EFFICIENT REALTIME FPGA IMPLEMENTATION OF THE TRACE TRANSFORM

Suhaib A. Fahmy[§] *, Christos-Savvas Bouganis*[§] *, Peter Y.K. Cheung*[§] *and Wayne Luk*[‡]

[§] Department of Electrical and Electronic Engineering
Imperial College London, Exhibition Road, London, SW7 2BT, England.
email: {s.fahmy, ccb98, p.cheung}@imperial.ac.uk

[‡] Department of Computing
Imperial College London, 180 Queen's Gate, London, SW7 2AZ, England
email:wl@doc.ic.ac.uk

ABSTRACT

The Trace Transform is a novel image transform that is able to exhibit useful properties such as scale and rotation invariance and occlusion robustness. As a result, it is particularly suited to a variety of classification and recognition tasks including image database search, token registration, activity monitoring, character recognition and face authentication. The main obstacle to the widespread use of the transform is its high computational complexity. This has precluded a detailed investigation of transform parameters. This paper presents an architecture and implementation of a Trace Transform engine on a Virtex-II FPGA. By exploiting the inherent parallelism in the algorithm and the use of optimised functional blocks, a huge performance gain is achieved, exceeding realtime video processing requirements for a 256 x 256 image.

1. INTRODUCTION

The study of image recognition relies heavily on properties in 2D shape and texture. Indeed, the extraction of features from an image has been used extensively in image classification and matching. However, many of these feature extraction methods focus primarily on properties seen from the human perspective. It is, though, useful to expand the horizon, using features which may not have meaning to humans, but which perform well in characterising complex images. With this in mind, Kadyrov and Petrou proposed the Trace Transform in [1] and developed it further in [2, 3, 4]. The transform is a redundant representation of an image, from which features can be extracted. It is a generic transform in the sense that the mathematical definition is extensible.

This work was partially funded by the UK Research Council under the Basic Technology Research Programme (GR/R87642/02) and by the EPSRC Research Grant (EP/C549481/1).

The strength of the transform lies in its ability to extract features that are robust to affine transformations, occlusion and even non-linear deformations. In fact, careful selection of the transform functionals [2] can allow not just robustness, but recovery of transformation coefficients, which would allow an affine-transformed image to be returned to its original state.

The Trace Transform has proved to be a very powerful tool, showing excellent results in image database search, industrial token registration, activity monitoring, character recognition and face authentication [5, 6]. The primary obstacle to further application investigation has been the computational complexity of the algorithm.

FPGAs provide an ideal platform for accelerating the Trace Transform through exploitation of the inherent parallelism existent within the algorithm. Furthermore, the flexibility of the platform suits the generic nature of the transform, since alternative functionals can be swapped in an out with ease. Acceleration of the Trace Transform allows for further investigation of the transform in terms of both functionals and applications.

We present in this paper the first hardware implementation of the Trace Transform. Our architecture is:

- fast: with performance exceeding real-time processing requirements;
- simple: occupying a very small area on the FPGA;
- flexible: allowing us to investigate Trace functionals thoroughly.

2. ALGORITHM

2.1. Overview

It is important before considering our implementation, that we introduce the algorithm briefly so as to clarify the principles that will be discussed in this paper. The Trace Transform of an image is a transformation from the spatial domain

1-4244-0 312-X/06/$20.00 ©2006 IEEE.

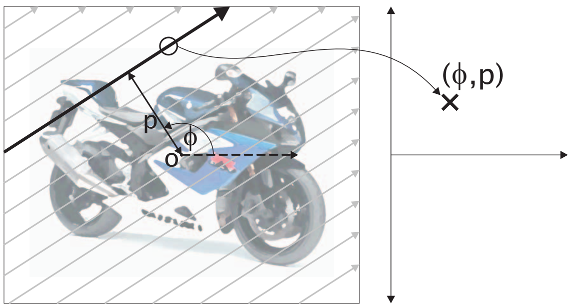

Fig. 1. Parameters of a Trace line

to a domain with parameters ϕ and p. Each (ϕ, p) point in the transformed image is the result of applying a defined functional on the intensity function of a line that crosses the original image tangential to an angle ϕ and at a distance p from the centre (see Figure 1). The resultant representation is an $n_\phi \times n_p$ image where n_ϕ is the number of angles considered and n_p the maximum number of lines across the image for each angle. An image can be traced with any number of functionals, each producing a corresponding Trace. Any functional can be used to reduce each trace line to a single value. The simplest functional, the sum of pixel intensities along the lines, yields the Radon Transform [7] of an image. An example of a transformed image is shown in Figure 2.

