Terrain-Relative Navigation for Precision Landings based on a Hierarchical Localization Approach

by

Hector Alejandro Li Sanchez

B.S. Electrical Engineering, University of Pittsburgh, 2019

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2021

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Hector Alejandro Li Sanchez

It was defended on

November 5, 2021

and approved by

Dr. Samuel Dickerson, PhD., Associate Professor, Department of Electrical and Computer Engineering

Dr. Ahmed Dallal, PhD., Assistant Professor, Department of Electrical and Computer Engineering

Thesis Advisor: Dr. Alan George, PhD., Department Chair and Mickle Chair Professor, Department of Electrical and Computer Engineering

Terrain-Relative Navigation for Precision Landings based on a Hierarchical Localization Approach

Hector Alejandro Li Sanchez, M.S.

University of Pittsburgh, 2021

Terrain-relative navigation (TRN) refers to a class of algorithms that can be used to obtain the precise location of a vehicle in a GPS-denied environment by using a pre-computed terrain map. This map consists of recognizable landmarks that can be used to establish correspondences between spacecraft measurements and the terrain. Although several TRN approaches have been successfully implemented in the past, there is still a need for robust, flexible, and efficient TRN algorithms capable of supporting localization across wide trajectory paths without the need for additional sensor measurements. In this work, a new TRN system based on hierarchical localization is designed and evaluated. The proposed system aims to take advantage of recent advances in algorithms for autonomous navigation in terrestrial environments to improve the efficiency over previous TRN implementations. An extensive evaluation of the proposed system is conducted using a lunar imagery dataset. To assess the feasibility of deploying the proposed TRN system in a real scenario, the system is realized using a Xilinx Zynq-7045 SoC—a device featured on relevant space platforms. Experimental results show that the system is capable of localizing over 90% of images featuring a wide range of scale, rotation, and illumination changes. In addition, leveraging the hardware acceleration capabilities of the FPGA contained within the target platform allows for processing of images at a rate of at least 25 FPS, which is sufficient for real-time operation.

Keywords: Terrain-Relative Navigation, Feature Extraction, Field Programmable Gate Array (FPGA), Computer Vision, System-on-Chip (SoC).

Table of Contents

List of Tables

List of Figures

Preface

This research was supported by industry and agency members of the NSF Center for Space, High-Performance, and Resilient Computing (SHREC) and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783. I would like to thank Dr. Alan George (University of Pittsburgh), Dr. Chris Owens (Astrobotic), Dr. Sebastian Sabogal (NASA GSFC), and Noah Perryman (University of Pittsburgh) for their exceptional mentorship throughout this project. Special thanks to Astrobotic for providing lunar imagery data used in this work.

In addition, I would like to thank my parents and friends for their continuous support throughout the years. Thanks to my girlfriend Pia for her emotional support, especially while I was conducting this research during the pandemic. Finally, I would like to thank my fellow graduate students at SHREC for their assistance and insightful discussions.

1.0 Introduction

Space missions that require landing on planetary objects other than the Earth present a unique set of challenges and constraints. In order to land safely and efficiently, it is crucial to have a precise knowledge of the spacecraft location in real-time so that correct navigation decisions can be performed. Localization in space is a difficult problem due to a lack of GPS, and approaches based solely on inertial measurement units (IMUs) suffer from drift over time. Terrain-relative navigation (TRN) refers to a class of computer vision algorithms that can be employed to aid autonomous navigation by greatly reducing the uncertainty in position [1]. This is accomplished by comparing images from an onboard camera during landing with a terrain map that is obtained *a priori*. Using this data, a position estimate relative to the terrain can be obtained, which addresses the problem of drift in dead reckoning methods. With reduced position uncertainty, highly precise landings can be achieved. Increasing landing precision is valuable as it enables safe landings closer to science objectives or in areas with relatively more hazardous terrain [2].

Several approaches to TRN have been designed over the years, with the Mars 2020 Perseverance rover being one of the most recent successful implementations of this technique. TRN approaches vary widely [3] in how the map information is represented, the methods used to compare camera images to the terrain map, and how this information is used to create a position estimate. Despite the successes of previous TRN implementations, improvements in cost, robustness, and efficiency of TRN systems are highly desirable. TRN systems for flagship missions such as the Mars 2020 Perseverance rover require years of R&D, cost millions of dollars to develop, and still have limitations in their range of operation due to performance constraints [2]. Consequently, there is a need for new TRN algorithms to support future missions, especially with the proliferation of commercial space operations.

In this research, the design and evaluation of a novel TRN system based on a hierarchical visual localization approach is presented. This approach draws inspiration from recent advances in algorithms for autonomous navigation of terrestrial vehicles. A hierarchical approach centered around image retrieval and feature matching greatly extends the amount of

terrain that can be covered by the map due to a highly efficient map representation and the method used to compare images from the camera with the map. Additionally, the system is designed to target a hybrid CPU+FPGA system, leveraging the acceleration capabilities of an FPGA to meet real-time processing requirements. A terrain map consisting of real lunar imagery obtained from a prior landing mission is used to evaluate the performance of the proposed TRN approach using several metrics.

2.0 Background

This chapter first presents an overview of previous works in TRN, highlighting the goals and motivations behind a new TRN approach. Additionally, the main concepts behind hierarchical localization—the basis for the proposed approach—are presented. Finally, several sections detail the mathematical background for the algorithms employed in the proposed system.

2.1 Terrain-Relative Navigation

One of the earliest uses of camera measurements to aid in localization during space missions was the Mars Exploration Rover Descent Image Motion Estimation System (MER DIMES) [4]. In this system, a series of three images taken during descent were used to reduce the uncertainty in the spacecraft horizontal velocity. Since then, several TRN systems have been independently developed.

Most TRN approaches are based on the concept of mapped landmarks. In [5], the authors present an overview of the TRN sub-system of the Lander Vision System (LVS) designed for the Mars 2020 Perseverance rover. This design is based on a two-stage process, consisting of a *coarse* and *fine* localization step. In the coarse localization step, three large image patches are matched to the map using a frequency-based correlation method. Using these matches, a general estimate of the position is obtained (within 2 to 3 km). This is followed by a fine localization step where a series of small image patches (up to 100) are matched using spatial correlation. This system was successfully deployed as part of the landing system for the Mars 2020 Perseverance rover. However, the correlation methods used in this approach require a reliable initial estimate of the spacecraft position. Namely, the approach requires a highly accurate estimate of the altitude (within 200 m), as template matching is not robust to changes in scale. Additionally, the high computational complexity of performing the template matching limits the size of the map to a single 1024×1024 image. The OSIRIS-

REx feature-based navigation system [6], which was designed to collect rock samples from the near-Earth asteroid Bennu, is also based on mapped landmarks. Here, the location of known features on the surface of the asteroid are tracked using a digital terrain map (DTM) and a spatial correlation approach to match the features. Due to the matching approach, this system is also highly dependent on an initial position estimate.

An approach that relies on similar principles, yet is fundamentally different than mapped landmark systems, is presented in [7]. Here, the authors exploit the abundance of craters on the lunar surface for a TRN algorithm based on crater detection. Since crater rims are elliptical in shape, they can be modelled as a conic laying on the lunar surface. By combining this assumption with a crater detection front-end and leveraging the geometric properties of conics, an invariant descriptor can be created. This descriptor can then be used to match against a database of craters. Though this approach is promising in some contexts, the dependence on distinguishable and favorable crater distributions limits its use in general scenarios. This is especially true for small bodies such as asteroids with non-uniform shapes. Finally, there are a few *map-free* methods, such as the one presented in [8], which resemble visual odometry techniques. In this system, there is no reference map, and a statistical framework is employed to generate a pose estimate without the need to identify features. However, the lack of an absolute reference means that it is impossible to avoid the accumulation of drift.

To address the problems associated with spatial correlation, the authors in [9] present the Optical Precision Autonomous Landing (OPAL) system. In this approach, the map representation consists of a set of 3D feature points which are computed using a custom feature extractor similar to SIFT [10]. These points serve as landmark features that are robust to perspective and illumination changes. Then, feature points are extracted from the camera during landing and matched to the database points to create a pose estimate. This approach resembles previous works in visual localization for terrestrial applications [11, 12, 13]. The feature point representation has an advantage over correlation-based methods, as the features' robustness to scale and rotation changes means that there is less reliance on an initial pose estimate. Additionally, representing the map using feature points instead of image patches results in reduced memory consumption.

To build upon the success of previous TRN systems, this work aims to incorporate elements from state-of-the-art hierarchical visual localization methods. One of the current limitations of feature-based TRN approaches is that when considering a large terrain map, the amount of features required to cover the map area is very large. This results in a high computational cost to match camera features to the database, as well as an increased likelihood of erroneous matches. To address this problem, a hierarchical localization approach is adopted, combining principles from [2], [9], and [13]. The main objective of the approach is to only select a subset of the feature points in the database for matching. This is accomplished by performing the matching using two stages: a *coarse* localization stage and a *fine* localization stage. In the coarse localization stage, high-level properties of the image are used to determine which feature points from the database are suitable candidates for matching. Then, feature point correspondences between camera and database points are established during a fine localization stage. An additional objective of this study is to speed up the matching of features by using approximate nearest neighbor (ANN) search techniques instead of a brute-force search. The following sections describe the mathematical background for the algorithms used to realize this pipeline in detail.

2.2 Scale-Invariant Feature Transform (SIFT)

In this section, a brief review of SIFT feature extraction is provided. This includes highlighting the major steps of the algorithm and associated computations. A detailed description of the SIFT algorithm can be found in [10].

The first step in the SIFT pipeline is the construction of a difference of Gaussians (DoG) pyramid used for keypoint detection. First, a scale-space pyramid is created by convolving the input image I with a series of 2-D Gaussian kernels G, defined in (2-1). The standard deviation σ_i of each Gaussian filter is computed by starting with a base standard deviation σ, then successively multiplying by a constant factor k to generate each subsequent σ_i.

$$G(x, y, \sigma_i) = \frac{1}{2\pi\sigma_i^2} e^{\frac{-(x^2+y^2)}{2\sigma_i^2}}. \tag{2-1}$$

In this design, the scale-space pyramid contains five scales, following the suggestion from the original authors of the SIFT algorithm [10] to use a base standard deviation $\sigma = 1.6$ and a constant factor $k = \sqrt{2}$. The Gaussian image corresponding to the last scale is downsampled and serves as the input image for the next octave in the pyramid. This process can be repeated an arbitrary number of times. To construct the DoG pyramid, each Gaussian image in the scale-space pyramid is subtracted from the image corresponding to the next scale.

After computing the DoG pyramid, candidate keypoints are selected by performing non-maximum suppression. For each pixel in the DoG pyramid (excluding the first and last scales), a 3×3×3 window consisting of neighboring pixels within the current and adjacent scales is created. Points are selected as candidate keypoints if they are local extrema within this window and have a magnitude larger than a certain threshold.

Once a point is selected as a candidate keypoint, another test is performed to reject features with a strong edge response for improved stability. First, the trace and determinant of the Hessian matrix H at the candidate keypoint location are computed by using the second-order gradients. This process can be observed in (2-2) through (2-4). From these values, the edge response of the candidate keypoint is evaluated using the inequality in (2-5). If the inequality is true, then the point is selected as a feature. The variable r serves as a thresholding parameter and can be adjusted to modify the detector's sensitivity.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2-2}$$

$$Trace(H) = D_{xx} + D_{yy} \tag{2-3}$$

$$Det(H) = D_{xx} \times D_{yy} - D_{xy}^2 \tag{2-4}$$

$$\frac{Trace(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{2-5}$$

Computation of each SIFT descriptor is performed using the Gaussian image from the scale-space pyramid corresponding to the octave and scale in which the keypoint was detected. First, a main orientation is assigned to the keypoint so that the feature descriptor is invariant to changes in rotation. To accomplish this, the gradient orientation and magnitude

of each pixel is calculated in a window patch centered at the keypoint location. Then, an orientation histogram consisting of 36 evenly-spaced bins is computed. The bin with the highest magnitude is selected as the main orientation.

The next step consists of computing the SIFT histogram. Each pixel in a square region around the keypoint location is rotated according to the assigned main orientation. Next, the region is divided into 16 sub-regions in a 4×4 square pattern. The size of the regions is dependent on the scale at which the keypoint was detected. An orientation histogram consisting of eight bins is computed for each sub-region. Trilinear interpolation is used to distribute the magnitude from each pixel into adjacent histogram bins. The 16 histograms are concatenated to produce an unnormalized vector consisting of $16 \times 8 = 128$ values.

