This is a brief tutorial on how to use the image logging utility that comes with the GuiBender package. The image logging is more useful in debugging and monitoring than the regular logging which is provided for your use as well. Learning by examples should be our approach. Therefore, the entire tutorial consists of several examples from the three basic backends. All you need is a quick look a them to understand the image logging.

EXAMPLE 1 – Template matching of a blue circle

After changing doing

Settings().image_logging_level(logging.INFO)
Settings().image_logging_destination("/tmp/imglogs")

and running a unit test about finding a blue circle with something like

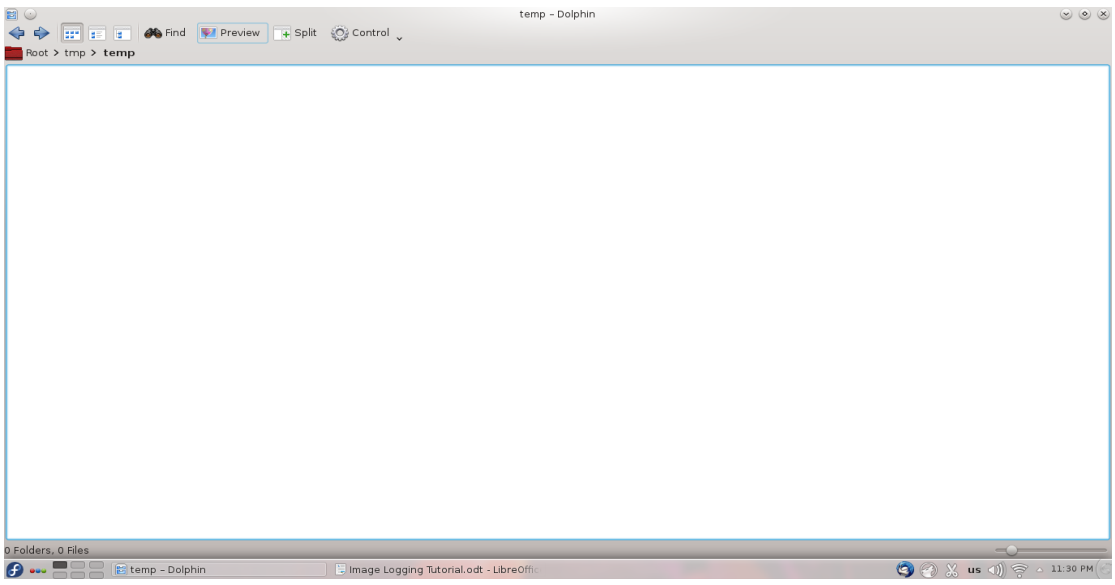
[root@R2 guibender]$ python tests/test_region.py RegionTest.test_hover

we see the following sequence of images appear in the */tmp/imglogs* directory:

1) imglog1-0needle-shape_blue_circle.png:

Here is how to read the name:
imglog1 – step number – one for each image matching performed during the logging
0needle – each step logs the needle (searched image), the haystack (image where search is done), and the hotmap (result from matching, i.e. "where it is warmer")
shape_blue_circle.png – file name used for the needle

2) imglog1-1haystack-noname.png:



