Measuring rotor speed using a smartphone camera

M. Le Boucher and T. Faget

(Dated: 13 juillet 2017)

We propose an approach to measure the rotation speed of a rotor using a mark and a smartphone. We introduce a general method to approximate the rotation speed based on various videos of a rotating rotor, which leads to relative errors under 0.5% and converges over time. We then present a mobile application we designed to use the camera device to take videos and handle the calculation.

I. SPATIAL ANALYSIS

A. Description

This method is based on a spectral analysis of fixed pixels on the image, hereinafter called tracked pixels. These are spread over a region of interest which is pointed by the user, and within which the mark is assumed to pass every turn.

We store the tracked pixels' luminosity value over time, and then perform a Fourier's transform on the data. The fundamental frequency of the obtained spectrum is the frequency at which the observed phenomenon is repeated. It is worthwhile to note that the phenomonon in itself does not matter; even the motion blur, generated by the high rotation speed, passes through the region of interest at the same frequency as that of the mark itself. We then apply a Hamming window over the spectrum to slightly reduce noise peaks.

The approximated rotations per minute (r.p.m.) r are then computed by

$$r = \frac{n_f \times f_s}{N_s} \,,$$

where n_f is the order of the fundamental frequency, f_s the sample frenquency and N_s the amount of samples.

Figures (1) and (2) present the results of this method for two rotation speeds, each time filmed by two different devices at different sample rates close to 120 frames per second.

Figure (3) shows how the error evolves with respect to sample time.

B. Limits

As predicted by the Nyquist–Shannon sampling theorem, this approach can only efficiently approximate the rotation speed of rotors rotating at less than half the sampling rate. For a sampling rate of 30 frames per second, the upper limit turns out to be 900 r.p.m., which is slightly under the aimed interval (around 1,600 r.p.m.). We therefore need a sampling rate of at least 60 — preferably 120 — frames per second to fall within the correct range.

C. Results

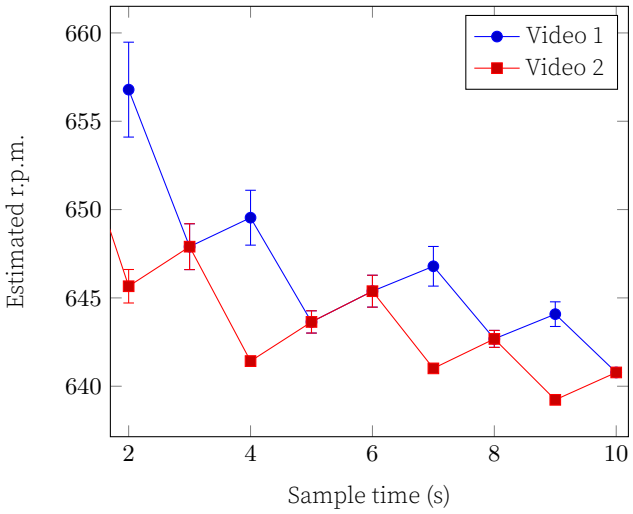

Figure 1: Estimations for a real rotation speed of 640 r.p.m., based on one single tracked pixel within the region of interest. Vertical bars show relative errors.

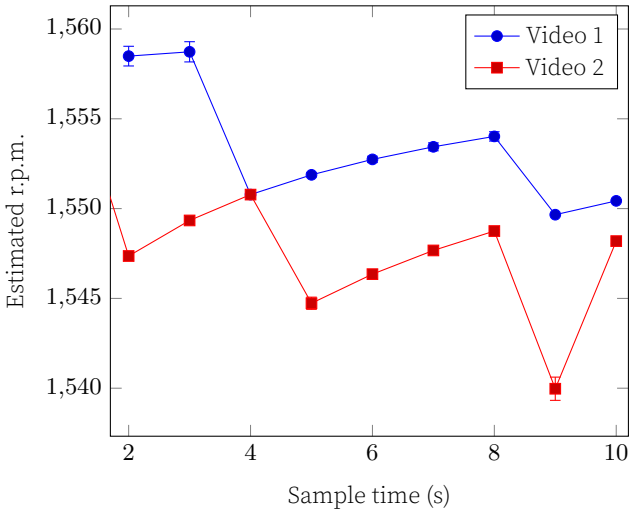

Figure 2: Estimations for a real rotation speed of 1,550 r.p.m., based on 10 tracked pixels within the region of interest. Vertical bars show relative errors.

