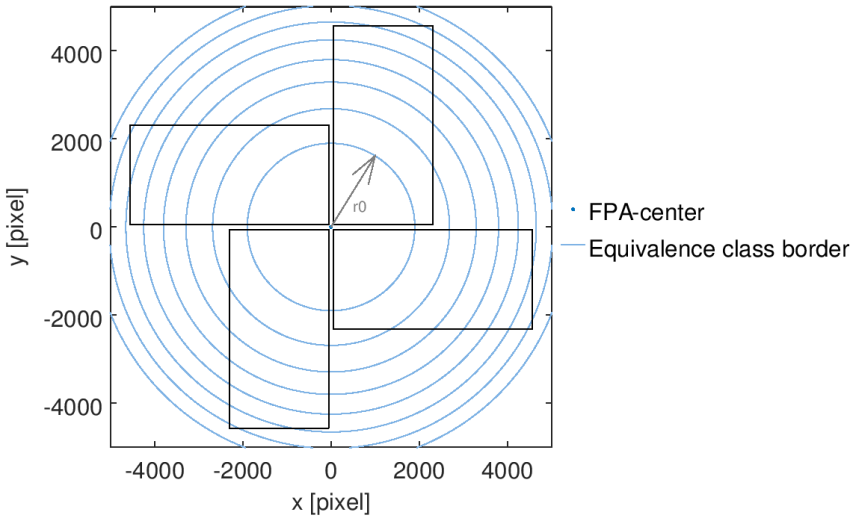

Figure 6: Example partitioning of the Focal Plane Assembly

We partition parameter $\mathcal{P}$ into equivalence classes P_{r_n}. Each class P_{r_n} corresponds to a circular FPA area with inner radius r_{n-1} and outer radius r_n.

$$\mathcal{P} = P_{r_1} \cup P_{r_2} \cup ... \cup P_{r_n} \cup ... \cup P_{r_{I_\mathcal{P}}} \text{ with } 1 \leq n \leq I_\mathcal{P} \tag{4}$$

Let S denote the set of available stars. A star $s \in S$ lies in an equivalence class P_{r_n} if following condition holds.

$$r_{n-1} \leq p(s) < r_n, \text{ with } p(s) = \sqrt{{x_s}^2 + {y_s}^2} \tag{5}$$

where (x_s, y_s) is the position of star s on the FPA and $p(s)$ is the distance of the star s to the FPA center.

Star magnitude

A useful partitioning of magnitude values into equivalence classes is not obvious. Our idea is to partition the flux range into $I_\mathcal{M} \in \mathbb{N}$ equidistant parts that represent the equivalence classes. We define Equation 6 to obtain the upper limit of a sub-range.

$$F_{m_j} = F_{7.0} + j\frac{F_{5.5} - F_{7.0}}{I_\mathcal{M}} \tag{6}$$

F_{m_j} is the flux of magnitude m_j and $j = 1...I_{\mathcal{M}}$ represents the j-th equivalence class of parameter $\mathcal{M}$. $F_{5.5}$ and $F_{7.0}$ are the numbers of photons for magnitude 5.5 and 7.0. We calculate the flux values $F_{5.5}$ and $F_{7.0}$ by using Equation 3 from Section 4.2. Furthermore, we use Equation 7 to recalculate the magnitude m_j from the calculated flux limit F_{m_j} of the flux sub-range.

$$m = -2.5 * log\left(\frac{F_m}{F_0 TQA}\right) \tag{7}$$

From a formal point of view, we partition the parameter $\mathcal{M}$ into equivalence classes M_l.

$$\mathcal{M} = M_{7.0} \cup ... \cup M_{l_j} \cup ... \cup M_{5.5}, \text{with } l_j \in \mathbb{R} \text{ and } 5.5 \leq l_j \leq 7.0. \tag{8}$$

Each equivalence class M_{l_j} is a magnitude sub-range with upper limit l_j. Each available star s lies in an equivalence M_{l_j} if it satisfies the condition in Equation 9.

$$l_{j-1} \leq m(s) < l_j \tag{9}$$

$m(s)$ denotes the observed magnitude of star s and l_j with $j = 1...I_{\mathcal{M}}$ is the upper limit of the j-th magnitude sub-range. The tester specifies the number of equivalence classes $I_{\mathcal{M}} \in \mathbb{N}$ of the parameter $\mathcal{M}$. Figure 7 illustrates an example partitioning of the magnitude range.

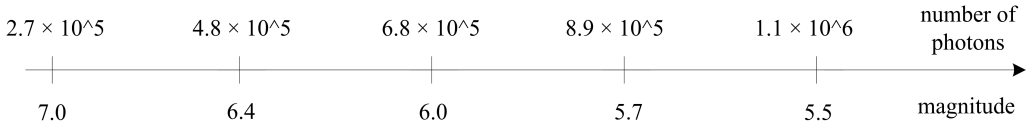

Figure 7: Example partitioning of magnitude range

Star sub-pixel

A third parameter elaborated in Section 4.2, is the sub-pixel position of a star center. The quality of the centroid estimation of stars close to a pixel border is as sensitive to movements as the estimation of stars with centroids on a pixel corner and vice versa. For this reason, we divide the pixel area into different sub-areas as shown in Figure 8.

The tester specifies the ratio r of the central area of the pixel to the pixel size, for example, 1/2, 3/5, etc. If a is the pixel size, then the length of the edge of the central area results from Equation 10.

$$b = \sqrt{a^2 r} \tag{10}$$

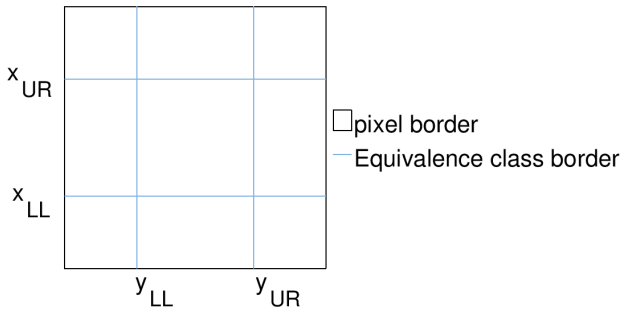

Figure 8: Example partition of a pixel

With this information, the we obtain the lower left corner l and the upper right corner u of the central pixel area, with

$$l = (\frac{a}{2} - \frac{b}{2}, \frac{a}{2} - \frac{b}{2}) \text{ and } u = (\frac{a}{2} + \frac{b}{2}, \frac{a}{2} + \frac{b}{2}) \tag{11}$$

Based on these corners, we partition parameter $\mathcal{E}$ into equivalence classes E_i with $i = 0...8$. The equivalence class E_i is the i-th pixel sub-area.

$$\mathcal{E} = E_0 \cup E_1 \cup ... \cup E_8 \tag{12}$$

A star s lies in an equivalence class if it satisfies the corresponding condition.

$$\begin{aligned}
E_0: &\quad 0 \leq e_x(s) < x_l \ \wedge\ 0 \leq e_y(s) < y_l \\
E_1: &\quad 0 \leq e_x(s) < x_l \ \wedge\ y_l \leq e_y(s) < y_u \\
E_2: &\quad 0 \leq e_x(s) < x_l \ \wedge\ y_u \leq e_y(s) < a \\
E_3: &\quad x_l \leq e_x(s) < x_u \ \wedge\ 0 \leq e_y(s) < y_l \\
E_4: &\quad x_l \leq e_x(s) < x_u \ \wedge\ y_l \leq e_y(s) < y_u \\
E_5: &\quad x_l \leq e_x(s) < x_u \ \wedge\ y_u \leq e_y(s) < a \\
E_6: &\quad x_u \leq e_x(s) < a \ \wedge\ 0 \leq e_y(s) < y_l \\
E_7: &\quad x_u \leq e_x(s) < a \ \wedge\ y_l \leq e_y(s) < y_u \\
E_8: &\quad x_u \leq e_x(s) < a \ \wedge\ y_u \leq e_y(s) < a
\end{aligned} \tag{13}$$

where $e_x(s)$ and $e_y(s)$ return the x-coordinate and y-coordinate of s in the pixel respectively.

Star PSF shape

We partition the parameter $\mathcal{G}$ in two equivalence classes G_G and G_{NG}. A star is in class G_G if is has a Gaussian-PSF. Otherwise the star covers class G_{NG}.

Based on these definitions, we define multidimensional coverage criteria to evaluate a given test suite with respect to its coverage of the full input domain.

4.4 Multidimensional Coverage Criteria

We define multidimensional coverage criteria on the input domain of satellite onboard image processing applications to measure the coverage of a test suite with respect to input parameter combinations. If the measured coverage of a test suite is not complete, our automated test generation algorithm automatically inserts test cases for missing combinations. This section presents our definitions of multidimensional coverage criteria on the input domain $I = \{\mathcal{P}, \mathcal{M}, \mathcal{E}, \mathcal{G}\}$.

The individual coverage of an input parameter denotes the ratio of equivalence classes that are covered by at least one test case from a given test suite to the number of equivalence classes of this input parameter. Equation 14 to 17 express the definition for the input parameters $\mathcal{P}, \mathcal{M}$, $\mathcal{E}$ and $\mathcal{G}$.

$$C_{\mathcal{P}} = \frac{\#\ covered\ equivalence\ classes\ of\ \mathcal{P}}{|\mathcal{P}|} \tag{14}$$

$$C_{\mathcal{M}} = \frac{\#\ covered\ equivalence\ classes\ of\ \mathcal{M}}{|\mathcal{M}|} \tag{15}$$

$$C_{\mathcal{E}} = \frac{\#\ covered\ equivalence\ classes\ of\ \mathcal{E}}{|\mathcal{E}|} \tag{16}$$

$$C_{\mathcal{G}} = \frac{\#\ covered\ equivalence\ classes\ of\ \mathcal{G}}{|\mathcal{G}|} \tag{17}$$

The coverage domain for our multidimensional coverage criteria is the Cartesian product of equivalence classes of the input parameters $\mathcal{P}$, $\mathcal{M}$, $\mathcal{E}$ and $\mathcal{G}$. Therefore, an input combination is a tuple of equivalence classes (P_i, M_j, E_k, G_l), where $P_i \in \mathcal{P},\ M_j \in \mathcal{M},\ E_k \in \mathcal{E}$ and $G_l \in \mathcal{G}$. Furthermore, a test case is a star represented by a tuple of parameter values $(p, m, e, g) \in (P_i, M_j, E_k, G_l)$. The following example test cases clarify these definitions.

