A Proposed Pipelined-Architecture for FPGA-Based Affine-Invariant Feature Detectors

Cristina Cabani & W. James MacLean
Department of Electrical & Computer Engineering,
University of Toronto, Toronto, Ontario, Canada, M5S 1A1
{cabani,maclean}@eecg.toronto.edu

Abstract

This paper describes a hardware architecture for an FPGA-based implementation of affine-invariant image feature detectors, following the algorithm of Mikolajczyk & Schmid. The architecture mimics the structure of the algorithm by implementing a multi-scale Harris corner detector which feeds candidate points into an iterative procedure to determine the local affine shape of the feature's neighbourhood (up to an undetermined rotation). Since the algorithm is iterative, and since we desire a high throughput rate, the iterations are "unrolled" into a sequence of identical computation blocks arranged in a pipeline architecture.

The modularity of the resulting architecture allows for scaling the implementation to devices of different resource capacity, as well as partitioning the algorithm over several devices. The final implementation, when completed, will be part of a smart-camera system which outputs features at the same time as the associated images.

1. Introduction

In recent years the role of feature detectors in computer vision has increased dramatically. Feature points have been used for object recognition & tracking, image segmentation, motion analysis and analysis of extended video sequences. The most useful feature detectors are invariant to translation, rotation, scale, or even more generally affine deformations of the image. The latter class of feature detectors is of particular interest as perspective distortion can be locally modeled as affine for smooth surfaces, which can in turn be assumed for small image regions. The affine-invariant feature detector (AIFD) implemented in this work is described in [12] and in earlier work [10, 11].

This paper will describe the proposed architecture for this implementation, which is currently underway. We hope that by describing the architecture at an early stage we can invite feedback from the vision community on the direction this work is taking.

1.1. Previous Work

Most modern feature detectors can trace their roots back to the Harris corner detector [6], which determines image regions to be interesting when they have sufficient image-gradient energy in orthogonal directions. While these features are not scale invariant, subsequent advances such as SIFT features [8, 9], scale-space representations [7] and affine-invariant [12] formulations have all taken the approach of searching for maximal responses over different scales to provide scale invariance. Rotational invariance is usually achieved with a *signature* or *descriptor* computed for the feature, where the descriptor identifies, and normalises with respect to, a dominant orientation. Given the usefulness of image features it is natural to want to compute them at high speed, or even frame-rates. A number of hardware approaches have been reported. There are several FPGA-based Harris corner detectors [1, 5], and an FPGA-based implementation of SIFT features is indicated in [13], although the implementation details are not public. These feature detectors are amenable to hardware implementation due to the lack of iteration in the associated algorithms. Our implementation tackles the implementation of AIFDs and the issue of iteration.

The structure of this paper is as follows. In Section 2 we describe how the feature detector in [12] is mapped to hardware. This section will assume the reader has knowledge of this detector algorithm, and for sake of brevity it will not be repeated in the paper. In Section 3 we will present an initial estimate of the FPGA resources required to implement each block. In Section 4 we discuss a number of issues that may affect performance of the implementation, and the pros and cons of various solutions. These are at present open issues for discussion. Finally, in Section 5 we describe the ongoing development process, and give future directions for this work.

2. Proposed Architecture

An overview of the proposed architecture can be seen in Figure 1. There are two main architectural components in our design: a multiscale Harris corner detector that is used to extract an initial list of candidate feature points, followed by a number of identical blocks that perform the iterative part of the feature detection algorithm. Although the iterative process could be implemented using a single block through which the points are repetitively cycled, this would result in lower throughput.[1] The number of iterator blocks sets an upper limit on the number of iterations a point can go through. Status signals are used to halt iteration early if the point converges or is rejected. The number of iteration blocks can be varied to fit the design onto FPGA devices of differing capacities, and also provide a simple way to partition the design across multiple devices. The initial points from the multiscale Harris corner detector are stored in a FIFO buffer before being fed into the iteration blocks. Points that are accepted during the iterative process are assembled into a feature list which then forms the output of the feature detector module. The following sections describe in greater detail the design of the multiscale Harris corner detector block and the iteration block.

