Accelerating The Optimal Trade-Off Circular Harmonic Function Filter Design on Multicore Systems

Anubhav Jain
Center of Excellence for Parallelization & Optimization
Tata Consultancy Services, New Delhi
anubhav.jain1@tcs.com

Amit Kalele
Center of Excellence for Parallelization & Optimization
Tata Consultancy Services, Pune
kalele.amit@tcs.com

ABSTRACT

Optimal correlation filters are widely used in signal processing and pattern recognition applications. Correlation filters are a set of synthesized spatial filters that produce controlled response with sharp peaks. While providing excellent discrimination capabilities correlation filters offer shift, rotation and scale invariance for 2D images. Correlation filters are optimized to enhance the recognition of consistent parts while suppressing the varying patterns. Synthesizing the correlation filters for pattern recognition applications involves several complex mathematical operations and requires high computation resources especially for high resolution images and videos. In this paper, we show that near real time performance can be achieved for the design of the OTCHF filter with help of optimization and parallelization on multicore GPUs and CPUs.

Keywords

Correlation filter, Circular Harmonic Function Filter, Muticore CPU, GPU, Performance Optimization, Parallel Computing, HPC

1. INTRODUCTION

Correlation filters finds their application in variety of pattern recognition applications such as image and video watermarking, fingerprint, iris and face detection. Their ability to discriminate under several variations such as shift, scale, in-plane rotations, occlusion makes them extremely suitable for such applications and classification tasks. These filters are also robust to noise and illumination variations .

Correlation filtering is a process of correlating a digital image or a signal with a precomputed template (filter) optimized to return expected response [12, 3]. The expected magnitude response in correlation filtering are sharp peak(s) in the correlation output at locations where there is a match between the template and the signal satisfying the constraints of the template design. Advanced correlation filters have been applied with significant success in image recognition applications [16, 6, 2]. This success is mainly attributed to:

- The advent of the modern high-speed multicore processors with the massive parallel processing abilities which can meet the computational demands imposed by correlation based methods.
- The correlation filters can be optimized for the desired response for a set of predetermined constraints.

Correlation filters are designed with the help of a set of training data containing images or frames. This training data represents the anticipated set of distortions yielding pre-determined responses to these training images.

In this paper, we consider the Optimal trade-off circular harmonic function filters (OTCHF), which are designed to account in-plane rotation distortion in pattern recognition application such as face recognition [13, 3], target detection, iris recognition etc [1].

The design of OTCHF filter allows us to specify the desired response for a range of in-plane rotations. The optimization task involves maximizing the expected correlation filter output level at the origin where the match occurs, while minimizing the correlation function levels elsewhere. The optimization procedure of the template considers all or at least some of the possible variations of the pattern to recognized, these variations include rotation, scaling and translation.

For pattern recognition applications whose input is a video, the number of frames and the resolution of the videos could impose huge computational workload. The training procedure is amenable to parallel processing. The modern day CPUs and GPUs are extremely powerful machines, equipped with multiple compute cores, they are capable of performing multiple tasks or same task on multiple data in parallel. Exploiting their parallel processing capabilities, with performance optimization techniques and usage of optimal libraries for FFTs and linear algebra routines [8, 7], could lead to many fold improvement in the performance.

In this paper, we present an optimal and parallel implementation of design of the OTCHF on Intel and Nvidia multicore platforms and report significant improvements in the performance.

ICPE'16, March 12 - 18, 2016, Delft, Netherlands

ISBN 978-1-4503-4080-9/16/03…$15.00

DOI: http://dx.doi.org/10.1145/2851553.2851579

[1]The work reported in this paper was carried out for a video water marking application

2. RELATED WORK

Several methods have been proposed for obtaining different kinds of distortion invariance, most of them based on the use of filters partially matched to the target.

For rotation invariance, a suitable approach is to use the circular-harmonic functions (CHF's) of the image $f(r,\theta)$ in polar coordinate domain. Rosen and Shamir [12] proposed rotation invariant circular-harmonic component filter for automatic target detection.

Ryan et. al. [6] developed a scale-invariant correlation filter design based on the Mellin radial harmonic (MRH) transform. Minimum average correlation energy (MACE) MRH filters can approximates a user-specified scale response. Some applications where multiple scale responses are required, multiple correlation filters are used to account scale changes. Each correlation filter providing a unique scale response.