Example 1: $(1030.4, 6.5, (0.45, 0.13), G) \in (P_{2000} \times M_{6.56} \times E_3 \times G_G)$:
The test case comprises a star whose position is in the FPA area with radius 2000. The star belongs to equivalence class $M_{6.56}$, because its magnitude value is between 6.25 and 6.56. The star center is located in the middle-left pixel sub-area. That corresponds to equivalence class E_3. The star is part of equivalence class G_G, because it has a Gaussian-PSF shape.

Example 2: $(579.1, 6.5, (0.95, 0.2), G) \in (P_{2000} \times M_{6.56} \times E_6 \times G_G)$:
The test case comprises a similar or the same star as in the first example. But on the contrary, it belongs to equivalence class E_6 because the center of the star is positioned nearby the upper left pixel border.

Example 3: $(187.7, 5.9, (0.93, 0.07), NG) \in (P_{2000} \times M_{6.0} \times E_6 \times G_{NG})$:
The test case comprises a star from equivalence classes P_{2000}, $M_{6.0}$ and E_6. Because of its non-Gaussian-PSF, it is part of the equivalence class G_{NG}.

The multidimensional coverage criterion is completely satisfied if the test cases in a test suite cover all possible input combinations at least once. The number of required covered input combinations for a complete coverage is $|\mathcal{P} \times \mathcal{M} \times \mathcal{E} \times \mathcal{G}|$. In the remaining sections, we denote a test suite with complete coverage on the input domain with respect to our multidimensional coverage criteria as a complete test suite. The achieved coverage C results from the ratio of the number of input combinations that are covered by at least one test case to the total number of input combinations.

$$C = \frac{\#\ covered\ input\ combinations}{|\mathcal{P} \times \mathcal{M} \times \mathcal{E} \times \mathcal{G}|} \tag{18}$$

Applying Algorithm 1, the partitioning approach computes the individual and multidimensional coverage of a given test suite. The input parameters $\mathcal{P}$, $\mathcal{M}$, $\mathcal{E}$, and $\mathcal{G}$ contain $I_{\mathcal{P}}$, $I_{\mathcal{M}}$, $I_{\mathcal{E}}$, $I_{\mathcal{G}}$ equivalence classes respectively.

Input: Test suite TS
Output: Multidimensional coverage Cov of TS

$C_{\mathcal{P}} = C_{\mathcal{M}} = C_{\mathcal{E}} = C_{\mathcal{G}} = C = \emptyset$;
foreach *tc with (p, m, e, g)* $\in TS$ **do**
    i_P = getPosECId(p);
    $C_{\mathcal{P}} \leftarrow C_{\mathcal{P}} \cup i_P$;
    i_M = getMagECId(m);
    $C_{\mathcal{M}} \leftarrow C_{\mathcal{M}} \cup i_M$;
    i_E = getPixECId(e);
    $C_{\mathcal{E}} \leftarrow C_{\mathcal{E}} \cup i_E$;
    i_G = getModECId(g);
    $C_{\mathcal{G}} \leftarrow C_{\mathcal{G}} \cup i_G$;
    $C \leftarrow C \cup (i_P, i_M, i_E, i_G)$;
end
$Cov_{\mathcal{G}} = |C_{\mathcal{P}}|/I_{\mathcal{P}}$;
$Cov_{\mathcal{M}} = |C_{\mathcal{M}}|/I_{\mathcal{M}}$;
$Cov_{\mathcal{E}} = |C_{\mathcal{E}}|/I_{\mathcal{E}}$;
$Cov_{\mathcal{G}} = |C_{\mathcal{G}}|/I_{\mathcal{G}}$;
$Cov = |C|/(I_{\mathcal{P}} \cdot I_{\mathcal{M}} \cdot I_{\mathcal{E}} \cdot I_{\mathcal{G}})$

Algorithm 1: Calculation of individual and multidimensional coverage

For each test case in the given test suite, the algorithm gets for each input parameter the index i_P, i_M, i_E, i_G of the corresponding equivalence class from $\mathcal{P}$, $\mathcal{M}$, $\mathcal{E}$ and $\mathcal{G}$. The algorithm adds the indexes to the sets $C_\mathcal{P}$, $C_\mathcal{M}$, $C_\mathcal{E}$ and $C_\mathcal{G}$ respectively. Moreover, it inserts the tuple (i_P, i_M, i_E, i_G) into the set C that contains all covered input combinations. Because the algorithm uses the union operator to add the tuples to the set, each tuple is included in the set only once. The algorithm applies Equation 14 to 18 to compute the individual and multidimensional coverage.

Input: Value g of $\mathcal{G}$
Output: Index i of equivalence class G_i

```
1 if g == gaussian then
2     return 0
3 else
4     retun 1
5 end
```

Algorithm 2: Get equivalence class index

Algorithm 2 shows the function *getModECId()* as an example for determining the equivalence class index of a parameter value. The function checks which equivalence class the input value g of parameter $\mathcal{G}$ belongs to and returns the corresponding equivalence class index i_G.

The partitioning approach assesses the quality of different test suites with respect to their coverage of the input space of a satellite on-board image processing application, using multidimensional coverage criteria. The next section describes our automated test generation algorithm that systematically selects and evaluates test cases for the test execution. It uses our formulated multidimensional coverage criteria to deliver a complete test suite.

4.5 Automated Test Generation

This section explains the systematic generation and evaluation of a test suite with complete coverage of the input domain according to our proposed multidimensional coverage criteria.

Our test approach is based on the partitioning of the input domain into disjoint sub-domains with respect to our equivalence class definitions of the input parameters of the PLATO FGS algorithm. Our approach applies Algorithm 1 (Section 4.4) to evaluate a given test suite with respect to its coverage of equivalence class combinations. If the multidimensional coverage of the given test suite is not complete, the approach applies Algorithm 3 to generate missing test cases.

Input: Universe of input combinations U, set of input combinations C covered by test suite TS
Output: Test suite TS with complete multidimensional coverage

```
1 Cov = computeMultidimensionalCoverage(TS);
2 if Cov < 1 then
3     W ← U \ C;
4     foreach w ∈ W do
5         tc = generateTC(w);
6         TS ← TS ∪ tc;
7     end
8 end
```

Algorithm 3: Generate complete test suite

The algorithm generates the set W that contains all input combinations that the given test suite does not cover. For each input combination in W, our algorithm generates a test case by randomly selecting values from the equivalence classes of the missing combinations. The algorithm adds the new generated test case to the test suite. In this way, it efficiently inserts missing but relevant test cases into the test suite. This increases the multidimensional coverage and therefore the error-detection capability of the given test suite. As a result, our automated test generation algorithm returns a complete test suite.

If the set of covered input combinations C is empty, then the set of uncovered input combinations W is equal to the universe of possible input combinations U. Hence, Algorithm 3 generates a new test suite that completely satisfies the multidimensional coverage criteria. Our automated testing framework only selects one test case per input combination. This efficiently reduces the number of redundant test cases for the test execution.

We demonstrate in Section 6.2 the efficiency of our test approach according to the error-detection capability of test suites with different multidimensional coverage on the input domain.

4.6 Summary

In the last sections, we have developed an automated test approach that systematically covers the input domain of the PLATO FGS algorithm as a case study.

As input domain we have considered four star parameters that affect the mathematical accuracy of the centroid calculation: position on the FPA, magnitude, sub-pixel position, and PSF shape.

The quality of the centroid calculation mainly depends on the combination of these input parameters. For that reason, we have defined dedicated equivalence classes for these input parameters of the FGS algorithm. The partitioning is based on specifications of the tester. This turns it into a flexible approach that can be easily adjusted to various applications.

The random selection of a test case from a given test suite per equivalence class combination leads to a minimum number of relevant test cases. In addition, we have defined multidimensional coverage criteria based on the equivalence classes to measure the number of input combinations covered by a given test suite. If a test suite covers all combinations of the equivalence classes, the test suite is complete with respect to our defined multidimensional coverage criteria.

We have defined an automated test generation algorithm that uses our multidimensional coverage criteria to automatically insert test cases for missing equivalence class combinations. This implies that our algorithm systematically generates a complete test suite with a minimum of test cases for the given input domain.

In the following sections, we use the generated test suite as the search space of our proposed genetic algorithm.

5 Genetic Algorithm for Automated Test Generation for Image Processing Algorithms

One major challenge of satellite on-board image processing applications is the observance of strict requirements with respect to their reliability and accuracy in hard real-time. Often, on-board image processing applications perform complex algorithmic computations. Therefore, it is hard to find test cases that are specifically tailored to provoke real-time critical behavior or scenarios where the accuracy gets critically low.

We define a test approach that automatically searches for test cases that increase the robustness of a given system. Our solution is based on genetic algorithms. We investigate the efficiency of our proposed genetic approach with the PLATO FGS algorithm as a case study. However, the input of the FGS algorithm is a combination of 30 stars. For a realistic test scenario, we only use stars with non-Gaussian-PSF shape. Using the complete test suite generated by Algorithm 3 from Section 4.5 without stars with Gaussian-PSF shape leads to

$$\binom{I_{\mathcal{P}} \cdot I_{\mathcal{M}} \cdot I_{\mathcal{E}}}{30} \tag{19}$$

possible input star combinations, where $I_{\mathcal{P}}, I_{\mathcal{M}}$, and $I_{\mathcal{E}}$ are the number of equivalence classes of the input parameters $\mathcal{P}, \mathcal{M}$, and $\mathcal{E}$ described in Section 4.3. The application of the equivalence class specification from Section 6.2 results in 1.6×10^{46} possible combinations. Due to this huge amount of possible star combinations, a manual search, as well as the execution of all possible combinations, is not feasible.

