A Novel, Efficient, Tree-Based Descriptor and Matching Algorithm

Spencer G Fowers, D.J. Lee, Dan Ventura, Doran K. Wilde

March 14, 2012

Abstract

This paper presents the development of a new feature descriptor derived from the BASIS descriptor that provides improvements in descriptor size, computation speed, matching speed, and accuracy. The TreeBASIS descriptor algorithm utilizes a binary vocabulary tree that is computed off-line using basis dictionary images (BDIs) derived from sparse coding and a test set of feature region images (FRIs), and the resulting tree is stored in memory for on-line searching. During the on-line algorithm, a feature region image (FRI) is binary quantized and the resulting quantized vector is passed into the BASIS tree, where a Hamming distance is computed between the FRI and the effectively descriptive BDI (EDBDI) at the current node to determine the branch taken. The path the FRI takes is saved as the descriptor, and matching is performed by following the paths of two features and iteratively reducing the distance as the path is traversed. Experimental results show that the TreeBASIS descriptor outperforms BASIS, SIFT, and SURF on frame-to-frame aerial feature point matching.

1 Introduction

Computer vision applications for low power, low resource, and embedded systems are becoming increasingly prevalent. Some examples of computer vision applications that would be useful if implemented on low-power small microprocessors or FPGAs are target tracking [1], object identification [2], optical flow [3], stereo vision [4], super resolution [5], image stabilization [6], and image rectification, localization, and pose estimation [1, 7, 8, 9, 10]. The initial step in most of these applications is some level of high quality feature detection, descrption, and matching. Computing feature descriptors is typically very computationally complex, requiring square root, division, multiplication, and exponential operations. The computation of these descriptors is challenging in low-power applications [11]. For example, it is difficult to implement a full floating-point unit in a small FPGA's hardware logic, so some FPGA implementations of feature descriptors off-load the actual descriptor computation to a CPU for complex mathematical operations [9, 12, 13] or perform other algorithmic simplifications (such as conversion to fixed point) in order to accommodate a hardware implementation [14][3].

Feature descriptors take feature points obtained from a detector [15, 16, 17, 18], and compute a unique description of each point [15, 19, 20, 21]. A good feature descriptor will uniquely describe a feature point allowing it to be correctly identified and matched in subsequent images. These descriptors typically require a large amount of storage space. For example, each 128-element SIFT descriptor requires 1024 bytes per descriptor [15], and each 64-element SURF descriptor requires 512 bytes [19]. These algorithms can easily return more than 1,000 features from a 640×480 pixel resolution image, making storage space an issue on a low-resource platform.

As the use of low resource platforms such as smartphones and FPGAs becomes more pervasive for vision applications, the need for low resource feature detection and description algorithms will continue to increase. By reducing the computational complexity of a feature descriptor algorithm and reducing the descriptor memory footprint, we can reduce the amount of computational power required and increase the speed at which low-resource systems can produce processed vision sensor information. With a feature descriptor specifically designed for low-resource systems, we can bring high quality computer vision algorithms into the realm of low-resource systems such as micro unmanned aerial and ground vehicles (UAVs and UGVs).

The BAsis Sparse-coding Inspired Similarity (BASIS) descriptor, presented in [22], was developed as a feature descriptor well-suited for low-resource applications. The premise behind the BASIS algorithm is that, if the basis function set returned from sparse coding [23] is kept constant, the contribution of each basis function can be used to uniquely describe a feature [24, 25]. These basis functions comprise a dictionary of "feature characteristics" that can be weighted and combined to uniquely reconstruct the region around any feature in an image. BASIS computes descriptors by comparing the region around a feature (FRI) to a set of basis dictionary images (BDIs) obtained using sparse coding.

The comparison of FRIs to BDIs provides a basis similarity measure (BSM) matrix, which is broken into regions, thresholded, and stored as a descriptor vector consisting of ternary-digits. Matching between features is achieved by computing a carry-free, element-wise subtraction of each ternary digit in a descriptor element and summing the resulting values to form a similarity indicator sum (SIS). The cumulative SIS is then used as a distance metric between two features. Features in one image are compared to features in another, and the resulting matches are sorted by distance. The top match for each feature in image 1 is retained, and after removing duplicates, the resulting list of matches is returned from the BASIS algorithm. Because BASIS descriptors consist of ternary-digits, standard distance metrics (Euclidean, Mahalanobis) are incompatible and make it difficult to compare matching accuracy between BASIS and other detector/descriptor algorithms such as SIFT and SURF. In order to compare descriptors, researchers in [22] computed a frame-to-frame homography between images from the *Idaho* aerial image dataset using each algorithm and compared the overall effectiveness. BASIS provides a 43% smaller descriptor memory footprint than that of commonly used detector/descriptor methods, provides good descriptor matching accuracy for frame-to-frame feature matching applications, and has been fully implemented in FPGA hardware.

