FPGA Acceleration for Feature Based Processing Applications

Gooitzen van der Wal, David Zhang, Indu Kandaswamy,
James Marakowitz, Kevin Kaighn, Joe Zhang, Sek Chai,
SRI International, Princeton, NJ

Abstract

Feature based vision applications rely on highly efficient extraction and analysis of features from images to reach satisfactory levels of performance and latency. In this paper, we describe the implementation of an algorithm that combines distributed feature detector (D-HCD) with a rotational invariant feature descriptor (R-HOG). Based on an algorithmic comparison with other feature detectors and descriptors, we show that our algorithms have the lowest error rate for 3D aerial scene matching. We present implementation on a low-cost Zynq FPGA that achieves 15x speedup, 5x reduction in latency over a quad core CPU. Our results show the considerable promise of our proposed implementation for fast and efficient robotic and aerial drone / UAV applications.

1. Introduction

Embedded platforms that are powered by computer vision capabilities are becoming prevalent. In these systems, size, weight, power, and computational latency are key constraints, especially for small battery powered devices as smartphone, robots, and aerial drones. They use image features to detect and track objects, maintain camera pose and estimate motion, and classify and recognize objects – and thus, feature extraction and analysis must be efficient and effective.

In this paper, we propose an embedded algorithm for feature based processing that is optimized for multi-modal sensor (e.g. LWIR to Visible) alignment, 3D aerial scene matching, and robust tracking of objects. Our algorithm combines distributed feature detector (D-HCD) with a rotational invariant feature descriptor (R-HOG) followed by exhaustive-search matching to better process aerial imagery. As shown in Figure 1, aerial imagery for persistent surveillance often has sparse features that rotate in the scene. Our algorithm leverages these characteristics to arrive at an efficient descriptor impervious to the camera motion on aerial drones. We further improve robustness with distributed sampling of features across the aerial camera view. In a detail comparison of algorithmic performance against other feature descriptors, we show that our approach has the highest algorithmic performance.

This paper also provides details and results of an FPGA-based implementation of our algorithm with embedded sensor/processor board that can be mounted on aerial drones. On the Zynq FPGA, we show acceleration of key computations in the FPGA fabric with ARM software support for serial elements. We aim to enable more capable vision applications by improving computation latency and processing rate for on board aerial drone processing.

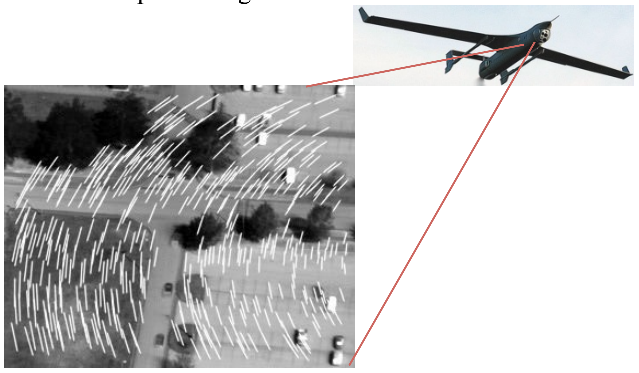

Figure 1. Feature Based Analysis using proposed distributed feature detector and rotational invariant descriptors for on-board UAV processing.

We make the following contributions in this paper:

- A feature based vision algorithm for spare feature and rotational invariance characteristics (D-HCD and R-HOG).
- A FPGA implementation on Zynq platform, with service-based API for flexibility in using the hardware accelerators: Harris Corner Detector (HCD), Histogram of Oriented Gradient (HOG), and full-search symmetric-descriptor matching.
- A small-embedded sensor-processor board with integrated sensors and cameras for on board processing on aerial drones.

This paper represents a complete body of work for our proposed algorithm as we describe the research from algorithm analysis, to FPGA implementation, to embedded hardware design. We provide detailed comparisons for the design choices with experimentally measured results from our physical implementation. We advocate this level of depth in embedded vision design as there are many tradeoffs between algorithmic and hardware performance.