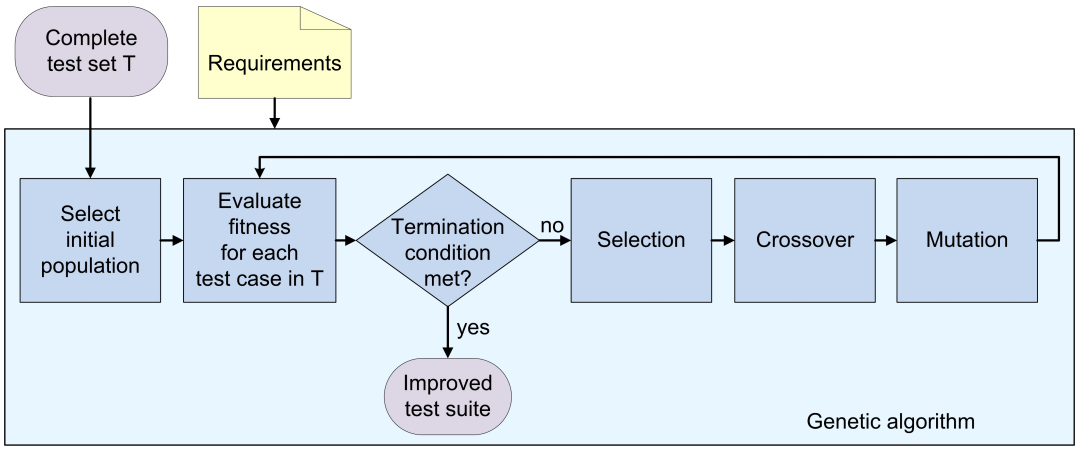

Figure 9: Overview of the automated test case generation approach

Typically, the robustness of satellite on-board image processing applications depends on different parameters, for example, the real-time behavior and the mathematical accuracy of the algorithm. Hence, our objective is to find test cases that provoke long execution times or inaccurate results or both of the satellite on-board image processing application. We present a genetic algorithm that transforms the test case generation into an optimization problem. Therefore, we define a novel two-criteria fitness function that is specifically tailored for the domain of satellite on-board image processing application.

Figure 9 shows the workflow of our proposed approach. As the figure depicts, a test set with complete coverage on the input domain with respect to our multidimensional coverage criteria is an input of our genetic algorithm (see Section 5.2). To make the approach more flexible, the tester inserts a parameter specification for the genetic algorithm. This specification includes, for example, the population size, the mutation rate, termination conditions, etc.

As Figure 9 shows, our proposed genetic algorithm evaluates the test cases with respect to their fitness values. It iteratively evolves promising test cases using evolutionary mechanisms, namely selection, crossover, and mutation. As a result, it delivers test cases that satisfy the given test criteria. Our idea is that the genetic algorithm automatically searches for test cases that support robustness testing by provoking mission-critical behavior with respect to run time and mathematical accuracy. The following sections describe our genetic approach using the PLATO FGS as a case study.

5.1 Assumptions and Limitations

The same assumptions as described in Section 4.1 apply for our proposed automated genetic approach. Except, that the genetic approach examines all parts of the PLATO FGS algorithm.

Performance and mathematical accuracy of the FGS algorithm depend among other things on the number and distribution of preselected guide stars. Currently, the algorithm gets 30 guide stars as input. Therefore, in the following sections, a test case is a combination of 30 different stars. A simulated time series exists for each star. It comprises 1000 exposures with movements of the camera.

For realistic tests, the complete test set only contains stars with non-Gaussian-PSF shape. We denote a test set as complete if it covers the input domain with respect to our multidimensional coverage criteria. This means the set includes one star for each combination of equivalence classes of the input parameters except the PSF shape.

For the evaluation of our test approach, we use the TASTE-value as a qualitative measure of the mathematical accuracy of the FGS algorithm. Hence, a low quality index corresponds to a high accuracy of the FGS algorithm.

5.2 Proposed Genetic Algorithm

We define a genetic algorithm to support robustness testing of satellite on-board image processing applications. Our algorithm automatically searches for test cases in a given test set. These test cases are specifically tailored to provoke mission-critical behavior with respect to run time and mathematical accuracy. This section describes the components and strategies of our developed genetic algorithm.

Representation

In terms of our proposed genetic algorithm, a test case represents an individual with 30 genes. A gene is a tuple of equivalence class identifiers (i_P, i_M, i_E, i_G) where $P \in \mathcal{P}$, $M \in \mathcal{M}$, $E \in \mathcal{E}$, and $G \in \mathcal{G}$ are equivalence classes of the respective input parameters of the FGS algorithm (cf. Section 4.3). The gene values are limited to the equivalence classes of the corresponding input parameter. Figure 10 clarifies the definition by representing the example stars from Section 4.4 as individuals.

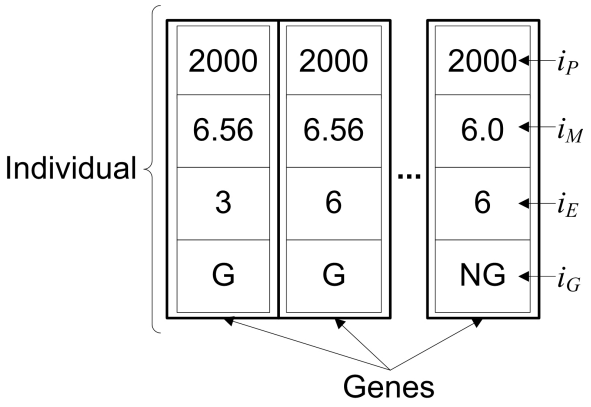

Figure 10: Example individual representation

Initial Population

Our genetic algorithm uses a given complete test set with respect to our coverage criteria (cf. Section 4.4) as its search space. From this space, the algorithm randomly selects 30 stars to form an individual. Each selected star covers a different combination of equivalence classes. Hence an individual does not include a gene twice. The tester specifies the population size. Thus the genetic algorithm generates individuals until the required population size is reached. Algorithm 4 shows the initialization process.

```
Input: Test set TS, population size popsize
Output: Population population with popsize individuals
1 for i ← 1 to popsize do
2     for j ← 1 to 30 do
3         gene = getRandomTestStar(TS);
4         individual ← individual ∪ gene;
5     end
6     population ← population ∪ individual;
7 end
```

Algorithm 4: Select initial population

Fitness Function

In the evaluation process (Algorithm 5), our genetic algorithm sends each individual in the current population as input to the FGS algorithm. The resulting execution time and quality index are crucial for the fitness value of an individual. Hence, the genetic algorithm uses our two-criterion fitness function that depends on both parameters. To capture a trade-off between them, we applied the weighted sum.

$$\begin{aligned} fitness(c) &= f_{time}(c) \cdot w_1 + f_{taste}(c) \cdot w_2, \text{ with} \\ f_{time}(c) &= \frac{t}{a_{time}}, \\ f_{taste}(c) &= \frac{taste}{a_{taste}}, \\ 0 \leq w_i \leq 1,\ i = \{1,2\} &\text{ and } \sum_{i=1}^{2} w_i = 1 \end{aligned} \tag{20}$$

$fitness(c)$ provides the fitness value of the individual c. Individuals with a high fitness value are more effective, also called fitter, than individuals with lower fitness values. $f_{time}(c)$ calculates the fitness value of the individual c with respect to the execution time. It measures the execution time relative to a reference value a_{time} specified by the tester. a_{time} defines, for example, the average of known execution times. $f_{taste}(c)$ calculates the fitness value of the individual c with respect to the quality index delivered by the FGS algorithm. It measures the delivered quality index relative to a reference value a_{taste}, for example, the average of known quality values, defined by the tester. Moreover, the tester specifies the weighting factors w_i to determine which parameter has a stronger impact on the new generation. For example, if the weight of one parameter is set to zero, the genetic algorithm optimizes new generations only for the other parameter.

Input: Population *population*, fitness parameter weights w_{time}, w_{tatse}, reference values a_{time}, a_{taste}
Output: population fitness *populationFit*, longest execution time *fittestTime*, worst quality index *fittestTaste*

```
1  maximalFit = 0;
2  populationFit = 0;
3  foreach individual ∈ currentPopulation do
4      time, taste = FGS(individual);
5      fitValue = time/a_time · w_time + taste/a_taste · w_taste;
6      if fitValue > maximalFit then
7          maximalFit = fitValue;
8          fittestTime = time;
9          fittestTaste = taste;
10     end
11     individual.fit = fitValue;
12     individual.prevFitSum = populationFit;
13     populationFit += fitValue;
14     individual.afterFitSum = populationFit;
15 end
```

Algorithm 5: Fitness evaluation

In addition, Algorithm 5 computes the fitness value of the whole population and the accumulated fitness value of the individual's predecessor. The genetic algorithm uses these fitness values in the selection process.

Selection

After calculating the fitness $fitness(c)$ of each individual in the population, our genetic algorithm computes the selection probability $p(c_i)$ of an individual c_i using Equation 21. F_t denotes the total fitness of the population of generation t.

$$p(c_i) = \frac{fitness(c_i)}{F_t} \quad \text{, with } F_t = \sum_{i=1}^{popsize} fitness(c_i), \\ \sum_{i=1}^{popsize} p(c_i) = 1 \text{ and } 0 \leq p(c_i) \leq 1 \tag{21}$$

The selection mechanism applies the *stochastic universal sampling* method [GKK04, p. 80]. Figure 11 illustrates the stochastic universal sampling with 6 individuals. Table 1 shows the fitness values and selection probabilities of each individual.

