DetectorChecker Instruction Manual [1]

Julia A Brettschneider, Tomas Lazauskas, Oscar T Giles, Wilfrid S Kendall

1 Introduction

Quality assessment for digital X-ray detector panels is an important issue, and evidently impacts the quality of the resulting images. Detector panels being costly, decisions concerning replacement or refurbishing deserve careful justification. Whether or not individual damaged pixels are actually problematic depends on where they are located and on how close they are to each other. Thus spatial statistical analysis of dysfunctional pixels can be instrumental in assessing the relevance of damage. An overview of common types of spatial patterns of dysfunctional pixels can be found in [2].

The *DetectorChecker webapp* is freely accessible via the url [2] :

`https://detectorchecker.azurewebsites.net`

The web app allows users to assess digital X-ray detector panel quality via spatial analysis of dysfunctional pixels. It is built using the *R* webtool *shiny*, based on an open source *R*-package of the same name (`https://shiny.rstudio.com/`), and is connected to an open archive of detector data contributed by the user community: it is envisaged that users will be able to choose to create an account to upload their data to this archive, to facilitate more extensive analysis of data trends concerning detector damage.

The quality assessment provided by *DetectorChecker* can be used to support decision making about refurbishment and purchase of digital X-ray detectors. It can also be used to monitor the state of the detector over time and to link this to particular types of usage. It is hoped that *DetectorChecker* will assist in highlighting the occurrence of particular damage patterns, thus guiding users in assessing likely reasons for such damage.

At the beginning of a session, *DetectorChecker* allows users either (a) to select one of the common detector layouts, or (b) to enter parameters defining a custom layout. Users can then upload data in the form of `xml` or `tiff` files; *DetectorChecker* will then identify dysfunctional pixels. Users can then instruct *DetectorChecker* to create numerical and graphical summaries about the spatial distribution of the dysfunctional

[1] This manual refers to version 1.0.5 of *DetectorChecker* and version 1.0.6 of *DetectorChecker webapp*.

[2] Problems with running the web app are frequently resolved as follows: check (a) web browser is full upgraded; (b) adblocker plugins are disabled or not interfering with operation of web app; (c) Javascript blockers (e.g. NoScript) are disabled or not interfering with operation of web app.

pixels, either on the pixel level or on the *event level* (automatically identifying and working with clusters of damage). The event level is likely to be more closely linked to the reasons for damage, hence may provide a clearer picture of what is going on. Finally, users can request that severely damaged regions of the detector are masked so as to exclude them from the analysis. This enables the effect of one specific damaged region to be decoupled from an assessment of the remainder of the detector.

DetectorChecker has been developed as follows: JAB wrote the original *R*-code and contributed to the design of the web app, TL converted the code into an *R*-package *DetectorChecker*, and designed and built the *DetectorChecker webapp*, which was further developed by OTG, while WSK initiated, supervised and tested the project at all stages. The authors thank The Alan Turing Institute for providing funding for this project. They are also grateful to Jay Warnett (at Warwick Manufacturing Group) and Andrew Ramsay (during his time at Nikon Metrology, UK) for data from Perkin Elmer detectors and for guidance on X-ray machines and damages. They also wish to thank Nicola Tartoni and Ian Horswell (at Diamond Lightsource, UK) for guidance on detector types and data from Pilatus and Excalibur detectors. Further thanks for helpful discussions and feedback are given to Martin Turner and Tristan Lowe (University Manchester) and Martin O'Reilly (The Turing Institute). *DetectorChecker* and *DetectorChecker webapp* build on previous work supported by EPSRC grant EP/K031066/1, and were funded by Turing seed fund projects awarded by the Alan Turing Institute under EPSRC grant EP/N510129/1.

2 Workflow

The interactive web app is structured around tabs that allow the user to choose actions from the left-hand panel and to view numerical and graphical results on the right-hand panel. Tooltips (in pop-up boxes) are available. Outputs can be saved using standard mouse click functions from the operating system. Figure 1 presents web app functionalities and possible work-flows graphically.

