Nature-Inspired BASIS Feature Descriptor and Its Hardware Implementation

November 15, 2011

Spencer G Fowers[1] D.J. Lee[2] Dan Ventura[3] James K. Archibald[4]

Abstract

This paper presents the development of a color feature descriptor well-suited for low resource applications such as UAV embedded systems, small microprocessors, and field programmable gate array (FPGA) fabric, and its hardware implementation. The BAsis Sparse-coding Inspired Similarity (BASIS) descriptor utilizes sparse coding to create dictionary images that model the regions in the human visual cortex. Due to the reduced amount of computation required for computing BASIS descriptors, reduced descriptor size, and the ability to create the descriptors without the use of floating point, this approach is an excellent candidate for FPGA hardware implementation. The BASIS descriptor was tested on a dataset of real aerial images with the task of calculating a frame-to-frame homography. Experimental results show that the software and bit-level accurate BASIS descriptor outperforms SIFT and SURF. The bit-level accurate hardware version of the BASIS descriptor also results in no loss of accuracy when compared with the software version. BASIS descriptors are more space efficient than other descriptors, and can be computed entirely in FPGA fabric, allowing the descriptor to operate at real-time frame rates on a low power, embedded platform such as an FPGA.

1 Nomenclature

B = Basis dictionary image set
β_i = Basis image i
β_H = Basis image height

[1]PhD Candidate, Department of Electrical and Computer Engineering, 459 CB, Brigham Young University
[2]Professor, Department of Electrical and Computer Engineering, 453 CB, Brigham Young University
[3]Associate Professor, Department of Computer Science, 3366 TMCB Brigham Young University
[4]Professor, Department of Electrical and Computer Engineering, 451 CB, Brigham Young University

β_W = Basis image width
ω = Octave (scale)
FRI_H = FRI pixel height
FRI_W = FRI pixel width
I_1 = image 1
I_2 = image 2
p_1 = point in image 1
p_2 = point in image 2
H = Homography relating image 1 to image 2
BSM = BASIS similarity measure
SIS = similarity indicator sum

2 Introduction

Computer vision applications for low power, low resource, and embedded systems are becoming increasingly prevalent. Target tracking [25], object identification [9], optical flow [28], stereo vision [24], super resolution [13], image stabilization [15], rectification, localization, and pose estimation [25][17][26][2][6] are all examples of computer vision applications that can be realized on a low-power embedded platform. Computer vision applications such as image registration, object detection, motion estimation, super resolution, image stabilization, pose estimation, and target tracking all require some level of high quality feature description and matching. The Open Source Computer Vision library, which is often used in computer vision applications, now contains a feature detection and description library that includes the MSER, Star, SIFT, SURF, and FAST feature detectors and descriptors. As low resource platforms such as smartphones become more prevalent, the need for low resource vision algorithms will continue to increase. There are now apps available for major smartphone operating systems that provide motion detection, color detection, barcode scanning, OCR, accessibility aid, and scene recognition. A search on the Android Market (http://market.android.com) for the terms motion detector, OCR, and color detector yielded over 1000 app results as of September, 2011.

Feature descriptors take feature points obtained from a feature detector (such as the DoG, Harris, FAST, or Harris Affine), and compute a unique description of that point [18][1][27][11]. A good feature descriptor will uniquely describe a feature point to distinguish it from other feature points, thus allowing for correct identification and tracking. SIFT creates descriptors by computing orientations and magnitudes of intensity gradients [18]. This works well in object recognition tasks on gray scale images and provides features invariant to rotation and scale. SURF computes its descriptors using integral images and Haar wavelet transforms [1]. CCH computes a histogram of intensities for a feature point compared to a circular region around the point [11]. CDiKP uses a method similar to SIFT but compresses the descriptor using a Walsh-Hadamard kernel [27]. The resulting descriptors from these methods are vectors of double-precision floating point numbers. For example, a SIFT descriptor is

a vector of 128 double-precision numbers, and a SURF descriptor is 64 double-precision numbers). These descriptors require a large amount of storage space: each 128-element SIFT descriptor requires 1024 bytes [18], and each 64-element SURF descriptor requires 512 bytes [1]. The OpenCV implementation of SURF uses 128-element descriptors, making their storage size 1024 bytes per descriptor. The storage space becomes an issue because these algorithms can easily return more than 500 features from a 640x480 pixel resolution image. Also, today's most commonly used descriptors (SURF and SIFT) are very difficult to implement in a limited resource application due to the complex descriptor calculations and the use of floating point. For example, it is difficult to implement a full floating-point unit in an FPGA's hardware logic, so most FPGA implementations of feature descriptors off-load the actual descriptor computation to a CPU for floating point computations [2] or perform other algorithm simplifications (such as conversion to fixed point) in order to accomodate a hardware implementation [23][28].

The BAsis Sparse-coding Inspired Similarity (BASIS) feature descriptor provides significantly better size efficiency than existing detector/descriptor methods, is fully implementable on a low-resource or FPGA platform, and still provides good descriptor matching accuracy. This paper outlines both the development of the BASIS algorithm and its hardware implementation. Our intended application is a low power, embedded vision system for small UAVs. By focusing on hardware during our descriptor design we have developed a descriptor that does not require major simplifications to the algorithm in order to accommodate hardware implementation. Our target system is a small FPGA platform based on a Xilinx Virtex-6 FPGA. Image information is fed into the FPGA system from a small CMOS camera at up to 60 frames per second. This vision system will be used for small unmanned aerial vehicle (UAV) and unmanned ground vehicle (UGV) image stabilization, rectification, and pose estimation. Our goal is to develop an accurate descriptor for the task of frame-to-frame image registration, and implement the descriptor on our FPGA platform.

In this paper we describe the development and hardware adaptation of our new BASIS feature descriptor that implements a derivative of sparse coding. Feature regions in an image are described by their similarity to pre-computed basis images. The resulting similarity values uniquely describe the features and allow them to be matched in subsequent images. Because our descriptor does not use floating point, it is an excellent fit for an FPGA implementation. Due to the more efficient descriptor size, it also requires less memory than other popular algorithms to compute and store descriptors. Section 3 describes the BASIS descriptor algorithm. Section 4 outlines the hardware implementation of the descriptor. In Section 4.2, we provide the results from an application of the bit-level accurate version of the BASIS descriptor on selected datasets, as well as comparisons to SIFT and SURF. In Section 5 we discuss our conclusions and future work.

