DNG noise model

Document version: 1.2
Document date: 30th July, 2015

1. Document scope

This document describes the process for generating the DNG noise model for a given camera model, using the `dng_noise_model.py` script (which builds on the ITS infrastructure). The output of this test is a C code snippet than can be cut-and-pasted into the camera HAL for a device; note that this is a per-model set of parameters, not a per-unit set.

2. Running the test

2.1. Physical setup

This script works by capturing a series of images at varying gains and exposure times. The images should contain as little real image content as possible. To ensure best results, use several methods of filtering out real image content:

- Point the camera at a reasonably evenly illuminated flat featureless surface such as a wall (doesn't need to be perfectly uniformly lit, a slow brightness gradient across the image is OK).
- Use a diffuser in front of the lens (a simple technique is to just put scotch tape over the lens).
- Avoid flickering light sources.

Additionally, there are some constraints on the illumination level of the scene. The script needs to capture a large series of images over a wide range of exposures. The scene illumination level should remain constant throughout this process.

Here is an example of an image captured from a device ready to run the script. The soft changes in illumination are acceptable:

2.2. Invoking the script

Once the ITS environment is set up, the following command runs the script:

```
python tests/dng_noise_model/dng_noise_model.py
```

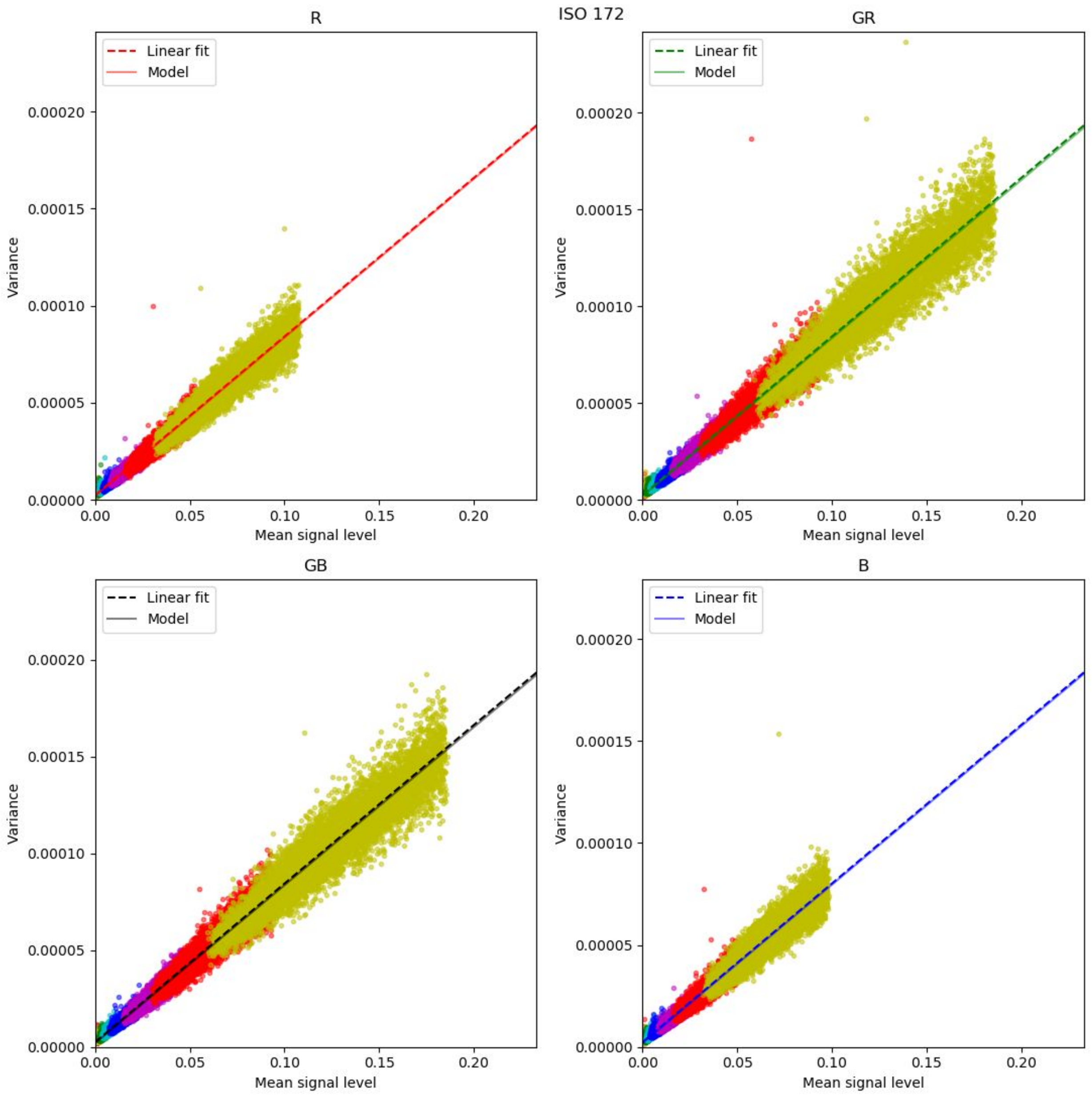
This captures several raw images from the device, copies them back to the host machine, and performs some analysis.