The GPUs are widely used in audio, image and signal processing and in correlation filtering [5, 14, 15] and [10]. However, to the best of our knowledge, a parallel implementation of the OTCHF filter design for pattern recognition on GPUs is not reported in the published literature.

3. OPTIMAL TRADEOFF CIRCULAR HARMONIC FUNCTION FILTER

Applications such as face recognition, fingerprint recognition, digital watermarking and other common pattern recognition tasks requires classification performance to be invariant to small in-plane rotation, shift and scaling. The optimal tradeoff circular harmonic function filters (OTCHF) or the correlation filters are designed for in-plane rotation distortion, allows us to specify the desired response for a range of in-plane rotations and also enable shift invariance. Certain level of scale invariance can also be achieved by appropriate training. The trade off parameters for this filter are noise variance and peak sharpness.

An optimized correlation filter would yield similar correlation outputs in response to test images that are from the same class as the training images while providing distortion-tolerant correlation outputs.

We utilize the circular harmonic decomposition in the frequency domain rather than in the space domain. We used finite impulse response (FIR) filter design methods to determine the CHF coefficients. Specifically, the luminance channel is used for computing correlation filter. In this paper, we have implemented the correlation filter for a video which consists of several frames where each frame is considered as an image. The steps involved in the filter design are as follows:

1. Compute the two dimensional Fourier transform $F(u,v)$ of each training image $f(x,y)$.
2. Compute the index matrix which maps the Fourier transformed image $F(u,v)$ to the polar coordinates to obtain $F(\rho,\phi)$, where $\rho = \sqrt{(u^2+v^2)}$ and $\phi = tan^{-1}(\frac{v}{u})$.
3. Compute the harmonic function $F(\rho)$ by operating Fourier transom along the ρ axis on $F(\rho,\phi)$.
4. Define the 'desired correlation function matrix by setting '1' at locations the response should be maximum and '0' at other location in the matrix where the response must be minimal.
5. Obtain the optimal circular harmonic function (CHF) weights/coefficients (C_k) by computing Fourier transform of the above matrix.
6. Compute P_{FOM}, the figure of merit.
7. Compute the filter harmonics $H_k(\rho)$ [2] with inputs matrices P_{FOM}, C_k, index matrix and training images, $F(\rho)$ etc. as

$$H_k(\rho) = \lambda_k \cdot \frac{F_k(\rho)}{P_{FOM}(\rho)}$$

where $\lambda_k = C_k / \int_0^\infty |F_k(\rho)|^2 / P_{FOM}(\rho)\rho d\rho$

For the given m training images, sequence of matrix operations are performed to get the filter harmonics $H_k(\rho)$. We assume that each frame consist of $n \times n$ pixels. We assume that the matrices P_{FOM}, C_k and index matrix ID are computed and stored. The algorithm for computing the correlation filter $H(\rho,\phi)$ is as follows:

1. for $j = 0$ to $n-1$
   (a) j^{th} harmonic of each frame (=j^{th} column) is extracted and a $n \times m$ temporary matrix $tmpF_k$ is formed as depicted in Figure 1.

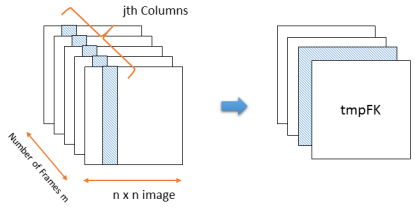

Figure 1: Formation of tmpFK matrices

   (b) Compute $(n \times m)$ matrix $\text{TP}_2[i,k] = \frac{tmpF_k[i,k]}{P_{FOM}[i,k]}$, element by element division.
   (c) Compute $(n \times m)$ matrix $\text{TP}_3[i,k] = \text{TP}_2[i,k] \times ID[i,k]$, element by element multiplication.
   (d) Compute complex conjugate transpose $tmpF_k^*$.
   (e) Compute the product $V = tmpF_k^* \times \text{TP}_3$.
   (f) Compute inverse of matrix V as V^{-1}
   (g) Compute $\text{TP}_4 = \text{TP}_2 \times V^{-1}$
   (h) Extract the j^{th} column of the C_k and compute j^{th} column of filter harmonic $H_k(\rho) = \text{TP}_4 \times C_{kj}^*$, where * denotes complex conjugate.
2. end for
3. Compute the correlation filter $H(\rho,\phi)$ by taking the inverse Fourier transform of $H_k(\rho)$ and converting it back to cartesian form.