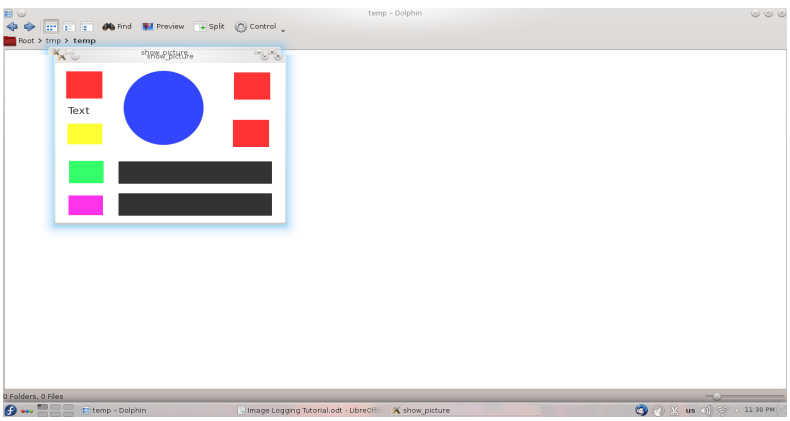
Here is how to read the name:
imglog1 – step number is the same, this is the haystack of this matching case (first during logging)
1haystack – don't mind the numbers, they are only to make the order more intuitive, first you see the needle to know what is searched for, then the haystack to know that was the situation and finally you can go deeper in the debugging process if there are still questions
noname.png – file name used for the haystack (or noname if a screenshot was made)

2) imglog1-3hotmap-template1-0.13444212079.png:

Here is how to read the name:
imglog1, 3hotmap – should be clear by now
template1 – backend used is template matching, 1 stays because this is the first and only match (perhaps you can see it on the upper border – it is almost invisible because it is far below the acceptable similarity)
0.13444212079 – this is the similarity – quite bad – check the haystack to see why

The first step was far from finding any match because the window did not appear yet. But the second matching attempt was far better:

imglog2-0needle-shape_blue_circle.png



imglog2-1haystack-noname.png

And the hotmap for the template matching becomes:

imglog2-3hotmap-template1-1.0.png

This is a 1.0 match. Besides a greyscale hotmap of the similarity across the entire haystack, the location of the maximum is circled to make it easier to spot. Believe me, it seems easy here but some template hotmaps could be quite hairy so this is definitely useful.

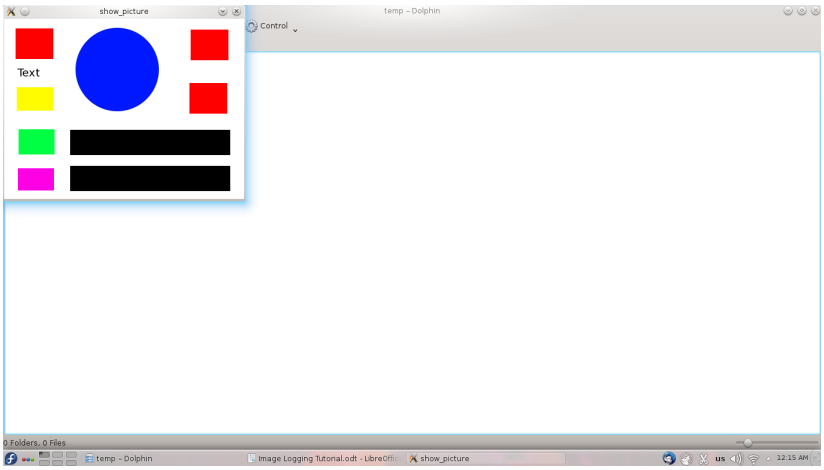
EXAMPLE 2 – Template matching of multiple objects (find_all)

Running the find_all unit test like

[root@R2 guibender]$ python tests/test_region.py RegionTest.test_find_all

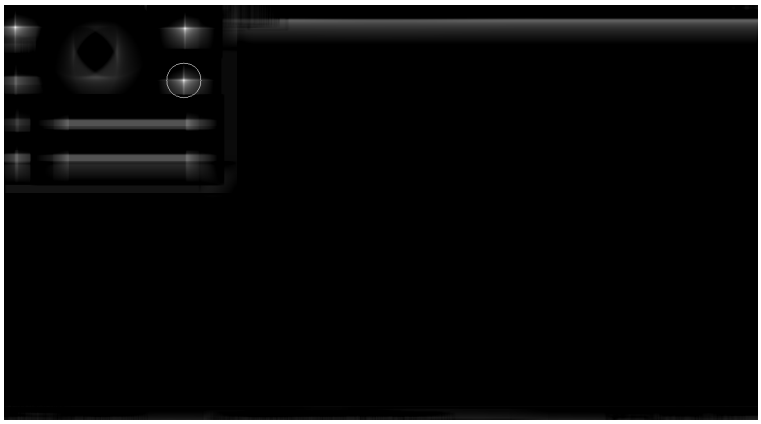
makes several matching steps one of which with a red and another a pink box.

