Fault Injection for Robustness Testing of Satellite On-Board Image Processing

Fehlerinjektion für Robustheitstests der integrierten Bildverarbeitung von Satelliten

Carlson M. Büth

WWU MÜNSTER

Münster, 2021

Bachelor's thesis

Fault Injection for Robustness Testing of Satellite On-Board Image Processing

Fehlerinjektion für Robustheitstests der integrierten Bildverarbeitung von Satelliten

Carlson M. Büth

Embedded Systems Group

Supervisor:	Ulrike Witteck, M. Sc.
First examiner:	Prof. Dr. Paula Herber
Second examiner:	Jun.-Prof. Dr. Benjamin Risse

Computer Science Department
University of Münster, Germany

Münster, July 13, 2021

Contents

1. Introduction

On-satellite image processing is subject to extremely high safety and accuracy standards, as error-free operation in hard real-time is mission critical in ensuring reliable telescope imagery. Satellite on-board software must be guaranteed to provide several years of continuous service and maintainability. The upcoming PLATO spacecraft (PLAnetary Transits and Oscillation of stars) is a medium-class mission of the European Space Agency (ESA) in collaboration with the German Aerospace Center (DLR) [ESA17]. The mission's purpose is to detect terrestrial exoplanets in the habitable zone of stars and characterize its properties, e.g., radius, mass, etc., at high resolution. PLATO FGS (Fine Guidance System) is an algorithm that provides the necessary attitude data with an accuracy of milliarcseconds from the image data. The input domain involved in the image processing application is enormous, so an automated test approach for PLATO based on equivalence classes was developed in order to find reduced test sets [WGH20b], which are essential for validating the FGS's robustness in determining the satellite's attitude.

The fine guidance system is crucial for PLATO, because its failure compromises the mission objective. For PLATO, the FGS is required to calculate accurate results and provide the attitude data in time. Therefore, they must undergo extensive testing on Earth to ensure their flawless functioning in orbit. The FGS has to accomplish its task despite cosmic rays and sensor degradation. Running tests early in the development phase is not possible without enormous expenses. The test procedure must therefore be conducted with no real image data of the actual space environment. Fortunately, there is the PLATO simulator PlatoSim, which simulates photometric time series of CCD images, also including the satellite's position in space [Mar+14]. The developed equivalence classes of [WGH20b] help to make the input range manageable and to test reliability and robustness thoroughly and systematically. However, the developed testing approach has only been evaluated with a few fault classes yet, all of which were injected into the FGS code. In particular, how well does the test approach perform if, for instance, cosmic rays or sensor degradation are simulated in the input images?

To overcome this problem, we propose a systematic fault injection approach in input data for on-board satellite image processing. Goal of this work is to evaluate the robustness of the FGS algorithm while using the test approach of [WGH20b]. We require our solution to meet the following criteria:

1. There should be a selection of domain-specific, realistic and relevant fault classes.

2. The injection should be automated for distinct classes of faults.
3. Determination of the FGS's robustness for each fault class and how well it performs in the presence of faults.

We propose fault classes based on recent literature about, e.g., hot and dead pixels, brighter-fatter effect, cosmic-ray events, and assess their relevance for the PLATO mission. For the selection of relevant fault classes, we have consulted with the PLATO development team at DLR and incorporated their expertise. After describing the origin and relevance of each fault, we model them to obtain a corresponding fault term. Our automated fault injection wraps PlatoSim and is able to inject in two ways: Faults already implemented in PlatoSim are configured before simulation and external faults are applied to the simulation results afterwards. We realized the implementation in such a way that adding further faults is simple and as non-invasive as possible. For the evaluation, we compare the accuracy and precision of the FGS by error class in multiple data representations. In order to analyze the equivalence classes of the test approach, we compare class wise accuracy and precision with various test suite coverages. Finally, we discuss the results of the fault injection and summarize the robustness of the FGS.

The key contributions of our work lie in systematically testing an FGS for distinct classes of input faults. On the one hand, there is prior work to make star trackers resistant to single event upsets through filters and redundancy. This is beneficial and happens quickly because errors are detected and corrected on hardware level, but a detailed analysis on fault classes is missing. The authors deliberately omit subsequent processing of the images, making the benefit uncertain [ARM17; Ara+19; ARM20]. A much more detailed analysis was done for DLR's OSIRIS project, but it dealt with the complete communication and not the guidance system [Mis20]. Yet another difference is that these studies refer to errors in FPGAs, but not in the image sensor. Bugs have already been injected for PLATO, but these were only code errors and no radiation effects [WGH19]. An analysis of the detailed input faults was already performed in 2001 at the Jet Propulsion Laboratory, which also considered the impact on star trackers [Han+01]. Independent fault class analysis provides domain-specific insight into which faults must be minimized to increase mission safety.

The rest of this thesis is structured as follows. First we present the background and the theoretical basis of the work. The PLATO mission and the FGS, which we use for the case study, are presented in more detail. The concepts of equivalence class testing, robustness testing and fault injection are then summarized. In Chapter 3, we start with the literature review and introduce the selected fault classes in the following. Subsequently, we present the implementation and explain how additional fault classes can be implemented. In Chapter 4, we evaluate the robustness of the FGS using the accuracy as our criterion. Finally, we summarize our results in Chapter 5.

2. Preliminaries

First, we will outline the context of our case study PLATO and the associated Fine Guidance System as well as fundamentals for this work. Second, we present the equivalence class test on which this thesis will build. Lastly, this section briefly reviews the basic idea and purpose of fault injection, as well as its core principles.

2.1. PLATO Mission and the Fine Guidance System

In 2026, the European Space Agency (ESA) will launch a spacecraft for the international mission PLAnetary Transits and Oscillation of stars (PLATO). The proposal was selected to be funded through the Cosmic Vision 2015–2025 program. Mission goal is to detect habitable, terrestrial exoplanets in immediate vicinity of solar-type stars. This characterization requires the achievement of various sub-objectives. All of them are the determination of physical parameters at high precision. The German Aerospace Center (DLR) is leading the international consortium for PLATO and is developing the software for the fast data processing unit (FDPU).

Achieving such high precision requires a unique satellite payload. For the study it is necessary to observe stars of planetary systems with low noise ratio over long periods of time. The orbiter consists of 104 CCDs (charge-coupled device) with a total resolution of about 2.03 Gpx, the largest combined resolution a satellite has had to date. Four CCDs compose a camera. The

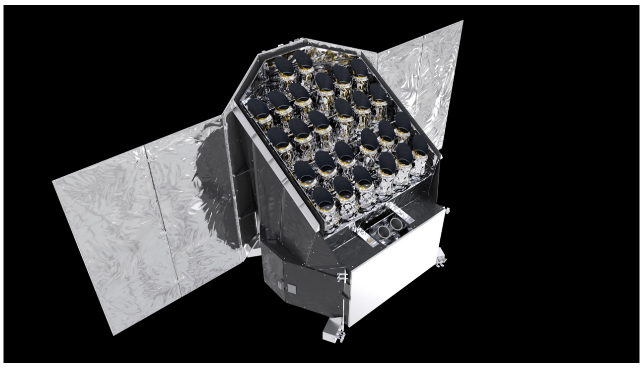


Figure 2.1.: Artist's impression of PLATO [ESA19].

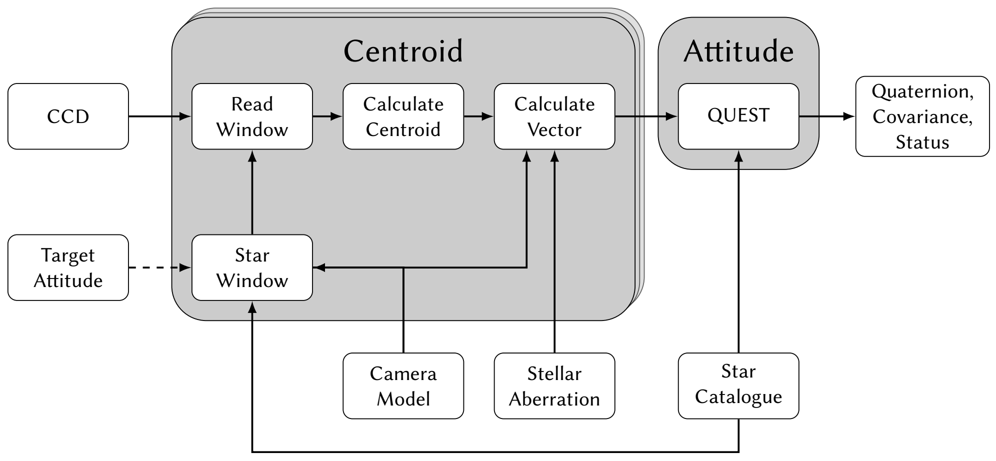


Figure 2.2.: FGS algorithm overview [Gri20].

majority of the cameras (24 of them) are responsible for long exposures of the stars. These are called normal cameras and have an exposure time of 25 s. Two so called fast cameras, with an exposure time of 2.5 s, are mainly responsible for attitude determination. A rendering of the provisional satellite model can be seen in Fig. 2.1. Using the Fine Guidance System (FGS), image data from the fast cameras is processed into attitude data and delivered to the spacecraft's Attitude and Orbit Control System (AOCS) at a resolution of milliarcseconds. This is the crux of the matter for precision, so this component is considered particularly mission-critical. Should the FGS not meet its accuracy and timing requirements, PLATOs mission objectives cannot be achieved [ESA17; Wit18].

The FGS—overview in Fig. 2.2—is another level of attitude control, not included in every ordinary spacecraft and intended for higher accuracy. Aim of the algorithm is to provide high-resolution attitude data so that PLATO can subsequently confirm and, if necessary, adjust its orientation.

As can be seen in Fig. 2.2, it receives a target attitude, the spatial orientation in which the fast cameras ought to be. The star catalog contains information about the position and magnitude of the guide stars. These stars are used to determine the attitude of the telescope by comparing their reference directions in the star catalog with the calculated positions on the CCD. Based on the star catalog and the target attitude, the FGS algorithm calculates for each star a window on the CCD which is read out separately as imagette. Thus, each imagette contains one respective guide star. For each imagette, a two-dimensional Gaussian distribution is fitted using a nonlinear least squares method to iteratively refine the model parameters called centroid. The centroid's position is the predicted position of the star in the imagette. With the camera model and stellar aberration in mind, the direction of the star as seen by the fast camera, called star vector, is calculated. From all calctualed guide star directions and the corresponding reference directions in the star catalog, an attitude quaternion is calculated. The QUaternion ESTimator (QUEST) algorithm solves this problem efficiently and iteratively [Shu78]. Accompanying this is the computation of the error covariance matrix of the measurement and a status for the validity of the solution [Gri20].

Another difficulty to be considered is the feasibility of testing. In development, before the satellite is on-orbit, it is not possible to test on real stellar image data from PLATO itself. Various aspects such as heat resistance can be simulated in cyro and heat chambers, but the complex interaction of FGS and AOCS requires digital simulation. For this case and other image processing systems, a realistic simulator for PLATO called PlatoSim has been developed. PlatoSim simulates "time series of CCD images [...] including [...] all important natural sources of noise." [Mar+14]

2.2. Equivalence Class Testing

Process times on satellites are tight for scientific missions. A criterion is needed to evaluate if the FGS is running correctly. Due to the enormous input domain, it is not possible to execute all test cases. Also, it is not sufficient to select test cases by hand, because the relevance of individual test cases is not obvious and there are too many candidates to consider.

Equivalence class partitioning (ECP) is an appropriate strategy in this situation, leveraging the principle of "divide and conquer". Complexity of the test problem is reduced by continued decomposition of the input space. The test selection simplifies to one case per class, which represents all possible test cases of the class. A reduction of the number of test cases is achieved. Equivalence class partitioning is well suited for non-state based software and continuous input domains [Lig09, Sec. 2.2].

The approach presented in [WGH20b] applies ECP in the context of the PLATO FGS. Advantageously, tests are generated automatically, while ensuring coverage of all classes with respect to a predefined multidimensional coverage criterion. This step allows to have a test set capable of testing the FGS with adequate runtime. The approach works by parameterizing the input domain and partitioning it into equivalence classes. A test has to consider the multidimensional coverage criterion and can cut down time for all redundant tests. Figure 2.3 shows an overview of the approach. For each selected test case in the test suite, PlatoSim simulates a series of imagettes that are input to the FGS. Subsequently, the test framework automatically evaluates the FGS results based on a predefined test criterion.

In addition, in [WGH20a] the approach is extended by a genetic algorithm using the partitioned search space. This additionally searches for a set of mission critical test cases, increasing robustness and by that mission safety.

One drawback, however, is that the ability of the testing approach to detect errors has not yet been systematically evaluated. Errors in the source code such as missing assignment, wrong assignment and wrong condition have been

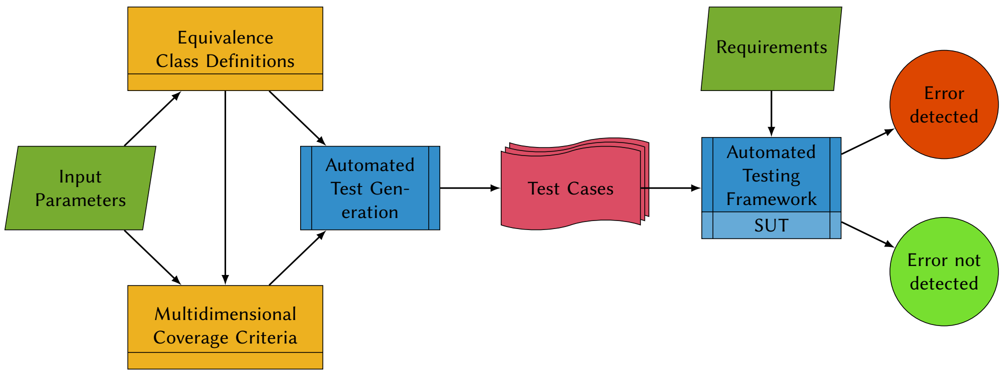

Equivalence Class Definitions
Input Parameters
Multidimensional Coverage Criteria
Automated Test Generation
Test Cases
Requirements
Automated Testing Framework
SUT
Error detected
Error not detected

Figure 2.3.: Overview of the partitioning approach [WGH20b].

tested. To ensure the FGS's robustness and to determine the error capability, the equivalence class test must be evaluated against errors in the image data and the behavior must be analyzed accordingly.

2.3. Robustness and Fault Injection

For many systems, in particular for safety systems, robustness is an important requirement. The aim is to meet safety-related requirements without interruption. Robustness is the trait of software to perform "acceptable" in spite of unwanted conditions. For example, electronic devices should continue to work correctly after a power outage. Depending on the application, different resuming actions may be appropriate [FMP05]. In the IEEE Standard Glossary of Software Engineering Terminology it is defined as "The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions." [IEE90]

For dependability of computer systems, Fault injection (FI) is an important evaluation method. The technique studies faults and failures by intentionally introducing them into source code or data [HTI97]. FI is useful for many cases, e.g., robustness testing or determining error detection capability. The behavior of the impaired system can be observed and, if desired, optimized as given. Doing so ensures that a system meets the requirements even under the influence of faults. This is done in order to increase error coverage while also increasing robustness.

> "The computer system has to provide the expected service despite the presence of faults." [CCS99, p. 50]

FI has been practiced since the 1980s and has been applied to various types of systems over the years. A distinction is made between hardware-implemented (HIFI) and software-implemented (SWIFI) approaches. HIFI works by introducing errors into the system from the environment, e.g., a manipulated transmission channel. SWIFI emulates the errors directly in the software and is part of the software itself, including for instance incorrect initializations or corrupted memory cells. Often this method is implemented on the level of hardware description languages like the VHSIC Hardware Description Language (VHDL) to guarantee a correct behavior. To reach a use it needs a way to communicate an error (failure mode) [Mor+19].

3. Fault Injection Approach for Satellite On-Board Image Processing

To recapitulate: The goal of our work is to develop an FI approach for satellite on-board image processing. With that we want to determine the robustness of the FGS algorithm while using the test approach of [WGH20b]. For this purpose, we choose FI and observe the behavior under controlled error influence.

Our approach is sketched in Fig. 3.1 and builds on the testing procedure of Fig. 2.3. For clarity reasons, the imagette generation is explicitly shown. In order to make an injection, we need to gather relevant fault classes and how they can be modeled. For a fault that is already part of PlatoSim, the corresponding configuration must be passed automatically. The strength of a fault is defined in the error variances. For a fault not implemented in PlatoSim, its injection must be done directly into the output imagettes. This is represented by the two arrows pointing from the fault injection. Our task is to integrate and automate the fault injection into the imagette generation process.

In Section 3.1, we present the studied literature from which the errors were taken and or used to justify their application relevance and importance. Next, we introduce the selection of the analyzed fault classes, give their modeling, and outline their implementation in Section 3.2. In Section 3.3, we describe the automation approach employed and its implementation. Finally, in Chapter 4, we evaluate the impact of the fault classes on the test's fault capability.

3.1. Satellite Image Faults: a Literature Review

For selection of fault classes, we did a systematic literature review. For the selection of fault classes, we did a systematic literature review. The procedure was to search for literature that is preferably up-to-date and/or comprehensive. We included sources that focus in detail on individual types of image processing errors. In addition, we also looked for faults that are unique to the aerospace context. With their reference to the CCD sensor architecture, we ensure that the image-related problems are also applicable to our case study.

Mainly, we refer to the two most comprehensive sources. One is a theoretical-experimental paper published by the well-known Jet Propulsion Laboratory (JPL) and California Institute of Technology (Caltech). They have analyzed

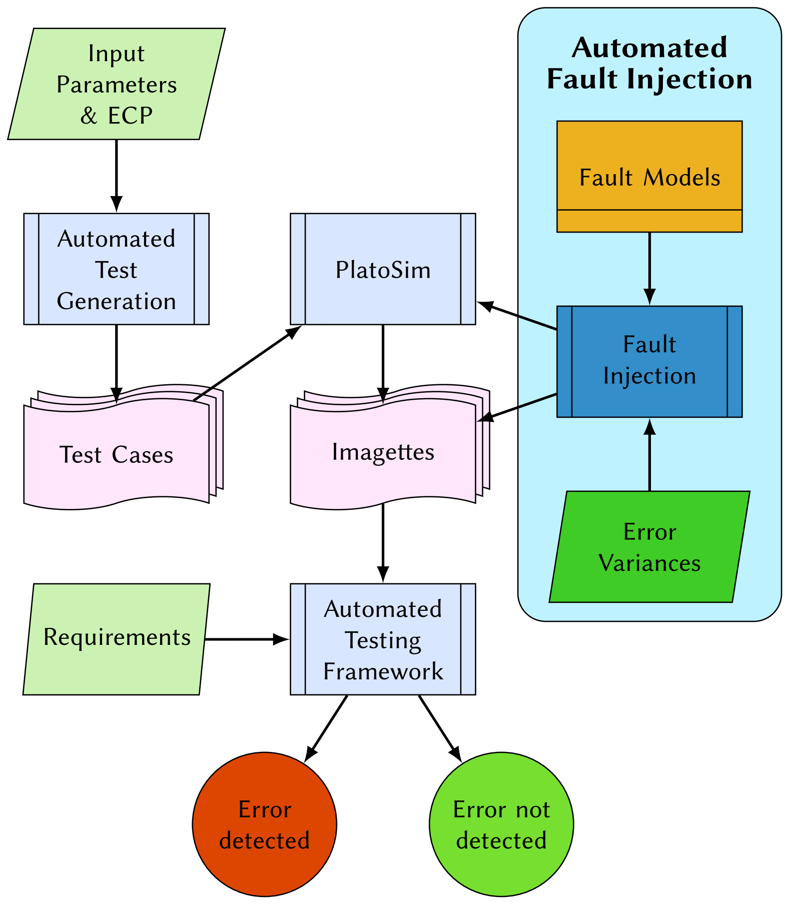


Figure 3.1.: Our approach to error injection intervenes on the equivalence class test, extending the testing procedure developed by [WGH20b].

the effects of several types of faults on star trackers and have summarized their theoretical and experimental efforts in the development of environmentally robust imagers for star tracker. Identical to the FGS, they fitted a two-dimensional Gaussian to determine the stellar center, but using a 5 px × 5 px imagette instead of 6 px × 6 px. They have modeled errors, such as photoresponse non-uniformity and photon shot noise, mathematically and then, in contrast to our work, analytically determined standard deviations depending on the integration time t_{int} [Han+01].

The other main source is a dissertation on image processing robustness written at the Polytechnic University of Turin (POLITO). The main focus of their work lies on techniques for implementing efficient and robust real-time embedded image processing hardware accelerators. In Section 2.3 *(Imaging sensors and related issues)* the sensor architectures CMOS and CCD are described and thus fault classes are motivated. These include de-focus blur, dark current, and others [Tro16].