In this paper we describe the development of a novel modification to the original BASIS descriptor. Our new algorithm, TreeBASIS, creates a vocabulary tree using a small basis dictionary to partition a training set of feature region images (FRIs). This vocabulary tree is computed off-line and stored in the algorithm for on-line descriptor computation and matching. Basis dictionary images (BDIs) are quantized and their intensity values are represented using a binary vector. TreeBASIS computes feature descriptors by quantizing the feature region image (FRI), passing it through the tree, and recording its path. Matching 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 feature from the second image.

TreeBASIS provides a much smaller descriptor than BASIS, SIFT, or SURF, requires less computation for creating descriptors, and includes a novel descriptor matching algorithm that reduces processing time for matching descriptors. It also provides improved feature point matching accuracy on the *Idaho* dataset presented in [26]. Section 2 describes the TreeBASIS algorithm. In Section 3, we provide a comparison between the BASIS and the TreeBASIS descriptors on the *Idaho* dataset. Finally, in Section 4 we discuss our conclusions and future work.

2 TreeBASIS

TreeBASIS features three major components: building the vocabulary tree, computing descriptors, and matching descriptors between two images. The tree can be created off-line, on standard desktop computing hardware. 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.

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 [27] to partition a training set of FRIs, F. The theory behind sparse coding states that if the K-SVD is trained on a very large dataset of images, the basis dictionary, B, returned by the K-SVD can be used to reconstruct a very large variety of natural images [24]. An example basis dictionary, obtained from running the entire *GoogleMaps* dataset [26] through K-SVD, is shown in Figure 1. According to sparse coding theory, a basis dictionary set, B, similar to the one shown in Figure 1 will be useful to describe any set of natural images, not just those from the *GoogleMaps* dataset from which it was created.

The dictionary set, B, is used to partition a training set, F. The training set consists of a large quantity (50,000+) of FRIs (30×30 pixels) taken from various images similar to those that are expected to be encountered during real-time processing. We chose a BDI and FRI size of 30×30 pixels in order to provide additional information about the pixels surrounding a feature point. A region larger than 30×30 would run the risk of containing too much background data or content not associated with the feature. Conversely, reducing the size may not retain enough information to keep FRIs unique. In the hardware implementation of BASIS [22], the size of the FRIs was reduced to 24×24 pixels which resulted in a minor loss of accuracy.

Choosing a training set of images similar to those expected to be encountered during on-line processing will improve accuracy. However, the generic nature of the basis dictionary images, B, and the small 30×30 pixel size of the FRIs provide some invariance to the differences between the training set images and the images the algorithm may see in real-time, so F may consist of more generic training images and the algorithm will still obtain a high degree of accuracy.

In order to reduce computation time during both off-line and on-line stages, the BDIs and FRIs are divided into regions, averaged, and binary thresholded (Figure 2). For our experiments, we set the number of regions for binary thresholding to 100. This caused the 30×30 pixel FRIs and BDIs to be divided into 100 regions of 3×3 pixels. In order to threshold the BDI or FRI, first the average gray value of the 30×30 pixel image is computed as

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

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

The original BASIS algorithm compares each FRI from an image to each BDI, which requires a lot of computation and memory to store the complete basis dictionary and complete set of FRIs. In addition to quantizing the BDIs and FRIs, TreeBASIS utilizes a binary tree structure which lets us avoid comparing each FRI with every BDI.

Figure 3 illustrates the tree creation process. Each node of the tree is created by taking a set F of training FRIs and determining the most *effectively descriptive* BDI (EDBDI), β_{ED}, from the dictionary B. By most *most effectively descriptive* we mean the BDI that most evenly partitions the training set

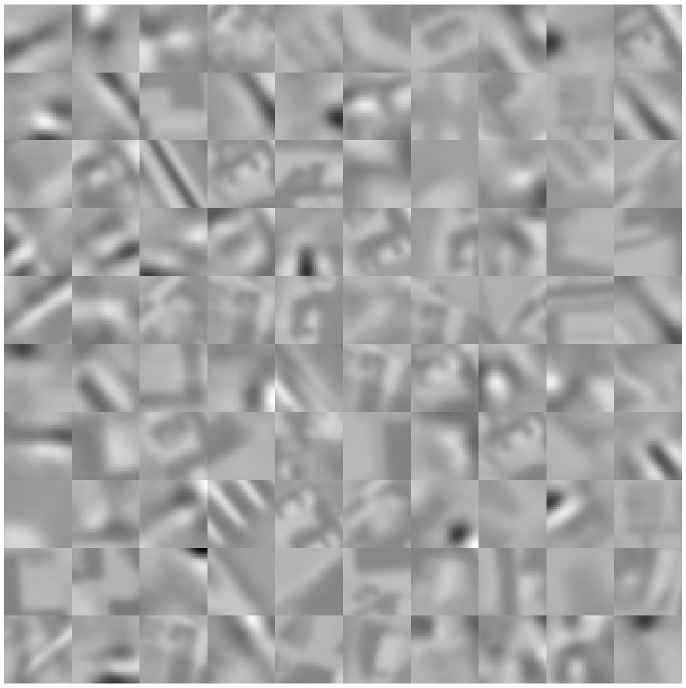

Figure 1: The basis dictionary, B, returned from KSVD for the entire *GoogleMaps* dataset.