2.1. Multiscale Harris Detector

The multiscale Harris detector is really just a Harris corner detector [6] in which the combination of image gradient information over the neighbourhood of interest is performed using Gaussian weights at different scales (standard deviations). The integration scales are based on σ_{I0} according to $\sigma_{Ii} = \varepsilon^{i-1}\sigma_{I0}$ for $i = 0 \ldots N$, where σ_{I0} and ε are user-selectable at run-time. For our implementation we set $N = 5$. The differentiation scales are set by $\sigma_{Di} = s\sigma_{Ii}$ where s can be set at run-time. The choices of value for N are somewhat limited by the available FPGA resources, as each scale i requires seperate filters, image buffers, combinatorial logic and other resources.

The Gaussian derivatives are computed by first creating simple image gradients using the Sobel operators, and then smoothing with a 11×11 Gaussian filter. This approach allows re-use of the Gaussian filters for integrating the gradient information, and thus is preferable over designing seperate masks for the Gaussian derivatives. The Sobel operators can be implemented using only adders, and therefore have a simple hardware implementation. The Gaussian filters are separable, and implemented as two 1-D convolutions in the x- and y-directions. Once the image derivatives are computed, the components L_x^2, L_{xy} and L_y^2 are integrated using the same Gaussian filter architecture, but using σ_{Ii} instead of σ_{Di}. This allows creation of the second-moment matrix

$$\mu = \sigma_{Di}^2 g(\sigma_{Di}) * \begin{bmatrix} L_{xi}^2 & L_{xi}L_{yi} \\ L_{xi}L_{yi} & L_{yi}^2 \end{bmatrix} .$$

[1]There is no reason the iteration block design cannot be re-used in a lower-throughput, state-machine design in smaller capacity devices.

The cornerness matrix is then computed as $|\mu| - \alpha \operatorname{tr}^2(\mu)$. Maximal results from 3×3 neighbourhoods that are above a specified threshold are then exported, along with their scale information, to be used as input to the iteration modules.

2.2. Iterative Estimation of Affine Shape

Figure 3 shows the overall architecture of each iteration block. In each iteration, the spatial and scale localisation of each initial point is refined, and an estimate of affine shape is updated through computation of the local second-moment matrix μ. The inputs to each iteration are coordinates of the feature, the associated integration scale, and the current shape transformation matrix U for the feature. The shape matrix, upon convergence, encodes an affine transformation that makes the image gradient energy over the feature's neighbourhood isotropic. The feature's status (accepted, rejected, in progress) is also carried between iterations. This allows features to be rejected at any iteration, and avoids accepted features undergoing further iterations. Each iteration block contains the following sub-blocks: *Normalise Window* to warp the image locally in an attempt to normalise the affine shape according to the current shape matrix, *Find Integration Scale* to refine the estimate of the integration scale (thus leading to improved localisation and estimation of local shape), *Find Differentiation Scale* does the equivalent for the differentiation scale, *Spatial Localisation* refines the current estimate of the feature's location and finally *Update Shape Matrix* refines the current estimate of the affine shape.

The *Normalise Window*, *Find Differentiation Scale* and *Update Shape Matrix* blocks all require the computation of eigenvalues and eigenvectors of 2×2 matrices. For matrices of this size we chose to derive analytical expressions for these quantities, avoiding the need for complicated algorithmic blocks. The square root operations required are based on the `altsqrt` core provided by Altera [3].

The **Normalise Window** block creates a warped version of the image in the neighbourhood of the feature. The warp is according to the current estimate for U, and attempts to select a region that is more isotropic. The image warp is computed by applying the warp to the pixel coordinates in the neighbourhood of the feature, and re-estimating the associated pixel values through bilinear interpolation from the original image pixels, stored in memory. The warped image region is 13×13 pixels. The warped image patch is denoted W.

The **Find Integration Scale** block, shown in Figure 4, is an extended version of the simplified Harris-Laplace detector from [12], and involves a similar architecture to the multiscale Harris detector of Section 2.1. The current value of σ_I is used to create $N = 5$ candidate scales in the range $\sigma_I/\varepsilon \ldots \sigma_I\varepsilon$. The Laplacian-of-Gaussian (LoG) of the incoming image patch W is computed for each candidate