3 The BASIS Descriptor

3.1 Sparse Coding

As with all descriptors, BASIS descriptors need to somehow uniquely identify a feature point in two distinct images. Our method of describing feature points is derived from sparse coding. Sparse coding is the process of creating a dictionary of non-orthogonal basis functions, and optimizing over that dictionary to find coefficients which allow a target signal to be reconstructed from the dictionary. The dictionary (because it is non-orthogonal) has a higher dimensionality than the target signal, but the benefit comes from using sparsity as an optimization constraint to allow reconstruction of the target signal from as few basis functions as possible [5]. The process flow for sparse coding begins with using an algorithm such as the K-SVD to develop a basis dictionary. This requires training the K-SVD on a large dataset of images in order to create generic basis functions that can be used to reconstruct a large variety of target signals. Reconstruction then happens by again using the K-SVD or similar optimization stage to recreate the target signal, while optimizing over sparsity of basis functions. Some example applications of sparse coding for images are image reconstruction (or inpainting) [19][12], denoising [4], feature extraction for denoising purposes [14], and scene classification [16]. In these examples, an image with noisy or missing regions is used as a target signal, and, using a pre-defined basis dictionary, sparse coding allows researchers to inpaint the degraded or missing areas and reconstruct the damaged image.

In a paper by Olshausen [20, 21], sparse coding was applied to a very large dataset of natural images in order to look for similarities between sparse coding dictionaries and the human visual cortex V1. The resulting dictionary consisted of very basic geometric shapes. Olshausen postulated that this implied that all natural images are thus composed of these same types of geometric shapes: "The receptive fields that emerge from this algorithm strongly resemble those found in the primary visual cortex, and also those that have been previously deduced by engineers to form efficient image representations." [21] Olshausen used entire images as input to the basis dictionary creation. However, in our application for frame-to-frame feature point matching, the images used to create a basis dictionary set are small regions around a feature point. Because these feature region images are small and the types of intensity texture that form these features are limited, we postulate that our resulting basis dictionary set (unlike the basis dictionary sets created using entire images) will be similar not only across the set of all natural images, but also across the set of all images containing man-made objects.

Because sparse coding has been shown to be effective in recreating lost portions of images in denoising and image reconstruction applications, the set of basis functions used by the sparse coding algorithm can be thought of as potential descriptions of image areas. Our work is based on the fact that, if the basis function set is kept constant, the coefficients representing the contribution of each basis function can be used as a feature descriptor. In the same

way, the sparse coding algorithm can generate a set of basis images using a set of feature region images (small pixel regions around a detected feature in an image) as input. Linear combinations of some or all of the resulting basis images can re-create the feature region images. These basis images comprise a dictionary of feature characteristics that can be combined in varying degrees to uniquely reconstruct the feature region around any feature in an image. This approach is similar in theory to vector quantization, used in signal compression and video codecs, which processes entire frames [10]. By building dictionaries for all three channels of an image, we can uniquely describe color features as well as grayscale.

The BASIS descriptor algorithm consists of two portions: initial basis dictionary creation, and BASIS descriptor computation. Dictionaries can be computed off-line, on standard desktop computing hardware, and then stored in memory on the on-line system. Once the basis dictionaries have been created off-line, BASIS descriptors can be computed in real time on a low-resource platform.

3.2 Off-Line Dictionary Creation

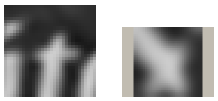
We define a small pixel region centered around a feature point as a feature region image, or FRI. Features are detected in sample images using a feature detector, and FRIs (Fig. 1), are saved to disk as individual images. Using sparse coding, we create a set of basis images (or functions), $B = \beta_1 \dots \beta_n$, from a large dataset of FRIs. In our experiments, a dataset of over 300,000 FRIs was used to create each basis dictionary. If a smaller dataset is used, overfit can result and the basis functions $\beta_1 \dots \beta_n$ will not be generic. The set of basis images, B, is then saved to disk and used in real time as the basis dictionary set.

For this off-line processing step, we detected features using the color Difference of Gaussians (DoG) [8, 7]. The color DoG performs the DoG on all three channels of the YCbCr color space which provides color features from the Cb and Cr channels as well as grayscale features from the Y channel. Using the color DoG also provides scale invariance due to its use of scale space. This provides us with a large variety of basis images to use as input to the sparse coding portion of our algorithm. The color DoG returns, for each feature it detects, its position (x, y), the scale (or octave), and the channel (Y, Cr, or Cb) in which the feature was detected. Although this work uses the Color DoG for basis image creation, any repeatable feature detector may be used to create the dataset for basis dictionary creation.

For each feature obtained by the color DoG, we create an FRI centered on the feature point. In order to create uniform FRIs for features found at different scales, the initial dimensions of the FRI are dependent on the scale at which the feature was detected. Because the DoG algorithm upscales the original image by a factor of 2 before creating the scale space pyramid, the size of the FRIs are computed as

$$FRI_H = 2 * \beta_H * 2^{(\omega-1)} FRI_W = 2 * \beta_W * 2^{(\omega-1)} \quad (1)$$

where β_H and β_W are the desired width and height of the final basis dictionary functions, and ω is the scale (or octave) in which the feature was found. In our experiments, we chose $\beta_H = \beta_W = 30$ pixels. At this point, the FRI is then scaled to $\beta_H \times \beta_W$ using bilinear interpolation so that all FRIs are the same size, regardless of the scale at which the feature was found. Fig. 1(a) shows an original FRI image, and Fig. 1(b) the resulting re-scaled FRI. For our



(a) (b)

Figure 1: The FRI is originally obtained at a larger size ($FRI_H \times FRI_W$) (a), and re-scaled to ($\beta_H \times \beta_W$)(b).

experiments, we obtained eight video sequences (30 seconds or less of 640x480 resolution) of natural and non-natural scenes (Fig. 2). For each frame in these sequences, we performed the above process for generating FRIs, and saved these FRIs for future processing. Each video sequence provided over 300,000 FRI images. This set of over 300,000 FRI images was input into the K-SVD algorithm [3] which produced a set of 100 30x30 basis images $B = \beta_1 \ldots \beta_{100}$. Fig. 3(a)-3(c) shows examples of three basis image sets for three different video sequences. We also then combined the FRI images (approximately 8x300,000 FRI images) from all eight video sequences and processed them using the K-SVD to generate a combined basis image set of 100 30x30 basis images (Fig. 3(d)). As we theorized, because the feature regions are small and the types of intensity texture that form these features are limited, the basis image set from all eight videos combined looks very similar to the individual basis image set for each video sequence regardless the contents of the scenes. Because the basis image sets were very similar, we created a final basis image set (dictionary) by randomly selecting 128 basis images for each channel of our image (Y,Cr, and Cb) for a total of 384 basis images to be used as dictionary images for the BASIS descriptor computation.

