Equivalence Class Definition for Automated Testing of Satellite On-Board Image Processing

Ulrike Witteck[1(✉)], Denis Grießbach[1(✉)], and Paula Herber[2(✉)]

[1] Institute of Optical Sensor Systems, German Aerospace Center (DLR), Adlershof, Berlin, Germany
{ulrike.witteck,denis.griessbach}@dlr.de
[2] Embedded Systems Group, University of Münster, Münster, Germany
paula.herber@uni-muenster.de

Abstract. On-board image processing technologies in the satellite domain are subject to strict requirements with respect to reliability and accuracy in hard real-time. Due to the large input domain of such processing technologies it is impracticable or even impossible to execute all possible test cases.

As a solution we define a novel test approach that efficiently and systematically captures the input domain of satellite on-board image processing applications. We first partition each input parameter into equivalence classes. Based on these equivalence classes we define multidimensional coverage criteria to assess the coverage of a given test suite on the whole input domain. Finally, our test generation algorithm automatically inserts missing but relevant test cases into the given test suite such that our multidimensional coverage criteria are satisfied.

As a result we get a reasonably small test suite that covers the complete input domain. We demonstrate the effectiveness of our approach with experimental results from the ESA medium-class mission PLATO.

Keywords: Image processing · Software testing · Equivalence class partitioning · Satellite systems

1 Introduction

On-board image processing applications in the satellite domain are subject to strict requirements with respect to reliability and mathematical accuracy in hard real-time. The large input domain of such applications makes manual testing error-prone and time-consuming. To overcome that problem, we need a test approach that automatically and systematically generates test cases for such image processing applications. The major problem of the automated generation of test cases is the large amount of input parameters and their possible combinations. This leads to a high number of test cases which makes the systematic and efficient coverage of the complete input domain expensive.

M. van Sinderen and L. A. Maciaszek (Eds.): ICSOFT 2019, CCIS 1250, pp. 3–25, 2020.
https://doi.org/10.1007/978-3-030-52991-8_1

Automated test approaches for different domains, for example, for automotive and railway applications, are presented in [2,7]. The authors investigate applications with huge input domains and complex functional behavior. However, their focus is on event-driven, reactive real-time systems and the approaches are not tailored to the domain of on-board image processing applications.

In this paper we present an extended version of our test approach given in [14]. This approach systematically selects test cases from the huge input domain given in image processing applications. Our objective is to achieve a high coverage of the input domain using a reasonably small test suite. To achieve that goal we adopt the equivalence class partition testing method. This method partitions a given domain into disjoint sub-domains called equivalence classes [13]. Only some test values are used as representatives from each class. That reduces the number of required test cases [1], but still systematically covers the respective domain. We use that method to partition each input parameter of the on-board image processing application into equivalence classes. Furthermore, we define multidimensional coverage criteria that combines individual coverage criteria for each input parameter. Finally, we specify a test generation algorithm that uses our multidimensional coverage criteria to automatically assess given test suites with respect to their coverage on the whole input domain. Moreover, the algorithm removes redundant test cases and inserts missing but relevant test cases. As a result we get a reasonably small test suite that covers the complete input domain of satellite on-board image processing applications.

To investigate the efficiency of our test approach using equivalence class definitions, we use the Fine Guidance System (FGS) algorithm of the European Space Agency (ESA) mission PLAnetary Transits and Oscillation of stars (PLATO) as a case study [14]. The FGS algorithm is a satellite on-board image processing algorithm to calculate the high-precision attitude of the spacecraft by comparing tracked star positions with known star positions from a star catalog. Recent studies have shown that some of the input parameters as presented in [14] can be partitioned more beneficial. In this paper we therefore present redefined equivalence classes for two input parameters: object position and subpixel position on the image plane. Moreover, we use an improved test criterion to investigate the effectiveness of our test approach. The experimental results show the effectiveness of our partitioning approach in terms of an increased error detection capability.

This paper is structured as follows: In Sect. 2, we briefly introduce equivalence class partition testing and give an overview of the ESA PLATO mission including the FGS algorithm. In Sect. 3, we outline related work about equivalence class testing for real-time systems. In Sect. 4, we present our redefined equivalence classes as well as the automated test generation algorithm for satellite on-board image processing applications. In Sect. 5, we present our experimental results and compare them with the results presented in [14]. We conclude with a summary in Sect. 6.

2 Preliminaries

We introduce the general concept of equivalence class partition testing and give an overview of the PLATO mission and its mission-critical FGS algorithm to understand the remainder of this paper.

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 the necessary test cases from a huge input domain is a major problem when testing an application [11].

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. 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. 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 [1,7,11].

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 [1].

2.2 Context: PLATO Mission

PLATO is an ESA mission in the long-term space scientific program "Cosmic Vision" [5]. The German Aerospace Center (DLR) manages the international consortium for developing the payload and scientific operation of the project [3].

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 large Field of 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 a fast camera FoV is $38.7° \times 38.7°$. The cameras are equipped with four Charge Coupled Devices (CCD) in the focal plane, each with 4510×4510 pixels.

Each fast camera comes with a data processing unit running the FGS algorithm. It calculates attitude data with an accuracy of milliarcseconds from the image data. This data is supplied to the spacecraft attitude and orbit control system. The FGS is regarded as being a mission-critical component which implies an extensive test procedure.

Many spacecraft missions use a FGS to obtain accurate measurements of the spacecraft orientation. We use the PLATO FGS algorithm as a case study to investigate the efficiency of our test approach. The attitude calculation of a telescope is based on measured star positions on the CCD compared to their reference directions in a star catalog. Figure 1 gives an overview of the FGS algorithm [6].

The autonomous attitude tracking is initialized with an initial attitude given by the space craft. For each pre-selected guide star, an initial sub-window position is calculated by means of the camera model, which transforms from sky coordinates to pixel coordinates and vice versa [6]. Guide stars are predefined stars in a star catalog that satisfy given criteria. For example, the star magnitude is within a certain range, the star has very low contamination, etc. The FGS algorithm calculates centroids after reading 6×6 pixel sub-window every 2.5 s from the full CCD image.

