8-2014

Software Porting of a 3D Reconstruction Algorithm to Razorcam Embedded System on Chip

Kevin Curtis Gunn
University of Arkansas, Fayetteville

Software Porting of a 3D Reconstruction Algorithm to Razorcam Embedded System on Chip

Software Porting of a 3D Reconstruction Algorithm to
Razorcam Embedded System on Chip

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering

by

Kevin Curtis Gunn
University of Arkansas
Bachelor of Science in Computer Engineering, 2012

August 2014
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Dr. Christophe Bobda
Thesis Director

_____
Dr. David Andrews
Committee Member

_____
Dr. Patrick Parkerson
Committee Member

ABSTRACT

A method is presented to calculate depth information for a UAV navigation system from Keypoints in two consecutive image frames using a monocular camera sensor as input and the OpenCV library. This method was first implemented in software and run on a general-purpose Intel CPU, then ported to the RazorCam Embedded Smart-Camera System and run on an ARM CPU onboard the Xilinx Zynq-7000. The results of performance and accuracy testing of the software implementation are then shown and analyzed, demonstrating a successful port of the software to the RazorCam embedded system on chip that could potentially be used onboard a UAV with tight constraints of size, weight, and power. The potential impacts will be seen through the continuation of this research in the Smart ES lab at University of Arkansas.

ACKNOWLEDGEMENTS

I would first like to thank my mentor and ally, Dr Christophe Bobda, for his immense support and guidance throughout my Graduate studies. I would also like to thank my advisory committee, Dr David Andrews and Dr Patrick Parkerson, for their time, wisdom, and guidance throughout my academic career at the University of Arkansas. I would like to acknowledge as well my fellow members of the Smart Embedded Systems Research Lab, particularly Michael Mefenza and Franck Yonga, for their friendship and invaluable support of my research. The Freshman Engineering Program at University of Arkansas deserves a nod, as well, for the opportunity they awarded me to work in exchange for funding my two years of Graduate studies.

I would like to thank my mother and father for their emotional as well as financial support throughout my entire life. Thanks to my sister, for a lifetime of inspiration, and my brother for his continued love and support. My extended family deserves thanks as well, of course, especially my Oma and Opa who have blessed me with substantial moral as well as financial support throughout my life. I would like to also thank my close friends from Saint Louis and Fayetteville, as well as my peers at the University of Arkansas.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Motivation

Recent developments in civil Unmanned Aerial Vehicles (UAV) are creating new opportunities, which will benefit many fields and applications. For example, small UAVs equipped with high-tech sensors, cameras, and communication devices can improve traffic control, surveillance, inspection of infrastructure such as high-voltage power transmission lines, pipelines, buildings, airports, and railways. UAVs have recently been used for aerial photography, crowd surveillance in concerts and other large events, film productions, documentaries on animals, roof inspections, and appraisal - with incredible results. Using semi-autonomous navigation, UAVs can perform their tasks alone by detecting and avoiding obstacles, even in the presence of a distracted operator. Autonomous navigation of UAVs is one aspect of the research being currently conducted in the Smart Embedded Systems (SmartES) Lab of the University of Arkansas.

1.2 Objective

The goal of this thesis is to investigate existing 3D reconstruction algorithms for their portability to a FPGA embedded system for eventual use onboard a UAV. Upon selecting a proper method, it will be implemented in software, designed to provide the best overall performance and accuracy within the limitations of the openCV framework. The method will first be evaluated in software on PC, and then ported to the RazorCam embedded smart camera system developed in the SmartES lab at the University of Arkansas for further testing and evaluation.

1.3 Approach

The approach consists of an algorithm used to calculate depth information from Keypoints in two consecutive image frames using a monocular camera sensor as input and implemented with the OpenCV library. This method was first implemented in pure software and run on a general-purpose Intel CPU, then ported to the RazorCam Embedded Smart-Camera system on chip and run on an ARM CPU onboard the Xilinx Zynq-7000 System on Chip. This type of system has benefits over existing LiDAR-based systems, which are expensive, heavy, and have high power consumption (which has a negative impact on the battery life and reduces the flight duration of the UAV). Using a monocular camera-based system in combination with a 360-degree lens, we can produce an accurate representation of the world around the camera to base the UAVs navigation behavior on. A 3D representation that will provide the distance to objects in view is preferable to a single 2D object extraction model. Existing 3D models are computationally expensive and difficult to use in embedded systems with tight constraints of size, weight and power (SWAP); this approach will attempt to overcome those constraints.

1.4 Organization of this Thesis

The organization of the rest of the thesis is as follows: Section 2 will go over background information, explaining key concepts and work related to this research. Section 3 discusses the complete approach to achieving the objective outlined in Section 1.2, as well as the methodology involved in different software design and development decisions. Section 4 will present the process and results of an evaluation of the accuracy and performance of the software implementation explained in Section 3. Section 5 will summarize the work and draw

conclusions, informing the reader of potential impacts of this research as well as further work that could potentially improve it in the future, followed by a list of references.

2. BACKGROUND

2.1 Key Concepts

A rudimentary understanding of the following key concepts is crucial to understand the approach presented in section 3.2.

2.1.1 Unmanned Aerial Vehicle (UAV)

A UAV, or unmanned aerial vehicle, is essentially exactly what it sounds like – a flying machine that does not have a human pilot on board. UAVs come in many different shapes and sizes, each tailored for the specific task they are needed to accomplish. For example, there are military UAVs (usually referred to as 'drones', such as the General Atomics MQ-9 Reaper shown in Figure 2) that carry weapons on board and can be flown by a pilot in a base station thousands of miles away. They may also be retrofitted with other accessory devices, such as optical or infrared cameras, ocean color sensors, or Doppler radar, for use in weather forecasting and oceanography (see Figure 3). These drones are highly robust, cutting-edge technology, however research and development continues and is moving towards even more complex, smaller, and increasingly lifelike drones that can hide in plain sight known as 'Micro Drones' (see Figure 4). There are also civilian (or 'civil') UAVs that can look like anything from a thin flying wing to a helicopter with four to eight synchronized rotors. They can be flown manually by remote control (in this case, they are referred to as a MAV – micro air vehicle), or autonomously. For the purposes of this paper, UAV refers to a civil UAV – specifically a quad rotor model, which is very light and that we wish to be flown semi-autonomously (see Figure 1 below).

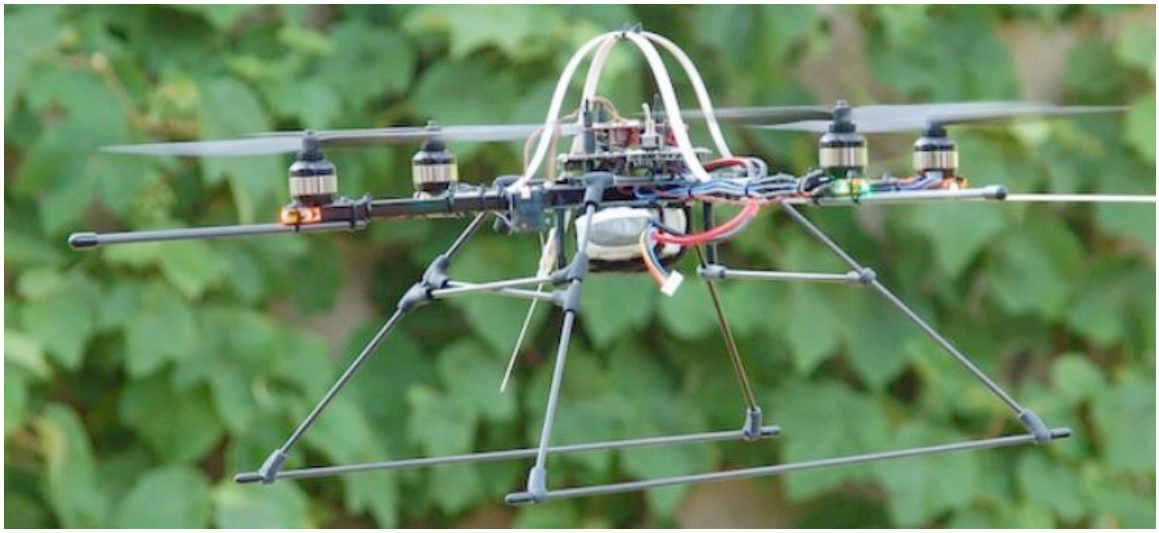

Figure 1: RazorCopter Quad Rotor UAV.

Photo by Christophe Bobda [46], Used with Permission

Figure 2: General Atomics MQ-9 Reaper Drone UAV.

Photo by Sgt. Ricky Best USAF [47], Public Domain

Figure 3: GA MQ-9 NASA Altair Satcom UAV.
Photo by Tom Tschida [48], Public Domain

Figure 4: AeroVironment Nano Hummingbird UAV.
Photo by AeroVironment Inc. [49], Used with Permission

2.1.2 Autonomous / Semi-autonomous Navigation

Autonomous navigation refers to any vehicle or robot that can navigate automatically without any input from an operator. Semi-autonomous navigation is similar, except with limited input from an operator, usually by remote control. These navigation systems are commonly achieved through some use of GPS, LiDAR, monocular or stereo cameras, along with some type of 3D reconstruction algorithm that can collect depth information from the UAV's surrounding environment using these tools.

2.1.3 3D Reconstruction Algorithm

The 3D reconstruction algorithm constructs a map of the surrounding environment by collecting depth information using some form of sensor input (camera, GPS, LiDAR) or other data (Range Image, Google Maps, GIS). The result of these algorithms can vary from being highly detailed to rudimentary, depending on the specific calculations done and quality of the sensor(s).

At the core of any 3D reconstruction algorithm, there is some calculation performed to get a depth value for every object in the field of view. For the purposes of this research, 3D reconstruction refers to a camera-based system that takes, at minimum, two images represented in two dimensions (x, y) and outputs one Range Image containing points associated with objects in three dimensions (x, y, z). A Range Image will be explained more in-depth in (2.1.12). This added depth dimension allows designers to write code allowing the UAV system to recognize objects in its way and plan a path to avoid them – creating essentially a simple semi-autonomous navigation system.

2.1.4 Global Positioning System (GPS)

The Global Positioning System or GPS is a satellite-based navigation system designed to give exact, real-time location coordinates to any device within line of sight to at least four unique GPS satellites. GPS Satellites are positioned in specific geosynchronous orbits, so as to maintain robust operation and provide optimal coverage.

UAVs can navigate using GPS by obtaining pinpoint coordinates of its location then simply keeping track of where it is using a electronic map of the surrounding environment provided by GIS, Google Maps, or another such service. GPS is what military drones use and most commercial UAVs utilize it as well. One drawback of GPS navigation for UAVs is that if an immediate obstacle is not detected in the GPS map data the UAV could crash. Also, if a UAV is used in an indoor environment or some other such place that is out of range of GPS satellites, GPS navigation would be rendered completely useless and manual flight would have to be utilized. Therefore, this approach was not pursued.