F (see Algorithm 1). The EDBDIs are the BDIs that most effectively portray important feature characteristics, because half of the FRIs match them closely, while the other half do not. By building a tree in this way, we drastically reduce the number of FRI-BDI comparisons because we are, in essence, training the tree to only compare an FRI to the BDIs that will help it differentiate this FRI from all other FRIs it may see.

Given sets $B = \{\beta_0 \dots \beta_n\}$ and $F = \{f_0 \dots f_m\}$. For $\beta \in B$, we define the entropy of over the set F with respect to β as

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

where

$$p_L = \frac{|F_L|}{|F|}$$

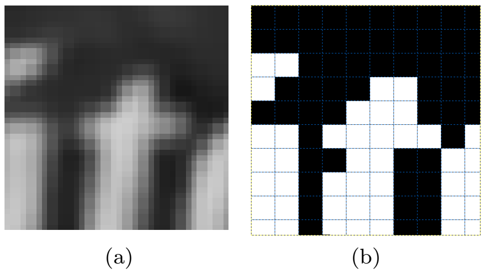

Figure 2: An example binary thresholding of an FRI. This reduces the memory footprint of the image from 900 8-bit pixels to a 100-bit vector, and comparison requires 100 1-bit compares, instead of 900 1-byte comparisons.

$$p_R = \frac{|F_R|}{|F|}$$

$$F_L = \left\{ f \,\middle|\, f \in F, h(f, \beta) \leq \frac{|\beta|}{2} \right\}$$

$$F_R = \left\{ f \,\middle|\, f \in F, h(f, \beta) > \frac{|\beta|}{2} \right\}$$

where $h(x, y)$ returns the Hamming distance between x and y. Conceptually, p_L is the proportion of F whose hamming distance is less than or equal to $\frac{|\beta|}{2}$, while p_R is proportion of F whose hamming distance is greater than $\frac{|\beta|}{2}$, where $|\beta|$ is the number of bits in β (the number of quantized regions). In our implementation, $|\beta| = 100$. The Hamming distance was chosen 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 pixel region, and not indicate which regions of f_j and β_i are similar. By using the Hamming distance, we are indicating how many of the 100 unique regions of f_j are similar to β_i, rather than a simple average of the similarities and disparities.

Using Equation 2, we find the EDBDI for a node of the tree,

$$\beta_{ED} = \underset{\beta \in B}{\operatorname{argmax}} \, E_\beta(F) \tag{3}$$

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

The process continues until the remaining subsets of F can no longer be split. The partitions at each node are not guaranteed to be perfectly even because there is no guarantee that there exists a β_i in the set B that perfectly divides

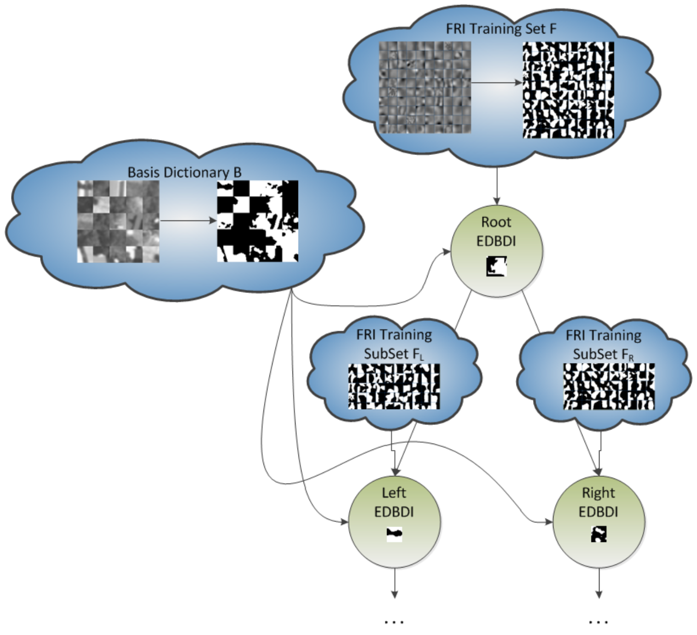

Figure 3: The tree creation process. The most effectively descriptive basis dictionary image (EDBDI) is chosed from the set B as the β that most evenly divides the set of training FRIs, F. The tree is split using this EDBDI, and the subsets of F are passed to the left and right children.

the set of F into equal halves. Because of this, the tree is not guaranteed to be balanced. However, because the entire set of B is evaluated for it's partitioning ability, the best β_i is chosen at each node to split the set as evenly as possible. Once the tree has been constructed, it is saved to disk so that it can be re-loaded for on-line processing.