The rest of this paper is organized as follows: Section 2 describes proposed algorithm with brief descriptions of a selected group of feature detectors and descriptors. We provide our evaluations of the algorithms on analysis of

3D tracking using aerial videos. In Section 3, we describe the FPGA implementation of the algorithm, with details on the data flow and service-based API. Section 4 provides embedded sensor-processor hardware and analysis of implementation results. We offer a summary of our work in Section 5, with insights for future work.

2. Feature based processing

We consider a number of feature detectors and descriptors. Feature detectors (e.g. HCD, STAR, FAST, SIFT, SURF, ORB, BRISK, MSER, GITT [1,2]) are developed to select localized salient features from 2D images. Feature descriptors (e.g. HOG, BRIEF, FREAK) describe the selected features as representative feature vectors. Some feature detectors, such as SIFT, SURF and ORB are also feature descriptors.

In this section, we provide our evaluations to these feature based algorithms based on 3D reconstruction analysis of a set of aerial videos.[3] Performance of most feature functions is based on the OpenCV library (single-threaded, quadcore, IPP and MKL support[4]) running on a 2.8GHz CPU.

2.1. Feature detection algorithms

Performance evaluations of the selected feature detectors are listed in Table 1 as average processing time per frame. The input sources are multiple 640x480 aerial videos with 580 frames. Figure 1 illustrates an example aerial imagery that is processed. The maximum number of features per frame is set to be 2000.

Table 1. Selected feature detectors for evaluation

Detector	Descriptions	Avg Time(s)
SIFT	Local multi-scale level features, histogram of weighted gradient locations and orientations in blocks; Slow but robust.	0.172
FAST	Corner feature encoded by the contrast of the circle of surrounded pixels. Fast but sensitive to noise.	0.015
STAR	A variant of CenSurE detector, is a center-surround extrema by bi-level LoG Operator and Harris measure.	0.016
SURF	Approximate SIFT, integral of images and determinants of Hessian matrix are used for detection of key points.	0.265
ORB	The combination of oriented FAST and rotated BRIEF features. Fast and efficient alternative of SIFT.	0.045
HCD	Local corner detector based on the determinant of Harris matrix.	0.030
D-HCD	HCD features, computed in tiles and evenly distributed.	0.032

SIFT and its variant SURF are stable but slow. Since they are multi-scale (either subsampled or of full spatial resolutions), higher latency and more buffer allocation in hardware is less favorable. FAST and STAR are speedy detectors, but less features detected and sensitive to noise is a disadvantage. HCD is a relative speedy detector. It has stable performance on a pre-filtered or subsampled image, and is relatively easy to implement in hardware. We found that using HCD, modified to select a well distributed set of features across the image works well in conjunction with the feature descriptors we tested. This evenly distributed HCD, a variant of the original HCD detector, is called the D-HCD in this paper. Its performance is listed in the last row of Table 1.

D-HCD first finds all features with strength above a threshold, and then divides the image into tiles (e.g. 10x10), and selects the best N features in each tile, with maximum number of features of N x #tiles (< maximum number of features). The best N features in each tile may not be the highest salient features in the entire image, but well distributed and strong enough to represent spatial information of the scene.

2.2. Feature description algorithms

We have used the same video sequences to evaluate feature descriptors. The descriptor time includes descriptor generation based on the same number of detection points, and descriptor matching. The simulation sets 20% of the image width as the distance threshold in both directions. For example, if 1000 descriptor points are used and each point will match with 40 points, totally 40,000 matches are executed. For symmetric matching, this number is further doubled. Performance evaluations of the selected feature descriptors are listed in Table 2. The average time for each descriptor is the descriptor time based on the same number of feature points per frame from the entire aerial video sequence.