2.1.5 Light Detection and Ranging (LiDAR)

Light Detection and Ranging, or LiDAR, is a range finding technology that measures range by shining a laser in every direction and measuring the amount of light reflected back. LiDAR can be used to make highly detailed 3D reconstruction, but is an emerging technology so is highly expensive and thus not widely available. This is what is used in high profile autonomous navigation projects, such as the Google Car. LiDAR is, unfortunately, too expensive (about $50k) for this approach as well as being too heavy and power-intensive to feasibly be used on the UAV.

2.1.6 Stereo Camera

The term Stereo Camera refers to a viewing device that has, at minimum, two cameras spaced apart by some constant, known distance. A typical Stereo Camera system, referred to as a Binocular Vision system, has two cameras spaced apart to simulate human eyes. The main application of a stereo camera is to take 3D images, utilizing a technique called stereo triangulation. This requires each camera to be calibrated and the exact distance between each camera to be known. Points in images from each individual camera are matched and then triangulated to produce a depth dimension (z), and can be stitched together, with software, to form the 3D images. A Stereo Camera is not good for our approach, though, because a UAV can fly in any direction and the Stereo Camera can only view the area directly in front of it. The UAV would have to blindly fly in whichever the direction the Stereo Camera was not facing, or else have a very large, cumbersome system with multiple views, which is undesirable.

2.1.7 Monocular Camera

A Monocular Camera is essentially a viewing device with one camera. When in motion, a monocular camera can mimic the functionality of a Stereo Camera. The system can be incredibly lightweight, inexpensive, and if properly implemented can serve the same purposes as a LiDAR for reconstructing the environment around the UAV for a small fraction of the cost. Monocular cameras are different from stereo cameras in that, with the use of a three hundred and sixty degree lens (sometimes referred to as a ‘fisheye’ lens), a Monocular camera can see in all directions. Therefore, a monocular camera will be utilized in the Approach presented later.

2.1.8 Open Computer Vision Library (OpenCV)

The Open Computer Vision Library, or OpenCV, is an open source library for computer vision and machine learning applications. It was designed to give programmers a unified infrastructure for computer vision software, as well as spur the use of computer vision in the commercial sector. To that end, it has been massively successful – companies such as Microsoft, Google, Intel, IBM, Honda, Toyota, and Sony make extensive use of the library in real-world projects, as well as many lesser-known start-ups. There are interfaces for C, C++, Python, Java, and MATLAB, as well as support for all operating systems - so it is friendly for a wide range of users. OpenCV also sees a great deal of use from the research community. Overall there are over seven and a half million downloads and around fifty thousand total users. OpenCV was developed mostly for use in real world applications and systems, and includes implementations for over five hundred different algorithms for everything from camera calibration to triangulation.

The dedicated library support, massive user base, commercial adoption, multiple interfaces, user-friendliness, and wide range of functionality are all reasons OpenCV was the library chosen to implement the Approach.

2.1.9 Keypoints

Keypoints, otherwise known as interest points or feature points, are specific points in a two-dimensional image that contain information relevant to solving a specific computational task. They are usually the result of some mathematics, algorithm, or neighborhood operation that is applied to a set of pixels, which can be reproduced reliably.

The idea of keypoints is very general, and the content of a keypoint depends on the task it is being used for. For example, keypoints can be detected to find the corners in an image using Harris Corner Detection algorithm, according to [1]; a method for finding Keypoints outlining objects, or "blobs," using the SIFT detector is outlined in [2]. Extraction of keypoints can require either a small or large amount of computing power, depending on the specific keypoint detector used and the complexity of the keypoint being detected.

2.1.9.1 Keypoint Detector

A Keypoint Detector is essentially a method or algorithm that allows us to generate a set of useful Keypoints for a given input image. There are a myriad of different Keypoint Detection algorithms, each useful in their own way, but we will focus on ones that detect Keypoints of object structures or "blobs" in images. Specifically, we will look at the SIFT, SURF, and BRISK [3] Keypoint Detectors, the reasons for which will be discussed next. The algorithms are all conveniently implemented in openCV, the library chosen for the approach as defined in (2.1.8).

Barandiaran et al [4] discuss the results of a thorough study on commonly used Keypoint detectors. The performance metrics used to evaluate the detectors were repeatability, distinctiveness, quantity, and efficiency. Those four metrics can then be combined to form two essential performance features: quality and efficiency. Quality here refers to a detectors ability to generate a precise, accurate, robust, and dense set of Keypoints. Quality is relative, though, and changes in response to the demands of different Computer Vision applications. Efficiency refers to the speed that the interest points are generated and how many resources and needed to perform the extraction. So, for my approach, these detectors will be examined from the point of view of someone needing to use them for 3D reconstruction in a real-time Embedded System, which will be explained later in (2.1.13).

To account for the fact that any detector used must be transform invariant, especially when used in a real time system, Barandarian et al [4] used three sets of test images, containing both photometric and geometric transformations. A set of synthetic images was also used for testing purposes. This type of test data ensures that the detectors will be robust under any operating conditions – lighting changes, intensity differences, position/orientation of the camera, or any intrinsic properties of the specific camera sensors. The three sets of test images can be seen below in (Figure 5).

Figure 5: Sample Images from Data Sets Used for Keypoint Evaluation [4]

The detectors evaluated in [4] are: Harris Corner Detector (HARRIS), Scale Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF) [5], Features from Accelerated Segment Test (FAST) [6], Binary Robust Invariant Scalable Keypoints (BRISK), Maximally Stable Extremal Regions (MSER) [7], Modified Center Surround Extrema or Modified CenSurE (STAR) [8], Oriented FAST and Rotated BRIEF (ORB) [9], and Multiscale 2D Nonlinear Scale Space (KAZE) [10]. Though this list does not include all of the detectors that were initially considered for the approach, all of the big players are evaluated and this paper gave an excellent cross section of the state-of-the-art for Keypoint Detectors. The authors used the openCV

implementation of each detector for their evaluation, as well, so their work was a perfect fit for my analysis.

The first evaluation performed was detection density. It compared how many points each detector generated for each set of test images, limited to 6k total. ORB, KAZE, and FAST had the most consistently dense sets of points, which is what we want for 3D reconstruction. The results of this test can be seen below in (Figure 6).

	SIFT	SURF	BRISK	ORB	FAST	HARRIS	MSER	STAR	KAZE
Graffiti	1,108	2,505	1,080	6000	5,759	699	555	874	5,209
Boat	1,451	4,088	3,691	6000	4,850	3,426	192	1,923	5,209
Brick	1,821	5,371	1,511	6000	5,458	5,571	1,447	1,461	5,209

Figure 6: Results of Detection Density Evaluation [4]

Next, the invariance of each detector to geometric transformations was evaluated. For this, four different tests were run: rotation-similarity transform, scale-similarity transform, affine transform, and perspective transform.

The rotation similarity test results showed that ORB has a solid repeatability around 55%. What startled me was the fact that this test showed that FAST actually has substantially higher repeatability when the image is rotated 180 degrees. Rotational invariance is not so important in our application, though, because our camera should optimally be at 0 degrees during flight. The results of this test can be seen in the graph below in (Figure 7).

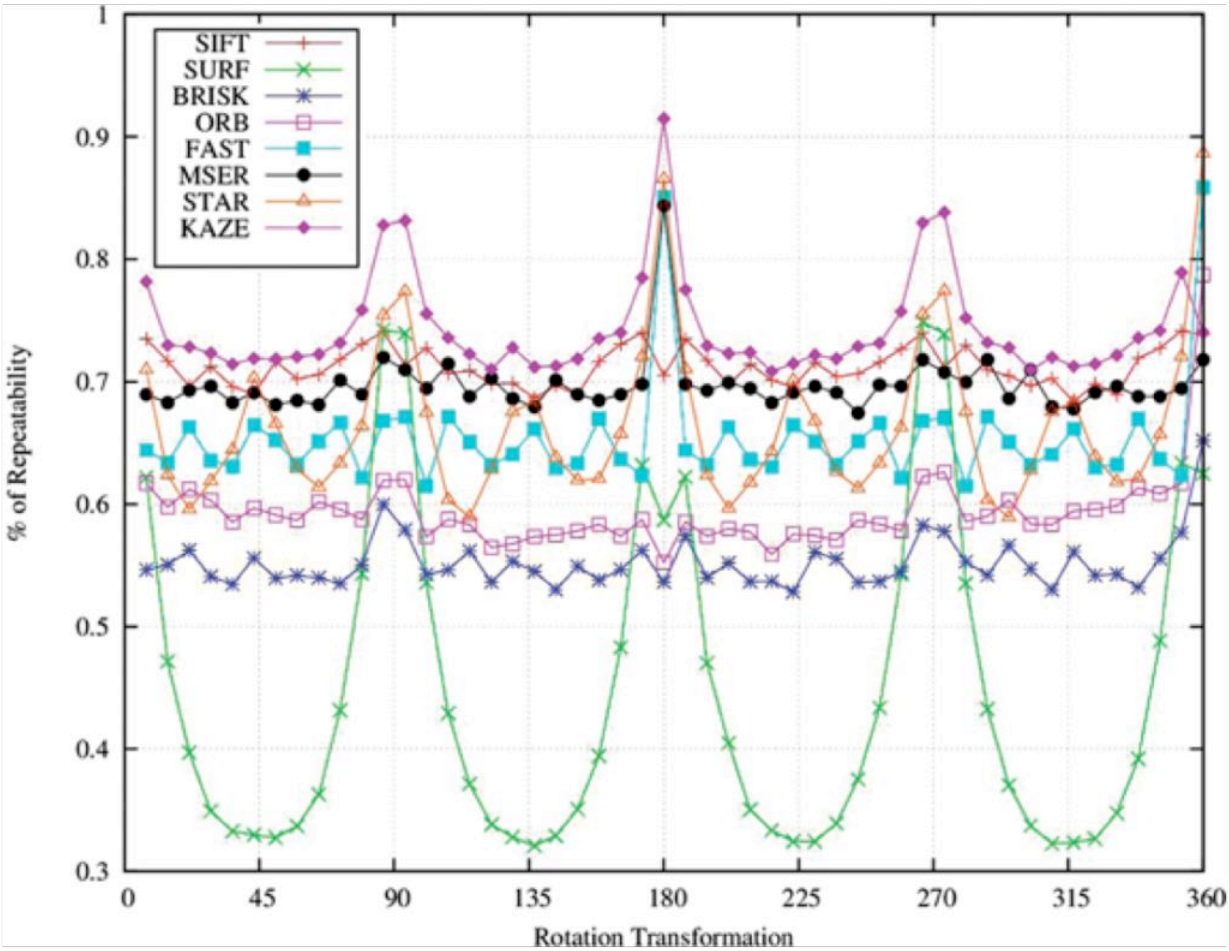

Figure 7: Results of Rotation Similarity Test [4]

The scale similarity test results shows that SIFT is the most robust even with very large scale factors; MSER and BRISK showed better results than SIFT, however, when objects size increased in the image scaling. It is worth noting that FAST is not scale invariant, so would not be good for the Approach outlined in (3)! The results of scale similarity test are shown in the graph in (Figure 8).

