Android Camera Rolling Shutter Skew

Document version: 1.0

Document date: 10th July 2017

1. Document Scope

This document details how rolling shutters can distort images and how the nature of these distortions can be used to measure a camera's shutter skew. We also walk through the ITS validation test for measuring this: `test_rolling_shutter_skew.py`.

2. Background

The camera HAL reports its rolling shutter skew, but sometimes it reports it incorrectly, so we need a way of verifying that what's reported is what's actually happening.

Figure 1: LED panel

The piece of equipment driving this test is a 10x10 Image Engineering LED-PANEL V3 (figure 1) with fine control over the frequency of the LEDs. It has a rolling shutter mode, which is what this test is based on. In this mode, only a single column of LEDs is lit at any given instant. The LEDs then sweep from left to right in a fixed amount of time. Because of the mechanics of a rolling shutter, an image of the panel in this mode (if the LEDs are moving fast enough) will produce a slanted line. Based on the properties of this slanted line, we can calculate the shutter skew.

This test *should* be adaptable to other types of LED panels, but no tests have been done to confirm this is the case.

3. Running the Test

3.1. Physical Setup

The phone should be facing the LED panel and should be perpendicular to it (shown below). Any angle in the phone's orientation with respect to the panel will cause inaccuracies in the calculations. The phone should be as close as possible, while keeping the entire 10x10 LED grid in view and still allowing the camera to focus on it.

Figure 2(a): Side view of phone & LED tracer setup

Figure 2(b): Front view of phone & LED tracer setup

3.2. Invoking the Test Script

There are a number of arguments which can be supplied to the test script (the `--help` flag gives the details), but the bare minimum to get it to run is:

```
python test_rolling_shutter.py --led_time=<LED TIME> --device_id=<DEVICE ID>
```

The `led_time` is how long a single column of LEDs is lit for (in milliseconds), and if our setup were the same as figure 2(b), we would write `--led_time=2`, because the blue display reads “2.00ms”. Your phone’s `device_id` can be found using `adb devices`. For example, `--device_id=704KPQJ000684` was used for the setup in 2(a) and 2(b).

3.3. Collecting Data

Before the script is run, turn on the LED panel and set it to rolling shutter mode. A good rule of thumb for the time to start it at is the camera’s reported rolling shutter skew divided by 15 (rounding up to the nearest setting). For example, if the skew reported by your camera is ~32 ms, then 2.0 ms is likely to work well.

When the script is run, make sure the phone is in position. When the first bolded line below is printed to the console, keep the phone as steady as you can (if you’re holding it with your hand):

```
Running vendor 3A on device
Using exposure time of 1000000.0.
Starting capture
Capturing 60 frames with 1 format [yuv]
Finished capture
Rolling shutter skew reported by camera is 32.314464ms
```

When the second bolded line is printed, the phone can be set down.

3.4. Output Data

If the `--debug` flag is given, there will be four directories created under the debug directory (this directory can be user-specified or automatically generated):

- `raw`: contains the original, unmodified images from the capture.
- `mono`: contains monochrome versions of the raw images.
- `contour`: contains thresholded images with the contours drawn in red.
- `scratch`: contains raw images with miscellaneous debugging information drawn on top. If anything seems wrong with the final result, this should be the first directory you check. You should see three things on every frame:
    - Green circles around the LEDs in the largest cluster.
    - A blue rectangle containing all green points.
    - A yellow line fit through the green circles.

Examples for typical images found in these folders are given below. Additionally, the shutter skew reported by the camera will be saved in `raw/reported_skew.txt`.

