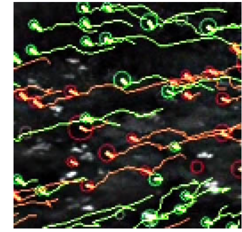
Real-time interest point tracking



Description	Interest point detection and tracking with FFTW and CUDA
Period	Spring 2012 – Fall 2012
Languages & Libs	C++, FFTW, CUDA
Tags	Computer Vision

As mentioned in an other article about *omnidirectional cameras*, my *Master's Thesis'* main topic was real-time interest point extraction and tracking on an omnidirectional image in a challenging forest environment. I found OpenCV's routines mostly rather slow and running in a single thread, so I ended up implementing everything myself to gain more control on the data flow and threads' dependencies. The implemented code would simultaneously use 4 threads on CPU and a few hundred on the GPU, executing interest point extraction and matching at 27 fps (37 ms/frame) for 1800×360 pixels ($\approx$0.65 Mpix) panoramic image.

In the topic of *visual odometry*, interest point extraction and matching is a crucial step, and it is no surprise that this is a widely investigated topic. Some methods are optimized for speed where as others are optimized for most accurate results. Also there are several different types of errors (interest point drifting, incorrect matches etc.), and how much the system relies on a-priori knowledge which was extracted at previous frames. Also some additional sensors such as Inertial Measurement Unit (IMU) might be available, and the most computationally efficient rely on accurate a-priori information and reliable readings from an IMU to have very good initial guesses on where interest points should be found from the next frame. However these methods are less general, and are more difficult to generalize and to apply in new situations. In addition, many visual odometry systems use stereo vision instead of just a single camera, and this also introduces its own difficulties but also advantages.

In my thesis and in this article I only focus on monocular vision algorithms with no a-priori knowledge of the interest points, environment or the motion between frames. Also I won't describe standard interest point algorithms such as SIFT or SURF, because they are well described in Wikipedia and are difficult to robustly apply in real-time scenarios. They are computationally intensive to extract and match, and don't have many adjustable parameters to ease the computational cost. Of course there are numerous others algorithms as well, but typically they have quite poor scale and rotation invariance.

The algorithm I came up with starts by converting the observed circular omnidirectional image into a equiangular panorama of desired resolution, such as 1800×360 pixels. This aspect ratio corresponds approximately to the *field of view* of the camera, which in this case was $360° \times (11 + 62)°$. These dimensions have small prime factor decompositions ($2^3 \times 3^2 \times 5^2$ and $2^3 \times 3^2 \times 5$), which is important for an efficient FFT calculation. The original circular image and the resulting panorama can be seen in Figure 1.

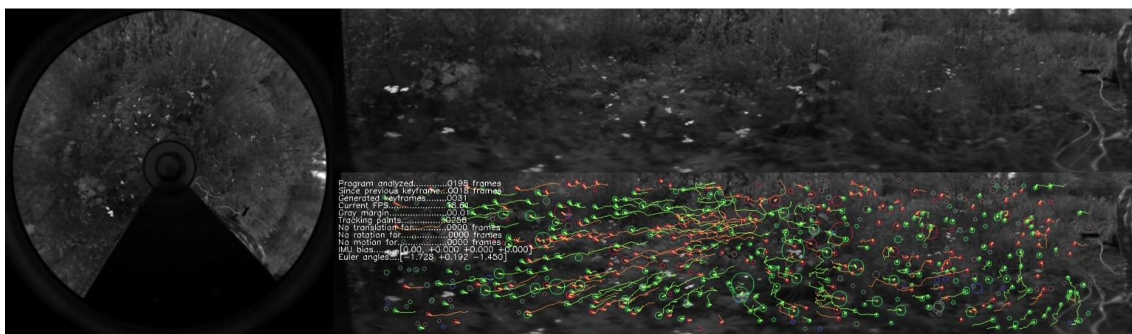


Figure 1: Example captured omnidirectional image, and the generated panoramic view.

The interest point extraction is based on the standard Laplacian of Gaussian (LoG) blob detector. Its benefits include excellent scale and rotation invariance, able to robustly divide interest points into two different classes (dark and bright blobs), are very stable and are plenty to be found in natural environments. Perhaps the only down-side is that it gives unstable response also along edges, but these are easy to detect and filter. One simple alternative would have been some of the numerous corner detectors, but they have only one class of detected points, are prone to image blur and corners are more difficult to observe in nature.