Further sources are presented and discussed in the corresponding sections. In general, it should be noted that the behavior of CMOS cameras can also be applied to CCD, since their structure is quite similar. In the majority of publications about CMOS sensors this is especially mentioned.

3.2. Selected Fault Classes

In the following subsections, we present our fault classes. In each case, we answer the following questions:

1. How do they arise?
2. Why could they be relevant?
3. How do we model them?
4. Are they already included in PlatoSim?

Before we discuss the fault classes, we lay out how noise effects can be modeled in general.

As defined in [Han+01, Sec. 2.3], we assume that the measured signal $U(i, j)$ is composed of two components, namely the "real" signal I_{ij} and a error term ε_{ij}, where i and j are the discrete coordinates of a corresponding pixel for an image.

$$U(i, j) = I_{ij} + \varepsilon_{ij} \tag{3.1}$$

Obviously, for an exposure we will never be able to distinguish these two components, however, in a simulation it helps to model them independently. For all our fault classes we just need to justify what ε_{ij} looks like.

For PlatoSim, the modeling is referenced by and further specified in [Rei18].

3.2.1. Readout Noise

Readout noise is the simplest class and arises from the read electronics' noise (in general called Front-End Electronics, FEE). It is assumed to be equally distributed in time and uncorrelated, since each pixel is read out separately. As the background is subtracted, its mean is zero. The larger the exposure time, the smaller the readout noise in relation to the real signal. However we are working with a fixed exposure time. Considering that each pixel is affected in a random manner, we want to check the impairment of the test approach.

PlatoSim models this with a normal distribution $\mathcal{N}$ around mean 0 where for every pixel a sample is taken.

$$\varepsilon_{ij} = \mathcal{N}\left(0, \sigma_{\mathrm{RN}}\right) \tag{3.2}$$

The parameter σ_{RN} is the variance of the readout noise [Han+01; Tro16].

3.2.2. Hot/Dead Pixels

From overheating or cosmic radiation like neutrons, hot and dead pixels can arise. Also a plain defect in the image sensor can result in such fault. Hot pixels are pixels which have an illumination independent component that increases their output. This can result in a pixel value near or as high as full saturation. The exposure of the dead pixel deviates in the opposite direction, resulting in a pixel output that is well below the average exposure of the sensor.

Both faults are relevant in this case, because a hot pixel might be brighter than the observed star. Also a dead pixel distorts the stars illumination, especially when it strikes the bright center of the star. We expect the two cases to cause a differing centroid.

This fault is missing from PlatoSim. For this fault, the pixel at (k, l) is set to a given intensity I_{HDP}. This intensity should be chosen at the limits of the dynamic range [Cha+15; Wan+12].

$$\varepsilon_{ij} = \begin{cases} I_{\mathrm{HDP}} - I_{ij} & \text{if } (i,j) = (k,l) \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

3.2.3. De-Focus Blur

Perhaps the most recognizable fault is de-focus blur. This results when the focal plane and the image sensor do not match in position. The light diverges and a blurred image appears. Even if a pure blur should not affect the centroid position, there could be unknown effects.

The error term of the effect is the difference the PSF function causes compared to before the mapping

$$\varepsilon_{ij} = \mathrm{PSF}\left(I_{ij}\right) - I_{ij}. \tag{3.4}$$

PlatoSim simulates the optics. Therefore, a PSF can be selected to map the de-focus as desired [Tro16].

3.2.4. Brighter-Fatter Effect (BFE)

For the same star time the observed shape changes at a longer exposure. The common assumption is that each pixel functions independently. However, reality shows that with higher luminosity, the covariance between neighboring pixels increases. This effect arises from the electric fields emanating from the charges collected in the CCD. The longer a pixel is illuminated, the more repulsive it is to further intensity, which is then more likely to be picked up by neighbors [Cou+18]. This error is particularly relevant because the shape of the star is distorted in a non-trivial way. Consequently, this is transferred to the centroid and its position.

For the implementation, the authors of PlatoSim use the pertubated change in the pixel through the model proposed by [Guy+15, Eq. (11)]. The change is influenced by all directly neighboring pixels X as [Rei18]

$$\varepsilon_{ij} = \frac{1}{4} \sum_{X} \sum_{k,l} a_{kl}^{X} \cdot I_{kl} \cdot (I_{ij} + I_X) \,. \tag{3.5}$$

The second sum repeatedly covers a window of adjacent pixels (k, l). For computability reasons they are included only up to a fixed distance $R \leq d\big((i,j),(k,l)\big)$. With increasing distance the contributed terms shrink and can therefore be neglected. The coefficients a_{kl}^{X} can be determined from the physical problem. For further reading we recommend both [Cou+18; Guy+15].

3.2.5. Dark Current

Inherently, the CCD has an electric field in order to function. This field is present even when there is no external signal. Due to this influence, a fixed pattern noise is generated. Another name for this is thermal noise, because heat is induced by the electric field. A crucial difference to the readout noise is that the expected value is not zero. This could further influence the Gaussian fit for the centroid.

Because this is a fixed pattern noise a dark signal is sampled by PlatoSim's authors as

$$I_{ij,\text{ dark}} = \mathcal{N}\left(\mu = n_{DS} \cdot t, \sigma = n_{DS} \cdot t \cdot f_{\text{DSNU}}\right),$$

where t is the exposure time and n_{DS} the dark signal rate. From this, the error term originates with

$$\varepsilon_{ij} = \mathcal{N}\left(\mu = I_{ij,\text{ dark}}, \sigma = \sqrt{I_{ij,\text{ dark}}}\right). \tag{3.6}$$

The central parameter here, in addition to the dark signal rate n_{DS}, is the deviation $f_{\text{DSNU}} \in [0,1] \subset \mathbb{R}$ [Han+01; Tro16]. PlatoSim implements it exactly this way.

3.2.6. Galactic Cosmic Ray (GCR)

Cosmics are the event where radiation penetrates the image sensor and deposits energy along the particle's path. Energy can be in the form of electrons, which are counted in the CCD. This behavior is exhibited by cosmic protons and helium ions. These events are possible over a wide energy spectrum, so on the CCD a GCR might be brighter than a star [Bru+15]. We assume that an intense GCR near a star can impair the centroid determination.

PlatoSim includes GCRs, but distributes them across the whole CCD imager. For us it is relevant that we can decide whether a GCR should be in an imagette or not. Therefore, we implement this fault ourselves, but described as in PlatoSim [Mar+14]. In addition, we extend the functionality with a parameter that determines whether a GCR must hit through the center of the image or not.

A GCR is characterized by the sub-pixel entry point $\vec{x} \in \mathbb{R}^2$, entry angle $\alpha \in [0, 2\pi[$, trail length $l \in \mathbb{R}^+$, and intensity $I_{\mathrm{GCR}} \in \mathbb{R}^+$. They will all be sampled from uniform distributions over their possible values. $\vec{x}$ must be in the imagette and for the length and intensity PlatoSim's intervals are adapted. With the decay function

$$f(t) = \exp\left(\frac{-t^2}{2 \cdot \sigma^2}\right),$$

the intensity is distributed over the trail, with $t \in]0, l]$ and a normalization factor σ.

With this information, a line L is drawn by discretizing it into trace points which distribute a corresponding proportion of the intensity to the respective pixel on which they fall. This is added to the intensity, so this yields

$$\varepsilon_{ij} = L(\vec{x}, \alpha, l, I_{\mathrm{GCR}}). \tag{3.7}$$

3.3. Implementation with PLATO Simulator

We give a general description of the implementation. To automate the fault injection process, we chose to encapsulate the imagette generation. The test case generation is excluded. Instead, we start directly with a given test suite and the simulation settings. Figure 3.2 shows the fault injection process. We start directly with a star catalog and a simulation setting. The star catalog contains several stars that represent the test cases of a given test suite. There is one class per fault, inheriting either PlatoSim-Fault or External-Fault type. The fault settings can be used to specify which errors are injected and with which parameters they are called. Since PlatoSim is a separate program, PlatoSim-Faults can occur only once in the fault settings, but external ones can occur as often as desired.

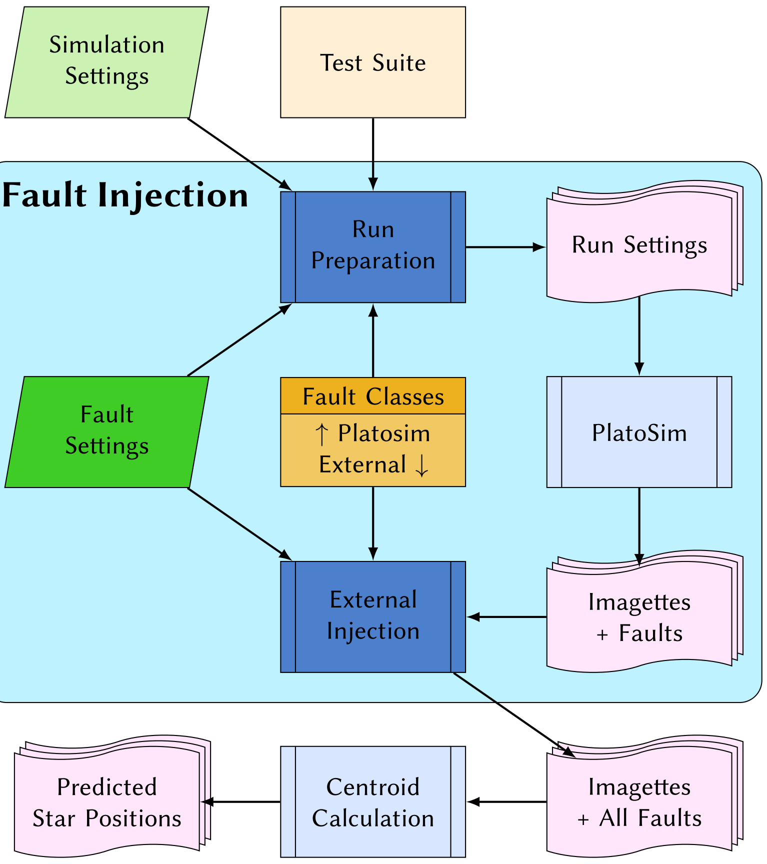


Figure 3.2.: Overview of the injection workflow.

The injection happens in two steps. In the first, a run setting is created for each star by taking the simulation settings into account and making appropriate settings per PlatoSim type fault. PlatoSim generates imagettes for each run setting, which in turn contains faults. The external injection then modifies the imagettes so that each desired external fault is also covered. Finally, the centoid calculation can be called to evaluate the star positions using the FGS.

Our fault injection code can be executed via a command line interface, with call parameters for application settings. PlatoSim is written in C++, but has a Python interface, so we chose to write the code in Python 3. We developed the fault injection in an internal GitLab repository of the DLR, so the code is not public.

Example fault classes

For a general overview, we show how faults inheriting from the two fault class types can look like.

Faults that are already implemented in PlatoSim need to inherit from `BaseFaultPlatosim` and have a class parameter named `classParameters`. This `dict` has to contain one entry per fault parameter. Each is later called by its key (e.g., `'CCDIncludeReadoutNoise'`) and the value (`dict`) is always built the same way. The `'type'` attribute specifies the parameter type and must be understood by Python as a type. Before the injection, it is checked if the passed variable is of the type. `'platosimPath'` points to the corresponding PlatoSim configuration parameter. `'assert'` has to be a lambda function that represents an assertion for the parameter. The assertion must always evaluate to a `bool` so that the program can check if the passed value is allowed before calling PlatoSim.

```python
class ReadNoise(BaseFaultPlatosim):
    """
    Readout noise due to imperfect amplifiers and ADC electronics.
    The measurement is on average correct, but deviated with a fixed
    variance. CCD and FEE (front end electronics) term are added in
    quadrature.
    """

    classParameters = {  # bool to turn CCD readout noise on of off
        'CCDIncludeReadoutNoise': {
            'type': bool,
            'platosimPath': "CCD/IncludeReadoutNoise",
            'assert': (lambda b: True)  # type check in class
        },
        'CCDMeanReadoutNoise': {  # float for CCD readout noise mean
            'type': float,
            'platosimPath': "CCD/ReadoutNoise",
            'assert': (lambda rn: rn >= 0)
        },
        'FEEMeanReadoutNoise': {  # float for FEE readout noise mean
            'type': float,
            'platosimPath': "FEE/ReadoutNoise",
            'assert': (lambda rn: rn >= 0)
        }
    }
```

External faults inherit from `BaseFaultExternal` and their `classParameters` is defined the same way, except without the `'platosimPath'` attribute. Important to notice is the `fault_field()` method. For each image, this function is called so that an individual fault can be generated for each imagette. It has to return an `edge_length` × `edge_length` matrix (`numpy.ndarray`) containing the fault. First, the class constructor (`__init__()`) is overridden and passed an argument describing the faults injection behavior. `'additive'` will add the `field` to the images, `'multiplicative'` will multiply them and `'absolute'` will assign our `field` masked by `mask` (also returned by `fault_field()`) to the image. The parameters for a class instance can be accessed as seen in line 26.

```python
class GalacticCosmicRay(BaseFaultExternal):
    """
    Cosmic rays leave trails in CCD imagers. This class generates
    intensity that will be added to the imagettes. HitCenter
    specifies whether the trail goes through the imagette center.
    """

    def __init__(self, *args, **kwargs):
        super(GalacticCosmicRay,
                self).__init__('additive', *args, **kwargs)

    classParameters = {
        'TrailLength': {
            # Interval for the allowed length of the cosmic trails,
            'type': list,  # expressed in pixels.
            'assert': (lambda l: len(l) == 2 and type(l[0]) == float
                                 and type(l[1]) == float and
                                 0 <= l[0] < l[1])
        },  # ...
    }

    def fault_field(self, edge_length):
        """Return cosmic ray of class 'through center' or not."""

        field = np.zeros((edge_length, edge_length), dtype='<u2')
        trail_length =
        ↪  self.configParameters['TrailLength']['setTo']['Variable']
        # ... some code constructing a GCR
        return field
```

4. Evaluation: Robustness and Accuracy

With the implemented approach we evaluate the robustness of the FGS and analyze how well the testing approach presented in [WGH20b] performs. To observe the robustness of the FGS in terms of accuracy and precision, we study the performance of the FGS's centroid determination under different settings. For this, we use equivalence class based tests along with our error injection approach.

To visualize the results of the FGS, we plot the Accuracy and Precision in the following sections. Accuracy is given as residual position error Δd and precision as its standard deviation $\mathrm{SD}(\Delta d)$. If the FGS is robust under the influence of faults, accuracy and precision should remain as low as possible. As long as both indicators do not grow with increasing error expression, the FGS is also robust against it. If, on the other hand, we experience large Δd under a fault, the FGS is not robust against it.

4.1. Simulation Configuration

One of our goals was to confirm the effectiveness of the equivalence classes. For the coverage criterion, different coverages of the given test suites are calculated and compared via the automated evaluation. For the systematic analysis of the fault classes, we select a characteristic parameter for each class, e.g. readout noise variance. We change this parameter and execute the automated testing approach repeatedly.

We proceed by picking a characteristic parameter for each fault and scaling it. That is, we take the default PlatoSim setting and multiply the considered parameter by $0, 2^0, 2^1, 2^2, \cdots$. For each step we simulate imagettes with injected faults for test suites with coverages of 100 %, 90 %, 75 %, and 50 % (with respect to the coverage criterion presented in [WGH20b]). Our imagettes have a size of $6\,\mathrm{px} \times 6\,\mathrm{px}$. All test suites have only one representative test case per equivalence class combination. Therefore, the 50 % test suite has half as many test cases as the 100 % test suite. The reduced test suites contain representatives for randomly selected equivalence class combinations, but are independent of each other. The reduced test suites have randomly selected representatives, but are independent of the other test suites. From each simulation, we obtain the residual of the position per star. That is, the difference between expected position on the CCD sensor and that of the determined centroid. For greater significance, the residual per star is averaged over 1000 exposures. Per fault, the simulation for the four test suites takes between a half to a day. This uses a 64-bit Linux machine with 62 GiB of memory and ten cores of the Intel(R) Xeon(R) Gold 6126 CPU @ 2.60 GHz in parallel.

To detect outliers, the median absolute deviation (MAD) criterion is applied per configuration. For every star the distance to the median is computed, called Z-score. We classify as outliers the stars with a Z-score larger than 3.5, as proposed by Iglewicz and Hoaglin [IH93]. This criterion is stronger than a deviation from the arithmetic mean, since a few outliers strongly distort the mean if their residual is in larger orders of magnitude. For the implementation, we refer to [Kin14]. This criterion is applied to individual configurations in each case, since these provoke different FGS behaviors. Thus, for a case at 100 %, the threshold may be different from that at 90 %. However, this is taken into account in the comparison.

4.2. Evaluation of Fault Effects

To analyze the behaviors, we present the data in several ways. For the first fault type readout noise, we introduce our evaluation approach of the data representation and then use it on the following ones as well.

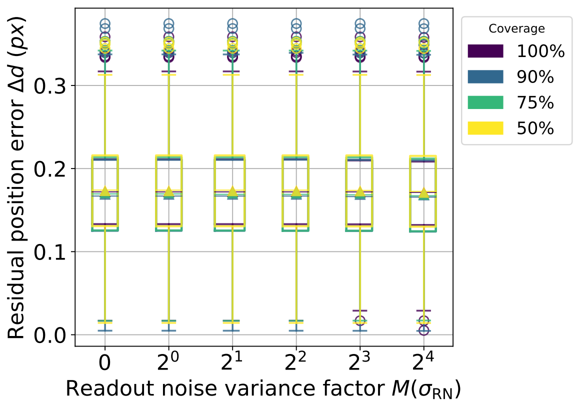
Readout Noise

As described in Section 3.2, there is a noise component due to the CCD itself and a noise component due to the FEE, which are summed in quadrature. Since for PLATO the CCD component is fixed at $44\,\mathrm{e}^-/\mathrm{pixel}$[1] and is difficult to optimize, only the FEE component is scaled. By default, the FEE component is at $200\,\mathrm{e}^-/\mathrm{pixel}$, and we scale it with $M(\sigma_{\mathrm{RN}})$.

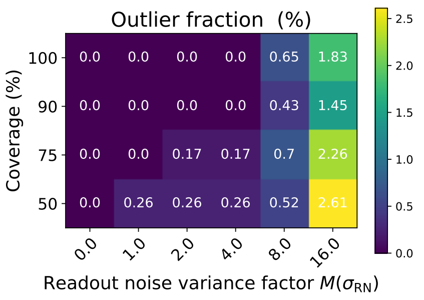
For readout noise, the data, without outliers, is plotted as boxplots in Fig. 4.1a. Along the abscissa the multipliers of the noise variance $M(\sigma_{\mathrm{RN}})$ are shown, which increases cubically. On the ordinate, the average residual per star Δd is plotted in pixels. For each factor $M(\sigma_{\mathrm{RN}})$ and coverage there is a boxplot.

With the first look at Fig. 4.1a no $M(\sigma_{\mathrm{RN}})$ correlation becomes clear, only that the test suites are not completely alike. Even the data points outside of the whiskers keep their structure. Median and arithmetic average lie around 0.175 px. Expected was a performance degradation with increasing $M(\sigma_{\mathrm{RN}})$, but even with $2^4 \cdot 200\,\mathrm{e}^-/\mathrm{pixel} = 3200\,\mathrm{e}^-/\mathrm{pixel}$, the centroid algorithm seems to work the same as without FEE noise. This speaks in favor of the position determination using the Gaussian method of the centroid algorithm. The FGS is robust to readout noise for the $M(\sigma_{\mathrm{RN}})$ considered.

[1] e^-: Charge of the electron.



(a) Boxplots for the centroids residual error dependent on the scaled **readout noise variance factor** $M(\sigma_{\mathrm{RN}})$. The four colors represent different test suite coverages.



(b) Outlier fractions broken down by test suites of different coverages and the scaled **readout noise variance factor** $M(\sigma_{\mathrm{RN}})$.

Figure 4.1.: Boxplots and outlier fractions for the readout noise fault.

We note that even with the largest noise here, the intensity of the stellar center in the imagette is still two orders of magnitude larger than the noise variance. A higher $M(\sigma_{\mathrm{RN}})$ was not configurable in PlatoSim, though such high FEE noise is not realistic either. With larger $M(\sigma_{\mathrm{RN}})$, an increase of δd should occur, thus the FGS should become less accurate.

The outliers, while undesirable for the main case, are just as important for the global analysis. We count the outliers detected via the MAD criterion and divide the number by the size of the respective test suite. This gives us the ratio of outliers, denoted outlier fraction. Plotted against coverage and readout noise variance factor $M(\sigma_{\mathrm{RN}})$ this is shown in Fig. 4.1b. From $M(\sigma_{\mathrm{RN}}) \in [0, 4]$ there are almost no outliers, at $M(\sigma_{\mathrm{RN}}) = 8$ outliers increase to more than 0.4 % for all coverages, and at 16 to more than 1.4 %. This indicates a decreasing performance with increasing $M(\sigma_{\mathrm{RN}})$, as anticipated. Contrary to intuition, the outlier fraction is largest at 50 %. However, since we have a single sample of test suites, we can only state that the outlier fraction of all test suites behaves similarly with increasing $M(\sigma_{\mathrm{RN}})$.