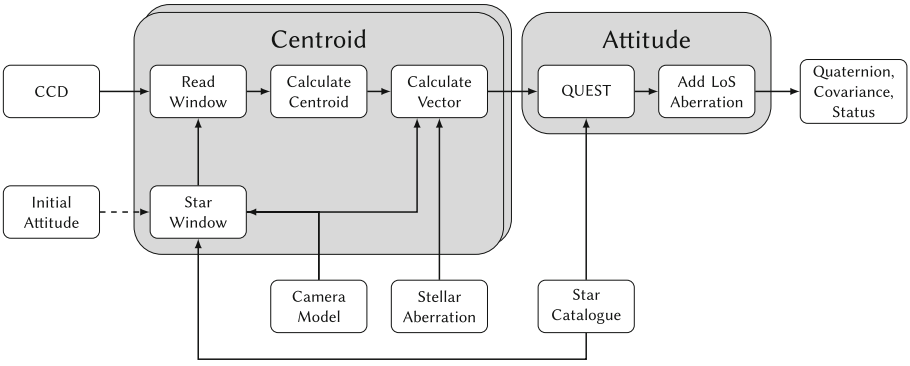

Fig. 1. Overview of the FGS algorithm [6].

A linear center of mass calculation estimates the initial 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. Equation 1 shows the Gaussian PSF observation model $h(i, j)$ of a single pixel [6].

$$h = \frac{I_m}{2\pi\sigma^2} \int_{i}^{i+1} e^{-\frac{(u-u_C)^2}{2\sigma^2}} du \int_{j}^{j+1} e^{-\frac{(v-v_C)^2}{2\sigma^2}} dv + D + \xi \tag{1}$$

The FGS algorithm uses the measured pixel intensities 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 FGS algorithm calculates the correction by means of the QR-decomposition [6]. In the next step, the pixel coordinates of the calculated centroid position are transformed into star direction vectors in the camera boresight reference frame. The x- and y-axis of the detector and the optical axis of the camera describe the boresight reference frame.

Finally, the FGS algorithm calculates an attitude, including covariance, from at least two star directions in the boresight reference frame and the corresponding reference vectors from a star catalog [14].

3 Related Work

Equivalence class partition testing "is probably the most widely described, and one of the most widely practiced, software testing techniques" [8] Various studies investigated equivalence class partition testing strategies for different domains, for example, railway, automotive, avionics, etc. [7]. We present some previously published work on equivalence class partition testing for real-time systems.

In the automotive domain, DaimlerChrysler Research developed a test approach, called Time Partition Testing (TPT), to test the continuous behavior of control systems. Bringmann and Krämer [2] explained the principle of the TPT approach using an exterior headlight controller as an example. In most cases, automotive embedded control systems are based on complex functional behavior and large input domains. To increase the test efficiency the TPT approach systematically selects test cases revealing redundant or missing test scenarios. Using a graphical 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 various 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 paper. However, the behavior of systems from this class is not dependent on the arrival time of input values. Hence, the TPT approach is not applicable to the system class that we consider [14].

In [7], the authors presented 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, the approach 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. Based on these classes, they generates a complete test suite in the following sense: First, at least one test in the suite fails if an application that violates a given specification is tested. 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 our considered satellite application. Hence, we present an approach that does not need such a description [14].

4 Equivalence Class Partitioning for Automated Test Generation

Satellite on-board image processing applications require various input parameters such as position of an object in the image, its brightness, sub-pixel position, its shape to distinguish different objects, etc. This leads to a huge input domain which makes testing expensive. Especially manual tests are error-prone and time-consuming. Thus, a test approach is needed that automatically and systematically generates test cases for such applications. However, a major challenge for automated test generation is the very large number of possible input parameter combinations. This potential enormous amount of test cases makes it hard to efficiently capture the complete input domain.

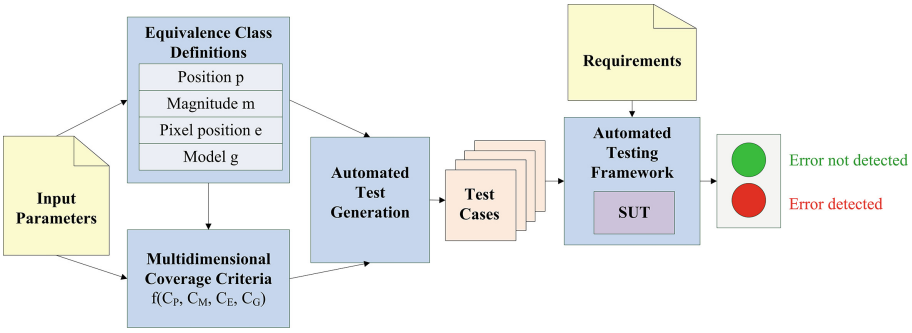

Fig. 2. Overview of the partitioning approach [14].

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. Moreover, our test approach assesses and enhances a given test suite. To evaluate the efficiency of our test approach, we investigate a case study, namely the PLATO FGS algorithm as described in Sect. 2.2.

Since satellite on-board image processing algorithms are subject to extremely strict requirements with respect to reliability and mathematical accuracy, such algorithms require extensive testing.

Figure 2 depicts an overview of our proposed partitioning approach. Our key idea is to define equivalence classes on input parameters that are typically used by satellite on-board image processing applications, namely position, magnitude, sub-pixel position, and distribution model. In this paper we present updated equivalence class definitions to partition the individual parameters. Recent studies have shown that some of the equivalence class definitions presented in [14] can be redefined more effectively. In a second step, we define multidimensional coverage criteria based on a combination of the individual criteria for each input parameter. After that, we define a test generation algorithm that automatically selects test cases that completely cover the whole input domain according to our multidimensional coverage criteria.

Our test objective is to automatically detect errors in the on-board image processing application code. To achieve this, our test generation algorithm selects a test case for each equivalence class combination from a given test suite as representatives. This reduces the number of redundant test cases. Furthermore, our algorithm generates new test cases for missing but relevant input combinations to reach a complete coverage of the input domain. The result is a reasonably small test suite that covers the whole input domain of the image processing application with respect to our multidimensional coverage criteria. The selected test cases serve as input for our automated testing framework. Moreover, we insert requirements for the automated evaluation of the image processing application results. If the test cases do not meet the requirements, an error is detected [14].

The following sections describe the mentioned steps of the partitioning approach in more detail, applying our new equivalence class definitions. We use the PLATO FGS algorithm as a case study.