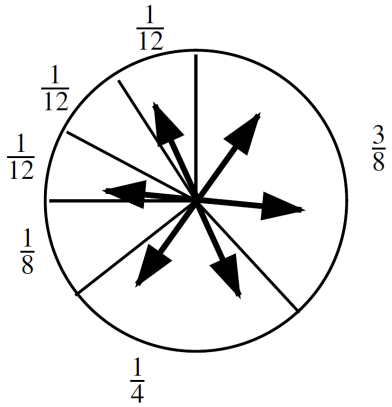

Figure 11: Stochastic Universal Sampling[Wei15]

Individual c	*fitness(c)*	*p(c)*
1	45	$\frac{3}{8}$
2	30	$\frac{1}{4}$
3	15	$\frac{1}{8}$
4	10	$\frac{1}{12}$
5	10	$\frac{1}{12}$
6	10	$\frac{1}{12}$
	120	1

Table 1: Example fitness values and selection probability of individuals

As Figure 11 depicts, each individual gets a section on an imaginary roulette-wheel proportional to its fitness value. The method arranges as many pointers around the roulette wheel as there are individuals in the population. The pointers are at the same distance from each other.

Input: Population *currentPopulation*, population size *popsize*, population fitness *populationFit*
Output: Population *selectedPopulation*

```
1  selectedPopulation ← ∅;
2  pointer = getRandomDoubleNumber(0, 1/popsize);
3  for i ← 1 to popsize do
4      foreach individual ∈ currentPopulation do
5          prevSelectProp = individual.prevFitSum/populationFit;
6          afterSelectProp = individual-afterFitSum/populationFit;
7          if prevSelectProp ≤ pointer < afterSelectProp then
8              selectPopulation ← selectPopulation ∪ individual;
9          end
10     end
11     pointer = pointer + (i-1) · 1/popsize;
12 end
```

Algorithm 6: Stochastic universal sampling selection of individuals

As Algorithm 6 shows, it randomly selects the first pointer. After turning the roulette wheel once, the algorithm inserts each individual into the new population that satisfies Equation 22. This means it selects for each pointer that individual whose field the pointer points to [GKK04, p. 80].

$$\sum_{j=1}^{j<i} p(c_i) \leq pointer < \sum_{j=1}^{j\leq i} p(c_i) \tag{22}$$

pointer represents the imaginary pointer indicating the roulette-wheel section of individual c_i. The genetic algorithm selects each individual c_i at least $\lfloor p(c_i) \cdot popsize \rfloor$ times and at most $\lceil p(c_i) \cdot popsize \rceil$ times into the new generation. The selection method ensures that the algorithm selects the fittest individual in any case [GKK04, p. 80].

Crossover

We adopt the *parameterized uniform crossover* strategy [GKK04, p. 89]. As Algorithm 7 shows, the crossover mechanism randomly chooses two not yet selected individuals as parents in each loop. For every single gene of the parents, the genetic algorithm decides according to the crossover probability p_c whether the gene is exchanged or not. The tester specifies the crossover probability p_c [GKK04, p. 89].

Input: Population *parentPopulation*, population size *popsize*, crossover probability p_c
Output: Population *childPopulation*

```
1  childPopulation ← ∅;
2  for i ← 1 to popsize/2 do
3      parentA = getRandomIndividual(parentPopulation);
4      parentB = getRandomIndividual(parentPpopulation);
5      for j ← 1 to 30 do
6          x = getRandomDoubleNumber(0,1);
7          if x ≤ p_c then
8              childA[j] = parentB[j];
9              childB[j] = parentA[j];
10         else
11             childA[j] = parentA[j];
12             childB[j] = parentB[j];
13         end
14     end
15     childPopulation ← childPopulation ∪ childA ∪ childB;
16 end
```

Algorithm 7: Crossover

Mutation

Algorithm 8 shows the mutation process of the genetic algorithm. It decides for each gene according to a mutation probability p_m whether it mutates or not [GKK04, p. 98]. If the gene mutates, the genetic algorithm randomly selects a new star from its search space, which is not contained in the individual, as a gene. The tester specifies the mutation probability p_m.

```
Input: Population population, mutation probability p_m, test set TS
Output: Population mutatedPopulation
1  mutatedPopulation ← ∅;
2  foreach individual ∈ population do
3      for i ← 1 to 30 do
4          x = getRandomDoubleNumber(0,1);
5          if x ≤ p_m then
6              individual[i] = getRandomTestStar(TS);
7          end
8      end
9      mutatedPopulation ← mutatedPopulation ∪ individual;
10 end
```

Algorithm 8: Mutation

Termination Condition

The genetic algorithm terminates in four cases:

- it reaches a specified number of generations,
- no improvement of the best solution in the last n generations, or
- the FGS algorithm takes a predefined longest execution time

The tester specifies these criteria.

5.3 Automated Test Generation

This section describes our genetic approach that automatically searches for test cases that provoke mission critical behavior of the PLATO FGS algorithm. The objective of our test approach is to find star combinations that provoke long execution times or inaccurate results of the satellite on-board image processing application.

The approach utilizes the partitioning of the input parameters of the FGS algorithm (see Section 4.3). A given test set contains one star for each equivalence class combination of the parameters. This results in approximately 2.8×10^{55} possible

combinations using the equivalence class specification from Section 6.2. Testing all possible combinations is therefore infeasible. Hence, our key idea is to develop a genetic algorithm that is specifically tailored to find test cases in this specific domain.

Input: Test set TS, population size $popsize$, fitness parameter weights w_{time}, w_{tatse}, fitness parameter references a_{time}, a_{taste} crossover probability p_c, mutation probability p_m, maximal number of generations T, maximal execution time $maxTime$, worst quality index $maxTaste$, number of observing generations n, minimal variance var
Output: Individual that provokes longest execution time and worst quality index

```
1  t = 0;
2  P ← ∅;
3  fitTime = 0;
4  fitTaste = 0;
5  variance = 0;
6  P ← createInitialPopulation(TS, popsize);
7  popFit, fitTime, fitTaste ← evaluation(P, w_time, w_taste, a_time, a_taste);
8  while t < T AND (fitTime < maxTime OR fitTaste < maxTaste
     AND variance > var) do
9      P ← selection(P, popsize, popFit);
10     P ← crossover(P, popsize, p_c);
11     P ← mutation(P, p_m, TS);
12     popFit, fitTime, fitTaste ← evaluation(P, w_time, w_taste, a_time,
         a_taste);
13     t++;
14     if n ≤ t then
15         variance ← getFitnessVariance(n);
16     end
17 end
```

Algorithm 9: Proposed genetic algorithm

Algorithm 9 gives an overview of the structure of our defined genetic algorithm using the components described in Section 5.2. The complete test set with respect to our equivalence classes, except stars with Gaussian-PSF shape, is its search space. Therefore, this space is reduced to the equivalence classes of the input parameters. Our genetic algorithm creates the initial population by randomly selecting stars from its search space. Due to our defined two-criterion fitness function, it automat-

ically generates test cases to provoke long execution time of the FGS algorithm or an increased quality index or both. By specifying the weights of the criteria in the fitness function, the tester is flexible to determine the test goal.

The applied selection method *stochastic universal sampling* reduces the evolutionary pressure. This means, it also selects test cases with low fitness values in the new generation. Therefore, the method preserves the variability in the population. Hence, our algorithm scans the search space extensively to find test cases that globally provoke the most critical system behavior [GKK04, pp. 79-83]. The *parameterized uniform crossover* strategy in the crossover function generates new combinations by mixing the selected individuals. The mutation function adds never inserted stars or removed stars into the current population. Depending on the mutation probability, the mutation function preserves the diversity in the population or inserts minimal changes to find test cases that locally provoke critical behavior [GKK04, p. 42].

By applying the crossover or mutation process, stars may appear twice in an individual. To avoid this, we specify that the genes do not swap or mutate if the new gene is already in the individual. Our genetic algorithm iteratively evolves individuals until they satisfy our specified termination conditions. The algorithm terminates if it reaches a predefined number of generations T or the achieved maximum execution time or quality index of a generation exceeds a specified maximum execution time *maxTime* or maximum quality index *maxTaste* respectively. An additional termination criterion is the variance of the maximum fitness values achieved in the last n generations [BMP12]. If the value is lower than a predefined variance *var*, the condition is satisfied. The *getFitnessVariance()* function calculates the fitness variance of the last n generations.

Using our proposed genetic algorithm, the test approach improves a given test set to efficiently achieve a worst-case execution time and inaccurate results of the FGS algorithm. If the test detects violations of the requirements, the FGS algorithm has to be corrected and tested again.

5.4 Summary

In the last sections, we have defined a test approach that automatically and systematically searches for star combinations that provoke long execution times and inaccurate results of the PLATO FGS algorithm.

We have defined a genetic algorithm that uses test cases as individuals. Each test case is a combination of 30 stars. The search space of our genetic algorithm contains one star for each equivalence class combination. This reduces the space to our defined equivalence classes.

We have adopted the commonly used evolutionary mechanisms selection, crossover, and mutation such that they prevent the variability of stars in the populations.

Therefore, our genetic algorithm extensively scans the search space and does not converge prematurely.

Our main result is a two-criteria fitness function that is based on the measured execution time and quality index of the FGS algorithm. We have specified the quality index as a qualitative measure of the mathematical accuracy of the FGS algorithm. Using our specified fitness function our genetic algorithm automatically searches for star combinations that provoke long execution times and high quality indexes of the FGS algorithm. In this way, these combinations are an improvement of a given test set to support robustness testing.

The tester is able to specify the termination conditions and the impact of each criterion to the fitness value of a star combination. This makes our approach adaptable to various test goals.

In the following sections, we demonstrate the efficiency of our partitioning approach and genetic approach on the example of the PLATO FGS algorithm.