Finally, the descriptor is normalized and each vector entry is quantized to eight bits, creating the final output descriptor. In the original SIFT implementation [10], normalization is performed using the L_2-norm. However, further research in [14] demonstrated that square-root normalization (rootSIFT) results in more accurate feature descriptors. For rootSIFT normalization, each entry is first divided by the L_1-norm. Then, the square root of each value is computed to produce the final normalized value.

2.3 Vector of Locally Aggregated Descriptors (VLAD)

Context-based image retrieval (CBIR) is a computer vision application consisting of searching across a large database of digital images. In most CBIR scenarios, the task is to retrieve one or more database entries whose content most closely resembles that of a given query image. In the context of autonomous navigation, CBIR can be used for a *visual localization* or *place recognition* task. In order to search the database, a compact image representation is needed, typically by combining local descriptors (such as SIFT features). Some popular algorithms used for this purpose include Fisher vectors [15, 16], bag-of-features (BOF) [17], and the vector of locally aggregated descriptors (VLAD) [18]. In this work, VLAD is chosen as a global descriptor due to its high representation accuracy, search efficiency, and low memory consumption per descriptor.

The VLAD descriptor requires training of a codebook, similar to BOF. The codebook $C = \{c_1, ..., c_k\}$ consists of k visual words (in this case, SIFT descriptors) and is learned using a clustering algorithm such as k-means. Then, a VLAD descriptor for a given image can be computed using a set of local descriptors. Each local descriptor x is aggregated by first finding the nearest visual word c_i for each descriptor using a nearest neighbor search. The difference between the descriptor and the visual word $x - c_i$ is computed, and the differences for each descriptor assigned to cluster c_i are accumulated. When using a local descriptor with d dimensions and a codebook containing k visual words, the resulting VLAD representation contains a total of $D = d \times k$ dimensions.

After all local descriptors are aggregated, the VLAD descriptor is normalized in a two-step process as presented in [19]. First, an intra-normalization step where the accumulated vectors corresponding to each codebook word v_i are L_2-normalized is performed. Then, an inter-normalization step where the entire descriptor is L_2-normalized is performed to create the final output descriptor. Finally, the dimensionality of the VLAD descriptor is optionally reduced using principal component analysis (PCA) to reduce its memory footprint and increase the search efficiency [18].

2.4 Approximate Nearest Neighbors (ANN) Search

In order to establish correspondences between query and database features, a nearest neighbors search must be performed. The only way to obtain the true k-nearest neighbors for a given query vector x is to exhaustively compute the distance between the query vector and each database vector y_i, using a metric such as the Euclidean distance. The complexity of an exhaustive nearest neighbors search where N query vectors are matched to M database vectors and each vector has d dimensions is $O(dNM)$. Therefore, this approach presents scalability issues when the dimensionality of the vectors and the amount of comparisons to make is large. To address this issue, significant research has been devoted to algorithms for an *approximate* nearest neighbors (ANN) search. With these algorithms, the computational complexity and memory consumption of the search is greatly reduced, at the

cost of returning vectors with a high probability of being the true nearest neighbors instead of a guarantee. This work focuses on two ANN approaches to improve the search efficiency: Product quantization and the inverse file data structure.

2.4.1 Product Quantization (PQ)

In product quantization, database vectors are encoded into a compact representation to speed up computing distances between query and database vectors [20]. The process used to encode each database vector is shown in Figure 2.1. First, each database vector is separated into m groups of equal size. Then, a set of m subquantizers are trained on the database vectors using a clustering method such as k-means. Each subquantizer will produce a total of k replication values (for this work, 256 replication values are used). To encode the vector, each of the m groups is first quantized to its nearest value learned by the subquantizers. Then, each group is encoded using an 8-bit value corresponding to its index in a lookup table of all replication values. For a vector with d dimensions encoded using m subquantizers, the number of vector elements is reduced by a factor of d/m.

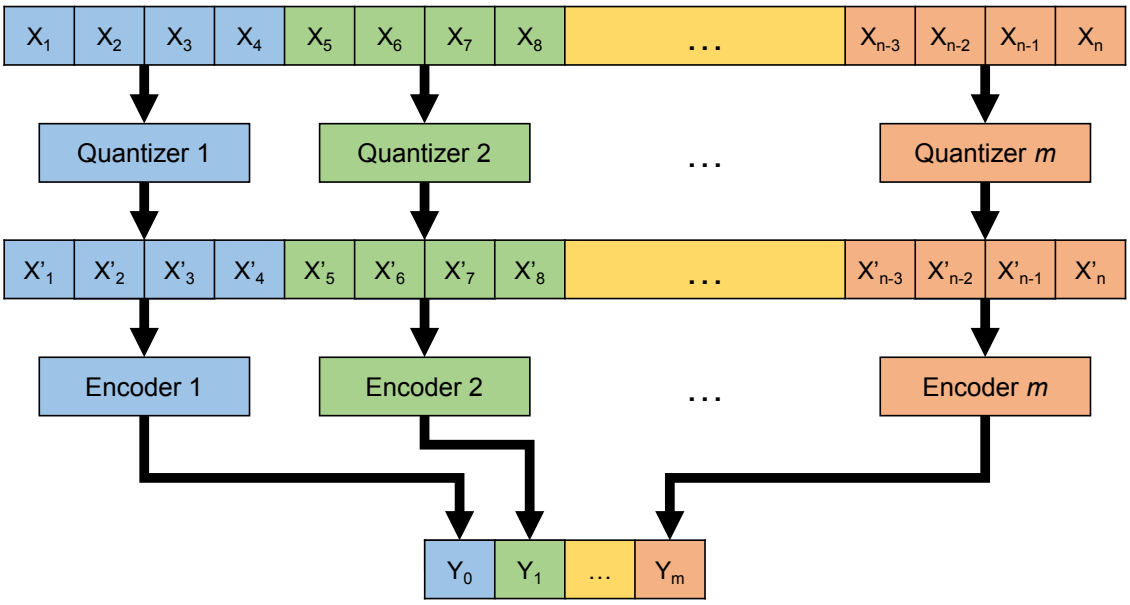


Figure 2.1: Conceptual diagram for PQ vector encoding.

In addition to reducing the amount of memory required to store each vector, using PQ speeds up the computation of the nearest neighbors search. As previously discussed, computing an exhaustive nearest neighbors search for N d-dimensional query vectors across a database with M entries requires $O(dNM)$ multiplications and additions. When using PQ, each group is encoded to one of k discrete values. Therefore, it is possible to pre-compute the distances between the query vector and each of the m groups for all replication values k and store them in a lookup table. Then, computing the distance between each query vector and database vector consists of m table lookups and $m - 1$ additions. Overall, the total number of multiplications and additions are reduced to $O(dNk)$ and $O(mNM)$ additions, respectively. In practice, $k << M$, so the number of multiplications needed to perform the search is reduced substantially.

2.4.2 Inverse File (IVF)

An additional technique used to speed up computation is to use an inverse file data structure to reduce the number of vector comparisons. Here, a coarse quantizer consisting of C codebook words learned through k-means is used to separate database vectors into C groups. Each database vector is assigned to the group corresponding to its nearest codebook word in the coarse quantizer. Then, when performing the search, the nearest codebook word for each query vector is also computed. Each query vector is only matched with database vectors belonging to the same cluster. With this approach, the average number of vector comparisons required is decreased by a factor of C. The accuracy of the search is slightly decreased, as it is possible that the true nearest neighbors of a query vector do not belong to the same cluster as the query vector.

3.0 Approach

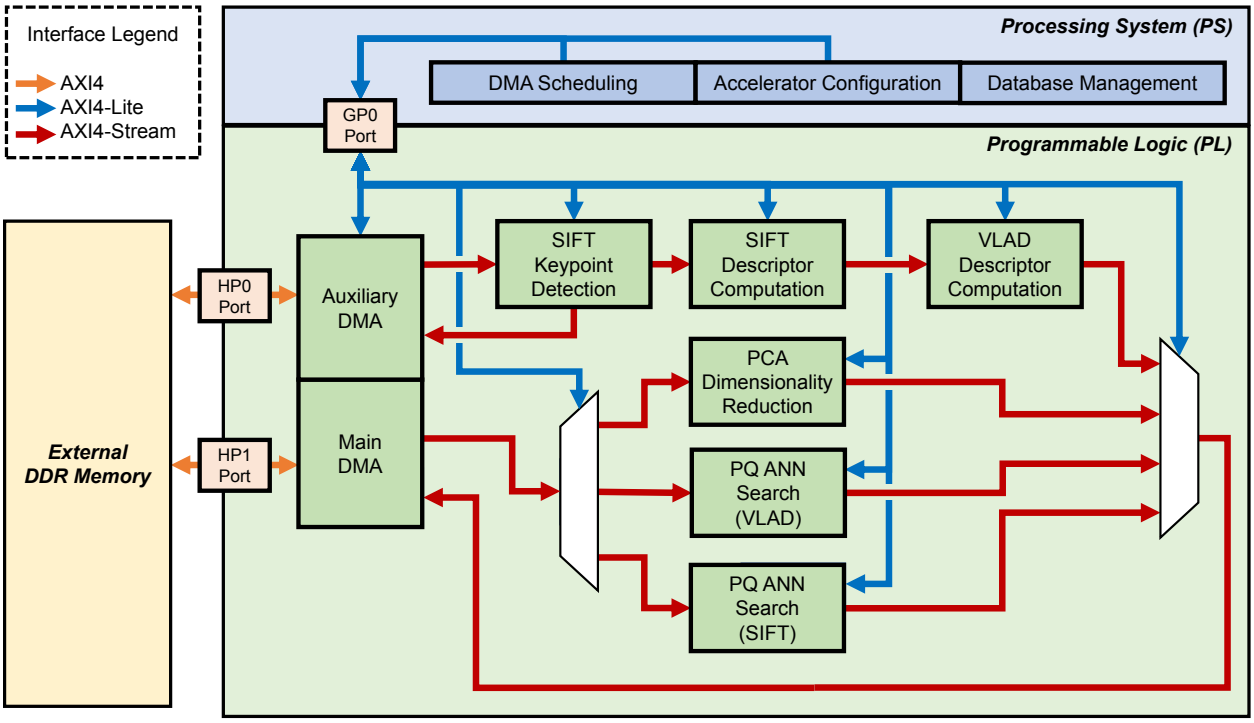


Figure 3.1: System architecture diagram highlighting major sub-components.

This chapter presents a detailed description of the proposed hierarchical TRN pipeline and its realization on a hybrid CPU+FPGA System-on-Chip (SoC) device. A hybrid architecture was chosen to leverage the parallelism enabled by an FPGA to accelerate compute-intensive parts of the application, while the CPU is used to perform most of the control-oriented segments. The overall system architecture is shown in Figure 3.1. For each input image (which will also be referred to as the query image), the task is to establish feature point correspondences between the query image and the database. The TRN pipeline consists of three major stages: First, in the *feature extraction* stage, SIFT and VLAD features are extracted from the input image. Then, in the *coarse localization* stage, a nearest neighbor search between the extracted VLAD descriptor and a pre-computed database yields a rough estimate of the map region that corresponds to the view of the camera. Finally, the

SIFT features extracted from the query image are matched to a subset of the database features in the *fine localization* stage. The system operates using a collection of stream-based accelerators. Each accelerator module also contains an Advanced eXtensible Interface 4 - Lite (AXI4-Lite) interface that is used to configure each module and initiate direct memory access (DMA) transfers. The CPU manages configuration of the DMA controller, accelerator modules, and selection of the database features.

3.1 Feature Extraction

The first major stage of the TRN pipeline is feature extraction. Here, the input image (with a resolution of 1024×1024 pixels in this design) is transformed into a compact representation consisting of local and global feature descriptors. This process is illustrated in Figure 3.2. SIFT feature extraction is performed to obtain a set of N local descriptors, where each descriptor contains 128 dimensions. Then, these features are aggregated into a single VLAD descriptor using 16 codebook clusters, which serves as a global representation for the image. A series of FPGA modules were designed to perform these tasks concurrently and efficiently. These modules are described in the following subsections.

