Android camera sensor fusion

Document version: 1.3
Document date: 21th November, 2017

- Updated PASS/FAIL specification to +/− 1ms to match CameraITS script.
- Updated example test run to have current checkerboard.pdf

1. Document scope

This document details how camera+motion sensor fusion is defined for Android cameras, and walks through the ITS validation test for this capability: `test_sensor_fusion.py`.

2. Running the test

2.1. Physical setup

When physically moving the camera (which is required when running this test), the only degree of freedom should be rotational movement that is in the same plane as the image sensor, about the camera's optical axis. When looking at the images generated by this test, the center of the image should be stationary between frames, with the overall scene only rotating; no translation, and no skewing or other perspective transformations.

This can be easily achieved by physically constraining the camera's movements with a simple test rig, as follows:

- Take two sheets of cardboard, one on top of the other, and poke a hole through them both with a pin or nail.
- Use something like a split pin or an eyelet to connect the two sheets together through that pin-hole, so that one sheet is able to rotate on top of the other sheet; example connectors:
  http://www.amazon.com/dp/B00B84NRMU
  http://www.amazon.com/dp/B004LWSFAK
- Tape the bottom cardboard sheet to the top of a table; the top sheet should now be able to rotate on top of the table in a constrained manner.
- Tape the phone to the top sheet so that the camera is directly above the center of rotation; now, when the top sheet is rotated, the camera is rotated, with the center of rotation being about the camera's optical axis.
- Mount or suspend a checkerboard target above the camera, for example by taping to the underside of a cardboard box which is placed on the table above the camera. If you have a light box for the ITS setup, then this is convenient to use; simply tape the a printout of the checkerboard chart to the roof of the light box.

Using this simple rig, the only degree of freedom is rotation about the optical axis, which will correspond to the Z axis of the gyro.

2.2. Invoking the test script

The test is run within the ITS infrastructure, which is now a part of the CTS Verifier test suite. Once this is set up properly, the sensor fusion test can be invoked as follows:

```
python test_sensor_fusion.py
```

This generates a large amount of data, and in particular saves all data needed for the analysis portion of the test as files in the current directory. The test can be re-run on the saved data at a later time as follows:

```
python test_sensor_fusion.py replay
```

2.3. Collecting data

When the test is run (without the `replay` argument), it collects both motion sensor data and image data from the device, and requires that the device be physically moved at the right time during this data collection. The test prints the following console output during its operation; note the bolded line:

```
Starting sensor event collection
Running vendor 3A on device
```

```
Capturing 320x240 with sens. 190, exp. time 16.7ms
Capturing 210 frames with 1 format [yuv]
Reading out sensor events
Dumping event data
Dumping frames
Best correlation of 0.001884 at shift of 3.45ms
```

When the bolded line is printed, the user needs to physically move the device in a specific way for a few seconds, characterized as:

- The movement should be at most moderately fast; no rapid or jerky movements. Rotating at a rate of around 10-20 degrees/second is about the limit of what should be done.
- The set of movements should span around 10-15s after the bolded message is printed, to ensure that the actual time of image capture (which lasts for 7s) overlaps with the movements.
- The movements should have a pattern such as “rotate for 1s, then stationary for 1s, then rotate again for 2s, then stationary for 0.5s, ...”. There shouldn’t be a repeating pattern, and the goal is to generate a simple motion trace that will make correlating motion and image sensors easy and robust.

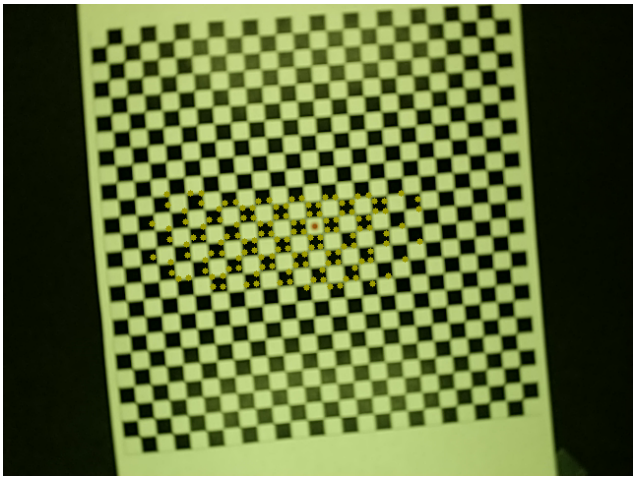
The test takes a little while to run, as it transfers 210 YUV frames from the device to the host PC and dumps them to disk.