4.1 Assumptions and Limitations

In the following, we consider systems whose input are objects in an image. In the case study, the observed objects are stars with magnitudes between 5.5 to 7.0, uniformly distributed in the image [6].

We consider four parameters that affect the mathematical accuracy of the FGS algorithm: the guide star position, its magnitude, sub-pixel position, and PSF shape. The evaluation of the test is based on the precision of the centroid position calculated by the FGS centroid algorithm as described in Sect. 2.2. The input of the centroid calculation is a single star image. Hence, we define a test star as a test case for the automated test generation.

4.2 Input Parameter Partitioning

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. Moreover, each pixel contains noise, which in combination with the signal determines its Signal-to-Noise Ratio (SNR). The centroid calculation needs at least 5 linear independent equations to estimate the 5 unknown parameters of the pixel observation (cf. Eq. (1)).

The FGS input star parameters named in Sect. 4.1 affect the mathematical precision and accuracy of the centroid estimation. Hence, we define the input domain as a set of input parameters I. The set includes the position on the Focal Plane Assembly (FPA) $\mathcal{P}$, the magnitude $\mathcal{M}$, the sub-pixel position $\mathcal{E}$ and the PSF shape $\mathcal{G}$. The tester specifies start values to calculate the borders of the equivalence classes. This makes our approach more flexible and parameters can also be excluded from the analysis [14].

In this section we describe how the quality of the centroid calculation depends on these parameters and present our partitioning concepts for each input parameter in I.

Position on the FPA. Among others, the distribution of the star signal depends on the star position on the FPA. Due to optical aberrations of the telescope, the PSF shape of the star is wider in the FPA corner than close to the center. If the other input parameters contain reasonably good, constant values then a small PSF leads to a low number of pixels with a high SNR. In case of a wide PSF, more pixel contain a signal but the SNR is low. Both cases can be sufficient for an accurate parameter estimation [14].

In [14], our idea is to partition the FPA into equally sized, circular areas. Recent studies have shown, that the PSF changes not only with the distance to the FPA center but also with the polar angle. In the study each class of parameter $\mathcal{P}$ contains two stars per class of parameter $\mathcal{E}$. The stars have a constant medium magnitude as well as worst-case non-Gaussian PSF. Figure 3 depicts the residual noise of stars per circular FPA area. The figure shows that the residual noise is lower if the star is positioned near the FPA corner or near the FPA border. Moreover, the figure illustrates that the equivalence class borders have been well chosen since the residual noise of the stars is changed between neighboring classes.

Figure 4 shows that the residual noise also depends on the polar angle of the stars. The figure depicts the residual noise of stars per polar angle area. Figure 4 shows that the residual noise is different for each class. However, we consider only stars in the image area of the CCDs. That means, for some polar angle areas particular circular areas can not be covered by a star. Therefore, these polar angle areas contain fewer stars than others. Moreover, the stars in these polar angle areas are located near the FPA center. Hence, the residual noise for that area is low. However, the polar angle area between 90° and 135° contains less stars but the residual noise is high. This indicates, that this area is not suitable to select guide stars for the PLATO mission from there.

Bases on the study, we update our equivalence class definition of the input parameter $\mathcal{P}$ and additionally partition the polar angle in equally sized circular sectors.

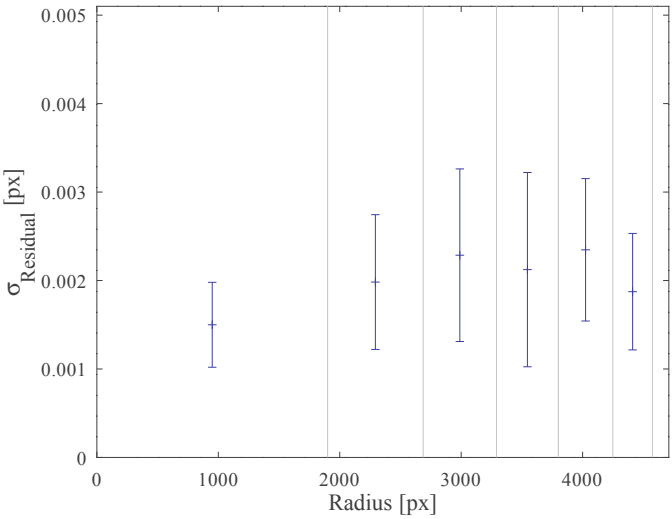

Fig. 3. Residual noise per radius of circular FPA areas.

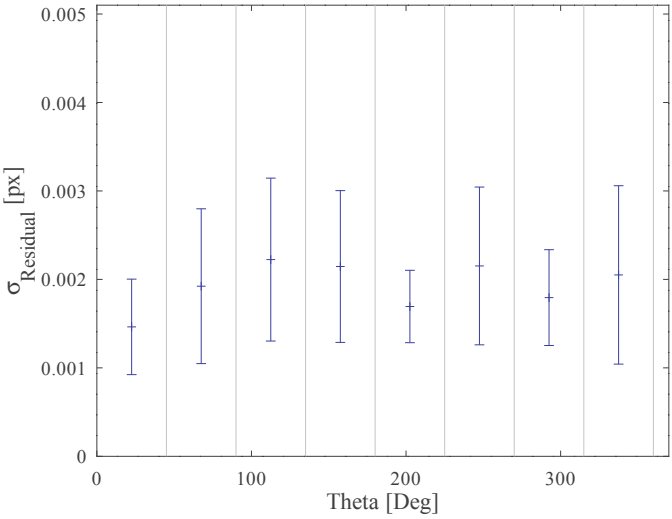

Fig. 4. Residual noise per polar angle of circular FPA areas.

The updated equivalence class definition is illustrated in Fig. 5. The rectangles represent the image area of the fast cameras CCDs and each circular ring sector corresponds to one equivalence class. The tester specifies the initial radius r_0 and the angle of the circular vectors θ_0.