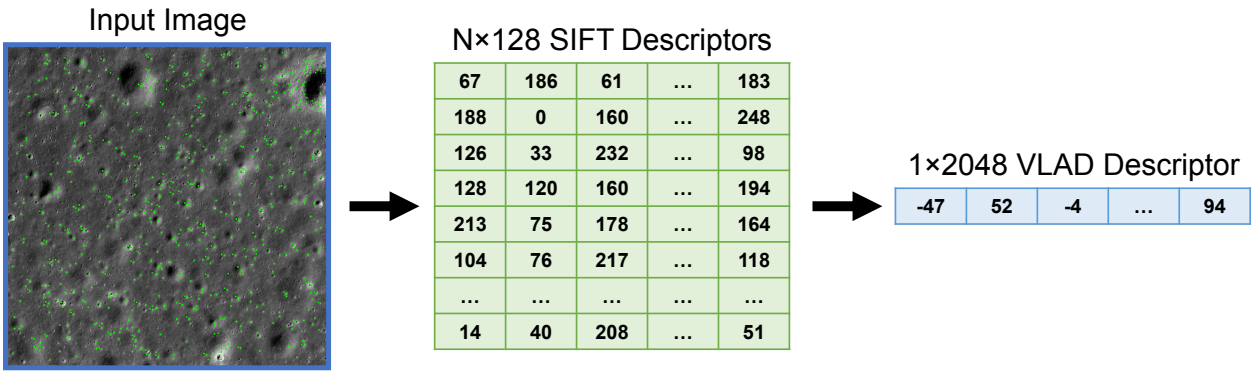


Figure 3.2: Illustration of the local SIFT features (green) and the global SIFT descriptor (blue) computed during feature extraction.

3.1.1 SIFT Keypoint Detection

A diagram illustrating the streaming architecture for keypoint detection is shown in Figure 3.3. To compute the DoG pyramid, each Gaussian image in scale space is constructed in parallel. To reduce resource utilization, the separability of the Gaussian kernel is exploited to compute each 2D convolution as two 1D convolutions operating across the rows and columns of the image. The vertical filtering step is performed first so that the resources required to implement the line buffer are shared across all scales. The coefficients of each Gaussian kernel are converted to fixed-point representation (with 8 integer bits and 8 fractional bits) so that the convolutions can be performed efficiently. The Gaussian image corresponding to the first scale is copied to an output stream used for computation of the SIFT descriptors. The input for the next octave is computed by downsampling the Gaussian image corresponding to the last scale by a factor of two and writing it to an output stream. A series of subtractors are used to compute the DoG pyramid from the scale-space pyramid. The resulting architecture produces a stream for each image in the DoG pyramid and processes one pixel per cycle.

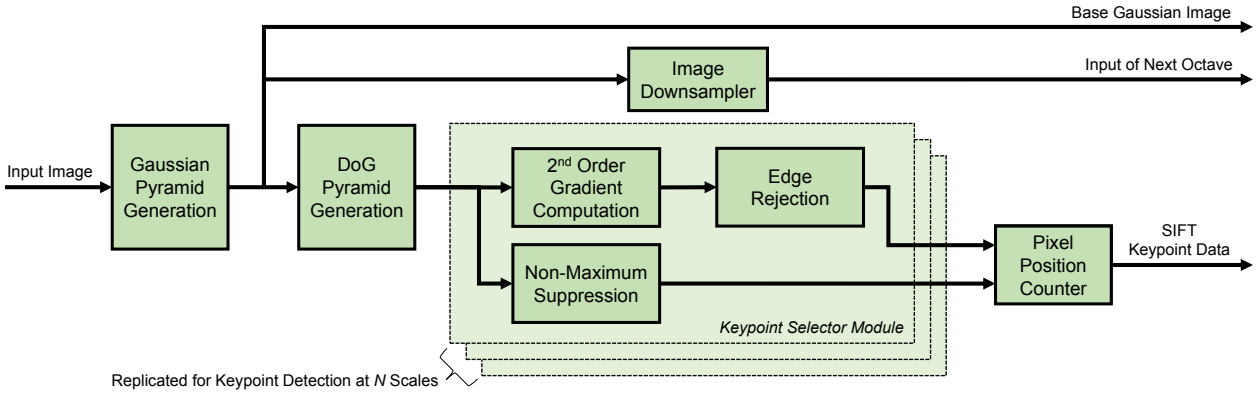


Figure 3.3: FPGA architecture diagram for stream-based SIFT keypoint detection.

The DoG pyramid streams are fed into a series of keypoint selection modules to detect SIFT keypoints. Each module contains two datapaths that implement the non-maximum suppression and edge-rejection tests in parallel. For non-maximum suppression, a 3×3 sliding window is instantiated for each scale, combining adjacent scales to form the 3×3×3 window. To perform the edge-rejection test, the second-order gradients of the DoG image

are calculated using a sliding window, feeding the resulting stream into another module that calculates the formulas shown in (2-3), (2-4), and (2-5).

Each test produces a single-bit output indicating whether or not the point should be rejected. If both tests pass, a 64-bit value containing the octave, scale, row/column position, and response strength of the resulting keypoint is written to the output stream. By performing detection across all scales in parallel, the system is able to perform the keypoint-detection step at a rate of one pixel per cycle regardless of the number of features detected.

3.1.2 SIFT Descriptor Computation

Based on prior works, it is evident that the SIFT descriptor computation is the main bottleneck. The problem is related to the large size of the descriptor window used for main orientation assignment and computation of the SIFT histogram. To implement the descriptor window using BRAM without the performance degradation introduced by stalling the keypoint-detection pipeline each time a descriptor is computed, a new architecture centered around parallel processing elements (PEs) is proposed. A diagram of the proposed architecture is shown in Figure 3.4. In this architecture, execution time is greatly improved due to two optimizations. Firstly, leveraging the simple dual-port configuration of the BRAM greatly reduces the number of stall cycles whenever a descriptor is computed. Additionally, this configuration enables the introduction of independent PEs, which can compute multiple SIFT descriptors in parallel.

To compute each SIFT descriptor, a 15×15 window is used for the main orientation assignment, while a 31×31 window is used to create the SIFT descriptor histogram. Due to the large size of the sliding window, it is more beneficial to realize it using BRAM storing 31+ lines from the image in a ring-buffer structure than to instantiate a large number of registers. By streaming data into this buffer as the Gaussian image is computed, the descriptor window buffer contains all of the pixels required to compute a descriptor for the keypoint located at the center of the 31×31 window at any given time. In previous works, a single-port BRAM is used to implement the buffer. For the rest of the thesis, such an implementation

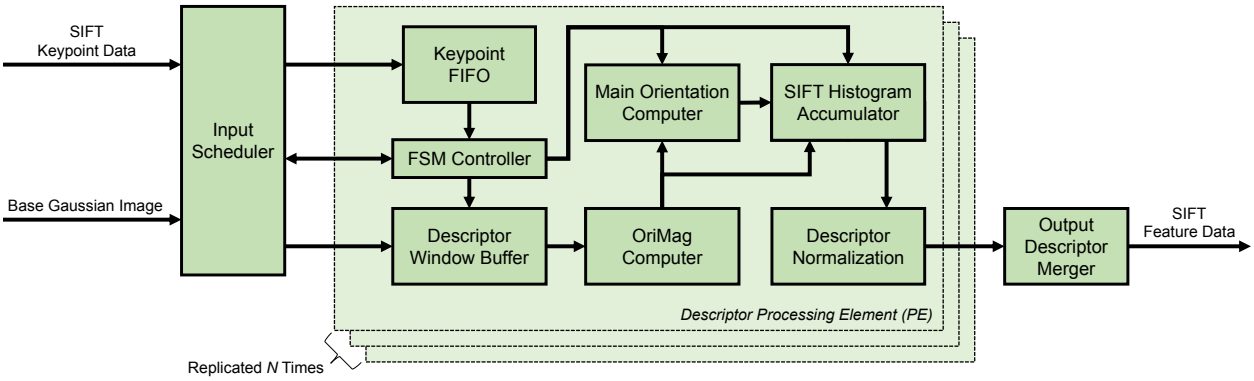


Figure 3.4: FPGA architecture diagram for stream-based SIFT descriptor computation.

will be referred to as the *naive* architecture. In Figure 3.5, a series of conceptual diagrams illustrate the limitation of this naive architecture and how the proposed solution addresses this limitation. Within each diagram, KPT and DSC represent operation of the keypoint-detection and descriptor-computation pipelines, respectively. Pixels corresponding to the location of detected keypoints are labeled with a letter.

In the naive implementation, it is not possible to write new pixels into the descriptor window buffer while also reading pixels needed for the descriptor computation from the buffer because there is only one memory port. Whenever a descriptor is computed, the keypoint detection pipeline must stop completely until the entire descriptor is computed. Consequently, the overall execution time will increase linearly with the number of features, degrading the performance of the system. In the proposed architecture, this would correspond to a delay of 1,382 cycles per feature. The time required to compute each SIFT descriptor is illustrated in Figure 3.5 by a delay of five cycles.

To address this issue, the dual-port configuration of the BRAM is leveraged, where reads and writes can be performed concurrently at two independent addresses. Using this configuration, values needed to compute the descriptor can be read independently of the write port used to store new pixels from the keypoint-detection step. Thus, stalling the keypoint-detection pipeline can be avoided until another keypoint is detected. Even though computing each descriptor still requires the same number of cycles, the overall execution

time decreases because the keypoint-detection and descriptor-computation pipelines operate in parallel for some time. This is illustrated in the second diagram of Figure 3.5. Keypoints are expected to be located sparsely within the image, so many stall cycles can be eliminated using this strategy without any additional resource costs.

Further performance improvements can be achieved by instantiating multiple PEs, which can compute SIFT descriptors independently of each other. It is simple to understand how using N descriptor PEs should decrease descriptor computation time by a factor of N. In practice, the actual performance improvement is greater due to the dual-port BRAM configuration. The amount of stall cycles eliminated is inversely proportional to the distance between two consecutive keypoint locations. When multiple PEs are used, keypoints are distributed equally across all PEs, so sparsity of keypoints within each PE is increased. This can be seen when comparing the second and third diagrams in Figure 3.5. When using one PE, the keypoint-detection pipeline must stall once keypoint B is reached until computation of descriptor A is completed. When using two PEs, descriptors A and B are computed by independent PEs. Hence, the keypoint-detection pipeline would not need to stall until keypoint C is reached. This benefit is strengthened as more parallel PEs are added.

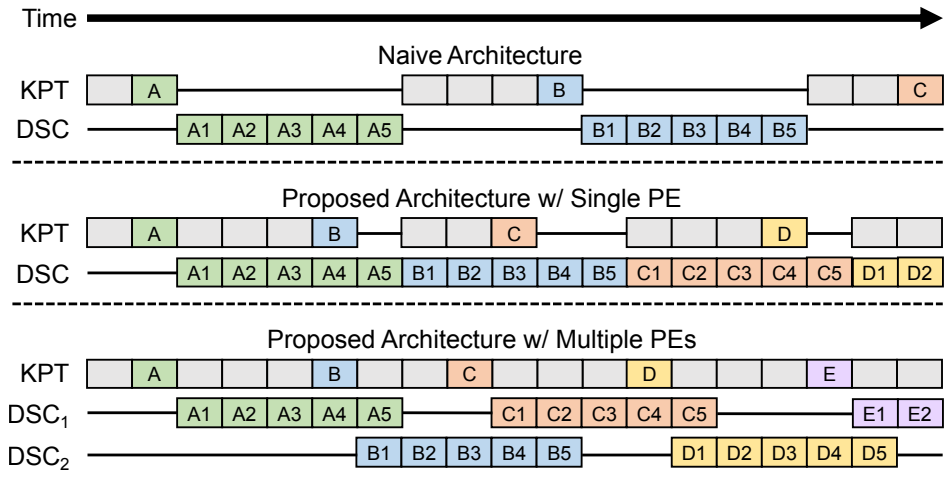


Figure 3.5: Conceptual timing diagrams illustrating how the proposed architecture reduces the amount of stalls in the keypoint-detection pipeline.

The orientation and magnitude (OriMag) windows are generated serially, with the window used for main orientation assignment being calculated first. The first-order gradients of the image are computed using a 3×3 sliding window, which is then connected to another pipeline that computes the magnitude and orientation in parallel. The required inverse tangent and square-root operations are implemented as fixed-point functions using the CORDIC IP module available as part of the Xilinx IP library [21]. Input gradients are stored as 9-bit signed integers, the output magnitude is stored as a 10-bit unsigned integer, and the output orientation is stored as a 6-bit unsigned integer corresponding to 5.625 degree increments.