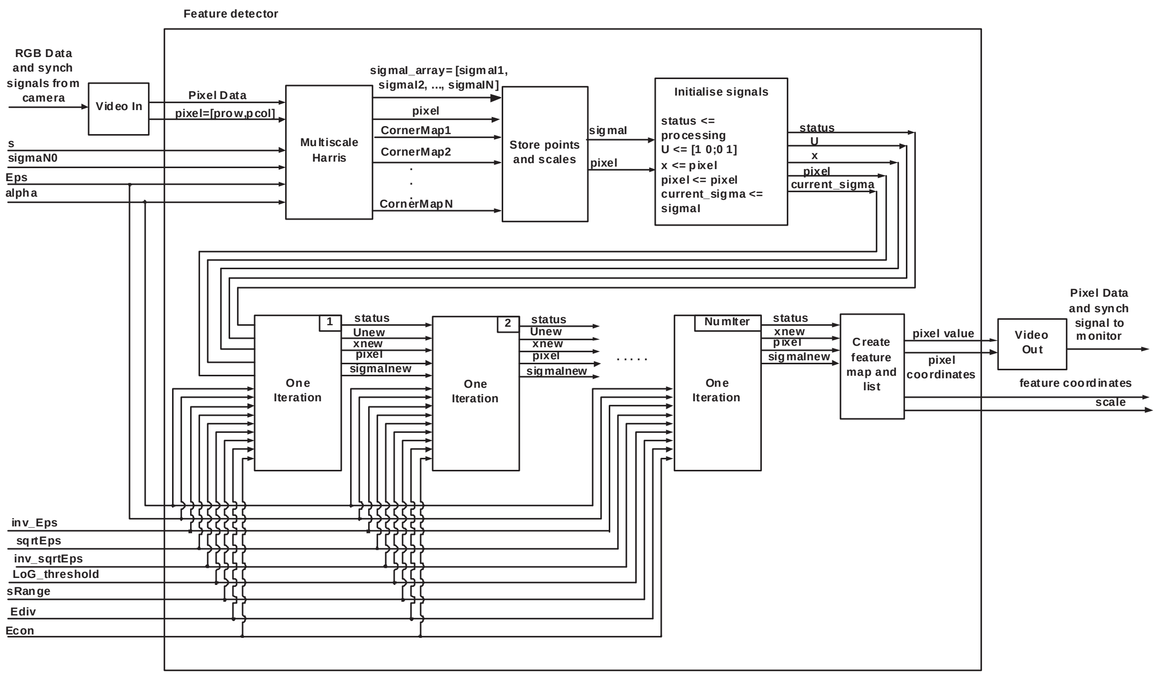

Figure 1: High level overview of proposed feature detector architecture.

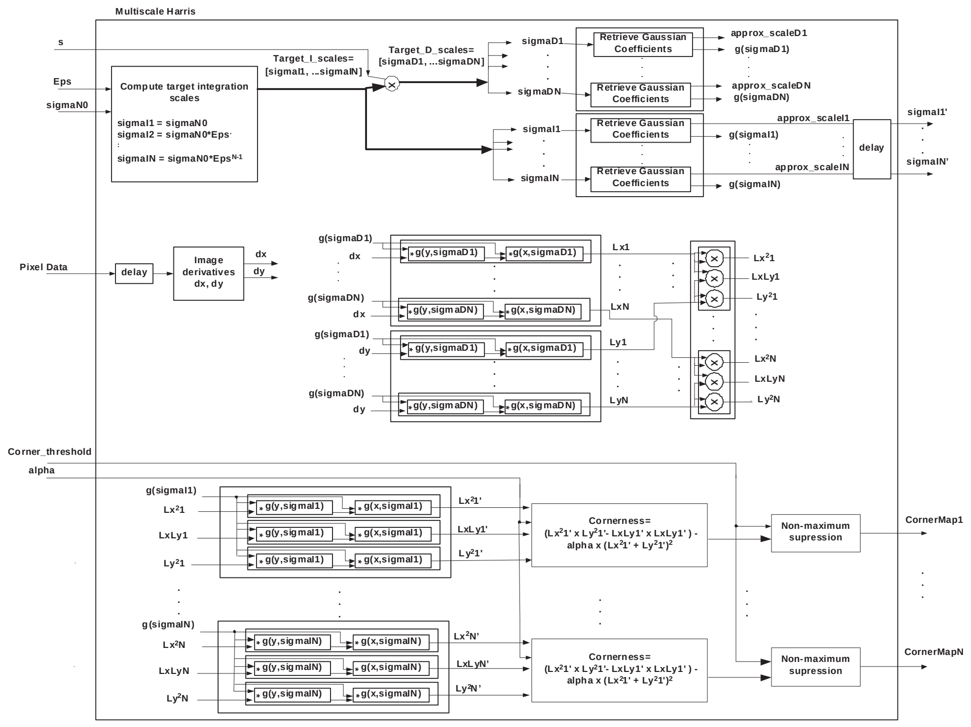

Figure 2: Architecture of multiscale Harris corner detector.