For comparison of the equivalence classes, we report the residual position error Δd per equivalence dimension in Fig. 4.2. Solely the simulation with most impaired Δd, where $M(\sigma_{\mathrm{RN}}) = 16$, is considered. Δd is summarized in the arithmetic average with the inter-star standard deviation. This value looks at the accuracy per test case (already averaged over 1000 exposures) and indicates how much the test cases vary. Stars that regularly show critical behavior increase this deviation. The colors encode the four test suite coverages. Leftmost, in the inner circle, in the radius class, the best performance is found for a radius of 0 px to 2353 px.

The next larger radius class produces the largest deviation Δd on average. Due to the distance to the optical axis, the optics generate an abberation,

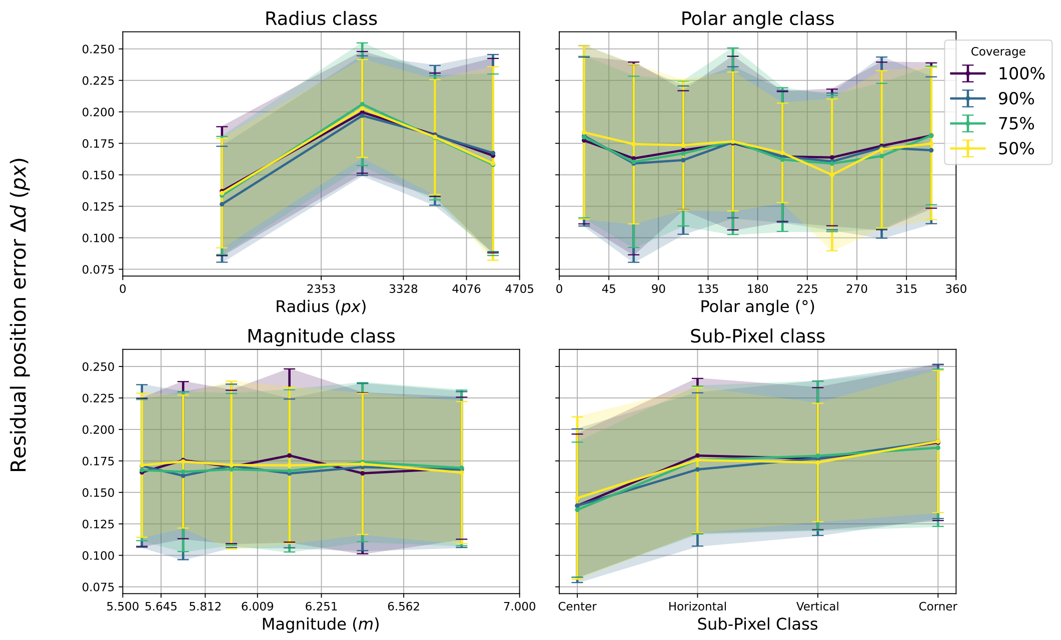


Figure 4.2.: Class-wise performance of different test suite coverages measured by the centroids residual error, for the **readout noise variance factor** $M(\sigma_{\mathrm{RN}}) = 16$.

which could lead to inaccuracies. The further radii have lower nominal values, but also increasing standard deviation. Thus, the unpredictability of the outer classes becomes clear. The behavior of the radius is not an inherent property of the readout noise and is also observable for the further faults.

As a final point, for this class, we see that the variance of the test suites is larger with higher coverage. The 50 % test suite is always enclosed by the others with one exception. This suggests that test suites with greater coverage provoke a wider range of behavior.

Angular class, top right in Fig. 4.2, exhibits wiggle, but all expected values overlap with the standard deviations of the rest. Readout noise does not produce radial dependence, as we expected. The same is true for the magnitude class. The smaller the apparent magnitude m, the brighter a star appears and the more intense it is seen in the imagette. Since the noise has not reached a significant level, all magnitudes represented here are equally unaffected.

The results for the sub-pixel classes are similarly distributed between the different test suite coverages. Stars that lie in the center of a pixel have the least deviation on average. Horizontal and vertical pixel sides show a larger and very similar deviation. Stars in the corner of a pixel show the poorest performance. This tendency of pixel classes is also common and we will encounter it repeatedly.

The final method of visualization is similar to the previous one. Only difference is that we consider the results from the test suite with 100 % coverage,

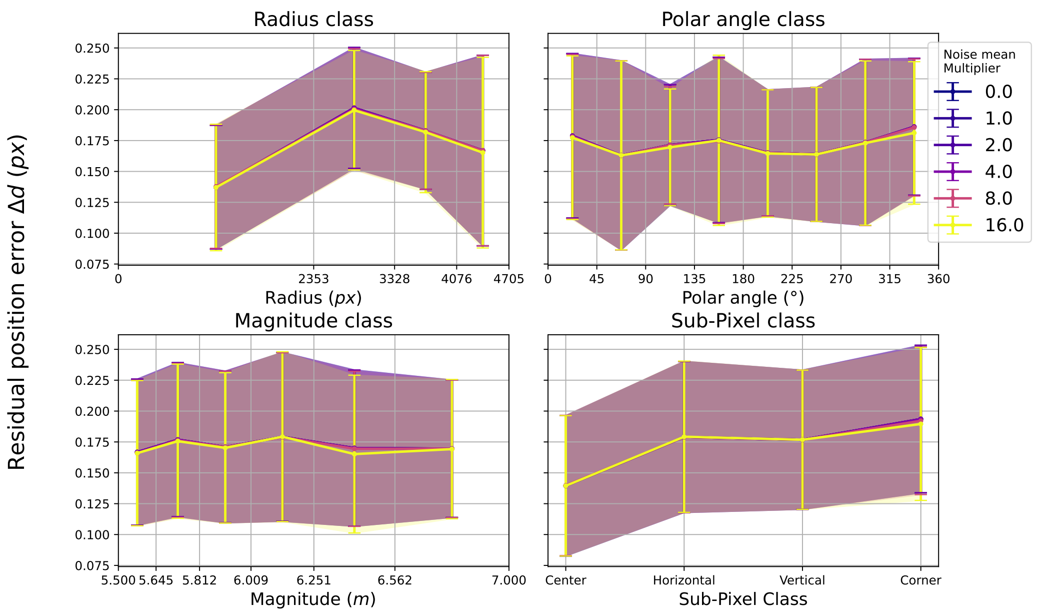
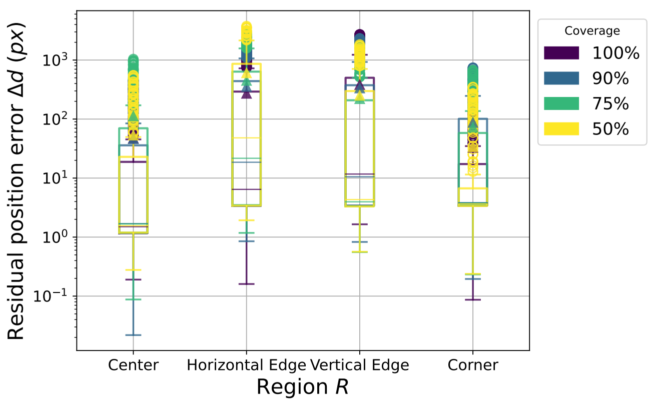


Figure 4.3.: Class-wise performance of various scaled **readout noise variance factors** $M(\sigma_{\mathrm{RN}})$ for a test suite coverage of 100 %.

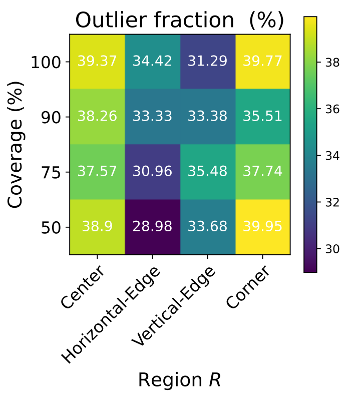
while we distinguish between the considered parameters $M(\sigma_{\mathrm{RN}})$. At the current fault (Fig. 4.3), this approach does not yet give us any special insight. Analogous to the boxplots for this fault (Fig. 4.1a), the dynamic range of $M(\sigma_{\mathrm{RN}})$ shows no differences within the classes. Therefore, the FGS is robust even across classes for readout noise.

For an additional level of confidence, we not only look at the accuracy, but also at the precision. In other words, we have drawn all four previous data representations one more time, but this time using the intra-star standard deviation. Which shows us how precisely the FGS determines star positions. A restricted standard deviation is an important quality specification for fine guidance systems in general. In order to not clutter this section, these plots are outsourced to Appendix A and are mentioned intermittently when considered relevant. However, we do not provide a complete, separate analysis of the precision of the FGS in this thesis.

For readout noise, the accuracy in Fig. A.1 increases steadily, even for stable accuracy. It is also noticeable that the precision decreases with increasing magnitude and dimmer stars, as Fig. A.2 shows. There we also see that the sub-pixel classes behave inversely to the nominal value. In the center class, where the nominal value shows the smallest deviation, the standard deviation is the largest. In the corner of a pixel, we see the smallest standard deviation SD(Δd). The coverages and outlier fractions do not behave differently from the nominal values.



(a) Boxplots for the centroids residual error dependent on the **imagette region with hot pixel** R. The four colors represent different test suite coverages. Here, Δd is plotted on a logarithmic scale.



(b) Outlier fractions broken down by test suites of different coverages and the **imagette region with hot pixel** R.

Figure 4.4.: Boxplots and outlier fractions for the hot pixel.

Hot/Dead Pixel

For this fault class, a distinction is made between hot and dead pixel. For both, a single pixel is modified. The parameterization is not done by a continuous factor, but by the position of occurrence. The imagette is divided in such away that (i, j) is in one of the 4 regions: center, horizontal edge, vertical edge and corner. It is partitioned so that one side is divided into three equal parts, if possible. If the edge length cannot be divided by an integer of 3, the middle part is enlarged.

First we consider a hot pixel with 80 % intensity, which according to [Cha+15] can occur for a hot pixel. In Fig. 4.4a we start with boxplots for hot pixel. All regions cause unacceptable results for the FGS, because only a few cases are below a deviation of 1px. The FGS is not robust against a hot pixel of this intensity, even in the center where the star is located. This is true except for a few cases in the center, where the hot pixel may have fallen exactly on the center of the star. There are some differences between the coverages. While they are similar per region, they do not have a clear trend in terms of coverage. One would like to have complete test suites so that one covers all relevant test cases. However, here we see that the 100 % test suite does not always cover most borderline cases, since otherwise its box would always have to span all others. From this we conclude that chance is involved in taking the test suites. The representative test case for a equivalence class does not always produce exactly the same behavior as all other possible test cases for that equivalence class.

Figure 4.4b again shows outliers. All outlier fractions are between 28 % to 40 % and show a similar structure over the different test suite coverages, where

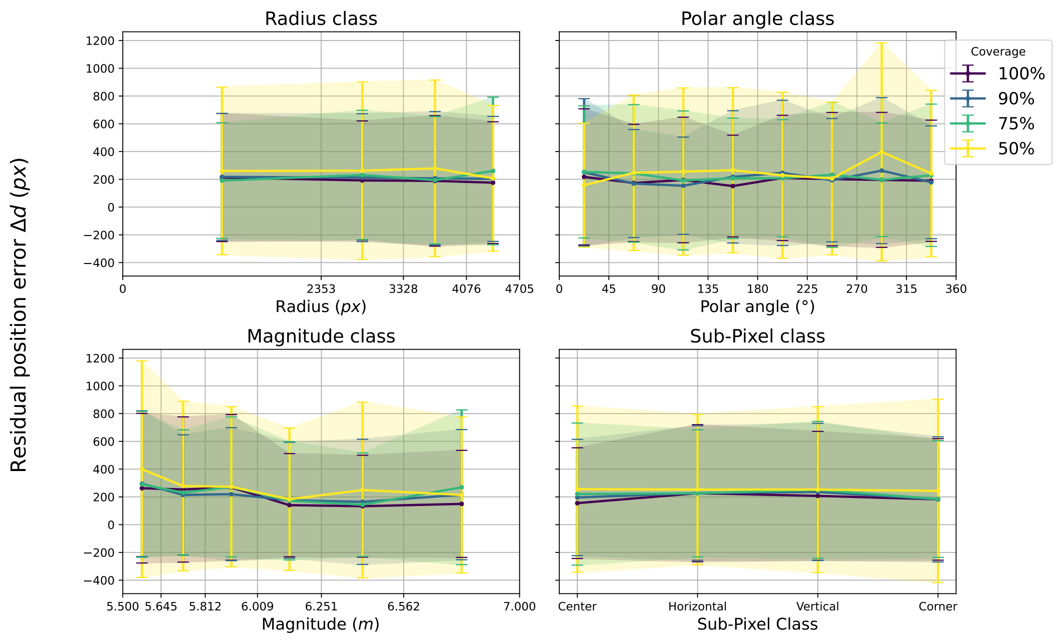


Figure 4.5.: Class-wise performance of different test suite coverages measured by the centroids residual error, for all **imagette region with hot pixel** *R*.

center and corner are consistently larger than the two edge regions. It is better to take a test suite with complete coverage, because, in absolute numbers, this involves more outliers. Center and corner are more affected by outliers according to MAD criterion, but if one would count the values of $\Delta d > 2\,\text{px}$, the edge regions would bring higher ratios. This can be seen in Fig. 4.4a, since for the regions the boxes begin above $\Delta d > 2\,\text{px}$, but the one from the center does not. However, since the FGS is not robust in case of hot pixels already in the first place, the statement of the outlier fraction is not as crucial as for the readout noise (Fig. 4.1b).

The equivalence class comparison per test suite in Fig. 4.5 generally has a higher nominal deviation Δd. This time, all regions are included in the plot, instead of just one, since we do not have a most impaired fault. The equivalence class dependence of the radius and sub-pixel class is no longer present and the brightness class shows larger deviations for brighter stars with a magnitude smaller than 6 m. The magnitude of a star does not affect the extent on the CCD, so the increasing deviation cannot be explained by this, but should come from the centroid algorithm in the FGS. The results of the test suite with 50 % coverage here encloses all others, suggesting that more edge cases were found in it. This supports the statement that the representative test cases do not always produce exactly the same effect as the rest of the test cases of their equivalence class. Otherwise, the results of the test suite with 100 % coverage would have to enclose the others. But the fact that for the hot pixel the FGS is

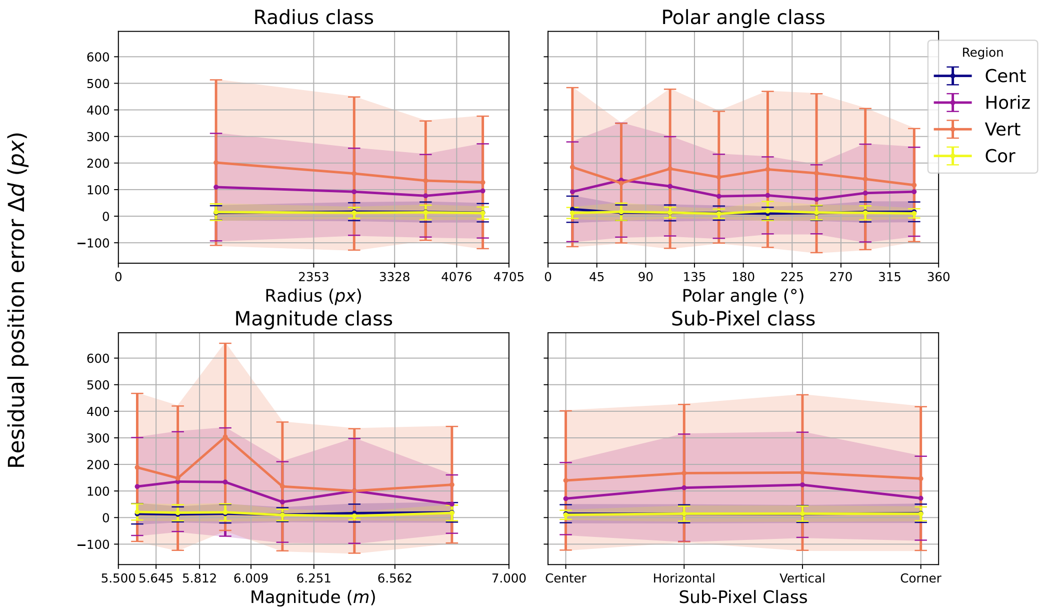


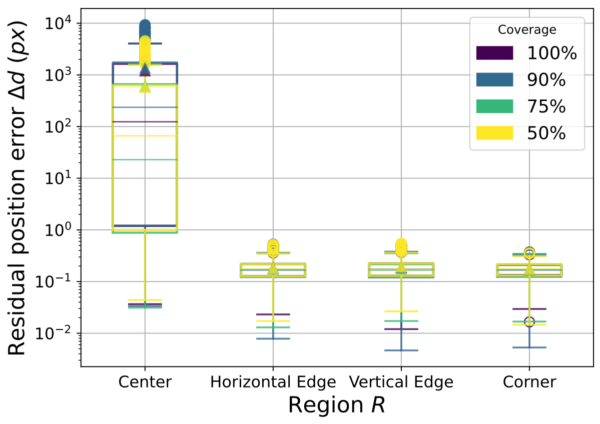
Figure 4.6.: Class-wise performance of various **imagette region with hot pixel** R for a test suite coverage of 100 %.

not robust anyway reduces the significance of the larger variance of the results from the test suite with 50 % coverage.

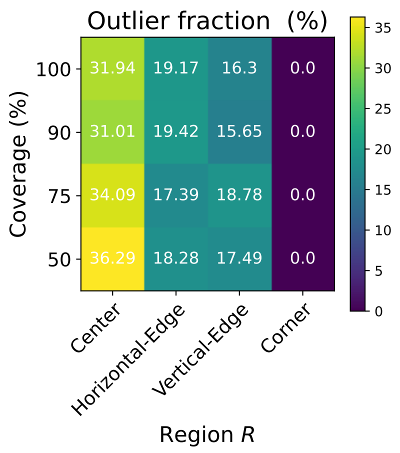
Figure 4.6 repeats the missing dependencies of the equivalence classes. The larger variance of the horizontally and vertically regions corresponds to the boxplots in Fig. 4.4a.

For the dead pixel with 20 % of background intensity, the boxplots result in Fig. 4.7a. According to [Wan+12], a dead pixel is described as a pixel with less than 40 % of mean intensity. In the center, the dead pixel brings an even larger deviation than all of the hot pixel. In this case, intensity is missing where it should be and this throws the positioning out of balance. The remaining classes, however, work acceptably. On average they have a deviation of 0.2 px, which is similar to the FGS' performance without faults. Except for dead pixels in the center, the FGS is robust. Figure A.7a depicts that the the precision of the FGS is lower if the dead pixel is positioned in the center region than if it is in one of the edge regions.

Figure 4.7b shows us that despite the generally bad behavior if the dead pixel is in the center region, there are still more than 30 % outliers with even larger residual error. For the edge regions the outlier fraction remains below 20 %. The robustness is attacked here, even if the general case with a deviation of about 0.2 px performs on average within the specifications. For the corner region there are no outliers at all, which means the FGS is completely robust to a dead pixel there. In the horizontal and vertical edge sector as well as in the



(a) Boxplots for the centroids residual error dependent on the **imagette region with dead pixel** R. The four colors represent different test suite coverages. Here, Δd is plotted on a logarithmic scale.



(b) Outlier fractions broken down by test suites of different coverages and the **imagette region with dead pixel** R.

Figure 4.7.: Boxplots and outlier fractions for the dead pixel.

corner sector the pixels usually have low intensity, so in most cases the FGS still seems to work when this low intensity in these regions is further reduced.

For Figure 4.8, all regions R are included for the equivalence class comparison. However, the center region determines the behavior in this case, since its large deviation in the arithmetic mean is most influential. For all four subplots, no equivalence class dependence can be seen. However, it is striking that the test suite with 50 % coverage produces the smallest variance, closely followed by the test suite with 75 %. Test suite with 90 % and 100 % coverage share the largest standard deviation, which most often is twice as large, and thus cover more critical test cases than the test suites with lower coverage. This speaks strongly to the advantage of a complete test test suite.

When comparing the residual position error Δd per imagette region in Fig. 4.9, the scaling of the deviation is adjusted in order to examine the behavior of the three robust cases. The diagram shows in the background in dark blue only the standard deviation bars of the results for the center region. The results for the center region of the radius, angle, and sub-pixel classes behave similarly to the case without faults. In the magnitude class, the faults of the edge regions cause a slightly increasing deviation to larger magnitudes. Since the stars are fainter, the effect of the dead pixel could be proportionally larger.

