Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2014, Article ID 606210, 12 pages
http://dx.doi.org/10.1155/2014/606210

Research Article

TreeBASIS Feature Descriptor and Its Hardware Implementation

Spencer Fowers, Alok Desai, Dah-Jye Lee, Dan Ventura, and James Archibald

Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602, USA

Correspondence should be addressed to Dah-Jye Lee; djlee@byu.edu

Received 20 January 2014; Revised 11 August 2014; Accepted 20 October 2014; Published 10 November 2014

Academic Editor: Martin Margala

This paper presents a novel feature descriptor called TreeBASIS that provides improvements in descriptor size, computation time, matching speed, and accuracy. This new descriptor uses a binary vocabulary tree that is computed using basis dictionary images and a test set of feature region images. To facilitate real-time implementation, a feature region image is binary quantized and the resulting quantized vector is passed into the BASIS vocabulary tree. A Hamming distance is then computed between the feature region image and the *effectively descriptive basis dictionary image* at a node to determine the branch taken and the path the feature region image takes is saved as a descriptor. The TreeBASIS feature descriptor is an excellent candidate for hardware implementation because of its reduced descriptor size and the fact that descriptors can be created and features matched without the use of floating point operations. The TreeBASIS descriptor is more computationally and space efficient than other descriptors such as BASIS, SIFT, and SURF. Moreover, it can be computed entirely in hardware without the support of a CPU for additional software-based computations. Experimental results and a hardware implementation show that the TreeBASIS descriptor compares well with other descriptors for frame-to-frame homography computation while requiring fewer hardware resources.

1. Introduction

Finding and matching local regions of interest or "features" in real-time is of increasing interest on low power, low resource embedded systems. Relevant computer vision applications involve target tracking [1, 2], object identification [3], optical flow [4], stereo vision [5], super resolution [6], image stabilization [7], and image rectification, localization, and pose estimation [1, 8–11]. These vision-based applications require high quality feature description and matching. Feature descriptors such as SIFT, SURF, FAST, Star, MSER, and Sparse-Coding Inspired Similarity (BASIS) were designed for these computations.

Although robust and innovative, most associated algorithms cannot be considered hardware friendly. Implementing these most commonly used descriptors in custom logic circuits on a Field Programmable Gate Array (FPGA) is a challenging task, since a full floating-point unit is required. For this reason, hardware implementations of these feature descriptors typically offload the actual descriptor computations to a CPU for floating-point operations [10] or they rely on a simplified algorithm (e.g., using fixed point computation) that maps more readily to hardware [4, 12].

The main challenge with commonly used feature descriptors is that they required substantial memory and computing power. More recently proposed descriptors address these drawbacks. New schemes based on intensity comparisons include Binary Robust Independent Elementary Features (BRIEF) [13, 14], Binary Robust Invariant Scalable Keypoints (BRISK) [15], and Speeded Up Surround Extrema (SUSurE) [16]. BRIEF and BRISK use much smaller descriptors and faster feature detectors than SIFT and SURF; BRIEF has shown improved performance over SIFT and SURF for object recognition. Other approaches based on dimensional reduction techniques have been developed, such as Linear Discriminant Embedding (LDE) [17] and Principal Component Analysis (PCA) [18]. Another alternative to SIFT and SURF is a specific set of 256 learned pixel pairs selected for reducing correlation among the binary tests [19].

We previously described a nonfloating point algorithm called BASIS that was designed specifically for FPGA implementation [20]. The sparse coding basis dictionaries used by

BASIS are nonorthogonal and individual basis images may contain redundant information [21] that reduces the accuracy and increases the descriptor size. To reduce the redundant information, the BASIS descriptor can be improved by implementing a novel computation method based on a vocabulary tree. The new TreeBASIS algorithm that we present in this work improves and extends this idea. The new TreeBASIS algorithm is even better in resource usage and accuracy than BASIS. With significantly reduced descriptor size, increased efficiency, and simplified calculations, it is fully implementable on a limited-resource FPGA platform while providing better feature descriptor accuracy than previously proposed schemes.

The TreeBASIS descriptor is designed specifically for hardware implementation so no simplifications are required that can compromise performance in realized systems. Although our target system is a small FPGA device based on a Virtex-6 VLX760 FPGA, the approach is well suited for a wide variety of commonly used devices. Our vision system will be used on a small unmanned ground vehicle (UGV) and an unmanned aerial vehicle (UAV) for image stabilization, rectification, and pose estimation.

This paper describes the development and implementation of the TreeBASIS feature descriptor. The approach uses a derivative of sparse coding with a novel vocabulary tree-based descriptor computation method. A vocabulary tree is created by using a small basis dictionary to partition a training set of feature region images (FRIs), small pixel regions surrounding detected feature points. The vocabulary tree is computed off-line and stored in memory for on-line descriptor computation and matching. Basis dictionary images (BDIs) are quantized, and their intensity values are represented using a binary vector. TreeBASIS feature descriptors are computed by quantizing an FRI, passing it through the tree, and recording its path.

Section 2 presents an overview of the TreeBASIS descriptor. Section 3 describes hardware components that perform the associated computation and matching algorithms, including detailed synthesis reports and discussion on FPGA size and memory requirements. Section 4 includes a discussion of conclusions and future work.

2. TreeBASIS

In the TreeBASIS approach, a vocabulary tree is created using a small basis dictionary to partition a training set of FRIs. This vocabulary tree is computed off-line and stored for on-line descriptor computation and matching. TreeBASIS computes features descriptors by passing an FRI through the tree and recording its path. The matching of descriptors between images is achieved by traversing the descriptor-paths of features from the first image and comparing each node to the descriptor-path of the features from the second image.

TreeBASIS consists of three major components: building the vocabulary tree, computing descriptors, and matching descriptors between two images. The tree can be created off-line on a standard desktop computer that is completely separate from the on-line system. Once the tree has been created, it can be loaded into memory and used in real-time on a low-resource platform to compute TreeBASIS descriptors and match them in subsequent images.