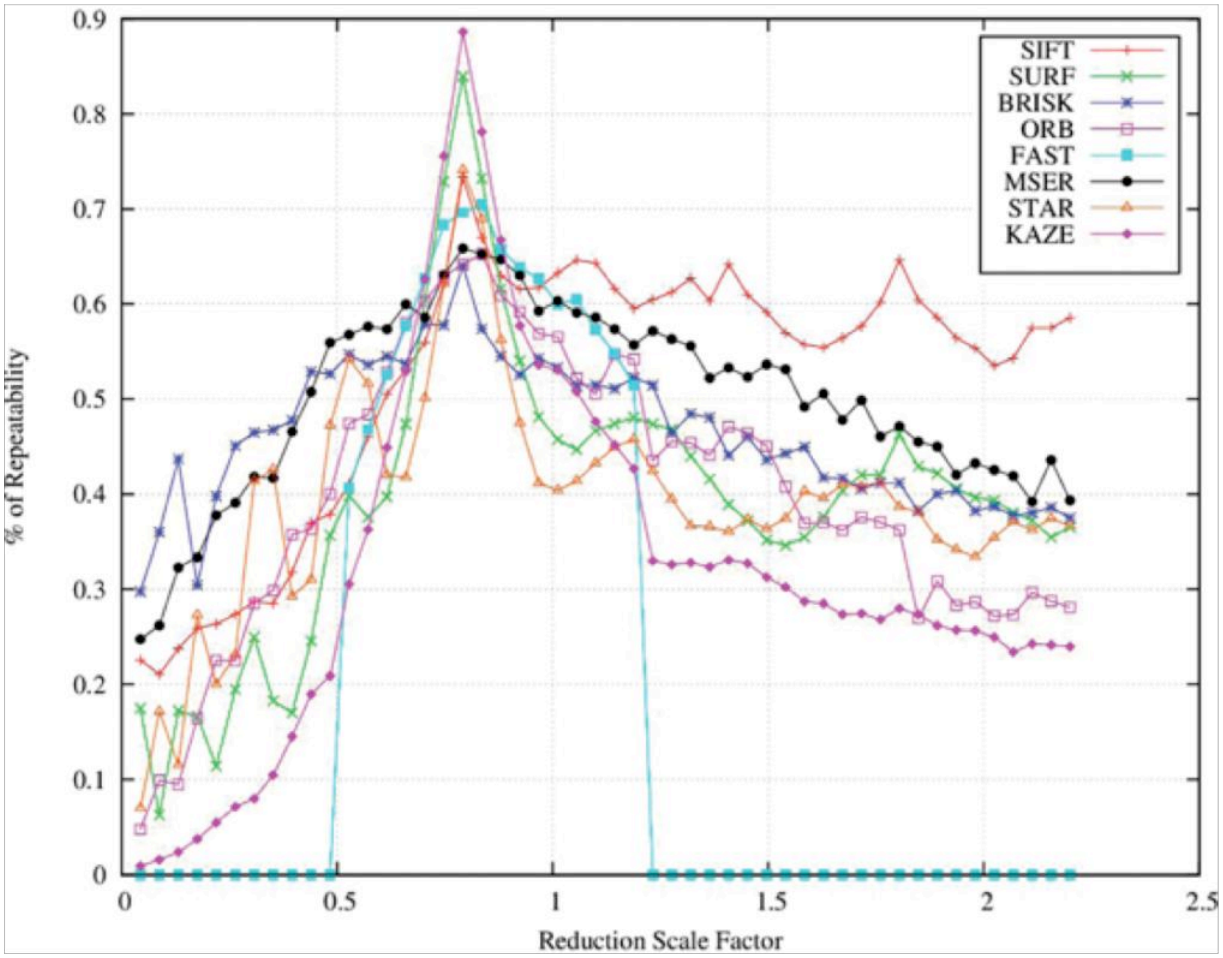

Figure 8: Results of Scale Similarity Test [4]

The affine transformation test may be the most relevant for my approach, as they test the changes in camera viewpoint. Every detector performed about the same, at a robust level, but none were fully affine invariant. KAZE showed the best result, however, standing out above the rest. A graph of the affine transformation test results can be seen in (Figure 9) below.

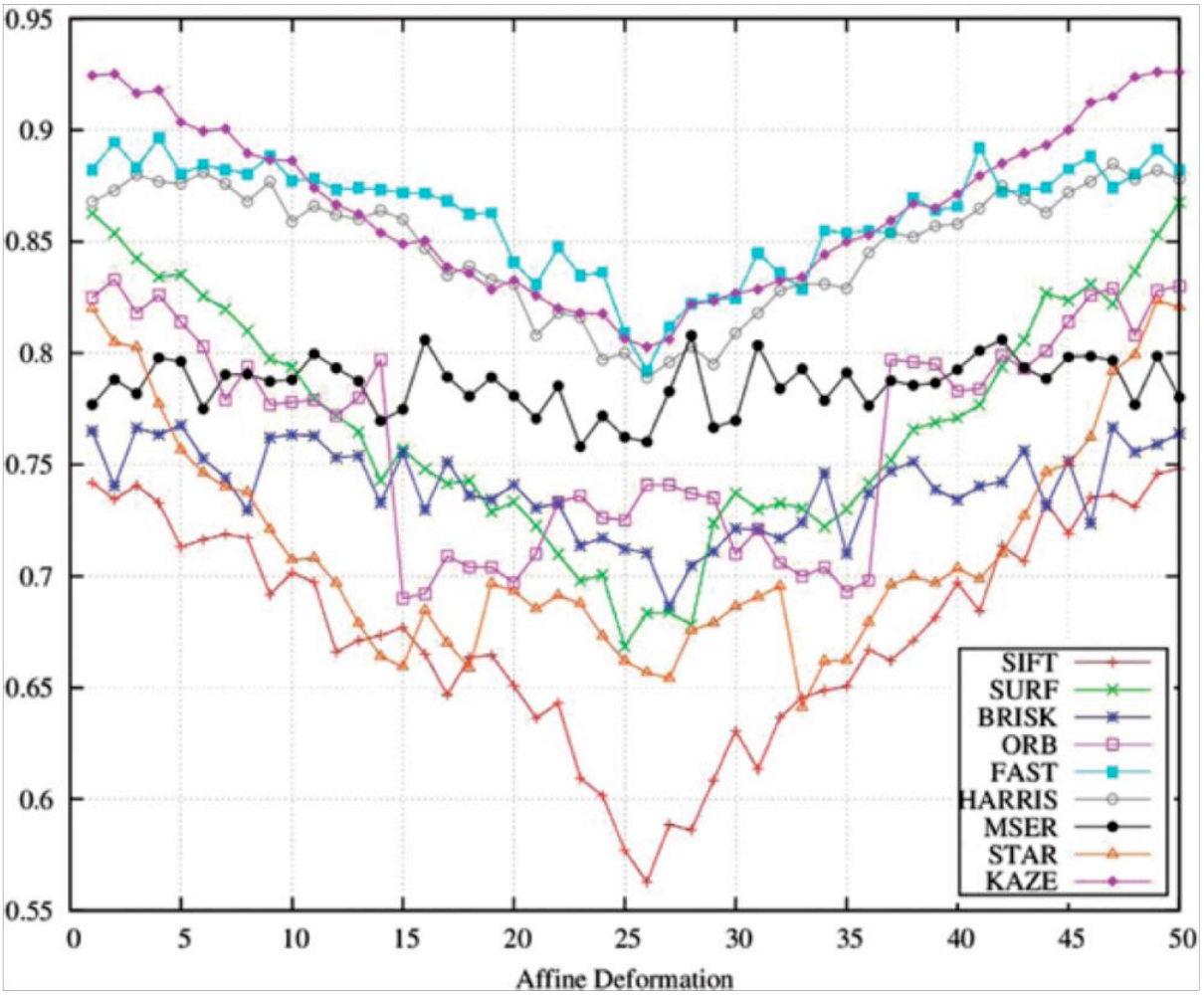

Figure 9: Results of Affine Transformation Test [4]

The last test for geometric transformation was perspective transformation. Though none of the detectors were perspective invariant, BRISK, ORB, followed by KAZE showed the best results although it is worth noting that all detectors tested were fairly sensitive to perspective transformation. The results of this test can be viewed in (Figure 10).