Because we are using an efficient tree structure, our 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. Even with a training set F, of 300,000 elements, a perfectly balanced tree could divide the set into 300,000 leaf nodes at a depth of $\log_2(300,000) = 19$. In the on-line application, FRIs only need to be compared to the specific elements of B along a single path through the tree, which means an FRI in this case would only be compared against at most 19 elements of B. This gives the tree the ability to provide many unique descriptors while the structure maintains an efficient method for comparing FRIs to BDIs.

However, there is a tradeoff with very deep trees. The memory footprint of the BASIS tree grows exponentially with each level, and the number of levels is directly proportional to the number of training images used in its creation. While a larger tree may allow for discrimination of a larger number of features,

Algorithm 1 *TreeBASIS algorithm pseudocode.* B is the dictionary set of BDIs. F is the training set of FRIs. c is a unique node ID number. Typical parameter values: $|B| = 64$, $|F| = 50,000$.

```
TreeBASIS($B, F, c$)
    $ID \leftarrow c$
    $MaxEnt \leftarrow 0$
    $F_L \leftarrow \emptyset$
    $F_R \leftarrow \emptyset$
    for all $\beta \in B$ do
        $L \leftarrow \emptyset$
        $R \leftarrow \emptyset$
        for all $f \in F$ do
            if $h(f, \beta) \leq \frac{|\beta|}{2}$ then
                $L = L \cup f$
            else
                $R = R \cup f$
        $E_\beta = -\frac{|L|}{|F|} log_2 \frac{|L|}{|F|} - \frac{|R|}{|F|} log_2 \frac{|R|}{|F|}$
        if $E_\beta > MaxEnt$ then
            $MaxEnt = E_\beta$
            $\beta_{ND} = \beta$
            $F_L = L$
            $F_R = R$
    if $|F_L| = 0$ then
        $Leaf \leftarrow 1$
        return
    else
        $Leaf \leftarrow 0$
        TreeBASIS($B \setminus \beta_{ND}, F_L, 2c + 1$)
        TreeBASIS($B \setminus \beta_{ND}, F_R, 2c + 2$)
```

the size of the tree grows exponentially and becomes intractable for a limited-resource platform. Similarly, the number of BDIs used to split the tree affects memory requirements and accuracy. If more BDIs are used, there are more possibilities on how to split the tree, allowing for a greater distinction between features. However, more BDIs require more storage space for the binary quantized vectors for each BDI. Additionally, the BDIs are intended to represent unique feature characteristics, and using too many BDIs runs the risk of unnecessarily representing duplicate feature characteristics. To ameliorate this, the entropic approach to tree construction biases the algorithm to favor balanced, less deep trees over unbalanced, deeper trees. However, the size of the initial dictionary set B will still impact the tree depth.

In order to examine the effect of different BDI set sizes and different training set sizes on tree accuracy, an initial pool of 443 basis dictionary images was created by running the KSVD on seven distinct video sequences. From this initial pool, we selected a random group of 32, 64, and 128 BDIs, and then hand-selected an additional set of 32 BDIs that appeared to be the most distinct among the entire pool for our basis dictionary, B. For the training set F, we randomly selected sets of 3,000, 6,000, 50,000, 175,000 and 300,000 FRIs from the *GoogleMaps* dataset [26], which contains a total of 300,000 FRIs. The *GoogleMaps* dataset was created using the Google maps API (http://code.google.com/apis/maps/). The Google maps API allows a user to download a satellite or high-resolution aerial image of a location given the latitude, longitude, and zoom level (altitude). Images were retrieved from the maps API using specific latitude and longitude coordinates. These coordinates were then varied slightly and a new image was captured. To provide feature rich images, the initial latitude and longitude coordinates were chosen to be inside rural areas where buildings and landmarks are easily visible. The *GoogleMaps* dataset was created using the Google maps API (http://code.google.com/apis/maps/). The Google maps API allows a user to download a satellite or high-resolution aerial image of a location given the latitude, longitude, and zoom level (altitude). Images were retrieved from the maps API using specific latitude and longitude coordinates. These coordinates were then varied slightly and a new image was captured. To provide feature rich images, the initial latitude and longitude coordinates were chosen to be inside rural areas where buildings and landmarks are easily visible.

We built a tree using each of these BDI sets and training sets and found that the set of 64 randomly selected BDIs, when paired with the 50,000 element training set provided a tree that performed accurately on the *Idaho* dataset [26] (Figure 4) while using a manageable training set size. The resulting tree has a maximum depth of 17, implying that the maximum size of any path taken in this tree will be 17 elements long. The trees created using the hand-selected B did not perform any better than the randomly selected set of 32 B, demonstrating that, first, each BDI does contain important characteristics that can be useful in discriminating FRIs, and second, that a BDI we see as being a potential EDBDI may not truly be effective at partitioning the training set. All BDIs are all outputs from the KSVD algorithm, which implies that the optimization stage found each BDI to be useful in describing and reconstructing feature region images. As such, it is difficult for us to place added emphasis on any individual BDI by hand, and a random selection proved to be just as effective for creating training set partitions.