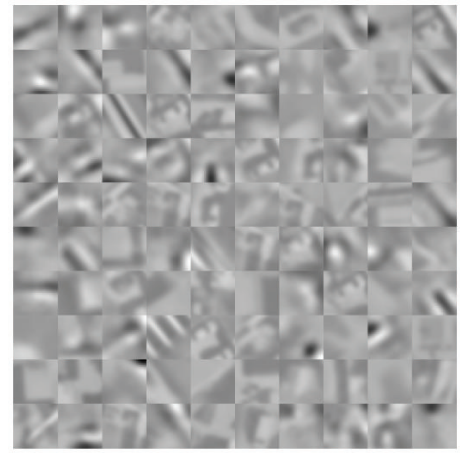


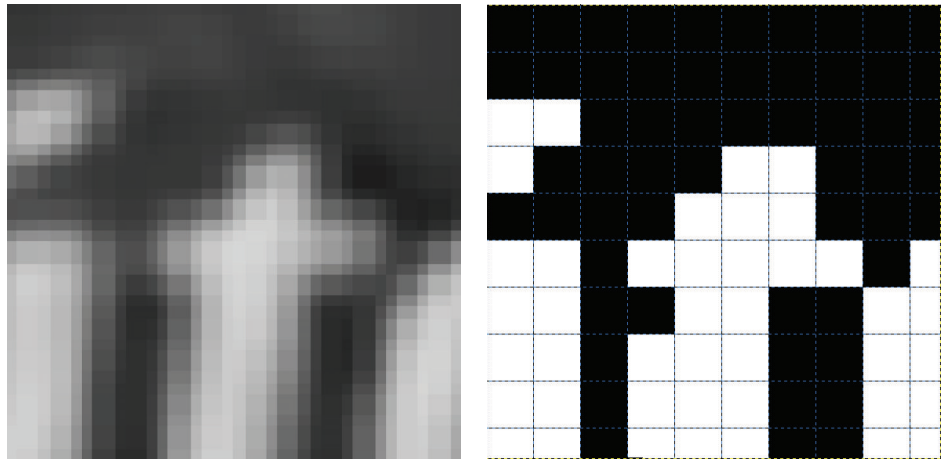
Figure 1: The basis dictionary, B, returned from K-SVD using the entire GoogleMaps dataset.

2.1. Off-Line Tree Creation. The off-line tree creation stage of TreeBASIS utilizes a dictionary of basis images returned from the K-SVD sparse coding algorithm [22] to partition a training set of FRIs (30×30 pixel regions surrounding a detected feature) denoted as F. Sparse coding theory states that if K-SVD is trained using a very large dataset of images, the basis dictionary, B, returned by K-SVD can be utilized to reconstruct a great assortment of natural images [23]. An example of such a basis dictionary is shown in Figure 1, obtained from the GoogleMaps dataset. Sparse coding theory suggests that this basis dictionary would be useful in describing *any* such set of natural images.

The dictionary set, B, is used to partition the training set, F, consisting of at least 50,000 FRIs, each 30×30 pixels. Each FRI is taken from an image or video frame near a feature point detected by the feature detector. The 30×30 region size was empirically determined to produce the best results. Larger regions tend to contain too much data or content not associated with pixels associated with feature points. Smaller regions tend to retain too little useful information and result in FRIs that are not unique. Experiments using the hardware implementation of the BASIS descriptor show a reduction in accuracy if the size of the FRI is reduced [20].

When a training set of images is chosen similar to those expected to be encountered during real-time, then the feature matching accuracy can improve [24]. However, the generic nature of the basis dictionary images, B, and the FRIs provide some invariance to the differences between the training set images and the images the algorithm may see in real-time. The TreeBASIS algorithm will result in a high level of feature matching accuracy on completely different image datasets, even if Fconsists of more generic training images.

In order to reduce computation time, the BDIs and FRIs are divided into subregions and the average value in each



(a) (b)

Figure 2: An example of the binary thresholding of an FRI. This approach reduces the memory footprint of the image from 900 8-bit pixels to a 100-bit vector. The smaller vector requires 100 1-bit hardware comparisons, instead of 900 8-bit comparisons.

subregion is calculated. Based on that average value, binary thresholding is performed (as shown in Figure 2). The first step in thresholding a BDI or FRI is computing the global average gray value of the entire 30×30 region as

$$g = \frac{\sum_{x,y} I(x, y)}{p}, \tag{1}$$

where p is the number of pixels in the image (900 in this case) and $I(x, y)$ is the intensity value at pixel (x, y). Next, the FRI or BDI is divided into 100 3×3 subregions and the intensity values of each pixel in a region are averaged. If the resulting average intensity of each 3×3 subregion is greater than the global average intensity g, the subregion is set to one; otherwise the subregion is set to zero. This results in a 100-element (single bit) binary quantized vector that is then used in place of the original BDI or FRI. This operation provides two benefits. The comparison against g provides invariance to shadows, shading, and highlights. By using a single-bit binary vector, the number of comparisons at each stage of the computation is reduced, along with drastically reducing the memory footprint of each BDI and FRI.

Relative to the original BASIS algorithm, this approach requires significantly less computation and memory to compare an FRI from an image with a BDI from the dictionary. In addition to quantizing the BDIs and FRIs, another major advantage TreeBASIS provides is that it utilizes a binary tree structure to avoid comparing each FRI with every BDI in the dictionary.

The tree creation process is illustrated in Figure 3. Each node of the tree is created by taking a set of training FRIs (starting with F) and determining the *most effectively descriptive* BDI (EDBDI), β_{ED}, from the dictionary B. An EDBDI is that BDI from a dictionary that most evenly partitions a given set of FRIs. The EDBDIs are the BDIs that most effectively portray critical feature characteristics because half of the FRIs match them closely, while the other half do not. By building a tree in this manner, the number of FRI-BDI comparisons is drastically reduced. Moreover, the criteria for inclusion in the tree ensure that FRIs will only be compared with those BDIs most likely to help distinguish this FRI from all other FRIs that may appear.

Given sets $B = \{\beta_0 \cdots \beta_n\}$ and $F = \{f_0 \cdots f_n\}$, we define the entropy over set F with respect to β ($\beta \in B$) as

$$E_\beta(F) = -p_L \log_2 p_L - p_R \log_2 p_R, \tag{2}$$

where

$$\begin{aligned} p_L &= \frac{|F_L|}{|F|}, \\ p_R &= \frac{|F_R|}{|F|}, \\ F_L &= \left\{ f \mid f \in F, h(f, \beta) \leq \frac{|\beta|}{2} \right\}, \\ F_R &= \left\{ f \mid f \in F, h(f, \beta) > \frac{|\beta|}{2} \right\} \end{aligned} \tag{3}$$

and where $h(x, y)$ is the Hamming distance between x and y. Conceptually, p_L is the portion of F whose Hamming distance is less than or equal to $|\beta|/2$, while p_R is the portion of F whose Hamming distance is greater than $|\beta|/2$. $|\beta|$ is the number of bits in β, the number of quantized regions. $|\beta|$ is set to 100 in our implementation. The Hamming distance is used in order to retain spatial similarity information from the FRI-BDI comparisons. Simply differencing f_j from β_i or using a method such as sum of absolute differences (SAD) would average out the differences across the entire 30×30 region, without indicating which regions of f_j and β_i are similar. Because the Hamming distance is used, the number of the 100 unique subregions of f_j that are similar to β_i can be calculated, rather than a simple average of similarities and disparities.