3.3 Calculating Descriptors (Online) - The Modified Census Transform

The resulting basis dictionaries are stored in BRAM where they can be accessed by the on-line portion of the algorithm. Only the on-line portion of the BASIS descriptor needs to be implemented in the FPGA system. The on-line portion of the BASIS descriptor algorithm, shown in Fig. 4, takes a list of features detected using the FAST feature detector [22] and returns a descriptor for each feature. Our desired application of frame-to-frame image registration for a small unmanned aerial or ground vehicle (UAV/UGV) requires a fast, low-power feature detector, and does not need invariance to rotation greater than 10 degrees due to the nature of our data. Because of these requirements, the FAST feature

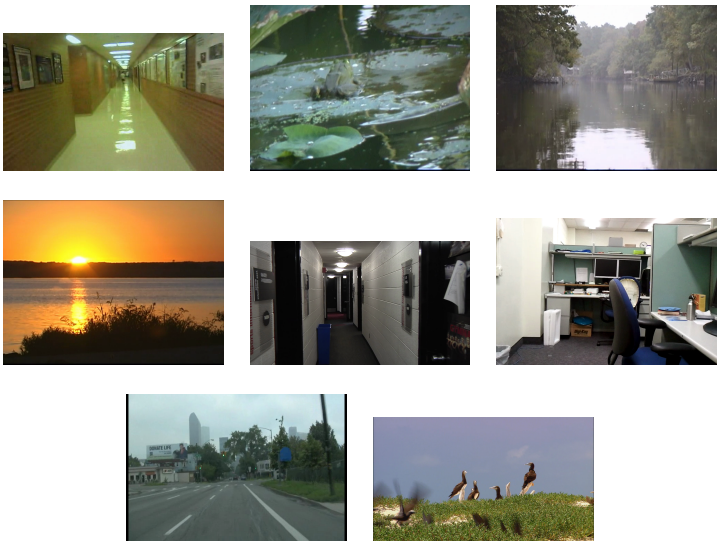
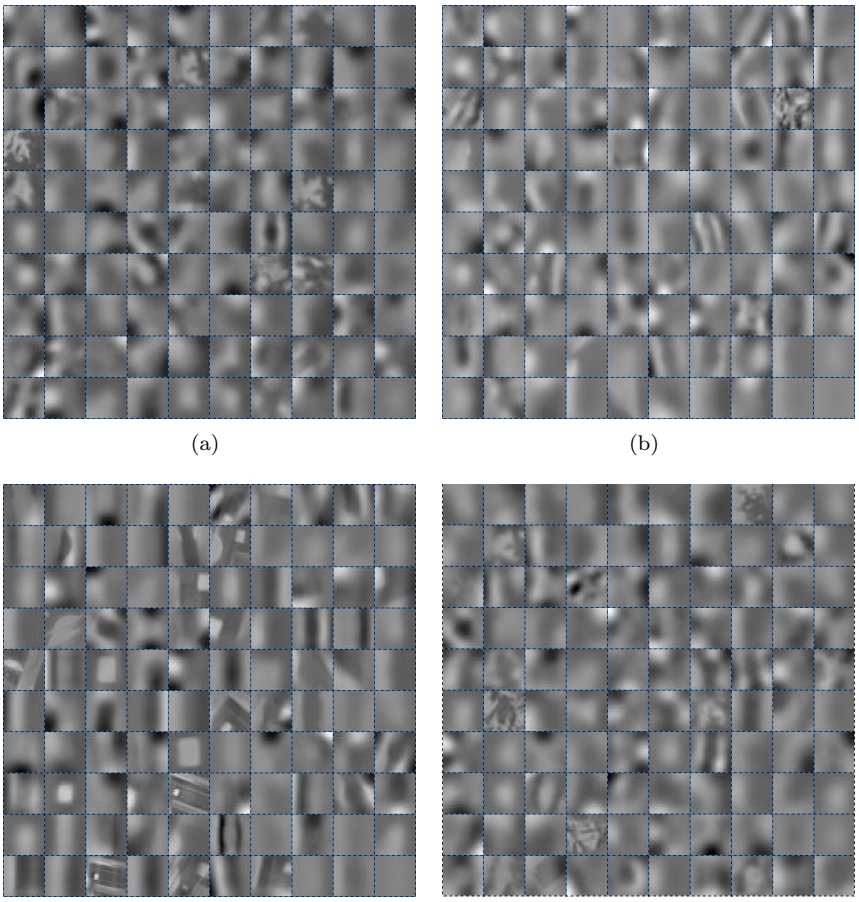


Figure 2: Typical frames from each of the 8 video sequences used to generate FRIs as input to the K-SVD.

detector [22] proved to be an excellent fit for our application. The FAST detector provides more features and a higher repeatability of features than the color DoG, but it does not provide color features, orientation, or scale invariance. If rotation or scale invariance is needed for a given application, however, the only change to the algorithm is to add a feature orientation and/or a scale value to each feature. As described in Section II(a), a scale-appropriate FRI is obtained from the original image. The FRI is then scaled down to the same dimensions as the elements of the basis dictionary B so that a descriptor can be calculated.

The intent of the descriptor is to describe how similar this FRI is to each individual basis image β_i in the dictionary. Because each β_i demonstrates a simple characteristic of a feature (such as an edge, gradient, or corner), the entire 30x30 pixel area of each β_i may not be of equal value. Fig. 5 shows a β_i (a) and an example FRI (b). This FRI matches well with this β_i in the upper right corner (both images have very similar pixel values), but not well in the lower half of the image (there are large portions of the FRI that are black where the β_i has a much higher value). Performing a simple difference and then summing the result would average out the resulting similarity, and the important information about which regions of the FRI match well would be lost. To resolve this issue, we developed a unique similarity measure, derived from the Census transform [29], that we call the BASIS similarity measure (BSM), which retains the spatial information of BASIS similarity. The first step in the BSM calculation is to perform a direct pixel-by-pixel subtraction of the FRI from the β_i (Fig. 6).



(a) (b) (c) (d)

Figure 3: Various basis image sets (or basis dictionaries, B). All of these sets contain geometric shapes visibly similar to those found by Olshausen in [20]. (a) was obtained from a video sequence of a drive through a downtown area. (b) was obtained from a video sequence of a frog in a pond. (c) was obtained from a video sequence of movement down a typical building hallway. (d) was obtained by combining FRIs from each of the eight video sequences (natural and non-natural). The combined basis image set in (d) is very similar to those seen in all the other basis image sets we created. Because of this, the basis image set (dictionary) we used for our algorithm to create descriptors will work on both natural and non-natural images.