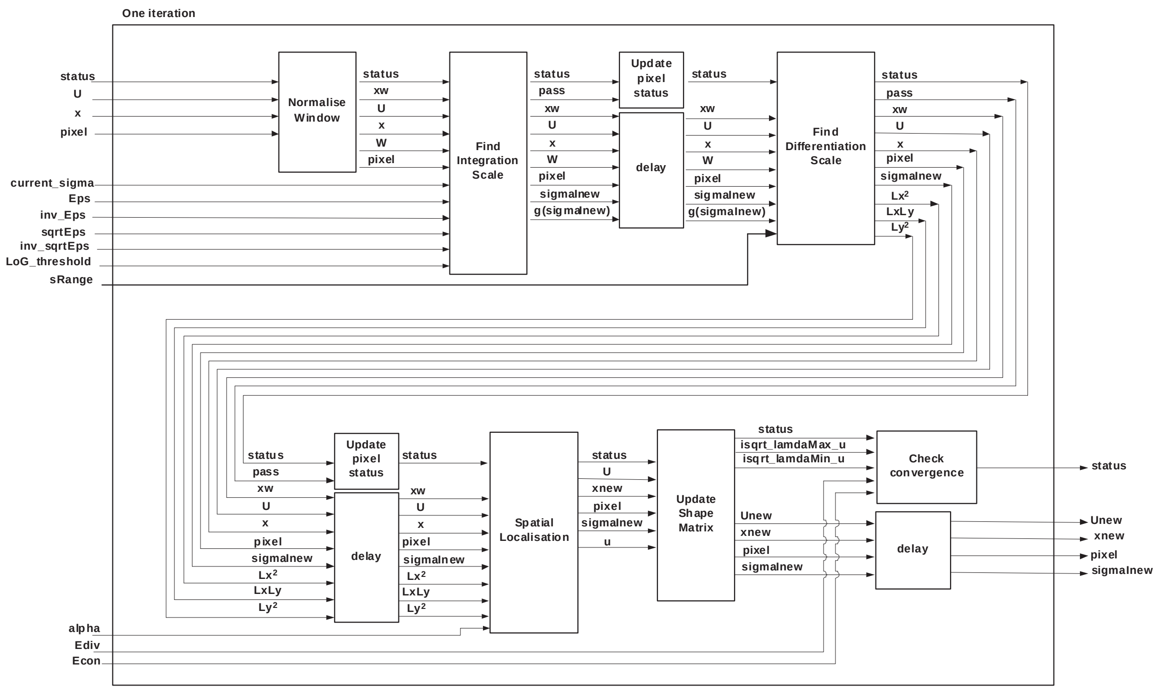

Figure 3: Structure of individual iteration block.

scale, and the scale producing the largest LoG becomes the new estimate for σ_I. If the largest result is below a pre-set threshold, this module can set the current feature's status as rejected. The values of $\varepsilon^{1/2}$, ε^{-1}, $\varepsilon^{-1/2}$ are assumed to be downloaded at runtime to avoid the cost of accurately estimating these in hardware.

The **Find Differentiation Scale** block attempts to find the value for σ_D that maximizes the value of the isotropy measure $\mathcal{Q} = \lambda_{\text{min}}(\mu)/\lambda_{\text{max}}(\mu)$ where μ is the second moment matrix as before. The differentiation scale is selected from $\sigma_{Di} = s_i\sigma_I$ where σ_I is the value refined in the *Find Integration Scale* module, and s_i takes on $M = 6$ evenly spaced values in the range $0.5 \ldots 0.75$ as suggested in [12]. The new value for σ_D will affect Gaussian smoothing kernels used in subsequent estimates of the second-moment matrix u. If none of the $\mathcal{Q}$s for the different s_i are above 0.2, status is set to rejected. Since the second-moment matrix is computed for each s_i, the value of μ for the selected differentiation scale is passed on to the **Spatial Localisation** block, shown in Figure 6, which recomputes the location of the maximum of the corner function over the warped image patch W. This location becomes the new feature location in the warped coordinates.

The location of the current feature with respect to the original image is then computed as $\vec{x}' = \vec{x} + U(\vec{x}'_w - \vec{x}_w)$, where the w subscript indicates coordinates with respect to the image patch W, and $\vec{x}'_w$ is the new maximum's location in W.