imglog2-0needle-shape_blue_circle.png

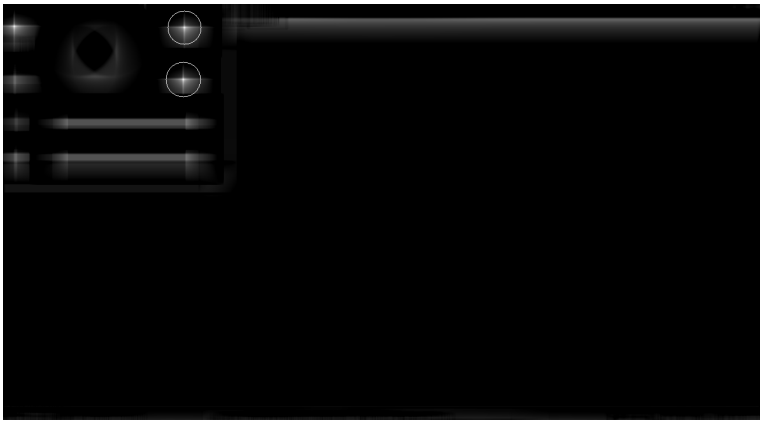


imglog4-0needle-shape_pink_box.png *imglog3-1haystack-noname.png*

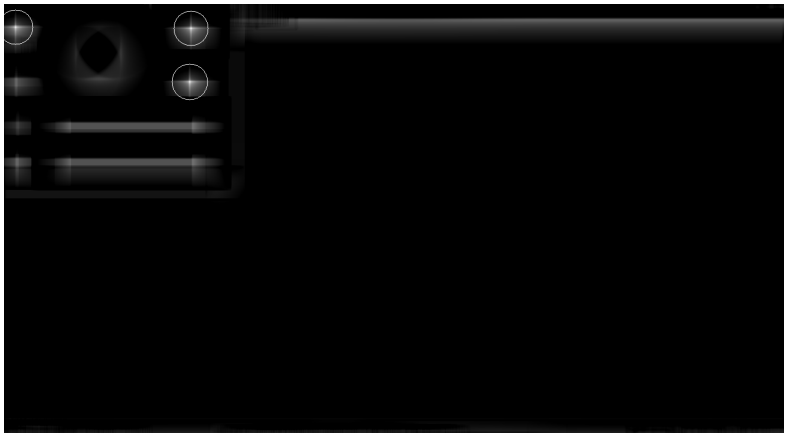
The resulting hotmaps for the red box template matching are now several, one for each match made:



imglog3-3hotmap-template1-1.0.png



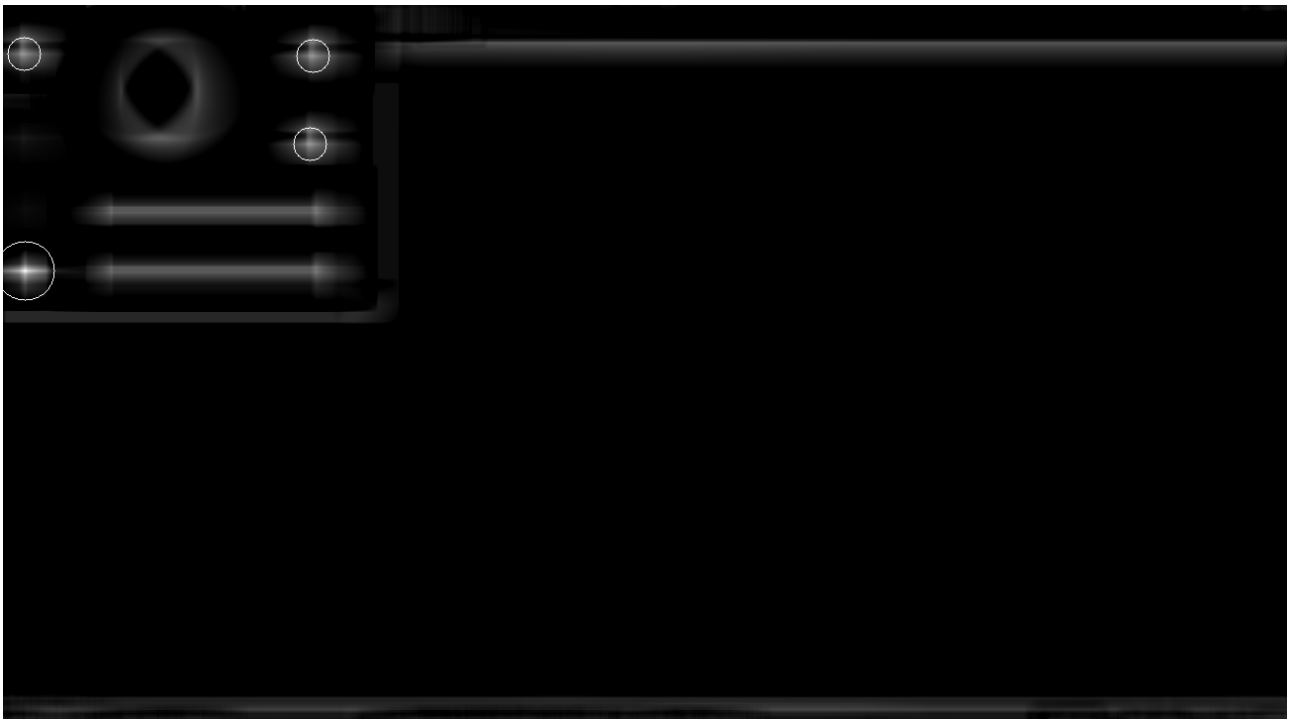
imglog3-3hotmap-template2-0.999999880791.png



imglog3-3hotmap-template3-0.999999403954.png

Observations? The hotmaps for all matches are arranged in a decreasing order of similarity. Also there is a gradation of the circled areas helping to locate the new match and to understand the order of visiting the matches. All circled matches are acceptable, i.e. they are with sufficient similarity. If there are 0 matches to satisfy some similarity there is still at least one hotmap from the find_all with the best unacceptable match (like the first hotmap from example 1).

The pink box is more interesting since it matches differently according to shape or color:



imglog4-3hotmap-template4-0.589033663273.png

This is always reflected in the way they are circled since the radius is proportional to the similarity.

EXAMPLE 2 – Feature matching of viewport transformed object

imglog1-0needle-n_ibs.png *imglog1-1haystack-h_ibs_viewport.png*

The feature hotmap looks quite different but is not any more difficult to understand:

[root@R2 guibender]$ python tests/test_imagefinder.py ImageFinderTest.test_features_viewport

imglog1-3hotmap-feature-0.828729281768.png

1) red dots are detected features
2) yellow dots are matched features
3) green dots are verified features
4) blue dots are projected points

1) and 2) are visible only from DEBUG
3) is visible from INFO logging level
4) from WARN logging level