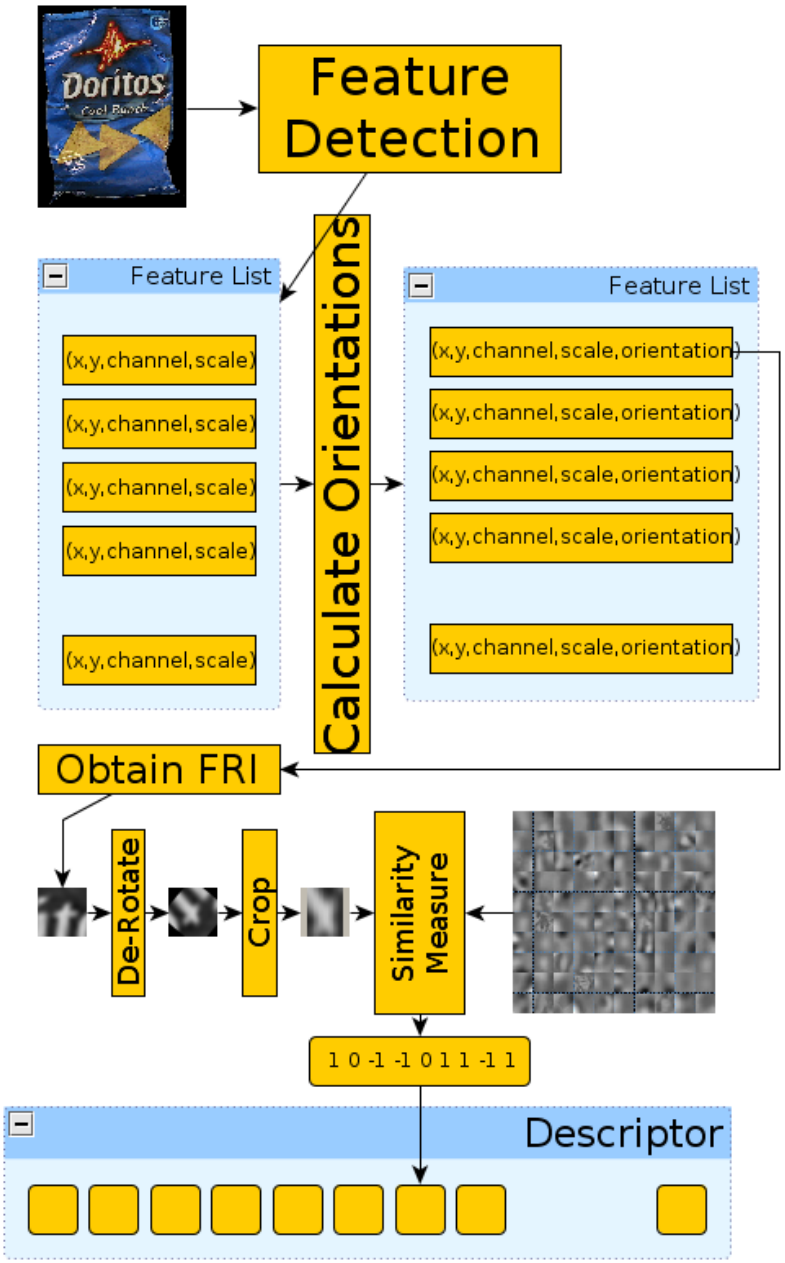
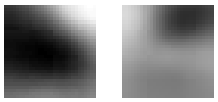


Figure 4: The process flow of the on-line portion of the BASIS descriptor.



(a) (b)

Figure 5: An example basis dictionary image (a) and FRI (b). Portions of the basis dictionary match the FRI well, but other regions do not. A region-aware similarity measure is needed to record this information.

Because the FRI and the β_i are both represented as positive intensity values from 0-255, the result of the subtraction is a matrix (or *polarity image*) with values in the range [-255,255]. Each value in the matrix corresponds to the *polarity* of a pixel. A positive polarity value indicates that the pixel intensity of the basis image is brighter than the intensity of the corresponding pixel in the FRI. A negative polarity value means the basis image pixel is darker than the corresponding FRI pixel. A zero polarity value represents equal intensity values of the two corresponding pixels. For clarity of discussion, we will normalize these integer values to real numbers in the range [-1,1]. Polarity allows us to measure the similarity between an FRI and each of the basis images in the dictionary. This is described below. This resulting polarity image is segmented

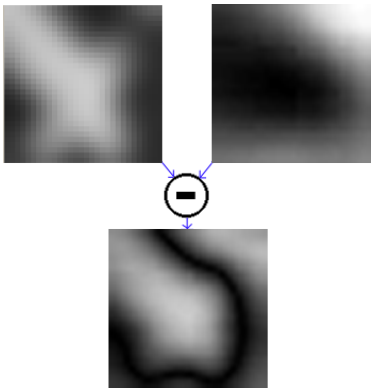


Figure 6: The first step in calculating the BSM is to perform an element-wise subtraction of the FRI from the basis dictionary image. The matrix, with the difference values normalized to the range [-1,1] is called the polarity image.

into 9 equal-sized regions (Fig. 7). In our example using 30x30 pixel FRIs, the region size is 10 pixels by 10 pixels. Each of the 9 regions is then polled for its *polarity bias*, a ternary quantization of the pixel's value. If a pixel's polarity value (between -1 and 1) in the 10×10 region has a value less than a small negative threshold $-\epsilon$, it casts a vote of -1 (negative polarity). If the value is

greater than a small positive threshold ϵ, it casts a vote of 1 (positive polarity). If the value is in between -ϵ and ϵ (very small difference between the basis image and FRI pixels), it casts a vote of 0 (zero polarity). The votes for all 100 pixels are tallied, and the majority vote is the resulting regional vote of the 10x10 pixel region. This process generates 9 regional votes for the 9 10×10 regions. These 9 regional votes are then concatenated into a 9 ternary digit value, and saved as a descriptor element that describes the similarity between an FRI and one basis image. Recall that there are a total of 384 basis images (128 per color channel) in a dictionary, which results in 384 9-ternary-digit descriptor elements. These 384 descriptor elements are then stored in a 384-element array as the feature's descriptor. Although this descriptor has three times as many elements as the SIFT descriptor, each element is only 9 ternary digits long, and uses only 18 bits, for a total of 864 bytes per descriptor, a reduction in memory requirements of 71.9% per descriptor element.

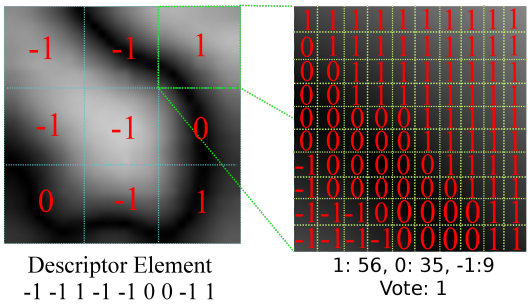


Figure 7: Similarity measures are computed for a polarity image by allowing each pixel in a 10x10 region to vote based on its thresholded value. The majority vote for a 10x10 region is the overall vote for that region, and the 9 regional votes are concatenated into a 9-ternary-digit descriptor element.