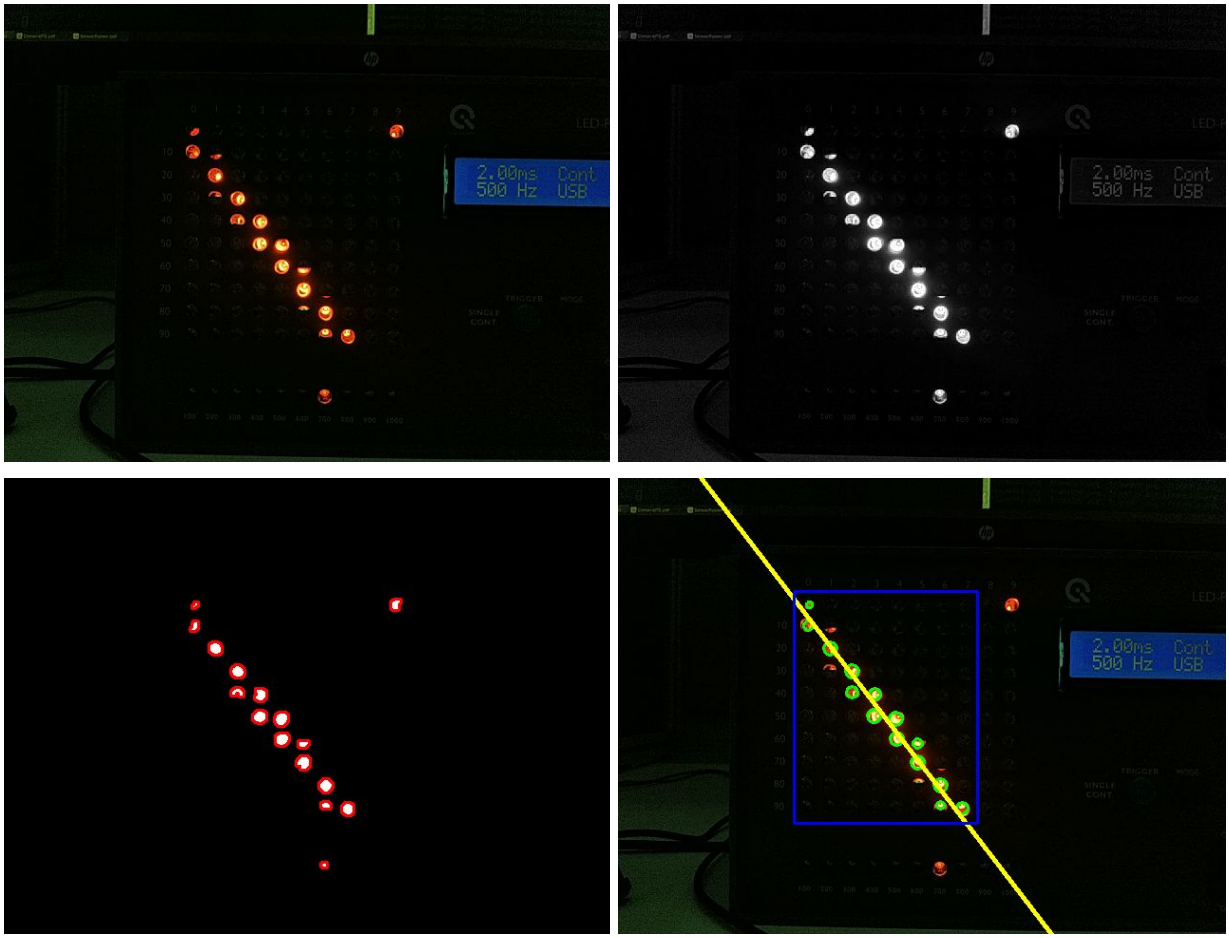

Figure 3: raw image from camera (top left), monochromatic version of raw image (top right), thresholded image with contours drawn in red (bottom left), and raw image with debugging information drawn over it (bottom right).

4. Code Walkthrough

4.1. Data Collection

The `collect_data` and `load_data` functions simply capture the camera frames and reported shutter skew, and load them from files, respectively. The collection is done using the ITS device control methods, with the same manual gain and exposure duration used for all frames.

4.2. Finding Contours

The `find_contours` function first generates a thresholded version of the raw image's red channel (using Otsu's method). Noise is then removed before passing it to OpenCV's `findContours` method.