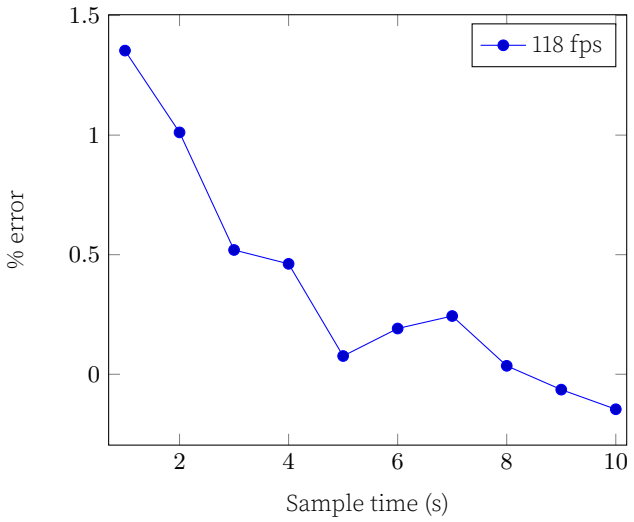

Figure 3: Mean error percentages with respect to sample time, based on 8 videos of rotors rotating at various speeds, between 580 and 1,550 rotations per minute. At 118 fps, the error percentage never goes beyond 2% and seems to converge to approximately 0%.

II. IMPLEMENTATION : AN ANDROID APPLICATION

A. Logic and use cases

We implemented the logic described in (**I A**) as an Android application, with the aim of requiring the minimum knowledge and intervention from the end user.

Two main approaches are currently available, dealing with the two main situations we have to deal with ; either the device supports high speed recording or not. Since no proper API currently allows to easily manipulate data flows at high speed rates, such as 120 frames per second, high speed recording has to be handled in a slightly different way than "normal" recording, namely at 30 frames per second. In the latter case, we are able to process the frames live.

However, due to many hardware issues, and as Android does not provide a direct access to the camera buffers, we could not achieve a sufficiently satisfying approach to process while recording at high speeds. To overcome the problem, we first record an entire video file (encoded in MP4), and then process the file. The mathematical logic remains exactly the same. Additionally, Android does not provide efficient ways to access frames from a video file either, which significantly reduces the efficiency of the method.

By default, the whole calculation is processed once every second, though this parameter can easily be changed in the fragment's private attributes. An UI access to the calculator's parameters can easily be implemented using those attributes.

B. Math helper

The proposed implementation of the calculator (interfaced as `SpeedCalculator`), called `SpatialAnalyzer`, uses several dedicated mathematical methods that were specifically designed for our own use. In particular, they mostly use `DoubleBuffer`s, a specific Android optimized implementation of buffers.

Those methods are fully unit tested under `src/test` and reliable for our purposes. Although the current performances are satisfying, some of them could probably be optimized.

C. Extensions

The application is designed to be as generic as possible, and is fully documented. The fragmented architecture handled by a single master activity and the numerous interfaces make it easy to extend the current logic or even implement entirely new approaches. The architecture of the application is detailed in figure (**4**).

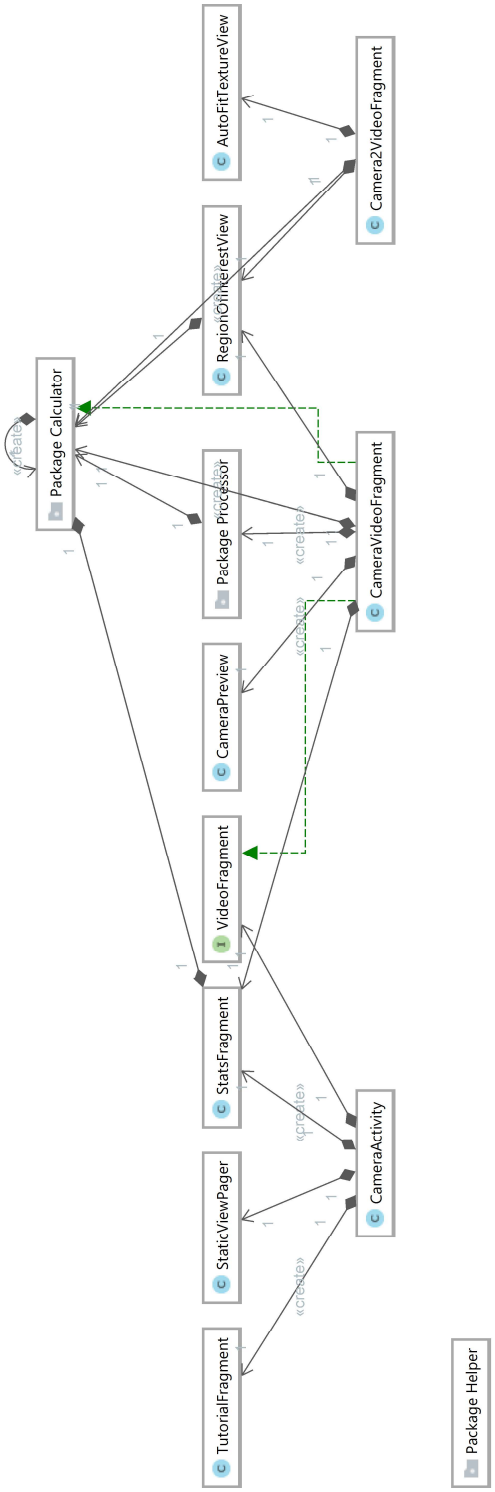

Figure 4: Architecture of the application.