The **Update Shape Matrix** block is the last block, and computes a new shape matrix $U' = \mu^{-1/2}U$ where μ is the second-moment matrix evaluated at $\vec{x}'_w$ using the refined estimates for σ_I and σ_D. The value of $\mu^{-1/2}$ is computed analytically from the eigenvalues and eigenvectors of μ. This module also normalises U' so that its maximum eigenvalue is unity.

At this stage in the iteration block, a test for convergence is applied. It is based on the idea that as the eigenvalues of μ become closer to being equal, μ becomes closer to a pure rotation (and hence the energy of the warped image gradient becomes isotropic). Once the quantity $1 - \sqrt{\lambda_{\text{min}}/\lambda_{\text{max}}}$, where λ_{min} and λ_{max} are the eigenvalues of μ, drops below the threshold of 0.05, the feature is considered to have converged. The outputs of the iteration module are the current feature location with respect to the original image $\vec{x}'$, current estimates for σ_I and U, and finally the feature status information. These values serve to identify the location and spatial extent of the feature, and can be subsequently used to compute feature descriptors if desired.

3. Estimated Resource Usage

In this section we will report initial estimates for the expected resource requirements for implementing the proposed architecture. The estimates will be given in two forms. The first is counts for the different basic process-

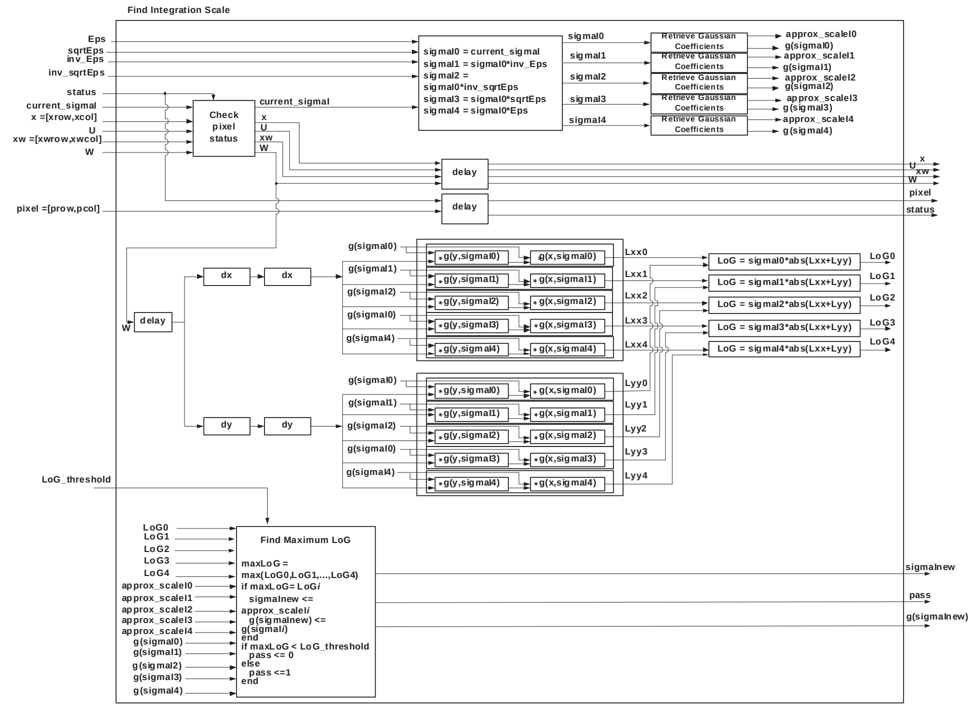
Figure 4: Architecture of module for finding σ_I.

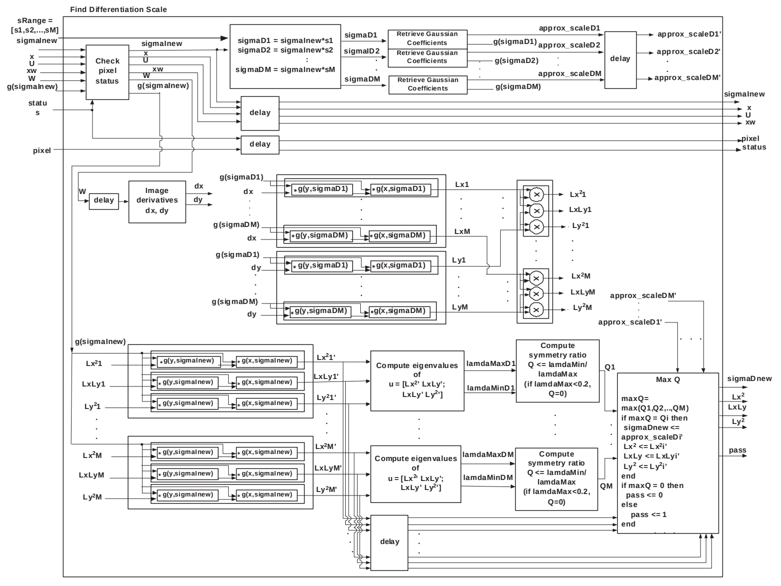
Figure 5: Architecture of module for finding σ_D.