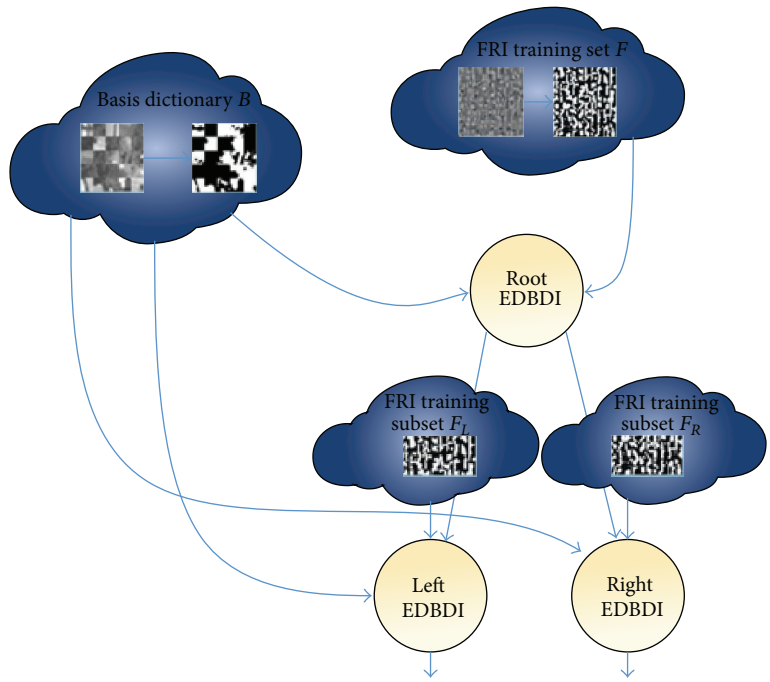


Figure 3: The tree creation process. The most effectively descriptive basis dictionary image (EDBDI) is chosen from the set B as the β that most evenly divides the set of training FRIs, F. The tree is split using this EDBDI, and the subsets of F are passed to the left and right children.

Using (2), the EDBDI for a node of the tree,

$$\beta_{\text{ED}} = \arg\max_{\beta \in B} E_{\beta}(F) \tag{4}$$

is the $\beta \in B$ which most evenly divides F. β_{ED} is chosen as the EDBDI for the node, and the remaining elements of B are used with F_L to create the left child and with F_R to create the right child. Each node stores the β_{ED} it used for splitting (the quantized version of β_{ED}), and a node number unique to the entire tree.

The process goes on until remaining subsets of F can no longer be split. Since there is no guarantee that any β_i in the set B will evenly divide the set F, the partitions at each node are not guaranteed to be perfectly even. Because of this, the tree is not guaranteed to be balanced. To split the node as evenly as possible, the best β_i from the entire set of B is chosen. The completed tree is saved to disk so that it can be reloaded for on-line processing.

Because an efficient tree structure is used, the training set F can be very large while still maintaining fast real-time comparison speeds. A binary tree structure contains 2^n unique paths, where n is the number of leaf nodes in the tree. With 64 EDBDIs the entire set of 50,000 FRIs can be partitioned with a tree depth of 17 levels [24]. A binary tree with 17 levels contains 2^{17}-1 nodes, meaning each of the 64 EDBDIs could be used, on average, over 2,000 times in the tree. This demonstrates how effectively these BDIs partition feature region images. Even though the tree contains 2^{17}-1 nodes, the on-line portion of the TreeBASIS algorithm only needs to hold the data for 64 EDBDIs. The individual tree nodes simply contain a reference to which EDBDI is used, along with pointers to the left and right children.

For any given application, the corresponding B and F are passed to the algorithm which creates a tree that is saved to disk. Each node in the tree contains only an index indicating which of the 64 BDIs is used for partitioning and pointers to its left and right children. When the tree is reloaded from disk, the quantized BDI vector associated with each index can be loaded from memory when it is needed for comparison.

2.2. Calculating Descriptors. Once the tree structure is created, descriptors can be quickly computed in real-time. Figure 4 summarizes the on-line portion of the TreeBASIS descriptor algorithm. (More details can be found in [24].) The algorithm takes a list of features detected using the FAST feature detector [25] and returns a descriptor for each feature. The FAST detector does not offer a dominant orientation or scale measurement. When considering frame-to-frame feature matching for UAV or UGV applications, scale changes and rotation between frames are small and can be ignored. This allows us to use the FAST detector and avoid trade-offs associated with providing increased invariance [26].

For each detected point, an FRI is obtained. Each FRI is a binary threshold over the same number of regions, and the resulting vector is passed into the tree. A Hamming distance h is again computed between the binary FRI vector and the binary EDBDI vector that is saved at the given node.

We then check the similarity between the FRI and the EDBDI at the node. If they are very similar then they are passed to the right child and if they are dissimilar then they are passed to the left child. This is necessary, since the fact that an FRI differs from a given BDI is just as informative as knowing that the FRI is very similar to a BDI. Rather