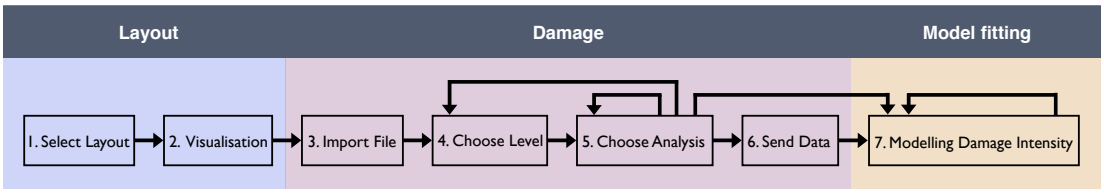

Figure 1: **Possible workflows using *DetectorChecker*, involving the *Layout*, *Damage* and *Model fitting* tabs.**

The basic work flow consists of eight steps, some of which can be run multiple times

with different options. Steps 1 to 2 concern determination of the detector panel layout. In Step 3 users upload a file either containing the coordinates of dysfunctional pixels or arising from test images obtained from the detector screen. Steps 4 to 5 concern the core functions of the web app, involving analysis of spatial patterns of dysfunctional pixels. This is followed by an opportunity to share the data with the user community in Step 6. Step 7 allows choice between some simple spatial functions as covariates in fitting of a linear model for occurrence of dysfunctional pixels. This gives statistical guidance in identification of factors that have an influence on the decay of pixels.

The steps are explained in more detail in the following itemised list.

1. **Select Layout:** Users choose from a list of common detector layouts using a drop down menu. Alternatively, users can upload a plain text file with their own layout specifications (see Section 3).

2. **Choose Visualisation:** The default visualisation is a graphical representation of the layout. Alternatively, pixelwise functions related to the layout can be visualised. This includes measuring the distances to the centre, the corners or the edges of the panel or the edges of the sub-panels. These spatial functions are provided as covariates in later model fitting (see Section 7).

3. **Import File:** Upload a file with information on dysfunctional pixel locations. This can be an `xml` file containing the coordinates of the damaged pixels or a `tiff` file containing an array of pixel intensities corresponding to the mask describing the dysfunctional pixels. These files can be created by manufacturer maintenance routines (e.g. Perkin Elmer's *Underperforming Pixel Specification* [3, Section 5.2]) in `xml` or `tiff` format. An image with open squares indicating dysfunctional pixels is then plotted automatically. Examples can be found in sub-directories of DetectorChecker/tree/master/inst/extdata of the Github repository for the web app (pointed to by *Examples* in the *Help* tab of the web app).

4. **Choose Level of Assessment:** Damage can be assessed at two different levels (see Section 6). If the *Pixels* option is selected, each damaged pixel is treated as an equally relevant occurrence. If *Events* is selected then combinations of adjacent pixels form the basis of analysis, according to user choice of combination rules.

5. **Choose Analysis:** Spatial statistics tools to explore the spatial patterns of damage are available to the user:

    - Spatial density plots based on spatial distribution of damaged pixels;

- Counts of damaged pixels associated with sub-panels of the detector;
- Arrows indicating nearest neighbours;
- Rose plot of the corresponding angles of arrows;
- Plots of K-, F- and G-functions to explore spatial randomness on different scales;
- Plots of K-, F- and G-functions corrected for spatially inhomogeneous intensity.

Plots are displayed on the right-hand side of the app display. The user can repeat this step running through different choices. Mouse-clicks on individual sub-panels produce individual plots showing only the sub-panel in question: double-clicking displays the corresponding plot or report for the sub-panel in question.

6. **Send Data:** Optional data uploading using email address as unique identifier.
7. **Modelling Damage Intensity:** Choose a covariate related to detector layout for use in a linear model for pixel damage, and fit the model.

3 Detector Layout

Preset layouts can be chosen corresponding to common detector panels. These include Perkins Elmer (full and cropped versions), Pilatus and Excalibur. The web app will automatically display the chosen layout in the sub-tab *Layout Analysis.* The sub-tab *Summary* lists the parameters of the layout.

Alternatively, users may define their own layout. It is assumed to be composed of *sub-panels* arranged in a rectangular grid. They are also referred to as *modules* or *read out groups (ROG)*. Sub-panels can be seamless (as in the Perkin Elmer design; for details see [3]), or include gaps between rows and/or columns. The dimensions of the sub-panels and the sizes of the gaps can vary, so long as the resulting layout still forms a grid. In other words, the gaps between adjacent columns of sub-panels need to be the same in all rows, and the gaps between adjacent rows need to be the same in all columns. Figure 2 shows an example of such a layout.