To compute the main orientation, each value in the OriMag window is used to create an orientation histogram. A histogram containing 32 bins instead of 36 bins is used, which simplifies assigning each data point from the OriMag window to a bin. While the histogram is being computed, a simple comparator circuit keeps track of the bin index with the largest magnitude. Once the entire window has been processed, a flag is asserted to indicate that the main orientation is valid and computation of the main descriptor histogram can begin.

The SIFT descriptor histogram is computed in a similar fashion to the main orientation assignment, processing data one sample at a time. However, a more complex circuit is needed to accumulate the histogram due to the required rotation and triliniear interpolation. To implement the rotation, the coordinates of each sample (x, y) relative to the window center are rotated using (3-1) and (3-2) according to the main orientation θ. A small lookup table is used to compute the sine and cosine of the main orientation angle, since the angle is limited to only 32 discrete values.

$$x' = x\cos\theta - y\sin\theta \tag{3-1}$$

$$y' = x\sin\theta + y\cos\theta \tag{3-2}$$

For the trilinear interpolation, each data point may contribute to up to eight entries in the descriptor vector. Therefore, eight BRAM arrays are used to accumulate the SIFT descriptor histogram, each containing 128 entries. For each datapoint, the rotated coordinates (x', y') and orientation for each data point are used to calculate how the magnitude should distributed across the eight bin indices. Each BRAM array can be accessed independently,

so the accumulation can be pipelined to process one data point per cycle. Once the entire window has been processed, values from the eight BRAM arrays are summed using an adder tree, and a 128-entry array containing the unnormalized SIFT descriptor is streamed to the next sub-module.

The complete unnormalized histogram is streamed into a final module that performs normalization as done in rootSIFT. The input is first converted to floating-point representation and stored in a single BRAM cell, while the L_1-norm of the histogram is computed in parallel. Floating-point operations are implemented using modules available in the Xilinx IP library [22], which make use of dedicated DSP resources for area efficiency. Each entry in the histogram is divided by the L_1-norm, followed by a square root operation. Finally, the normalized entries are converted from floating-point to fixed-point representation and quantized to eight bits. Outputs from each PE are merged into a single output stream using a series of FIFOs and a multiplexer.

3.1.3 VLAD Descriptor Computation

As local SIFT features are computed, they are streamed out to a module responsible for computing a single VLAD descriptor representing the entire input image. In Figure 3.6, a diagram illustrating the FPGA architecture for the VLAD descriptor computation module is shown. Codebook coefficients are stored in BRAM using an 8-bit signed representation and are configured via the AXI-Lite interface. SIFT features are received via an AXI-Stream interface and are temporarily stored in a small set of registers during aggregation. To perform the aggregation, a nearest neighbor search is performed between the incoming descriptor and each of the 16 VLAD codebook cluster centers. Eight subtractions are performed in parallel in order to maximize performance for the given stream data width. Additionally, a processing element consisting of the eight parallel subtractors and nearest neighbor search sub-modules can be replicated to aggregate up to N descriptors in parallel. Once the nearest cluster center is found, the SIFT residual is computed and added to the VLAD descriptor memory. The index of the nearest cluster and the resulting SIFT residual are written to the output stream, which will be used during the fine localization stage. Overall, it takes 336 cycles

to aggregate each feature, which is less than the number of cycles needed to compute each SIFT descriptor. The SIFT and VLAD modules operate concurrently, so performing the aggregation of the SIFT features does not affect the overall execution time.

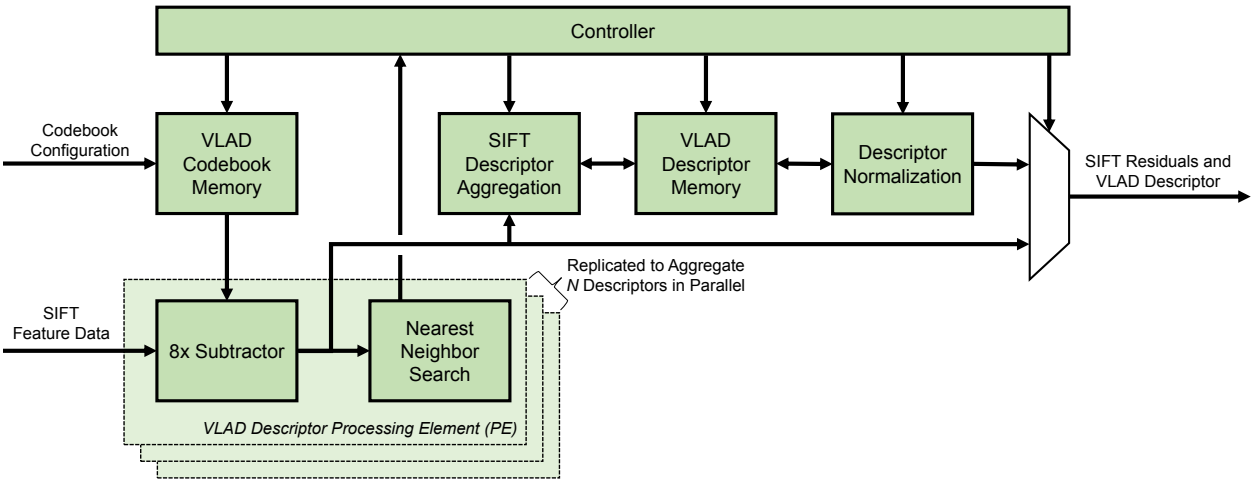


Figure 3.6: FPGA architecture diagram for stream-based VLAD descriptor computation.

Once all SIFT descriptors have been aggregated, the VLAD descriptor undergoes a normalization stage. Following the method in [19], normalization consists of an intra-normalization step followed by an inter-normalization step. The VLAD descriptor is first converted to floating-point using modules from the Xilinx IP Library [22]. Then, a pipeline consisting of multiplication, accumulation, and square root floating-point modules is realized to perform the normalization. The intra-normalization and inter-normalization steps are performed serially, so the same pipeline is reused between steps to reduce resource utilization. After the inter-normalization step, values are quantized to an 8-bit signed representation and are written to the output stream. The normalization process takes a total of 6,313 cycles and is performed once per image.

3.2 Coarse-Level Localization

After SIFT and VLAD descriptors have been extracted from the image, the next stage in the TRN pipeline is to perform a coarse localization step. In this stage, the VLAD descriptor obtained during the feature extraction stage is used to retrieve a set of map tiles. This stage consists of first reducing the VLAD descriptor dimensionality from 2048 to 128 using PCA, then performing an ANN search across the entire VLAD database and retrieving the top k matches. The objective is to retrieve database tiles that correspond to the same area of the map that is visible in the camera image.

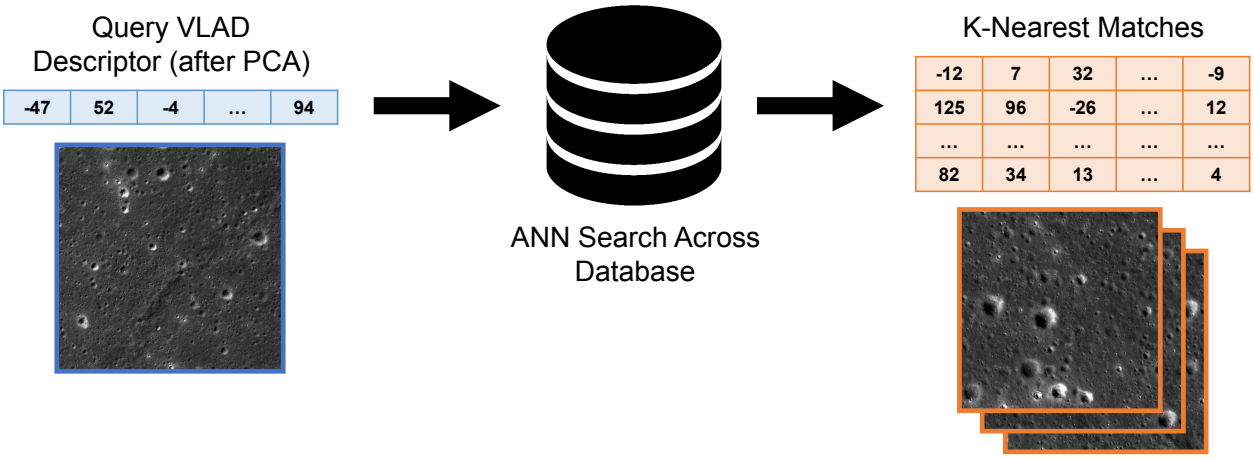


Figure 3.7: Illustration of the coarse-level localization stage using a nearest neighbors search.

3.2.1 PCA Dimensionality Reduction

Performing the nearest neighbor search using the full VLAD descriptor (which contains 2048 dimensions) is cost-prohitive in terms of execution time and memory consumption, so the dimensionality of the vector is reduced from 2048 to 128 via PCA. During creation of the database, the principal components of the database VLAD vectors are computed, and the 128 components that capture the highest variance are kept. Coefficients are quantized to an 8-bit signed representation.

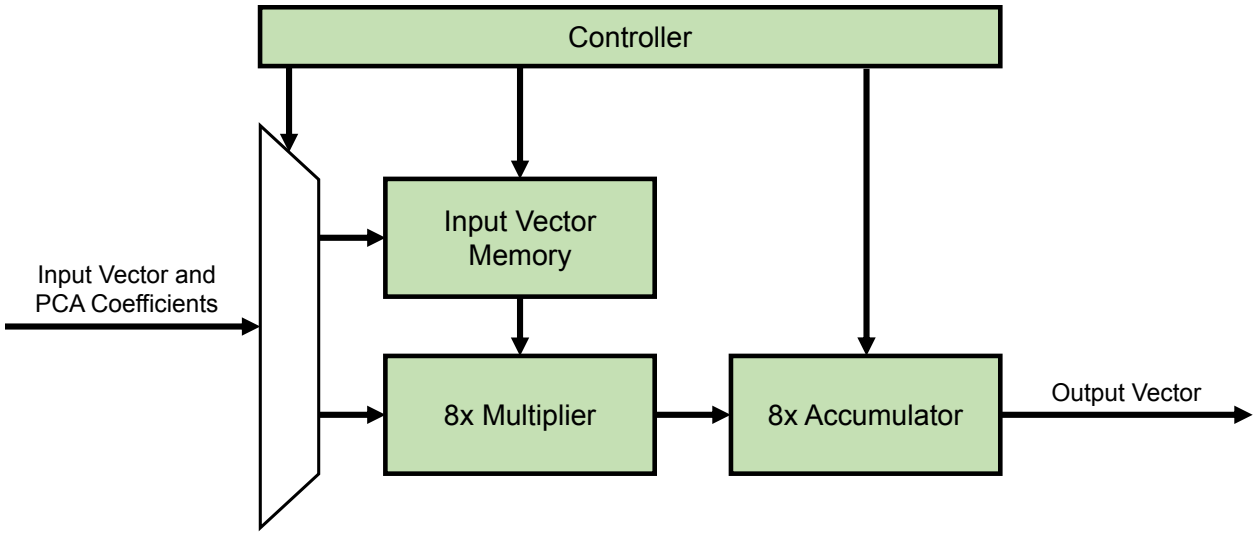


Figure 3.8: FPGA architecture diagram for stream-based PCA dimensionality reduction.

To perform the dimensionality reduction with high efficiency, an accelerator module following the architecture shown in Figure 3.8 was constructed. For this accelerator, a 64-bit wide stream containing both the input vector and PCA coefficients is used. First, the input vector (consisting of 2048 8-bit signed values) is read from the stream and stored in a small memory. Then, PCA coefficients are read from the stream at a rate of eight values per cycle. A set of multipliers and accumulators are used to perform the vector/matrix multiplication. Since the PCA coefficient matrix is fixed and computed ahead of time, its values are rearranged for optimal dataflow. The execution time for this step depends solely on the input and output vector dimensions. For input and output dimensions of size N and M, the dimensionality reduction takes a total of $(N + NM + M)/8$ cycles.

3.2.2 ANN Search using PQ

After reducing the dimensionality of the query VLAD descriptor, an ANN search is performed to retrieve the top k tiles from the database. The ANN search is performed in the FPGA using the accelerator architecture shown in Figure 3.9. This design operates using a single input and output stream and consists of three stages. In the first stage, the query

VLAD descriptor (after PCA) is read from the input stream and stored in a register array. Then, the PQ codebook is streamed through the module. During this stage, the asymmetric distance lookup table containing the L_2 distance between the query vector and each of the codebook words is computed. By computing the lookup table in this fashion, there is no need to store the codebook coefficients in BRAM, which saves resources. Finally, database features are streamed through the module. As data is read from the input stream, the distances between query and database vectors are computed efficiently using the asymmetric distance lookup table. After each distance is computed, the distance is sent to a module responsible for keeping track of the k-nearest neighbors using a list. Once all database vectors have been processed, the match indices and distances are written to the output stream.