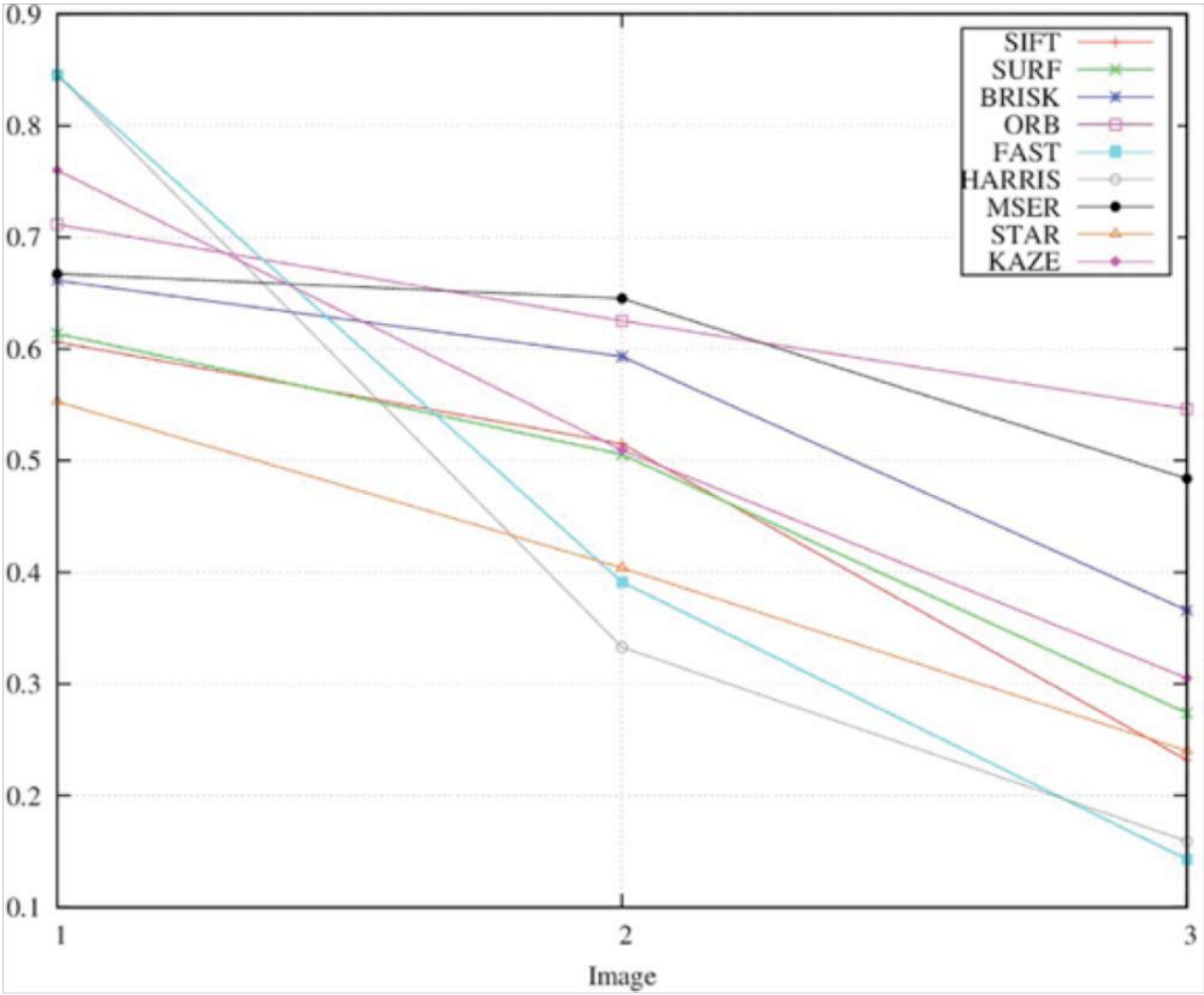

Figure 10: Results of Perspective Transformation Test [4]

The next set of tests was run to check the detectors invariance to photometric transformations. For this, four different transform tests were run: exposure photometric, noise photometric, and blurring photometric.

The first test, exposure photometric, checks how robust the detectors are to light intensity changes. A set of sample images consisting of fifteen different images of the same scene with varying lighting conditions was used for this test. Each detector was affected somewhat; repeatability decreased for each one as the intensity decreased (which is to be expected). The most consistent results came using BRIEF, SURF, followed by MSER. BRIEF was chosen to be

the most robust and light variation invariant. The noise photometric test evaluated the detectors against noise in the image. None were fully invariant to noise but BRISK and ORB faired the best. The last test, blurring photometric, evaluated the detectors against blurred images. The BRISK, ORB, and SURF detectors did the best here. The full results for these tests can all be viewed in [4].

The final evaluation for the detectors is efficiency. To test this, an image set of graffiti art was used and each detector extracted its maximum possible number of points. The fastest detectors were found to be BRISK, BRIEF, and STAR. HARRIS and ORB were very close to those three as well; the results of this test can be seen in (Figure 11). All possibly would work for our purposes, as they are efficient enough to work in real time. SIFT and SURF do not seem like they will be fast enough, but they are still the most widely recognized and used. I would still like to look at FREAK and GLOH to see if they may be more portable to embedded systems; that did not seem to be discussed at all in [4]. BRISK seems to be the overall winner, as it has the balance between being efficient and yet still robust.

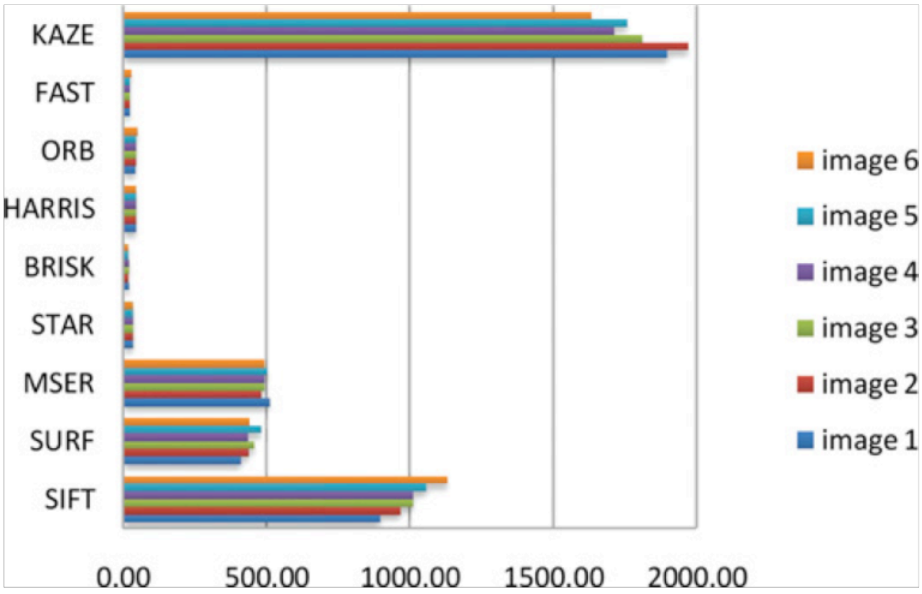

Figure 11: Efficiency Test Results, in Milliseconds [4]

For my approach, I would like to test the difference between SIFT, SURF, and BRISK detectors, based on the results of [4]. SIFT and SURF because of their industry standard reputation, and BRISK because it has been shown by [4] to be faster than both.

2.1.9.2 Keypoint Descriptor

A Keypoint Descriptor is essentially a Keypoint attribute that can be computed. They describe some type of characteristic of each specific Keypoint in each image. Descriptors are usually utilized for matching between Keypoints in multiple images; that is what they will be used for in my approach. Keypoint matching will be explained in (2.1.9.3). Descriptors should ideally be scale, transform, rotation, and translation invariant; the SIFT, SURF, and BRISK detectors are all of the above and will be explained in the next sections.

2.1.9.3 Keypoint Matching

Keypoint Matching is a method to match Keypoints between different images of the same scene or objects. When Keypoints are detected in an image, then Descriptors are computed, a variety of different methods can then be used to match the Keypoints using the info contained in the Descriptor. The type of Descriptor used should always match the Computer Vision application and the type of matching being used.

OpenCV implements two different methods for Keypoint Matching: brute force normal and brute force hamming, each with additional crosscheck matching optional. It is pointed out in [11] that Hamming distance should be used for SIFT and SURF, and Normal for BRISK when performing matching using OpenCV. The results of Hamming distance brute force matching of SURF keypoint matching drawn in OpenCV can be seen in (Figure 12) below.

Figure 12: Brute Force SURF Keypoint Matching Output in OpenCV

2.1.9.4 Scale Invariant Feature Transform (SIFT)

Lowe [2] outlines an algorithm for finding distinct scale and rotation invariant features in images, which can be used to provide matching for a range of different situations including 3D viewpoint change, noisy input images, and changes in lighting. This method was coined Scale Invariant Feature Transform or SIFT. The paper also provides a method for using these features for object recognition through matching features in the image to those stored in a database, but that is not relevant to our work. We merely are considering using the SIFT algorithm to obtain scale invariant feature points of the outlines of objects in our input images or each video frame.

SIFT detects Keypoints by utilizing a series of different scale Gaussian convolution filters, which are run on the input images. Next, the difference of the successive blurred images is taken (referred to as DoG or difference of Gaussian). Keypoints are defined as the minima and maxima of the DoG at different scales; the exact Keypoints are determined by comparing pixels in each DoG image to the eight nearest "neighbors" at that specific scale, as well as the nine neighbor pixels in each neighboring scale image. If the value for the individual pixel being considered is a min or max for all the values being compared, then it is considered a Keypoint.

An example of this process can be seen below in (Figure 13) where the X represents a random Keypoint being considered.

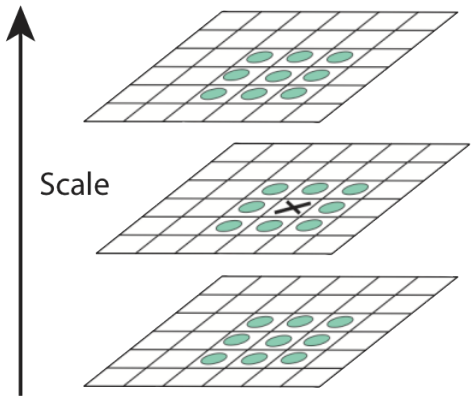

Figure 13: Detecting maxima and minima of Difference of Gaussian images by comparing the pixel at X with its neighbors in a 3x3 region at adjacent as well as current scale level to find SIFT keypoints [2]

Now that the SIFT Keypoints have been obtained, the next step is to calculate the SIFT descriptor for each individual Keypoint. First the gradient orientation and magnitude of certain sample points in a specific region around each individual Keypoint are calculated; this can be seen on the left side of (Figure 14). These points are then weighted with a Gaussian window operation, which is represented by the circle overlay. The sample points are then collected, for each 4x4 subregion, into orientation histograms (or 'bins'), which can be seen on the right side of (Figure 14). The example in (Figure 14) shows a 2x2 subregion calculated from 8x8 sample point regions, just for illustration. The descriptor is then finally represented using a vector that contains values of all bins.

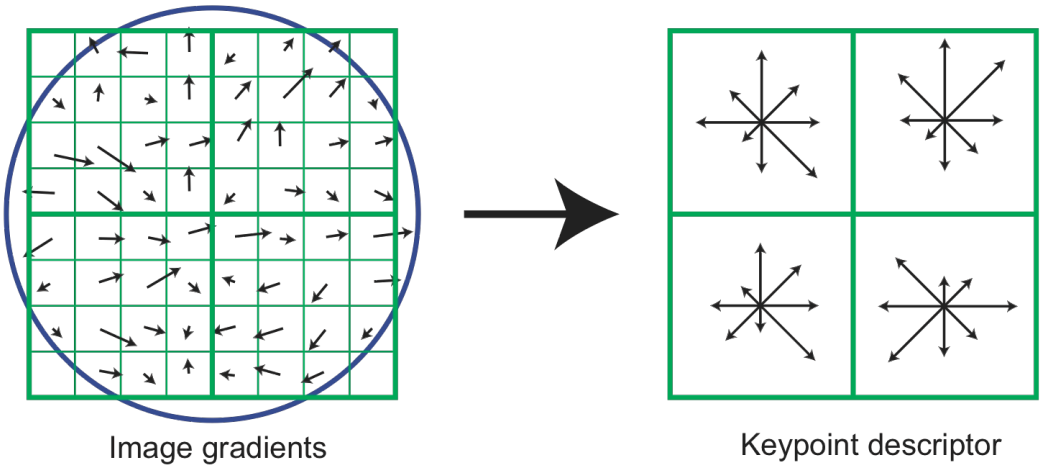

Figure 14: SIFT Keypoint Descriptor Illustration [2]

The strength of this approach is that the Keypoints are resistant to changes in lighting, noise, and changes in perspective. SIFT features are also scale and rotation invariant, and highly distinctive and therefore easier to match. The weakness of this approach is that it is one of the most computationally expensive according to [3]-[5]; improvements have been researched and implemented, however, so it could be considered obsolete. It is still used in many applications.