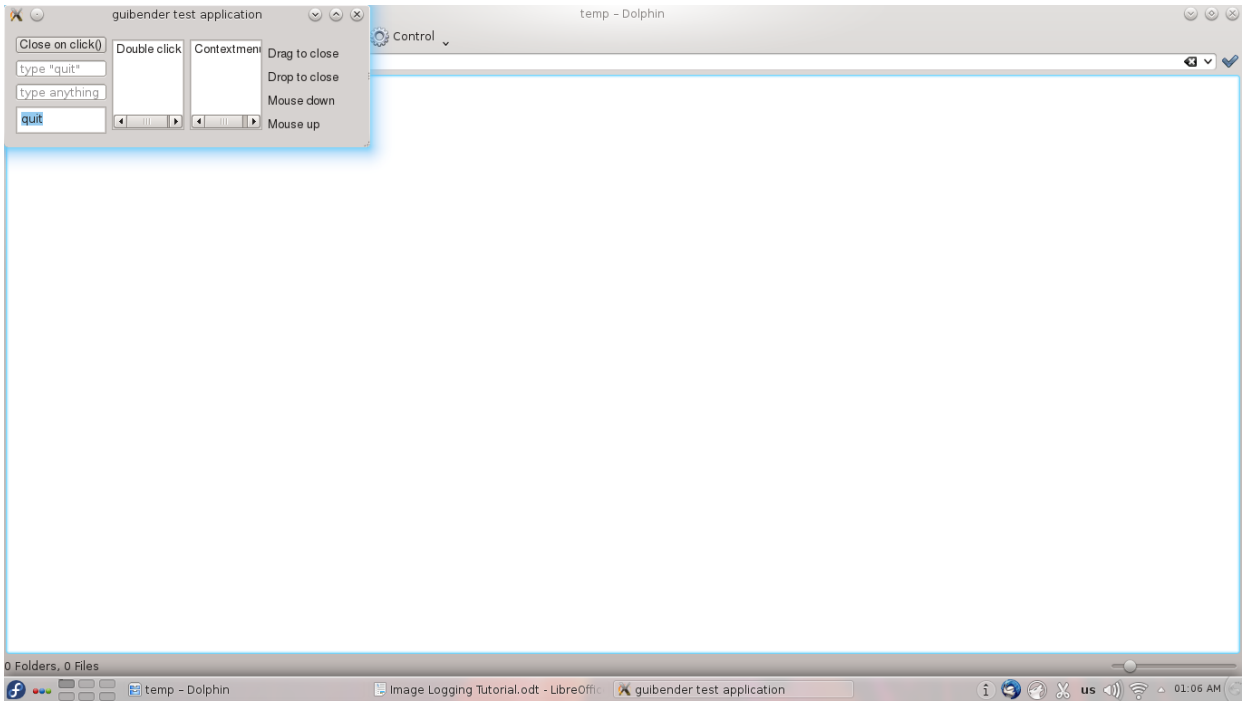
***details*
**3) verification happens through RANSAC and other methods – in a few words these features would meet the projection requirements (valid object transformation)*
**4) a clicking point, a corner point or some other point that is important to the searching*

EXAMPLE 2 –Hybrid matching of a button with a different window theme

[root@R2 guibender]$ python tests/test_region.py RegionTest.test_click

And we get the hardest example in this tutorial - the hybrid method. It is important to know a bit about how the method works in order to understand the image output. Since template matching is too strict and therefore sometimes rigid and feature matching is too flexible and therefore sometimes loose, the hybrid method uses some good aspects from both. It performs a find_all template matching with a lower similarity to remove a lot of noise that would otherwise be distracting for the feature matching. Then however, we apply the more feature matching on each of the remaining choices where template matching would rather fail. The summary: microscale for feature matching and macroscale for template matching. Hence, the logged hotmaps contain both feature and template hotmaps.

imglog2-0needle-qt4gui_button.png	- the needle
imglog2-0needle-qt4gui_button.match	- the needle match settings
imglog2-1haystack-noname.png	- the haystack

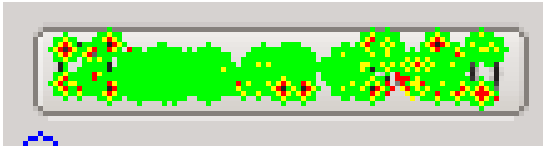


Now when the objective is clear let's check the hotmaps:

imglog2-3hotmap-hybrid-0feature-0.88996763754.png
imglog2-3hotmap-hybrid-0template-0.521731436253.png

Each find_all result has a pair of feature and template hotmaps with their respective similarities.