It may be easier to consider drawing multiple parallel lines across an image. This can be done for any angle. Then a functional can be applied to the pixel intensities in each line to yield a value for the (ϕ, p) point in the new domain. How the corresponding line pixels are selected is decided by the specific implementation. Standard interpolation methods including nearest-neighbour and bi-cubic can be used. The functional can be one of many proposed in the initial paper, or any other form of function, that reduces the sample vector to a single value. This might include sum, median, mode, sum of differences, RMS, etc. Table 1 shows some of the 22 functionals proposed by the Trace Transform authors. These were chosen for their strength in texture classification and their robustness to different forms of image transformation. It is clear that functionals some are computationally more intensive than others, and they also vary in nature. Accelerating these functionals as well as fully pipelining them can yield some significant performance boosts.

For feature extraction, further steps are needed, as shown in Figure 2. Firstly, a "diametrical" functional (D) is applied to the columns of the Trace image, reducing the space to a single vector. Finally, a "circus" functional (C) is applied to this vector to yield a single value feature. By combining a number of functionals at each stage, numerous features can be extracted. We are focusing solely on the first step of the Trace Transform since this is where most of the computational complexity lies; the diametrical and circus functionals are applied fewer times.

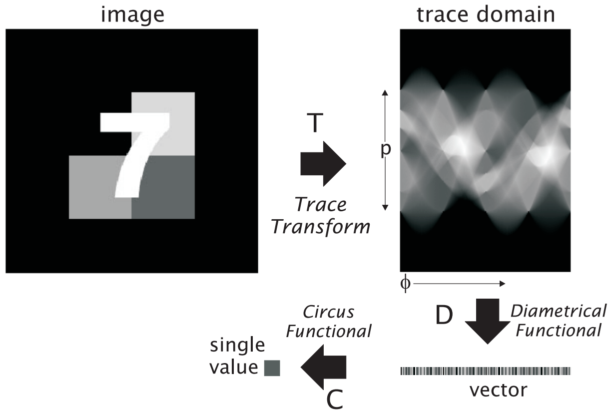

Fig. 2. An image, its trace, and the subsequent steps of feature extraction

Conceptually, the algorithm can be thought of as consisting of two separate parts: the line tracer and the functional blocks. The line tracer takes the parameters ϕ and p and outputs the pixel intensities along the corresponding line. The functional blocks each take these pixel intensities and apply the relevant functional to them.

2.2. Computational Complexity

There are a number of parameters of the transform that can be adjusted. Firstly, the number of angles to consider, n_ϕ. In the extreme case, we may consider all lines that intersect the image at any angle down to 1 degree increments or lower. It is however possible to consider larger increments, and this will reduce the computational complexity. It is also possible to vary the distance between successive lines, $1/n_p$. Again, in the extreme case, this could be a single pixel increment from one line to the next. Thus for a $N \times N$ image, there would be a worst-case of $\sqrt{2} \cdot N$ lines for each angle. It is further possible to vary the sampling granularity, n_t, along the lines, with the worst-case being to read every pixel, resulting in a maximum of $\sqrt{2} \cdot N$ pixels per line. It is important to note however, that these parameters will affect the performance of the algorithm, and an in-depth study is needed to determine the trade-offs involved.