2.1.9.5 Speeded Up Robust Features (SURF)

Bay et al. [5] introduced a new rotation and scale invariant interest point descriptor and detector called Speeded Up Robust Features (SURF). The SURF detector relies on a basic approximation of the Hessian Matrix, and so is sometimes called the "Fast-Hessian" detector. The descriptor is based on a distribution of Haar-wavelet responses for each interest point neighborhood. It is similar to the SIFT detector in it's implementation, however, a reduced 64-dimensions are used to store bins (just like the example in Figure 14), compared to SIFT that uses a 128-dimensional vector, which reduces feature matching and computation time as well as increasing robustness. The trace of the Hessian, also known as the sign of the Laplacian, is

computed during detection and can be used for fast indexing during matching, as well. The SURF descriptor consists of two steps: orientation assignment, in which a reproducible orientation for the keypoints is calculated for rotational invariance, and descriptor extraction, in which the descriptor is calculated in similar fashion to SIFT. The SURF descriptor differs from SIFT in using less bins for histogram subregions, and instead of gradient features it uses more simple Haar wavelet response features. The strength of this approach is that is outperforms SIFT without losing Keypoint precision or repeatability, according to [5].

2.1.9.6 Binary Robust Invariant Scalable Keypoints (BRISK)

Binary Robust Invariant Scalable Keypoints (BRISK), proposed by Leutenegger et al., is a method for Keypoint Detection and Description designed to outperform the widely accepted SIFT and SURF methods while maintaining the same robust and useful output. Essentially, the BRISK detector is a happy medium between the high quality robust description of SIFT, and the low computation cost of the FAST detector. The BRISK method consists of two steps: scale-space keypoint detection and keypoint descriptor. In the scale-space keypoint detection step, keypoints are calculated for the image and scale dimensions using saliency criteria. For higher efficiency, keypoints are detected in octave layers within the image pyramid. Quadratic function fitting is then utilized to calculate the location and scale of each individual keypoint. In the keypoint detection step, a scaled concentric circle sampling pattern is first applied to a neighborhood around each keypoint, to calculate the keypoint characteristic direction descriptor. A second sampling pattern is then applied to the neighborhood around each keypoint, called the oriented-BRISK, to calculate the pairwise brightness comparison descriptor. These descriptors allow for efficient matching of keypoints between different images, due to their binary properties. The BRISK method is shown in [3] to be a faster alternative to SIFT and SURF while

providing similar robust, quality results, as well as being useful for real-time, limited computation power applications.

2.1.11 Camera Calibration

Camera Calibration, described in detail in [Chapter 11, 45], is a computer vision technique that allows calculation of intrinsic and extrinsic parameters of a pinhole model camera system. The intrinsic parameters, consisting of a geometric and distortion model, are made up of the Intrinsic Matrix (K) and the Distortion vector that can both be seen in (Figure 15). The extrinsic parameters consist of a rotation (R) and translation matrix (T).

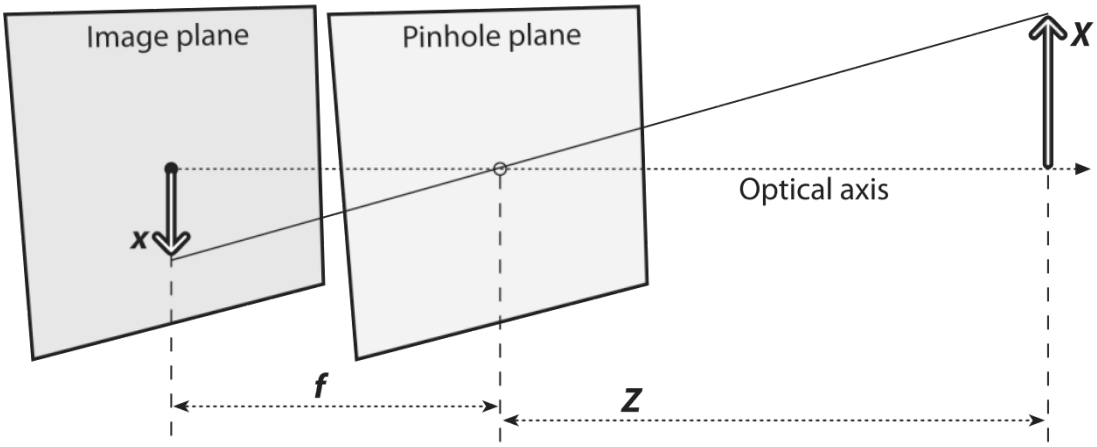

Figure 15: Pinhole Camera Model [45]

When using a pinhole model digital camera, light enters and is projected onto the imaging sensor as seen in (Figure 15); the intersection of the image plane and the optical axis is known as the principal point, f is the focal length of the sensor, Z is the length from object being photographed to the sensor, X is the length of the object, and x is the image of the object on the image plane. It follows that:

$$-x = f\frac{X}{Z}$$

therefore, the actual center of the image sensor is not considered the principal point since it is usually not perfectly aligned with the optical axis due to normal manufacturing errors (most digital cameras use cheap image sensors). This introduces the need for two parameters that will model the displacement of the center of coordinates with the actual physical image sensor: C_x and C_y. Now when any point is projected on the image plane (x,y), by a projective transform we can obtain:

$$x = f_x \frac{X}{Z} + C_x, \qquad y = f_y \frac{Y}{Z} + C_y \ .$$

Since pixels are usually rectangular in shape for images, rather than perfectly square, two more focal length parameters need to be introduced:

$$f_x = F.s_x, \qquad f_y = F.s_y \ ,$$

where F is the actual physical focal length (in mm) and Sx and Sy are the size of an image element (in ppm). These parameters together form the camera matrix (K) as seen in (Figure 16), and allow the use of Homogeneous coordinate system, which in turn allow the projective transform of a physical point to a projected point. In other words, it allows a single point to be represented by (X, Y, Z) rather than a Cartesian (x,y) system of normal digital images. This will be very useful later when we want to calculate real distances in the image using Triangulation data.

The fact that no lenses are perfect, introduces to problem of distortion into a pinhole camera (or any other lens-based) system. Many problems can be introduced during the manufacturing process that can lead to small abnormalities. There are two types of distortion that must be accounted for: radial and tangential. Radial distortion is a small distortion characterized by a Taylor series of equations:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad ;$$

it will produce a fisheye effect on the image if not corrected. Radial distortion is always 0 at the center of the image and will increase, moving toward the left and right periphery of the image. Tangential distortion is caused by the lens not being exactly parallel to the imaging plane; it can be accounted for by adding two additional parameters: p1 and p2:

$$x_{corrected} = x + [2p_1 y + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 x] \quad .$$

Now that we have defined all the parameters needed for correcting lens distortion, as well as a geometric model for mapping physical coordinates to projected points, we can fully define the camera intrinsic parameters, seen in (Figure 16). These parameters together give a complete specification of the camera behavior in the ideal pinhole camera model, allowing a relationship to be modeled between the coordinates of a real-world point to the coordinates of a pixel in the image, as well as solving all distortion problems. The Extrinsic parameters characterize the rest of the camera's non-ideal behaviors.

$$k_1, k_2, p_1, p_2, k_3$$

Distortion Coefficients

$$\begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsic Matrix (K)

$$R = R_z(\theta), R_y(\varphi), R_x(\psi)$$

Rotation Matrix

$$T = (x, y, z)$$

Translation Matrix

Figure 16: Camera Intrinsic Parameters (Distortion Coefficients and K) and Extrinsic Parameters (Rotation and Translation Matrices)

2.1.11.1 Camera Calibration with OpenCV

Camera Calibration in OpenCV can quickly estimate the intrinsic and extrinsic parameters of a camera, using an input sequence of multiple images containing some previously known calibration pattern. The most common calibration pattern is an NxN chessboard; this is what was used for calibration of the camera in this Approach. OpenCV has a built-in function that implements the actual calibration algorithm called calibrateCamera that was very helpful and well documented. The algorithm needs a minimum of two different views of the chessboard to compute the Calibration, however the more views that are used increase the accuracy of the result. In the Approach here, an 8x8 chessboard was used with 25 different views. The results from OpenCV Camera Calibration of the PC webcam used in the pure software implementation can be viewed in (Figure 16).

```
<?xml version="1.0"?>
<opencv_storage>
<Intrinsics type_id="opencv-matrix">
<rows>3</rows>
<cols>3</cols>
<dt>f</dt>
<data>
9.51050232e+02     0.4.28997437e+02     0.8.80731934e+02     1.25398397e+00     0
0.1</data></Intrinsics>
</opencv_storage>
```

Figure 16: OpenCV PC Webcam Camera Calibration Results (XML format)

2.1.12 Structure from Motion (SfM)

The term Structure from Motion (SfM) refers to a method that enables geometric structures to be extracted from sequential images taken with a single, moving camera. In other words, it allows us to calculate information that will enable us to triangulate keypoint matches from our images and effectively allow us to build a Range Image, which can be the backbone of a 3D reconstruction system. The SfM method chosen to use for this Approach is laid out by

Hartley and Zisserman [12] in Chapter Nine of their famous book *Multiple View Geometry in Computer Vision.* The method utilizes Epipolar Geometry (described in Section 2.1.12) and was adapted to OpenCV by Baggio et al [11]; a basic illustration is provided in (Figure 17). This method assumes that the camera used for gathering input images was fully calibrated beforehand, resulting in a calibration matrix (K) of the cameras intrinsic parameters. First, the motion of the camera is calculated for two successive image frames by calculating the Essential Matrix (E), using the Fundamental Matrix (F). To obtain F, Keypoints are detected in each frame, matched, and input into an OpenCV function called findFundamentalMat. After F is obtained, we can calculate E using F and K, to obtain the camera projection matrixes P and P1. Finally, we can use matrices P, P1, and the set of matched Keypoints to perform Triangulation to obtain a Range Image.