De-Focus Blur

In order to create a de-focus blur, the optical mapping is parameterized in PlatoSim. To simulate a realistic optic, PlatoSim provides an interchangeable optical model. For a de-focus we exchange the optics with those where the focal

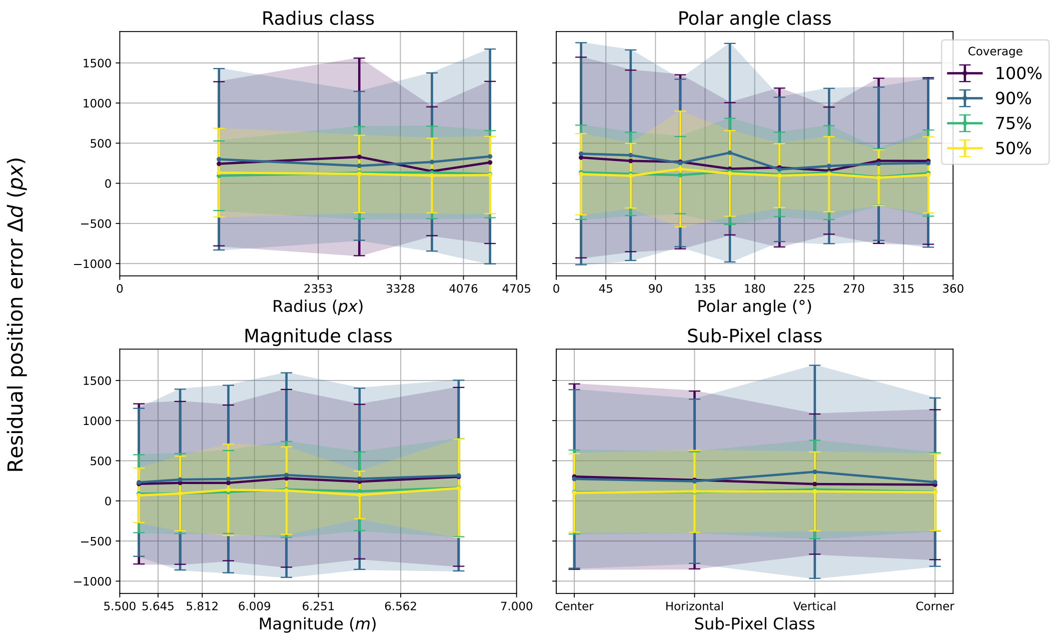


Figure 4.8.: Class-wise performance of different test suite coverages measured by the centroids residual error, for all **imagette region with dead pixel** R.

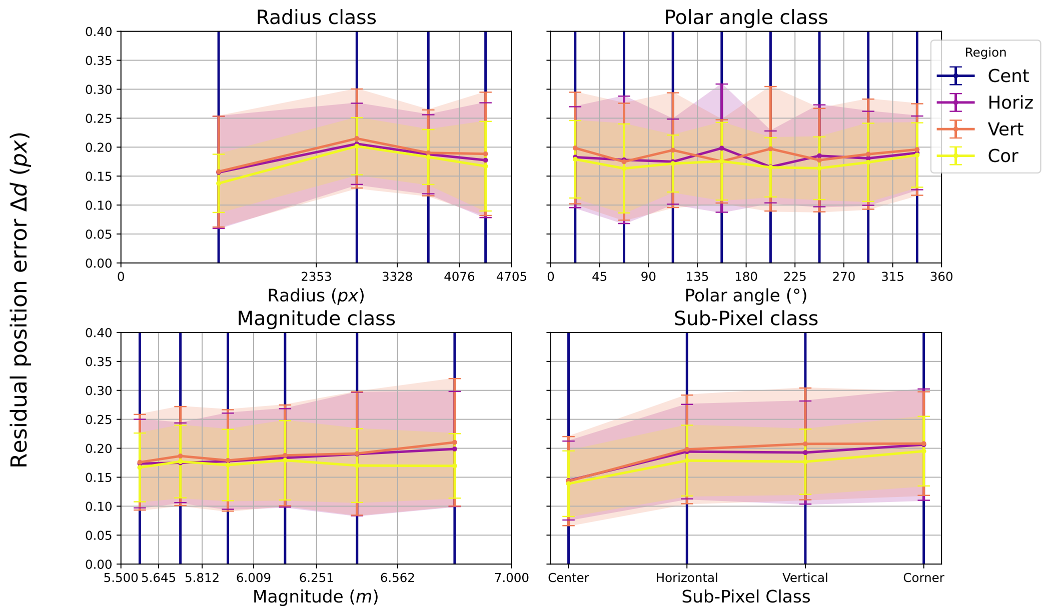
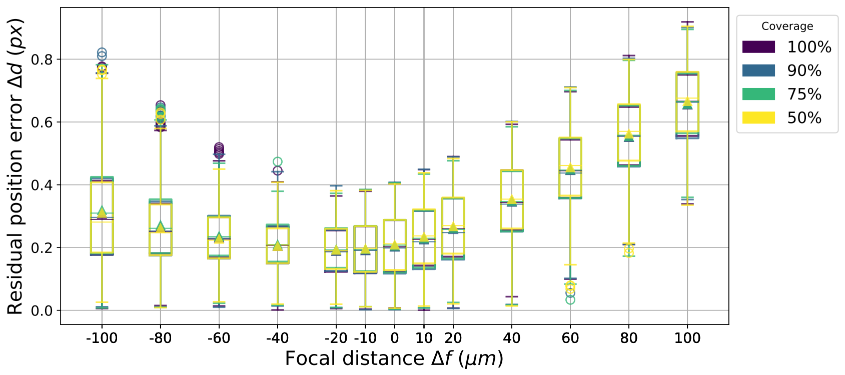
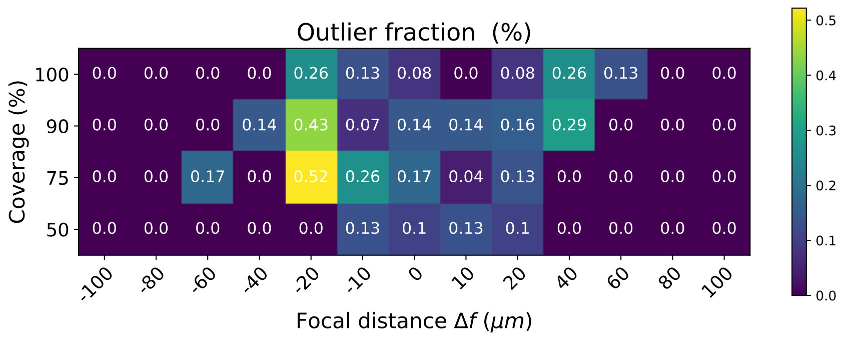


Figure 4.9.: Class-wise performance of various **imagette region with dead pixel** R for a test suite coverage of 100 %.



(a) Boxplots for the centroids residual error dependent on the **focal distance** Δf (for blue light). The four colors represent different test suite coverages.



Coverage (%)	-100	-80	-60	-40	-20	-10	0	10	20	40	60	80	100
100	0.0	0.0	0.0	0.0	0.26	0.13	0.08	0.0	0.08	0.26	0.13	0.0	0.0
90	0.0	0.0	0.0	0.14	0.43	0.07	0.14	0.14	0.16	0.29	0.0	0.0	0.0
75	0.0	0.0	0.17	0.0	0.52	0.26	0.17	0.04	0.13	0.0	0.0	0.0	0.0
50	0.0	0.0	0.0	0.0	0.0	0.13	0.1	0.13	0.1	0.0	0.0	0.0	0.0

(b) Outlier fractions broken down by test suites of different coverages and the **focal distance** Δf (for blue light).

Figure 4.10.: Boxplots and outlier fractions for de-focus blur with blue light.

length has an offset Δf. This way blur can be generated in different degrees. For the simulation, we have optical models for blue and red light at a light temperature of 6000 K.

We start with blue light and its boxplots in Fig. 4.10a. As the focal length difference increases, the position deviation Δd also increases. However, Δd is not symmetric around 0 µm. On the one hand, Δd increases more for positive focal length differences Δf, and the boxplots in the negative direction grow mainly in variance. On the other hand, for this sample, the minimum deviation is on average at −20 µm. The minimum intra-star standard deviation, and thus the highest precision, is even at −60 µm (Fig. A.10a). This indicates a miscalibration, since the lowest variance should be at 0 µm. The FGS depends on Δf and is not robust to deviations. This results from misaligned optics imaging the star somewhere else and possibly in a different shape on the CCD. This can also be seen in Fig. A.10a, as the precision always stays within acceptable limits for larger Δf. This means that the stellar center is accurately detected by the FGS, but is located elsewhere.

Outliers occur at a low level of less than 0.6 %, as shown by Fig. 4.10b. The

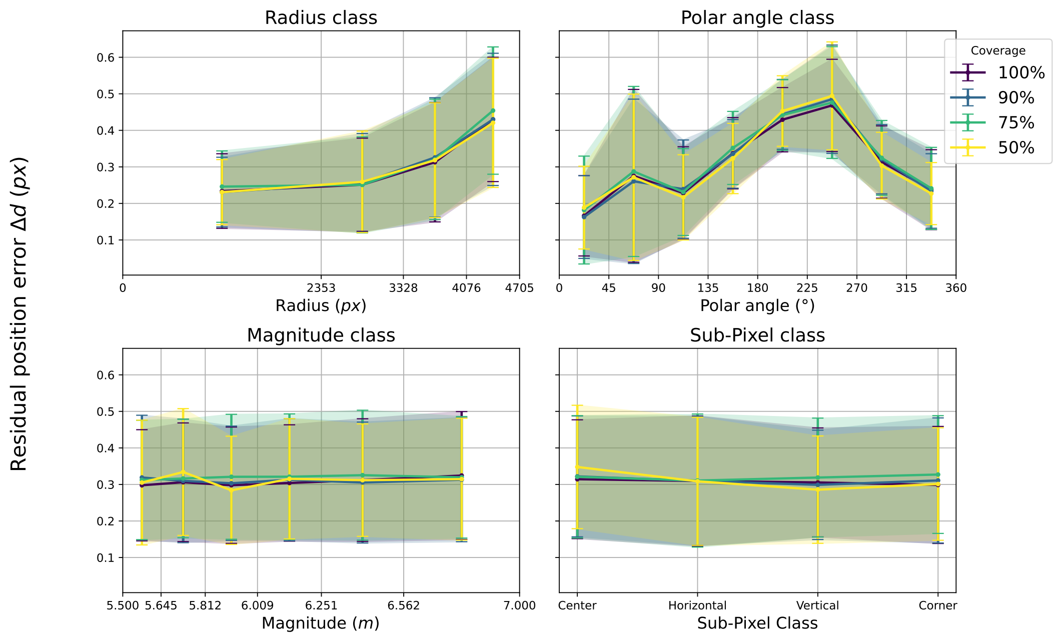


Figure 4.11.: Class-wise performance of different test suite coverages measured by the centroids residual error, for the **focal distance** $\Delta f = -100\,\mu\text{m}$ (for blue light).

defocused optics with $|\Delta d| \geq 80\,\mu\text{m}$ seem to suppress these, rather than amplify them. Figure A.10b, on the other hand, has outliers distributed throughout at roughly the same levels.

Figure 4.11 shows the equivalence class comparison among the test suites with different coverages at fixed focal distance Δf. For this extreme case at $-100\,\mu\text{m}$, the radial dependence is amplified. The two inner radius classes show the least deviation Δd and the outermost is affected the strongest. As for the magnitude class, it does not show any dependence, and neither does the sub-pixel class anymore. The latter may come from the fact that with the defocused optics the classes are dispersed. That is, stars that were previously part of a sub-pixel class appear in another region. The polar angle class shows various effects. The results of the polar classes with angles neighboring 225° have the largest deviation, those neighboring 45° the smallest. In between, the deviations are distributed continuously up and down. We will not investigate the reason for this further, but it must lie in the optics. What we cannot discover here, however, is a trend among the coverages, because they remain very similar.

In addition to the previous plot, we see the effect of the focal distance Δf everywhere in Fig. 4.12. The radius class shows that only for negative Δf the deviation increases with larger radius. With positive Δf the residual error looks similar to that of the readout noise, but with a larger deviation.

To check if the off-center position of the minimal error Δd is due to the

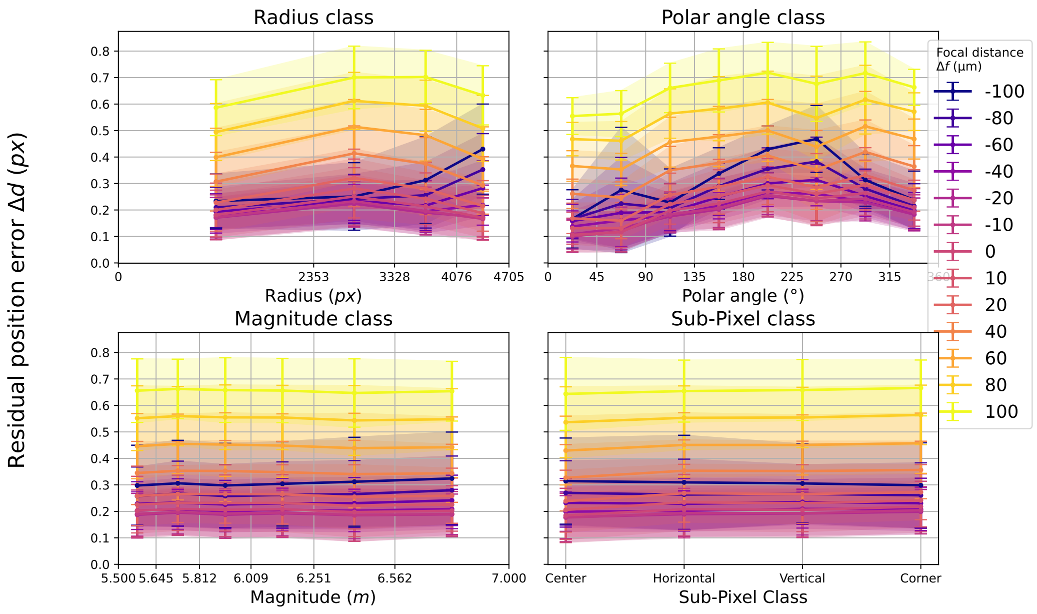
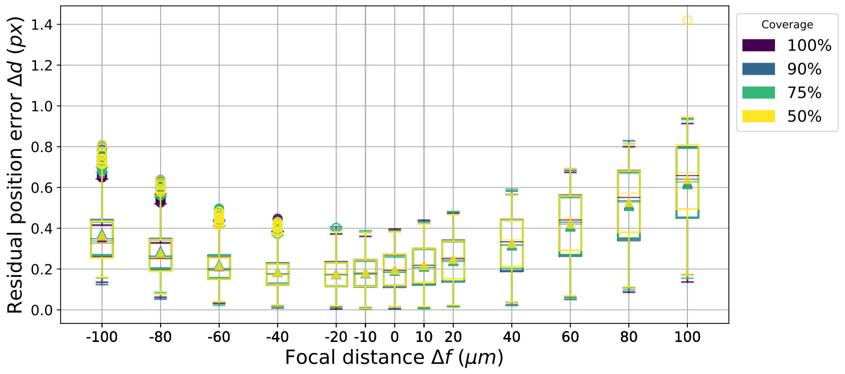


Figure 4.12.: Class-wise performance of various **focal distances** Δf for a test suite coverage of 100 % (for blue light).

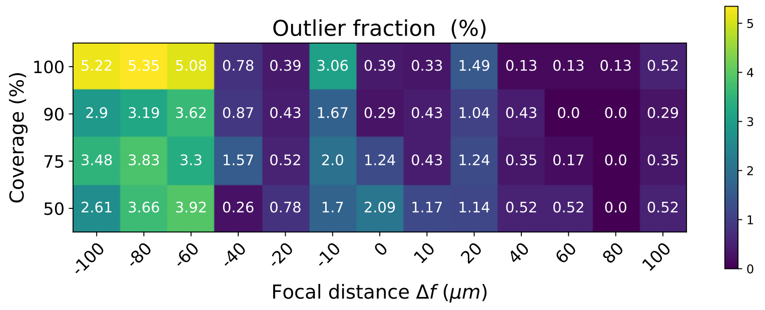
wavelength of the light, we simulated the same effect as before, with the optics for red instead of blue light. However, as Fig. 4.13a shows, the minimum here is also at $-20\,\mu\text{m}$ (for precision at $-40\,\mu\text{m}$, Fig. A.13a). This further points to a miscalibration of the optics. Difference to the blue light here is that the boxplots in positive Δf have a slightly wider distribution and those in the negative direction a smaller one. Regarding the mean values, the optics for blue and red light are similar. The boxes and medians of the different test suite coverages are in close proximity, so no superiority of one coverage over the other can be inferred.

The red light produces more than ten times the number of outliers in the maximum compared to the blue light, as Fig. 4.13b reveals. In over 96 % of the combinations there is at least one outlier. But especially in $-60\,\mu\text{m}$, $-80\,\mu\text{m}$, and $-100\,\mu\text{m}$ outliers are provoked. In this case, most notably for results with test suite of 100 % coverage. We did not investigate further what exactly causes the outliers, as the analysis of the optics is too complicated for the scope of this thesis. One difference from the results using the blue light is that the results using the red light have a broader spectrum of wavelengths, so optics may have a greater influence.

In comparison to blue light, Fig. 4.14 shows the same radial dependence, but the results for test cases in polar angle classes are distributed differently. The results per polar angular class vary among each other and several maxima and minima can be seen.



(a) Boxplots for the centroids residual error dependent on the **focal distance** Δf (for red light). The four colors represent different test suite coverages.



Coverage (%)	-100	-80	-60	-40	-20	-10	0	10	20	40	60	80	100
100	5.22	5.35	5.08	0.78	0.39	3.06	0.39	0.33	1.49	0.13	0.13	0.13	0.52
90	2.9	3.19	3.62	0.87	0.43	1.67	0.29	0.43	1.04	0.43	0.0	0.0	0.29
75	3.48	3.83	3.3	1.57	0.52	2.0	1.24	0.43	1.24	0.35	0.17	0.0	0.35
50	2.61	3.66	3.92	0.26	0.78	1.7	2.09	1.17	1.14	0.52	0.52	0.0	0.52

(b) Outlier fractions broken down by test suites of different coverages and the **focal distance** Δf (for red light).

Figure 4.13.: Boxplots and outlier fractions for de-focus blur with red light.

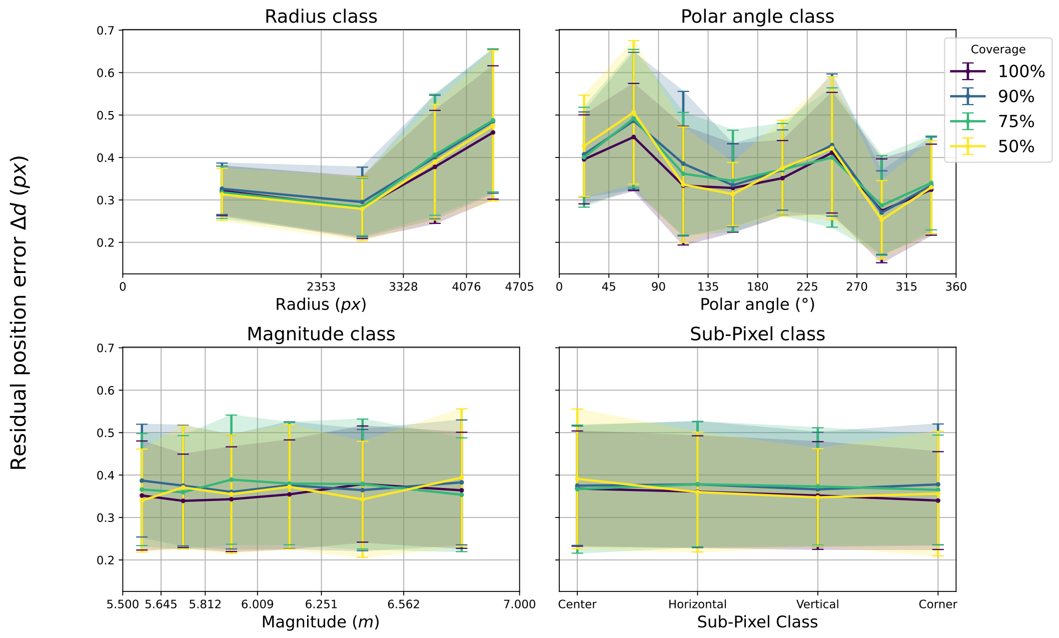


Figure 4.14.: Class-wise performance of different test suite coverages measured by the centroids residual error, for the **focal distance** $\Delta f = -100\,\mu\text{m}$ (for red light).