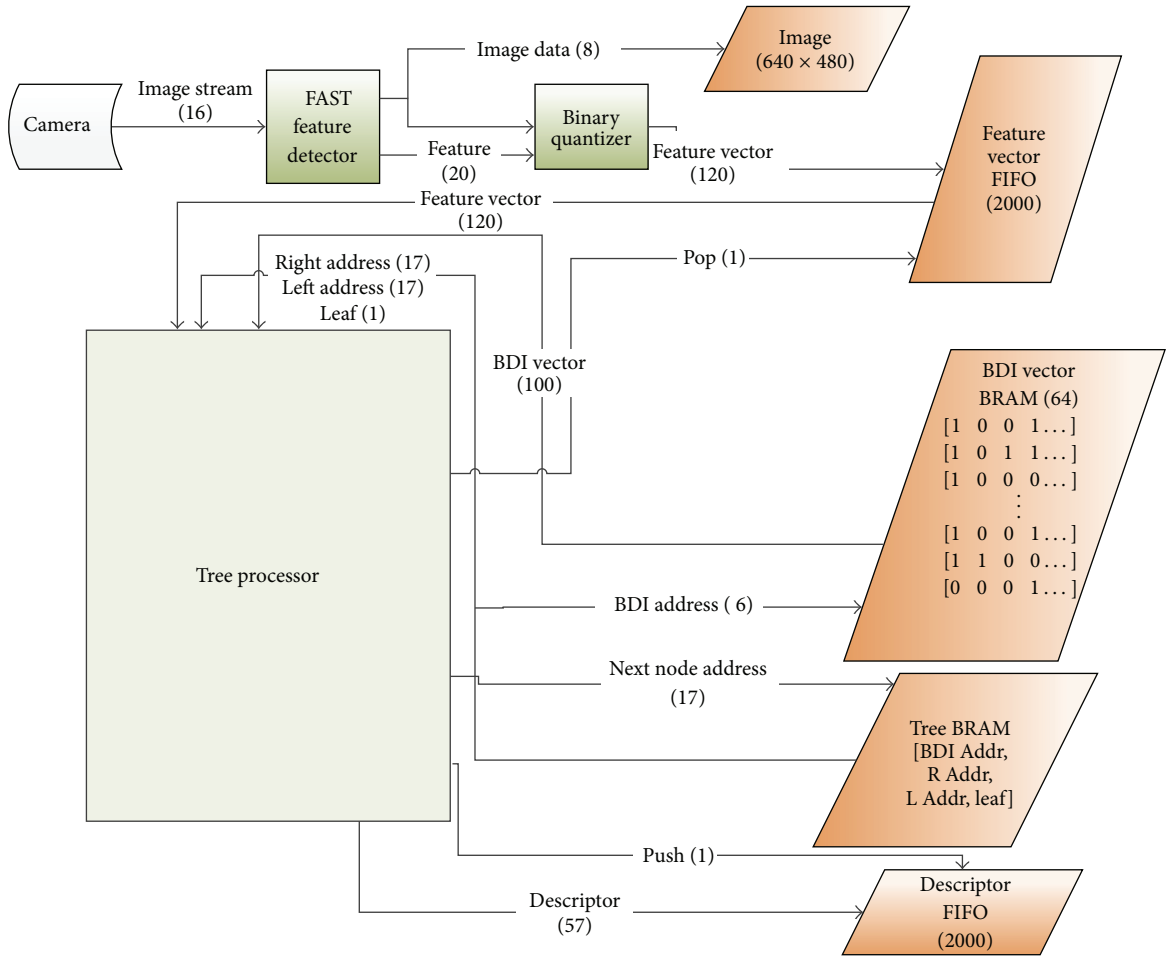


Figure 4: The overall hardware design of the TreeBASIS descriptor algorithm. Processing components are shown as green squares and memory components are shown as orange rhomboids. Bit widths of signal paths are denoted next to or below the signal path name in parentheses.

than keeping a unique node number, a simple binary value indicating a left branch (0) or right branch (1) at each node is sufficient to retain the entire path's information. The resulting descriptor length then is equal to the depth of the tree. For any given node in the path, each level requires only one bit (0 or 1) to record if the FRI went down the left branch or right branch. A 17-level tree will produce descriptor-paths that are at most 17 bits long compared to the 2,304 bits used in a BASIS descriptor and the 8,192 bits used to represent the double-precision floating point values of a SIFT descriptor.

2.3. Comparing Descriptors. In order to reduce the number of comparisons, tree structures are commonly used. The advantage of the TreeBASIS algorithm is that the tree is already used to compute the descriptor, so an additional tree for matching is not required. As explained earlier, each node in the BASIS tree is represented by an EDBDI, which represents a feature characteristic. Each node therefore represents a comparison and the result of an FRI characteristic. As a result of this, if two features' descriptors follow the same path, they must contain the same feature characteristics represented by the EDBDIs. Conversely, if they do not follow the same path, then they differ on a given feature characteristic and do not match. It is very important to determine at what depth of the tree the paths diverge. During the tree creation, the training set of FRIs is split in half at the root node. Features that diverge at the root differ more markedly than features diverging at a node much deeper in the tree.

The partitions at each node are not guaranteed to be perfectly even, so all paths in the tree may not progress to the deepest possible level. Because of this fact, descriptors may have variable lengths. The distance between two features is computed from the paths that are traversed. Initially, the distance is set to a large value. If the path elements are equal, the distance is reduced by

$$d = d\left(1 - \frac{p}{m}\right), \tag{5}$$

where p is the length of the path traversed so far and m is the total path length of the shorter descriptor path. For the same descriptor, the distance computation returns zero value. If the descriptors differ at any element along the path, the distance computation halts and the current value of d is returned. In

Table 1: The first 15 words in the tree ROM.

		Data		
Memory location	EDBDI ROM address	Left address	Right address	Leaf
0	101110	00000000000000001	00100001000011110	0
1	110100	00000000000000010	00010000101110011	0
2	101010	00000000000000011	00001000101001110	0
3	001101	00000000000000100	00000100010111111	0
4	011000	00000000000000101	00000010001111000	0
5	101001	00000000000000110	00000001000010001	0
6	101000	00000000000000111	00000000100001010	0
7	111111	00000000000001000	00000000010010001	0
8	100100	00000000000001001	00000000001010010	0
9	100001	00000000000001010	00000000000101101	0
10	110010	00000000000001011	00000000000011010	0
11	100110	00000000000001100	00000000000010011	0
12	101101	00000000000001101	00000000000010000	0
13	000000	00000000000001110	00000000000001111	0
14	110001	00000000000000000	00000000000000000	1

this way, the distance reflects how many of the paths of two descriptors are similar. Before the on-line portion begins, the maximum depth of the tree is known, which helps to allocate enough memory to store a maximum-depth path.

3. TreeBASIS Hardware Implementation

This section describes the development of actual hardware components that execute computation and matching algorithms using the TreeBASIS descriptor. No changes need to be made to the algorithm to implement it in hardware. The matching results of the bit-level accurate software version are identical to those of the hardware version. The TreeBASIS algorithm was implemented in VHDL and simulated, including the complete process of calculating feature descriptors and matching two descriptors. We discuss these simulation results and the size and clock rate of the resulting hardware realized on an FPGA.

3.1. TreeBASIS Descriptor. TreeBASIS, as described in Section 2, computes feature descriptors as the path an FRI takes as it is pushed down a precomputed vocabulary tree. The vocabulary tree is computed off-line on a standard desktop computer and then saved to disk. During on-line processing, the system loads a copy of the vocabulary tree and uses the data in the tree to determine which EDBDIs need to be compared to the FRI in question, and then the branching choices are recorded as a descriptor.

The off-line portion of the TreeBASIS software algorithm is used to provide a tree for the on-line portion of the TreeBASIS hardware implementation. The on-line portion of the TreeBASIS algorithm performs Hamming distance calculations which can be implemented in hardware using XNOR gates and an adder. The binary quantization of the FRIs and BDIs is also fully implementable in hardware without mathematical simplifications. Matching two descriptors requires a basic comparison of two 100-bit vectors which can be implemented with basic hardware comparators and adders. Therefore, no mathematical or other simplifications are necessary to implement the TreeBASIS descriptor in hardware. Because no simplifications are required, the accuracy of the hardware implementation is identical to the bit-level-accurate software version.

3.2. Descriptor Hardware Implementation. Figure 4 shows the top-level hardware layout of the TreeBASIS descriptor computation system. It consists of three major processing cores: a feature detector, a binary quantizer, and a tree processor. Additionally, it contains four large memory structures: image BRAM, feature vector FIFO, BDI vector ROM, and tree ROM. The image BRAM ($640 \times 480 \times 8$ bits) contains a copy of the image as it is passed out of the feature detector. The feature vector FIFO (2000 120-bit words) holds up to 2,000 feature vectors. This includes the 20 bits of (x, y) location information and the 100-bit binary quantized version of the 30×30 pixel FRI. The BDI vector ROM (BRAM, with 64 100-bit words) contains the binary quantized versions of all 64 EDBDIs that are used in the tree.