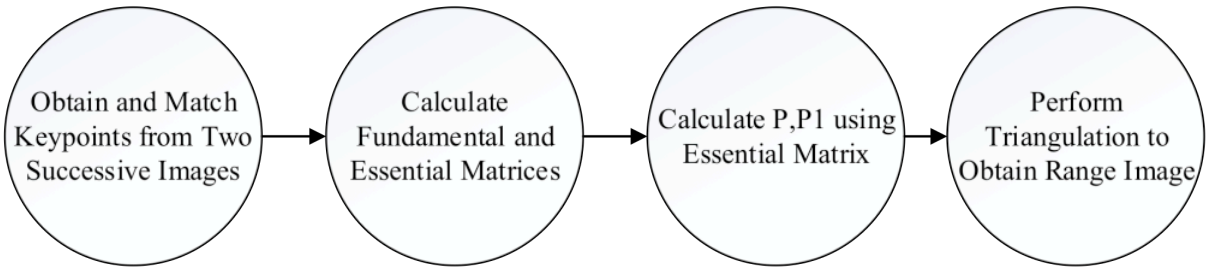

Figure 17: Basic Structure from Motion Dataflow

2.1.13 Epipolar Geometry

Epipolar Geometry is defined by Hartley and Zisserman [12] as "the intrinsic projective geometry between two views. It is independent of scene structure, and only depends on the cameras' internal parameters and relative pose." Essentially, it is the geometry involving the intersection of two image planes, with the baseline adjoining the two camera centers serving as an axis. There is some terminology that is important to know when dealing with Epipolar

Geometry: epipole, Epipolar plane, and Epipolar line; these terms are illustrated in (Figure 18). An epipole is a point of intersection of the baseline with an individual image plane; in (Figure 18) the epipoles can be found at e and e'. An Epipolar plane is any plane (∏) that contains the baseline; in (Figure 18) each baseline intersects the image plane in the individual epipolar lines l and l'. An Epipolar line is the intersection of a Epipolar plane with the image plane; as the position of the three dimensional point X in (Figure 18) changes, Epipolar planes change around the baseline. All Epipolar lines will intersect at the epipole, which can be seen in (Figure 18) as well. It is clear from fully examining Epipolar Geometry that it is very useful for our approach; it is essentially a geometric system, which allows us to easily calculate 3D points given a set of corresponding points on two image planes. This method is also known as Triangulation, which is discussed in-depth in Section 2.1.14.

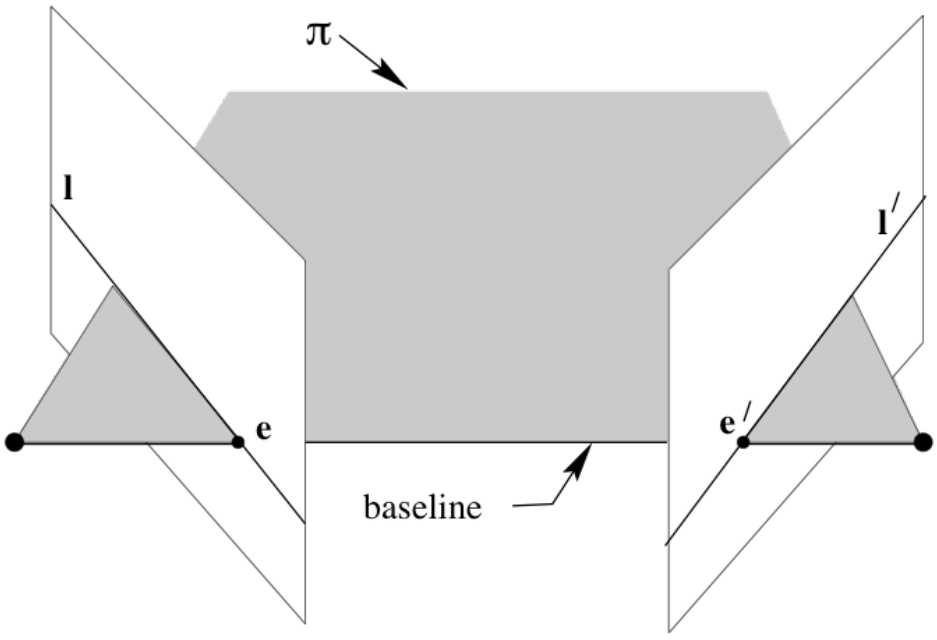

Figure 18: Epipolar Geometry [12]

2.1.13.1 Fundamental Matrix

The Fundamental Matrix (F), defined as "the algebraic representation of Epipolar geometry," [12] is a concept first introduced by Luong and Faugeras in [43]. A full proof, derivations, and in-depth explanation of this concept can be found in [Chapter 9, 12] and [44]; full examples of calculating the Fundamental Matrix in various situations can be found in [Chapter 11, 12]. To construct the Fundamental Matrix using matched points from consecutive images, we consider each pair of matching points from image one (x) and image two (x') to have a Epipolar line l' connecting them. The "Epipolar line is the projection in the second image of the ray from the point x through the camera center of the first camera," [12]. This forms a mapping, which follows the equation:

$$\mathbf{x} \mapsto \mathbf{l}'$$

of a point in the first image to the corresponding Epipolar line in the second image. Each mapping forms a correlation, or a "projective mapping from points to lines," [12] which over iterations forms the 3x3 homogeneous Fundamental Matrix, which follows the equation:

$$\mathrm{F} = [\mathbf{e}']_{\times}\mathrm{P}'\mathrm{P}^{+}$$

$$\text{where } \mathbf{e}' = \mathrm{P}'\mathbf{C} \text{ and } \mathrm{PC} = 0.$$

As well as the condition:

$$\mathbf{x}'^{\mathrm{T}}\mathrm{F}\mathbf{x} = 0$$

where F is the Fundamental Matrix, e' is the projection of the first cameras center, P' is the second camera projection matrix, P+ is the pseudo-inverse of P, C is the common camera center of P and P', and x'T and x are two corresponding image points. These equations clearly show that the camera projection matrices P and P1 can be easily obtained using the Fundamental

Matrix. To obtain normalized results, however, the Essential Matrix must be calculated using F to find a normalized P and P1.

2.1.13.2 Essential Matrix

The Essential Matrix (E) is defined as "the specialization of the fundamental matrix to the case of normalized image coordinates," [12] and is a concept that was first introduced by Longuet-Higgins in [42]. Basically, the Essential Matrix has useful additional properties and assumes that both cameras are calibrated beforehand (whereas the fundamental matrix does not). A very basic proof and explanation of E follows, for a full proof see [Chapter 9, 12].

To define E, first consider a camera matrix that is decomposed to the equation:

$$\mathtt{P} = \mathtt{K}[\mathtt{R} \mid \mathbf{t}]$$

where x = Px is any point in a image. If K is known, which in the Approach here it is, its inverse can be applied to x to obtain a point:

$$\hat{\mathbf{x}} = \mathtt{K}^{-1}\mathbf{x}$$

This image point is now expressed in what is known as normalized coordinates. It is essentially a point x with respect to the calibration matrix (K) that also acts as the identity matrix. Now consider two camera matrices that are normalized:

$$\mathtt{P} = [\mathtt{I} \mid \mathbf{0}] \text{ and } \mathtt{P}' = [\mathtt{R} \mid \mathbf{t}]$$

The fundamental matrix that corresponds to this pair of matrices is the Essential Matrix; it takes the form of a 3x3 homogeneous matrix that follows the equation:

$$E = [T]xR = SR$$

where E is Essential Matrix, S is skew-symmetric matrix, and R is rotation matrix. The Essential Matrix can be calculated using normalized coordinates from two images, or directly from F and K using the equation:

$$\mathtt{E} = \mathtt{K'}^{\mathsf{T}}\mathtt{F}\mathtt{K}$$

After the Essential Matrix is obtained, the camera matrices (P and P1) can be calculated using the equations:

$$\mathtt{P'} = [\mathtt{U}\mathtt{W}\mathtt{V}^{\mathsf{T}} \mid +\mathbf{u}_3] \text{ or } [\mathtt{U}\mathtt{W}\mathtt{V}^{\mathsf{T}} \mid -\mathbf{u}_3] \text{ or } [\mathtt{U}\mathtt{W}^{\mathsf{T}}\mathtt{V}^{\mathsf{T}} \mid +\mathbf{u}_3] \text{ or } [\mathtt{U}\mathtt{W}^{\mathsf{T}}\mathtt{V}^{\mathsf{T}} \mid -\mathbf{u}_3]$$

These are four possible choices for the second camera matrix (P1) that satisfy the two possible factorizations of E=SR given by the singular value decomposition (SVD) of E into the equations:

$$\mathtt{S} = \mathtt{U}\mathtt{Z}\mathtt{U}^{\mathsf{T}} \quad \mathtt{R} = \mathtt{U}\mathtt{W}\mathtt{V}^{\mathsf{T}} \quad \textit{or} \quad \mathtt{U}\mathtt{W}^{\mathsf{T}}\mathtt{V}^{\mathsf{T}}$$

The first camera matrix (P) is always assumed to be the matrix:

$$\mathtt{P} = [\mathtt{I} \mid \mathbf{0}]$$

Each one of the solutions for P1, as seen in the equations and illustrated in (Figure 19) must be tested, as only one of them will actually contain the points and be a correct view of the scene for that particular camera. Testing a single point to see if it is in front of both cameras (P and P1) is a sufficient test to figure out which solution works and is a mathematically correct P1. This is a very basic introduction to Essential matrices; there are many special situations to consider and additional notes that can all be found with in-depth explanations in [12] and [42].