To specify your own layout:
▶ In Step 3 choose *user-specified.* A "Browse" button appears. Click on the button to locate and upload a plain text file containing a list of parameters specifying the layout; the Appendix 8 gives two examples. (Note that the plain text file will have

the extension "`.txt`"; users who use MS Word to construct the file should export the result as a "`.txt`" file.)

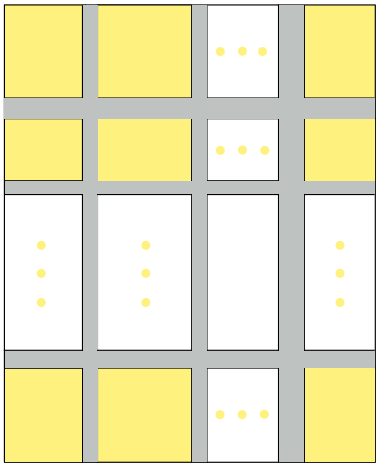

Figure 2: **General layout with sub-panels and gaps.**

4 Dysfunctional pixels

Dysfunctional pixels are referred to by many names including *bad, dead, erratic, stuck, hot, defective, broken and underperforming.* Maintenance protocols may include the creation of so-called *bad pixel maps* containing their locations. Pixels flagged as damaged in such maps are identified by a combination of criteria based on signal intensities, noise levels, uniformity and lag (see e.g. [3]). Common formats for bad pixel maps are `xml` for coordinates or `tiff` for masks. *DetectorChecker* can process both of these formats.

After uploading the file, *DetectorChecker* automatically produces an image indicating the dysfunctional pixel locations by open squares. An example for a highly damaged detector panel is shown in Figure 3. The pixels are not plotted to scale but are magnified to ensure visibility of isolated damaged pixels.

We distinguish the types of spatial arrangements of dysfunctional pixels defined in [2]:

1. singletons
2. doubles
3. triplets

4. larger clusters

5./6. upper / lower vertical lines

7./8. right / left horizontal lines

The example shown in Figure 3 contains singleton dysfunctional pixels scattered all over the panel, lines of dysfunctional pixels intersecting with the horizontal midline indicating the two rows of sub-panels, a small cluster of dysfunctional pixels in the top right corner, and a high density region in the bottom right corner.

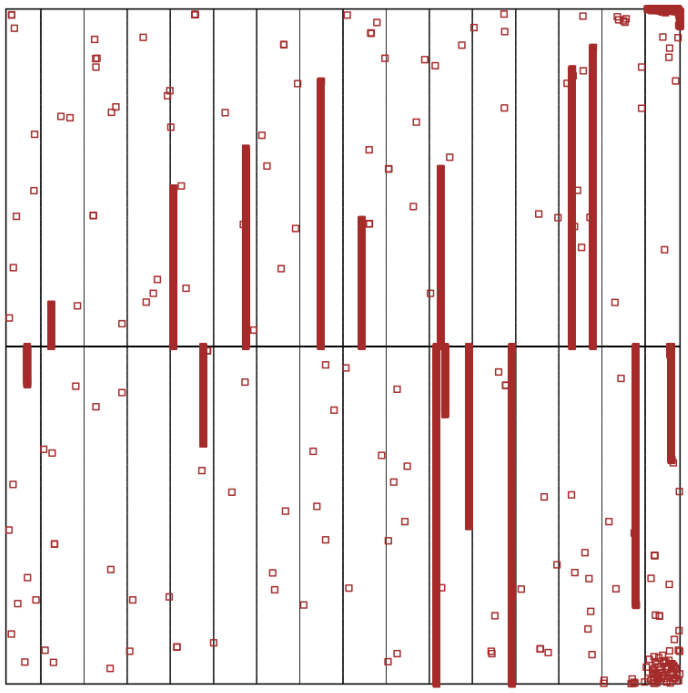

Figure 3: **Representation of detector panel with dysfunctional pixels.**

5 Spatial analysis

Spatial arrangement of dysfunctional pixels can provide important clues to the reasons for damage. Step 5 of the *DetectorChecker* work-flow offers a number of exploratory tool to investigate this.

To see approximate areas of concentrated damage:

▶ Choose *Density*.

To visualise the directions between nearest dysfunctional pixels by arrows:

▶ Choose *Arrows*.