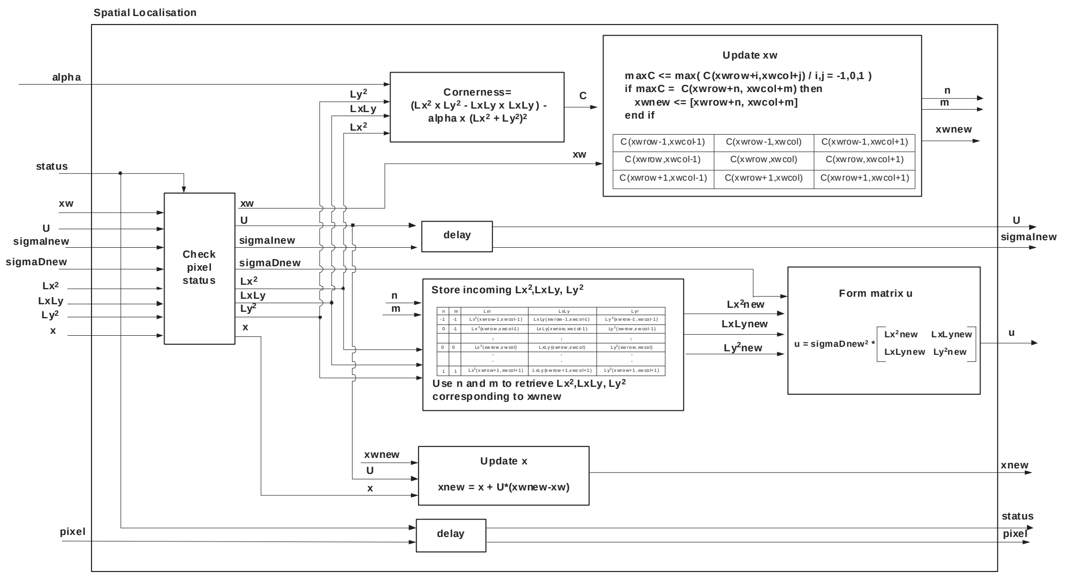
Spatial Localisation
alpha
status
xw
U
sigmaInew
sigmaDnew
Lx²
LxLy
Ly²
x
Check pixel status
Cornerness= (Lx² x Ly² - LxLy x LxLy) - alpha x (Lx² + Ly²)²
C
Update xw
maxC <= max(C(xwrow+i,xwcol+j) / i,j = -1,0,1)
if maxC = C(xwrow+n, xwcol+m) then
xwnew <= [xwrow+n, xwcol+m]
end if
n
m
xwnew
delay
Store incoming Lx²,LxLy, Ly²
Use n and m to retrieve Lx²,LxLy, Ly² corresponding to xwnew
Form matrix u
u = sigmaDnew² * [Lx²new LxLynew; LxLynew Ly²new]
Lx²new
LxLynew
Ly²new
u
Update x
xnew = x + U*(xwnew-xw)
xnew
pixel
status

Figure 6: Architecture of module for refining the spatial localisation of the feature.