Table 2. Selected feature descriptors for evaluation

Descriptor	Descriptions	Avg Time(s)
SIFT	Local multi-scale level features, histogram of weighted gradient locations and orientations in blocks; Scale and rotation invariant.	0.561
FREAK	A cascade of binary strings computed by comparing image intensities over a retinal sampling pattern. Scale and rotation invariant.	0.181
BRIEF	Binary string descriptor using simple intensity difference tests. Scale invariant.	0.015
SURF	Approximate SIFT, faster and scale and rotation invariant.	0.031
ORB	The oriented BRIEF features; Scale and rotation invariant.	0.083
HOG	Cell based histogram of oriented gradients.	0.068
R-HOG	HOG variant; rotation invariant	0.203

SIFT takes the longest time. FREAK and R-HOG takes less than half of the time. FREAK, BRIEF, SURF and HOG descriptors take the least amount. However, since R-HOG is the OpenCV HOG in addition of finding the peak angle in the histogram of gradient orientation to make it rotation invariant, its source code in C is not optimized as the OpenCV HOG that runs 1/3 of time, but similar when the R-HOG is optimized.

R-HOG first computes the gradients of image at both x and y directions to obtain orientation and magnitude. The gradient magnitude is then multiplied by a weight that is a function of distance from the center point (key point). After smoothing of the histogram it computes and interpolates the dominant gradient angle. With a normalized angle where 0 is the dominant orientation by subtracting the dominance angle, a 128 bin histogram is generated with respect to 16 cells, and 8 gradient orientations. The floating point 128 bin histogram is then clipped and normalized/quantized to a 128-byte unsigned vector.

We did not consider other algorithms, such as optical flow based Kanade–Lucas–Tomasi (KLT) [9]. This is because KLT makes use of spatial intensity information, which is not suitable for multi-modal aerial imagery, and not robust for low light scenarios.

2.3. Performance evaluation

To select a robust feature detector and descriptor for hardware acceleration, we have run simulations of tracking of features on these video sequences using a combination of detectors and descriptors. The tracking algorithm generates temporal feature tracks from each matched point. A minimal valid track must be valid for at least 3 consecutive frames. This is possible because the camera loiters around the interested region due to the persistent surveillance flight patterns. The number of tracks is an index of the robustness of the feature tracking. Another one is the reprojection error [5]. After bundle adjustment, the distance between the mapped 3D points from the same track would reflect the reprojected error. The last index is the alignment of the warped feature points from all images with respect to a reference image. There are cases that the reprojection error is small enough, but the warped points slowly drift.

Table 3 lists the tracking results of a few combinations of feature functions. MSER-HOG 3D tracking has high projection error (4.22) and its 2D alignments of key points are *bad*. HCD-BRIEF has low reprojection error. However, since its track numbers are far less, the alignments have shown *drift* over time. Only SURF-SURF, SIFT-SIFT and D-HCD-R-HOG have good track numbers, less reprojection errors and *good* performance. D-HCD takes 15ms and selects 1280 points while SIFT takes 124ms on 2350 points.

SURF is a variant of SIFT, and both need to generate multiple scales or full resolution images. For reduced latency and complexity, D-HCD-R-HOG is a better candidate. Table 4 lists the 3D tracking performance using various descriptors on features detected by D-HCD. The good tracking comes from either R-HOG or SIFT.

Table 3. Evaluation of feature functions based on 3D reconstruction results

Detector (s/pixels)	**Descriptor (s/pixels)**	**L2-norm (s)**	**# of tracks**	**Reproj Error**
SURF	SURF			good
0.124/2350	0.265/2300	0.171	288459	0.79
SIFT	SIFT			good
0.219/1100	0.281/1100	0.063	128972	1.68
STAR	HOG			bad
0.016/130	0.015/130	0.015	46270	1.50
ORB	ORB			bad
0.015/2000	0.024/2000	0.031	233550	0.92
MSER	HOG			bad
0.156/250	0.031/250	0.015	11004	4.22
HCD	BRIEF			drift
0.015/200	0.015/200	0.015	49763	1.11
GFTT	BRIEF			drift
0.031/1200	0.015/1200	0.046	112400	1.03
FAST	HOG			drift
0.015/10000	0.234/10000	2.56	199317	1.10
D-HCD	R-HOG			good
0.015/1280	0.343/1280	0.125	352159	0.78