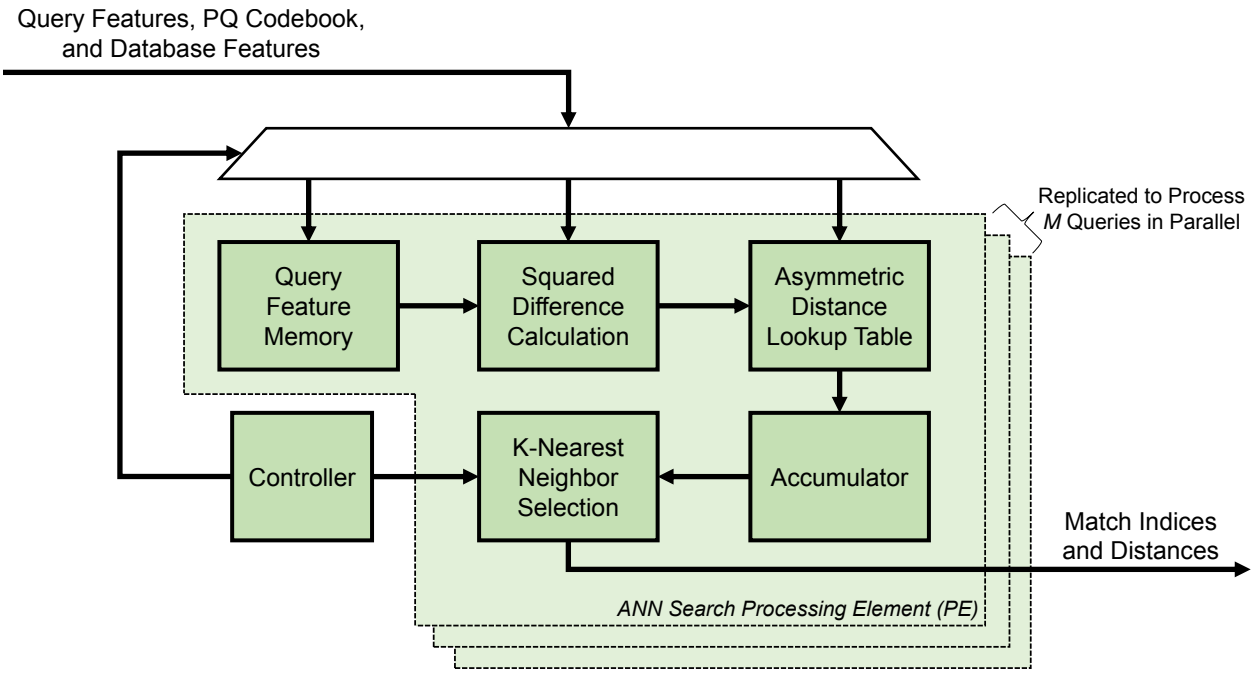


Figure 3.9: FPGA architecture diagram for stream-based PQ ANN search.

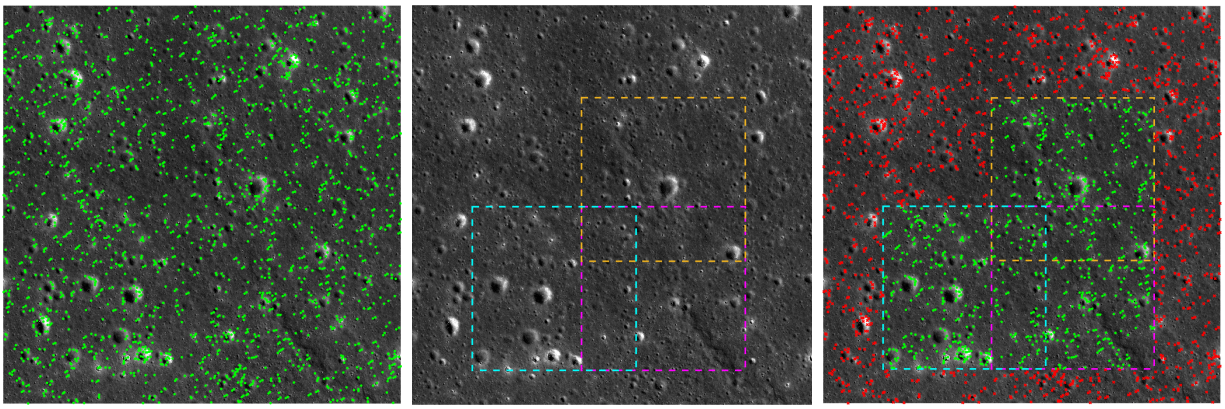
The design shown in Figure 3.9 contains several configurable parameters. These include the dimensionality of the vectors, the number of subquantizers used for the PQ encoding, and whether the vectors are signed or unsigned. Additionally, the architecture supports perfoming search for multiple queries in parallel by replicating PEs to match the number of

query vectors. This flexibility allows for efficient reuse of this module to match the SIFT features, which is done in the next stage of the TRN pipeline. The execution time for this module is dependent on the various configuration parameters. Consider a search consisting of N d-dimensional query vectors and a database containing M vectors, each encoded using m subquantizers with k replication values. The number of cycles required to compute and return the S nearest matches for each query is shown in (3-3). Since the number of database vectors is usually much larger than all other variables, the number of subquantizers m plays a significant role in the time needed to perform the search.

$$T_{PQ} = (Nd + kd + Mm)/8 + NS \tag{3-3}$$

3.3 Fine-Level Localization

The third stage in the TRN pipeline is a fine localization stage, where SIFT features from the query image are matched to the database. This is accomplished by using coarse localization results to select a subset of the database SIFT features, followed by another ANN search. Additionally, an inverse file data structure is used to reduce the amount of vector comparisons. The process used to select the database SIFT features is depicted in Figure 3.10. Each region of the database map contains a dense amount of features, as shown in Figure 3.10a. However, most of the features are irrelevant for a given query, as they are located outside the camera view. Using the coarse localization results, it is possible to obtain an estimate of the region of interest that should align with the view of the camera, as shown in Figure 3.10b. The SIFT database features are stored in memory using a tree data structure such that they are grouped into tiles. Using this approach, it is possible to determine which database SIFT features lie inside the region of interest as pictured in Figure 3.10c without the need to evaluate the coordinates of each individual feature.



(a) (b) (c)

Figure 3.10: Illustration of the database SIFT feature selection process using coarse localization results. (a) Database SIFT feature locations for a section of the terrain map. (b) Tile boundaries corresponding to the retrieved VLAD database indices. (c) Selection of a subset of database SIFT features that lies within any of the tile boundaries.

3.3.1 ANN Search using PQ+IVF

To match the SIFT features, another instance of the FPGA module used during the coarse localization step is used, with a few modifications to the configuration. Multiple PEs are instantiated to process several query descriptors at the same time. The two nearest matches for each query feature are returned to perform an outlier rejection step using the ratio of their distances, as described in [10]. In order to speed up the computation and increase the accuracy of the search, the vector search is performed using an inverse file. Because the inverse file and VLAD descriptor use the same number of clusters and are trained using the same data, they share identical codebooks. The indices computed during aggregation of the SIFT features can be used to separate the query features into clusters. Then, only database and query features that belong to the same cluster are compared. This reduces the number of comparisons per query and consequently speeds up the computation of the matching step significantly.

Using the inverse file data structure also contains benefits regarding PQ encoding. As described in [20] and shown in Figure 3.11, the distribution of vector residuals has a smaller variance and is normally distributed, which enables a more efficient encoding of database vectors. Consequently, the distortion error introduced by PQ encoding is reduced, leading to a more accurate search. Thus, it is more beneficial to use the vector residuals for matching instead of the original SIFT descriptors. Once again, the earlier computation of the VLAD descriptor can be leveraged to obtain the residuals for the query SIFT features without any additional cost. This is because computing a VLAD descriptor already requires computing the vector residuals between each SIFT feature and its nearest codebook word. The VLAD accelerator module is configured to output the SIFT descriptor residuals instead of the full descriptor values for this purpose.

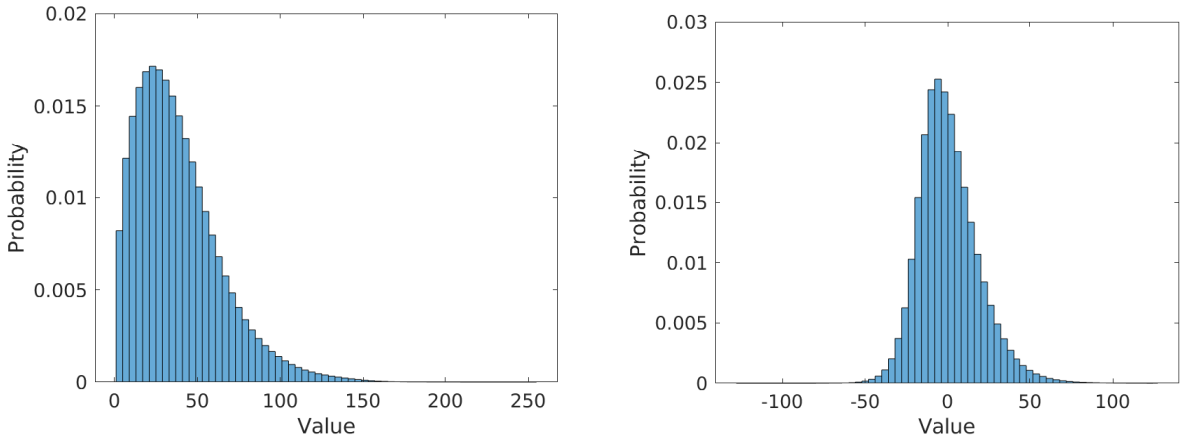


Figure 3.11: Histograms summarizing the value distribution of (a) original database SIFT descriptors and (b) residuals between original database SIFT descriptors and their nearest codebook word.

4.0 Performance Evaluation

An extensive evaluation of the proposed TRN system was performed to demonstrate its viability in a real-world scenario. To accomplish this, a custom visual localization dataset was constructed using real lunar images obtained from the Lunar Reconnaissance Orbiter (LRO) spacecraft. Then, the system described in the previous chapter was realized on a development board that resembles the architecture of a relevant space computing platform. Finally, experimental results are presented and the performance of the TRN system is analyzed using several metrics.

4.1 Experimental Design

This section contains details about the experimental design that is used to evaluate the proposed TRN approach. First, the procedure used to generate the dataset used for evaluation is described. Then, details about the target platform and the configuration of the FPGA modules are presented.

4.1.1 Evaluation Dataset

There are no standard datasets for the evaluation of TRN algorithms, so a custom dataset was created for this purpose. First, a lunar terrain map with a size of 11,171×47,903 pixels and a ground sample distance of 1.6m/px was acquired from LRO imagery. The terrain map covers the area surrounding the landing site for the Chang'e 3 moon landing mission. From this terrain map, a dataset consisting of 5,000 query images—each with a resolution of 1024×1024 pixels—was constructed. Some examples of the query images derived from the terrain map are shown in Figure 4.1. A transformation involving random changes to the translation, rotation, scale, illumination, contrast, and Gaussian noise was employed to generate each query image.