The following parameters are used in discussing computational complexity:

- n_ϕ: the number of angles to consider;
- n_p: the number of distances (inverse of the interline distance);
- n_t: the number of points to consider along each trace line;
- n_T: the number of Trace functionals;
- C_ϕ: the operations required for trace line address generation;

No.	Functional	Details
1	$T(f(t)) = \int_0^\infty f(t)dt$	Radon Transform
3	$T(f(t)) = \left[\int_0^\infty \|f(t)\|^4 dt\right]^{\frac{1}{4}}$	
6	$T(f(t)) =\text{median}_t\{f(t), \|f(t)'\|\}$	
9	$T(f(t)) = \int_0^\infty r f(t)dt$	$r = \|l - c\|, l = 1, 2, \ldots, n$, $c =\text{median}_l\{l, f(t)\}$
15	$T(f(t)) =\text{median}_{t*}\{f(t*), \|f(t*)^{\frac{1}{2}}\|\}$	$f(t*) = \{f(t_{c*}), f(t_{c*+1}), \ldots, f(t_n)\}$

Table 1. Example Trace Functionals

- $C_T = \frac{1}{N}(C_{t1} + C_{t2} + \cdots + C_{tN})$: the average operations per sample for each functional.

Firstly, for the address generation, each of n_ϕ angles requires C_ϕ operations. Secondly, for each of n_T functionals, we are required to process n_t points on each of the n_p lines for each of the n_ϕ angles. If each pixel uses on average C_T operations, we require a total of $n_\phi n_p n_t n_T C_T$ operations for the Trace.

Therefore the total computational complexity is given by (1).

$$n_\phi C_\phi + n_\phi n_p n_t n_T C_T \quad (1)$$

For an $N \times N$ image, these parameters may have values: $n_\phi = 180$, $n_p \simeq N$, $n_t \simeq N$. This yields (2).

$$180 \times C_\phi + 180 \times N^2 \times n_T \times C_T \quad (2)$$

Considering that a standard system may include 8 to 10 functionals, and that functionals are not necessarily computationally simple in and of themselves, the high computational complexity is clear. Clearly the values of C_ϕ and C_T depend on the implementation platform, as software on a PC, these values are likely to be high, while with custom designed hardware, the values may approach 1 as the design becomes more pipelined. It is also clear that parallelisation of angles, line accesses and functionals has the potential to reduce computational complexity significantly. Hence without parallelisation, the algorithm becomes too complex for mainstream use.

3. ACCELERATION OF THE TRANSFORM

3.1. Framework

From the previous section, there are a number of areas where it is possible to accelerate the transform. Firstly, the rotations can be accelerated through some simplification of the line tracing algorithm. Furthermore, it is possible to process multiple rotations in parallel. Next, the functionals can be accelerated in and of themselves; by applying standard arithmetic principles and parallelising within the functionals, we can optimise them for speed. Finally, we can run multiple functionals in parallel. These four areas of acceleration should see significant performance gains over a software implementation.

It is important to note however, that all these proposals will be subject to the limitations of the implementation platform. As an example, though multiple parallel rotations sound promising, we must consider that with our image stored in a single bank of single-ported onboard RAM, we can only access one address location per cycle. With parallel functionals, we must pay close attention to the storing of results, since many large data buses are undesirable, and there may be insufficient connectivity to write all the results in parallel.

3.2. Functionals

The aim of this research is to construct a flexible framework for further investigation of Trace Transform functional properties. As such, the key to this implementation is the scalability of the architecture. Since the functions vary significantly as shown in Table 1, each family of similar functionals must be accelerated individually. For the purposes of designing this framework, we selected some of the simpler ones, as a proof of concept. We have previously published a novel algorithm for weighted-median calculation [8]. Median and weighted median are used in 14 of the 22 originally proposed functionals. Given that a functional only has to produce a result at the end of each line, there is sufficient time for a large number of complex functionals, as long as they are well pipelined.

4. PROPOSED ARCHITECTURE

4.1. General Overview

Our implementation consists of a few simple blocks. The Top-Level Control block simply oversees communication between the other blocks. Tthe Rotation Block produces an output that is the raster-scan of the input image rotated by an angle, specified by the Top-Level Control Block. Following this, each Functional Block reads the rotated image and calculates the results for each line. Finally, the Aggregator polls each of the functional blocks in turn before storing the results in the output RAM. The overall architecture is shown in Figure 3.