Table 4. Evaluation of feature descriptors based on 3D reconstruction results

Detector (s/pixels)	**Descriptor (s/pixels)**	**L2-norm (s)**	**# of tracks**	**Reproj Error**
D-HCD	SIFT(s)			good
0.016/1280	0.1/1280	0.085	399989	0.83
D-HCD	ORB			bad
0.016/1280	0.015/1100	0.047	249860	1.15
D-HCD	HOG			drift
0.016/1280	0.055/1280	0.085	198912	1.07
D-HCD	FREAK			bad
0.015/1280	0.078/1180	0.078	229909	1.03
D-HCD	BRIEF			drift
0.015/1280	0.015/1130	0.047	324758	0.697
D-HCD	SURF			bad
0.015/1280	0.031/1280	0.063	367	54
D-HCD	R-HOG			good
0.015/1280	0.343/1280	0.125	352159	0.78

Based on the above analysis, we selected D-HCD as feature detector, and R-HOG as feature descriptor in the FPGA acceleration. SIFT as a descriptor performed well also, but was much more computationally expensive. BRIEF is less complex but may not always work as well.

3. FPGA Acceleration platform

The FPGA provides an excellent acceleration platform, because of the inherent parallelism that can be achieved, access to local memory, and can be reconfigured to the computational need of application. The challenge is to take advantage of this parallelism efficiently and provide the data from memory when needed at the bandwidth required. In addition, the new FPGA SoC architecture, incorporating dual ARM processors and sharing its memory, provides an excellent architecture for efficient acceleration mixing FPGA acceleration fabric and low power CPU based processing.

Figure 2 shows the generic FGPA vision acceleration architecture we use. The base architecture includes the dual ARM processing section, video input, video pre-processing, video crosspoint, filters, multi-port access to the shared DDR port, and a video output driver for HDMI. The Feature processing modules described below were added to the “Vision Devices” with access to video crosspoint and/or DMA ports to DDR[6].

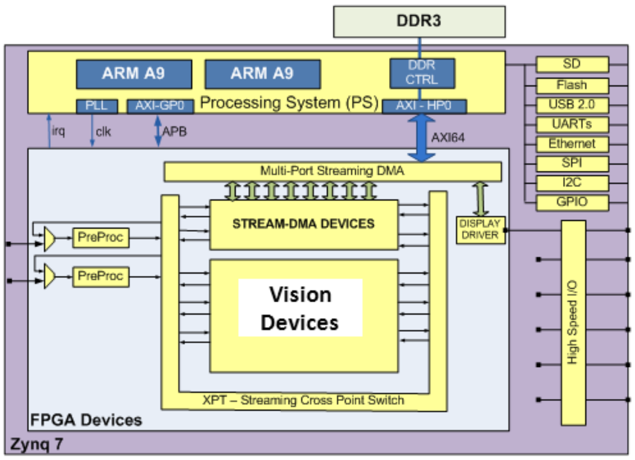

Figure 2. FPGA block diagram showing video devices in the FPGA fabric that accelerates vision algorithms

The hybrid FPGA/ARM architecture also provides the capability to perform integrated algorithm verification, since the full baseline software algorithm can be executed on the ARM.

3.1. Feature processing acceleration

There are a variety of articles on feature processing implementations in FPGAs such as [7,8]. But most of them are related to feature matching, and not the applications mentioned in this paper. The D-HCD, R-HOG feature detector and descriptor and a feature matching function were implemented in a Zynq FPGA to provide acceleration for a range of applications that require large amount of rotation invariant feature matches on a low power platform.

The diagram below shows the sequence of processing steps, and the use of shared memory between FPGA and ARM processing. The video input is processed in-line for image enhancement, filtering (or multi-resolution image generation), and HCD. The HCD also computes an angle/magnitude image that is used by the feature descriptor (HOG). The HOG and MATCH functions perform a sequence of accelerations based on a list of features (Key Points) to process. Both of these types of accelerations require a different data buffering/caching mechanism that enables the efficient computations in the FPGA. In this implementation, the ARM is also used for the HCD tile sorting and Match list generation.