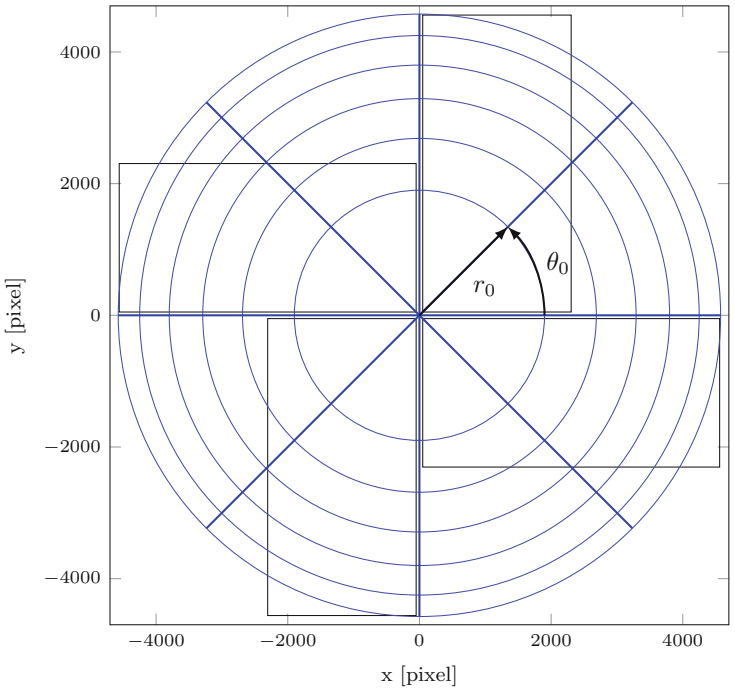

Fig. 5. FPA equivalence class example.

We partition parameter $\mathcal{P}$ into equivalence classes $P_{(r_i,\theta_j)}$. Each class $P_{(r_i,\theta_j)}$ corresponds to a circular ring sector of the FPA with inner radius r_{i-1} and outer radius r_i as well as right polar angle θ_{j-1} and left polar angle θ_j.

$$\mathcal{P} = P_{(r_0,\theta_0)} \cup P_{(r_0,\theta_1)} \cup ... \cup P_{(r_0,\theta_m)} \cup ... \cup P_{(r_n,\theta_m)} \tag{2}$$

where n is the number of radius border and m is the number of polar angle border.

Let S denote the set of available stars. A star $s \in S$ lies in an equivalence class $P_{(r_i,\theta_j)}$ if following condition holds:

$$r_{i-1} \leq p(s) < r_i, \text{ with } p(s) = \sqrt{{x_s}^2 + {y_s}^2} \tag{3}$$

and

$$\theta_{j-1} \leq t(s) < \theta_j, \text{ with } t(s) = \arctan \frac{x_s}{y_s} \tag{4}$$

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

Sub-pixel Position. In addition to the position on the FPA, the sub-pixel position of the star also affects the SNR in a pixel. If the centroid is positioned

in the center of the pixel, most star flux is accumulated in a few pixels with a high SNR. 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. But due to movement, the centroid may move to neighbor pixels. This leads to variations in the pixel illumination and the apparent centroid position [14].

In [14], we divide input parameter $\mathcal{E}$ into 9 sub-areas, whereas each area corresponds to one equivalence class. In this paper, we join the corner areas, the vertical border areas, the horizontal border areas, and the center area of the pixel to one equivalence class each. The 4, equally sized equivalence classes are shown in Fig. 7. Areas with the same pattern belong to the same equivalence class.

Figure 6 depicts the mean value and standard deviation of the residuals for stars in the respective pixel area. The stars are located in the same class of parameter $\mathcal{P}$ as well as have a constant medium magnitude and a worst-case non-Gaussian PSF. The figure shows that the residual noise is higher for stars positioned in a pixel corner than in the pixel center. The residual noise of stars in the horizontal border classes or vertical border classes is lower than the residual noise in the corner classes but higher compared to the center class. It is therefore beneficial to join the equivalence classes of input parameter $\mathcal{E}$ defined in [14].

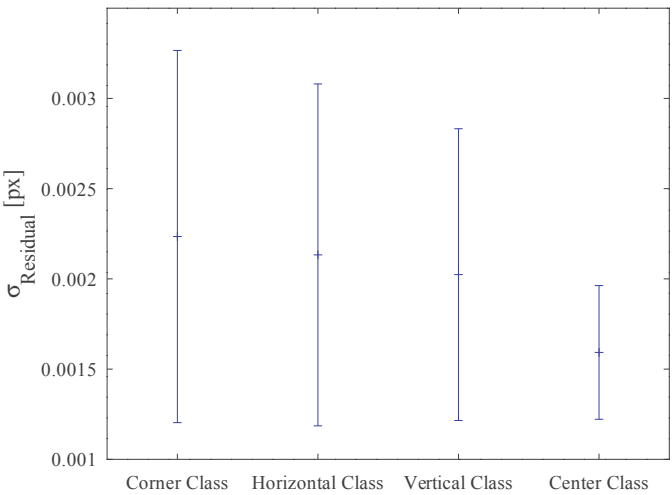

Fig. 6. Residual noise per pixel class.

The tester specifies the ratio r of the central area of the pixel to the pixel area, 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 Eq. (5).

$$b = a\sqrt{r} \tag{5}$$

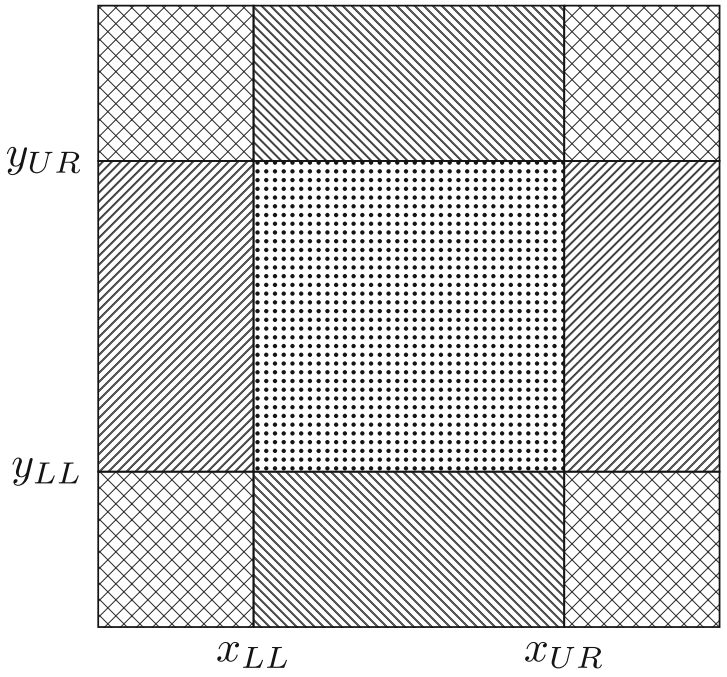

Fig. 7. Example borders of pixel equivalence classes.