3.4 Comparing Descriptors

The descriptor element values for each feature are 9 ternary digits, and thus cannot naturally be represented using an integer or floating point value, which means standard comparison metrics such as Euclidean or Mahalanobis distance (which both require complicated computations) cannot be used. We develop a unique similarity measure similar to a Hamming distance to quickly compare BASIS descriptors of two features. The proposed BASIS descriptor allows us to find the best matched feature points between two consecutive frames. If two BASIS descriptor element regional votes have the same polarity, i.e., (0, 0) , (1, 1), or (-1, -1), they can be considered an excellent match, because they both matched equally well to a given basis image region. However, if one regional

vote has a zero polarity but the other has either positive or negative polarity, i.e., (0, 1) or (0, -1), we can still consider this a good match. Conversely, if the two BASIS descriptor element regional votes have opposing polarities, i.e., (1, -1), we now know that the regions are an even worse match because one FRI matched the β_i, and one did not.

Equation 2 shows an example descriptor element similarity calculation. For each element in the descriptor, our similarity measure sums the absolute-values of no-carry subtractions of each of the 9 ternary digits.

$$\begin{array}{cccccccccc} 0 & 0 & 0 & 1 & 1 & 1 & -1 & -1 & -1 & \\ 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & \\ \hline \sum(& 0 & 1 & 1 & 1 & 0 & 2 & 1 & 2 & 0 \ ) = 8 \end{array} \tag{2}$$

The subtractions result in 9 digits where each digit is a member of the set 0,1,2. These individual digits are summed together to create the similarity indicator. A similarity indicator close to zero implies an excellent match, and a higher valued similarity indicator indicates a poor match. The similarity indicators for all 384 descriptor elements are then summed, and the resulting difference value (called the *similarity indicator sum*, or SIS) provides a comparison between two features.

4 Hardware Implementation

To prove the feasability of developing a hardware version of the BASIS descriptor algorithm, we created top-level hardware designs, tested the output using a bit-level accurate software version of the BASIS descriptor, and developed the system in VHDL. Figure 8 shows the top level of our hardware design. Data from an image sensor is fed into a feature detection system. There are a large number of feature detection cores that have already been developed in VHDL, so we do not outline how the detection step is performed. The output of the feature detection core is a list of x and y locations in the image where features were detected.

As soon as features are returned from the detector, the descriptor stage can begin computing descriptors. Counters iterate over each feature in the list, and over each image in the basis dictionary. Figure 9 shows the image retrieval stage for an FRI. The retrieval stage for a dictionary image is almost identical. The feature x and y locations (obtained from the list in memory) and the image location in memory, along with the image parameters (height, width, and depth) are used to compute the starting address in memory of the FRI. The FRI height and width are then used, along with the image parameters, to calculate new addresses as the FRI is read out of memory. The FRI data out is fed into nine 8 pixel by 8 pixel regional block RAM (BRAM) memory units. An FRI row counter and column counter keep track of the current row and column of the 3x3 block of regional BRAMs, and the output of these counters is fed into a MUX controlling the write enable lines on all nine regional BRAMs using the pseudo code in Algorithm 1.

The output of one regional BRAM from the FRI and one regional BRAM from the Basis Dictionary Image (BDI) are connected to one of nine differencing blocks, as shown in Fig. 8. The details of the differencing block are shown in Fig. 10. The differencing block contains three main components: an addressor, a voter, and a vote accumulator. The addressor retrieves one pixel from the BDI BRAM and one pixel from the FRI BRAM and passes them into the voter. The voter block subtracts the BDI pixel value from the FRI pixel value and thresholds the result, returning a vote of 1, 0, or -1. This vote is passed out of the voter on three separate, XOR pins, vote(-1), vote(0), and vote(1). When the voter is finished, it returns a done signal to the addressor and the vote accumulator. The vote accumulator consists of three adders that add up the individual votes. When the addressor has reached the end of the BRAMs, the resulting `argmax` from the accumulator (-1, 0, or 1) is returned from the differencing block. The nine regional votes from the nine differencing blocks create one descriptor element (DE). Once all nine blocks are finished, the DE is pushed onto a FIFO until the entire descriptor has been computed. Once the descriptor FIFO has received all of the DE's, the FIFO contains a complete descriptor, and is appended with the feature's (x, y) location and stored in the

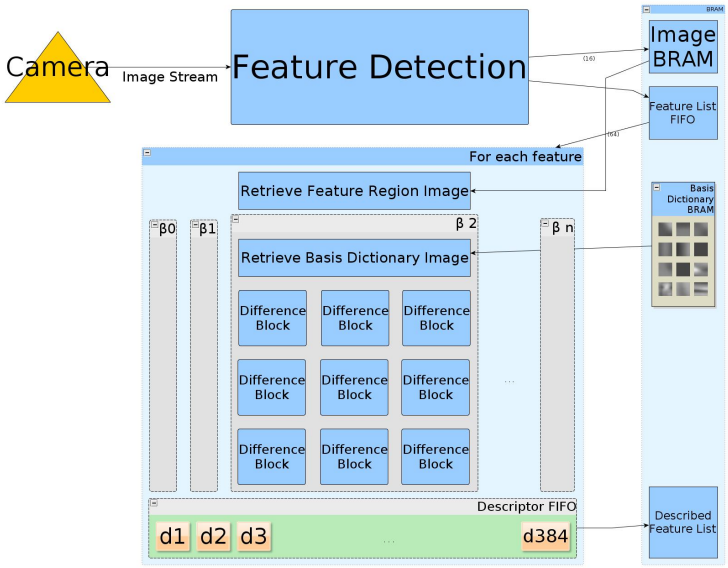


Figure 8: The top-level hardware design of the BASIS descriptor. Each feature from a feature detector core is processed through the FRI-Basis Dictionary Image comparator section, and the resulting descriptor is stored in memory along with the (x, y) location of the feature in a described feature list. There is a potential for paralellizing the FRI-Basis Dictionary Image comparator section, providing even greater speedup.

Algorithm 1 Regional BRAM counter and write-enable logic.

```
addr ← rowcount * 23 + colcount
if rowcount < 8 and colcount < 8 then
  writeEnable ← "0001"
else if rowcount >= 8 and rowcount <= 16 and colcount < 8 then
  writeEnable ← "0002"
end if
if plboutvalid = '1' then
  colcount ← colcount + 1
else
  colcount ← colcount
end if
if colcount = 24 then
  colcount ← 0
  rowcount ← rowcount + 1
else
  rowcount ← rowcount
end if
if rowcount = 24 and colcount = 2 then
  done ← '1'
end if
```

described feature list.