The $H_k(\rho)$ computation involves frequent matrix-matrix, matrix-vector operations and matrix inversions (done using standard factoring techniques). For larger, higher resolution and longer duration videos and images, it becomes a cumbersome computational task. We observed that for a 49 frame

Table 1: Time consumed by each step

Steps	Time in seconds
step(a) tmpFk Formation	3.791
step(b) matrix TP_2 computation	1.950
step(c) matrix TP_3 computation	0.292
step(d) conjugate transpose	0.157
step(e) compute V	39.225
step(f) compute V^{-1}	1.013
step(g) compute TP_4	27.429
step(h) compute $H_k(\rho)$	1.177
Total	**74.34**

video (with resolution of 2112×2112 pixels), the time spent for computing $H_k(\rho)$ on the single core sequential implementation was approximately 75 sec ($> 90\%$ of the over all time). The sequential implementation was developed using `OpenCV` library [9] and `FFTW` library [4]. The table 1 represents the breakup of time consumed by individual step(from step (a) to step (h). These operations are amenable for parallelization and it is possible to achieve significant improvement in the execution speed.

4. PARALLEL IMPLEMENTATION

With the advent of multicore CPUs and many core GPUs, compute intensive applications have much to gain with parallel computing. The GPUs, which were primarily designed for graphical operations, are also proved to be equally effective in general purpose computing. Figure 2 represents general architecture of modern multicore CPUs and GPUs.

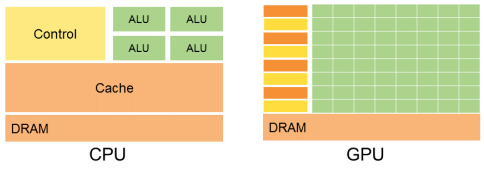

Figure 2: Multicore CPU-GPU architecture

As shown in the figure above, the GPUs have large number of light weight cores compared to multicore CPUs. Unlike CPU cores, the GPU cores are designed to carry out same instruction at a time but on different data. This enables huge data parallel through-put. On the other hand, CPUs have much more powerful cores, which are capable of carrying out different tasks at the same time at very high speeds. A typical CPU-GPU setup is shown in Figure 3. The CPU works as a master and offloads compute intensive work to GPU. The data transfer between CPU and GPU happens over PCI bus, which was often proved to be a performance bottleneck. However, this has not been the case with today's PCI express buses.

With this brief overview of today's multicore systems, we now present our approach for parallel implementation of correlation filter $H(\rho, \phi)$. In the following section, we describe the GPU implementation of the algorithm (1) mentioned in the previous section.

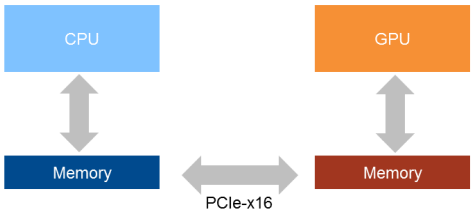

Figure 3: CPU-GPU over PCIe

4.1 Experimental Setup

In this section we present the details of all the hardware setup (CPUs/GPUs) used for the performance tuning and running the benchmarks. All the performance figures, reported in this paper, were obtained on the following hardware.

- GPU systems:
  - **Device1:** The Nvidia's Fermi $C2075$ GPU with 1.1 GHz 448 cores, 5.5GB RAM.
  - **Host1:** The Intel Westmere 2.93 GHz 4 cores with 24GB RAM.
  - **Device2:** The Nvidia's Kepler $K20x$ GPU with 796 MHz 2496 cores, 5GB RAM.
  - **Host2:** The Intel Ivy Bridge 2.1 GHz, dual socket, 6 cores/socket, 16GB RAM.
- Multicore CPU system: The Intel Xeon E5 2650 v2, Ivy Bridge 2.6 GHz dual socket, 8 cores/socket, 24GB RAM.