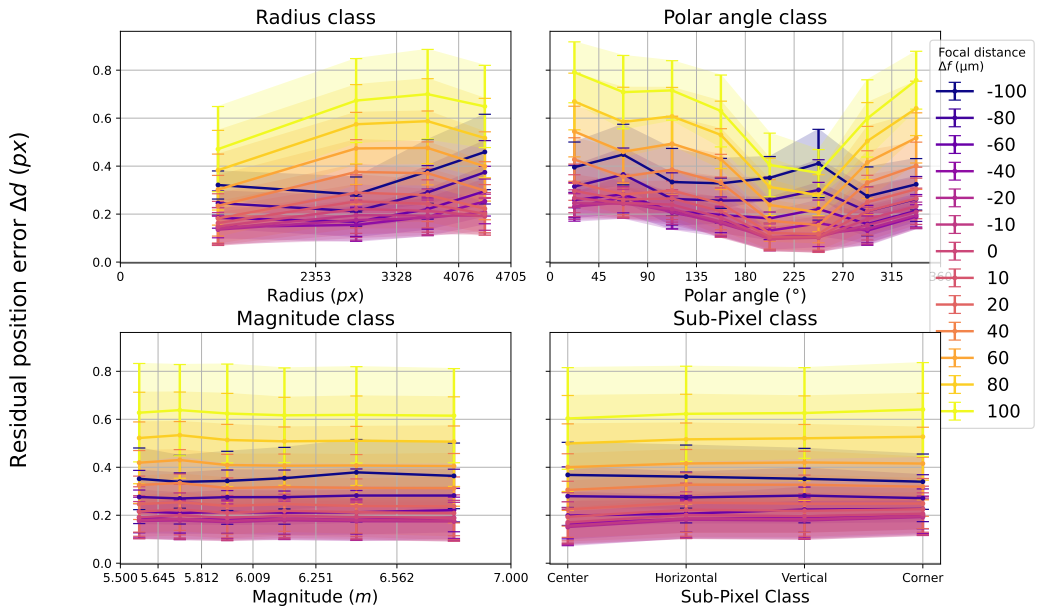
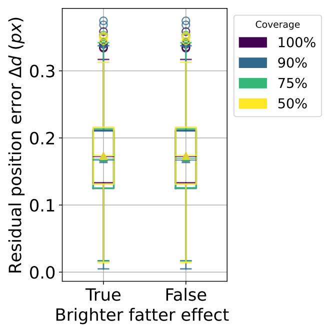
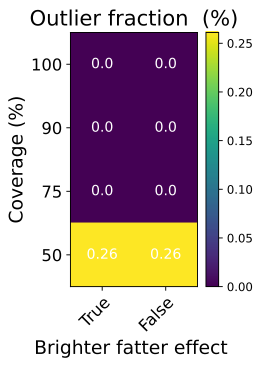


Figure 4.15.: Class-wise performance of various **focal distances** Δf for a test suite coverage of 100 % (for red light).



(a) Boxplots for the centroids residual error dependent on the active/inactive **Brighter-Fatter Effect**. The four colors represent different test suite coverages.



(b) Outlier fractions broken down by test suites of different coverages and the active/inactive **Brighter-Fatter Effect**.

Figure 4.16.: Boxplots and outlier fractions for the BFE.

Figure 4.15 only underlines the statements about Fig. 4.12.

Brighter-Fatter Effect

The parameterization for this fault class is not defined by the extent of the fault, but whether the fault is present or deactivated. It might bring insight to parameterize this effect, but we do not realize it in the context of this bachelor thesis. However, it would be possible to amplify the fault via the coefficients of the fault model.

The boxplots in Fig. 4.16a look almost identical. The BFE does not affect the FGS in the extent given in PlatoSim, so it is robust.

The outliers in Fig. 4.16b seem identicaland the outlier fraction is not affected by the BFE in our sample. The test suite with 50 % coverage having the only outliers shows that there is too few statistical evidence to make firm statements about the coverages, except that even with 100 % coverage, there is still chance involved in obtaining the relevant test cases.

The class comparison of coverages in Fig. 4.17 looks identical to that of readout noise (Fig. 4.2) and does not yield any further insights either. Same holds for the comparison of the parameterization in Fig. 4.18, where the data of the simulations with BFE overlap with those without BFE.

Dark Current

The dark current fault class is modeled differently in theory than the readout noise, but it is similar to it in that it is a noise sampled from a normal distribution and in practice it leads to a similar effect. For this purpose, the dark current was scaled by the nominal value of the dark signal.

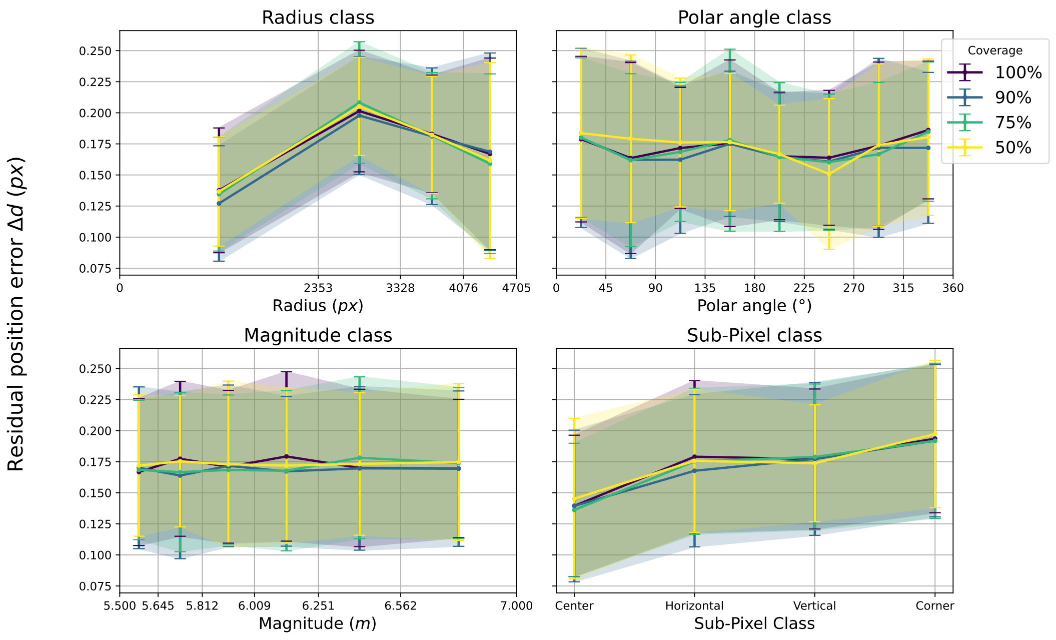


Figure 4.17.: Class-wise performance of different test suite coverages measured by the centroids residual error, for the active **Brighter-Fatter Effect**.

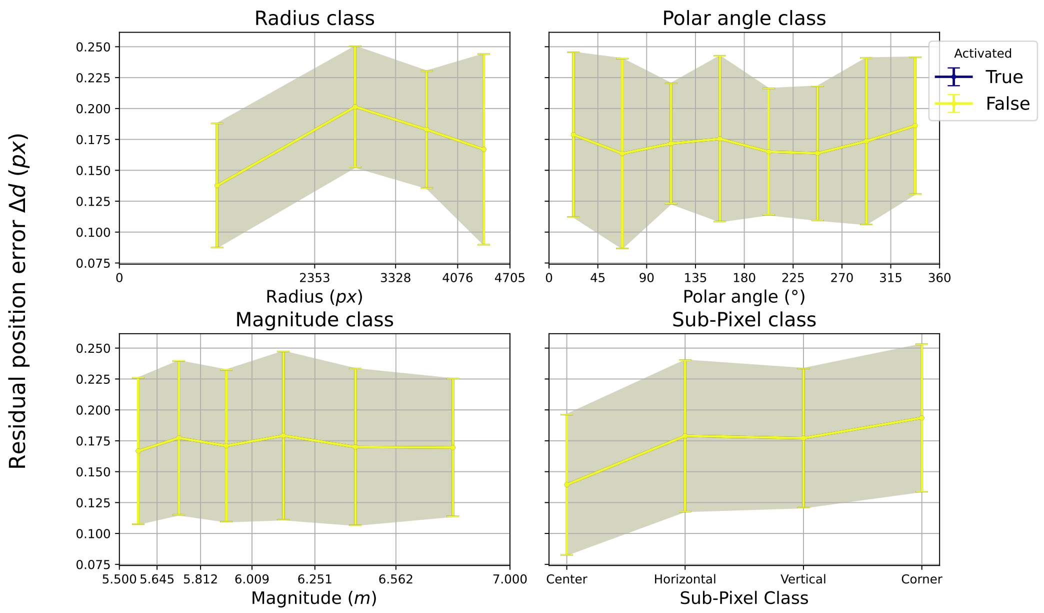
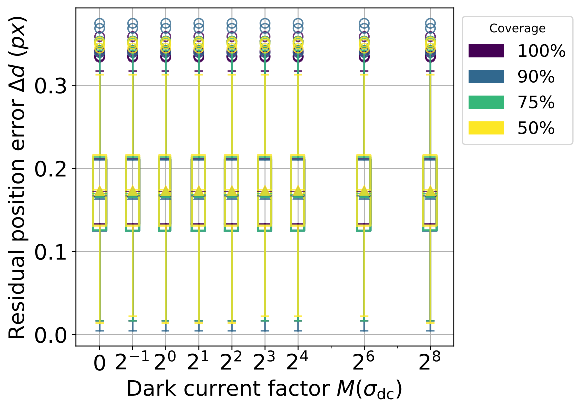
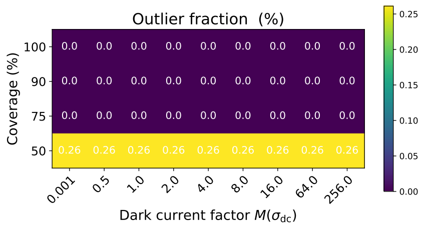


Figure 4.18.: Class-wise performance of active/inactive **Brighter-Fatter Effect** for a test suite coverage of 100 %.



(a) Boxplots for the centroids residual error dependent on the scaled **dark current factor** $M(\sigma_{\text{DC}})$. The four colors represent different test suite coverages.



(b) Outlier fractions broken down by test suites of different coverages and the scaled **dark current factor** $M(\sigma_{\text{DC}})$.

Figure 4.19.: Boxplots and outlier fractions for the dark signal fault.

The boxplots in Fig. 4.19a are also similar to the readout noise, which show no degradation during increasing dark current factor $M(\sigma_{\text{DC}})$. So the FGS is also robust against dark signal of up to 256 times. Unlike the readout noise, even the precision is robust in Fig. A.19a, only indicating an increasing deviation at 256. Furthermore, the outliers in Fig. A.19b are stable. Different to the readout noise here is that the outliers (Fig. 4.19b) in the considered $M(\sigma_{\text{DC}})$ show no change at the higher $M(\sigma_{\text{DC}})$. All $M(\sigma_{\text{DC}})$ produce the same outlier fraction.

When comparing equivalence classes on coverages in Fig. 4.20 and when comparing on the $M(\sigma_{\text{DC}})$ in Fig. 4.21 the same observation can be made as for the readout noise.

Galactic Cosmic Ray

The final fault class that we consider in this thesis is the GCR. For the parameters of this class we refer to standard settings of PlatoSim. Trail length is between 3 px to 9 px, since a shorter one is similar to the hot pixel and the longest one that fits our imagettes is $\sqrt{2} \cdot 6\,\text{px} < 9\,\text{px}$ long. We distinguish between GCRs that pass through the center region of the imagette and those that explicitly do not pass through it.

Figure 4.22a shows that the median with a GCR through the imagette center region region strongly resembles the median without a GCR. Figure 4.22b depicts, however, that this always produces outliers greater than 6 %. So the FGS can be fortunate with a GCR, or fall victim to the outlier. GCRs that do not pass through the center region have a slightly larger impact. Median and mean are slightly higher, but most importantly we always have more than 13.5 % outliers.

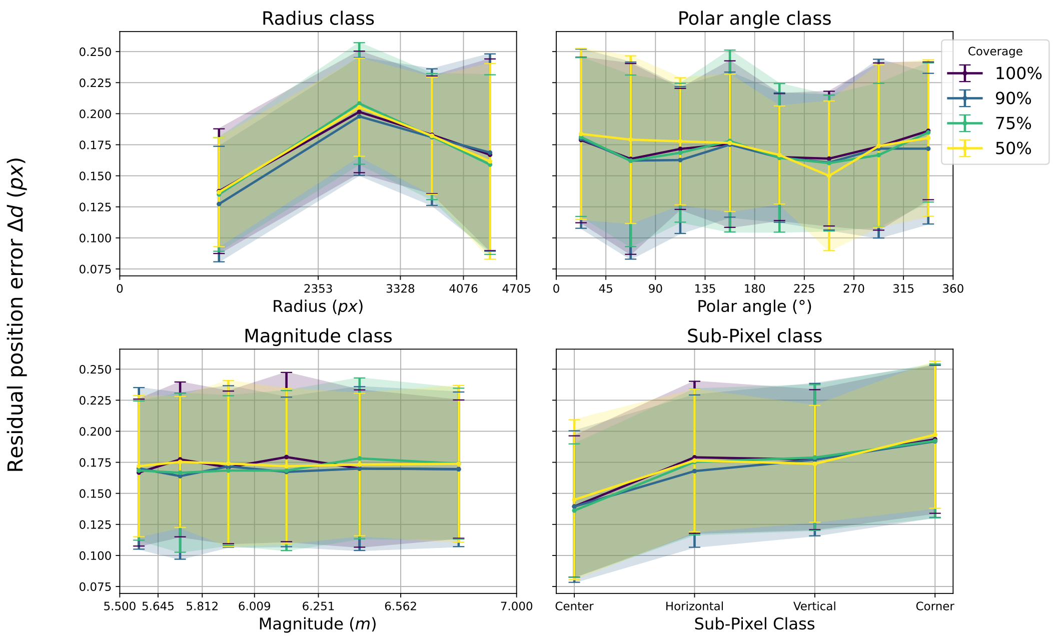


Figure 4.20.: Class-wise performance of different test suite coverages measured by the centroids residual error, for the **dark current factor** $M(\sigma_{\mathrm{DC}}) = 256$.

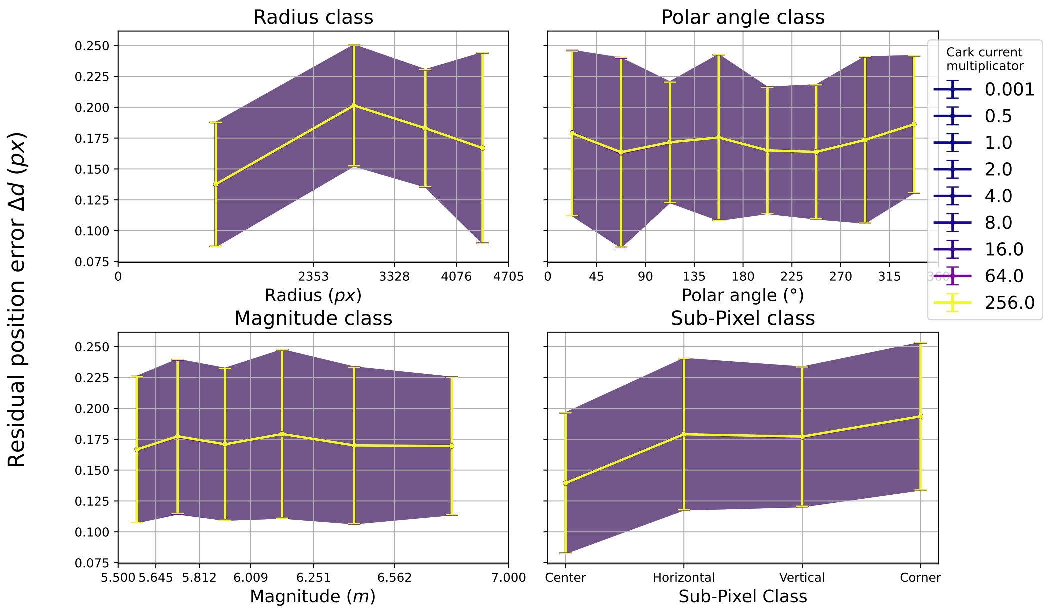
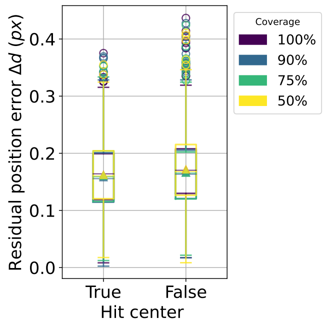
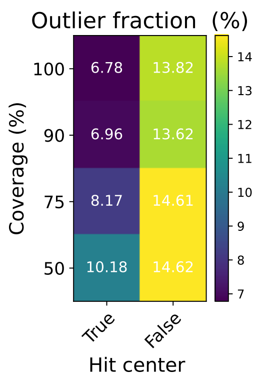


Figure 4.21.: Class-wise performance of various scaled **dark current factor** $M(\sigma_{\mathrm{DC}})$ for a test suite coverage of 100 %.



(a) Boxplots for the centroids residual error dependent on **galactic cosmic rays** hitting / not hitting the imagette center region. The four colors represent different test suite coverages.



(b) Outlier fractions broken down by test suites of different coverages and the **galactic cosmic ray** hitting / not hitting the imagette center region.

Figure 4.22.: Boxplots and outlier fractions for the GCR.

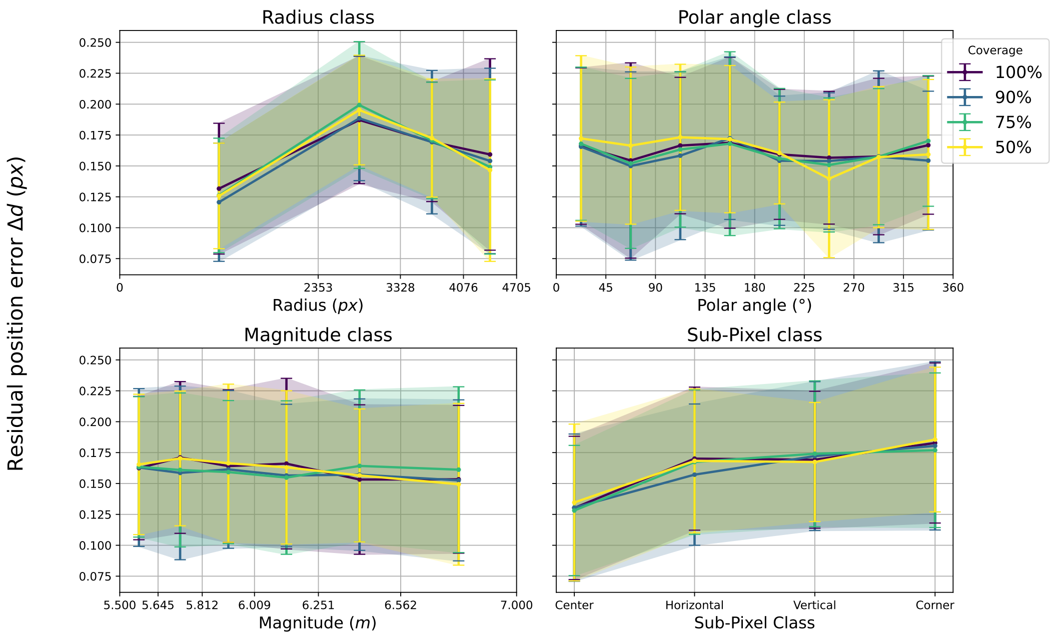


Figure 4.23.: Class-wise performance of different test suite coverages measured by the centroids residual error, for the **galactic cosmic ray** hitting the imagette center region.