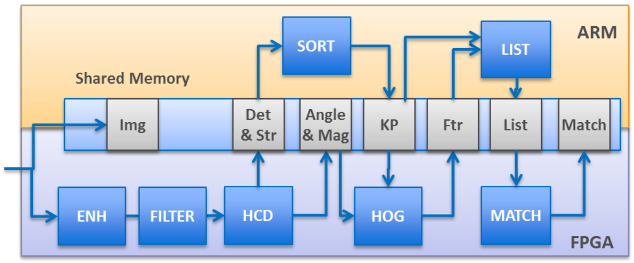

Figure 3. Feature Processing Pipeline. Shared memory facilitates acceleration between software and hardware components.

3.2. Feature Detector

The feature detector described here is a standard Harris Corner Detector (HCD) followed by a tile sort to provide an evenly spaced set of features across the image and provides target for the maximum number of features used for the application.

The HCD takes as input a video stream and compute 3x3 gradients for every pixel in-line. This is followed by the computation of the corner strength for each pixel, and a local maximum suppression function over a 5x5 region, and a threshold function. All the features found and then stored in memory as a list with image location and strength. Note that the 5x5 max-suppress function would also require a 4-line buffer with Strength values stored. Instead of implementing such a large line buffer, we used a 6 line buffer at the input, used multiple gradient, and corner strength computations, to provide a 5-line strength stream to the 5x5 max detector.

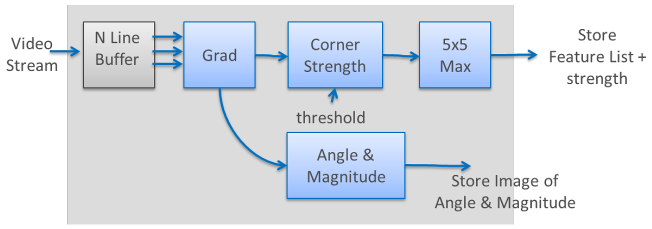

Figure 4. Functional diagram of the HCD feature detector and Angle & Magnitude generator for the feature descriptor.

For our current implementation, the tile sorting is done on the embedded ARM without significant loss of time, and can be implemented in-line with the HCD output.

Since the Feature Descriptor requires the Angle and Magnitude for the feature region, it is most efficient to compute this from the output of the gradient filters and store the data in memory for easy retrieval by the feature descriptor. It therefor does not require any additional processing time for the application.

3.3. Feature Descriptor

The feature descriptor implemented is a rotation invariant HoG (Histogram of Gradients). The rotation invariance is achieved by first computing an angle histogram around a selected feature, determining the dominant angle. The feature histogram that follows is then normalized for the dominant angle. In this implementation we used a 27x27 window to determine the dominant angle, and we use a 45x45 window for computing the feature vector. These sizes are configurable. The feature vector in our case is defined as an angle of 8 bins for 4x4 cells around the feature location, generating a vector descriptor of size 128. The feature vector is then normalized to 8 bit.

Since the feature descriptor is computed only for the feature list from the detector, the limiting factor is accessing the data needed to compute each feature vector. We optimized this by using an Angle and Magnitude buffer, which is then accessed by the histogram computations. The buffer implemented is fast enough to implement four parallel angle and feature histogram modules with a single buffer. For large images, the processing can be performed in vertical image strips.

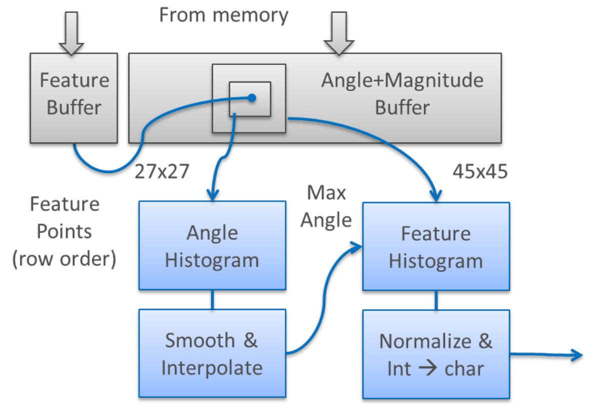

Figure 5. Feature Descriptor diagram. The feature descriptor is normalized for the dominant angle computed by the angle histogram. The buffer optimizes DRAM access and can support four sets of angle and feature histogram modules (4x speed up).