The *SIFT* detector approximates Laplacian of Gaussian via Difference of Gaussian, which is based on calculating differences between images which have blurred with Gaussian blur at different scales. An other fast detector CenSurE (Center Surround Extremas for Realtime Feature Detection and Matching) is based on approximating DoG by spending effort in calculating slanted integral images, which can be then used to calculate convolutions of arbitrary large areas in a constant time, but also has its own trade-offs.

The extraction algorithm was implemented a quad-core processor in mind, and this is reflected in all steps. Most of the time is spent on processing the most detailed level of the image pyramid, and the smaller pyramid levels aren't that significant. The first two steps are shown in Figure 2. At the first step the LoG is calculated at four different scales in parallel, utilizing the FFTW library. LoG could be implemented as a standard convolution over the image, but this would make it suitable only for small kernels. Instead when calculating the convolution via forward Fourier transform, element-wise multiplication and inverse Fourier Transform the runtime doesn't depend on the chosen scale σ. Also scaling the response by a constant is done for "free", because the applied mask can be pre-multiplied by the desired amount.

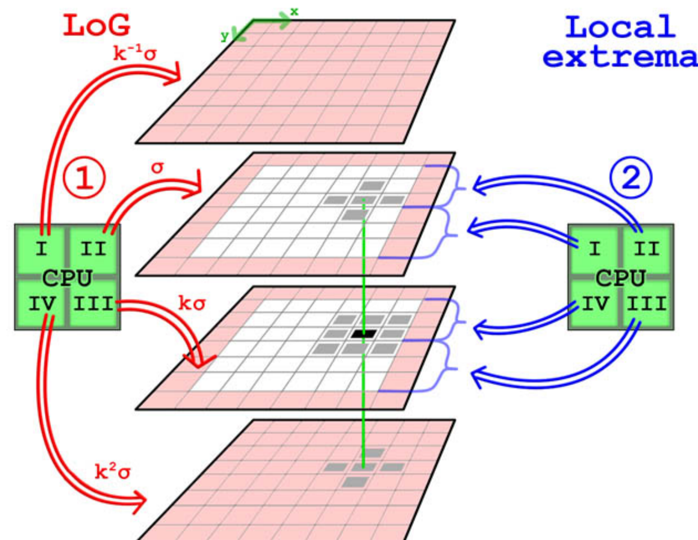


Figure 2: First two steps of the extraction steps, and how the task is executed parallel in four threads.

Once LoG responses have been calculated, the local scale-space minima and maxima are searched. Only the middle two layers can be searched, and also they have a margin at their borders. These margins are indicated by the pink color in Figure 2. Each pixel is compared against its 5 + 8 + 5 neighbors (indicated in gray), to determine if all of them are either smaller or larger than the pixel's value. The pixel is first compared against its neighbors within the same scale, to better utilize caches of the processor. To keep all four cores busy, each core is responsible for finding local extrema from one half of an image. Found extrema need to be filtered for edge responses and those which have too low local contrast.

Once interest points have been found, descriptors can be extracted. Unfortunately my novel algorithm haven't been published yet, and thus I won't reveal all the details yet. But I can say that the overall descriptors are extremely fast to calculate, affine-invariant(!), highly robust and their matching fits well in the CUDA programming model. The number of descriptors is $10\times - 20\times$ of the number of interest points, but a very good heuristic exists to efficiently ignore most of the implausible matches with little extra cost. This is mostly true for the case when a relatively high-frame rate video feed is analyzed.

In the implemented tracking system, GPU is used to match frames 1 and 2 while CPU is busy extracting features from frame 3. Thus the frame rate is determined by the slowest component of the system. This depends on the image resolution and various other parameters, but typically these are quite balanced.

I have ran extensive comparisons of this algorithm against the popular *SURF*, and found out my algorithm gaining a lot lower error rates in different matching scenarios. The used dataset was the well-known Leuven, which has a good set of different image distortions such as blur, viewpoint change and zoom + rotation. Example pairs of images can be seen in Figure 3.

Figure 3: Example pairs of images from the Leuven image set.