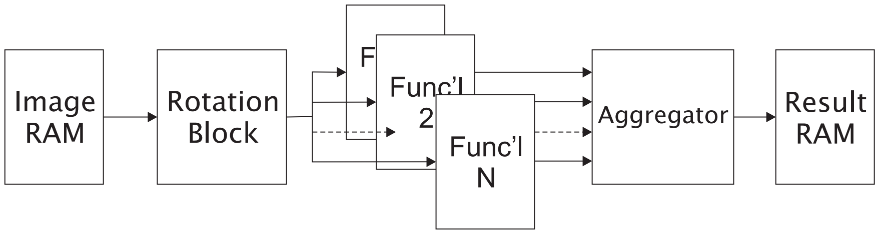

Fig. 3. Architecture Overview

4.2. Top-Level Control

The Top-Level Control block simply initialises each of the blocks, before sending the first rotation angle to the Rotation Block. When the last result for each rotation is ready, the most latent functional sends a signal, informing the Top-Level Control block to initiate the next rotation and so on. Once all angles are completed, the full result can be read from the output SRAM.

4.3. Rotation Block

As detailed previously, the line tracer should sample pixel values in the original image that fall along the designated trace line. Hence, this functional unit should be able to produce a set of addresses from which to read pixel values in the original image, given ϕ and p as inputs. However, given that we know that we wish to trace the image at all values of those inputs (within our sampling parameters), we are able to simplify the process. Rather than trace the image in such a manner, we can rotate the image itself, then sample in rows. This would simplify our circuitry somewhat by applying the trigonometric rotation functions once over the whole image, and allowing the subsequent blocks to have clearly ordered inputs.

An important caveat must be stated here. We have assumed that the input image is masked and when rotated will not suffer from detrimental cropping. Since face images in a face authentication system are always masked ellipses, this is a reasonable assumption. It is important to note that the Trace Transform itself only performs well when the subject of interest is masked; background noise is highly detrimental to its performance. Hence this assumption is valid for a Trace Transform implementation in any domain.

The line tracer now simply takes an angle as its input and produces a rotated version of the original image at its output. To simplify the system further, and negate the need for an image buffer inside the system, we configure the line tracer to produce the output in raster-scan format. Thus it reads from the source image out of order. This also precludes the need for any data addresses to pass through the system, since the address structure is inherent in the data.

Since we are working in hardware, we may also consider doing multiple rotations in parallel. However given that the image is stored in on-board RAM, we would require duplicates of the input image in other RAMs in order to achieve this, as well as further rotation blocks running in parallel. There is however a small trick that we use to quadruple our performance without increasing area. If we consider that a rotation by any multiple of 90 degrees is simply a rearrangement of addressing, which can be implemented in software or through minimal hardware, we can then deduce that any rotation above 90 degrees can be implemented as simply a rotation by a multiple of 90, then the remaining angle. To clarify this point, a rotation by 132 degrees is the same as a 90 degree rotated image being rotated through 42 degrees. Furthermore, given that we have ample wordlength in our external RAMS, we simply store the four orthogonal rotations (0°, 90°, 180°, 270°) in a single word. Then we calculate four rotations in parallel by using the line tracer to address the RAM, then splicing the resultant word. This effectively turns our single-ported RAM into a four-ported one.

The Rotation Block is implemented as follows: an on-board RAM contains the source image. This source image is, in fact, all four primary rotations and their respective masks concatenated as shown in Figure 4. Since the on-board RAMs available to us have a 36 bit wordlength, it is a perfect fit.

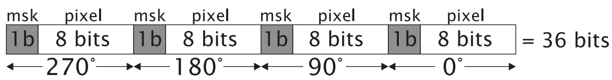

Fig. 4. Image RAM word

The block takes an angle as its input. This is used to index lookup tables containing sine and cosine values, which are used to implement the standard cartesian rotation equations in (3), fully pipelined.

$$x' = x\cos\theta - y\sin\theta; \;\; y' = x\sin\theta + y\cos\theta \qquad (3)$$

These equations all use simple 8 bit fixed point wordlengths. Nearest-neighbour approximation is used to avoid more complex circuitry. The resultant output of the block is a raster-scan of the rotated image still in its 36 bit concatenated form. This is fed to the functional blocks in parallel, where the computation occurs.