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

$$l = (\frac{a}{2} - \frac{b}{2}, \frac{a}{2} - \frac{b}{2}) \text{ and } u = (\frac{a}{2} + \frac{b}{2}, \frac{a}{2} + \frac{b}{2}) \tag{6}$$

Based on these corners, we partition parameter $\mathcal{E}$ into equivalence classes E_i with $i = 0...3$. The equivalence class E_i is the i-th pixel sub-area. A star s lies in an equivalence class if it satisfies the corresponding condition.

$$\mathcal{E} = E_0 \cup E_1 \cup ... \cup E_4 \tag{7}$$

$$\begin{aligned}
&E_0 : (0 \ \leq e_x(s) < x_l \ \vee \ x_u \leq e_x(s) < a) \wedge \ (0 \ \leq e_y(s) < y_l \ \vee \ y_u \leq e_y(s) < a) \\
&E_1 : (0 \ \leq e_x(s) < x_l \ \vee \ x_l \leq e_x(s) < x_u) \wedge \ y_l \leq e_y(s) < y_u \\
&E_2 : x_l \leq e_x(s) < x_u \ \wedge \ (0 \ \leq e_y(s) < y_l \ \vee \ y_l \leq e_y(s) < y_u) \\
&E_3 : x_l \leq e_x(s) < x_u \ \wedge \ y_l \leq e_y(s) < y_u
\end{aligned} \tag{8}$$

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

Magnitude. The measured star flux (photo-electrons per second) depends on the magnitude. The accumulated number of photo-electrons per pixel denotes the

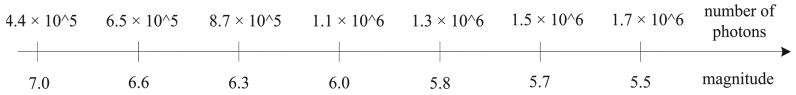

Fig. 8. Example partitioning of magnitude range [14].

illumination of a pixel. Equation (9) shows the relation between the magnitude m and the corresponding flux F_m in e^-/s.

$$F_m = F_0 TQA * 10^{-0.4*m} \tag{9}$$

with magnitude m, reference flux F_0 of a 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. A low magnitude corresponds to a high number of photo-electrons, that leads to a higher SNR per pixel.

A useful partitioning of magnitude values into equivalence classes is not obvious. Our idea is to partition the star flux range into $I_{\mathcal{M}} \in \mathbb{N}$ equidistant parts that represent the equivalence classes. We define Eq. (10) to obtain the upper limit of a sub-range.

$$F_{m_j} = F_{7.0} + j\frac{F_{5.5} - F_{7.0}}{I_{\mathcal{M}}} \tag{10}$$

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 $F_{7.0}$ correspond to the numbers of photons for magnitude 5.5 and 7.0. First, we calculate the flux values $F_{5.5}$ and $F_{7.0}$ by using Eq. (9). Then, we partition the flux range into equidistant sub-ranges. We use Eq. (11) to recalculate the magnitude m_j from the calculated flux limit F_{m_j} of the flux sub-range j.

$$m = -2.5\ log\left(\frac{F_m}{F_0 TQA}\right) \tag{11}$$

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

$$\mathcal{M} = M_{7.0} \cup ... \cup M_{l_j} \cup ... \cup M_{5.5} \tag{12}$$

with $l_j \in \mathbb{R}$ and $5.5 \leq l_j \leq 7.0$. Each equivalence class M_{l_j} is a magnitude sub-range with upper limit l_j. Each available star s lies in equivalence M_{l_j} if it satisfies the condition in Eq. (13).

$$l_{j-1} \leq m(s) < l_j \tag{13}$$

where $m(s)$ 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 8 illustrates an example partitioning of the magnitude range [14].

PSF Shape. The accuracy of the centroid calculation also depends on the PSF shape. In the best case scenario, the shape is a symmetric Gaussian-PSF. Then, the observation model (cf. Eq. (1)) perfectly fits the star. Therefore, the accuracy of the centroid calculation is high. In reality, the PSF shape is non-Gaussian. In that case, the observation model is less accurate and movements lead to stronger variations in the expected centroid positions [14].

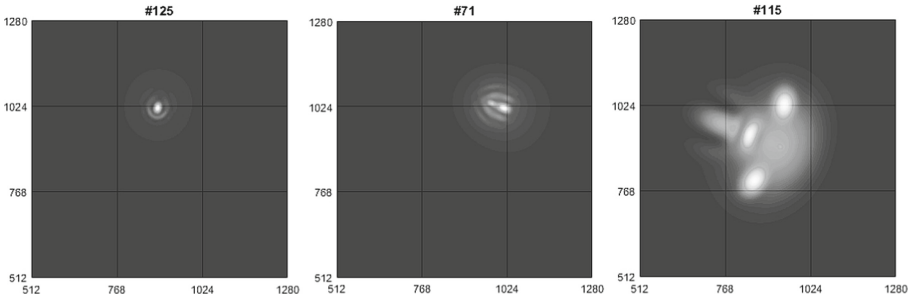

Fig. 9. Examples of different low quality stars [14].

We partition the input parameter $\mathcal{G}$ in two equivalence classes G_G and G_{NG} since two PSF shapes are distinctive. If a star has a Gaussian-PSF shape it is in class G_G otherwise it is in class G_{NG}.

Figure 9 shows some example stars with non-Gaussian-PSF shape that are less suitable as guide stars. These stars lead to inaccurate estimation results. What the three stars have in common is that their intensity is concentrated on a pixel edge. For all stars, the magnitude and FPA position are sufficiently good. However, a small variation due to movement leads to big changes of the illumination. Since the Gaussian-PSF observation model does not fit the PSF shape perfectly, the centroid estimation is less accurate.