With 64 EDBDIs we were able to partition the entire set of 50,000 FRIs with a tree depth of 17 levels. 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 again demonstrates the effectiveness of these BDIs at partitioning features.

Even though the tree contains $2^{17} - 1$ nodes, the on-line portion of the Tree-BASIS algorithm only needs to hold data for 64 EDBDIs, and the individual tree nodes simply contain a reference to which EDBDI is used, along with pointers to the left and right children. Table 1 lists the depth of the tree created for each pair of B and F set sizes. If a tree was able to perfectly partition the set F in

	Min. Depth Required	32 random	64 random	128 random	32 handpicked
3000	11	12	11	11	12
6000	12	16	13	12	13
50000	15	19	17	17	18
175000	17	23	21	19	21
300000	18	22			

Table 1: Tree depths for the various B and F set sizes. Recall that each tree contains $2^d - 1$ nodes, where d is the depth of the tree. A larger tree depth implies that it required more splits to fully partition the set F.

half at each node, it would require ceiling($\log_2 \frac{|F|}{2}$ leaf nodes to represent the tree. The division by 2 is due to the fact that once there are 2 FRIs or less in a node, partitioning stops, so each node can contain 2 FRIs. Therefore, the depth of a tree is a measure of its efficiency in splitting the set F. For the tree depths in Table 1, each level more than ceiling($\log_2 \frac{|F|}{2}$) implies reduced efficiency in partitioning the tree due to the algorithm being unable to find an EDBDI in the set of B that could partition F evenly.

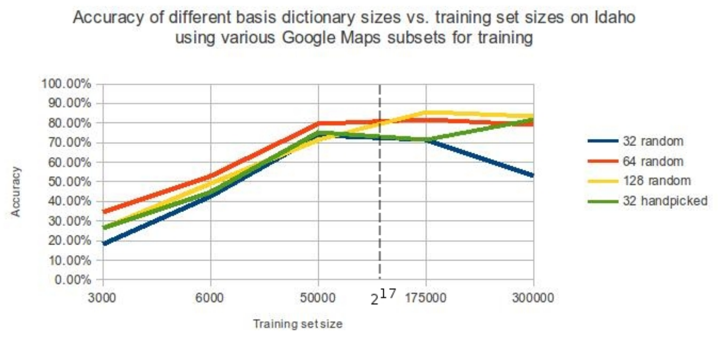

Figure 4: Comparison of differing basis dictionary sizes (B, denoted by individual lines) and training set sizes (F, measured along the x-axis) versus performance of the TreeBASIS descriptor. The tree used in our results was created using a basis dictionary B of size 64, and a training set F of 50,000 elements.

Our chosen B and F were passed into our algorithm which created a tree that was saved to disk. Each node in the tree contains only an index indicating which of the 64 BDIs was used for partitioning, and pointers to its left and right children. When the tree is re-loaded from disk, the quantized BDI vector associated with the index can be loaded from memory when needed for comparison.

2.2 Calculating Descriptors

Now that we have created an efficient tree structure, descriptors can be quickly computed in real-time. The BASIS descriptor is designed to be used in a frame-to-frame feature point matching application for a UAV equipped with a low power FPGA platform, and we have designed the TreeBASIS descriptor to perform the same function.

Only the on-line portion of the TreeBASIS algorithm needs to be implemented on the target platform. The on-line portion of the TreeBASIS descriptor algorithm, shown in Fig. 5, takes a list of features detected using the FAST feature detector [17] and returns a descriptor for each feature. The FAST detector does not provide a dominant orientation or scale measurement. For our frame-to-frame feature matching for UAV applications, the scale changes and rotation between frames are small and can be ignored. This allows us to use a faster detector, and avoid trade-offs associated with providing increased invariance [28].

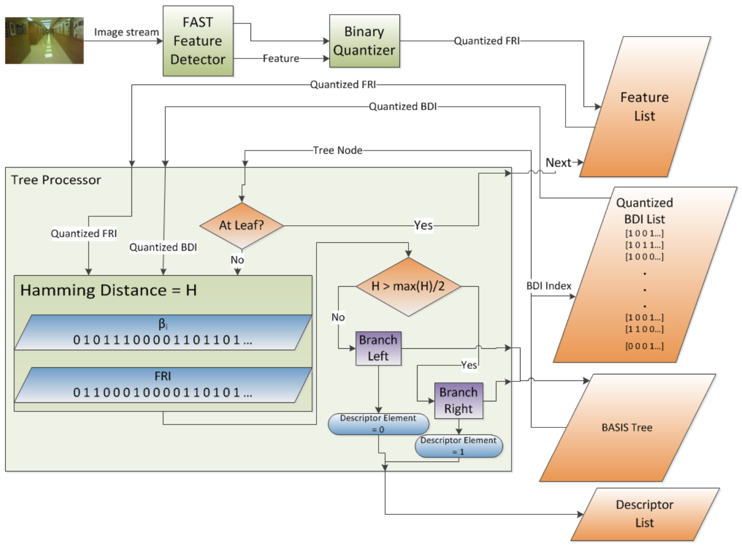

Figure 5: The on-line portion of the TreeBASIS algorithm takes a list of features from the FAST feature detector and returns, as a descriptor, the path the FRI traveled through the BASIS Tree.