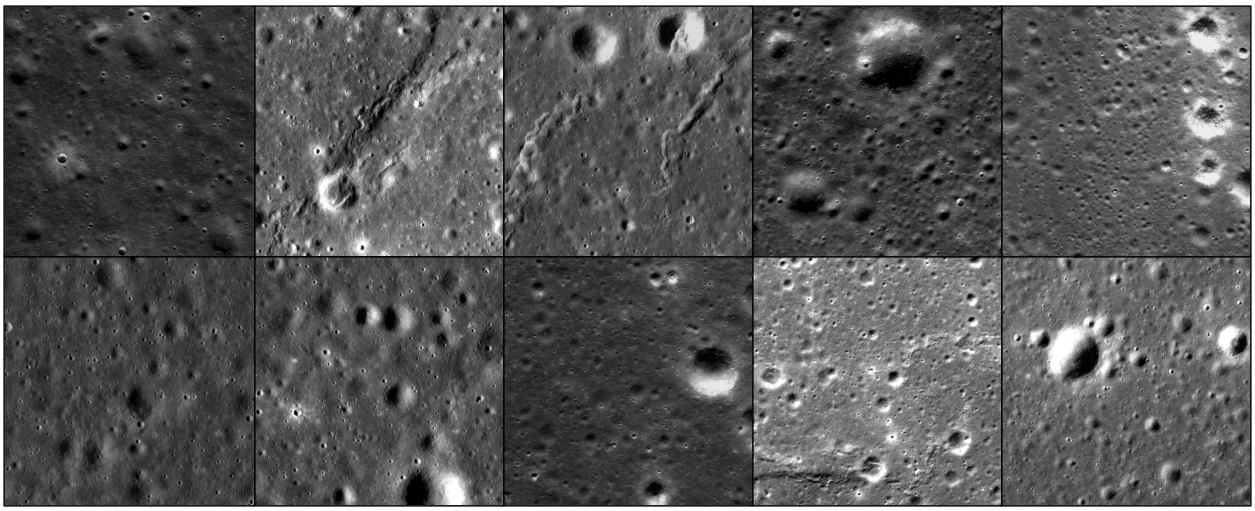


Figure 4.1: Examples of the query images derived from the Chang'e 3 landing site map.

A database consisting of VLAD and SIFT descriptors was derived from the terrain map. More details about the procedure used to generate the query images and the database structure can be found in the Appendix. The evaluation task using this dataset is as follows: For each query image, an estimate for the homography between the query image and the terrain map is computed. Then, this estimate is used to solve for an approximation of the translation, rotation, and scale parameters of the transformation. The approximation is compared to the ground truth value and is considered correct if the translation error is $\leq$5 pixels, the rotation error is $\leq$1.5 degrees, and the scale error is $\leq$0.05.

4.1.2 Test Platform

Since execution time is a critical part of the TRN application, the system should be evaluated using an embedded platform with computational power that resembles what could be used in a real mission. For this reason, the Xilinx ZC706 development board was used as the test platform [23]. This development board features the Xilinx Z-7045 SoC, which is the same device used by the SHREC Space Processor (SSP) [24]. This justifies the use of the ZC706 development board as an appropriate platform for experimentation.

The FPGA modules used to realize the TRN pipeline contain several configurable parameters that affect the performance of the overall system. The CPU and FPGA clock frequencies for the device are set to 800 MHz and 225 MHz, respectively. The SIFT keypoint detector is configured for a maximum resolution of 1024×1024 pixels. The SIFT and VLAD descriptor computation modules both use two PEs to decrease the latency of the feature extraction. For coarse localization, database VLAD descriptors are encoded using 32 subquantizers (leading to a size of 32 bytes per descriptor), and the ANN search module is configured to process a single query and return the ten nearest matches. For fine localization, 16 subquantizers (leading to a size of 16 bytes per descriptor) are used to encode database SIFT descriptors, while the ANN search module is configured to process 16 queries in parallel and return the top two matches for each query.

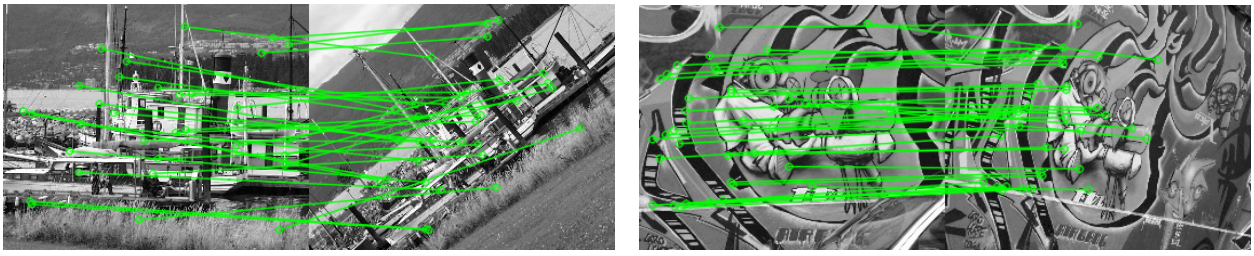
4.2 Results and Discussion

4.2.1 Localization Accuracy

In order to implement SIFT feature extraction on the FPGA, it was necessary to make certain modifications to the algorithm to optimize computation for hardware. For all other FPGA accelerators, their outputs were exactly the same as in software. As a result, it was necessary to analyze the SIFT accelerator by itself to ensure that the modifications performed did not compromise the quality of the resulting features. In the context of feature extraction, the simplest way to evaluate the quality of the resulting descriptors is to perform an image-matching task. For this task, a pair of images with a known homography is used to test the ability to detect feature points that can be identified across multiple scenes.

For this research, the popular *Affine Covariant Features* dataset [25] was used. Examples of the feature-matching results are shown in Figure 4.2b. Features were extracted from each image pair and matched through a nearest-neighbor search. Then, the known homography was used to compute the number of correct matches. The SIFT implementation found in OpenCV version 3.4 was used as a software baseline. Results of the evaluation are shown in

Table 4.1. From this data, it can be observed that the SIFT accelerator performs similarly to the software baseline. This validates the quality of the SIFT features produced by the proposed hardware implementation.



(a) (b)

Figure 4.2: Feature matching results using the proposed SIFT accelerator on images from the (a) Boat and (b) Graffiti sequences in the Oxford dataset. Only a small number of matches are drawn to reduce clutter.

After the validating the functionality of the SIFT FPGA accelerator, the entire TRN pipeline was evaluated using the lunar imagery dataset. For each query, an estimate of the transformation parameters used to generate the image was computed using local feature correspondences obtained from the TRN pipeline. Some examples of correctly localized images are shown in Figure 4.3. From this figure, it can be observed that queries with a wide range of scales, rotations, positions, and illumination conditions can be correctly localized. For each image, a subset of the local feature matches demonstrate that the system is somewhat robust to outlier matches. For the image highlighted with an orange frame (top right), there are several erroneous matches with a map tile that represents an area not actually visible in the query image. However, there were still enough correct correspondences to compute an accurate homography.

The coarse localization step is the most crucial stage of the pipeline, as it is impossible to establish true correspondences between the query and database features if there are no relevant tiles retrieved. One of the free parameters in the pipeline is the number of retrieved tiles used in the fine localization step. The localization performance of the system was

Table 4.1: Results for SIFT descriptor quality evaluation.

Sequence	**Implementation**	**# of Matches**	**# of Correct Matches**	**Matching Rate**
Boat	Software	545	518	95.05%
	Hardware	411	375	91.24%
Graffiti	Software	248	151	60.89%
	Hardware	143	76	53.15%
UBC	Software	584	561	96.06%
	Hardware	699	676	96.71%
Trees	Software	539	438	81.26%
	Hardware	702	581	82.76%
Leuven	Software	294	262	89.12%
	Hardware	331	299	90.33%

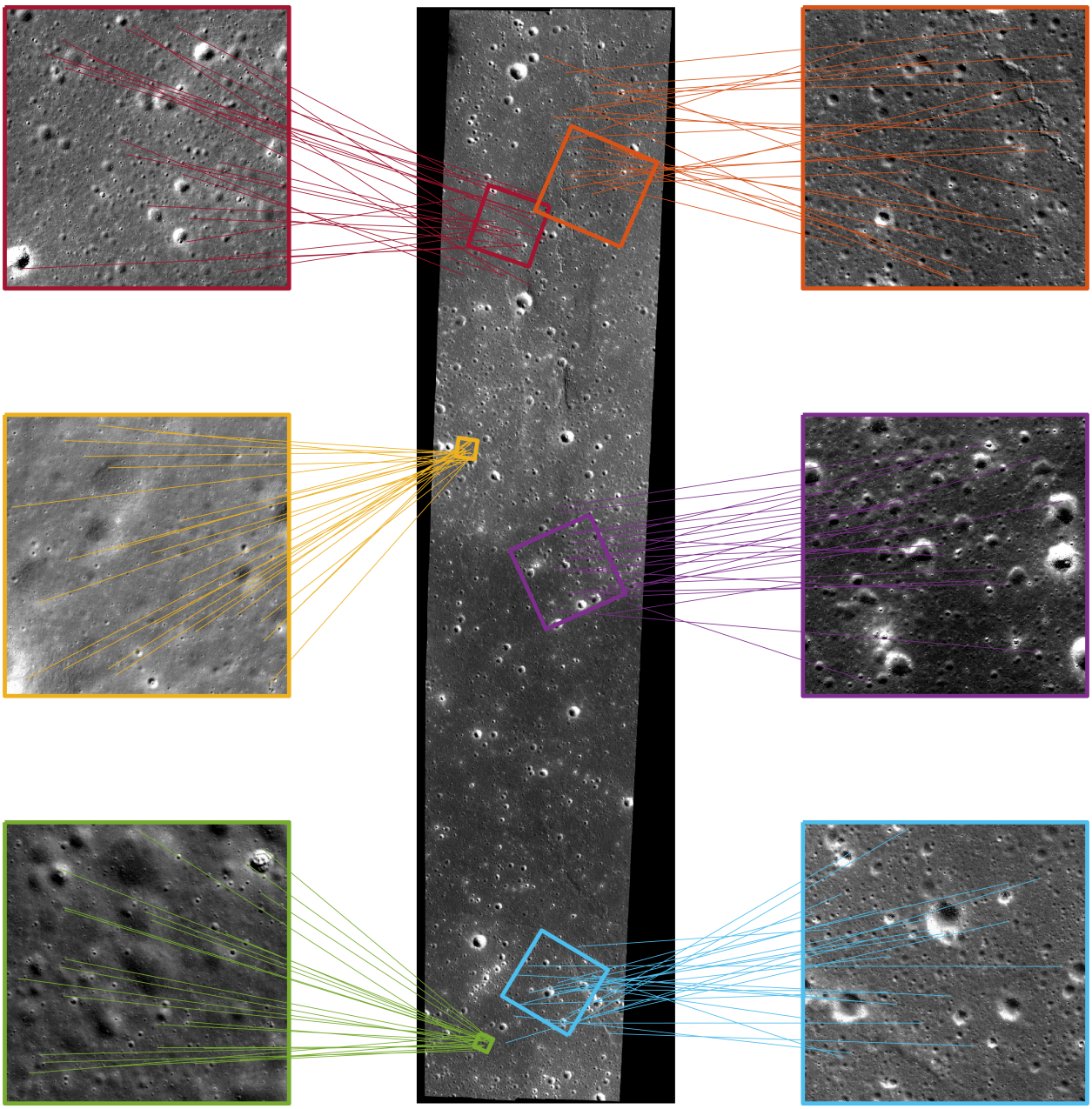


Figure 4.3: Examples of correctly localized query images.

evaluated for a varying number of retrieved tiles as summarized in Figure 4.4. From this data, a few insights about the effect of retrieving a higher number of tiles can be obtained. Firstly, as shown in Figure 4.4a, increasing the number of retrieved tiles leads to a higher percentage of correctly localized queries. When using only the nearest tile match, 77.5% of queries are correctly localized, with performance improving to 92.78% when ten tiles are retrieved. This behavior is explained by the fact that as more tiles are retrieved, there is a higher likelihood that there is at least one relevant item in the set of retrieved tiles. This also introduces more irrelevant tiles, but the SIFT matching step is robust enough to be largely unaffected. As shown in Figure 4.4b, one negative effect of retrieving more tiles is that the number of SIFT database features selected for matching will increase significantly. A median of 1,407 SIFT features are selected when only using the nearest tile match, while the median increases to 9,072 SIFT features when ten tiles are retrieved. This has implications for the execution time of the application, which will be discussed in the next section.