3.4. Feature Matching

The feature matching uses L2-norm, with symmetric matching, meaning that if the feature A in the reference image has a best match to feature B in the inspection image, then the best match for feature B in the inspection image shall be feature A.

The amount of feature matches to be computed can grow rapidly when the number of keypoints and match window increase. For large match sets, the FPGA implementation provides a very significant increase in processing speed. The example for R-HOG in Section 2 requires about 100,000 matches per frame. This processing speed is mostly limited by data access, and so is optimized using efficient pre-fetch. The match sequence is either predetermined for optimum performance, or can be implemented as part of the matching module by providing the keypoint list for each into the data buffer with a selected search window size.

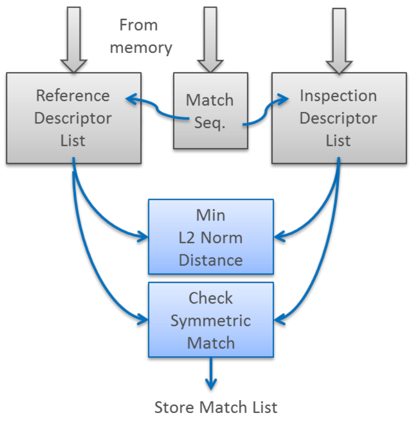

Figure 6. Feature Matching module, using buffers for the descriptor list and match sequence to provide data fast enough to the matching computations.

3.5. Software architecture

A key to rapid development of vision applications is the leveraging of a software stack to develop robust real-time applications. We have implemented a Vision Service Architecture for efficient low-latency and/or highly parallelized implementation of vision algorithm acceleration. A vision service is a C++ class that implements a defined image-process function. A Vision Service I/O is an image buffer in shared memory, and use one or more video devices to accelerate its function.

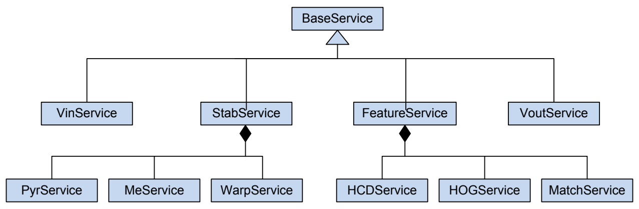

Figure 7. Vision software stack with service based API

The main objective of a vision service is to provide reusable building blocks for rapid development of image applications using the common infrastructure shown in Figure 7 above. Vision services are parameterized to easily support different requirements. For example, without changing code, a vision service can support different image sizes and formats by setting its parameters properly. Parameters may be set via XML configuration files at initialization or API functions at run-time.

As shown in Figure 7, all vision services are derived from the BaseService and must implement the common set of API functions defined therein. A vision service can enclose other vision services. For example the StabService in Figure 7 is only a container class providing Stabilization interfaces; the actual work is done by PyrService (generates image pyramids), MeService (estimate motion), and WarpService (warps image). For the Feature acceleration we implemented FeatureService.

Vision services provide for flexibility. For applications that have dedicated video devices for all functions, vision services can be set to keep the obtained video devices for the entire time. Some applications share video devices between functions. In this case vision services can be set to obtain and release devices when needed.

In addition to flexibility, the services architecture provides for optimized streaming based processing, minimizing memory accesses, parallelization of video functions, and minimization of latency with inline buffers. More details are provided in [6].

4. Results and application

The feature processing was implemented and tested on a Xilinx Zynq platform, with the FPGA clock at 140 MHz, and the ARM clock at 866 MHz. The performance for this implementation can achieve 30 Hz throughput for 1080p images, > 1000 feature points, and > 100,000 feature matches per frame with less than 3 frames latency

In software implementation on a high-end quad-core laptop (Intel core i7, 2.8GHz. 8 cores, ~100W) listed in Section 2, achieved about 2Hz for similar parameters using the D-HCD and R-HOG (Table 4). So the FPGA provides a speedup of 15x and 5x reduction in latency using a Z7020 running at < 4W. Latency can be further optimized.