For each feature returned from the detector, we obtain an FRI, which is binary thresholded into the same number of regions that was used to construct the tree, and the resulting vector is passed into the tree. At each node, a Hamming distance h is once again computed between the FRI binary vector in question and the binary EDBDI vector that was saved at the given node.

FRIs that are very similar to the current EDBDI ($h > \frac{|\beta|}{2}$)) are passed to the right child, and those that are dissimilar are passed to the left child. This is essential, because the fact that an FRI is dissimilar to a given BDI is just as informative as an FRI that is very similar to the BDI. When the FRI reaches a leaf node, the list of unique node numbers the FRI was compared to is returned as the feature's descriptor-path. Notice that because of the non-cyclical, binary nature of the tree, rather 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. Because each level requires only one bit to describe if the FRI went down the left branch or right branch of any given node in the path, 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

Tree structures are commonly used by feature detectors during matching in order to reduce the number of features that must be compared. An added benefit of the TreeBASIS algorithm is that the tree was already used to compute the descriptor, so the creation of an additional tree for matching is not necessary. Recall that each node in the BASIS tree is represented by an EDBDI, which represents an essential *feature characteristic*. Each node that an FRI passes through represents a comparison and a determination as to whether the FRI contains the given characteristic, or not. This means that, while comparing the descriptor-paths of two features, if two features follow the same path, they must contain the same feature characteristics represented by the EDBDIs they passed through. Conversely, if two features' descriptor-paths diverge, then they differ on a given feature characteristic and do not match. The depth at which the paths diverge is very important. The root node of the tree ideally splits the training set of FRIs equally in half. This means there is a 0.5 probability that any given FRI will go down the left or right branch. At each subsequent node, that probability is divided in half. If two features diverge at the root node, they are much more dissimilar than two features that diverge at a node much deeper in the tree.

Also, due to the fact that the partitions at each node are not guaranteed to be perfectly even, not all paths in the tree may progress to the deepest possible level. Because of this fact, descriptors may have variable lengths. For example, two FRIs, F_1 and F_2 may travel down paths of length n, while two other FRIs, F_3 and F_4 may travel paths of length m, where $m < n$. If F_1 and F_2 match m nodes correctly, that does not imply that they are as good a match as if F_3 and F_4 match all m nodes. In the former case, the features may be a good match, but not perfect, whereas the latter is a perfect tree match.

In order to accommodate these issues, given two features, the distance between the two features is computed as the paths are traversed. First, the distance is initially set to a large value. Next, the first element of each descriptor's path is compared. If the path elements are equal, the distance is reduced by

$$d = d(1 - \frac{p}{m}) \tag{4}$$

where p is the length of the path traversed so far and m is the total path length of the shorter descriptor path. Thus if the two descriptors are equivalent, the distance computation returns a value of zero. If the descriptors differ at any element along the path, the distance computation halts and current value of d is returned. In this way, the distance is a reflection of how much of the paths of two descriptors are similar, normalized by the overall path length. Because each descriptor ends when the path reaches a leaf node, two descriptors of differing lengths are guaranteed not to match on the final node of the shorter-length descriptor, and comparison will stop before passing the end of the shorter descriptor. Because the maximum depth of the tree is known before the on-line portion begins, each descriptor is allocated enough memory to store a maximum-depth path, even though some descriptor paths may not require this much storage.

3 Results

There should be something here – you can't start a section with a subsection.

3.1 Experiment Procedure

To create the Basis Tree for our evaluations, we used FRIs obtained from the *GoogleMaps* dataset. By using the *GoogleMaps* dataset for training, we were assured that there were no FRIs from the test dataset in our training set, so the evaluation data was completely separated from the training data.

Our evaluation dataset was the *Idaho* dataset also presented in [26]. Figure 6 shows two example images from the *Idaho* dataset. *Idaho* was created from real world images taken from a downward facing camera on an actual air flight. The images in the *Idaho* dataset were taken from a camera running at 30 frames per second, 640×480 pixel resolution. 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. The images used for the dataset were obtained from video frames that are one second apart to allow noticeable camera movement.