4.2 Correlation filter on GPU

The correlation filter computation involves number of matrix operations, which are extremely suitable for GPU computation. In this section we present the details of our implementation and measures taken for performance optimization. This implementation was specific to Nvidia platforms and was carried out in CUDA C. Though many CUDA kernels were implemented from scratch, some of the matrix computations were implemented with the help of routines/functions available in the highly optimized cuBLAS library for GPUs. The parallel algorithm implemented on the GPU is as follows:

1. Partition the total frames into $nPart$. Each partition is processed in sequence. Then for each part do:
2. Create n OpenMP threads. Each thread extracts m/n columns from each input frame to form m/n $tmpF_k$ matrices, where m is the width of input frames, and creates m/n number of $tmpF_k$ matrices in parallel.
3. Create nSt CUDA streams.
4. The $tmpF_k$ matrices are copied in batches to GPU asynchronously in each stream.
5. Carry out steps (b) to step (h) on each batch of $tmpF_k$ in each of the nSt stream.
6. Each stream computes a batch of $H_k(\rho)$ columns.

The following points were the main highlights of the implementation.

- **Accelerating matrix operations with cuBLAS:** The cuBLAS is a highly optimized CUDA C library for the linear algebra and matrix operations provided by the Nvidia. The $H_k(\rho)$ computations require matrix-matrix multiplication (step (e) and step (g)), matrix inversion in step (f) and matrix-vector multiplication in step (h). The cuBLAS also supports batch mode for matrix-matrix multiplication (`cublas<` t `>gemmBatched()`). The matrix inversion was computed using `cublas<` t `>getrfBatched` (computes LU factorization) and `cublas<` t `>getriBatched` (computes inverse of matrices with forward and backward triangular solvers) respectively.

- **Multi-level parallelism for higher GPU utilization:** To exploit the full potential of large number of the cores on GPU, it is required that the problem must be broken into large number of fine data parallel computations. It is observed that the `for loop` in the step 1 is amenable to parallelization. Multiple columns from the training images in step (a) were extracted and multiple $tmpF_k$ matrices were formed in parallel. This was achieved using multiple threads with the help of OpenMP on host or CPU.

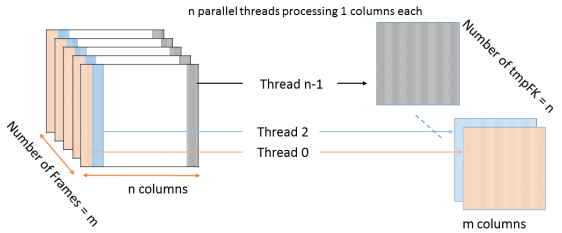

Figure 4: Parallel formation of tmpFK matrices

Similarly multiple $tmpF_k$ were processed in parallel i.e. the operations from step (b) to step (h) were carried out on multiple $tmpF_k$ concurrently on GPU. This was termed as batch processing. Secondly, each step within the loop was also parallelized. The matrix operation in step (b) to step (h) for each $tmpF_k$ are carried out in parallel. This two level parallelism effectively consumes huge compute power of the GPU.

- **Minimizing CPU-GPU data transfers:** Though today's PCI express buses are fast and support very good data transfer bandwidth, but they still fall short of compute speed and main memory (i.e. RAM) access speed by huge margins. The algorithm required the $tmpF_k$ matrices in step (b) and step (e), which in tern require same data transfer to GPU twice. For a high resolution video or large number of training images, this becomes and expensive operation. To avoid this twice data transfer, two copies of $tmpF_k$ matrices were maintained in the GPU. This significantly improved the performance but at the cost of double memory foot print.

- **Multi-Stream Computations:** The GPUs are typically high latency devices. But they support multiple stream computation, which allows hiding latencies by overlapping computations with data transfers. Streams can be imagined as concurrent compute pipelines which enable better utilization of the GPUs. To minimize the data transfers between host and device, two copies of $tmpF_k$ were maintained. The data was further divided into multiple batches and processed in multiple streams as shown in Figure 5. All the stream are launched simultaneously which results in overlapped computations and data transfers across streams. This multi-stream implementation provided further boost in the over all performance.

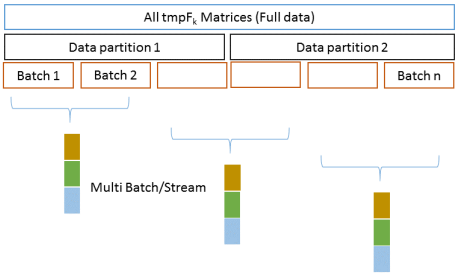

Figure 5: Multi-Batch & Multi-Stream computational

- **Coalesced data access:** The global memory load/store efficiency on GPU mostly depends on the data access pattern of the application. If the neighboring threads accessed global memory in strides then it results in poor load/store efficiency, which translates into poor performance. The data for $tmpF_k$ matrices, P_{FOM} matrix and index matrix ID is arranged to achieve coalesced data access pattern. This resulted in $\approx$ 85% global load/store efficiency.