4.2.1 Assumptions

The primary assumption in place is that the LED panel contains the highest-intensity red light in the image, since the contours are computed using only the red channel. When this assumption is correct

(which it should be in most reasonable setups), it improves the quality of the contours, compared to using all three color channels.

4.3. Clustering

The `proximity_clusters` function produces cluster assignments using a `CLUSTER_DISTANCE` (calculated from the image's size) based on the following criteria: each circle in a cluster is within `CLUSTER_DISTANCE` pixels of at least one other circle in the cluster. This criteria works well for our application because the circles that comprise the line we're trying to cluster should be somewhat continuous and within a somewhat constant distance of nearby circles. The diagram below illustrates what this criteria looks like in practice.

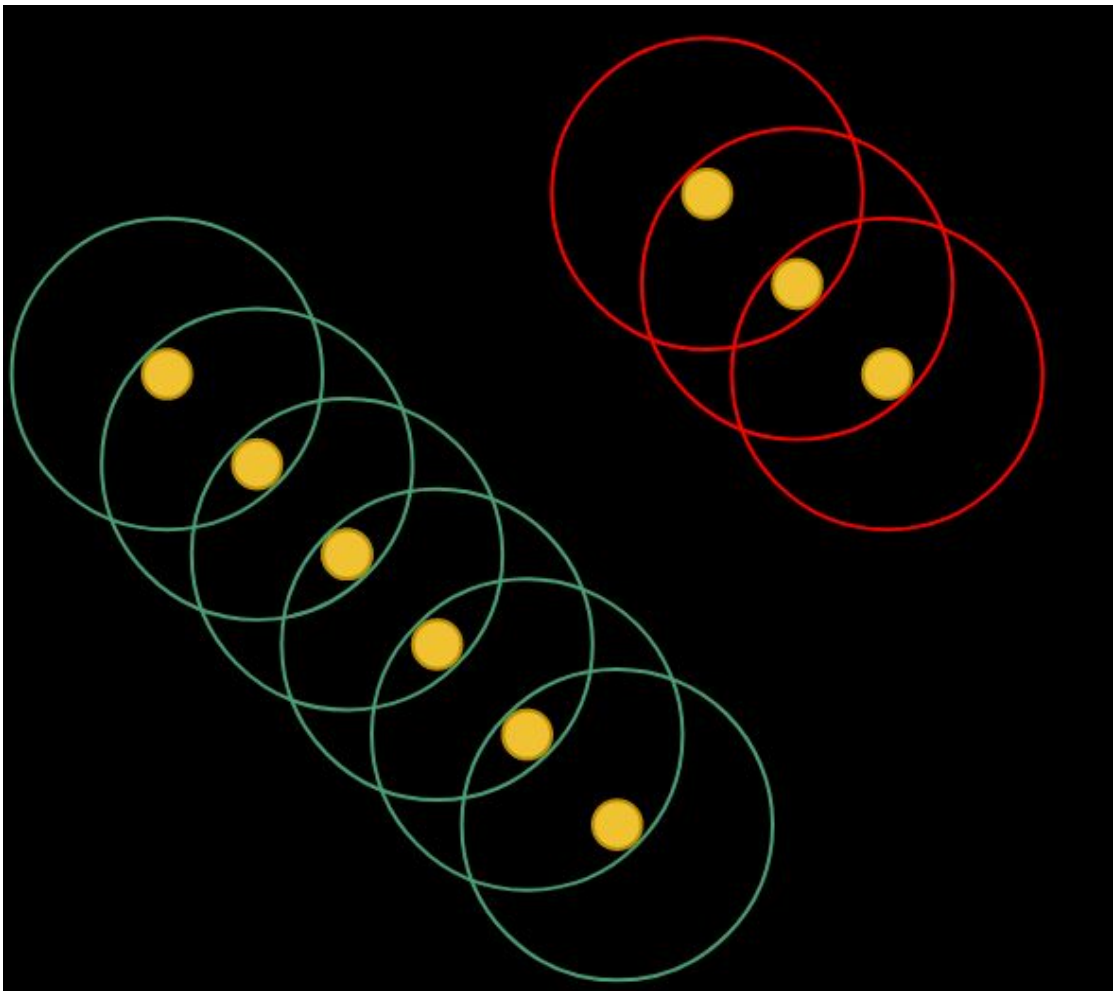

Figure 4: Illustration of Clustering Criteria

Let the circles surrounding each solid circle represent `CLUSTER_DISTANCE`. Because all of the circles with green outer circles are within `CLUSTER_DISTANCE` of one another, they are in the same cluster. The circles with red outer circles are in a separate cluster, because none of them are within range of any of the circles in the green cluster.

Note that not every frame that is captured is used. If there isn't a single "dominating" cluster, the frame is discarded from further calculation. An example of a frame that would be discarded is given below.

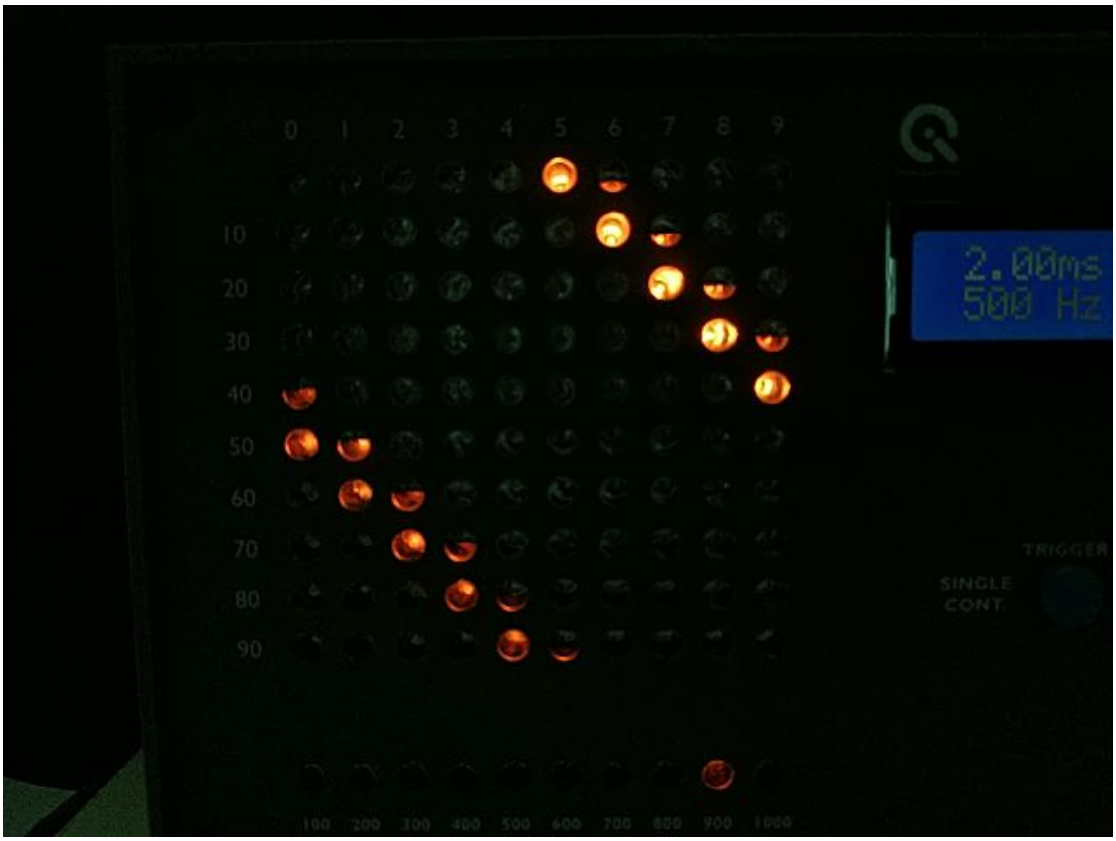

Figure 5: Frame with no dominant line

Above, there are two lines visible on the grid. Although they are both reasonably sized, neither is big enough to give a good line fit, so the frame is discarded.