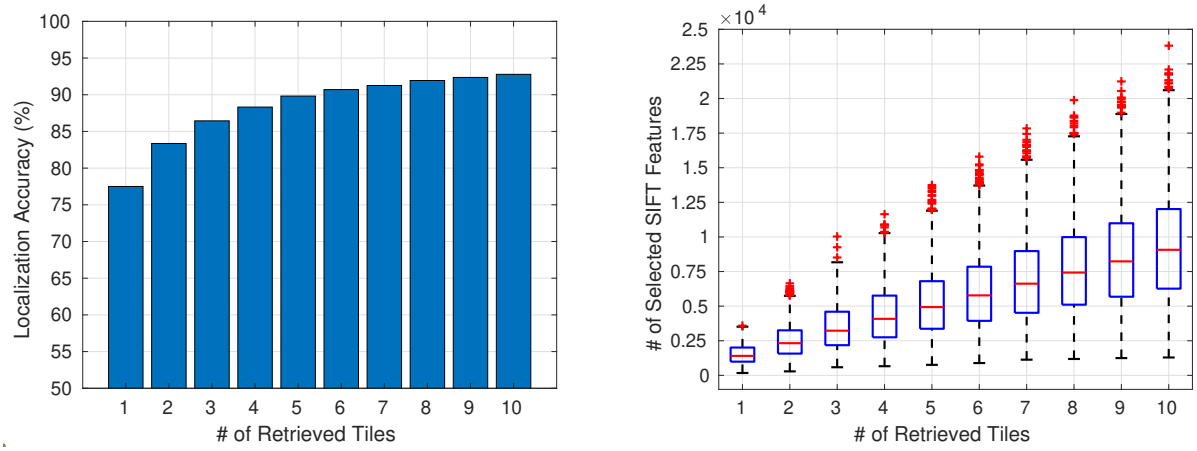
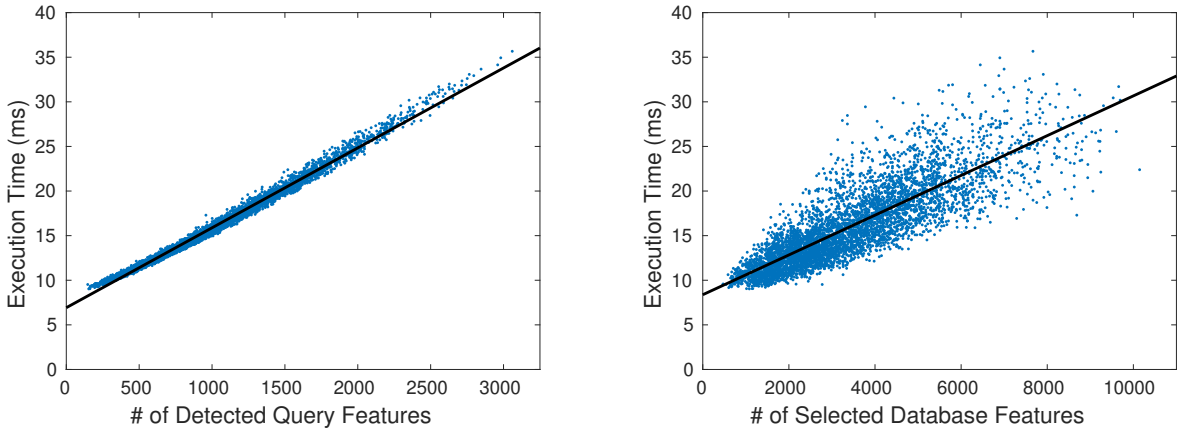


Figure 4.4: Analysis of the relationship between the number of database tiles retrieved and (a) the proportion of correctly localized queries and (b) the number of SIFT features from the database that are selected for matching.

4.2.2 Execution Time

The proposed TRN pipeline was evaluated on the target platform and the execution time to process each query was measured. Since this application is desired to run in real-time, it is important to understand how the execution time of the application varies across many input images. The number of SIFT features in the query and database sets used for matching have the largest effect on execution time and are the only variables that vary considerably across multiple images. Therefore, the relationship between these variables and the execution time was studied. For this experiment, the number of retrieved tiles K was set to four. In Figure 4.5, scatter plots demonstrate that the number of query features detected is the strongest predictor of the overall execution time. In both cases, the general trend is that the execution time increases linearly with the number of features. The R^2 values for the linear models shown in Figures 4.5a and 4.5b were 0.9908 and 0.7012, respectively, indicating a strong correlation between the number of features and execution time. All queries are processed within less than 40 ms, corresponding to a framerate of up to 25 FPS. This processing speed is suitable for real-time operation during a space landing scenario.



(a) (b)

Figure 4.5: Scatter plots illustrating how (a) the number of detected query features and (b) the number of selected database features affect the execution time of the system.

To further evaluate the execution time characteristics of the TRN system, additional tests were performed to explore the relationship between the number of tiles retrieved and the mean execution time. As previously discussed, increasing the number of retrieved tiles will improve the localization accuracy, but the average number of SIFT database features selected will also increase. Thus, it should be expected that increasing the number of retrieved tiles will negatively affect the execution time. In Figure 4.6, execution time results for a varying amount of retrieved tiles are presented. From this data, it can be observed that a higher number of retrieved tiles will result in a larger mean execution time. Setting the number of retrieved tiles to ten instead of one increased the average execution time by approximately 12.91% to 17.57%, depending on the number of detected query features. Overall, it can be concluded that the improved localization accuracy from retrieving a larger number of tiles is beneficial enough to justify the small execution time penalty.

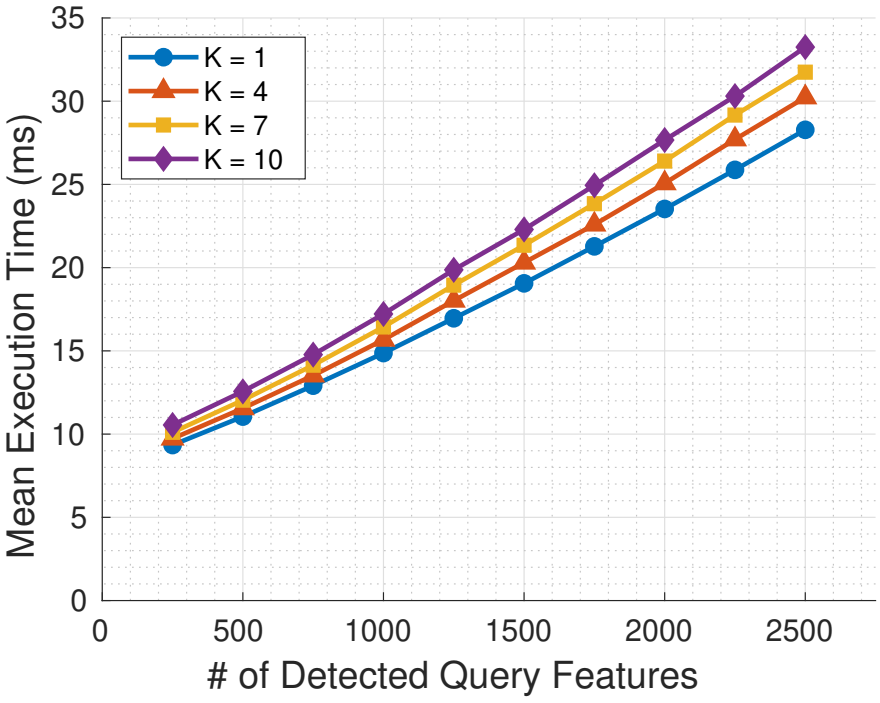


Figure 4.6: Comparison of the average execution time of the system for an increasing number of detected query features based on the number of retrieved tiles.

The final component of the execution time evaluation was to investigate how much time is required to perform each stage in the pipeline. In Figure 4.7, a breakdown of the mean execution time for each of the pipeline stages is presented. A few insights about the performance of the system can be obtained from this data.

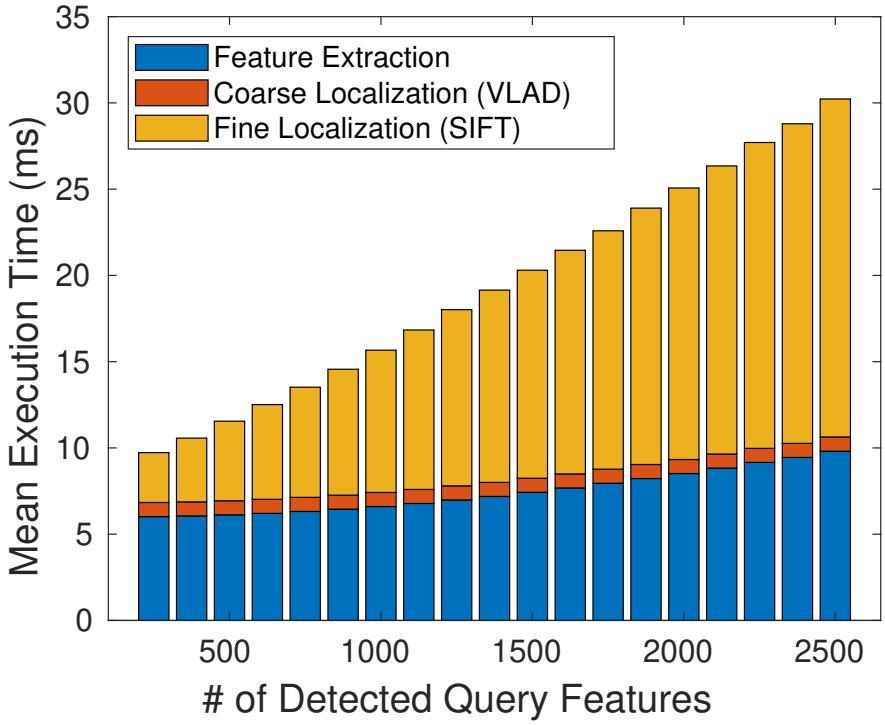


Figure 4.7: Breakdown of the execution time for each stage of the TRN pipeline.

The feature extraction stage consumes a large proportion of the execution time when the number of detected features is small, but this proportion decreases when more features are detected. This can be explained by considering the two components of SIFT feature extraction. The keypoint detection module will require processing the image at a rate of one pixel per cycle, so there will be a fixed execution time cost even when few features are detected. The descriptor computation module consumes additional cycles based on how many features are detected, but increases slowly due to the optimizations presented in the FPGA architecture discussion.

For the coarse localization stage, the execution time is constant regardless of the number of features and is a small proportion of the overall execution time. This behavior can be explained by considering that the coarse localization step consists of a PCA dimensionality reduction of the query VLAD descriptor followed by an ANN search across the entire VLAD database. Both of these steps require a fixed amount of operations, so there is little variability in the execution time. Additionally, the ANN search is performed very quickly (as it only takes four cycles to compute each vector distance), so the execution time is small even if the database has a large number of entries.

It was found that the fine localization stage dominates the execution time as the number of detected query features increase. The ANN search used for matching SIFT features is performed by processing 16 query vectors at a time. Therefore, more detected features will increase the number of iterations linearly. In addition, a high number of detected query features requires a larger amount of CPU processing to separate the query features by their corresponding cluster. These factors result in the fine localization step consuming the most amount of time when the number of detected features is large. Overall, optimizing this stage of the pipeline is the most promising strategy for improving the execution time of the system.

4.2.3 Resource Utilization

The resource-utilization figures for each of the sub-components of the TRN system are shown in Table 4.2. These values were obtained using post-implementation reports from the Vivado design tools. Here, the data movement network consists of the two DMA controllers as well as the interconnect used to enable communication between the processor system and the various accelerators through an AXI4-Lite interface. Overall, the system consumes less than a third of the available resources, which allows for flexibility in the implementation. The low resource utilization enables the incorporation of additional processing steps on the FPGA, if required by the application. Alternatively, additional PEs can be instantiated in the feature extraction and fine localization stages to speed up computation at the cost of increased resource utilization. Finally, the number of processing elements can be scaled down at the cost of increased latency to accommodate for resource-contrained devices.

Table 4.2: TRN system resource utilization on Zynq-7045 SoC.

Subsystem	**LUT**	**FF**	**BRAM18k**	**DSP**
Data Movement Network	10,436	13,889	44	0
SIFT Keypoint Detection	5,385	5,733	23	46
SIFT Descriptor Computation	14,572	14,908	84	12
VLAD Descriptor Computation	5,264	5,669	11	15
PCA Dimensionality Reduction	497	260	1	8
PQ ANN Search (VLAD)	2,065	1,749	9	0
PQ ANN Search (SIFT)	20,152	13,958	129	0
Total Utilization	58,371	56,166	301	81
(% of Available Resources)	26.70%	12.85%	27.61%	9.00%

5.0 Conclusions and Future Work

This research consisted of the design and evaluation of a new TRN system based on a hierarchical localization approach. The proposed system draws inspiration from successes in terrestrial autonomous navigation applications and is a promising solution for future space missions. The feature-based representation of the terrain map based on robust SIFT features enables localization without the need for a reliable initial estimate of the position. Using global image descriptors for a hierarchical search allows for an efficient matching of features obtained from a camera to the terrain map database. In order to meet the execution time requirements of the application under low SWaP-C constraints, the proposed TRN system was realized in a heterogeneous CPU+FPGA architecture. The Xilinx ZC706 development board was chosen for testing as it features the Zynq-7045 SoC–a device is also present in the SHREC Space Processor. By designing FPGA accelerators for each stage of the pipeline, the execution time was reduced to an acceptable level for real-time operation.