There is only one spot for feature matching analysis. This analysis gives:

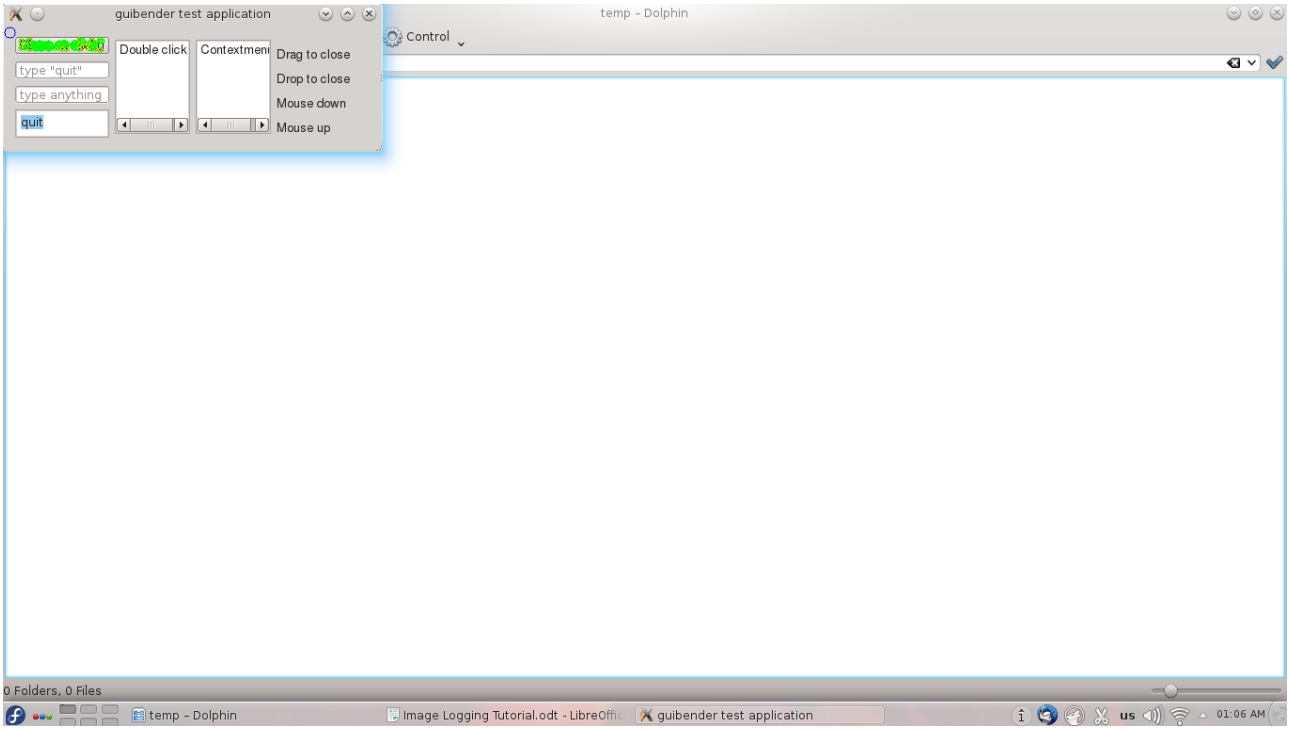


Not quite understandable for this size of image if you don't have the needle visible as well. It can also be easily amended by reducing the logging level.

Usually there are many more hotmap pairs which are sorted according to feature similarity.

The winner receives the simplified hotmap name:

imglog2-3hotmap-hybrid-0.88996763754.png



Ok, this is enough. For the 2to1 hybrid you only need to know that it divides the screen into subregions and performs feature matching in them, i.e. there is a hotmap for each subregion.

Hope this helps you always keep an eye on what is matched and makes your GUI test developing much more satisfying.