2.3. Script output

2.3.1. Plots

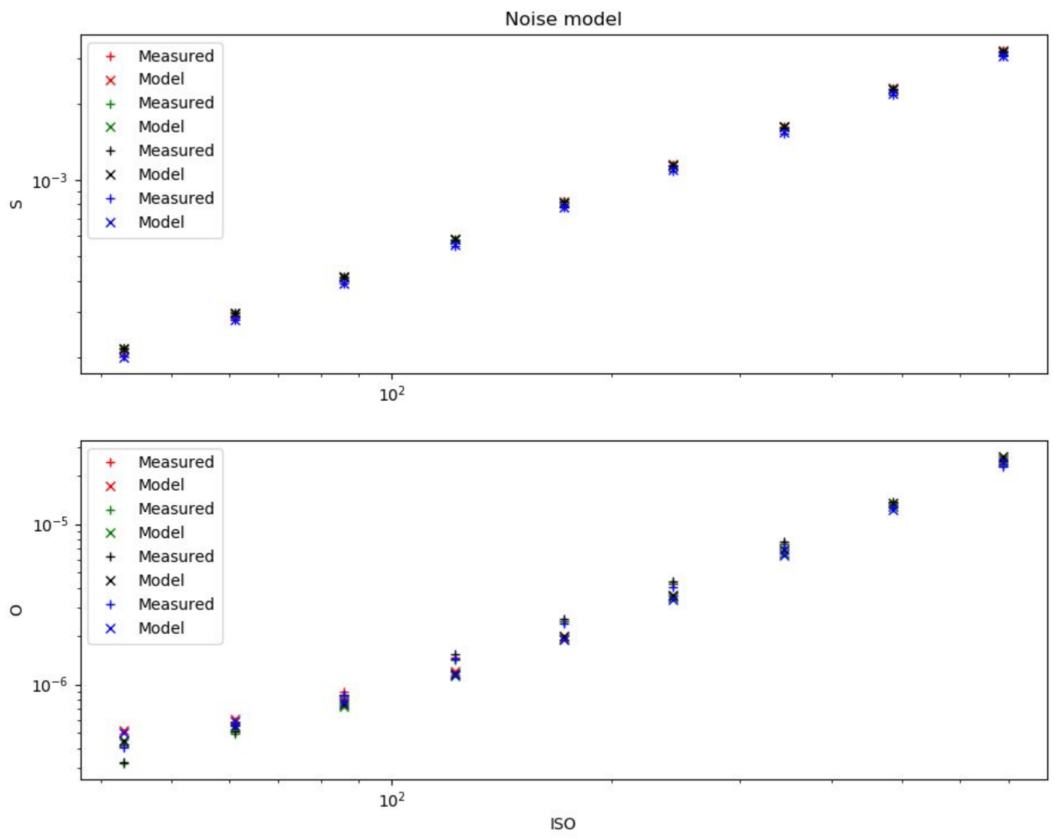
The script outputs several plots which should be inspected to ensure that it ran without problems.

The first set of images is `dng_noise_model_samples_iso<N>.png`, where N is the ISO. There should be a series of images for N ranging from the minimum to the maximum sensitivity. Each image shows a plot of a samples for the sensitivity, and two fit lines. Here is an example:



The samples are colored according to which exposure the sample was taken from. The linear fit is a linear regression of the samples in this plot, and the model fit line is the global noise model applied to this particular ISO. If the noise model describes the noise well, these two lines should be similar.

`dng_noise_model.png` shows the two components of the noise model for a given ISO (S and O), both as measured from the images, and predicted by the noise model. These two series should match closely, as in the following image:



2.3.2. Code snippet

The test also prints some code to the console, which will look like the following:

```
/* Generated test code to dump a table of data for external validation
 * of the noise model parameters.
 */
#include <stdio.h>
#include <assert.h>
double compute_noise_model_entry_S(int plane, int sens);
double compute_noise_model_entry_O(int plane, int sens);
int main(void) {
    for (int plane = 0; plane < 4; plane++) {
        for (int sens = 43; sens <= 11130; sens += 100) {
            double o = compute_noise_model_entry_O(plane, sens);
            double s = compute_noise_model_entry_S(plane, sens);
            printf("%d,%d,%lf,%lf\n", plane, sens, o, s);
        }
    }
    return 0;
}

/* Generated functions to map a given sensitivity to the O and S noise
 * model parameters in the DNG noise model. The planes are in
 * R, Gr, Gb, B order.
 */
double compute_noise_model_entry_S(int plane, int sens) {
    static double noise_model_A[] = {
4.7207840135242695e-06,4.652269569355134e-06,4.647851486795737e-06,4.471
545960443515e-06 };
    static double noise_model_B[] = {
4.662319177583059e-06,1.5499709752066674e-05,1.5865229541295095e-05,7.28
20355793415545e-06 };
    double A = noise_model_A[plane];
    double B = noise_model_B[plane];
    double s = A * sens + B;
    return s < 0.0 ? 0.0 : s;
}

double compute_noise_model_entry_O(int plane, int sens) {
    static double noise_model_C[] = {
5.267566497268042e-11,5.326240103730519e-11,5.5622783696977125e-11,5.045
64830704001e-11 };
    static double noise_model_D[] = {
4.1619950541273167e-07,3.3616733886889863e-07,3.4180902544231634e-07,4.0
3237344152375e-07 };
    double digital_gain = (sens / 695.0) < 1.0 ? 1.0 : (sens / 695.0);
    double C = noise_model_C[plane];
    double D = noise_model_D[plane];
    double o = C * sens * sens + D * digital_gain * digital_gain;
    return o < 0.0 ? 0.0 : o;
}
```