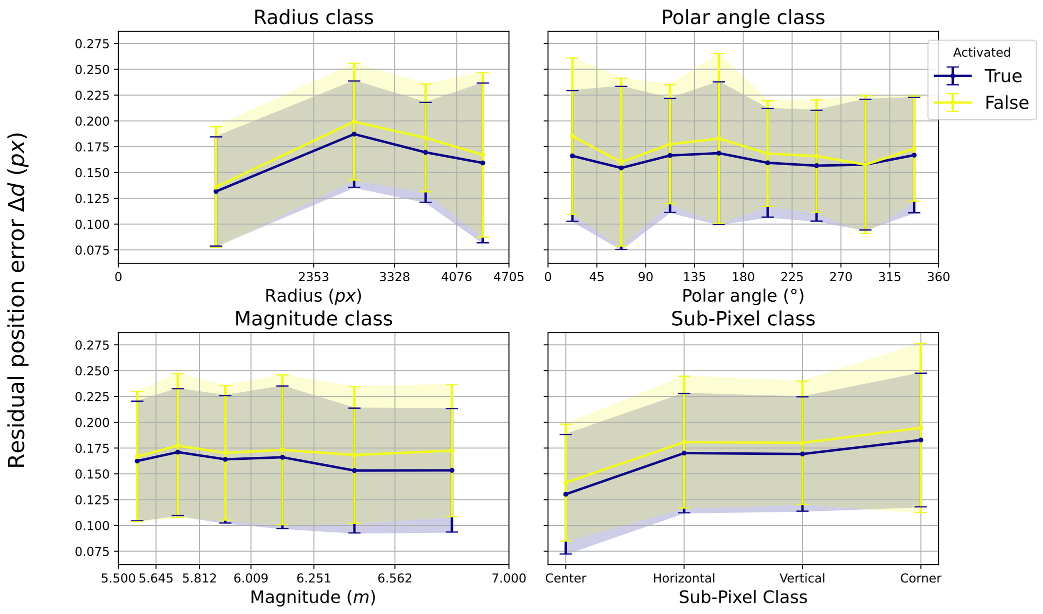


Figure 4.24.: Class-wise performance of **galactic cosmic rays** hitting / not hitting the imagette center region, for a test suite coverage of 100 %.

The comparison of test suites with different coverages Fig. 4.23 is not a gain of knowledge about the different test suites. Instead, it supports the previous evaluation.

It is evident from the comparison between the GCR types in Fig. 4.24 that for each equivalence class, the FGS produces a higher residual position error Δd from the GCR that does not pass through the center region. At the same time, Fig. A.24 shows that the precision of the FGS is worse when the GCR passes through the center region. This could be due to the altered star shape.

With this approach, we identified very robust faults, partially robust faults and faults that are not robust. The FGS is robust against readout noise even with a 16 fold magnitude, considering the accuracy, although the precision degrades by a factor of almost ten. Hot pixels at any position in a 6×6 imagette will always cause the FGS to malfunction. For dead pixels outlying the imagette center region, the FGS is robust to about 80 %. GCRs of the examined intensity affect the FGS less than dead pixels on the boundary, but tend to affect it more if they miss the imagette center region. For de-focus blur, the accuracy of the FGS decreases with larger focal distances. Against the BFE and dark current, the FGS is completely robust in the studied expressions, even concerning precision. The level of detail shows the sensitivity of our evaluation approach.

5. Conclusion

In this Bachelor's thesis, we developed an automated fault injection approach for satellite on-board image processing and applied it to the case study of the Fine Guidance System (FGS) of the PLATO mission. Due to the enormous input range of the FGS, we used an equivalence class test to determine the robustness of the FGS. Based on a literature review, we selected the fault classes and made a domain-specific selection in consultation with experts from the DLR. We considered a wide range of faults, from electronics noise to optical de-focus blur to cosmic rays. Then, we modeled these fault classes and integrated them into our custom-developed injection framework. In addition, we thoroughly evaluated the equivalence classes and test suites with various coverage. Thereby we were able to identify which faults in the input data the FGS are robust against in various settings. Furthermore, we were able to identify cases where the FGS should be improved. The general tendency is that the FGS is robust against faults with changes in the width or shape of the stars, including de-focus blur and brighter-fatter noise. In contrast, the FGS is not robust against punctual faults, such as hot and dead pixels.

Automated testing for faults is a significant factor in achieving application requirements in aerospace. Most importantly, it is central to systematically test for application-specific faults so that they can be purposefully compensated.

Our solution meets our criteria:

1. Our fault classes selected in consultation with experts at the DLR, are domain-specific, realistic and relevant.

2. The fault injection and execution of the FGS with any test set is automated to a single procedure.

3. Robustness is assessable for the individual equivalence classes via the plots for accuracy and precision.

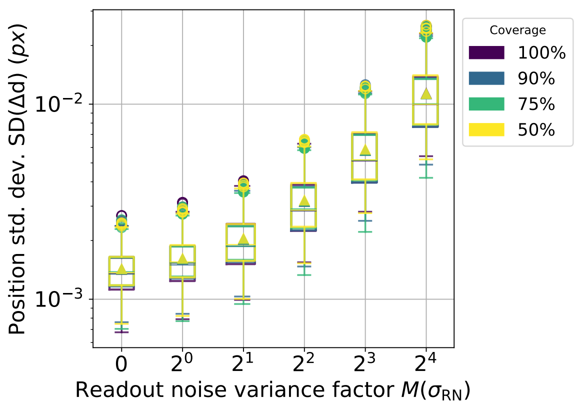
With our automated fault injection, we can determine whether the FGS is robust and how well it performs in the presence of faults. For a given test suite, the accuracy as well as the precision can be considered. We examined the overall accuracy of the FGS and the percentage of outliers with the median absolute deviation (MAD) criterion. The idea for the evaluation was a deviation distribution, like a histogram, where the occurred residuals are displayed. With the outlier criterion, we have obtained a flexible data representation by calculating the percentage of outliers. Instead of a histogram, we have used the inliers to display boxplots, which has provided a compact representation of the overall data distribution and has also allowed to distinguish between different test suite coverages and fault class parameters.

We also used our fault injection approach to evaluate the equivalence-based testing approach presented in [WGH20b]. Between the test suites of different coverages, we demonstrated no definite superiority of the complete test suite. Rather, we have found that in individual samples, it is possible that by chance a test suite with lower coverage contains more critical cases than one with higher coverage. However, we have shown that the test suites usually have similar fractions of outliers, which means that in absolute terms the full test suite usually contains the most critical cases.

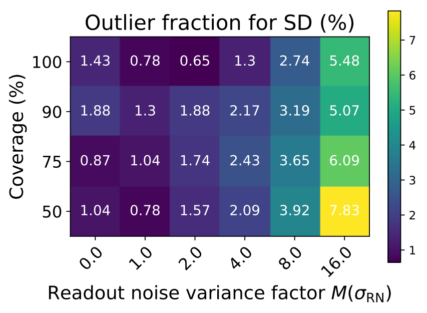
To get even more accurate results, the results of the test suites would have to be averaged much more. Repeatedly running test suite coverages would result in less scatter in the results. For a reliable error detection capability value, further sampling would also be necessary. It would also be desirable to test sets with complete and multiple covered equivalence class combinations to see if the outliers increase even more. Afterwards, it would also be possible to adjust the equivalence classes in feedback. More fault classes could be added to and examined in our system of automated injection and evaluation. On the other hand, the knowledge gained could be used to improve the FGS algorithm. In given space environments it may be relevant that the algorithm is robust under the influence of fault classes, under which it is not yet.

A. Additional Plots

Readout Noise



(a) Boxplots for the centroids position standard deviation dependent on the scaled **readout noise variance factor** $M(\sigma_{\mathrm{RN}})$. The four colors represent different test suite coverages.



Coverage (%)	0.0	1.0	2.0	4.0	8.0	16.0
100	1.43	0.78	0.65	1.3	2.74	5.48
90	1.88	1.3	1.88	2.17	3.19	5.07
75	0.87	1.04	1.74	2.43	3.65	6.09
50	1.04	0.78	1.57	2.09	3.92	7.83

(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the scaled **readout noise variance factor** $M(\sigma_{\mathrm{RN}})$.

Figure A.1.: Boxplots and outlier fractions for the readout noise fault, measured by the standard deviation.

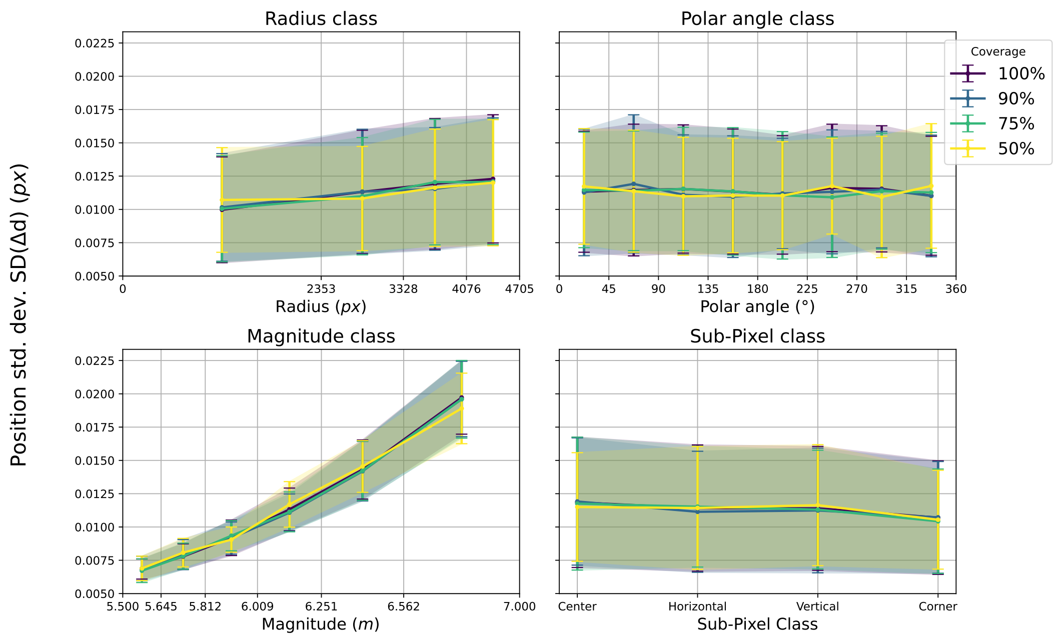


Figure A.2.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for the **readout noise variance factor** $M(\sigma_{\mathrm{RN}}) = 16$.

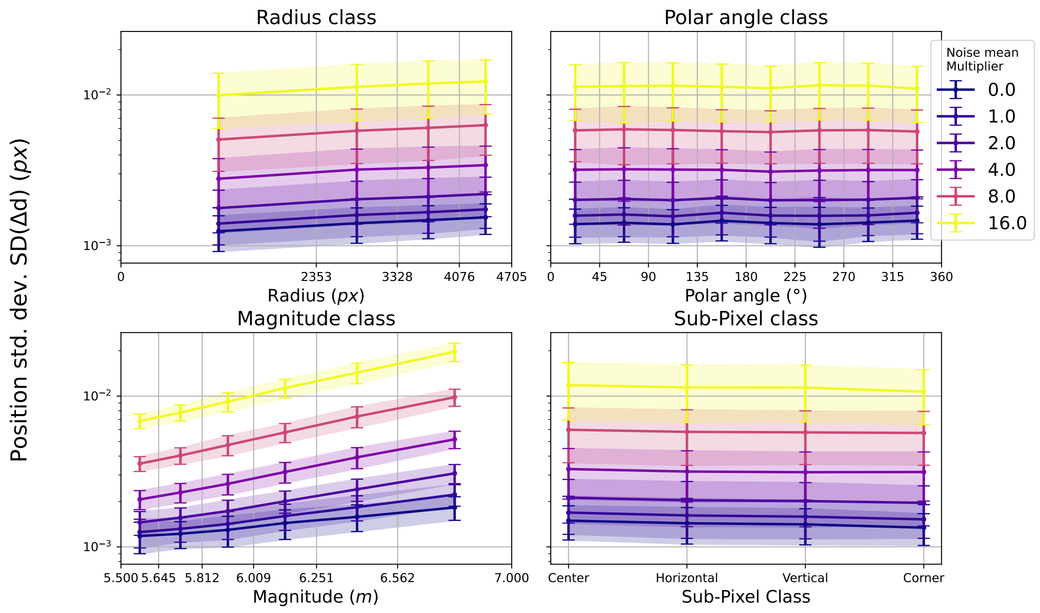
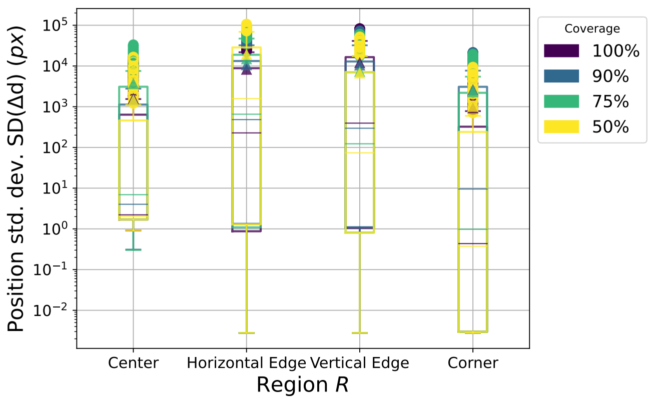
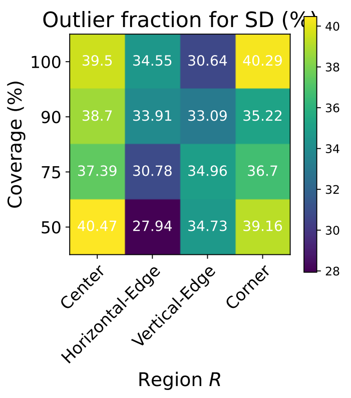


Figure A.3.: Class-wise performance of various scaled **readout noise variance factors** $M(\sigma_{\mathrm{RN}})$ for a test suite coverage of 100 %.

Hot/Dead Pixel



(a) Boxplots for the centroids position standard deviation dependent on the **hot pixel sector** S. The four colors represent different test suite coverages. Here, Δd is plotted on a logarithmic scale.



(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the **hot pixel sector** S.

Figure A.4.: Boxplots and outlier fractions for the hot pixel, measured by the standard deviation.

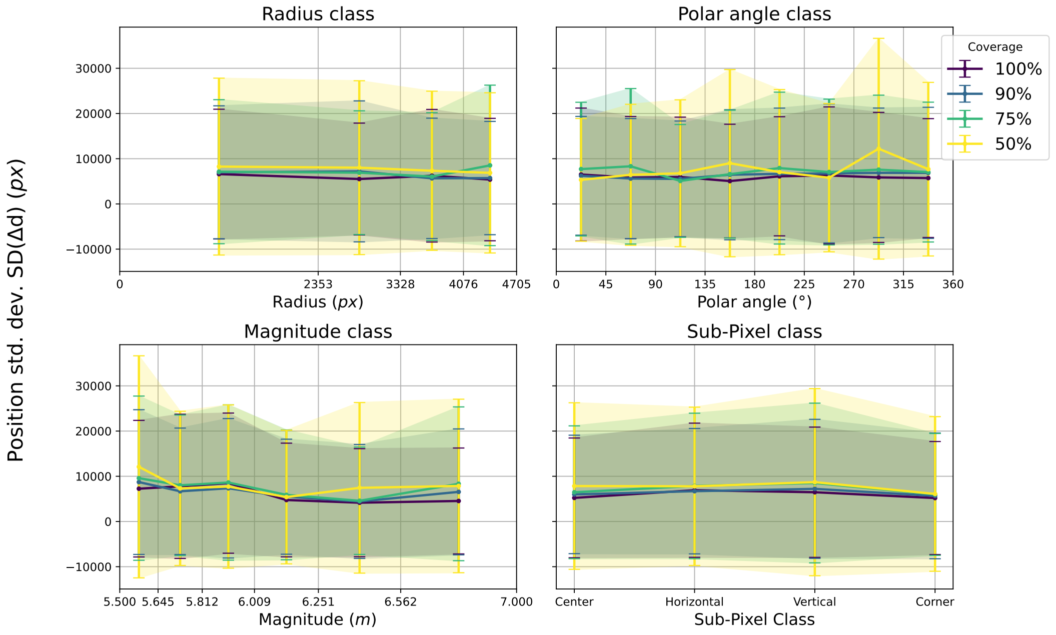


Figure A.5.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for all **hot pixel sectors** S.

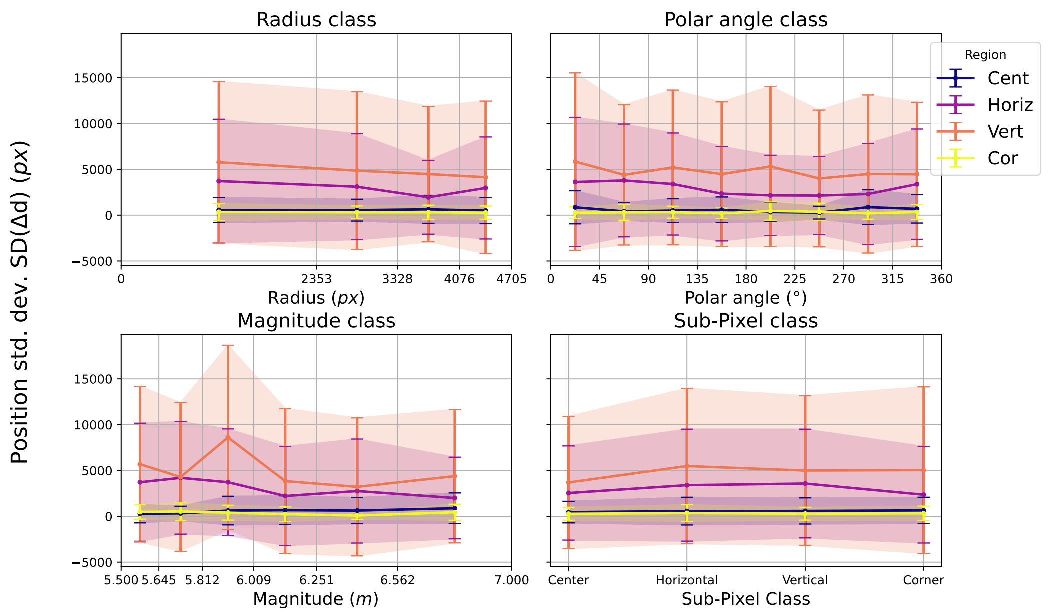
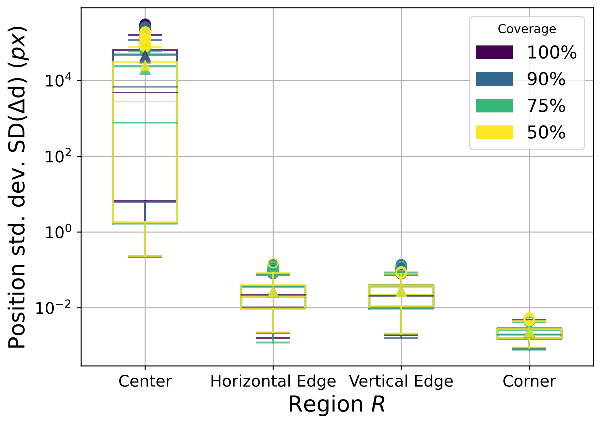
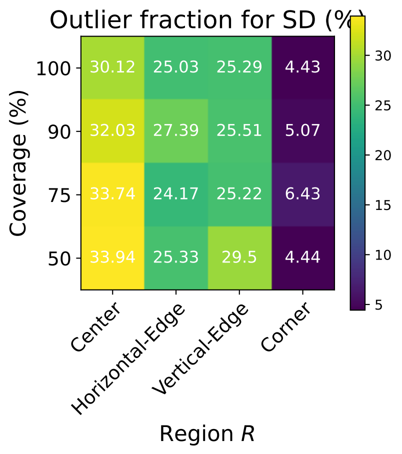


Figure A.6.: Class-wise performance of various **hot pixel sectors** S for a test suite coverage of 100 %.



(a) Boxplots for the centroids position standard deviation dependent on the **dead pixel sector** S. The four colors represent different test suite coverages. Here, Δd is again plotted on a logarithmic scale.



(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the **dead pixel sector** S.

Figure A.7.: Boxplots and outlier fractions for the dead pixel, measured by the standard deviation.

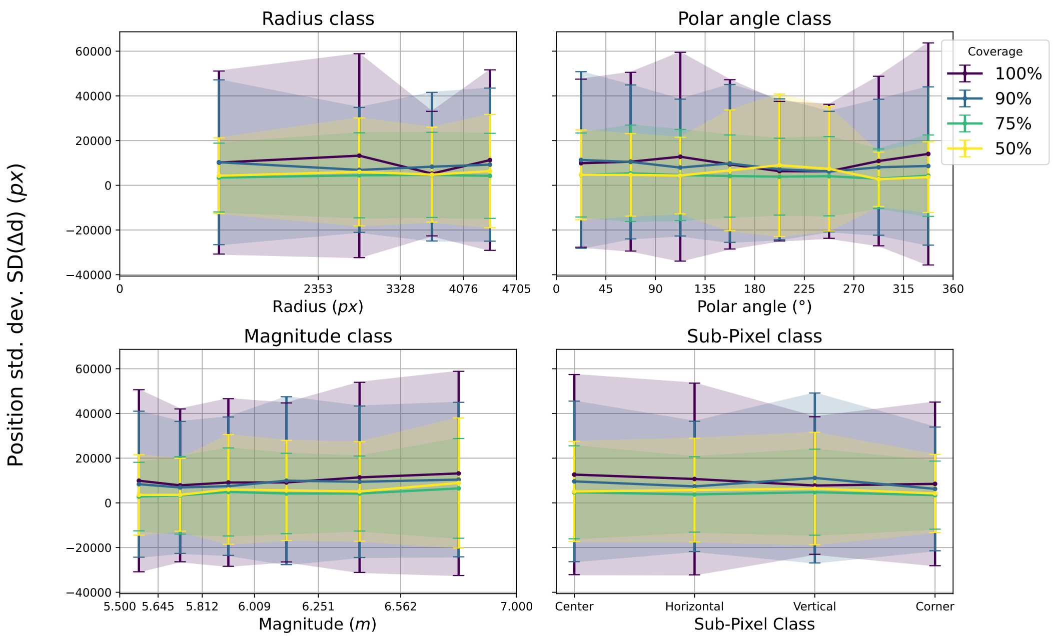


Figure A.8.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for all **dead pixel sectors** S.