4.3 Multidimensional Coverage Criteria

This section presents our definition of multidimensional coverage criteria on the input domain $I = \{\mathcal{P}, \mathcal{M}, \mathcal{E}, \mathcal{G}\}$ [14]. While individual parameter values might provide a good centroid estimation, a combination of parameters may change the quality of the results. To measure the coverage of a test suite with respect to input parameter combinations we define multidimensional coverage criteria on the input domain. If the measured coverage of a test suite is not complete, our automated test generation algorithm automatically inserts test cases for missing combinations.

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. Equations (14)–(17) show this definition for the input parameters $\mathcal{P}, \mathcal{M}$, $\mathcal{E}$ and $\mathcal{G}$.

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

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

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

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

The Cartesian product of the equivalence classes of the input parameters $\mathcal{P}$, $\mathcal{M}$, $\mathcal{E}$ and $\mathcal{G}$ is the coverage domain for our multidimensional coverage criteria. Hence, 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 $G_l \in \mathcal{G}$ [14]. Furthermore, a test case is a star represented by a tuple of parameter values $((p,t), m, e, g) \in (P_i, M_j, E_k, G_l)$. The following example test cases clarify these definitions.

Example 1

$$\big((1969.4, 322.5), 6.5, (0.3, 0.2), G\big) \in (P_{(2687,360)} \times M_{6.6} \times E_2 \times G_G)$$

The test star position is in the FPA area with outer radius 2687 and outer polar angle 225°. The star belongs to equivalence class $M_{6.6}$ because its magnitude value is between 6.3 and 6.6. The star center is located in the lower-middle pixel sub-area. That corresponds to the horizontal pixel areas and therefore to equivalence class E_2. The star is part of equivalence class G_G, because it has a Gaussian-PSF shape.

Example 2

$$\big((2551.9, 357.3), 6.5, (0.9, 0.8), G\big) \in (P_{(2687,360)} \times M_{6.6} \times E_0 \times G_G)$$

The test star is similar to the star in the first example, but it is positioned nearby the upper right pixel border and therefore belongs to equivalence class E_0.

Our multidimensional coverage criterion is fully 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 that completely covers the input domain with respect to our multidimensional coverage criteria as a complete test suite. The multidimensional coverage C results from the ratio of input combinations covered by at least one test case to the total number of input combinations.

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

Our test approach calculates the individual and multidimensional coverage of a given test suite using Algorithm 1. 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 [14].

For each test case in the given test suite, the algorithm computes the input parameter 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 indices 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. As 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 Eqs. (14)–(18) to compute the individual and multidimensional coverage.

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

1 $C_\mathcal{P} = C_\mathcal{M} = C_\mathcal{E} = C_\mathcal{G} = C = \emptyset$;
2 **foreach** *tc with ((p, t), m, e, g)* $\in$ *TS* **do**
3 i_P = getPosECId(p, t);
4 $C_\mathcal{P} \leftarrow C_\mathcal{P} \cup i_P$;
5 i_M = getMagECId(m);
6 $C_\mathcal{M} \leftarrow C_\mathcal{M} \cup i_M$;
7 i_E = getPixECId(e);
8 $C_\mathcal{E} \leftarrow C_\mathcal{E} \cup i_E$;
9 i_G = getModECId(g);
10 $C_\mathcal{G} \leftarrow C_\mathcal{G} \cup i_G$;
11 $C \leftarrow C \cup (i_P, i_M, i_E, i_G)$;
12 **end**
13 $Cov_\mathcal{G} = |C_\mathcal{P}|/I_\mathcal{P}$;
14 $Cov_\mathcal{M} = |C_\mathcal{M}|/I_\mathcal{M}$;
15 $Cov_\mathcal{E} = |C_\mathcal{E}|/I_\mathcal{E}$;
16 $Cov_\mathcal{G} = |C_\mathcal{G}|/I_\mathcal{G}$;
17 $Cov = |C|/(I_\mathcal{P} \cdot I_\mathcal{M} \cdot I_\mathcal{E} \cdot I_\mathcal{G})$

Algorithm 1. Coverage calculation [14].

Our partitioning approach uses individual and multidimensional coverage criteria to assess the quality of test suites with respect to their coverage on the input space of a satellite on-board image processing application [14].

4.4 Automated Test Generation

We present a test generation algorithm to automatically and systematically generate a test suite that completely covers the input domain according to our multidimensional coverage criteria. The complete test generation algorithm uses Algorithm 1 to assess a given test suite and systematically generates missing test cases based on this result.

Algorithm 2 generates set W that contains all input combinations not covered by the given test suite. For each input combination in W, the algorithm uses the procedure `generateTC` to generate a test case by randomly selecting values from the equivalence classes of the missing combinations. The algorithm adds the newly 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 we get a complete but reasonably small 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. Therefore, Algorithm 2 can be used to generate a new test suite that completely satisfies the multidimensional coverage criteria. From this test suite 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 [14].