4.1 Bit-level Accurate Software

In order to expose any disadvantages that adaptation into hardware may require, we developed a bit-level accurate software version of the BASIS descriptor. This version allowed us to compare results from the original software BASIS descriptor algorithm and the hardware version. Fortunately, the BASIS descriptor computation is not complex and does not require any high-level operators, so it can be modified easily for an FPGA. BASIS descriptor computation requires only subtraction, thresholding, and summation operators, all of which can be implemented in VHDL in a single clock cycle. Also, BASIS descriptor elements consist of ternary digits, not floating point numbers. These ternary digits are easily represented using only 2 bits per digit. Computing and storing the descriptor elements therefore requires no change from the original software design. The only modification to our software algorithm we made to provide bit-level accuracy was to modify the basis dictionary image and feature region image dimensions from 30 pixels by 30 pixels to 24 pixels by 24 pixels. Using 24×24 image sizes allowed us to segment the regional BRAMS so that each would hold 8 pixels, making the regional BRAM addresses require 3 bits instead of 4, and reducing slightly the amount of memory required to hold the basis dictionaries.

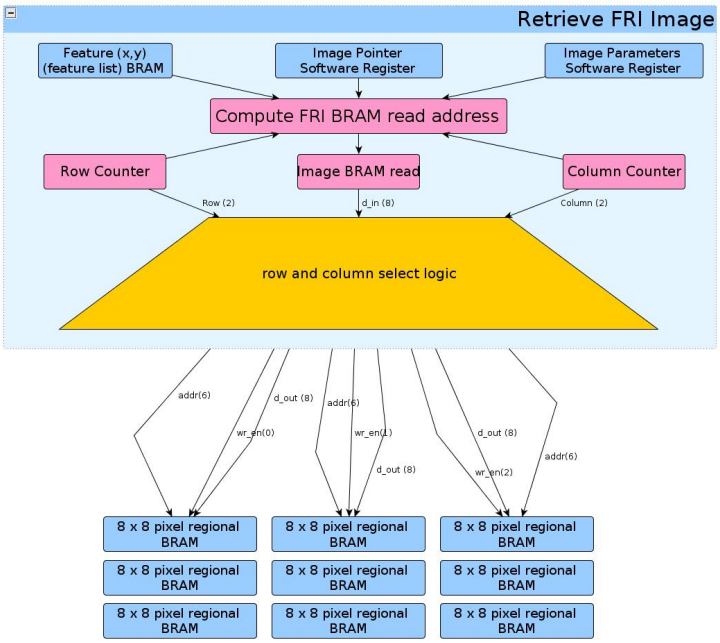


Figure 9: The FRI retrieval process. Using the feature list, the exact addresses in memory are calculated for BRAM transaction from the large image BRAM into the smaller BRAMs. Each BRAM holds an 8x8 pixel section of the full 24x24 pixel FRI and is called a regional BRAM.

4.2 Performance

Standard descriptor matching algorithms will compute distances between descriptors and return sets of matches from one image to the next. The BASIS descriptors and similarity measures can be used as input to any number of matching algorithms. Understandably, the use of any particular match finding algorithm would provide additional discernment (in order to choose matches) that could possibly mislead the reader as to the accuracy of the descriptor itself. For example, in object detection research, frequently researchers will avoid performing full image detection so that they can compare object detection accuracy independent of non-maximal suppression methods needed to do full image detection. To avoid this matching performance boost, we apply a very simple technique for testing the accuracy of our BASIS descriptor, and compare it to other feature descriptors. We use the OpenCV RANSAC algorithm *FindHomography* to compute a homography based on the detected features and resulting descriptor matches.

First, features are found in image $\mathbf{I}_1$ using the FAST feature detector. Our algorithm then computes BASIS descriptors for each feature. The same process

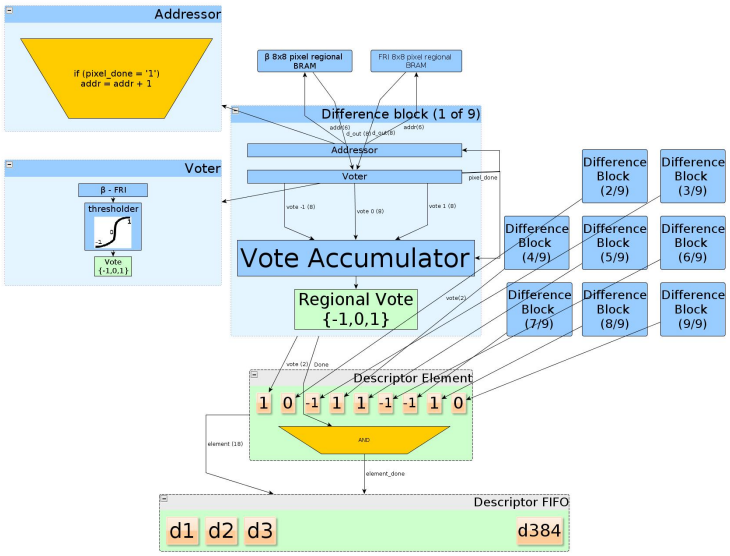


Figure 10: The differencing block is the main component of the BASIS descriptor. It differences an 8x8 pixel section and returns a regional vote for the section. The 9 differencing block vote outputs are then concatenated into an 18-bit descriptor element (DE).

happens with the second image, $\mathbf{I}_2$. Next, a similarity indicator sum (SIS) is calculated between each feature in $\mathbf{I}_1$and each feature in $\mathbf{I}_2$ and the results are sorted by lowest SIS value (closest descriptor match).

Next, our algorithm applies a uniqueness constraint which removes any features found in $\mathbf{I}_1$ whose closest match in $\mathbf{I}_2$ (top element in the sorted list) is the closest match for more than one feature in $\mathbf{I}_1$. This constraint removes features that did not provide enough information to create a good descriptor. As a verification step, the remaining features from $\mathbf{I}_1$ (those whose top match is not the top match of any other feature) and their associated match are input into the RANSAC algorithm in order to compute a homography, H relating $\mathbf{I}_1$to $\mathbf{I}_2$. The feature points from $\mathbf{I}_1$ are then warped by the homography using

$$p_2 = H * p_1 \tag{3}$$

$\mathbf{I}_2$ is then searched in the vicinity of p_2 and the closest feature is connected with a line, and the resulting "matches" image shown to the user. Using this method, it is obvious if the algorithm computes an incorrect homography, and thus easy to measure accuracy. If a homography is correct, feature points will easily match up and all lines will follow a common direction that is synonymous with the movement of the camera. If the algorithm cannot provide enough well-matched points to compute a correct homography, the resulting homography

will place all or most of the p_2 in incorrect locations, and these mismatches are obvious upon quick visual inspection. The BASIS descriptor utilizes the FAST feature detector, so we do not compare repeatability rates of FAST to other detectors in this paper, as those results can be found in previously published articles.