With all the above performance optimization, a significant speed up was achieved for correlation function computation. We report $\approx 35\times$ gain on Nvidia's $C2075$ GPU (see Figure 6).

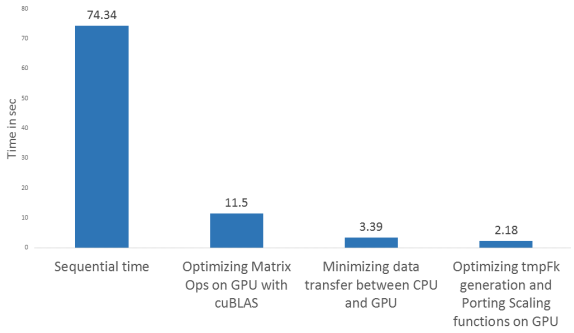

Figure 6: Performance on C2075 GPU

The modern GPUs are extremely capable machines. Figure 6 illustrates how well correlation filter design problem scales for above discussed optimizations.

4.3 Correlation filter on multicore CPU

Modern day CPUs are also equally efficient parallel processing machines. Equipped with less in number but more powerful compute cores, these CPUs also deliver high performance. We have also carried out parallel implementation on Intel multicore system. The `pthread` library [11] was used for multi-threaded application and the `OpenCV` library [9] was used for various image processing and matrix algorithms.

The parallelization approach adopted here was straightforward. The loop `for j = 0 to n-1` described in section 3 was parallelized with `pthreads`. The total number of frames, $tmpF_k$, were equally divided among all the available compute threads. Each thread processes $tmpF_k/N$ frames, where N is the number of threads. Further, parallelization of individual steps (a) to (h) was also implemented. It was observed that the overall performance degraded when inner multi-threading was enabled. This can be attributed to the lack of compute resources to accommodate nested parallelism. The readings presented here were obtained with internal parallelization turned off.

The following figure illustrates that correlation filter generation time decreases almost linearly as we increase number of the compute threads.

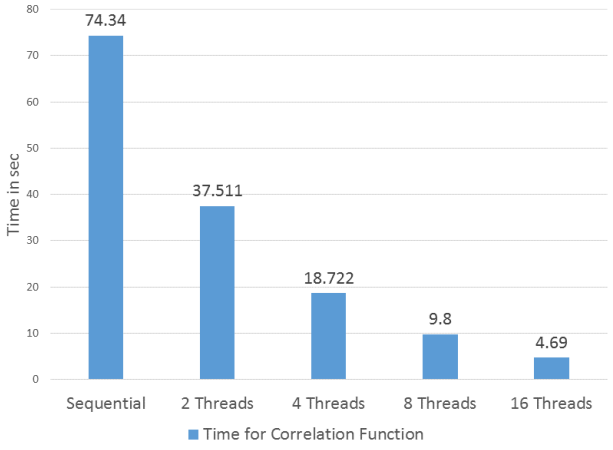

Figure 7: Performance on 16 core CPU

Compared to the sequential execution, scale up of $\approx 16\times$ was achieved on 16 physical core CPU system.

5. PERFORMANCE RESULTS

The results presented in the section 4.2 were obtained on the older Fermi C2075 GPU. The optimized GPU code was then executed on the latest Kepler series K20 GPU. This further boosted the performance. Figure 8 summarizes overall application performance gain achieved with parallelization. It also includes the energy savings while computations along with compute time.