4.4. Functional Blocks

The Functional Blocks follow a standard design. They await a "start" signal to indicate the first pixel of a rotation is arriving, before beginning to compute the results. The Functional Block splices the incoming signal into the four constituent rotations and masks. The mask corresponding to each pixel determines whether it is considered in the calculation. Unmasked pixels are simply ignored in the calculation. Since the image size is fixed, each Functional Block keeps track of the current row number, and position within the row, to avoid the use of many control signals. When the end of each row is reached, its stores its results in an output buffer and sends a "new result" signal to the Aggregator.

The result wordlength depends on the functional, though all results can be individually scaled if necessary. Note that each of the functional blocks is actually duplicated four times, once for each of the orthogonal rotations. We only implement three functional blocks here to prove the concept. This architecture will be used for algorithm exploration, and so we will be implementing a large library of blocks to be incorporated in different configurations.

No.	Functional
1	$T(f(t)) = \sum_0^N f(t)$
2	$T(f(t)) = \sum_0^N \lvert f'(t)\rvert$
3	$T(f(t)) = \left[\sum_0^N \sqrt{\lvert f(t)\rvert}\right]^2$

Table 2. Trace Functionals

4.4.1. Functional 1

This is the simplest of the functionals, and simply sums all the pixels in each trace line. The Trace Transform using this functional is the equivalent of the Radon Transform. The corresponding equation is shown in (4). Figure 5 shows the schematic diagram of the design.

$$T(f(t)) = \sum_0^N f(t) \tag{4}$$

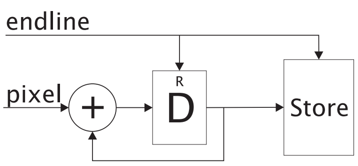

Fig. 5. Functional 1

4.4.2. Functional 2

This functional sums the absolute differences between adjacent pixels in each trace line. The corresponding equation is shown in (5). Figure 6 shows the schematic diagram of the design.

$$T(f(t)) = \sum_0^N |f'(t)| \tag{5}$$

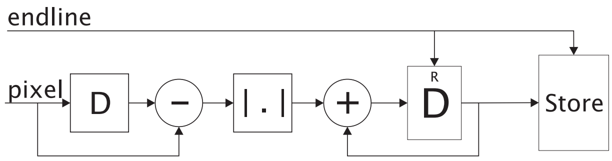

Fig. 6. Functional 2

4.4.3. Functional 3

This functional is the square of the sum of the square roots of the pixels in each trace line. The square root was implemented using a lookup table since the pixel intensities are only 8 bits wide. The square operation was implemented using the embedded multipliers. The corresponding equation is shown in (6). Figure 7 shows the schematic diagram of the design.

$$T(f(t)) = \left[\sum_0^N \sqrt{|f(t)|}\right]^2 \tag{6}$$

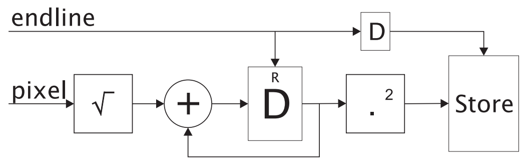

Fig. 7. Functional 3

4.5. Aggregator

The Aggregator polls the functionals in a round-robin fashion awaiting a "new result" signal. When received, it proceeds to store the four results from the current functional in a serial manner. This is done to avoid having a large data bus between the two units, since each result could be up to 24 bits in size. Since there is only a new result every 256 cycles, this still gives us sufficient time to read each result from each functional in series. The results are stored in another on-board RAM addressed using a concatenation of functional number, rotation and row number. The contents of this RAM can then be read by a host computer and the results used in further processing.