6 Evaluation

The evaluation of our proposed test approaches based on various test suites to investigate the efficiency of these approaches for satellite on-board image processing applications. Section 6.1 describes the implementation of our test environment. Sections 6.2 and 6.3 present the practical results from our test execution of the equivalence class partition testing and the execution of the genetic algorithm.

6.1 Implementation

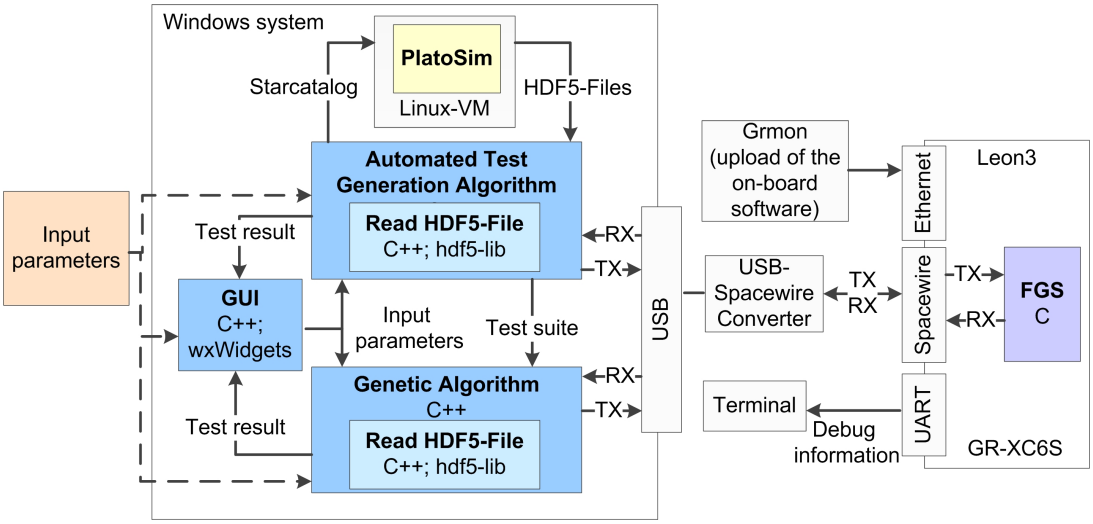

Figure 12: Block diagram of test setup

Figure 12 shows the block diagram of our test setup. As the figure depicts, our test environment runs on a Windows system. We implement the environment in C++ based on the model-view-controller pattern. Our goal is to improve the scalability and allow the tester to specify the input parameters with Graphical User Interface (GUI) or without GUI. We implement the GUI using the wxWidgets C++ library.

Our automated test generation algorithm returns a star catalog in order to simulate star data for missing input combinations. The catalog is a text file that includes right ascension, declination, and magnitude of the stars that should be simulated. We manually insert the catalog into the PLATO simulator PlatoSim [KU 18]. PlatoSim writes the simulated star data to a HDF5-file[3]. Therefore, our test environment uses an hdf5 C++ library to read the data. Each simulated star comprises an image sequence of several time steps of the star. Because PlatoSim is not developed for Windows systems, the simulator runs in a Linux virtual machine.

[3]HDF5 is a hierarchical file format for storing a large number of data types [The18].

As shown in Figure 12, we connect the Windows system via a SpaceWire[4]-Universal Serial Bus (USB) brick to a GR-XC6S Field Programmable Gate Array (FPGA) development board[5] running at 50MHz. A prototype of the PLATO FGS algorithm, written in C, runs on this board. Our test environment sends FGS algorithm input data and receives FGS algorithm results via SpaceWire. For that, it uses the C SpaceWire USB Application Programming Interface (API) Library for the SpaceWire USB brick. We load the FGS software with GRMON[6] onto the evaluation board. Via a Universal Asynchronous Receiver Transmitter (UART) interface we receive debug information, for example, stack size, hardware information, etc. in a terminal.

Our objective is to evaluate our approaches for the development and test of the FGS algorithm implementation. In addition, our goal is to test the execution time and mathematical accuracy of the FGS algorithm under realistic conditions. For example, the calculation with double values on the development board is slower than the same calculation on the windows system. Moreover, due to the different hardware components, the development board rounds the values differently than the windows system. This leads to different accuracies of the results. Therefore, we run the application under test on the target hardware and keep the test system in the software development cycle.

6.2 Experimental Results: Partitioning Approach

This section presents the experimental results of applying the proposed partitioning approach to generate a test suite for the test of the PLATO FGS algorithm.
In the experiments, we have specified the following parameters for the definition of equivalence classes on the input domain.

- initial radius r_0 for circular partitioning of the FPA: 1900 pixel
- number of magnitude sub-ranges: 6
- ratio r of the middle sub-area of the pixel to the pixel area: 0.2

The application of these parameters results in 8 equivalence classes of input parameter $\mathcal{P}$, 6 equivalence classes of the parameter $\mathcal{M}$ and 9 equivalence classes of input parameter $\mathcal{E}$. Additionally, the parameter $\mathcal{G}$ consists of two equivalence classes (G_G and G_{NG}). Thus, our automated testing framework needs 864 test cases to completely cover the whole input domain of the FGS algorithm.

[4]SpaceWire is a data-handling network for spacecraft defined in the European Cooperation for Space Standardization standard ECSS-E-ST-50-12C [Eur17].

[5]FPGA development board used for the development of Leon processor systems [PEN11].

[6]GRMON is a debug monitor for Leon Processors. Please refer [Cob17] for more information.

6.2 Experimental Results: Partitioning Approach

To practically evaluate our approach, we have investigated the quality of two different test suites with respect to their multidimensional coverage on the input domain. One test suite contains 82 randomly generated stars. Our automated test generation algorithm identifies 10 stars in the suite as redundant. The test suite achieves 8.3 % coverage of the input domain with respect to our defined multidimensional coverage criteria. Algorithm 3 from Section 4.5 enhances the suite to achieve complete coverage on the input domain. In the following, we call it the complete test suite. The other test suite contains 902 randomly generated but evenly distributed stars. We call it random test suite and did not improve it. Table 2 presents the coverage of the test suites for each input parameter as well as the achieved multidimensional coverage for each test suite.

Table 2: Individual and multidimensional coverage of the test suites

	random test suite	complete test suite
Number of test stars	902	874
Number of covered input combinations	112	864
$C_{\mathcal{P}}$ in %	87.5	100
$C_{\mathcal{M}}$ in %	16.7	100
$C_{\mathcal{E}}$ in %	100	100
$C_{\mathcal{G}}$ in %	100	100
Multidimensional coverage in %	13.0	100

Table 2 shows, the utilization of the equivalence class partitioning method reduces the random test suite by hundreds of redundant test cases. Thereby, the method increases the efficiency of the test process. The random test suite achieves a high individual coverage of three input parameters. But due to the low individual coverage of the input parameter $\mathcal{M}$, the multidimensional coverage of the test suite is also low. Furthermore, Table 2 exhibits that the complete test suite covers the whole input domain of the FGS algorithm.

For the assessment of the equivalence class definitions, we have automatically inserted some faults into the code of the centroid calculation of the PLATO FGS algorithm. These injected faults belong to three classes: missing assignment, wrong assignment and wrong condition. For each test execution, we have injected a single error at a different position in the code. Our objective is to check whether the test suite with complete multidimensional coverage of the input domain achieves a higher error-detection capability than the random test suite.

In each experiment, our test application sent 1000 exposures of each selected star from the test suites to the development board where the FGS centroid algorithm is running. Our test application averages the resulting centroid positions over all

exposures and compares it with the centroid position in the corresponding star catalog. If the deviation of the positions is greater than a predefined, required value, the test detected the error.
An excerpt from the output for a test case detecting a missing assignment error is given by Table 3. The high deviation between the calculated position and the given position reveals an error in the centroid calculation.

Table 3: Output for a sample test case

i_G	i_P	i_M	i_E	Star-Id	x in pixel	y in pixel	deviation in pixel	result
1	1	0	8	28	2.96×10^{19}	2.38×10^{19}	3.80×10^{19}	error detected

In total, during the experiments, we have injected three missing assignment errors, three wrong assignment errors, and three wrong condition errors. Table 4 summarizes the results of the experiments for both test suites.

Table 4: Test suites evaluation results

	random test suite	complete test suite
# test cases	112	864
execution time in h	~ 1.5	~ 11.8
# detected errors	1	3
# undetected errors	9	7
error-detection capability in %	10	30

Table 4 shows, that both test suites do not reveal all injected errors with respect to the given test criterion. The random test suite, as well as the complete test suite, detects one missing assignment error. In addition, 6 test cases from the complete test suite reveal one wrong assignment error and 1 test case detects one wrong condition error. During the test development, we detected an error in the FGS algorithm code that we did not inject. But using the specified test criterion, both test suites do not detect the error. Moreover, the tests do not detect three of the injected errors because the deviation only changes slightly in case of these errors. In addition, for some test cases, the deviation increases and in others decreases in such a way that the specified deviation value is not reached. The tests also do not detect the other three injected errors and the unintended error because these errors affect the other parameters estimated by the centroid calculation. Therefore, the deviation for all test cases is smaller than the predefined value in the test criterion. However, the error-detection capability of the complete test suite is with

respect to the specified test criterion three times higher than the error-detection capability of the random test suite. Nevertheless, for both test suites, the error-detection capability is low because we have specified that tests only detected errors if the deviation between the calculated centroid position and the reference centroid position from the star catalog rise above a specified value. However, not all injected errors affect the results in such a way that the deviation increases. Therefore, our specified test criterion is not suitable to detect all injected errors.