Input: Input combination universe U, covered input combination set C, test suite TS
Output: Complete test suite TS

```
1 Cov = computeMultidimCoverage(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 2. Generate complete test suite [14].

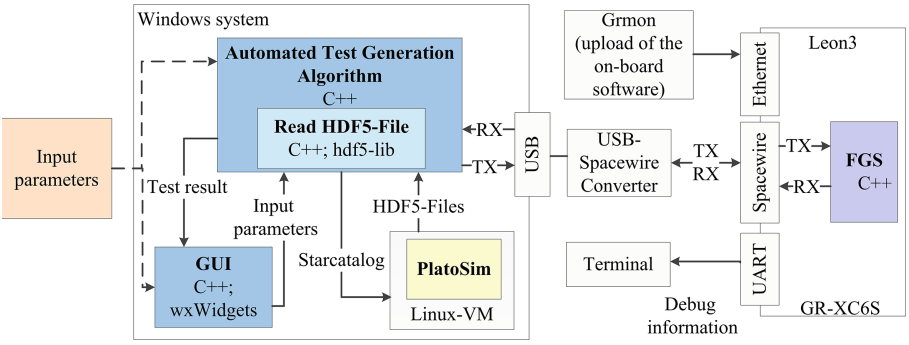

Fig. 10. Test setup [14].

5 Evaluation

We evaluate the applicability and error detection capability of our proposed test approach for satellite on-board image processing applications. For this purpose, we use the FGS algorithm of the ESA PLATO mission as case study and run the algorithms with various test suites.

5.1 Implementation

We have implemented the proposed partitioning and test generation algorithm in C++ based on the model-view-controller pattern. We allow the tester to specify input parameters with or without graphical user interface (GUI). Figure 10 shows the block diagram of our test setup. As the figure depicts, our test environment runs on a Windows system. Our automated test generation algorithm reads the star data of the test suite inserted by the tester. For missing test cases, our test generation algorithm returns a star catalog to simulate the star data. The catalog is a text file that includes right ascension, declination, and magnitude of stars that should be simulated. We manually insert the catalog into the PLATO simulator PlatoSim [9]. PlatoSim writes the simulated star data to HDF5 files [12]. Each file contains an image sequence of several time steps of a star in a hierarchical file format. Since PlatoSim is not developed for Windows systems, the simulator runs in a Linux virtual machine.

Figure 10 depicts, that we connect the Windows system via a SpaceWire USB brick to a GR-XC6S FPGA development board [10] running at 50 MHz. SpaceWire is a data-handling network for spacecraft defined in [4]. For that, our test environment uses the C SpaceWire USB API Library for the SpaceWire USB brick. A prototype of the FGS algorithm, written in C++, runs on the evaluation board. We load the software with the Leon debug monitor GRMON onto this board. Via a UART interface we receive debug information in a terminal. For example, stack size, hardware information, etc.

Our objective is to evaluate our approach for the development and test of the FGS algorithm implementation. Moreover, our goal is to test execution time and mathematical accuracy of the FGS algorithm under realistic conditions. For example, a calculation on the development board is slower than the same calculation on a Windows system. Therefore, we run the application under test on the target hardware and keep the test system in the software development cycle [14].

5.2 Experimental Results

In this section, we present the experimental results for generating a test suite using our redefined equivalence class definitions for testing the PLATO FGS algorithm.

Since the equivalence class borders used for the recent studies were been well chosen, as shown in Fig. 3, Fig. 4 and Fig. 6, we specify following start parameters for the experiment:

- Initial radius r_0 of FPA partitioning: 1900 pixel
- Initial polar angle θ_0 of FPA partitioning: 45°
- Number of magnitude sub-ranges: 6
- Ratio r of central sub-area to pixel area: 0.25

These start parameters lead to 48 equivalence classes of input parameter $\mathcal{P}$, 6 equivalence classes of parameter $\mathcal{M}$ and 4 equivalence classes of parameter $\mathcal{E}$. Input parameter $\mathcal{G}$ consists of two equivalence classes (G_G and G_{NG}). In the following we only consider equivalence classes of input parameter $\mathcal{P}$ that cover the image area of the CCDs. Therefore we use 36 classes of parameter $\mathcal{P}$. Thus, our automated testing framework needs 1728 test cases to completely cover the whole input domain of the FGS algorithm. That is twice the number of test cases used in [14]. The reason is that we refine the partitioning of input parameter $\mathcal{P}$.

To evaluate our approach, we investigate the error detection capability of a randomly generated test suite as well as a test suite generated by our algorithm. The randomly generated test suite contains 902 evenly distributed stars. The second test suite was generated by our automated test generation application using Algorithm 2 presented in Sect. 4.4. The test suite contains one test case for each input combination. Therefore it is complete with respect to our multi-dimensional coverage criteria and we call it complete test suite. Table 1 shows the coverage of the test suites for each input parameter as well as the achieved multidimensional coverage.

Table 1 shows that the utilization of the equivalence class partitioning method reduces the random test suite by hundreds of redundant test cases. Since there are no unnecessary executions of redundant test cases, the method increases the efficiency of the test process. The random test suite achieves a high individual

Table 1. Coverage values of the test suites.

	Random	Complete
Test stars	902	1728
Covered input combinations	256	1728
$C_{\mathcal{P}}$ [%]	100.0	100.0
$C_{\mathcal{M}}$ [%]	16.7	100.0
$C_{\mathcal{E}}$ [%]	100.0	100.0
$C_{\mathcal{G}}$ [%]	100.0	100.0
Multidimensional coverage [%]	14.8	100.0

Table 2. Output for a sample test case.

i_G	i_P	i_M	i_E	$StarId$	$deviation_x$ [px]	$deviation_y$ [px]	$result$
1	33	1	4	1017	112734.8	1097.8	Error detected
0	11	2	0	892	1.3×10^{-5}	4.4×10^{-5}	Error detected

coverage of three input parameters. However, due to the low individual coverage of input parameter $\mathcal{M}$, the multidimensional coverage of the test suite is low. Furthermore, Table 1 exhibits that the complete test suite covers the whole input domain of the FGS algorithm.

To assess the partitioning approach, we have automatically inserted some errors into the PLATO FGS algorithm code. These injected errors 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 if the complete test suite achieves a higher error detection capability than the random test suite.

In each experiment, our test application sent 1000 packets, with one exposure each, per selected test star to the evaluation board running the FGS algorithm, and evaluated the calculated centroids. After that, our test application calculated the residual between the resulting centroid positions and a given position calculated by PlatoSim for each exposure. In this paper, we use another test criterion than in [14]: if the standard deviation of the residuals is greater than a reference value, the test detects the error. Table 2 shows the output for two test cases that detect a wrong assignment error. The high deviation between the standard deviation of the residuals and the reference values reveals an error in the centroid calculation.

During the experiments, we have injected three missing assignment errors, three wrong assignment errors, and three wrong condition errors. Table 3 summarizes the experimental results for both test suites.

Table 3 shows that both test suites do not reveal all injected errors with respect to the given test criterion. All test cases in the random test suite, as well as all test cases in the complete test suite, detects all missing assignment errors. In addition, both test suites detect two wrong assignment errors. But the percentage of test cases that detect this error is different. In the first case, 31% of the test cases in the complete test suite and 16% of the test cases in the random test suite detect the wrong assignment error. In the second case, 31% of the test cases in the complete test suite and 19% of the test cases in the random test suite detect the error. This means, some test cases are more capable to find errors than others. The percentage of error detecting test cases in the complete test suite is about 1.8 times higher than the percentage of error detecting test cases in the random test suite. Hence, the error detection capability of the complete test suite