2.4. Output data

The script generates lots of output images in the current directory:

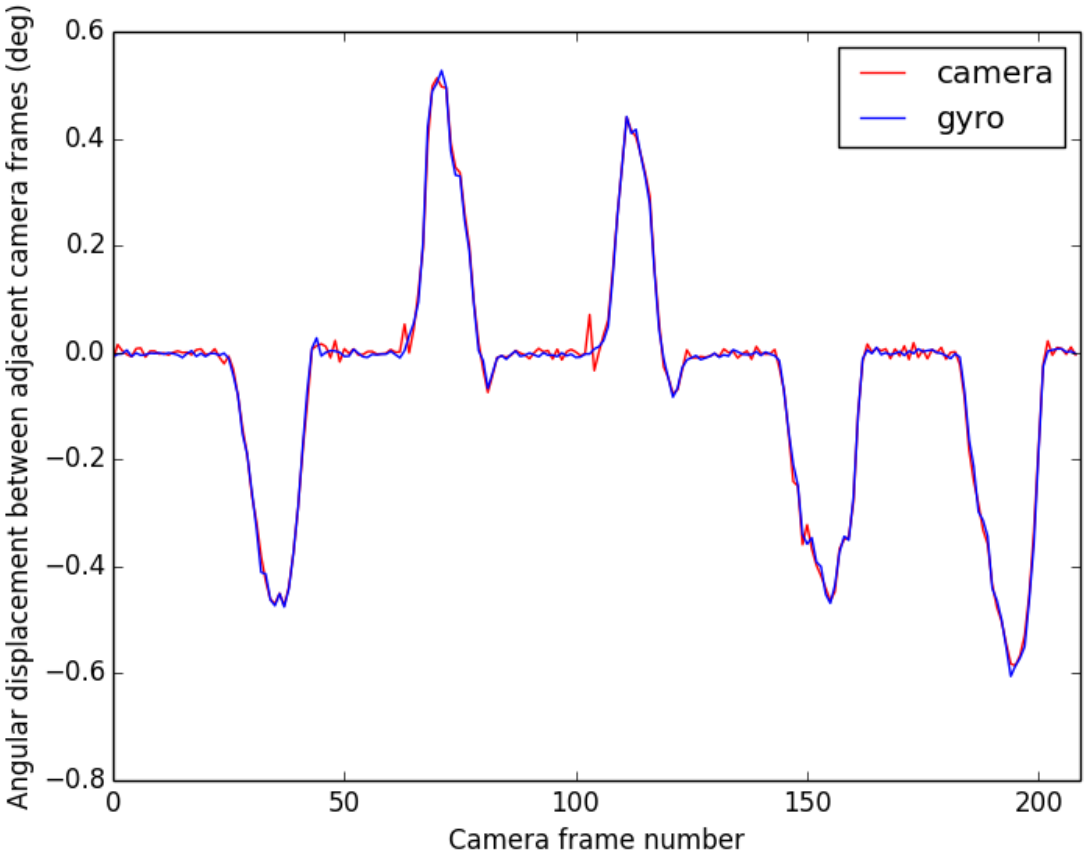
The actual collected data is dumped to `test_sensor_fusion_events.txt` (for the motion events and camera timestamps) and `test_sensor_fusion_frame000.jpg` through `test_sensor_fusion_frame209.jpg` (the 210 image frames captured). When the test is run with the `replay` argument, these are the files that are opened.