This shows, that the specified test criterion plays an important role in the success of the tests. We have specified another test criterion: the test passes if the distance between the centroid position calculated by an erroneous calculation and the centroid position resulting from an assumed error-free calculation exceeds a predefined value. In this case, both test suites reveal all injected errors and the unintended error. But not all test cases of the test suites detect the errors. This means special input combinations are more capable to detect errors than others. In case of three injected errors and the unintended error, the percentage of error-detecting test cases from the random test suite is up to one third compared to the percentage of error-detecting test cases of the complete test suite. In the other cases, the percentage of error-detecting test cases is approximately 99 % for both test suites. Therefore, for this test criterion, the complete test suite has a higher error-detection capability than the random test suite.

Our partitioning approach reduces the number of relevant test cases. Therefore, applying the approach increases the test efficiency. The results show that the error-detection capability of the test suite that completely satisfies our multidimensional coverage criteria is significantly higher than the capability of the random test suite. The success of our approach depends on the specified test criterion as well as on the definition of the equivalence classes. Therefore, further investigations are needed.

6.3 Experimental Results: Genetic Algorithm

This section presents the results of applying our proposed genetic approach to examine the execution time and mathematical accuracy of the PLATO FGS algorithm. Using the test set generated by the first approach, without stars with Gaussian-PSF shape, results in 1.5×10^{46} possible combinations of 30 stars. Testing this amount of combinations is infeasible. Therefore, we have investigated whether our approach efficiently examines many test cases to increase the confidence in the system behavior.

In the first tests, our genetic algorithm optimizes solutions for one fitness criteria: execution time or quality index. For that, we have set the respective weighting factor w_{time} or w_{taste} to 1 and the other to 0. In addition, we have set the reference values a_{time} and a_{taste} to 1. The resulting maximum execution time and quality index serves as reference values for subsequent experiments.

For the experiments, we have inserted the input parameter specification given by Table 5 to examine the satisfaction of a specified time requirement of the PLATO FGS algorithm. We have used the same parameter specification, except that we have set w_{taste} to 1 and w_{time} to 0, to investigate the qualitative measure of the accuracy of the FGS algorithm.

Table 5: Input parameters of the genetic algorithm evaluating execution time

Input parameter	Parameter value
Population size	24
Number of genes per individual	30
Weighting factor w_{time}	1
Weighting factor w_{taste}	0
Reference time in ms	254.45
Reference taste	1.68×10^{-10}
Maximum execution time in ms	300
Crossover probability p_c	0.5
Mutation probability p_m	0.06
Maximum number of generations	50
Observing Iterations	10
Permitted variance	1×10^{-5}

Due to the small population size, the diversity of genes is low. Only the mutation process inserts non-present genes into the population. Therefore, we have set the mutation probability to 0.06 to avoid premature convergence. In the case of premature convergence, the search ends in a random local optimum in early generations [GKK04, p. 42].

During the experiments, our test application sends each selected star combination to the development board. Then, the FGS algorithm subsequently calculates the centroids of the stars. The result and the execution time of the centroid calculation are always the same for a star exposure. Furthermore, the centroid calculation is the most time-consuming part of the FGS algorithm. Therefore, we have specified that the FGS algorithm calculates the centroid for each star exposure only once to be more efficient. Our test application saves the calculated centroid and time required for each exposure of the stars. If a star appeares another time in a selected combination the test application sends the known centroid to the FGS algorithm. The algorithm then calculates the star direction vectors and performs the QUEST algorithm. The execution time of the FGS algorithm for a star combination is the sum of the time required for the centroid calculation of each star in this combination and the execution time of the star vector calculation and the QUEST algorithm. Our test application averages the execution times and the re-

sulting quality indexes over 1000 exposures. Based on these values, our developed fitness function calculates the fitness value of the executed star combinations. The genetic algorithm terminated if

- it completed 50 generations, or
- the execution time of the FGS algorithm for one test star combination exceeded 300 ms, or
- the variance of the maximum fitness values of the last 10 observed generations fell below the permitted variance of 1×10^{-5}.

We have specified no termination condition with respect to the quality index because no PLATO requirement exists for this measure. For one test the genetic algorithm uses a given randomly generate test set, denoted as incomplete test set, as its search space. For another test, our algorithm uses given complete test set with respect to our multidimensional coverage criteria as search space. Our objective is to compare the capability of our genetic algorithm to provoke a long execution time or a high quality index of the FGS algorithm using the different search spaces.

Figure 13 to Figure 14 show the resulting maximum execution times and quality indexes for the incomplete test set and the complete set. In the experiments, our genetic algorithm evolves star combinations first for the execution time and then for the delivered quality index.

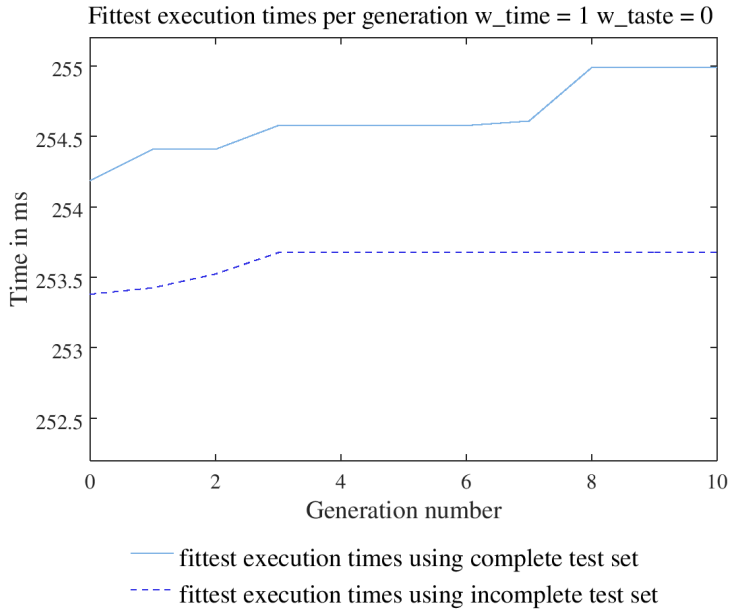

Figure 13: Maximum execution time per generation evaluating execution time

Figure 13 illustrates the fittest execution times achieved by the use of the incomplete test set and the complete set subsequently. The genetic algorithm optimizes star combinations based on their execution time. Both tests terminates after 10 generations because the variance of the maximum fitness values per generation fell below 1×10^{-5}. In both tests, the maximum execution times do not violate the specified timing requirement. Using the incomplete test set, our genetic algorithm provides a maximum execution time after three generations. However, Figure 13 shows that the maximum execution time was longer using the complete test set as search space.

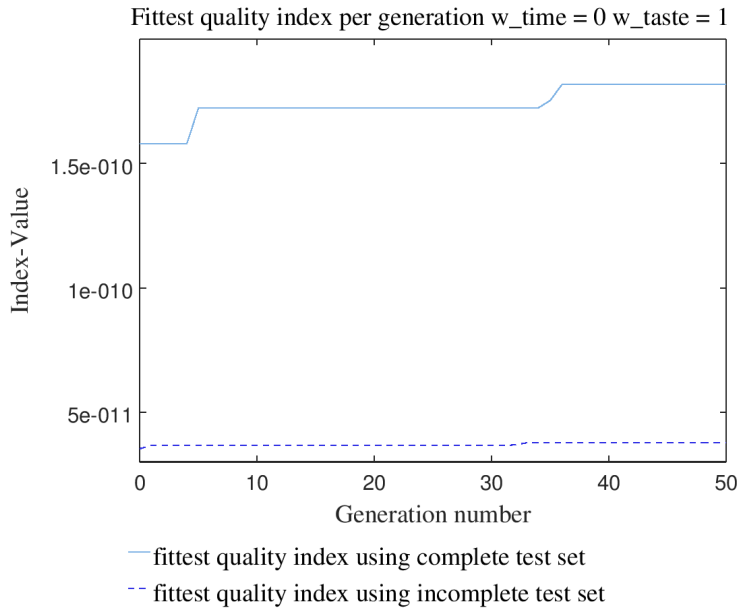

Figure 14: Maximum quality index per generation evaluating accuracy

Figure 14 shows the fittest quality indexes per generation using the incomplete test set as search space in one test and using the complete set in the other test. The genetic algorithm optimizes test star combinations based on the resulting quality index. Using the complete test set results in higher quality indexes than using the incomplete test set.

In both experiments, the complete test set achieves better results. The fittest solutions achieved by the complete test set covers 26 equivalence class combinations that are not covered by the incomplete test set. Because the complete test set covers all equivalence class combinations, it is more capable to provoke a long execution time or a high quality index than the incomplete test set. However, our genetic algorithm does not find test star combinations that violate specified timing requirements.

We have used the resulting maximum execution time and quality index as reference values a_{time} and a_{taste} for another experiment. We have compared the capability of the complete test set and the incomplete set to provoke a long execution time and a high quality index. Table 6 shows the input parameter specification for this experiment.

Table 6: Input parameters of the genetic algorithm evaluating two criteria

Input parameter	Parameter value
Population size	50
Number of genes per individual	30
Weighting factor w_{time}	0.5
Weighting factor w_{taste}	0.5
Reference time in ms	254.998
Reference taste	1.82×10^{-10}
Maximum execution time in ms	300
Crossover probability p_c	0.5
Mutation probability p_m	0.06
Maximum number of generations	75
Observing Iterations	75
Permitted variance	1×10^{-5}

To test a higher number of test star combinations, we have increased the population size to 50 individuals and the maximum number of generations to 75. In addition, we have set the weighting factors w_{time} and w_{taste} to 0.5. Hence, the execution time and the quality index affect the fitness value of a test star combination.
Figure 15 depicts the maximum fitness values per generation for evaluating star combinations with respect to execution time and quality index of the FGS algorithm. The figure shows that the fitness value of test cases from the complete test set is higher than for the incomplete test set. The fitness values for the incomplete test set do not evolve. The genetic algorithm finds a maximum fitness value in the first generations for this set. In case of the complete test set the fitness value evolves until generation 35.
Figure 16 shows that for both test sets the resulting quality index for these maximum fitness values have the similar course as the corresponding fitness values. Because the difference between the relative quality indexes is higher than the differences between the relative execution times (see Figure 17), the differences in the relative quality indexes leads to higher impact on in the fitness values. However, the test cases from the complete test set provoke longer execution times and higher quality indexes than test cases from the incomplete test set. The figures also show, that the execution time and the quality index do not depend on each other.