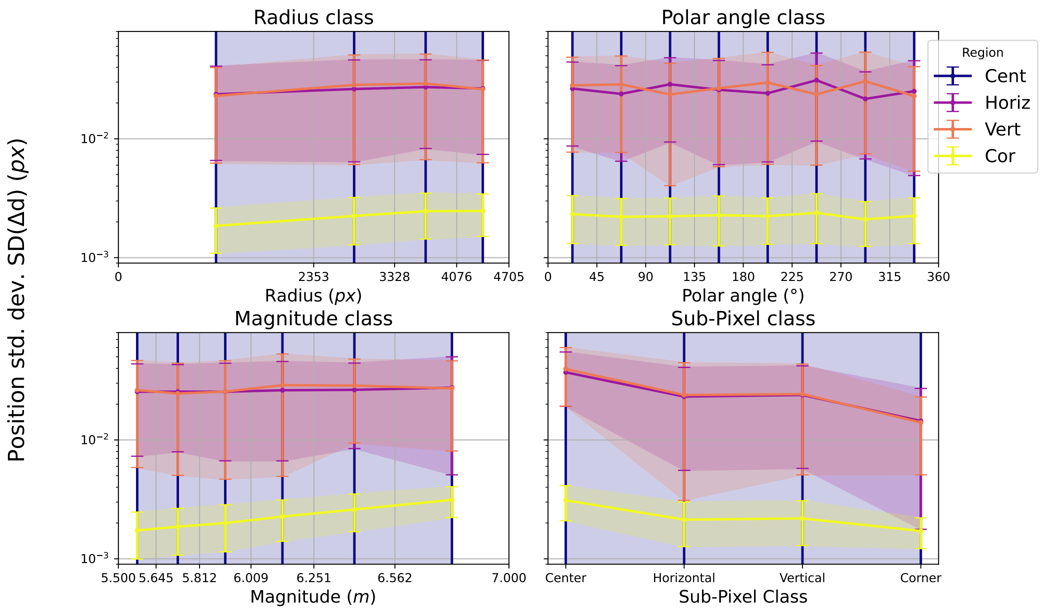
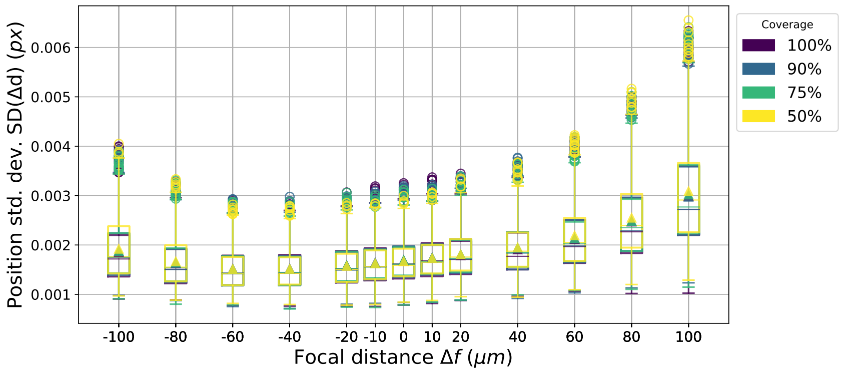
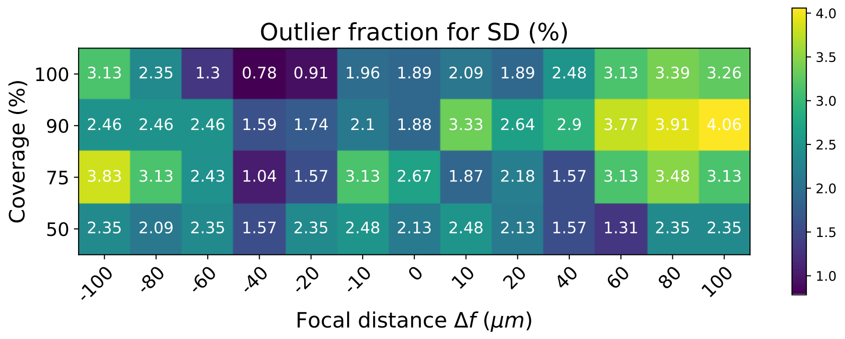


Figure A.9.: Class-wise performance of various **dead pixel sectors** S for a test suite coverage of 100 %.

De-Focus Blur



(a) Boxplots for the centroids position standard deviation dependent on the **focal distance** Δf (for blue light). The four colors represent different test suite coverages.



(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the **focal distance** Δf (for blue light).

Figure A.10.: Boxplots and outlier fractions for de-focus blur with blue light, measured by the standard deviation.

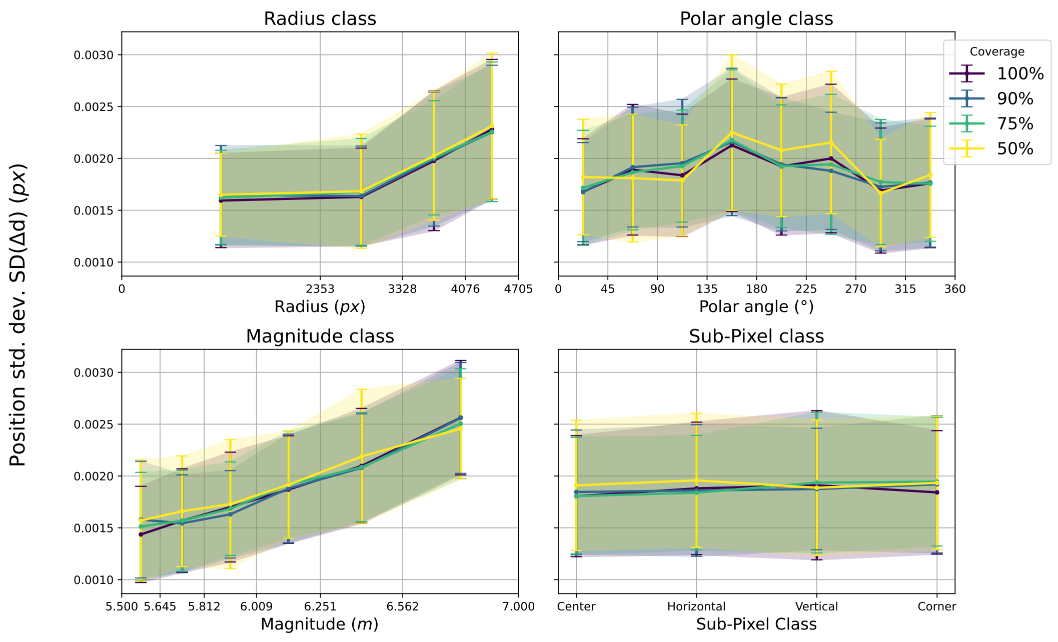


Figure A.11.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for the **focal distance** $\Delta f = -100\,\mu\text{m}$ (for blue light).

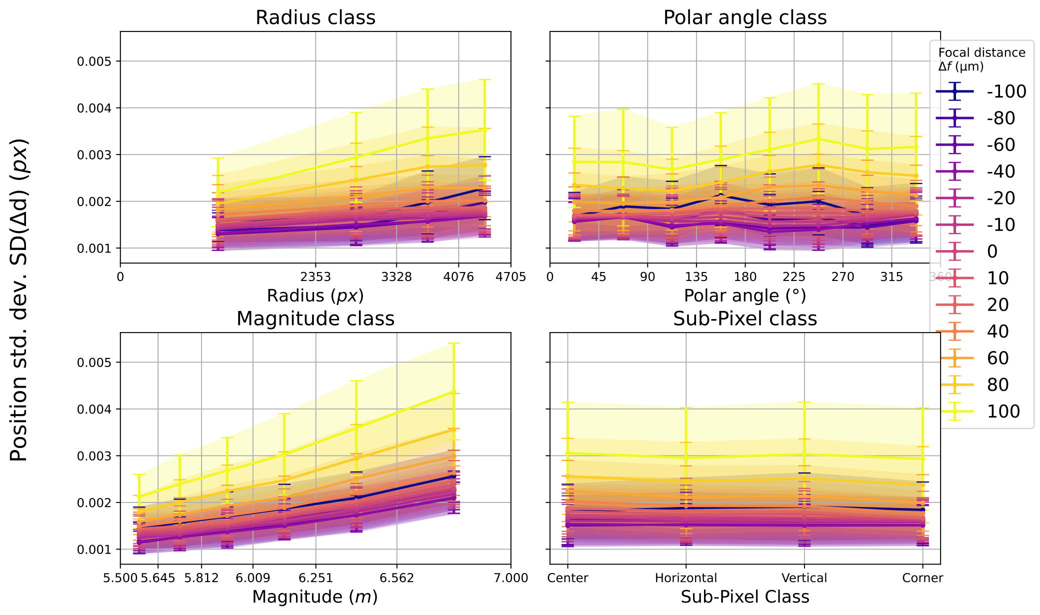
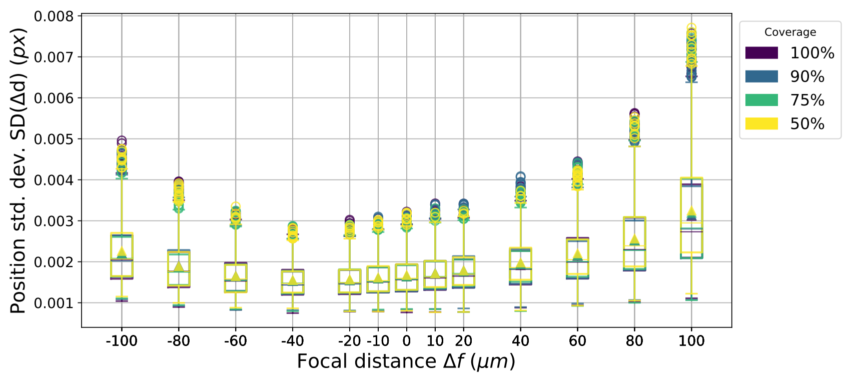
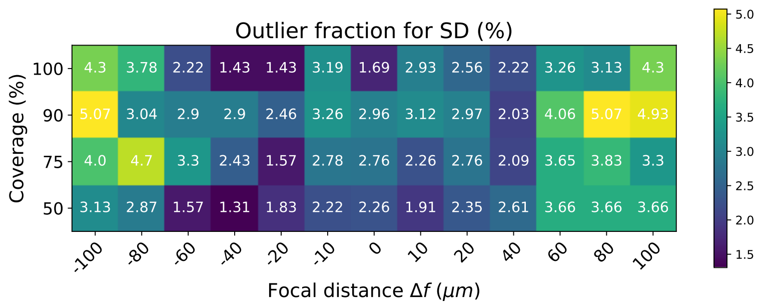


Figure A.12.: Class-wise performance of various **focal distances** Δf for a test suite coverage of 100 % (for blue light).



(a) Boxplots for the centroids position standard deviation dependent on the **focal distance** Δf (for red light). The four colors represent different test suite coverages.



(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the **focal distance** Δf (for red light).

Figure A.13.: Boxplots and outlier fractions for de-focus blur with red light, measured by the standard deviation.

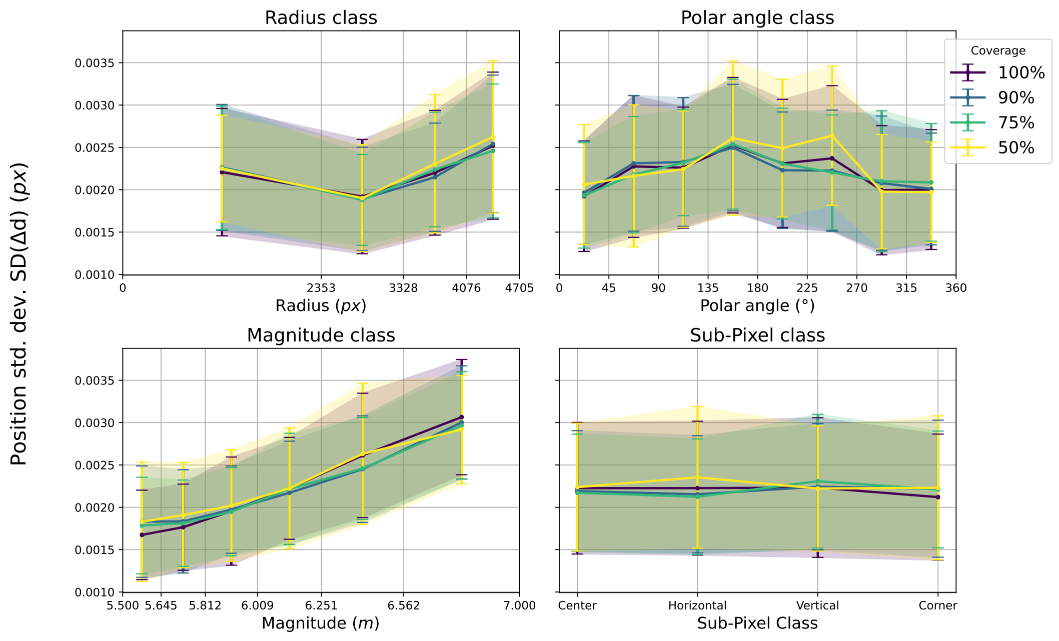


Figure A.14.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for the **focal distance** $\Delta f = -100\,\mu\text{m}$ (for red light).

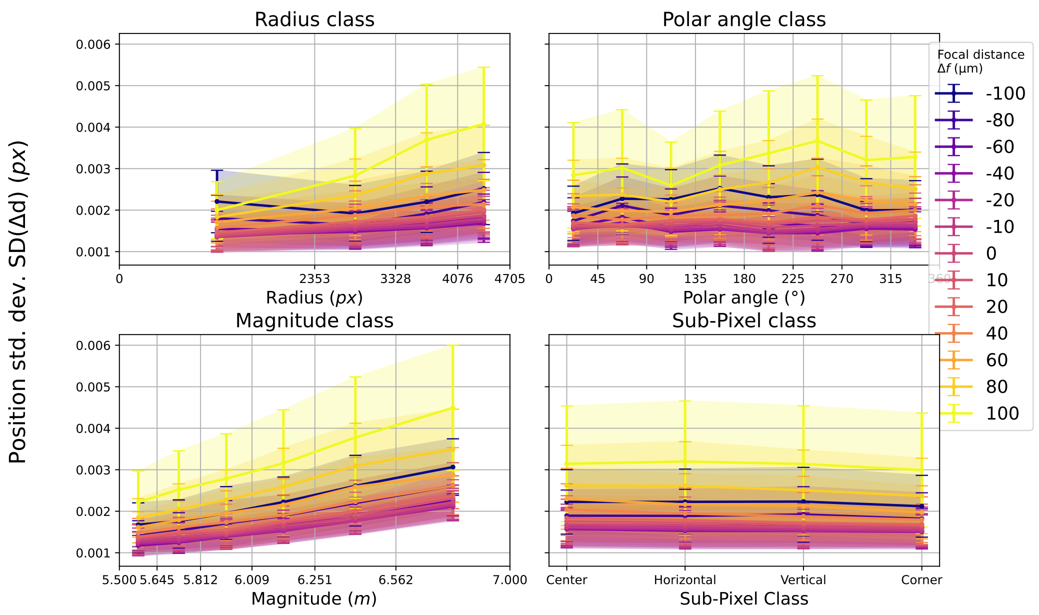
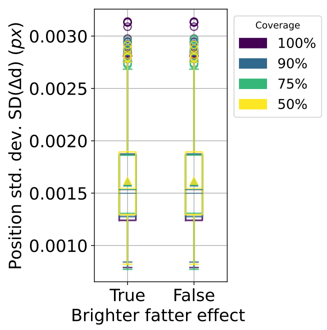
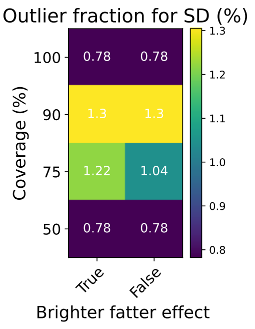


Figure A.15.: Class-wise performance of various **focal distances** Δf for a test suite coverage of 100 % (for red light).

Brighter-Fatter Effect



(a) Boxplots for the centroids position standard deviation dependent on the active/inactive **Brighter-Fatter Effect**. The four colors represent different test suite coverages.



(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the active/inactive **Brighter-Fatter Effect**.

Figure A.16.: Boxplots and outlier fractions for the BFE, measured by the standard deviation.

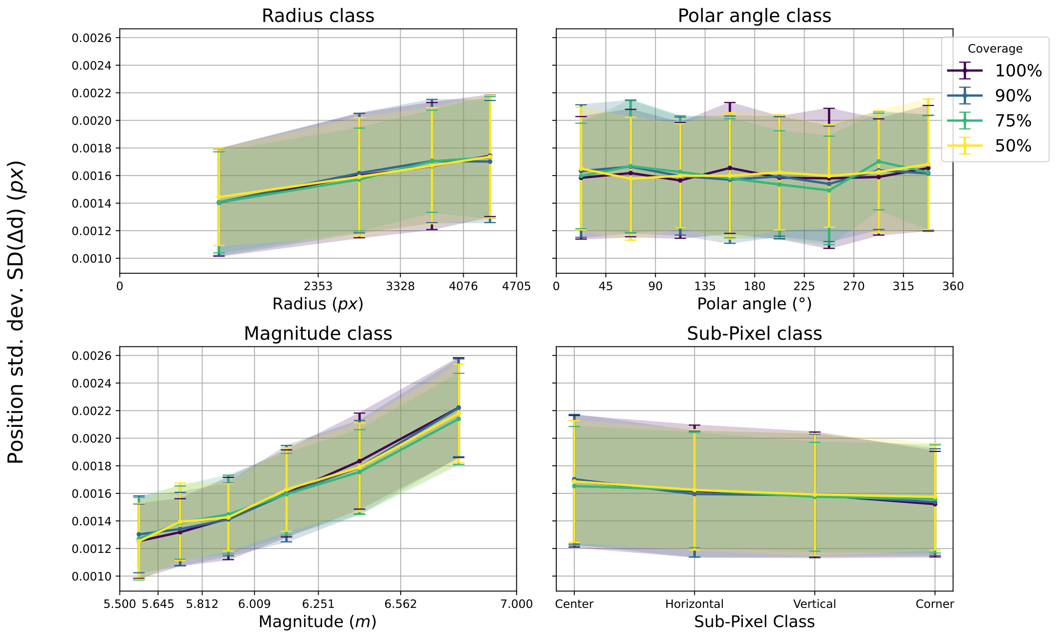


Figure A.17.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for the active **Brighter-Fatter Effect**.