To evaluate the performance of the system, a localization dataset was constructed by using terrain maps of the moon obtained from a real space mission. Using the proposed approach, it was possible to create an efficient representation for a terrain map covering over 800 squared kilometers. An extensive evaluation of the characteristics of the TRN system was conducted. The translation, scale, and rotation transformation parameters of up to 92.78% of the images in the dataset were correctly estimated, independent of a prior position estimate. All queries were processed within 40 ms due to the implementation of large-scale search algorithms and FPGA modules for acceleration of each of the pipeline stages. The proposed TRN system consumes less than a third of the available resources within the target platform. A tradeoff between resource utilization and execution time can be performed depending on the needs of the application.

There are several steps to extend this research and improve the capabilities of the proposed system. Firstly, additional evaluation using more realistic datasets will aid in understanding the capabilities of the proposed TRN system. Images used for evaluation in this study consisted of real lunar imagery, but did not use a model of the cameras that may be

available during space missions. Using images from a simulated descent will improve the quality of the evaluation dataset. In addition, replacing SIFT and VLAD descriptors with alternative feature representations—such as deep-learned features—is a promising direction to improve robustness. An evaluation of the tradeoffs between execution time, memory consumption, and localization accuracy when using these feature representations should be studied. Finally, integration of the proposed TRN algorithm into a guidance, navigation, and control (GNC) system will validate the utility of the new TRN algorithm as a tool to enable more precise landings in future missions.

Appendix Dataset and Database Generation Procedures

Details about the procedure used to generate the dataset and database used in the experiments are provided in this section. A dataset of 5,000 query images with a resolution of 1024×1024 pixels was used as an evaluation dataset for the TRN system. Constructing each query image consisted of two steps. First, a transformation consisting of applying a random set of changes to the position, scale, and rotation of the query relative to the terrain map were performed. Then, the illumination was modified by changing the brightness and contrast of the image, followed by the addition of Gaussian noise. For each query, the transformation parameters were determined by selecting values from a random, uniform distribution. In Table A1, the minimum and maximum values for each of the transformation parameters is listed. The scale parameters were chosen to cover a wide range of altitudes while also considering the degradation of rendering quality based on the ground sample distance of the map. The luminance, contrast, and noise parameters were chosen experimentally to provide a reasonable amount of distortion to the images.

Table A1: Transformation parameters used to generate dataset query images.

Parameter	**Minimum**	**Maximum**
Luminance	$+20$	-20
Contrast	-25%	$+25\%$
Gaussian Noise	$\sigma^2 = 0$	$\sigma^2 = 10$
Scale	$\times 0.25$	$\times 2$
Rotation	-180°	$+180^\circ$

In Figure A1, a visualization of the terrain map and the location of all database queries is shown. Due to its large aspect ratio, the map is split in half for better visualization. Queries correctly localized by the TRN system are shown in green; incorrectly localized queries are shown in red.

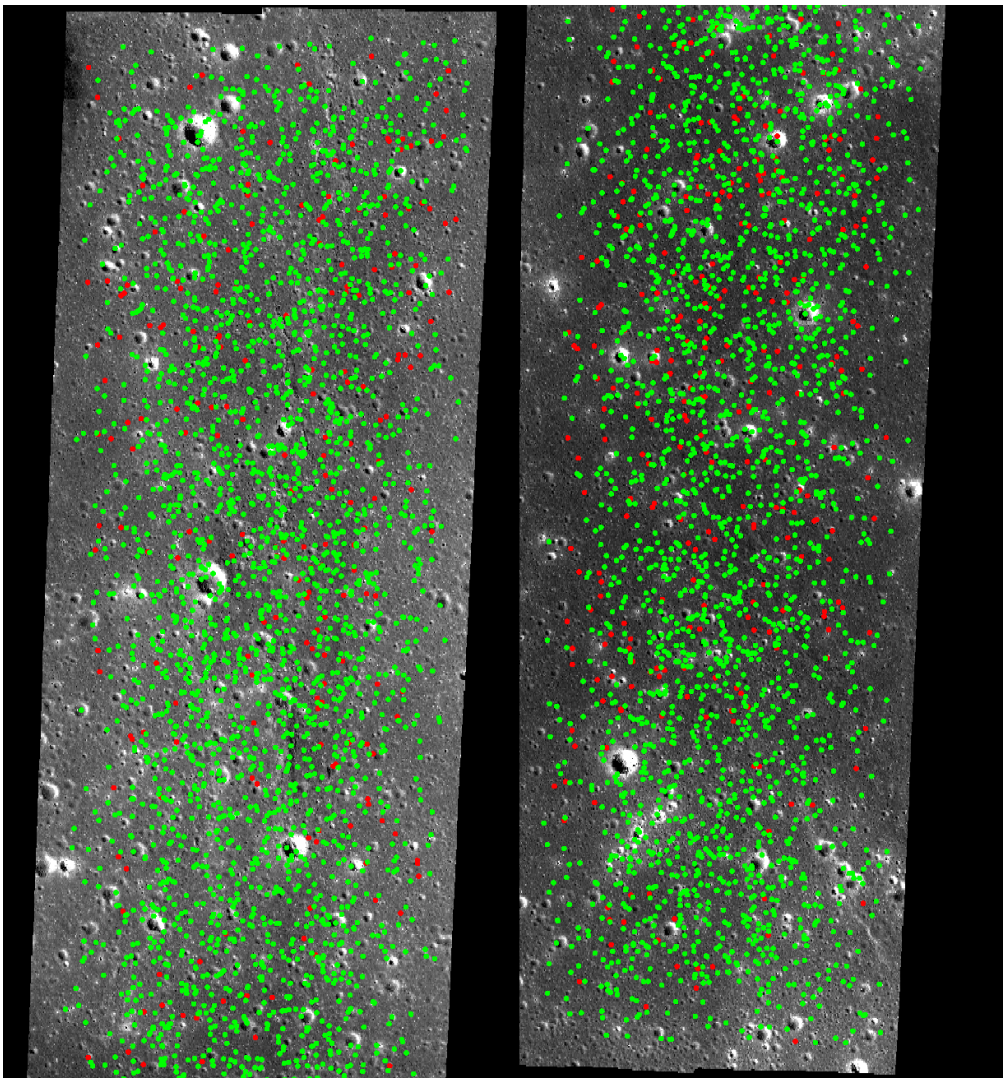


Figure A1: Visualization of the location of database queries with respect to the terrain map.

Generating the terrain map database consisted of the following procedure. First, SIFT features were extracted from the terrain map, which were used to train the VLAD codebook and SIFT PQ codebook. Then, the terrain map was divided into a uniform grid of squares, where each square has a side length of 1024/3 pixels. A set of tiles was constructed where each tile corresponds to a region of 3×3 squares in size. Tiles containing 150 or more SIFT features were selected as database tiles.

A VLAD descriptor was computed for each database tile, followed by a PCA dimensionality reduction. This set of VLAD descriptors was used to train the VLAD PQ codebook, which was subsequently used to compress the data. To increase robustness to altitude changes, the map was resized by several scaling factors and the entire process was repeated at each scale. Overall, a total of 9 map scales were computed, leading to a database size of around 30,000 VLAD descriptors and over 2 million SIFT features.

One concern with using such a large dataset in an embedded application is memory consumption, but this is mostly addressed by applying PQ to the data. By compressing the VLAD descriptors with PCA and PQ, the size of the VLAD database was reduced from ≈67.4 MB to ≈1.1 MB, corresponding to a 64× reduction in size. The SIFT descriptors are compressed using PQ only, which reduced their size from ≈388.3 MB to ≈47.1 MB, corresponding to an 8× reduction in size. The costs to store database SIFT feature coordinates, codebooks, and additional overhead for the database structure increase the overall memory consumption of the system to a total of ≈73.4 MB. This amount is fairly manageable for the target device used in this study, which contains 1 GB of DDR3 memory.

Bibliography

[1] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johnson, A. Ansar, and L. Matthies, "Vision-aided inertial navigation for spacecraft entry, descent, and landing," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 264–280, 2009.

[2] A. Johnson, S. Aaron, J. Chang, Y. Cheng, J. Montgomery, S. Mohan, S. Schroeder, B. Tweddle, N. Trawny, and J. Zheng, "the Lander Vision System for Mars 2020 Entry Descent and Landing," *Guidance, Navigation and Control 2017: Proceedings of the 40th Annual AAS Rocky Mountain Section Guidance and Control Conference*, pp. 143–158, 2017.

[3] A. E. Johnson and J. F. Montgomery, "Overview of Terrain Relative Navigation approaches for precise lunar landing," *IEEE Aerospace Conference Proceedings*, 2008.

[4] A. Johnson, R. Willson, Y. Cheng, J. Goguen, C. Leger, M. Sanmartin, and L. Matthies, "Design through operation of an image-based velocity estimation system for mars landing," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 319–341, 2007.

[5] A. E. Johnson, Y. Cheng, J. Montgomery, N. Trawny, B. Tweddle, and J. Zheng, "Real-time terrain relative navigation test results from a relevant environment for mars landing," *AIAA Guidance, Navigation, and Control Conference, 2013*, pp. 1–13, 2015.

[6] D. A. Lorenz, R. Olds, A. May, C. Mario, M. E. Perry, E. E. Palmer, and M. Daly, "Lessons learned from OSIRIS-REx autonomous navigation using natural feature tracking," *IEEE Aerospace Conference Proceedings*, 2017.

[7] J. A. Christian, H. Derksen, and R. Watkins, "Lunar crater identification in digital images," 2020.

[8] J. S. McCabe and K. J. Demars, "Anonymous feature processing for efficient onboard navigation," *AIAA Scitech 2020 Forum*, vol. 1 PartF, no. January, pp. 1–20, 2020.

[9] C. Owens, K. Macdonald, J. Hardy, R. Lindsay, M. Redfield, M. Bloom, E. Bailey, Y. Cheng, D. Clouse, C. Y. Villalpando, A. Hambardzumyan, A. E. Johnson, and A. D. Horchler, "Development of a signature-based terrain relative navigation system for precision landing," *AIAA Scitech 2021 Forum*, no. January, pp. 1–20, 2021.

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

[11] D. M. Chen, G. Baatz, K. Köser, S. S. Tsai, R. Vedantham, T. Pylvänäinen, K. Roimela, X. Chen, J. Bach, M. Pollefeys, B. Girod, and R. Grzeszczuk, "City-scale landmark identification on mobile devices," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 737–744, 2011.

[12] T. Sattler, W. Maddern, A. Torii, J. Sivic, T. Pajdla, M. Pollefeys, and M. Okutomi, "Benchmarking 6DOF Urban Visual Localization in Changing Conditions," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8601–8610, 2018.

[13] P. E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk, "From coarse to fine: Robust hierarchical localization at large scale," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 12708–12717, 2019.

[14] R. Arandjelović and A. Zisserman, "Three things everyone should know to improve object retrieval," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2911–2918, 2012.

[15] F. Perronnin and C. Dance, "Fisher kernels on visual vocabularies for image categorization," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[16] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the Fisher kernel for large-scale image classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6314 LNCS, no. PART 4, pp. 143–156, 2010.

[17] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[18] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3304–3311, 2010.

[19] R. Arandjelovic and A. Zisserman, "All about VLAD," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1578–1585, 2013.

[20] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[21] Xilinx, *CORDIC v6.0*, December 2020. Xilinx LogiCORE IP Product Guide (PG105).

[22] Xilinx, *Floating-Point Operator v7.1*, February 2021. Xilinx LogiCORE IP Product Guide (PG060).

[23] Xilinx, *Zynq-7000 All Programmable SoC Technical Reference Manual*, December 2017. Xilinx User Guide (UG585).

[24] N. Perryman, T. Schwarz, T. Cook, S. Roffe, A. Gillette, E. Gretok, T. Garrett, S. Sabogal, A. George, and R. Lopez, "Stp-h7-caspr : A transition from mission concept to launch," *Proceedings of the 35th Annual AIAA/USU Conference on Small Satellites*, 2021.

[25] 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, no. 1-2, pp. 43–72, 2005.