4.4. Fitting a Line

The largest cluster from the previous step is used to fit a line (the green circles on images in the `scratch` directory represent circles in the largest cluster). This is done by sending the center of every circle to OpenCV's `fitLine` function. That function assumes that every point in the cluster is of equal importance; however, weighting each point by the size of the circle may give better results. This may be worth looking into in the future, but for now, the current model works well enough.

4.5. Finding a Bounding Rectangle

The `find_cluster_bounding_rect` function computes an axis-aligned rectangle (no rotation) by finding the extreme coordinates on the x and y axes and padding the result by the average distance between circles in the cluster.

4.5.1 Assumptions

The main assumption this function makes is that the panel is not rotated in the image.

4.6. Finding the Number of Columns

The number of columns spanned by a cluster is computed using a simple greedy algorithm:

1. All points are treated as if they are projected onto the x axis.
2. Initially, choose the leftmost point in the cluster and set the number of perceived columns to one.
3. While there is another point that doesn't collide with the last chosen point, increment the number of perceived columns.

4.6.1 Assumptions

In this calculation, we assume there is at least one point in the cluster and the panel is not rotated in the image.

4.7. Calculating Shutter Skew

The `calculate_shutter_skew` function composes the results from all operations in previous sections to calculate the shutter skew. First, we'll define some variables (all spatial quantities are in pixels and all temporal quantities are in milliseconds):

$t_{shutter}$: how long it takes the shutter to move from the top of the sensor to the bottom

col : the number of columns spanned by the largest cluster

t_{led} : amount of time a column of LEDs is lit for

$w_{cluster}$: width of the largest cluster

h_{image} : height of the image

m : slope of the fitted line

The first intermediate value we want to find is how many milliseconds each horizontal pixel represents. Because we know t_{led}, we can multiply it by col, and we get the amount of time it took for the LED column to move from the left end of the cluster to the right end. Then if we divide by $w_{cluster}$, we have the ms/pixel quantity we were looking for.

The expression $t_{led} \cdot col$ only gives us the amount of time elapsed within the area of the cluster, but in most cases, the cluster won't be the size of the image. So we use our ms/pixel measurement to extrapolate to the rest of the image. The reason we fit a line to the points on the LED panel is because it allows us more precise measurement than discrete columns. In the previous paragraph's calculation, we were interested in the number of *columns* spanned by the *cluster*, but now we're interested in the

number of *pixels* spanned by the line between $y = 0$ and $y = h_{image}$ (i.e., the full image). And this quantity is given by $\frac{h_{image}}{m}$.

Now, we can summarize the entire calculation like so:

$$t_{shutter} = (\frac{col \cdot t_{led}}{w_{cluster}})(\frac{h_{image}}{m})$$

This function is run for every frame that is captured and the final shutter skew is computed as a weighted average (where the weights for each frame are the percentage of all circles in that frame that the largest cluster contains).

5. Quality of Results

Unsurprisingly, the quality of the setup affects the quality and consistency of the results. An example of the types of results you can expect with an “average” setup (similar to the setup given in 3.1) are given below. Tests were run 5 times for each row.

Camera Settings	Shutter Skew Reported by Device	Average Shutter Skew Measured by Test
640x480, 30 fps	32.315 ms	31.067 ± 1.023 ms
640x360, 60 fps	15.154 ms	14.947 ± 0.216 ms
640x360, 30fps	42.569 ms	41.099 ± 0.140 ms

6. Future Works

There are a few ways in which this test still has room for improvement:

- Detecting rotation of the LED panel and correcting the calculation based on the rotation.
- Calculating the `CLUSTER_DISTANCE` in a more robust way.
- Weighting points by light intensity or size (for when an LED isn’t completely lit in the captured image) during line fitting
- More sophisticated contour finding. Currently, it’s possible for one LED to generate two contours.