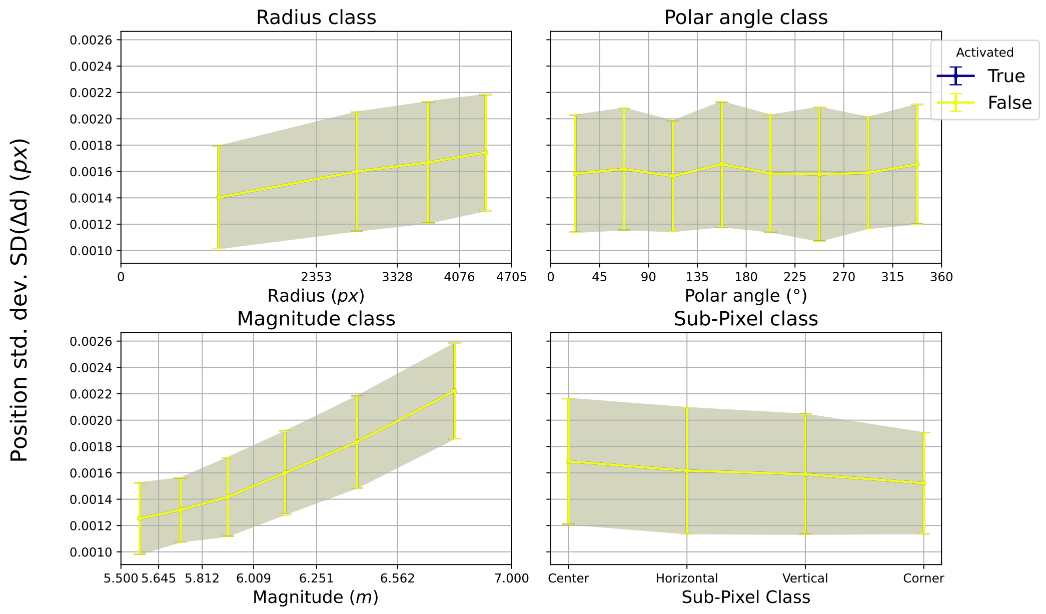
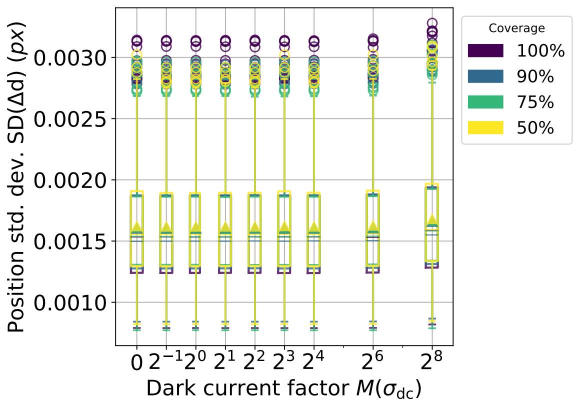
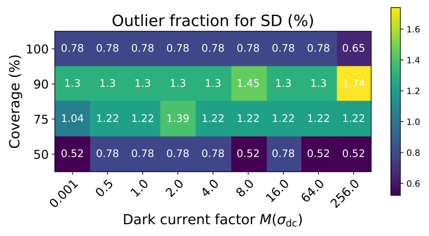


Figure A.18.: Class-wise performance of active/inactive **Brighter-Fatter Effect** for a test suite coverage of 100 %.

Dark Current



(a) Boxplots for the centroids position standard deviation dependent on the scaled **dark current factor** $M(\sigma_{DC})$. The four colors represent different test suite coverages.



Coverage (%)	0.001	0.5	1.0	2.0	4.0	8.0	16.0	64.0	256.0
100	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.65
90	1.3	1.3	1.3	1.3	1.3	1.45	1.3	1.3	1.74
75	1.04	1.22	1.22	1.39	1.22	1.22	1.22	1.22	1.22
50	0.52	0.78	0.78	0.78	0.78	0.52	0.78	0.52	0.52

(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the scaled **dark current factor** $M(\sigma_{DC})$.

Figure A.19.: Boxplots and outlier fractions for the dark signal fault, measured by the standard deviation.

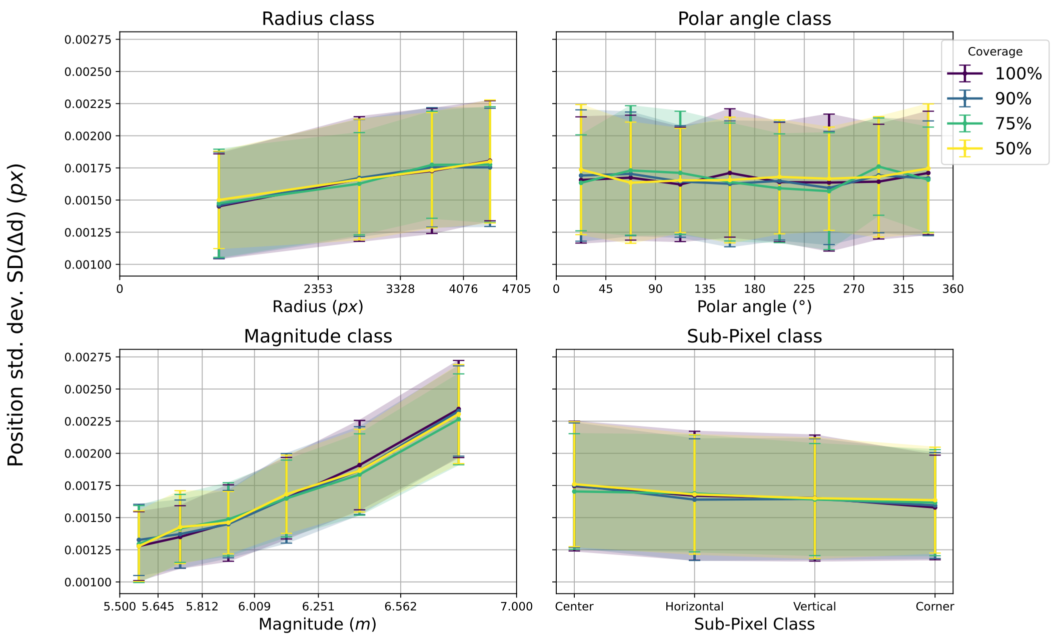


Figure A.20.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for the **dark current factor** $M(\sigma_{\mathrm{DC}}) = 256$.

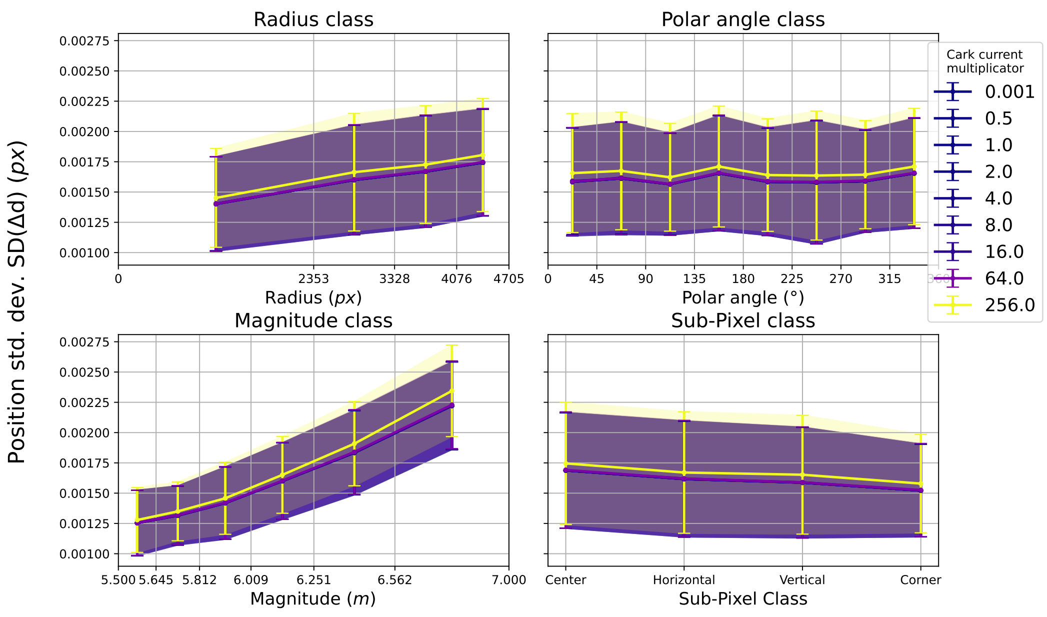
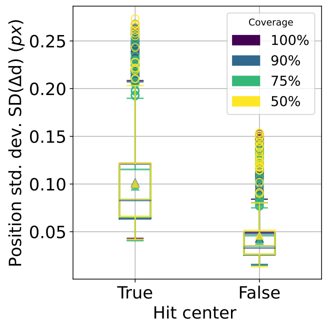
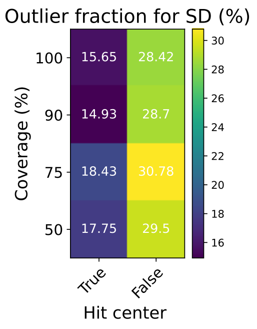


Figure A.21.: Class-wise performance of various scaled **dark current factor** $M(\sigma_{\mathrm{DC}})$ for a test suite coverage of 100 %.

Galactic Cosmic Ray



(a) Boxplots for the centroids position standard deviation dependent on **galactic cosmic rays** (not) hitting the imagette center region. The four colors represent different test suite coverages.



(b) Outlier fractions of standard deviation broken down by test suites of different coverages and the **galactic cosmic ray** (not) hitting the imagette center region.

Figure A.22.: Boxplots and outlier fractions for the GCR, measured by the standard deviation.

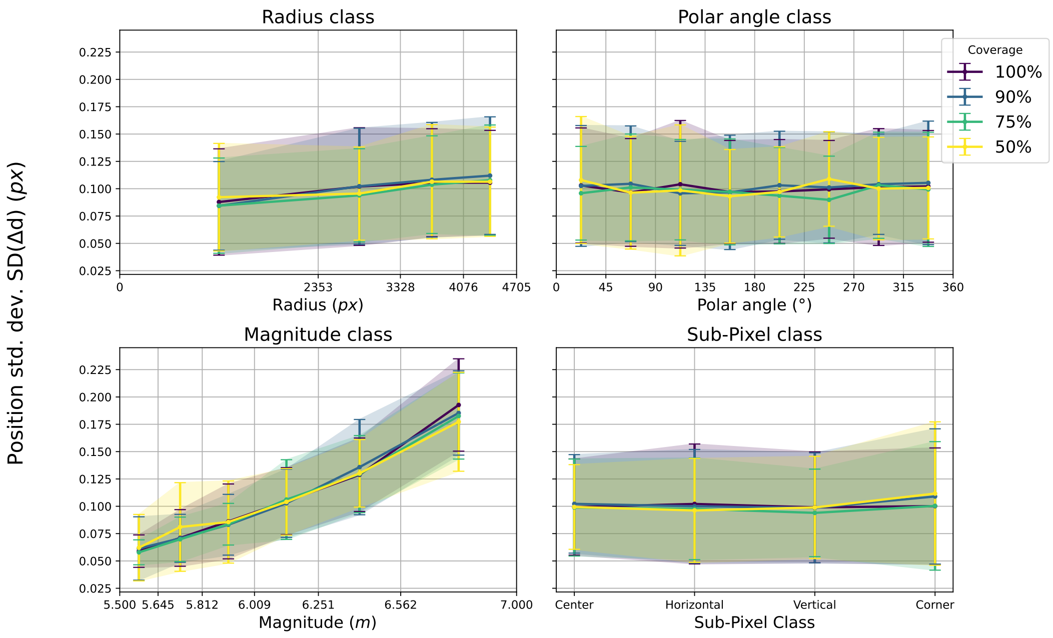


Figure A.23.: Class-wise performance of different test suite coverages measured by the centroids position standard deviation, for the **galactic cosmic ray** hitting the imagette center region.

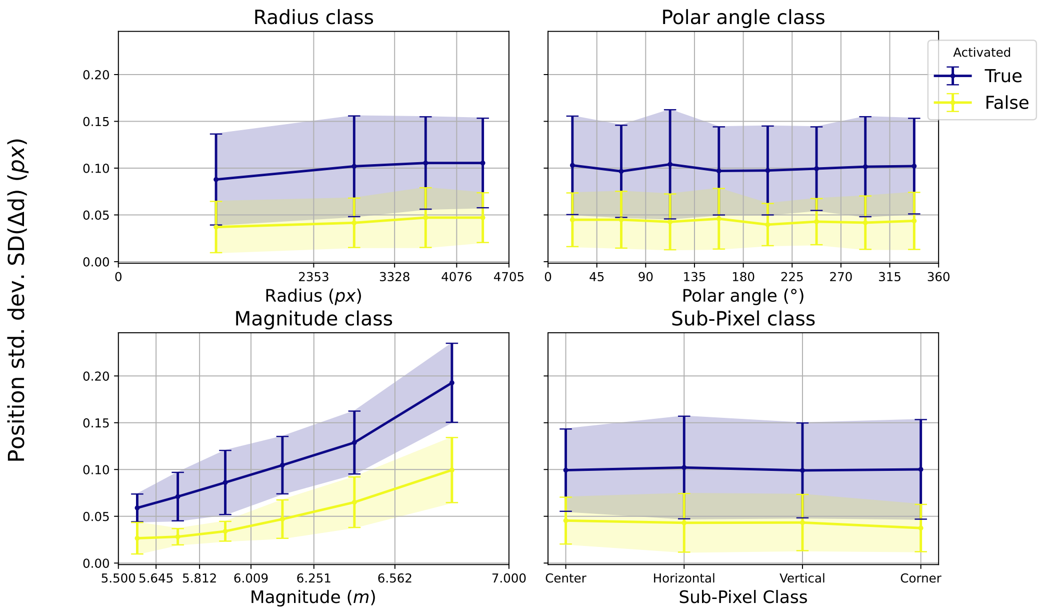


Figure A.24.: Class-wise performance of **galactic cosmic rays** (not) hitting the imagette center region, for a test suite coverage of 100 %.

Bibliography

[Ara+19] L. A. Aranda et al. “Protection Scheme for Star Tracker Images”. In: *IEEE Transactions on Aerospace and Electronic Systems* 55.1 (2019), pp. 486–492. ISSN: 1557-9603. DOI: 10.1109/TAES.2018.2849919.

[ARM17] L. A. Aranda, P. Reviriego, and J. A. Maestro. “Error Detection Technique for a Median Filter”. In: *IEEE Transactions on Nuclear Science* 64.8 (2017), pp. 2219–2226. ISSN: 1558-1578. DOI: 10.1109/TNS.2017.2666843.

[ARM20] L. A. Aranda, P. Reviriego, and J. A. Maestro. “Toward a Fault-Tolerant Star Tracker for Small Satellite Applications”. In: *IEEE Transactions on Aerospace and Electronic Systems* 56.5 (2020), pp. 3421–3431. ISSN: 1557-9603. DOI: 10.1109/TAES.2020.2971289.

[Bru+15] J. H. J. de Bruijne et al. “Detecting stars, galaxies, and asteroids with Gaia”. en. In: *Astronomy & Astrophysics* 576 (2015), A74. ISSN: 0004-6361, 1432-0746. DOI: 10.1051/0004-6361/201424018.

[CCS99] J.V. Carreira, D. Costa, and J.G. Silva. “Fault injection spot-checks computer system dependability”. In: *IEEE Spectrum* 36.8 (1999), pp. 50–55. ISSN: 1939-9340. DOI: 10.1109/6.780999.

[Cha+15] G. H. Chapman et al. “Single Event Upsets and Hot Pixels in digital imagers”. In: *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. ISSN: 2377-7966. 2015, pp. 41–46. DOI: 10.1109/DFT.2015.7315133.

[Cou+18] William R. Coulton et al. “Exploring the Brighter-fatter Effect with the Hyper Suprime-Cam”. In: *The Astronomical Journal* 155 (2018), p. 258. ISSN: 0004-6256. DOI: 10.3847/1538-3881/aac08d.

[ESA17] ESA. *PLATO Definition Study Report (Red Book)*. en. Tech. rep. ESA-SCI(2017)1. ESA, Apr. 2017, pp. 1–139. URL: https://sci.esa.int/web/plato/-/59252-plato-definition-study-report-red-book (visited on 03/11/2021).

[ESA19] ESA. *ESA Science & Technology - Artist's impression of PLATO.* 2019. URL: `https://sci.esa.int/web/plato/-/artist-s-impression-of-plato-1` (visited on 05/24/2021).

[FMP05] Jean-Claude Fernandez, Laurent Mounier, and Cyril Pachon. "A Model-Based Approach for Robustness Testing". en. In: *Testing of Communicating Systems.* Ed. by Ferhat Khendek and Rachida Dssouli. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 333–348. DOI: `10.1007/11430230_23`.

[Gri20] Denis Grießbach. *Fine Guidance System Performance Report.* en. Technical Report PLATO-DLR-PL-RP-0003. Berlin: Deutsches Zentrum für Luft- und Raumfahrt (DLR), Jan. 2020, pp. 1–40.

[Guy+15] A. Guyonnet et al. "Evidence for self-interaction of charge distribution in charge-coupled devices". In: *arXiv:1501.01577 [astro-ph]* (2015). DOI: `10.1051/0004-6361/201424897`.

[Han+01] Bruce R. Hancock et al. "CMOS active pixel sensor specific performance effects on star tracker/imager position accuracy". In: *Functional Integration of Opto-Electro-Mechanical Devices and Systems.* Vol. 4284. International Society for Optics and Photonics, 2001, pp. 43–53. DOI: `10.1117/12.426872`.

[HTI97] Mei-Chen Hsueh, T.K. Tsai, and R.K. Iyer. "Fault injection techniques and tools". In: *Computer* 30.4 (1997), pp. 75–82. ISSN: 1558-0814. DOI: `10.1109/2.585157`.

[IEE90] IEEE. "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: `10.1109/IEEESTD.1990.101064`.

[IH93] Boris Iglewicz and David Caster Hoaglin. *How to Detect and Handle Outliers.* en. ASQC Quality Press, 1993. ISBN: 978-0-87389-247-6.

[Kin14] Joe Kington. *Pythonic way of detecting outliers in one dimensional observation data.* Stack Overflow. 2014. URL: `https://stackoverflow.com/questions/22354094/pythonic-way-of-detecting-outliers-in-one-dimensional-observation-data#22357811` (visited on 06/28/2021).

[Lig09] Peter Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software.* de. second. Springer-Verlag, 2009. DOI: `10.1007/978-3-8274-2203-3`.

[Mar+14] P. Marcos-Arenal et al. "The PLATO Simulator: modelling of high-precision high-cadence space-based imaging". In: *Astronomy & Astrophysics* 566 (2014), A92. ISSN: 0004-6361, 1432-0746. DOI: `10.1051/0004-6361/201323304`.

[Mis20] Maria del Carmen Misa Moreira. “Validation and Verification of COTS FPGA in Radiation Environments: Simulation-based Fault Injection”. en. MA thesis. 2020. URL: `https://elib.dlr.de/140694/` (visited on 03/03/2021).

[Mor+19] Mehrdad Moradi et al. “Model-Implemented Hybrid Fault Injection for Simulink (Tool Demonstrations)”. en. In: *Cyber Physical Systems. Model-Based Design.* Ed. by Roger Chamberlain, Walid Taha, and Martin Törngren. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 71–90. DOI: `10.1007/978-3-030-23703-5_4`.

[Rei18] Sara Reigbo. *Signal & Noise Model for PlatoSim.* en. Technical Note PLATO-KUL-PL-TN-0007. Leuven , Belgium: Katholieke Universiteit Leuven, May 2018, pp. 1–14.

[Shu78] M. D. Shuster. “Approximate algorithms for fast optimal attitude computation”. en. In: *Guidance and Control Conference.* American Institute of Aeronautics and Astronautics, 1978, pp. 88–95. DOI: `10.2514/6.1978-1249`.

[Tro16] Pascal Trotta. “Enhancing Real-time Embedded Image Processing Robustness on Reconfigurable Devices for Critical Applications”. en. PhD thesis. Torino: Politecnico di Torino, 2016. URL: `https://iris.polito.it/retrieve/handle/11583/2641174/108761/thesis.pdf` (visited on 03/03/2021).

[Wan+12] Xingheng Wang et al. “Research of camera module sensor dead pixel detection”. In: *2012 IEEE Symposium on Electrical Electronics Engineering (EEESYM).* 2012, pp. 136–139. DOI: `10.1109/EEESym.2012.6258607`.

[WGH19] Ulrike Witteck, Denis Grießbach, and Paula Herber. “Test Input Partitioning for Automated Testing of Satellite On-board Image Processing Algorithms”. en. In: ed. by Marten van Sinderen and Leszek Maciaszek. Vol. 1. Prag, Tschechien: SCITEPRESS, 2019, pp. 15–26.

[WGH20a] Ulrike Witteck, Denis Grießbach, and Paula Herber. “A Genetic Algorithm for Automated Test Generation for Satellite On-board Image Processing Applications:” in: *Proceedings of the 15th International Conference on Software Technologies.* Lieusaint - Paris, France: SCITEPRESS – Science and Technology Publications, 2020, pp. 128–135. DOI: `10.5220/0009821101280135`.

[WGH20b] Ulrike Witteck, Denis Grießbach, and Paula Herber. “Equivalence Class Definition for Automated Testing of Satellite On-Board Image Processing”. en. In: *Software Technologies.* Ed. by Marten van Sinderen and Leszek A. Maciaszek. Vol. 1250. Series Title:

Communications in Computer and Information Science. Cham: Springer International Publishing, 2020, pp. 3–25. DOI: `10.1007/978-3-030-52991-8_1`.

[Wit18] Ulrike Witteck. “Automated Test Generation for Satellite On-Board Image Processing”. en. MA thesis. Technical University Berlin, 2018. URL: `https://elib.dlr.de/129355/` (visited on 03/01/2021).

Declaration of Academic Integrity

I hereby confirm that this thesis on *"Fault Injection for Robustness Testing of Satellite On-Board Image Processing"* is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

13.07.2021 [illegible]

(date and signature of student)

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

13.07.2021 [illegible]

(date and signature of student)