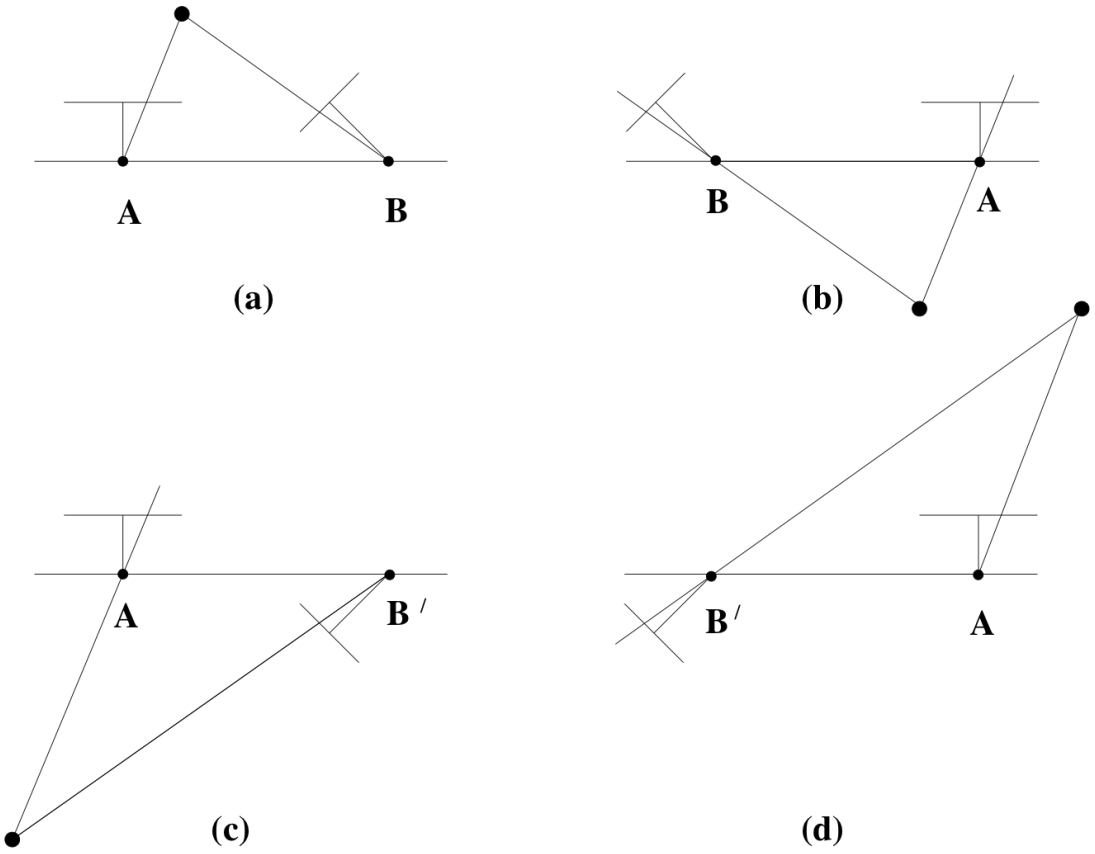

Figure 19: Camera Matrix P1 Possible Choices Illustration [12]

2.1.14 Triangulation

In Computer Vision, Triangulation is a method to reconstruct geometry from a series of two-dimensional images to form three-dimensional points (x, y, z). There are many different Triangulation methods applicable for Computer Vision, for example Ray Intersection, as demonstrated in [11].

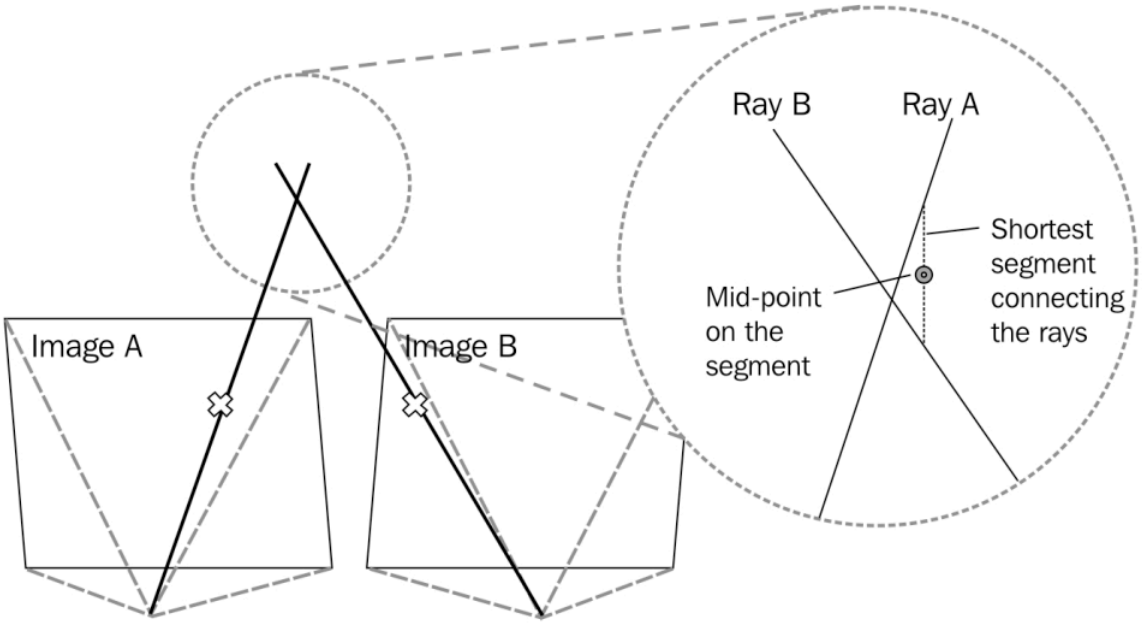

Figure 20: Ray Intersection Triangulation [11]

In Ray Intersection, seen illustrated in (Figure 20), one ray is constructed from each camera's central projection point – derived from the P and P1 camera matrices previously explained in (2.1.13.2) – and a Keypoint on a frame taken by that camera that matches one Keypoint on the other camera's frame. These rays will, theoretically, intersect at some point in 3D space – giving one 3D point from one Keypoint match in two camera frames. If they do not intersect, the median point on the smallest line connecting the projected rays will be used instead. This process is repeated for every set of Keypoint matches from the two camera frames. The end result is a set of 3D points representing the scene in front of the camera, or in the case of

my approach, the UAV. This set of 3D points is known as a Range Image and can be used by a 3D reconstruction algorithm to create a 3D map of the environment surrounding the UAV. An example of a real-time Range Image output can be seen in Section 3.2 as part of the Software Implementation. Essentially, a Range Image is a post-processed image that includes a depth value (z) for either all pixels in the image, or in our approach case, certain keypoints associated with objects. Often times this depth value will be associated with some kind of color, for visualization purposes; for example, darker color for large depth values and lighter colors for small depth values.

The approach presented in this paper will not utilize Ray Intersection Triangulation, as it has been shown to not be a robust or accurate method due to the fact that the projected rays do not intersect in a majority of cases [12]; though, admittedly, the best possible outcome will still be a mathematical guess. The method used in this approach is a slightly more reliable one, outlined by Hartley and Zisserman [Chapter 12.2, 12], known as Linear Triangulation. OpenCV does not include much implementation support for Triangulation, unfortunately, so this function had to be added to the software implementation in my approach using code from [11].

Linear Triangulation works by creating a system of linear equations to solve using two matching normalized (meaning multiplied by K) homogeneous points (one from each image frame) and the camera projection matrices for each frame, P and P1. The solution to this system of equations will yield a homogenous, normalized three-dimensional point, which can be used to form the Range Image. First, for each set of image points, we find three linearly independent equations given by the equations:

$$\begin{aligned} x(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{1\mathsf{T}}\mathbf{X}) &= 0 \\ y(\mathbf{p}^{3\mathsf{T}}\mathbf{X}) - (\mathbf{p}^{2\mathsf{T}}\mathbf{X}) &= 0 \\ x(\mathbf{p}^{2\mathsf{T}}\mathbf{X}) - y(\mathbf{p}^{1\mathsf{T}}\mathbf{X}) &= 0 \end{aligned}$$

where each $\mathbf{p}^{i\mathsf{T}}$ represents a row in P. These linear equations then follow the form AX = 0, which can be composed into the matrix:

$$\mathtt{A} = \begin{bmatrix} x\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{1\mathsf{T}} \\ y\mathbf{p}^{3\mathsf{T}} - \mathbf{p}^{2\mathsf{T}} \\ x'\mathbf{p}'^{3\mathsf{T}} - \mathbf{p}'^{1\mathsf{T}} \\ y'\mathbf{p}'^{3\mathsf{T}} - \mathbf{p}'^{2\mathsf{T}} \end{bmatrix}$$

where there are two equations representing each respective set image points, giving a total of four homogeneous unknowns. There are two ways to solve this set of equations; one is a homogeneous method, the other inhomogeneous. In this approach, the inhomogeneous method was used; by letting X = (X, Y, Z, 1) the set of equations AX=0 can be reduced to a set of four inhomogeneous equations with only three unknowns. This allows a least squares solution to be pursued to solve for the final three-dimensional coordinate X. This method is more mathematical in approach than Ray Intersection, and is proven to be more effective than it in [11]; it is not optimal, but is a simple implementation that allows for a quick prototype and proof-of-concept. Hartley and Zisserman present an optimal Triangulation method in [Chapter 12.5, 12].

2.1.15 Field Programmable Gate Array (FPGA) Embedded System

The Objective (1.2) of this paper states that the goal is to investigate existing 3D-based image representations for their portability to an FPGA Embedded System. A Field Programmable Gate Array (FPGA) is a type of integrated circuit (IC) chip built to have logic, which is reconfigurable by the end-user. Some hardware design language, such as VHDL, usually specifies these configurations. FPGAs accomplish this reconfiguration through arrays of reprogrammable logic blocks, which can be configured to represent anything from a XOR gate to a complex function. This ability to reconfigure the physical hardware in a system, after

manufacturing has been completed or even during execution, makes FPGAs especially useful for certain applications – specifically Embedded Systems, and in the case of the RazorCam, Image Processing in a Embedded System.

An FPGA essentially represents a happy medium between the flexibility of general-purpose CPU, which are not tailored for any specific task, and an Application-Specific Integrated Circuit (ASIC), which is engineered to perform one specific task. They are much lower cost than ASIC and require much less time to design, making them very useful for proof-of-concept or prototyping applications. This is why the FPGA was chosen for my approach – though not as fast as an ASIC, it will allow a prototype of my method to be designed and executed on physical hardware in a fraction of the amount of time. The FPGA also provides an advantage over using a general purpose CPU, as hardware accelerators can quickly be implemented on the FPGA and used to parallelize certain parts of a given algorithm. Even high-end CPUs have usually a maximum of eight physical processing cores for which to parallelize code; a FPGA could potentially have hundreds.

2.2 Related Work

This section will go over some research work that is related to what is presented in my approach. First, a few interesting Real-World Applications of UAVs under development will be discussed, proving to the reader why UAVs are a highly relevant subject for academic, industry, as well as military research. Next, some current research pertaining to various 3D Reconstruction Approaches that were considered for my approach will each be discussed to show the reader why the specific approach for my implementation was chosen. Lastly, some related research on Video-Based Embedded Systems is briefly discussed, so the reader can get an idea of the state-of-the-art in this area.

2.2.1 Real-World Applications of UAVs

Though there are a myriad of different unique real-world applications for UAVs, we will briefly cover only a few: surveying, inspection, and surveillance. These specific applications illustrate the importance of UAV research and it's immediate benefits to society as a whole. Each application will benefit greatly from the ability of the UAV to fly semi or completely autonomously, which is the end goal of this paper's 3D reconstruction approach for UAVs.

2.2.1.1 Surveying

The recent availability of low-cost UAV platforms has spurred the use of this technology in the area of Surveying. According to [14]-[16], low-cost UAVs have been used in surveying applications for everything from spotting brush fires, mapping archaeological areas, and agricultural operations, to just taking pictures of public events or mapping urban areas. This area

especially helps to illustrate the incredible range of applications that low-cost UAVs are capable of.