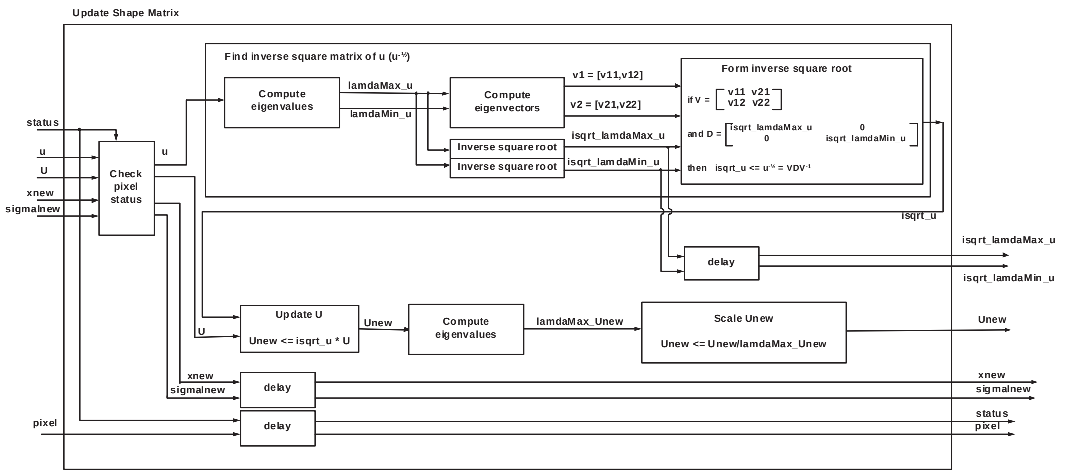

Figure 7: Architecture of module for updating the shape matrix.

ing units such as adders, multipliers, and dividers. These estimates can be used to generate rough estimates for the required logic resources on a target FPGA device independent of the manufacturer or device family. The second form of resource estimation will be related to the number of required logic elements on Altera Stratix S80 devices, four of which are available on the target development environment for the current implemetation.

The multiscale Harris corner detector module will require the following: 485 adders, ranging in size from 8 to 48 bits; 344 multipliers, ranging in size from 9 to 24 bits; and 230 buffers requiring 2972 Kbits of on-chip RAM. In total, the module will consume roughly 99% of a Stratix S80 FPGA.[2] It should be noted that the LUT usage is likely a significant over-estimate, as some of the multipliers will likely be targeted to the 176 dedicated multipliers and/or 22 DSP blocks on the device, thus not using general purpose LUTs. Also, some buffers will be automatically synthesized into flip-flops that accompany the LUTs used for other purposes. Finally, no attempt has been made yet to find the minimum size of multipliers that will yield acceptable performance, and this may further reduce resource usage. A similar analysis shows the iteration block as consuming 140% of the resources of an S80, but again this is expected to be an over estimate. Since it is highly desirable for the iteration block to reside on a single device, a careful analysis is required to determine precision requirements and how resources might be saved by reducing bit-widths to the minimum required sizes. Table 1 gives a summary of resource usage, and explicitly indicates the proportion of LUTs that are not used by multipliers in order to get a sense of potential savings by transferring as many multipliers as possible to the embedded multipliers and DSP blocks. The LUT counts that include multipliers are computed as if no embedded multipliers or DSP blocks were being used. The iteration block also has a number of elements not occuring in the multiscale Harris block: a number of square root operations (an 18-bit square root requires 139 LUTs, a 24-bit square root requires 223 LUTs), and some unsigned divisions (for 10-bit numbers this requires 128 LUTs). While these operations require substantially more resources than multipliers, they are far fewer in number and so are not the main concern when it comes to finding more efficient implementations.

It is evident that implementation of this algorithm on FPGAs will be very resource intensive. It therefore becomes a priority to seek more efficient methods to carry out the various stages of the computation. For example, if an efficient scheme can be devised for computing second-moments for different affine transforms (perhaps using basis filters as in the popular steerable filter implementations by Freeman and Adelson [4]), then considerable savings might be expected in both the *Multiscale Harris* block and the *find Differentiation Scale* blocks.

[2]An S80 has 79,040 LUTs, 7253 KBits on-chip RAM, 176 embedded multipliers and 22 DSP blocks. See `http://www.altera.com/products/devices/stratix/features/stx-dsp.html` for further details.

4. Anticipated Challenges

In addition to the task of finding more efficient computational approaches, there are a number of potential challenges to successful implementation of the proposed architecture. First, only a limited range of values for σ_I can be considered. This will limit the possible scales at which feature points can be found in the incoming images. This is related to the fact that the number of scanline buffers is proportional to the size of the Gaussian filter kernel multiplied by the width of the incoming image, thus limiting the size of the Gaussian filter kernel. While it may be possible to recursively filter the incoming image at different scales, this does not alleviate the problem completely as intermediate buffers will be necessary between successive stages. This issue is expected, at worst, to lower feature point yields but not prevent successful detection of features.