In order to compare the performance of the software BASIS descriptor to some commonly used detector/descriptor systems, we modified our code to allow for the use of any OpenCV implemented detector and descriptor. This allowed us to change the descriptor and/or detector, but keep the post-processing RANSAC step constant to allow an unbiased comparison. Our dataset, Idaho, was created from real world images taken from an actual air flight. The same variables and parameters were used across all examples. The images in the Idaho dataset were taken from a camera running at 30 frames per second, 640x480 pixel resolution. The images used for the dataset were obtained from video frames that are one second apart to allow noticeable camera movement. With this large gap between frames equating to larger movements and feature location shifts, the BASIS descriptor still performed very well on real images from a UAV-style application. In Figure 11, our bit-level accurate version of the BASIS descriptor found matches, and RANSAC correctly computed the homography shown. Figures 12 and 13 show two more examples from this dataset where the correct homography was computed.

To test our descriptors, we ran the same images through the system using BASIS, hardware-BASIS, the SIFT detector and descriptor and the SURF detector and descriptor. We ran 65 images through all four algorithms, and tallied for how many image pairs each algorithm computed a correct homography. The Idaho test set features large blank areas of fields with few features, populated urban scenes, and natural features such as mountains and rivers. While the movement between frames is mostly translation and rotation, obviously some perspective warping is present as a result of the plane banking in the air. SIFT performed the poorest in our tests, achieving 32% accuracy. SURF and the software BASIS descriptor performed very well on the dataset. Using SURF, the RANSAC algorithm was able to compute an accurate homography 63% of the time. The software BASIS descriptor performed slightly better. Using the software BASIS descriptor, RANSAC calculated the correct homography for 65% of the images. The bit-level accurate BASIS descriptor, hardware-BASIS, achieved a similar accuracy of 66%. Due to the very small number of modifications required to make the BASIS descriptor hardware-ready, there is no loss of accuracy between sofware and hardware implementations. It is noted that for our application, the 65% accuracy on aerial images with very few features is considered very significant and is more than enough for air vehicle pose estimation or image registration.

Additionally, average memory usage per image was calculated assuming that 800 features are kept for each image for all descriptors. BASIS descriptors require the least amount of memory space. Table 1 shows the results of the three software implementations, SIFT, SURF, and BASIS, and the bit-level accurate hardware-BASIS. As Table 1 shows, this change did not negatively

Figure 11: Resulting homography created by the bit-level-accurate version of the BASIS descriptor.

affect overall performance of the descriptor.

4.3 Memory Requirements and Speed

Our FPGA vision system consists of an image sensor that provides image data for a 640x480 pixel resolution image at a rate of 25MHz. Each pixel is 16 bits of RGB565 standard data. The pixel data is fed directly into the feature detection core, but can also be stored in SRAM for other applications to use. The hardware-BASIS algorithm was coded into VHDL, and all memory modules were declared as inferred BRAMs or LUT-RAMs on the FPGA fabric. In addition, BRAM storage is included for two full 640x480 16-bit frames and two 1000-element feature lists from the detector to allow for double-buffering of feature detector output. However, no memory was allocated for the feature detector because off-the-shelf FPGA feature detectors are readily available, and the purpose of this experiment was to show the feasibility of implementing the descriptor in hardware.

Due to the large amount of data being moved through the system, the main bottleneck of this hardware implementation is memory bandwidth and capacity. The system requires enough room in FPGA memory to hold two full image

Figure 12: Resulting homography created by the bit-level-accurate version of the BASIS descriptor.

frames, two feature lists, all required basis dictionaries, and the described feature lists. Table 2 shows the memory requirements of our system. Images are saved into memory as 16 bit (RGB565) images at 640 x 480 pixel resolution, requiring 4915200 bits, or 614400 bytes each. Although the size of the feature list varies depending on the detector used and the number of features found in the image, we make some basic assumptions for the sake of completeness. For our calculations here, we will limit the output of the feature detector to 1000 features. Each feature contains an x, y location. At 640x480 resolution, the x and y values can be represented using 10 bits. 1000 20-bit values require 2,500 bytes. Hardware-BASIS uses 128 24x24 pixel basis dictionaries per channel. Each pixel in these basis dictionaries is one byte. There are 73,728 pixels total in a basis dictionary for a given channel. To hold all three basis dictionaries in memory requires 221184 bytes. Finally the described feature list is also stored in BRAM. This described feature list contains 384 18-bit descriptors (9 ternary digits) and the (x, y) location of the feature for a total of 6932 bits per feature. With 500 features, this requires 433250 bytes for the described feature list.

The FPGA we have used for our research is a Virtex-6 VLX195T. The Virtex-6 VLX195T contains 688 18 Kb BRAMs, for a total of 12.384Mb of block ram on chip. We store the image buffers, feature list, and basis dictionary in BRAM.

Figure 13: Resulting homography created by the bit-level-accurate version of the BASIS descriptor.

We also store a portion of the resulting described feature list (before pushing it out to external memory), the 9 regional BRAMs for the FRI and the 9 regional BRAMs for the current BDI. These regional BRAMs hold 64 8-bit pixels each, for a total of 64 bytes per BRAM, requiring 1,152 bytes for all 18 BRAMs. Also stored in BRAM is the descriptor FIFO, which holds 384 descriptor elements, for a total of 864 bytes. The available memory in BRAM on the VLX195T allows us to satisfy all of the hardware-BASIS memory needs on chip in local memory, avoiding costly memory bottlenecks.

Because we are able to implement all required memories in FPGA BRAM or LUT-RAM, the Xilinx synthesis tools report that we are able to clock the entire descriptor algorithm in hardware at 400MHz. A typical FPGA feature detector will run at up to 60 frames per second, and the image sensor we use for our research outputs 640x480 pixels on a 25MHz clock. The logic utilization on the Virtex 6 is shown in Table 3. The descriptor system easily fits into the logic in the Virtex 6, leaving room for the feature descriptor components, thus providing a feature detection and description system that is wholly contained in FPGA fabric.

Algorithm	Avg. # of features per image	Average memory usage per image	Homography accuracy
SIFT	1200	819KB	32.099%
SURF	1200	819KB	60.000%
BASIS	1800	691KB	65.000%
Hardware-BASIS	1800	691KB	66.000%

Table 1: Accuracy results for all three software algorithms and the BASIS hardware adaptation on the Idaho dataset.

Object	Qty	Size (bytes)	Total Size
Image	1	614K	614K
Feature List	1	2,500	2.5K
Basis Dictionary	3	73.73K	221.2K
Described Feature List	1	433.3K	433.3K
Regional BRAM (BDI)	9	64	576
Regional BRAM (FRI)	9	64	576
Descriptor FIFO	1	864	864
Total:			1,273K

Table 2: Memory usage on the FPGA for the BASIS descriptor algorithm.