To obtain the resulting distribution of all the angles corresponding to the arrows:
▶ Choose *Angles*.

To obtain plots of functions examining complete spatial randomness:
▶ Choose *K-func.*, *F-func.*, or *G-func.*.

To obtain plots of functions examining complete spatial randomness, while correcting for spatial inhomogeneity:
▶ Choose *K-func.*, *F-func.*, or *G-func.*.

Note that plots of *K-func.*, *F-func.*, and *G-func.* all use scales determined by the data (the default option for the underlying *R* package). Therefore scales are not harmonized between different plots.

Having made a choice, click on the *Plot* button. Recall that clicking on a sub-panel, followed by double-clicking, produces a corresponding plot or report for the sub-panel in question. In the case of a density map, the smoothing chosen for the sub-panel is different from that chosen for the plot for the entire panel, so it will not simply be a detail of the larger image.

For details on the statistical methods and interpretation underlying these tools see [1].

6 Levels of Assessment

While the natural entity of damage are pixels, this may be neither the best way to detect the reasons for damage nor the best way to rank their importance. For this reason, a notion of a higher level analysis was introduced in [1], summarising groups of adjacent pixels as *events*. Speaking in general terms, lines are represented by one of their ends, and small clusters by their centre. For more details about the motivation and the technical definitions related to pixel and event perspectives see [1].

DetectorChecker permits use of either of the two alternative levels of quality assessment, *pixels* or *events*, defaulting to *pixels*. To perform the alternative event based analysis:
▶ In Step 4 choose *Events*. A list of types will pop up. Choose a subset of these types and click on the *Plot* button.
▶ In Step 5 choose one of the options for graphical analysis, as described in the previous section, and click on the *Plot* button.

7 Modelling

Step 2 provided the option to display some functions that assigned coordinate values to pixels: the distance to the edges, the corners or to the vertical or horizontal sub-panel

boundaries, if applicable. These spatial functions are used as covariates in a statistical model that aims to determine factors contributing to the decay of pixels.

▶ To visualise the spatial covariates, in Step 2 choose a specific covariate (options 2 to 7) and click on the *Display plot* button.

▶ In Step 7 choose a covariate and and click on the *Fit model* button.

After a delay for computation, the right hand side of the panel will display output from fitting a linear model (logistic regression) which will assess the size and significance value of the selected covariate.

References

[1] JA Brettschneider, Warnett JW, TE Nichols, and WS Kendall. Higher level spatial analysis of dead pixels on detectors based on local grid geometry. CRiSM Working Paper Series 17-02, University of Warwick Department of Statistics, 2017. `www.warwick.ac.uk/crism/research/17-02`.

[2] JA Brettschneider, J Thornby, TE Nichols, and WS Kendall. Spatial analysis of dead pixels. CRiSM Working Paper Series 14-24, University of Warwick Department of Statistics, 2014. `www.warwick.ac.uk/crism/research/paper14-24`.

[3] PerkinElmer. Reference Manual Digital Imaging, XRD 1621 AN/CN Digital X-Ray Detector, 2006. `https://cars9.uchicago.edu/software/epics/Manual_PKIaSi_XRD1621%202008-07-30.pdf`.

8 Appendix

To upload layouts in the web app as user-define detector layouts the list of parameters below has to be stored in a text file. The first example is a detector with photographic aspect ratio built in one piece (no sub-panels).

```
detector_width = 600
detector_height = 400
module_col_n = 1
module_row_n = 1
module_col_sizes = c(600)
module_row_sizes = c(400)
gap_col_sizes = c()
gap_row_sizes = c()
module_edges_col = NA
```

```
module_edges_row = NA}
```

The second example is a detector with irregular grid.

```
detector_width = 1720
detector_height = 1060
module_col_n = 7
module_row_n = 5
module_col_sizes = c(100, 200, 300, 400, 300, 200, 100)
module_row_sizes = c(100, 200, 400, 200, 100)
gap_col_sizes = c(10,20,30,30,20,10)
gap_row_sizes = c(10,20,20,10)
module_edges_col = NA
module_edges_row = NA
```

A further example, `user-defined.txt`, can be found in DetectorChecker/tree/master/inst/extdata of the Github repository for the web app (pointed to by *Examples* in the *Help* tab of the web app).