The entire code snippet can be pasted into a C file, compiled, and executed, and it will dump data that can be used to visualize the generated model by plotting it (for example using a Google Docs spreadsheet).

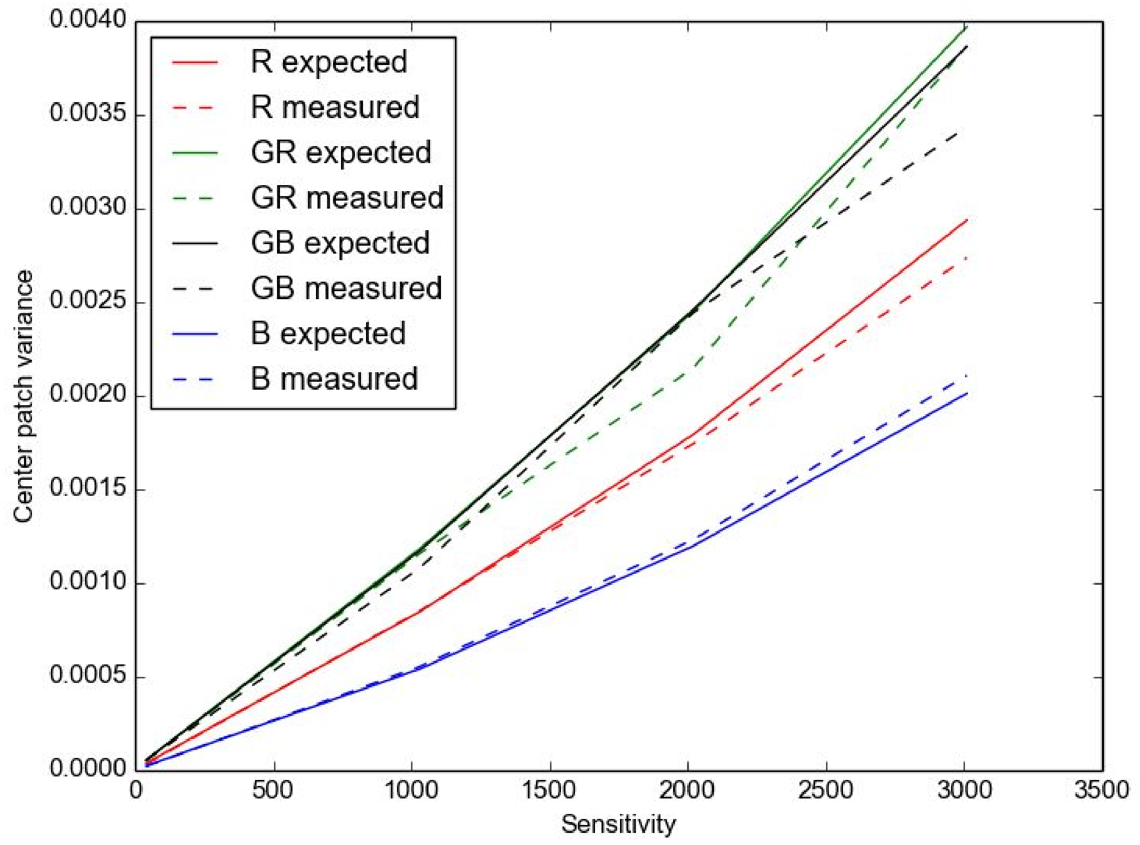
The bolded portion of the C code output can be used in the camera HAL to generate the O,S model parameters for any given sensitivity. The generated code can of course be modified as appropriate to fit within the camera HAL's code style and other requirements.

3. External validation of the DNG noise model

Once a device is up and running with the camera HAL producing the O,S model parameters with each raw shot, one of the automated ITS tests can be used to validate that it's working as expected. This is a "scene 1" ITS test, meaning that it assumes a simple grey card inside a light box as the test scene. The test can be run manually as follows:

```
python tests/scene1/test_dng_noise_model.py
```

The output of this test (aside from an automated pass/fail check) is a plot, `test_dng_noise_model_plot.png`, which depicts the measured variance of a center patch of the grey card in raw shots captured over a range of sensitivities, and compares these values with the variance that is expected at each sensitivity by the DNG noise model in the camera HAL (based on the O,S parameters returned in the capture result objects). A successful run will look as follows:


R expected
R measured
GR expected
GR measured
GB expected
GB measured
B expected
B measured
Center patch variance
Sensitivity