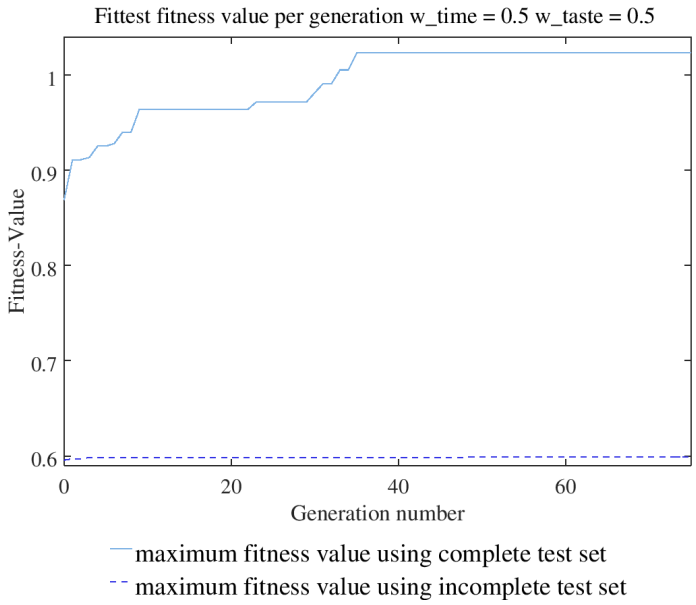

Figure 15: Maximum fitness value per generation evaluating execution time and accuracy

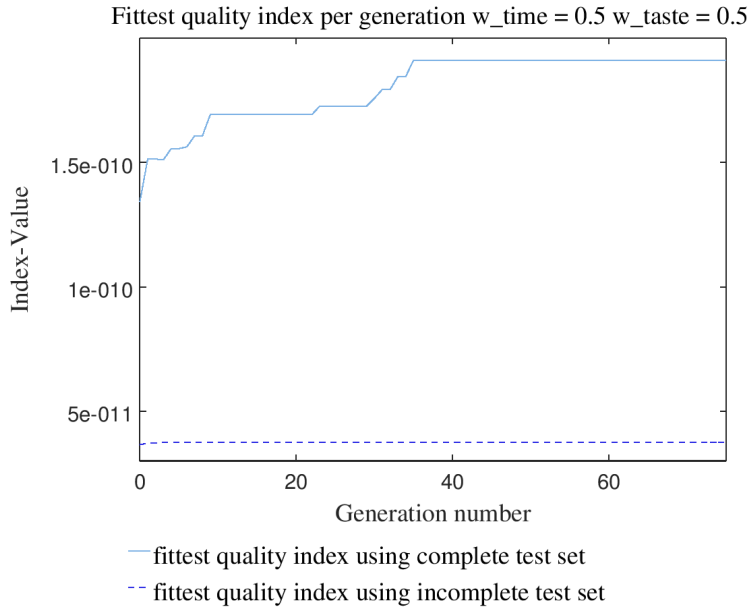

Figure 16: Maximum quality index per generation evaluating execution time and accuracy

In all tests using the complete test set as search spaces the longest execution time and the maximum quality index are higher than using the incomplete set.

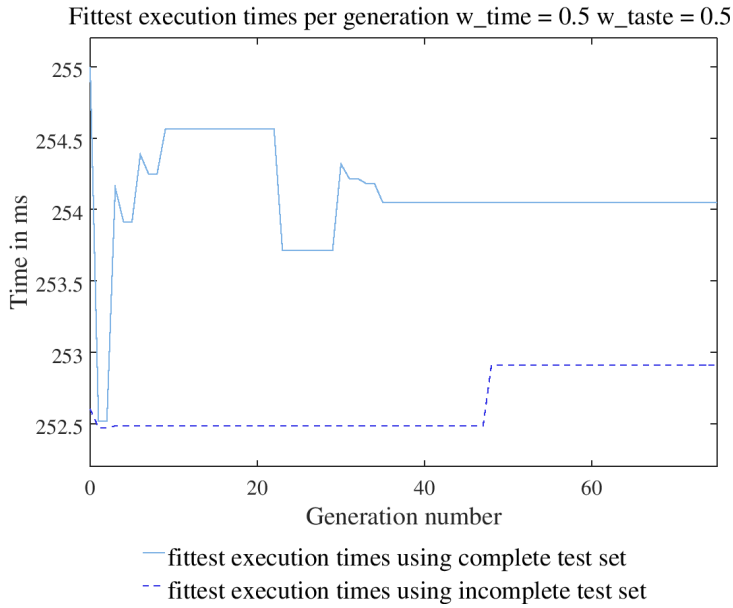

Figure 17: Maximum execution time per generation evaluating execution time and accuracy

The evolved solutions are better because the complete test set provides star combinations that provoke long execution times and inaccurate results of the FGS algorithm. These combinations are not possible with the random test set because it does not cover the corresponding equivalence class combinations. However, our proposed genetic algorithm automatically provides worse execution times and quality indexes in a few generations. Hence, it improves the efficiency of the software testing process. But it will never examine all possible 1.6×10^{46} combinations. Therefore we can not rule out if there are other combinations that provoke longer execution times or higher quality indexes. However it increases the confidence in the temporal behavior and accuracy of the satellite image processing application.

7 Conclusion

This section summarizes the results regarding our proposed partitioning approach and genetic algorithm. Moreover we discuss future work.

Due to the large input domain of on-board image processing applications, an enormous amount of test cases is possible. This makes it infeasible to capture the whole input domain and execute the test cases exhaustively. In addition, due to the complex computations, performed by the satellite on-board image processing applications, it is difficult to find test cases that provoke mission-critical behavior. Therefore we have presented two novel test approaches to overcome these problems: First, we have defined a partitioning approach that systematically generates a reasonably small test suite with complete coverage on the input domain of satellite on-board image processing applications. Second, we have defined a novel genetic algorithm specifically tailored to automatically find test cases that provoke mission-critical behavior of a given satellite on-board image processing application.

To achieve the goal of our first approach, we have defined the following parts.

- A dedicated partitioning for each input parameter of a satellite on-board image processing application.
- Multidimensional coverage criteria based on the equivalence class definitions to assess a given test suite with respect to its coverage on the input domain.
- An automated test generation algorithm that systematically generates missing but relevant test cases.

As a result, our approach is able to fully automatically generate test suites that are complete with respect to our defined multidimensional coverage criteria. The tester specifies the size of our equivalence classes. This makes our approach adaptable to other applications.

In our second approach, the genetic algorithm uses the complete test suite generated by our partitioning approach as search space to generate a combination of test cases. Because the test suite contains at most one test case for each equivalence class combination, the search space is reduced.

We have defined a novel two-criteria fitness function. It is based on the execution time and the mathematical accuracy of a given satellite on-board image processing application. Using that function our genetic algorithm automatically steers the search for test cases that provoke long execution times or inaccurate results or both. The tester is able to specify which criterion has more impact on the fitness value of a test case. In addition, the tester specifies the input parameters of the genetic algorithm, for example, population size, crossover probability, termination conditions, etc. This makes our proposed genetic algorithm more flexible and adaptable to different test goals and various on-board image processing applications.

We have investigated the effectiveness of our proposed test approach on the FGS algorithm as an application with high criticality for the PLATO mission. In the experiments, our automated test generation algorithm generates a test suite that is complete with respect to our multidimensional coverage criteria. To demonstrate the effectiveness of our partitioning approach, we have compared the error-detection capability of a randomly generated test suite and the generated complete test suite. The use of our equivalence classes of the input parameters reduces the number of redundant test cases in the randomly generated test suite by 87.6%.

In the experiments, we have successively injected 9 errors in the code of the FGS algorithm to investigate the error-detection capability of both test suites. We have used two different test criteria: First, a test case detects an error if the distance between the calculated centroid position and a given position is larger than a predefined value. Second, a test case detects an error if the distance of the erroneous calculated position and an assumed error-free calculated position exceeds a specified value.

We have observed that different test criteria lead to different test results. For the first specified criterion, the complete test suite detects 3 injected errors while the randomly generated test suite detects 1 injected error. Therefore, the error-detection capability of the complete test suite is about 3 times higher than the capability of the randomly generated test suite. But both test suites do not detect all injected errors.

In the second experiment, we have used the second test criterion. In this case, both test suites detect all 9 injected errors and one unintended error. However, not all test cases in the test suites detect all errors. In the case of 3 injected errors and the unintended error, the percentage of error-detecting test cases in the complete test suite is again about 3 times higher than for the randomly generated test suite. For the other 6 injected errors, the percentage of error-detecting test cases is for both test suites approximately 99%.

The experiments showed that a systematic test using our proposed partitioning approach increases the error-detection capability of a given test suite. This makes the partitioning approach efficient and effective. In addition, it facilitates the automated generation, the execution, and the evaluation of test cases.

To demonstrate the efficiency of our genetic approach, we have investigated the capability of the algorithm to automatically find test cases that support robustness testing of the satellite on-board image processing application. In subsequent tests, our genetic algorithm uses a randomly generated test set and the complete test set generated by our partitioning approach as search space. In the first experiments, we have specified that our genetic algorithm automatically evolves test cases with respect to the execution time or the mathematical accuracy of the FGS algorithm. In a second experiment, we changed the input parameters in such a way that our

algorithm automatically evolves test cases with respect to the execution time and the mathematical accuracy of the FGS algorithm.