Since the descriptor matching is based on a voting scheme, it proved to be a very robust in matching task. Two different ways of matching were evaluated. The first one is based on matching each interest point to its closest match, and the other one is based on threshold matching, where a point is considered matching to which ever point if this similarity threshold is exceeded. The first criteria is evaluated on a "1 - precision" and correct positive matching rate, and the other one is on false positive rate versus false negative rate. On the second graph EER stands for "equal error rate", in which the probability of false negative is equal to the probability of false positive.

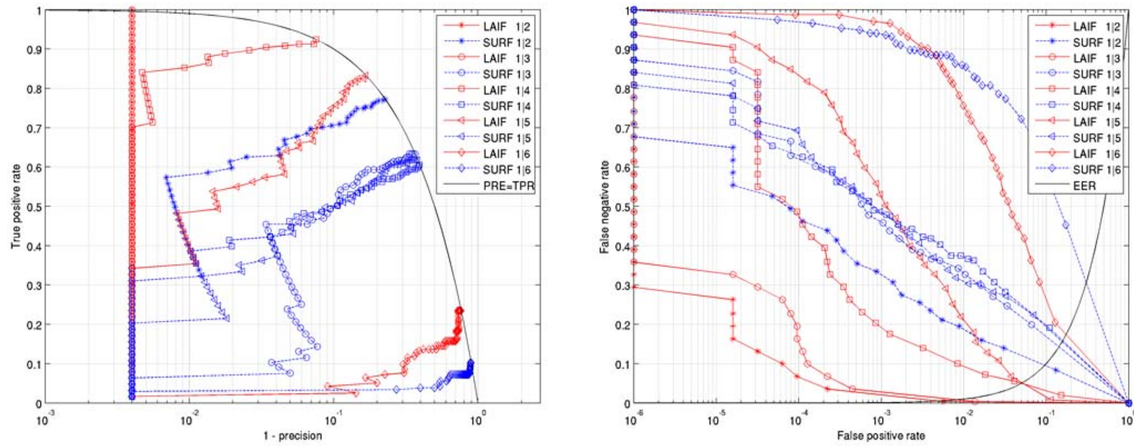


Figure 4: Receiver operating characteristic curves for boat series.

The performance for boat sequence is shown in Figure 4. In all scenarios the novel algorithm gained better accuracy than SURF, especially for those images which had the least difference in scale. The graffiti sequence resulted in similar results, as can be seen in Figure 5. I hope that in future I'll get this algorithm properly published, and then it can be reviewed by also other peers in the scientific community.

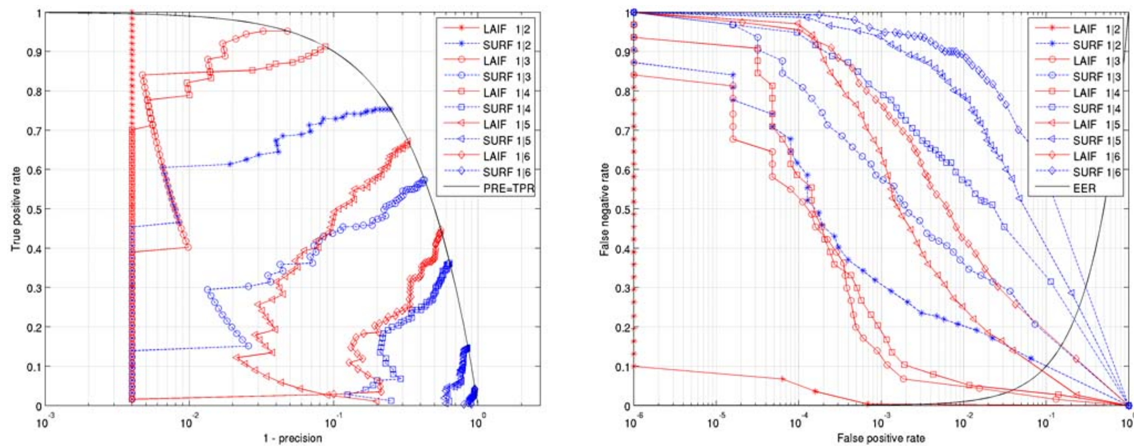


Figure 5: Receiver operating characteristic curves for graffiti series.