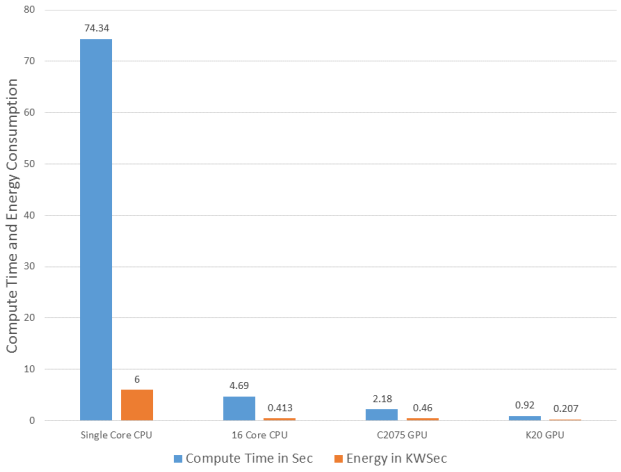

Figure 8: Performance gained on different multicore systems

6. CONCLUDING REMARKS

The OTCHF filters are widely used in many applications. A fast implementation would be desirable for large number of applications in pattern recognition, signal and image processing, which does not seem to be reported in previously published literature. The implementation reported in this paper can be easily adopted for other applications in correlation filtering.

We considered the OTCHF filter design problem on modern day multicore systems such as CPUs and GPUs. Both platforms delivered high performance with parallel processing and performance optimizations. We achieved $\approx 16\times$ and $41\times$ speed up on CPUs and GPUs respectively. Although GPUs performed better than 16 core CPU, huge effort was required in terms of changes done at design level and implementation level. Performance optimizations like exploiting two level parallelism, multi-stream computations and minimizing data transfers were the key factor in achieving the high performance on GPUs. On the other hand, parallelization on CPU required minimalistic code changes as compared to the GPU implementation.

The computational cost, in terms of energy consumption, was also drastically reduced with compute time. This enables scope for doing more simulations with in same duration and same cost, which effectively provides better results.

We conclude this paper on the note that, modern day CPUs and GPUs are extremely capable machines, however best results can only be obtained with performance optimizations.

7. ACKNOWLEDGMENTS

The authors are thankful to Srinivas Chalamala, TCS Innovation Lab Hyderabad, for many insightful discussions on the topic of pattern recognitions and its application in water marking.

8. REFERENCES

[1] A. M. B. V. K. Vijaya Kumar and R. Juday. Correlation pattern recognition. Cambridge University Press, 2005.

[2] A. M. B. V. K. Vijaya Kumar and A. Takessian. Optimal tradeoff circular harmonic function correlation filter methods providing controlled in-plane rotation response. In *IEEE Transactions On Image Processing*, volume 15, pages 84–89. IEEE, June 2000.

[3] M. B. V. K. Vijaya Kumar, Savvides. Correlation pattern recognition for face recognition. In *Proceeding of IEEE*, volume 94, pages 145–152. IEEE, November 2006.

[4] FFTW. Fftw library. www.fftw.org/, 2014.

[5] P. Irofti. Gpu parallel implementation of the approximate k-svd algorithm using opencl. In *Proceedings of European signal processing conference*, volume 15, pages 84–89. EUSIPCO, June 2014.

[6] R. A. Kerekes and B. V. K. V. Kumar. Correlation filters with controlled scale response. *IEEE Transactions on image processing*, 15(7):4012–4015, July 2006.

[7] Nvidia. cublas library. https://developer.nvidia.com/cuBLAS, 2008.

[8] Nvidia. cufft library. https://developer.nvidia.com/cuFFT, 2008.

[9] OpenCV. Opencv library. http://opencv.org/, 2008.

[10] K. Picos and V. H. Diaz-Ramirez. Object tracking under non uniform illumination conditions. GPU Technology Conference, 2014.

[11] Pthreads. Pthread library. https://computing.llnl.gov/tutorials/pthreads/, 2014.

[12] J. Rosen and J. Shamir. Circular harmonic phase filters for efficient rotation-invariant pattern recognition. *App. Opt.*, 15(5):2895–2899, November 1998.

[13] M. Savvides and B. V. K. V. Kumar. Efficient design of advanced correlation filters for robust distortion-tolerant face recognition. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, volume 15, pages 45–52. IEEE, June 2003.

[14] F. Trebien. An efficient gpu-based implementation of recursive linear filters and its application to realistic real-time re-synthesis for interactive virtual worlds, July 2009.

[15] M. J. Yousri Ouerhani and A. Alfalou. Fast face recognition approach using a graphical processing unit gpu. https://hal.archives-ouvertes.fr/hal-00782740/document, 2001.

[16] H. H. A. Yuan-Neng Hsu and G. April. Rotation-invariant digital pattern recognition using circular harmonic expansion. *OSA*, 21(22):4012–4015, November 1982.