An image is also saved that shows the features that were tracked across frames, for the camera’s motion estimation: `test_sensor_fusion_features.jpg`. An example from a test run:



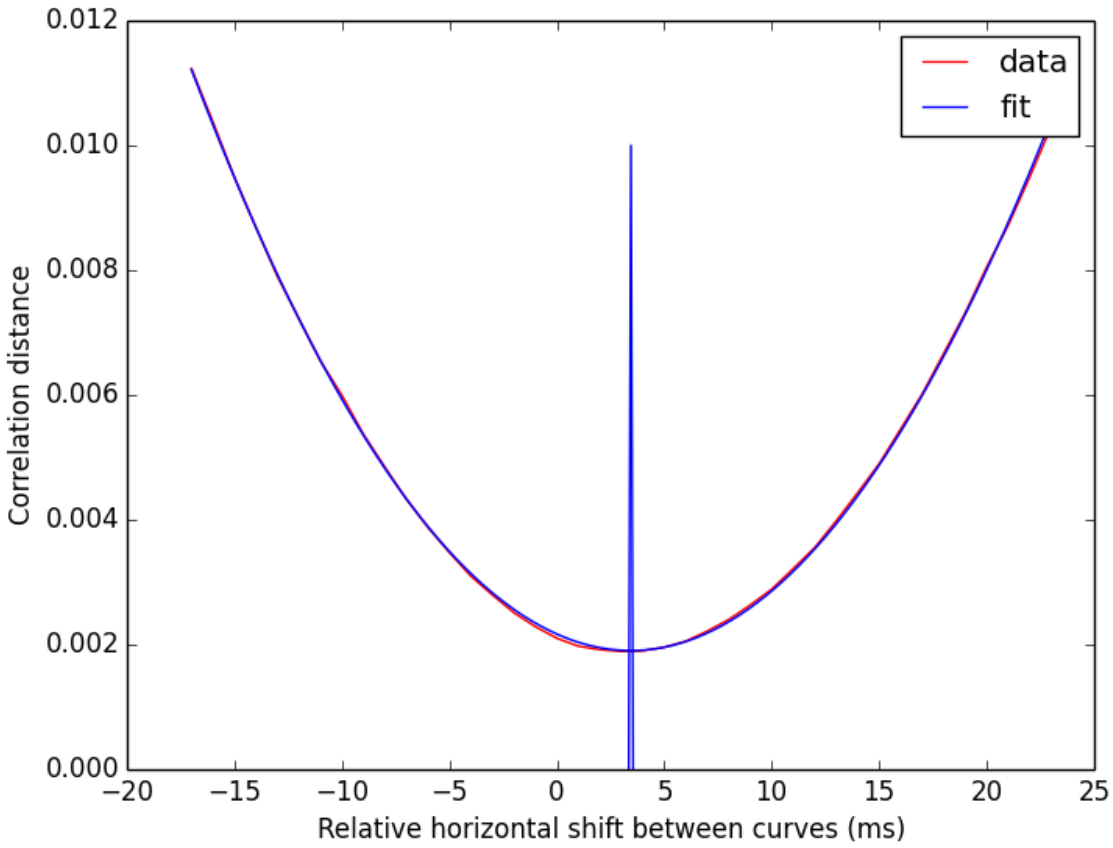
Note that the greenish hue to the image is due to the fact that simple manual controls are used to perform the capture, without worrying about white balance.

The key plot that is generated is `test_sensor_fusion_plot.png`, which shows the angular displacement measured between pairs of adjacent camera frames, as computed by analyzing the camera images vs. integrating the gyro samples:



The motions of the user during the test run are evident from this trace; there were alternating periods of 10-20 deg/sec rotation and periods of no motion.

When this particular test ran, the last printed console line stated that the “best correlation” was at a “shift of 3.45ms”; this describes the process where a range of candidate time shifts are applied to the camera timestamps, and for each time shift the gyro samples between camera frames are integrated to produce a motion trace as is shown in the above figure, with the time shift resulting in the best alignment (i.e. lowest correlation distance as computed by `scipy.spatial.distance.correlation`) being 3.45ms. The test saves one more plot, `test_sensor_fusion_plot_shifts.png`, which graphs the candidate time shifts between the traces on the X axis vs. the computed correlation distance on the Y axis. In this case, it’s visually clear from the plot that the minima is around 3-4ms.