In all experiments, test cases generated from the complete test set provoke maximum execution times that are approximately 1.5ms longer than for the randomly generated test set. The maximum qualitative accuracy, measured by a quality index, provoked by the test cases from the complete test set is about 3.7 times higher than for the other test set. However, we can not exclude that there might be other test cases that provoke longer execution times or less accurate results or both due to the huge amount of possible combinations. The results show that our genetic approach automatically searches for test cases that provoke long execution times and inaccurate results of a given satellite on-board image processing application.

Both proposed approaches, which defining a partition to systematically generate a reasonably small test suite and a novel genetic algorithm specifically tailored to automatically find special test cases, are suitable to generate test cases for black box tests of image processing applications. They systematically capture the large input domain and efficiently searches for test cases that support robustness testing. Therefore, the novel test approaches increase the efficiency of the test process and increase the test coverage of the applications.

Our work opens up several ways for future work, for example

- injecting errors in the binary code,
- adapt the fitness function of the genetic algorithm,
- considering additional parameters in the partitioning approach,
- investigate the efficiency of our test approaches for other applications.

So far, we have injected errors in the application code. But in space, many missions suffer from cosmic radiation that flips bits in the binary code or cause hot pixels in the input images. Therefore, we plan to investigate the efficiency of our approaches by injecting errors in the input data or in the binary code of the application.

In this thesis, we have considered the TASTE value as a qualitative measure of the mathematical accuracy. To investigate the accuracy of the application more precisely we will adapt the fitness function of our genetic algorithm. For that case, we consider the errors of the results, for example, angle errors for each axis, as a criterion for the mathematical accuracy.

In our definitions, we have only considered the input parameters of satellite on-board image processing applications. However, it may also be required to apply the equivalence class partitioning method to the output parameters of the application.

Furthermore, we have investigated our developed approaches by means of a single satellite on-board image processing application. Due to the flexibility of our approaches the suitability for other application, for example, blob feature extraction in the robotics domain [BBV00] [MWC+06], can be investigated.

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AOCS	Attitude and Orbit Control System
API	Application Programming Interface
CCD	Charge Coupled Device
COM	Center Of Mass
DLR	German Aerospace Center
ECSS	European Cooperation for Space Standardization
ESA	European Space Agency
F-DPU	Fast camera Data Processing Unit
FGS	Fine Guidance System
FOV	Field of View
FPA	Focal Plane Assembly
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDF5	Hierarchical Data Format version 5
PLATO	PLAnetary Transits and Oscillation of stars
PSF	Point Spread Function
QUEST	QUaternion ESTimator
S/C	Spacecraft
SNR	Signal-to-Noise Ratio
TPT	Time Partition Testing
TU	Technical University
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

References

[AM99] Alander, Jarmo T. ; Mantere, Timo: Automatic software testing by genetic algorithm optimization, a case study. In: *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, 1999, S. 1–9

[BBV00] Bruce, James ; Balch, Tucker ; Veloso, Manuela: Fast and inexpensive color image segmentation for interactive robots. In: *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on* Bd. 3 IEEE, 2000, S. 2061–2066

[BDH98] B.F. Jones ; D.E. Eyres ; H.-H. Sthamer: 97063.dvi. In: *THE COMPUTER JOURNAL* 41 (1998), Nr. 2, S. 98–107

[BK06] Bringmann, Eckard ; Krämer, Andreas: Systematic testing of the continuous behavior of automotive systems. In: *Proceedings of the 2006 international workshop on Software engineering for automotive systems* ACM, 2006, S. 13–20

[BMP12] Bhandari, Dinabandhu ; Murthy, CA ; Pal, Sankar K.: Variance as a stopping criterion for genetic algorithms with elitist model. In: *Fundamenta Informaticae* 120 (2012), Nr. 2, S. 145–164

[BN03] Broekman, Bart ; Notenboom, Edwin: *Testing embedded software*. Pearson Education, 2003

[BQ15] Bhat, Asma ; Quadri, SMK: Equivalence class partitioning and boundary value analysis-A review. In: *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on* IEEE, 2015, S. 1557–1562

[Cob17] Cobham Gaisler AB: *GRMON2 User's Manual*. `www.cobham.com/gaisler`. Version: 2017

[DLR17] DLR: *Grünes Licht für europäisches Weltraumteleskop PLATO*. `http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10081/151_read-22858/#/gallery/27241`. Version: 27.06.2017

[ESA12] ESA: *ESA's 'Cosmic Vision'*. `http://www.esa.int/Our_Activities/Space_Science/ESA_s_Cosmic_Vision`. Version: 19.10.2012

[Eur17] European Space Agency: *SpaceWire Standard*. `http://spacewire.esa.int/content/Standard/Standard.php`. Version: 2000-2017

[GKK04] Gerdes, Ingrid ; Klawonn, Frank ; Kruse, Rudolf: *Evolutionäre Algorithmen: Genetische Algorithmen - Strategien und Optimierungsverfahren - Beispielanwendungen*. 1. vieweg, 2004

[Gri17] Griessbach, Denis: *Fine Guidance System Algorithm*. 2.3(Draft1). DLR, Berlin, 13.04.2017

[HP13] Huang, Wen-ling ; Peleska, Jan: Exhaustive model-based equivalence class testing. In: *IFIP International Conference on Testing Software and Systems* Springer, 2013, S. 49–64

[HP16] Huang, Wen-ling ; Peleska, Jan: Complete model-based equivalence class testing. In: *International Journal on Software Tools for Technology Transfer* 18 (2016), Nr. 3, S. 265–283. – ISSN 1433–2779

[JES98] Jones, Bryan F. ; Eyres, David E. ; Sthamer, H-H: A strategy for using genetic algorithms to automate branch and fault-based testing. In: *The Computer Journal* 41 (1998), Nr. 2, S. 98–107. – ISSN 1460–2067

[JT99] Jarmo T. Alander ; Timo Mantere: Automatic software testing by genetic algorithm optimization, a case study. In: *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering* (1999), S. 1–9

[Kan04] Kaner, Cem: Teaching domain testing: A status report. In: *Software Engineering Education and Training, 2004. Proceedings. 17th Conference on* IEEE, 2004, S. 112–117

[KU 18] KU LEUVEN: *PLATO Simulator: Main Page*. `http://ivs-kuleuven.github.io/PlatoSim3/`. Version: 17.05.2018

[Lie02] Liebe, Carl C.: Accuracy performance of star trackers-a tutorial. In: *IEEE Transactions on aerospace and electronic systems* 38 (2002), Nr. 2, S. 587–599

[MA00] Mantere, Timo J. ; Alander, Jarmo T.: Automatic image generation by genetic algorithms for testing halftoning methods. In: *Intelligent Robots and Computer Vision XIX: Algorithms, Techniques, and Active Vision* Bd. 4197 International Society for Optics and Photonics, 2000, S. 297–309

[MC14] Markley, F L. ; Crassidis, John L.: *Fundamentals of spacecraft attitude determination and control*. Bd. 33. Springer, 2014

[Moh05] Moheb R. Girgis: Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm. In: *Journal of Universal Computer Science* 11 (2005), Nr. 6, S. 898–915

[MWC+06] Merino, Luis ; Wiklund, Johan ; Caballero, Fernando ; Moe, Anders ; De Dios, José Ramiro M. ; Forssen, P-E ; Nordberg, Klas ; Ollero, Anibal: Vision-based multi-UAV position estimation. In: *IEEE robotics & automation magazine* 13 (2006), Nr. 3, S. 53–62

[PEN11] PENDER ELECTRONIC DESIGN GmbH: *GR-XC6S-product_sheet*. 2011

[Pet09] Peter Liggesmeyer: *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. 2. Spektrum Akademischer Verlag, 2009. – ISBN ISBN 978–3–8274–2056–5

[SBW01] Sthamer, Harmen ; Baresel, André ; Wegener, Joachim: Evolutionary testing of embedded systems. In: *Proceedings of the 14th International Internet & Software Quality Week (QW'01)* (2001), S. 1–34

[Shu08] Shuster, Malcolm D.: The TASTE test. In: *Advances in the Astronautical Sciences* 132 (2008), S. 71–81

[SK09] Srivastava, Praveen R. ; Kim, Tai-hoon: Application of genetic algorithm in software testing. In: *International Journal of software Engineering and its Applications* 3 (2009), Nr. 4, S. 87–96

[SPA16] Sharma, Akshat ; Patani, Rishon ; Aggarwal, Ashish: Software Testing Using Genetic Algorithms. In: *International Journal of Computer Science & Engineering Survey* 7 (2016), Nr. 2, S. 21–33. `http://dx.doi.org/10.5121/ijcses.2016.7203`. – DOI 10.5121/ijcses.2016.7203. – ISSN 09763252

[SW15] Suszyński, Robert ; Wawryn, Krzysztof: Stars' Centroid Determination Using PSF-Fitting Method. In: *Metrology and Measurement Systems* 22 (2015), Nr. 4, S. 547–558

[The18] The HDF Group: *HDF5*. `https://portal.hdfgroup.org/display/HDF5/HDF5`. Version: April 05, 2018

[VM14] Varshney, Sapna ; Mehrotra, Monica: Automated software test data generation for data flow dependencies using genetic algorithm. In: *International Journal* 4 (2014), Nr. 2

[Wei15] Weicker, Karsten: *Evolutionäre Algorithmen*. 3., überarb. und erw. Aufl. Wiesbaden : Springer Vieweg, 2015. http://dx.doi.org/10.1007/978-3-658-09958-9. http://dx.doi.org/10.1007/978-3-658-09958-9

[WM01] Wegener, Joachim ; Mueller, Frank: A comparison of static analysis and evolutionary testing for the verification of timing constraints. In: *Real-time systems* 21 (2001), Nr. 3, S. 241–268

[WP17] Wohlfeil, Jürgen ; Peter, Gisbert: *Instrument Technical Requirement Document (TRD)*. 3.2 (Draft 7). DLR, Berlin, 18.07.2017