A second concern relates to a scanline buffer required by the *Normalise Window* block. This buffer is necessary to allow the generation of the warped image W on which the computations in each iteration are carried out. For large values of σ_I the support region being mapped could be quite large, requiring a large scanline buffer. Further, if this buffer is shared among the multiple iteration blocks, then synchronized access is require to prevent collisions when accessing memory. Much simpler, but more resource intensive, is to provide a separate scanline buffer for each iteration block. A further synchronisation issue exists if a feature point in the iteration pipeline attempts to warp scanlines that have not yet arrived, although this could be handle by marking the point as rejected and simultaneously re-inserting it into the initial point buffer for subsequent re-processing.

Finally, there is the issue of searching for cornerness maxima within the 8-neighbourhood of the current maximum. There is no guarantee that the new maximum will be within one pixel of the current one. One solution is to consider larger neighbourhoods (5×5 or 7×7). However, it may be that the cornerness function is smooth enough that even if the true maximum is not within the 8-neighbourhood, that the local maximum in the 8-neighbourhood will be in the right direction and the feature's convergence will be merely slowed down, but not prevented. It is possible that other issues will arise as implementation proceeds.

	LUTs	LUTs (excl. mults.)	Memory [Kbits]
Total: Multiscale Harris	78728 (99.6%)	22888 (29.4%)	2905 (40.0%)
Normalise Window	3325 (4.2%)	1987 (2.5%)	5.1 (0.08%)
Integration Scale	27032 (34.2%)	12186 (15.4%)	7.5 (0.1%)
Differentiation Scale	70551 (89.3%)	29335 (37.1%)	58.9 (0.8%)
Spatial Localisation	4329 (2.3%)	1803 (2.3%)	
Update Shape Matrix	4990 (6.3%)	2816 (3.6%)	
Total: Iteration Block	110227 (140%)	48127 (61.0%)	71.5 (1.0%)

Table 1: Resource Usage Estimates for Stratix S80

5. Conclusions

This paper presents an architecture for an affine-invariant feature detector based on the algorithm proposed in [12]. The implementation of this architecture is currently underway, using Altera's Stratix family of FPGAs on a custom development platform. The architecture is modular in the sense that one can choose the maximum number of iteration blocks to suit the implementation to a specific FPGA device. Should multiple devices be required, the iteration blocks provide a convenient partitioning of the implementation to simplify its distribution across several FPGAs. Although the proposed architecture shows multiple iteration blocks pipelined to maximize throughput of feature points, the same block can be used in a state machine that uses one iteration block to iterate feature point candidates, but at a lower throughput. The multiscale Harris corner detector module can also be parameterized according to N, the number of different integration scales used to search for inital points.

An initial estimate of FPGA resources required to implement this architecture, in terms of LUTs on Stratix series FPGAs, is given. While the estimates are intended to be conservative, they should provide a sense of the resource requirements for the implementation. As our implementation proceeds more accurate resource estimates will become available.

The ability to identify affine invariant feature points at frame rates is a valuable functionality to add to a smart camera, and is expected to advance the state of the art of frame-rate computer vision implementations.

References

[1] A. Benedetti and Pietro Perona. Real-time 2-D feature detection on a reconfigurable computer. In *Proceedings of the IEEE Conference on Computer Vision & Pattern Recognition*, 1998.

[2] Gustavo Carneiro and Allan D. Jepson. Local phase-based features. In *Proceedings of the 2002 European Conference on Computer Vision*, volume 1, pages 282–296, Copenhagen, Denmark, 2002.

[3] Altera Corporation. `altsqrt` Megafunction User's Guide, 2005.

[4] William T. Freeman and Edward H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991.

[5] Paolo Giacon, Saul Saggin, Gionanni Tomasi, and Matteo Busti. Implementing DSP algorithms using Spartan-3 FPGAs. *Xcell Journal*, Issue 53:22–25, 2005.

[6] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings Fourth Alvey Vision Conference*, pages 147–151, Manchester, United Kingdom, 1988.

[7] Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116, 1998.

[8] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh International Conference on Computer Vision*, pages 1150–1157, Kerkyra, Greece, 1999.

[9] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:90–110, 2004.

[10] Krystian Mikolajczyk and Cordelia Schmid. Indexing based on scale invariant interest points. In *IEEE International Conference on Computer Vision*, pages 525–531, 2001.

[11] Krystian Mikolajczyk and Cordelia Schmid. An affine invariant interest point detector. In *European Conference on Computer Vision*, volume 4, pages 128–142, 2002.

[12] Krystian Mikolajczyk and Cordelia Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004.

[13] Stephen Se, Timothy Barfoot, and Piotr Jasiobedzki. Visual motion estimation and terrain modeling for planetary rovers. In *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space,*, Munich, 2005.