The tree ROM (BRAM, with up to 2^n 41-bit words) holds the entire BASIS tree. The tree ROM has a word size of 41 bits, consisting of 4 elements per word. The first 6 bits are the address in the BDI ROM of the quantized EDBDI vector for the current node. The next 34 bits are the addresses in memory of the left and right children, and the least significant bit indicates whether or not this node is a leaf. Table 1 shows a few lines from a tree ROM built from a BASIS tree with 17 levels. The three major processing cores are discussed below.

3.2.1. Feature Detector. Feature detectors have previously been developed in VHDL [9, 27–31] and their implementations are discussed later in this paper. As such, an assumption is made that the feature detector used will output the current

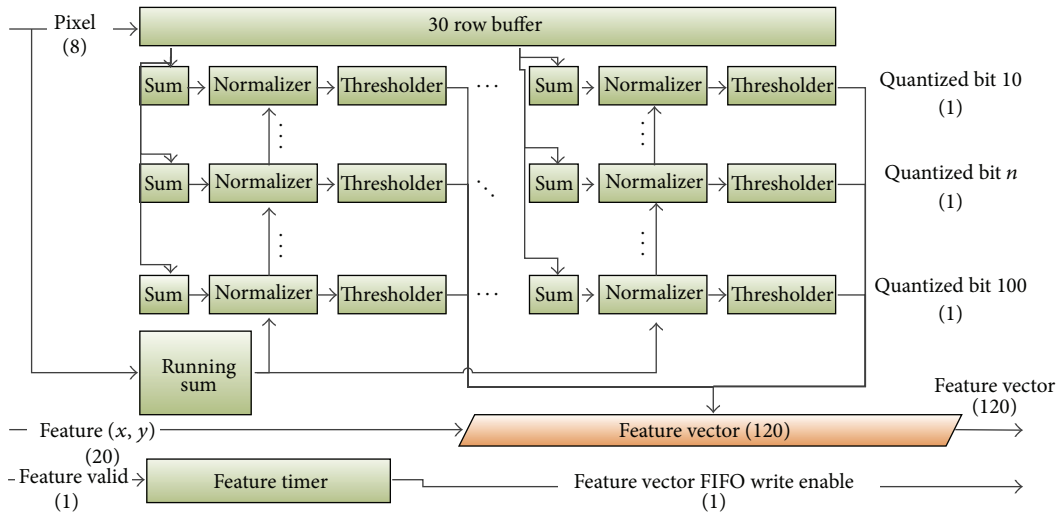


Figure 5: The binary quantizer processing core for the TreeBASIS hardware descriptor contains 30 row buffers which allow it to compute quantized FRI vectors while the image is streaming into the system.

image pixel as the image is streaming into the system, along with the (x, y) coordinates of the next valid feature and a valid feature signal.

3.2.2. Binary Quantizer. Figure 5 shows a graphical illustration of the binary quantizer core. The binary quantizer takes in a 30×30 pixel region centered around a feature point and performs two major operations. First, it sums and computes the average gray value of the entire 30×30 pixel area. Second, it binary thresholds each pixel in the region based on that pixel's comparison to the average.

In a software implementation, this quantization is done after the entire image is saved into memory and the 900 pixels are loaded from memory and operated on. In contrast, a hardware implementation can take advantage of the real-time streaming data flow to process these quantized values while the image is being captured.

The binary quantizer takes as input signals the x and y coordinates of a feature and a valid feature signal. While the image is streaming into the system pixel-by-pixel and row-by-row, the feature detector is processing features and passing each pixel through to the binary quantizer. The feature detector also maintains an (x, y) counter denoting the position of the current pixel in the image. When the feature detector determines that it has found a feature, it sets the valid feature output high and puts the (x, y) coordinates of the feature onto the feature bus.

The binary quantizer contains 30 row buffers. A row buffer is a simple barrel shifter of depth

$$d = I_W - F_W, \tag{6}$$

where I_W is the width of the image and F_W is the width of the FRI. Along with the barrel shifter, the row buffer contains F_W registers which are connected in-line with the shifter and are also accessible as outputs from the binary quantizer core. As each pixel comes into the quantizer, it is shifted into the first register of the first row. The pixel coming out of the last register of the first row is fed into the input of the barrel shifter. The output of the barrel shifter is fed into the first register of the second row. This process repeats for all 30 rows so that at any point in time a 30×30 pixel window is accessible via the registers. At the same time, the binary quantizer is computing a running sum of the values in the registers and holding the current running average of the 30×30 window. The 900 registers are grouped into 100 groups of 3×3 pixels (resulting in 100 quantized values, one for each 3×3 pixel subregion in the FRI). An adder attached to each group holds a running sum of the current 3×3 region. The output of the adder is subtracted from the running average of the entire FRI (Figure 6), and the result is fed into a thresholder which returns a single bit for each 3×3 subregion. A 120-bit register is used to store the 100 quantized bits plus the 20 bits containing the (x, y) position of the feature.

The valid feature signal from the feature detector is buffered so that it will go high when the desired feature point is located in the center of the 30×30 register block. When the valid signal goes high, it latches the quantized vector (along with the pixel coordinates) and then sends both to the feature vector output line along with a write-enable signal which inserts the quantized FRI into a feature vector FIFO. The implementation focuses on 640×480 resolution UAV images that could return several hundred features. A feature vector FIFO is built to hold up to 2,000 feature vectors per image. The parameters of the feature detector can be adjusted so that more or fewer features are returned if necessary.

As soon as there are any features in the feature vector FIFO, computation of descriptors begins. In this way, descriptors are being computed on features even before the entire image has been captured into memory. This streaming data flow allows even more features than would be possible if features are processed only during the space between frames. The output of the feature vector FIFO is connected to the tree processor component which computes the feature's descriptor.