Slice Logic Utilization:		
Number of Slice Registers:	986 out of 249,600	1%
Number of Slice LUTs:	2,177 out of 124,800	1%
Number used as Memory:	536 out of 48,640	1%
Number used as Dual Port RAM:	216	
Number used as Single Port RAM:	320	
Slice Logic Distribution:		
Number of occupied Slices:	610 out of 31,200	1%
Number of LUT Flip Flop pairs used:	2,218	
Number with an unused Flip Flop:	1,241 out of 2,218	55%
Number with an unused LUT:	41 out of 2,218	1%
Number of fully used LUT-FF pairs:	936 out of 2,218	42%
Number of unique control sets:	62	

Table 3: FPGA device utilization report for the BASIS descriptor algorithm on a Virtex 6 LXT195 FPGA.

5 Conclusion

The BASIS descriptor provides a unique method of describing feature points based on the characteristics they contain. The use of sparse coding algorithms to obtain basis dictionaries provides the BASIS descriptor with a dictionary set that resembles the receptive fields found in the visual cortex. These basis sets were found to be very similar across a spectrum of natural and non-natural training images. We have shown that BASIS descriptors perform well in the task of frame-to-frame image registration. BASIS descriptors are more space efficient than SIFT or SURF descriptors.

In this paper we have presented the hardware adaptation of the BASIS descriptor for application on a Virtex 6 VLX195 FPGA platform. With very little modification, we were able to implement the entire BASIS descriptor calculation in FPGA fabric without the need for a CPU or soft core CPU. The simplicity of the BASIS descriptor calculations allowed us to implement the entire BASIS descriptor system in such a way that it will operate at high clock rates, allowing the feature detector and image sensor to operate at full speed (60 fps in our application). The BASIS descriptor bit-level accurate software implementation results show that modifying the BASIS descriptor for hardware application resulted in no loss of accuracy.

Now that the entire detector and descriptor algorithms have been implemented in hardware, our future work will involve developing a new correlation method that can also be adapted to a hardware application. By developing a hardware correlation system, the entire vision system can be implemented in FPGA hardware, creating a complete system-on-a-chip solution for small, light-weight, embedded platforms.

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: speeded up robust features. *Computer Vision and Image Understanding*, 110:346–359, 2008.

[2] V. Bonato, E. Marques, and G. A Constantinides. A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(12):1703–1712, 2008.

[3] M. Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer Verlag, 2010.

[4] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[5] D. J Field. What is the goal of sensory coding? *Neural Computation*, 6(4):559–601, 1994.

[6] Wade S. Fife and James K. Archibald. Reconfigurable on-board vision processing for small autonomous vehicles. *EURASIP Journal on Embedded Systems*, 2007:Article ID 80141, 14 pages, 2007.

[7] Spencer Fowers. An effective color addition to feature detection and description for book spine image matching. *ISRN Machine Vision*, 1(1), 2011. to appear.

[8] Spencer Fowers, Dah-Jye Lee, and Doran K. Wilde. Color DoG: a three-channel color feature detector for embedded systems. In Ernest L. Hall David P. Casasent and Juha Röning, editors, *Intelligent Robots and Computer Vision XXVII: Algorithms and Techniques*, volume 7539, page 1, 2010.

[9] H. C Garcia, J. R Villalobos, and G. C Runger. An automated feature selection method for visual inspection systems. *IEEE Transactions on Automation Science and Engineering*, 3(4):394–406, 2006.

[10] A. Gersho and R. M Gray. *Vector Quantization and Signal Compression*, volume 159. Springer Netherlands, 1992.

[11] Chun-Rong Huang, Chu-Song Chen, and Pau-Choo Chung. Contrast context histogram - a discriminating local descriptor for image matching. In *Proceedings of the 18th International Conference on Pattern Recognition*, volume 4, pages 53–56, 2006.

[12] De-Shuang Huang, Donald C. Wunsch, Daniel S. Levine, and Kang-Hyun Jo. Image reconstruction using a modified sparse coding technique. In *Advanced Intelligent Computing Theories and Applications—With Aspects of Theoretical and Methodological Issues*, volume 5226 of *Lecture Notes in Computer Science*, pages 220–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[13] H. Huang and N. Wu. Fast facial image super-resolution via local linear transformations for resource-limited applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1363–1377, 2011.

[14] A. Hyvarinen, E. Oja, P. Hoyer, and J. Hurri. Image feature extraction by sparse coding and independent component analysis. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, volume 2, pages 1268–1273, 1998.

[15] Y. Kim, V. Jayanthi, and I. Kweon. System-on-chip solution of video stabilization for CMOS image sensors in hand-held devices. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1401–1414, 2011.

[16] H. Le Borgne, A. Guerin-Dugue, and N. E O'Connor. Learning midlevel image features for natural scene and texture classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(3):286–297, 2007.

[17] K. Lillywhite, D. Lee, B. Tippetts, S. Fowers, A. Dennis, B. Nelson, and J. Archibald. An embedded vision system for an unmanned four-rotor helicopter. In *PROCEEDINGS-SPIE THE INTERNATIONAL SOCIETY FOR OPTICAL ENGINEERING*, volume 6384, page 63840, 2006.

[18] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[19] J. Mairal, M. Elad, and G. Sapiro. Sparse representation for color image restoration. *IEEE Transactions on Image Processing*, 17(1):53–69, 2008.

[20] B. A Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[21] B. A Olshausen and D. J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

[22] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proceedings of the Tenth IEEE International Conference on Computer Vision*, volume 2, pages 1508–1515, 2005.

[23] J. Svab, T. Krajnik, J. Faigl, and L. Preucil. FPGA based speeded up robust features. In *Proceedings of the IEEE International Conference on-Technologies for Practical Robot Applications*, pages 35–41, 2009.

[24] B. Tippetts, D. Lee, J. Archibald, and K. Lillywhite. Dense disparity real-time stereo vision algorithm for resource limited systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1547–1555, 2011.

[25] B. Tippetts, K. Lillywhite, S. Fowers, A. Dennis, D. J Lee, and J. Archibald. A simple, inexpensive, and effective implementation of a vision-guided autonomous robot. In *Proceedings of SPIE*, volume 6384, page 63840P, 2006.

[26] Beau J. Tippetts, Dah-Jye Lee, Spencer G. Fowers, and James K. Archibald. Real-Time vision sensor for an autonomous hovering micro-UAV. *Journal of Aerospace Computing, Information, and Communication*, 6(10):570–584, 2009.

[27] Yun-Ta Tsai, Quan Wang, and Suya You. CDIKP: a highly-compact local feature descriptor. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4, 2008.

[28] Z. Wei, D. Lee, and B. E Nelson. A hardware-friendly adaptive tensor based optical flow algorithm. *Lecture Notes in Computer Science*, 4842:43, 2007.

[29] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In Jan-Olof Eklundh, editor, *Proceedings of the Third European Conference on Computer Vision*, volume 2, pages 151–158. Springer-Verlag, Berlin/Heidelberg, 1994.