The red curve is the actual data, and the blue curve is a polynomial fit to the red curve, with the actual minima determined from the fit curve to increase precision.

2.5. Test pass/fail

At the end of the test, assertions check that the time shift resulting in the best correlation is within +/− 1 ms, and that the correlation distance in this case is small. If these assertions are not tripped, then the test passes.

2.6. Using different exposure times

The sensor fusion math includes a term for the exposure times of the captured images. A single run of the test will use the same exposure time for all shots. For a device to properly support sensor fusion, this test should pass with any reasonable exposure time value used to capture the images. To verify this, this test should be run 2-3 times, with a different level of scene illumination for each run resulting in a different exposure time being used. Note that the exposure time used for the test is printed to the console when the test is run; it's the bolded line below:

```
Starting sensor event collection
Running vendor 3A on device
Capturing 320x240 with sens. 190, exp. time 16.7ms
Capturing 210 frames with 1 format [yuv]
Reading out sensor events
Dumping event data
Dumping frames
Best correlation of 0.001884 at shift of 3.45ms
```

3. Code walkthrough

This section walks through all of the steps of the `test_sensor_fusion.py` script, including providing details on assumptions and definitions used. A developer who is writing an app that makes use of the sensor fusion feature should be able to understand the basic model that is assumed here, and map it to their own model as needed.

3.1. Data collection

The `collect_data` and `load_data` functions simply capture the camera frames and motion data and store them to files, 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 throughout the 210-frame YUV burst. Frames are captured at QVGA resolution.

3.2. Computing camera rotation samples

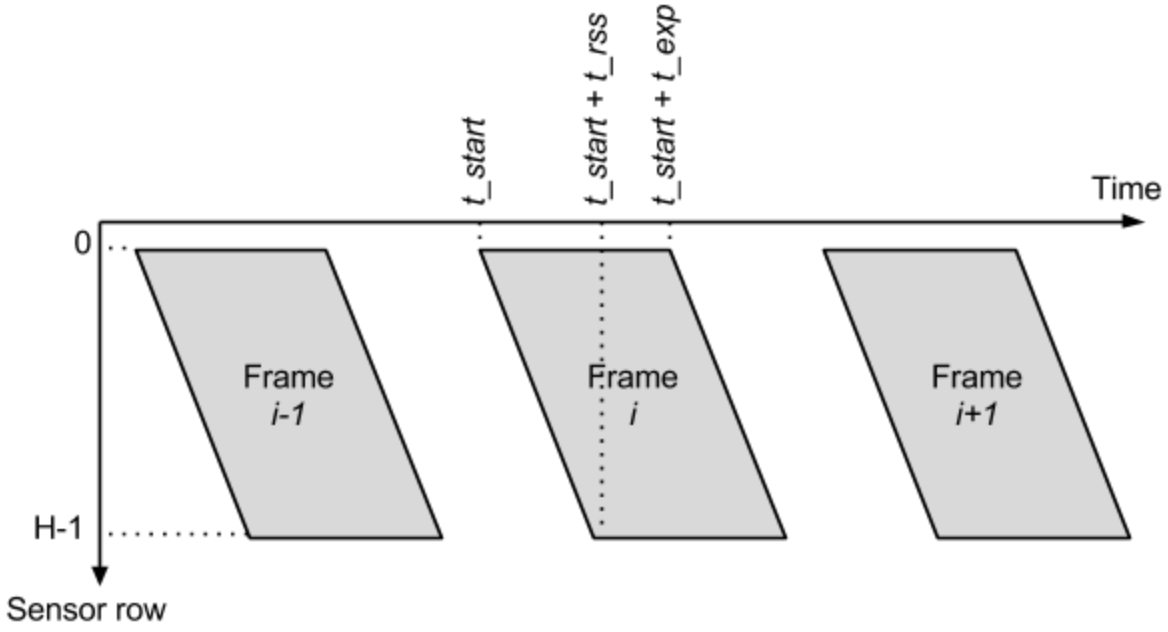
3.2.1. Camera sample times

For each camera frame, there are three time values reported by the HAL:

- ***t_start***: the timestamp of the start of the frame's exposure
- ***t_exp***: the amount of time that each pixel is exposed
- ***t_rss***: the "rolling shutter skew", which is the time difference between the start of the first row's exposure and the start of the last row's exposure

In the stored `test_sensor_fusion_events.txt` file, the triples of camera times are the (*t_start*, *t_exp*, *t_rss*) values for each frame.