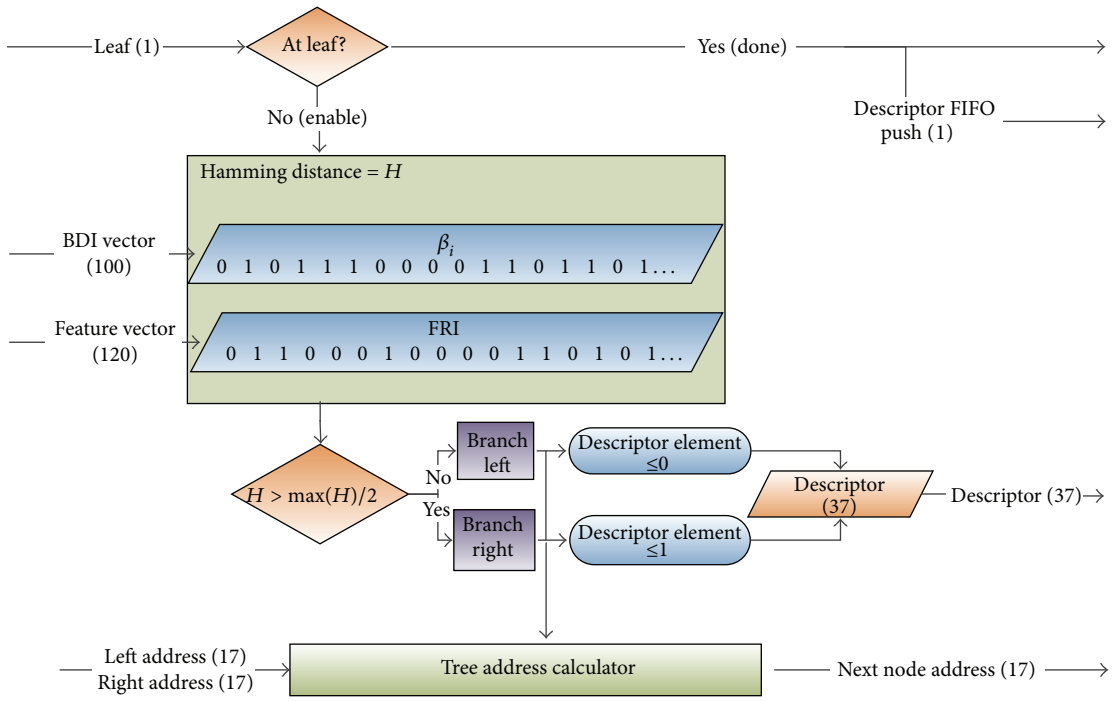


Figure 6: The tree processor core of the TreeBASIS hardware design navigates the tree ROM, pulls quantized EDBDIs from the BDI ROM, and makes decisions on which branch to follow based on the resulting Hamming distance between a node's EDBDI and the FRI.

Table 2: Six lines from a BDI ROM created from a dictionary of 64 EDBDIs.

Memory location	Data (quantized BDI)
0	00000111110000000000100000000010000111110000001111000000000001100000010011100111001111111100111111111
1	11100111111111111111011111111100111111110000000111000000001101000000011111000000111111000011111110000
2	00000011110000000011000000000100000000000000011000000011100001111111000111111111111111111111111111111
3	11111100001111111100011111111100000001110000000000000000000000110000000001111100000011111000001111110
4	11111111111111111111111111111111111111111111111111111111111111110000110000000000000000000000000000000
5	10000000001111000000111111100011111111101111111100111111000011110001101110001110111001111111100100 11

3.2.3. Tree Processor. Figure 6 shows a graphical illustration of the tree processor component. The tree processor component takes a 120-bit feature vector from the feature vector FIFO and computes a 37-element descriptor, with 20 bits for the (x, y) coordinates, and 17 bits for the path-decisions that comprise the descriptor. In order to compute the descriptor, the processor must be able to traverse the precomputed BASIS tree. For this purpose, software code was developed to take a precomputed BASIS tree and build an appropriate FPGA ROM that can be loaded during VHDL synthesis.

The code reads in the entire tree from disk and then recourses through the tree and stores each node as it passes into a one-dimensional array. Once the entire tree has been traversed, the code then iterates over the array and changes the tree-style pointers to the left and the right children into array indices where the left and right children have been stored. This results in a flat memory file that can be processed in the same fashion as a standard Turing machine. This array is then written to a ROM file which is read during VHDL synthesis.

Each node in the tree is associated with the EDBDI that was used to split the training data. Another program was written that takes in the basis dictionary used to create the tree and writes a VHDL ROM file that contains the 100-bit quantized vectors of each BDI. Table 2 shows a few lines from a BDI ROM containing 64 EDBDIs. The tree processor begins by loading location 0 from the tree ROM (Table 1) which contains the root node of the tree.

The first piece of data from the ROM, the EDBDI vector address, is wired directly to the BDI Vector ROM (Table 1) which contains the quantized vectors of all EDBDIs used to build the tree. The 100-bit quantized vector output of the EDBDI ROM is passed into the tree processor along with the remaining bits from the tree ROM which contain the left and the right child memory addresses and a leaf bit. The FRI vector (from the feature vector FIFO) and the EDBDI vector (from the EDBDI ROM) are passed into a Hamming distance calculator (XNOR gate and an adder). The resulting Hamming distance H is compared to $\widehat{H}$ ($\widehat{H} = |\beta|/2$). If $H > \widehat{H}$, a value of 1 is pushed into the 37-bit descriptor register (which

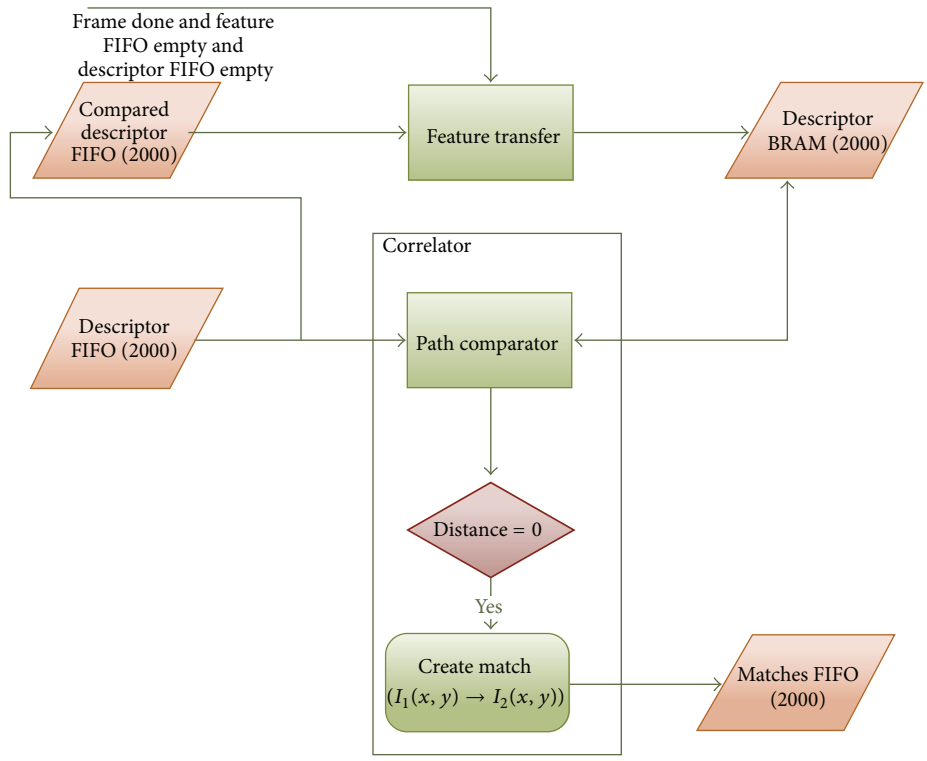


Figure 7: The matching subsystem of the TreeBASIS hardware implementation requires three additional memory structures and two processing cores.

already holds the 20 bits of (x, y) information that came with the FRI vector); otherwise a value of 0 is recorded. The 0 or 1 value is also passed to the tree address calculator, which is a simple MUX that chooses the left or right node address based on the value of the descriptor element. The resulting next node address is sent to the tree ROM, which starts this process over again at a new memory location in the tree ROM. When a leaf is reached (the leaf bit from the tree ROM is a 1), the finished descriptor is pushed into the descriptor FIFO, the next node address is set to 0, and a pop command is sent to the feature vector FIFO in order to obtain the next feature. This process is repeated until the feature vector FIFO is empty and the feature detector has finished.