Table 3. Test suites evaluation results.

	Random	Complete
Test cases	256	1728
Detected errors	5	5
Undetected errors	4	4
Error detection capability [%]	55.6	55.6

is higher. For randomly generated test cases it is not sure that it contains the necessary test cases to detect a special error. Furthermore, our test criterion only considers the centroid position not the other output parameters of the centroid calculation. These should also be taken into account in later experiments.

However, not all injected errors are suitable to investigate the error detection capability of a test suite because they lead to erroneous results for each input data. Thus, it is necessary to insert hard-to-find errors that only affects special input combinations. Otherwise one test cases is sufficient to detect an error.

Compared to the results presented in [14] the error detection capability of the complete test suite is increased. But also the random generated test suites reaches a higher error detection capability than presented in [14]. The reason for this is the improved test criterion.

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 higher than the capability of the random test suite. The complete test suite includes more test cases that detect special errors than the random test suite. Furthermore, the experiment shows that the new test criteria leads to a higher error detection capability for both test suites compared to the results presented in [14].

6 Conclusion

Due to the large number of input parameters of satellite on-board image processing applications and their combinations an enormous amount of test cases is possible. Hence, it is infeasible to capture the complete input domain and execute test cases exhaustively. We have developed a test approach for this specific domain that automatically and systematically generates test suites that completely covers the input domain. Our approach is based on the well-known equivalence class partition testing method.

In this paper, we have redefined some of the equivalence class definitions presented in [14]. To assess test suites with respect to its coverage of the input domain, we have also presented individual coverage criteria for each input parameter as well as multidimensional coverage criteria to measure the number of covered input parameter combinations. Finally, we have specified an automated test generation algorithm that systematically generates missing test cases with respect to our multidimensional coverage criteria. As a result, our approach is able to fully automatically generate test suites that are reasonably small but complete with respect to our multidimensional coverage criteria. The tester specifies the size of our equivalence classes. This makes our approach adjustable to available test times and also to other image processing applications. Moreover, it allows to exclude an input parameter from the analysis by having only one class.

We have investigated the effectiveness of our proposed test approach on the FGS algorithm as mission critical component 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 test approach with redefined equivalence class borders, we have compared the error detection capability of a randomly generated test suite and the generated complete test suite as well as with the complete test suite given in [14]. The use of our equivalence classes of the input parameters reduces the number of redundant test cases in the randomly generated test suite by 71.6%.

During the experiments, we have successively injected 9 errors in the FGS algorithm code to investigate the error detection capability of both test suites. In this paper, we have specified an improved test criterion: a test case detects an error if the standard deviation of the residual between the calculated centroid position and a given position is bigger than a reference value. We have observed that the changed test criterion leads to different test results compared to the results presented in [14]. In general, the error detection capability of the complete test suite as well as of the random test suite has increased. Both test suites detects all missing assignment errors and two wrong assignment errors. However, percentage of error detecting test cases in the complete test suite is about 1.8 times higher than for the random test suite. Thus, the error detection capability of the complete test suite is higher than the error detection capability of the random test suite. But both test suites do not detect all injected errors because the test criterion considers only the centroid position and not the other output parameters of the centroid calculation.

The error detection capability of both test suites is nearly the same, because not all injected errors are suitable to show the error detection capability of a test suite. The injected errors lead to erroneous results for each input data. Therefore, it is necessary to inject hard-to-find errors into the code that only affects specific input combinations.

However, 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 automated generation, execution, and evaluation of test cases.

So far, we have injected errors in the application code. But in space, many missions suffer from cosmic radiation that flips bits in binary code or cause hot pixels in input images. We plan to investigate the efficiency of our approach by injecting errors in input data or in the binary code of the application in future work. Finally, we have evaluated our approach with a single application. Later on, we plan to investigate the flexibility of our approach for other applications, for example, blob feature extraction in the robotics domain [14].

References

1. Bhat, A., Quadri, S.: Equivalence class partitioning and boundary value analysis-a review. In: International Conference on on Computing for Sustainable Global Development (INDIACom), pp. 1557–1562. IEEE (2015)

2. Bringmann, E., Krämer, A.: Systematic testing of the continuous behavior of automotive systems. In: International Workshop on Software Engineering for Automotive Systems, pp. 13–20. ACM (2006)
3. DLR: Grünes Licht für europäisches Weltraumteleskop PLATO (2017). http://www.dlr.de/dlr/desktopdefault.aspx/tabid-10081/151_read-22858/#/gallery/27241
4. ECSS Executive Secretariat: Space engineering, spaceWire - links, nodes, routers and networks (2008)
5. ESA: ESA's 'Cosmic Vision' (2012). http://www.esa.int/Our_Activities/Space_Science/ESA_s_Cosmic_Vision
6. Grießbach, D.: Fine guidance system performance report. Technical report PLATO-DLR-PL-RP-0003, Deutsches Zentrum für Luft- und Raumfahrt (DLR) (2019)
7. Huang, W., Peleska, J.: Complete model-based equivalence class testing. Int. J. Softw. Tools Technol. Transf. **18**(3), 265–283 (2014). https://doi.org/10.1007/s10009-014-0356-8
8. Kaner, C.: Teaching domain testing: a status report. In: Conference on Software Engineering Education and Training, pp. 112–117. IEEE (2004)
9. Marcos-Arenal, P., et al.: The PLATO simulator: modelling of high-precision high-cadence space-based imaging. Astron. Astrophys. **566**, A92 (2014)
10. Pender electronic desiGN GmbH: Gr-xc6s-product_sheet (2011)
11. Liggesmeyer, P.: Software-Qualität: Testen, Analysieren und Verifizieren von Software, 2nd edn. Spektrum Akademischer Verlag, Heidelberg (2009)
12. The HDF Group: HDF5, 05 April 2018. https://portal.hdfgroup.org/display/HDF5/HDF5
13. Varshney, S., Mehrotra, M.: Automated software test data generation for data flow dependencies using genetic algorithm. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **4**(2), 472–479 (2014)
14. Witteck, U., Grießbach, D., Herber, P.: Test input partitioning for automated testing of satellite on-board image processing algorithms. In: Proceedings of the 14th International Conference on Software Technologies - Volume 1: ICSOFT, pp. 15–26. SciTePress, INSTICC (2019). https://doi.org/10.5220/0007807400150026