The following figure shows how these three values are associated with each frame, where each parallelogram shows the integration window of the first pixel in each sensor row in a particular frame. The sloping sides are due to the rolling shutter; if this were from a global shutter camera, the parallelograms would be rectangles, and *t_rss* would be zero for all frames.



The first step in computing the camera rotation samples is to assign a single time to each frame. This is computed as the "middle" time for that frame, which in the above figure corresponds to a point in the exact center of each parallelogram. The equation for this is:

$$t\_frame = t\_start + \tfrac{1}{2}\, t\_rss + \tfrac{1}{2}\, t\_exp$$

The `get_cam_times` function performs this computation: given N input camera frames, it returns an array of N *t_frame* values.

3.2.2. Camera sample rotations

The `get_cam_rotations` function computes an angular displacement (in radians) between each pair of camera frames; that is, it takes as input the N frames, and returns an array of N-1 rotational displacement measurements *r_cam*.

This function uses OpenCV's feature tracking and optical flow calculations to compute a rotational distance between each pair of frames *i* and *i+1*. The `procrustes_rotation` function contains the math to compute the rotational component of a transformation that maps the set of tracked features from one frame to the next.

3.3.3. Assumptions

This whole process of computing the camera rotation samples makes some simplifying assumptions, in particular:

- The time corresponding to a frame is the "middle time", and aside from this the rolling shutter effect is simply ignored.
- The exposure duration is a part of this formulation, but only in a very simplistic way. To reduce the impact of this factor, the test is assumed to be run in a well-lit environment so that exposure times are relatively short.
- The camera movement is only a rotation about the optical axis, in a plane parallel to the camera sensor.
- The camera movement is assumed to be at most moderate, at most 20 deg/sec rotation, which reduces the impact of not modeling rolling shutter more comprehensively.

3.4. Computing gyro rotation samples

The raw data from the gyro is a sequence of (*t_gyro*, *r_gyro*) samples, where each sample corresponds to an instantaneous rad/sec rotational velocity measurement at a particular moment in time. The gyro data will likely be at a higher rate than the camera data (e.g. 200Hz vs. 30Hz) and will also likely be noisier. In this test, only the Z component is used, as this corresponds to rotations about the optical axis.

The `get_gyro_rotations` function takes the full set of gyro events and the N *t_frame* values as input, and returns N-1 *r_gyro'* samples as output, by integrating the gyro stream between each adjacent pair of camera timestamps to compute an angular displacement. This is a simple Euler integration, and it includes handling the fractional intervals between adjacent gyro samples which fall within different camera frame time windows.

The `get_best_alignment_offset` function iterates over candidate time shifts between the motion sensor and camera data, and for each candidate it applies the shift to the *t_frame* values, calls `get_gyro_rotations` to get the gyro motion trace *r_gyro'*, and then correlates this to the camera motion trace *r_cam*. The time shift (offset) which results in the best correlation is returned (after performing a polynomial fit to the data to find the true minima).

Once `get_best_alignment_offset` is complete, there are two arrays of length N-1 corresponding to the camera and motion sensor traces using the time-shift that results in the best correlation:

- ***r_cam***: angular displacement (rad) between each pair of camera frames
- ***r_gyro'***: angular displacement (rad) between each pair of camera frames

These two arrays should correlate very highly if the device is correctly implementing sensor fusion.

3.5. Plotting the results

The `plot_rotations` function draws a graph showing the *r_cam* and *r_gyro'* values for each pair of adjacent camera frames. If the device is working well with respect to fusion of motion and image sensors, then the two curves will be perfectly overlaid.