In order to measure the performance of TreeBASIS, we performed the same evaluation as that used on the original BASIS algorithm [26]. That is, a homography was computed from feature descriptors matched between images $\mathbf{I_1}$ and $\mathbf{I_2}$ using a RANSAC-based algorithm as follows. First, features were found in $\mathbf{I}_1$ using the FAST feature detector. We computed TreeBASIS descriptors for each feature, and the same process was repeated with $\mathbf{I}_2$. The descriptor distance metric described in Section 2.3 was used to find the distance between

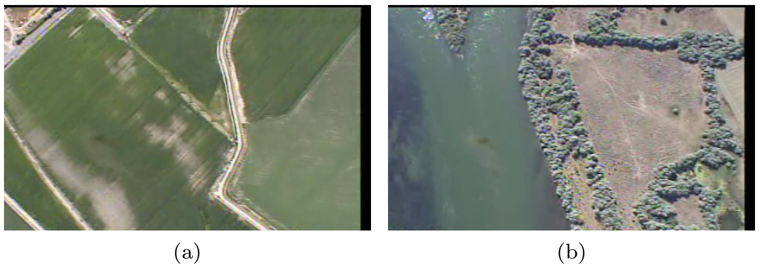
Figure 6: Sample images from the *Idaho* dataset.

each feature in $\mathbf{I_1}$ and all features in $\mathbf{I_2}$, and the matches whose distance was 0 (a perfect tree-path match) were kept. As a verification step, these features from $\mathbf{I}_1$ and their associated match were input into the RANSAC algorithm in order to compute a homography, H, relating $\mathbf{I}_1$ to $\mathbf{I}_2$. The feature points from $\mathbf{I}_1$ could then be warped by the homography using

$$p_2 = \mathrm{H} * p_1, \tag{5}$$

where p_1 is a feature point in $\mathbf{I}_1$ and p_2 is a feature point in $\mathbf{I}_2$.

Due to the nature of our dataset, we were able to assume that the homography relating $\mathbf{I}_1$ and $\mathbf{I}_2$ was a basic affine transform of the form

$$\mathbf{x}' = \mathrm{H}_{\mathrm{A}}\mathbf{x} = \begin{bmatrix} \mathrm{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x} \tag{6}$$

Because the rotation between frames is very minor (due to the inability of the airplane to make very sharp turns) along with the assumption that the airplane maintained a relatively uniform altitude between frames, we can identify a correct homography as simply a significant translation along with an affine matrix A that is almost equal to the identity matrix, $\mathbf{I}$. If the resulting homography H met these criteria, the result was considered an accurate match. Since the FAST feature detector is used in our experiments, the repeatability rates of detectors are not compared here, as those results can be found in previously published articles [29].

3.2 Experiment Results

As described in the previous section, we ran the images from *Idaho* through the TreeBASIS algorithm. TreeBASIS computed path-descriptors and saved perfect path-matches (distance = 0) between subsequent images. Using this list of perfect path-matches, we calculated a homography between each image pairs. For each returned homography, H, our system compared the elements along the diagonal of H to 1.0. If elements $H_{1,1}$ and $H_{2,2}$ were both within the range $[0.7, 1.3]$, we considered the homography computation successful. Values outside of the range $[0.7, 1.3]$ would define a rotation or scale change much larger than the aircraft could physically accomplish, which implies an incorrect homography. Table 2 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 using our metric. Our TreeBASIS algorithm, using 17-bit descriptors, and a tree built using 64 BDIs and a training set, F of 50,000 FRIs obtained an accuracy of 79.6% on the *Idaho* test set.

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%

Table 2: Accuracy results and memory footprints for BASIS and TreeBASIS on the *Idaho* dataset. Memory usage assumes 1,000 features per image are kept for each algorithm.

It may be difficult to understand initially why a reduction in descriptor size and computation would also provide an increase in matching accuracy. Recall that the basis dictionary set created by sparse coding is non-orthogonal. As stated in [26], due to the non-orthogonality of the basis set, many basis dictionaries may prove to be repetitive. That is, more than one BDI will describe the same "feature characteristic". In TreeBASIS, the development of the tree and the use of the entropy function guarantees that the only BDIs used are effectively descriptive BDIs (EDBDIs), which provides three distinct advantages.

First, by using only EDBDIs, we remove the redundancy inherent in the basis dictionary. The BDIs used in the BASIS tree are not redundant because describing the same feature characteristic would result in a poor division of the training set F which, by definition, would imply that the BDI is not an EDBDI. Second, the EDBDIs are the most effective BDIs of the entire basis dictionary, and by using them in the proper order (which order is developed due to the way the tree is created), fewer BDIs are required to discriminate between FRIs. Third, the tree structure reduces the number of comparisons required to discover the same information that the original BASIS descriptor computed. By comparing an FRI to an EDBDI at a node in the tree and branching based on that comparison, we remove an entire subset of EDBDIs that do not provide any necessary discrimination to the current FRI. This can be thought of as being similar to computing a full BASIS, SIFT, or SURF descriptor, but during matching, only looking at specific descriptor values in a pre-defined order and halting comparison when the descriptor values are no longer similar.