The FPGA utilization for the 7020 and 7045 are shown in Table 5 below. The "Infrastructure" represents the FPGA acceleration framework shown in Figure 2 above.

Table 5. FPGA utilization

Zynq		FF	LUTS	DSPs	RAMB16
7020	Feature Modules	21%	45%	64%	54%
	+ Infrastructure	33%	71%	72%	72%
7045	Feature Modules	5%	11%	16%	14%
	+ Infrastructure	8%	17%	18%	19%

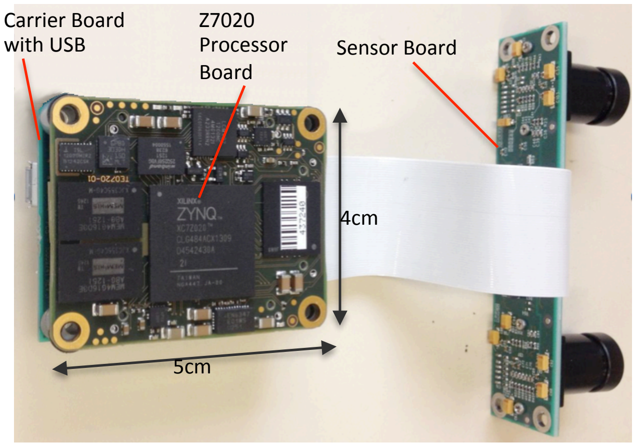

Figure 8. System Block diagram showing camera board stack and wireless/wired interface to smartphone device

We build a complete sensor and processor system, leveraging a commercial Zynq Z7020 board (4 x 5 cm, < 1 lbs) from Trenz Electronic. Our system also includes a sensor and camera board, and a custom carrier board. The small form factor facilitates mounting on small aerial drones. A smartphone or tablet can be connected wirelessly to visualize processed outputs on the ground. We have tested the system on actual flight data.

5. Conclusion

An efficient FPGA implementation of an algorithm (D-HCD, R-HOG, matching) for feature-based analysis is presented. The approach was found to perform very well for 3D scene tracking, multi-modal image registration (e.g. LWIR to Visible), object tracking in 3D space, camera pose estimation, and 3D structure from motion. The algorithms using distributed sampling of feature points and rotational invariant descriptors compared favorably over a set of common feature detector/descriptor algorithms for registering a large sequence of aerial imagery. Our hardware implementation achieves 15x speedup and 5x reduction in latency over quad core CPU. A service-based software layer provides a flexible programmer API to use the hardware accelerators. A complete hardware system integrating sensor and FPGA processors is also presented. The work demonstrates that our approach and implementation is highly effective and efficient for use in embedded applications. Our architecture is designed, verified, and evaluated over real aerial data.

References

[1] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: a survey", J. Foundations and Trends in Computer Graphics and Vision, Vol3, Issue 3, pp177-280, Jan 2008.

[2] Y. Li, S. Wang, Q. Tian, and X. Ding, "A survey of recent advances in visual feature detection", Neurocomputing, **149**(2015), pp736-751.

[3] http://computer-vision-talks.com/articles/2011-01-04-comparison-of-the-opencv-feature-detection-algorithms/

[4] https://software.intel.com/en-us/articles/signal-processing-usage-for-intel-system-studio-intel-mkl-vs-intel-ipp

[5] http://en.wikipedia.org/wiki/Reprojection_error

[6] E. Gudis, et al. "An embedded vision services framework for heterogeneous accelerators." *Embedded Vision Workshop, CVPR* 2013, pp.598-603.

[7] Jin Zhao, Sichao Zhu, and Xinming Huang, "Real-Time Traffic Sign Detection Using SURF Features on FPGA," IEEE HPEC, 2013

[8] Kosuke Mizuno, Yosuke Terachi, Kenta Takagi, Shintaro Izumi, Hiroshi Kawaguchi and Masahiko Yoshimoto, "Architectural Study of HOG feature Extraction processor for real-time object detection," IEEE Workshop on Signal Processing Systems, 2012

[9] Jianbo Shi and Carlo Tomasi. Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition, pages 593–600, 1994.