3.2.4. Descriptor Matching. The final step in the system is to compute matches between two subsequent images in a video sequence. Figure 7 illustrates the correlation core of the TreeBASIS hardware implementation. Recall that the Tree Processor core pushes descriptors into a descriptor FIFO. The correlation core contains three memory elements in addition to those mentioned previously.

A compared descriptor FIFO holds up to 2,000 37-bit descriptors. When a descriptor is popped out of the descriptor FIFO it is immediately pushed into the compared descriptor FIFO and passed to the path comparator core. A descriptor BRAM also holds up to 2,000 37-bit descriptors from the previously processed image. When the feature detector reports that it has finished, the feature FIFO is empty, and the descriptor FIFO is empty, the feature transfer core moves all elements in the compared descriptor FIFO into the descriptor BRAM in preparation for the next image to be processed. The matches FIFO holds are up to 2,000 40-bit matches. When the path comparator finds a match, the 20-bit (x, y) coordinates from image 1 (I_1) and the 20-bit (x, y) coordinates from image 2 (I_2) are concatenated and stored in this FIFO. They can be accessed by other hardware components or saved into RAM for software use. In addition to these memory structures, the correlation core contains two processing elements: the path comparator and the feature transfer core.

3.2.5. Path Comparator. The path comparator takes in a descriptor from the descriptor FIFO and from the descriptor BRAM. It compares the two 17-bit descriptors from the 37-bit vectors in order to determine a match. In the current hardware implementation, as well as in the software implementation, only identical descriptor paths are considered to be a correct match. Therefore, the path comparator computes the equivalency of the two 17-bit vectors. If the two vectors are equal, the 20-bit (x, y) coordinates from both descriptor vectors are concatenated together and the 40-bit match is pushed onto the matches FIFO. If the vectors are not equal, the path comparator increments the descriptor BRAM address and continues until it either reaches the last descriptor in the BRAM or a match is found. Once this condition is reached, the path comparator requests a new descriptor from the descriptor FIFO (which causes it to be recorded in the comparable descriptor FIFO) and resets the descriptor BRAM address to 0. The process continues until the frame is finished,

Table 3: Accuracy results and memory footprints for SIFT, SURF, BASIS, and TreeBASIS on the Idaho dataset. Memory usage assumes that 1,000 features per image are kept for each algorithm.

Algorithm	Average memory usage per image	Homography accuracy
SIFT	1,024 Kilobytes	34.7%
SURF	512 Kilobytes	73.5%
BASIS	288 Kilobytes	75.5%
TreeBASIS	2.1 Kilobytes	79.6%

the feature FIFO is empty, and the descriptor FIFO is empty. When this condition happens, an overall "completed" signal is asserted at which point the FIFO for the match can be accessed or saved in memory for software use.

3.3. Accuracy. The software system developed in Section 2 provides results with the same accuracy as those of the system implemented in hardware. The results of the bit-level-accurate TreeBASIS system are provided here for clarity of discussion. To calculate the homography accuracy the same procedure is carried out on the dataset as used in [31, 32]. Table 3 shows the results of the original BASIS algorithm alongside the results of the TreeBASIS algorithm. In the original BASIS descriptor, each descriptor is 128 ternary digits long, requiring 2,304 bits total per descriptor. The BASIS software algorithm (using 2,304-bit descriptors) achieved an accuracy of 75.5% on the *Idaho* test set [20]. The TreeBASIS algorithm, using 17-bit descriptors and a tree built using 64 BDIs and a training set of 50,000 FRIs obtained an accuracy of 79.6% on the *Idaho* test set. SIFT and SURF algorithms were implemented for comparison according to [29, 30].

3.4. Synthesis Results. The TreeBASIS Hardware descriptor was written in VHDL, synthesized, and implemented on a Xilinx Virtex-6 VLX760 FPGA. Table 4 shows the result of the system synthesis on this platform. These results are for the complete binary quantizer, tree processor, memory components, and correlator systems. The feature detector is not included in this summary. The TreeBASIS hardware system requires less than 1% of the slice registers on the FPGA, 5% of the slice LUTs, and 30% of available BRAMs.

Because the TreeBASIS descriptor system is so small and efficient, there is sufficient FPGA fabric remaining to implement additional computer vision components or to select a smaller FPGA if power and weight constraints dictate. The TreeBASIS system uses less than half of the available resources on the FPGA, making it an ideal fit for low resource applications.

All timing constraints of the TreeBASIS VHDL design were met without any reduction in clock rate. As such, the entire system will run at the VLX760's top clock rate of 400 MHz. Due to the use of a cascaded adder system and the streaming nature of the pixel data, it requires only one additional clock cycle to compute the binary quantized version of the FRI once a feature has been identified. It requires just 46 clock cycles to compute one descriptor. The image sensor used for this research outputs 640 × 480 pixels on a 25 MHz clock at 30 frames per second. At a clock rate of 400 MHz, the TreeBASIS descriptor FPGA system can calculate over 200,000 descriptors in the 33 milliseconds it takes to obtain one image (at 30 frames per second) using the 25 MHz image sensor clock speed or over 100,000 descriptors per frame if operating at 60 frames per second.

Table 4: The TreeBASIS descriptor system, including the correlator, takes a very small percentage of FPGA's available fabric.