4 Conclusion

In this paper we have presented the development of the TreeBASIS feature descriptor algorithm. The TreeBASIS descriptor provides a unique method of describing feature points based on the prominent feature characteristic components they contain. The use of sparse coding algorithms to obtain basis dictionaries provides the TreeBASIS algorithm with a sparse coding dictionary set that resembles the receptive fields found in the visual cortex, and that are generically applicable across a wide range of natural images. TreeBASIS improves upon the original BASIS algorithm by computing a vocabulary tree based on essentially descriptive basis dictionary images (EDBDIs). These EDBDIs allow us to create a tree that drastically reduces the number of FRI - BDI comparisons required to compute a descriptor, and provide a unique descriptor-path that allows for very fast descriptor computations, comparisons, and matching.

The intended application of our TreeBASIS descriptor is for UAV frame-to-frame feature matching. We tested our TreeBASIS algorithm against the original BASIS algorithm on the *Idaho* test set and found that it provides an increased homography accuracy over BASIS. TreeBASIS descriptors also have a much smaller memory footprint than existing descriptors such as SIFT, SURF, and even BASIS.

The original BASIS algorithm has been modified to fit into hardware on an FPGA platform for embedded and low-resource vision systems. Our future work will consist of modifying and implementing the TreeBASIS descriptor into hardware to provide a complete description and matching vision system for low-resource applications. By developing a hardware correlation system, the entire vision system can be implemented in FPGA hardware, creating a complete system-on-a-chip computer vision solution for small, light-weight, embedded platforms.

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 *Proceedings of the International Society for Optical Engineering*, 2006, vol. 6384, p. 63840P.

[2] 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. 394406, 2006.

[3] Z. Wei, D. Lee, and B. E Nelson, “A Hardware-Friendly adaptive tensor based optical flow algorithm,” *Lecture Notes in Computer Science*, vol. 4842, pp. 43, 2007.

[4] B. Tippetts, D. Lee, J. Archibald, and K. Lillywhite, “Dense disparity real-time stereo vision algorithm for resource limited systems,” *IEEE Transac-*

tions on Circuits and Systems for Video Technology, vol. PP, no. 99, pp. 1–1, 2011.

[5] 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. PP, no. 99, pp. 1–1, 2011.

[6] Y. Kim, V. Sankar J, 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*, vol. PP, no. 99, pp. 1–1, 2011.

[7] 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 of the International Society for Optical Engineering*, 2006, vol. 6384, p. 63840.

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

[9] 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, Dec. 2008.

[10] W. S. Fife and J. K. Archibald, Archibald, "Reconfigurable On-Board vision processing for small autonomous vehicles," *EURASIP Journal on Embedded Systems*, vol. 2007, pp. Article ID 80141, 14 pages, 2007, doi:10.1155/2007/80141.

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

[12] Lifan Yao, Hao Feng, Yiqun Zhu, Zhiguo Jiang, Danpei Zhao, and Wenquan Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *International Conference on Field-Programmable Technology, 2009. FPT 2009*. Dec. 2009, pp. 30–37, IEEE.

[13] M. Schaeferling and G. Kiefer, "Flex-SURF: a flexible architecture for FPGA-Based robust feature extraction for optical tracking systems," in *2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. Dec. 2010, pp. 458–463, IEEE.

[14] J. Svab, T. Krajnik, J. Faigl, and L. Preucil, "FPGA based speeded up robust features," in *Technologies for Practical Robot Applications, 2009. TePRA 2009. IEEE International Conference on*, 2009, p. 3541.

[15] D. G. Lowe, "Distinctive image features from Scale-Invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[16] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvey Vision Conference*, vol. 15, 1988.

[17] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, Beijing, China, 2005, pp. 1508–1515 Vol. 2.

[18] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, pp. 43–72, 2005.

[19] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: speeded up robust features," *Computer VisionECCV 2006*, p. 404417, 2006.

[20] Y. Tsai, Q. Wang, and S. You, "CDIKP: a Highly-Compact local feature descriptor," in *The 19th International Conference on Pattern Recognition*, 2008, p. 1.

[21] C. Huang, C. Chen, and P. Chung, "Contrast context histogram - a discriminating local descriptor for image matching," in *Proc. 18th International Conference on Pattern Recognition ICPR 2006*, 2006, vol. 4, p. 5356.

[22] S. G Fowers, D. J. Lee, D.A. Ventura, and J.K. Archibald, "Nature inspired BASIS feature descriptor and its hardware implementation," *Submitted to IEEE Transactions on Circuits and Systems for Video Technology*, Feb. 2012.

[23] D. J Field, "What is the goal of sensory coding?," *Neural computation*, vol. 6, no. 4, pp. 559601, 1994.

[24] B. A Olshausen et al., "Emergence of Simple-Cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607609, 1996.

[25] B. A Olshausen and D. J Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?," *Vision research*, vol. 37, no. 23, pp. 33113325, 1997.

[26] S.G Fowers, D.J. Lee, D.A. Ventura, and B.J. Tippetts, "A novel feature descriptor for Low-Resource embedded vision sensors for Micro-UAV applications," *Submitted to AIAA Journal of Aerospace Computing, Information, and Communications*, Jan. 2012.

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

[28] Tinne Tuytelaars and Krystian Mikolajczyk, "Local invariant feature detectors: A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007.

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