5. IMPLEMENTATION RESULTS

This design was implemented on a Celoxica RC300 board, using the DK Handel-C compiler. The board hosts a Xilinx Virtex-2 FPGA as well as on-board ZBT SRAMs. A host PC stores the input frames from a video device, or a series of images in one on-board RAM. The result of the Trace Transform is then read from another of the board's RAMs with display done on the PC. Synthesis results are shown in Table 3. The resultant clock-speed of 80MHz is only limited by development board libraries. This high speed was primarily due to the fully-pipelined nature of the design. Significant use of the channel communications provided for in Handel-C was made. The flexibility and power of this High-level language made the design simple to prototype.

Synthesis Results	
Clock Speed	80MHz
Frame Rate	26fps
Slices	2,070
BlockRAMs	6
Embedded Mults	8

Table 3. Synthesis Results

In comparison, a highly optimised MATLAB equivalent in software, running on a Pentium-4 2.2GHz, took just over 3 seconds to complete the same calculations. This hardware architecture thus gives over a 75 times speed-up. This acceleration increases with the number of functionals, as more functionals slow down the software version while not affecting the speed of hardware implementation.

The architecture completes four orthogonal rotations of a 256×256 image in 65,536 cycles. Each of these 65,536 pixels is passed to the functional blocks in order, with a new row starting every 256 cycles. In the last cycle of each row, the functional copies the results for each of the four orthogonal rotations to output buffers and continues with the next row. The Aggregator waits for a result to arrive at the first functional. It takes 7 cycles to store this result into the output RAM, then it continues with the other functionals in order, storing each of the results in the external RAM. It is free until the next results arrive, 256 cycles after the previous one. Once the last result is stored for a rotation, the Aggregator instructs the rotation block to start another rotation, and so the process continues.

The implementation serves as a flexible framework for Trace Transform applications. The simplicity of the architecture belies its power. More functionals can be added with ease. The only theoretical limit is where the functional result storage latency exceeds the time between successive row results. Given that for a 256×256 image, a new result comes in every 256 cycles, and that each functional result takes 7 cycles to store, this allows us space for 36 functionals. Assuming these functionals are each optimised and pipelined, they will not affect the overall latency of the system or its clock speed. Considering the original Trace Transform proposal included 22 functionals, some of which are redundant, this is a promising result. It is also important to note that the image size is flexible, since the system computes everything on a raster-scan stream of pixels.

6. CONCLUSION

We have shown how to accelerate the Trace Transform through parallelisation at various levels. The resultant implementation meets realtime requirements for a 256×256 pixel video stream. Using the framework presented here, it is our intention to embark on a detailed investigation of functional properties when applied to face authentication. By investigating a large combination of functionals, we aim to identify those that have the greatest discriminatory and similarity-grouping properties. We aim to use the system to run tests with thousands of faces in order to select and fine tune functionals for a full face authentication system.

7. REFERENCES

[1] A. Kadyrov and M. Petrou, "The trace transform as a tool to invariant feature construction," in *Fourteenth International Conference on Pattern Recognition, 1998. Proceedings.*, vol. 2, 1998, pp. 1037–1039 vol.2.

[2] ——, "The trace transform and its applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 8, pp. 811–828, 2001.

[3] ——, "Affine parameter estimation from the trace transform," in *16th International Conference on Pattern Recognition, 2002. Proceedings.*, vol. 2, 2002, pp. 798–801 vol.2.

[4] M. Petrou and A. Kadyrov, "Affine invariant features from the trace transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, vol. 26, no. 1, pp. 30–44, 2004.

[5] S. Srisuk, M. Petrou, W. Kurutach, and A. Kadyrov, "Face authentication using the trace transform," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 1, 2003, pp. I–305–I–312 vol.1.

[6] ——, "A face authentication system using the trace transform," *Pattern Analysis and Applications*, vol. 8, no. 1-2, pp. 50–61, 2005.

[7] S. Deans, *The Radon Transform and Some of its Applications*. John Wiley and Sons, 1983.

[8] S. Fahmy, P. Cheung, and W. Luk, "Novel FPGA-based implementation of median and weighted median filters for image processing," in *International Conference on Field Programmable Logic and Applications, 2005.*, 2005, pp. 142–147.