Device utilization summary		
Selected device: 6vlx7601760-2		
Slice logic utilization		
Number of slice registers	8787 out of 948480	0%
Number of slice LUTs	26300 out of 474240	5%
Number used as logic	26300 out of 474240	5%
IO utilization		
Number of IOs	78	
Number of bonded IOBs	78 out of 1200	6%
Specific feature Utilization		
Number of block RAM/FIFO	216 out of 720	30%
Number using block RAM only	216	
Number of BUFG/BUFGCTRLs	1 out of 32	3%

4. Conclusion

The TreeBASIS descriptor has been shown to be an effective feature descriptor for the task of UAV aerial frame-to-frame feature matching. It is ideally suited for low-resource implementations such as FPGAs. In this paper, we have presented the development and implementation of the hardware version of the TreeBASIS descriptor. There is no loss of accuracy from the software version to the hardware version because no simplifications or modifications to the algorithm were necessary. TreeBASIS takes advantage of the benefits sparse coding dictionaries provide and it optimizes descriptor size and comparison speed by using a tree structure to reduce the number of comparisons required. The TreeBASIS descriptor requires less descriptor size and computation time, yet it provides better feature matching accuracy than BASIS, SIFT, and SURF descriptors. The TreeBASIS hardware system, when implemented on a Virtex-6 VLX760 FPGA, utilizes less than 6% of the available slice logic and only 30% of the available BRAM. This allows the majority of the FPGA to be used for additional vision processing or other tasks that a limited resource system may require. The TreeBASIS descriptor provides a much needed solution to the task of high-level computer vision processing for low-resource systems. The TreeBASIS algorithm is target independent, so any platform can be used for the implementation without any modification. The entire TreeBASIS feature descriptor algorithm has been implemented in hardware, demonstrating the fact that it is a fast and efficient matching solution ideally suited for small, lightweight, embedded platforms.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] 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 *Intelligent Robots and Computer Vision XXIV: Algorithms, Techniques, and Active Vision, 63840P*, vol. 6382 of *Proceedings of SPIE*, Boston, Mass, USA, October 2006.

[2] B. Tippetts, S. Fowers, K. Lillywhite, D.-J. Lee, and J. Archibald, "FPGA implementation of a feature detection and tracking algorithm for real-time applications," in *Proceedings of the 3rd International Conference on Advances in Visual Computing (ISVC '07)*, Lake Tahoe, Nev, USA, November 2007.

[3] 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*, vol. 3, no. 4, pp. 394–406, 2006.

[4] Z. Wei, D. Lee, and B. E. Nelson, *A Hardware-Friendly Adaptive Tensor Based Optical Flow Algorithm*, vol. 4842 of *Lecture Notes in Computer Science*, 2007.

[5] B. J. Tippetts, D.-J. Lee, J. K. Archibald, and K. D. Lillywhite, "Dense disparity real-time stereo vision algorithm for resource-limited systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 10, pp. 1547–1555, 2011.

[6] 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*, vol. 21, no. 10, pp. 1363–1377, 2011.

[7] Y.-G. Kim, V. R. Jayanthi, and I.-S. 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*, vol. 21, no. 10, pp. 1401–1414, 2011.

[8] K. Lillywhite, D.-J. Lee, B. Tippetts et al., "An embedded vision system for an unmanned four-rotor helicopter," in *Intelligent Robots and Computer Vision XXIV: Algorithms, Techniques, and Active Vision*, vol. 6384 of *Proceedings of the SPIE*, Boston, Mass, USA, October 2006.

[9] B. J. Tippetts, D.-J. Lee, S. G. Fowers, and J. K. Archibald, "Real-time vision sensor for an autonomous hovering micro unmanned aerial vehicle," *Journal of Aerospace Computing, Information and Communication*, vol. 6, no. 10, pp. 570–584, 2009.

[10] 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*, vol. 18, no. 12, pp. 1703–1712, 2008.

[11] W. S. Fife and J. K. Archibald, "Reconfigurable on-board vision processing for small autonomous vehicles," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 080141, 2007.

[12] J. Šváb, T. Krajník, J. Faigl, and L. Přeučil, "FPGA based speeded up robust features," in *Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications (TePRA '09)*, pp. 35–41, November 2009.

[13] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: binary robust independent elementary features," in *Proceedings of the 11th European Conference on Computer Vision: Part IV*, Berlin, Germany, 2010.

[14] M. Calonder, V. Lepetit, M. Özuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: computing a local binary descriptor very fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, 2012.

[15] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: binary robust invariant scalable keypoints," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV '11)*, pp. 2548–2555, Barcelona, Spain, November 2011.

[16] M. Ebrahimi and W. W. Mayol-Cuevas, "SUSurE: speeded up surround extrema feature detector and descriptor for realtime applications," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '09)*, pp. 9–14, June 2009.

[17] G. Hua, M. Brown, and S. Winder, "Discriminant embedding for local image descriptors," in *Proceedings of the IEEE 11th International Conference on Computer Vision (ICCV '07)*, pp. 1–8, October 2007.

[18] Y. Ke and R. Sukthankar, "PCA-SIFT: a more distinctive representation for local image descriptors," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '04)*, vol. 2, pp. II-506–II-513, July 2004.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV '11)*, pp. 2564–2571, Barcelona, Spain, November 2011.

[20] S. G. Fowers, D.-J. Lee, D. A. Ventura, and J. K. Archibald, "The nature-inspired BASIS feature descriptor for UAV imagery and its hardware implementation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 5, pp. 756–768, 2013.

[21] C. Liu and H. Wechsler, "Face recognition using evolutionary pursuit," in *Computer Vision—ECCV '98*, H. Burkhardt and B. Neumann, Eds., vol. 1407 of *Lecture Notes in Computer Science*, pp. 596–612, Springer, Berlin, Germany, 1998.

[22] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3736–3745, 2006.

[23] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[24] S. G. Fowers, *Limited resource feature detection, description, and matching [Ph.D. thesis]*, Department of Electrical and Computer Engineering, Brigham Young University, Provo, Utah, USA, 2012.

[25] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Proceedings of the 10th IEEE International Conference on Computer Vision (ICCV '05)*, pp. 1508–1515, Beijing, China, October 2005.

[26] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: a survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007.

[27] J. Fischer, A. Ruppel, F. Weisshardt, and A. Verl, "A rotation invariant feature descriptor O-DAISY and its FPGA implementation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '11)*, pp. 2365–2370, September 2011.

[28] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *Proceedings of the International Conference on Field-Programmable Technology (FPT '09)*, pp. 30–37, December 2009.

[29] L. Chang and J. Hernández-Palancar, "A hardware architecture for SIFT candidate keypoints detection," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Lecture Notes in Computer Science, pp. 95–102, Springer, Berlin, Germany, 2009.

[30] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *Proceedings of the 18th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '10)*, pp. 3–10, May 2010.

[31] S. G. Fowers, D. J. Lee, D. A. Ventura, and B. J. Tippetts, "Novel feature descriptor for low-resource embedded vision sensors for micro unmanned-aerial-vehicle applications," *Journal of Aerospace Information Systems*, vol. 10, no. 8, pp. 385–395, 2013.

[32] G. Hartman, *Real-time color TreeBASIS feature matching on a limited-resource hardware system [M.S. thesis]*, Department of Electrical and Computer Engineering, BYU, Provo, Utah, USA, 2013.