2.2.1.2 Inspection

A great deal of attention has also recently been given to utilizing autonomous UAVs in Inspection applications. As demonstrated in [17]-[20], UAVs have seen a massive spike in use for jobs that are both tedious and dangerous, such as Inspection of bridges, high-voltage power transmission lines, gas pipelines, and building externals. The fact that autonomous UAVs can replace human workers in such dangerous situations is very beneficial; no longer will we have to endanger human life to accomplish such tedious, yet essential, tasks.

2.2.1.3 Surveillance

One, almost obvious, application of UAVs is Surveillance. The military makes extensive use of its UAV fleet for all kinds of Surveillance, including spying, targeted assassinations, and other related military applications as mentioned in [16]. In the civil sector, many of the Inspection applications discussed in (2.2.1.2) could also be considered Surveillance applications. The fact that UAVs could potentially be used for Surveillance on citizens, however, is a concern for many Americans. It is a definite grey area right now as far as the legalities go for any civilians using UAVs for surveillance of other citizens; the FAA does, however, strictly forbid flying UAVs in US airspace for anything other than hobby use. Surveillance applications, for the most part, seem to be more tailored towards military use.

One interesting Surveillance-related application of great benefit to the civil sector is UAV search and rescue. Doherty and Rudol [21] outline a scenario in which autonomous UAVs could potentially be used in emergency situations to rescue injured civilians – even attempting to

administer medical aid. This search and rescue technology is not far off; it is something we will see in the near future, as a research platform is currently being developed by [22].

2.2.1.4 Traffic Control

Another interesting application of autonomous UAVs is Traffic Control. Puri [23] performs a survey of commercially available UAVs, as well as current research, that can be applied to Traffic Control scenarios. UAVs can "provide a 'bird's eye view' for traffic surveillance, road conditions and emergency response," [23] which is unparalled by any kind of stationary Traffic Control device (red-light camera, pole-mounted sensor, automated radar, etc.) due to its mobility. This application is, however, another grey area that some Americans are not comfortable with; receiving a traffic violation from a UAV may not be so pleasant! It is, whichever way you feel about it, technology we will likely see put to use in the near future; real-world applications are currently under research and development by [24] and [25].

2.2.2 3D Reconstruction Approaches for UAVs

There are many different approaches to reconstructing the environment around a UAV. 3D Reconstruction is important for any autonomous UAV as it allows the vehicle to be able to move while avoiding obstacles and other obstructions to its movement. Path planning and obstacle avoidance for a UAV can be especially difficult, since the vehicle can potentially move in any direction – unlike a regular helicopter or airplane. Therefore, it is important to have an accurate representation of the whole environment surrounding the UAV, not just what is directly in front of it. This section will discuss the different 3D Reconstruction Approaches that were considered for my implementation, and why the Monocular Camera Approach was chosen over the rest.

2.2.2.1 LiDAR Approach

LiDAR, described in (2.1.5), is able to create highly detailed 3D Reconstructions of its surroundings. A method for using LiDAR to produce 3D Point Clouds, the same output as my approach described in (3.1), has been developed and outlined by [26]. Due to the high cost of obtaining a LiDAR, this approach was not chosen for my implementation. There are many speculations about when the cost of LiDAR will fall to a price-point where it will be more feasible for widespread academic research; nobody can really say. LiDAR is also big, bulky, and requires a lot of battery power to use - not so great for a lightweight UAV with limited battery capacity. There are promising developments, though, such as mini LiDAR designed specifically for UAVs proposed by [27]. This technology definitely has a promising future in autonomous vehicle applications – Google currently uses it in the Google Car, and there is a massive amount of research being done across the board.

2.2.2.2 Stereo Camera Approach

A Stereo Camera (2.1.6) is a camera system that utilizes triangulation to create 3D images. This type of camera system can be utilized to implement a 3D Reconstruction Approach for a UAV. Such Stereo Camera based navigation systems have been described and developed by [28] and [29], and so have been proven to be feasible for use in a theoretical UAV system. In a real-world application, however, Stereo Camera systems are not always the best choice for a UAV navigation approach. Magree et al. [30] explains that they are not a common payload for UAVs, as well as having issues with range measurement limitations involving the stereo baseline (distance between the cameras); at certain distances, this limitation causes Stereo Cameras range measurement to be equal with a Monocular Camera. So essentially, at long range, a Stereo Camera gives no computational or other benefit over using a Monocular Camera. When a

Monocular Camera is combined with a 360-degree lens, it can gather images in every direction surrounding the UAV; Stereo Cameras are not fully compatible with 360-degree lens. These two reasons, along with the cumbersome nature of many commercially available models, are mainly why a Stereo Camera system was not chosen for my approach.

2.2.2.3 Monocular Camera Approach

A 3D Reconstruction Approach using a Monocular Camera (2.1.7) was chosen for my implementation approach; some of the reasons for which were outlined in (2.2.2.1) and (2.2.2.2). Another reason for choosing the Monocular Camera sensor was the large amount of research currently being conducted in this area; it is clearly a hot topic. Magree et al. [30] describe their implementation of a terrain mapping method for UAV obstacle avoidance that utilizes a Monocular Camera sensor and feature extraction.

Another UAV 3D Reconstruction and obstacle avoidance method, outlined in [13] utilizes a Monocular Camera sensor and optical flow to achieve UAV obstacle avoidance. In this approach, a 'balance strategy' is used to avoid obstacles, which maintains equal distance optical flow on right and left sides of the UAV. If optical flow is greater on the left side, the UAV turns towards the right, and vice versa. The only problem with this approach, however, would be situations where there is no optical flow; for this reason, optical flow cannot be exclusively used for a robust UAV obstacle avoidance system, but is still worth examining as a potential tool for certain situations.

A method for UAV avoidance of obstacles approaching head-on, using a Monocular Camera sensor and SURF feature extraction, was outlined by Mori and Sherer [31]. It utilizes the SURF (or SIFT can be used) algorithm's scale invariant features to detect when objects are getting larger, using a method referred to as 'relative size.' If a detected object gets larger and

larger in the camera's field of view, it's relative size is considered to be increasing, meaning the object is getting closer to the UAV and avoidance actions must be performed. Feature point matching is used to match the objects between different image frames and calculate the relative size of each; since objects approaching the UAV head-on will always get larger until they almost take up the entire field of view, there is a high avoidance success rate for this approach.

The implementation approach described in (3) was influenced heavily from work performed by Lee et al. [32] on obstacle avoidance for lightweight UAVs utilizing a Monocular Camera sensor. In their paper, a method is proposed to detect three-dimensional coordinates of objects in Monocular Camera pictures, and how to use this information to plan a flight path so the UAV can avoid them. To reconstruct the 3D information of the objects from a 2D image, the MOPS [33] algorithm is performed to obtain the approximate 3D outline of the objects. Then, the SIFT algorithm is performed to obtain the three dimensional internal feature points of the objects. These two sets of information are then merged to obtain three-dimensional information of all objects in the image. Upon close examination of this approach, I noticed that the step in which the 3D points of objects were calculated from the SIFT or MOPS feature points was not explained in detail; in fact, it was assumed that this info would be obtained in their dataflow through a unspecified calculation. Therefore, I decided to focus on this aspect of a 3D Reconstruction approach in my implementation – calculating the 3D points of objects given feature points detected from two consecutive image frames taken by the Monocular Camera sensor to form what is essentially a depth map of the objects detected in the camera's field of view. This information can be used in a variety of applications, but it is assumed for the purposes of this paper that the depth map will be used later in a 3D Reconstruction and Navigation system for a UAV, similar to the method described in [34].

2.2.3 Video-Based Embedded Systems

This section highlights a few related areas of work in Video Based Embedded Systems. Related areas that will be examined include some general hardware/software smart-camera systems and the RazorCam Embedded Smart Camera system.

2.2.3.1 Hardware/Software Systems

A review of current embedded smart-camera based video processing systems performed by Rinner et al. [38] explains that many different system architectures are currently being used, including: general-purpose CPU, FPGA, and System on Chip (SoC) approaches. They go on to assert that most of the systems examined in the study today run simply on the general purpose CPU platform; however, a strong trend toward FPGA and SoC use is occurring. This is due to the fact that computing power needs for these systems is steadily rising; by shifting the system from a single general-purpose CPU architecture to a multi-FPGA array, for example, power consumption can be minimized while exploiting the parallelization inherent in that architecture for increased performance. Some examples of such FPGA-based smart camera architectures can be found in [39]-[41], where hardware acceleration on an FPGA is utilized to increase the performance of a system that is first completely implemented in software. Hardware/software partitioning and acceleration of identified critical sections is performed, much like in the approach presented in (3).

2.2.3.2 RazorCam Embedded Smart Camera System

The RazorCam Embedded Camera System (RazorCam), created by Mefenza, Yonga, and Bobda [35], [36], is an FPGA-based, rapid prototyping, high-performance Smart Camera system on chip, which allows designs to be quickly implemented and analyzed in realistic environments.

As seen on the left side of (Figure 21), the RazorCam consists of a motherboard module, a Xilinx Zynq-7000 FPGA, a TFT display, and a digital Monocular Camera sensor (for the approach outlined in (3), the infrared sensor is not used). RazorCam utilizes Linux as it's embedded operating system, which runs on the Zynq's on-board ARM processor, and was chosen for its solid and well-known development platform and tools.

Figure 21: RazorCam Embedded Camera System [37]

The system's uniform programmability and seamless integration of Hardware Accelerators are insured through the Component Interconnect for Data Access (CIDA), the design and implementation of which is outlined in [37]. The CIDA allows interfacing between hardware accelerators, hardware, and software through its Direct Memory Access (DMA) capabilities. As explained in [37], the drivers to control the CIDA from the embedded operating system have been developed and OpenCV was ported to the system and is fully accessible in the Linux environment. A few common image-processing accelerators have already been implemented, and described in [35], such as convolution, thresholding and segmentation. In the

Hardware Implementation outlined in (3.3), more accelerators will be added to the RazorCam system to optimize the Software Implementation method described in (3.2).

The FPGA internal architecture for the RazorCam can be seen below in (Figure 22). It utilizes the ARM CPU, simple Bus for reading and writing from the image sensor to DRAM as well as output to the TFT display, and a hardware acceleration chain for convolution modules (such as ROI Compression in the example) that are managed by the CIDA. In (Figure 22), the basic dataflow of the RazorCam can be seen by following along the purple arrows. First, image frames are read from the image sensor and go through color conversion to greyscale. Next, the image is passed through any hardware accelerated convolution modules that have been added to the system, and finally to the bus where it can be passed from the writer FIFO to the reader FIFO and displayed by the TFT display.

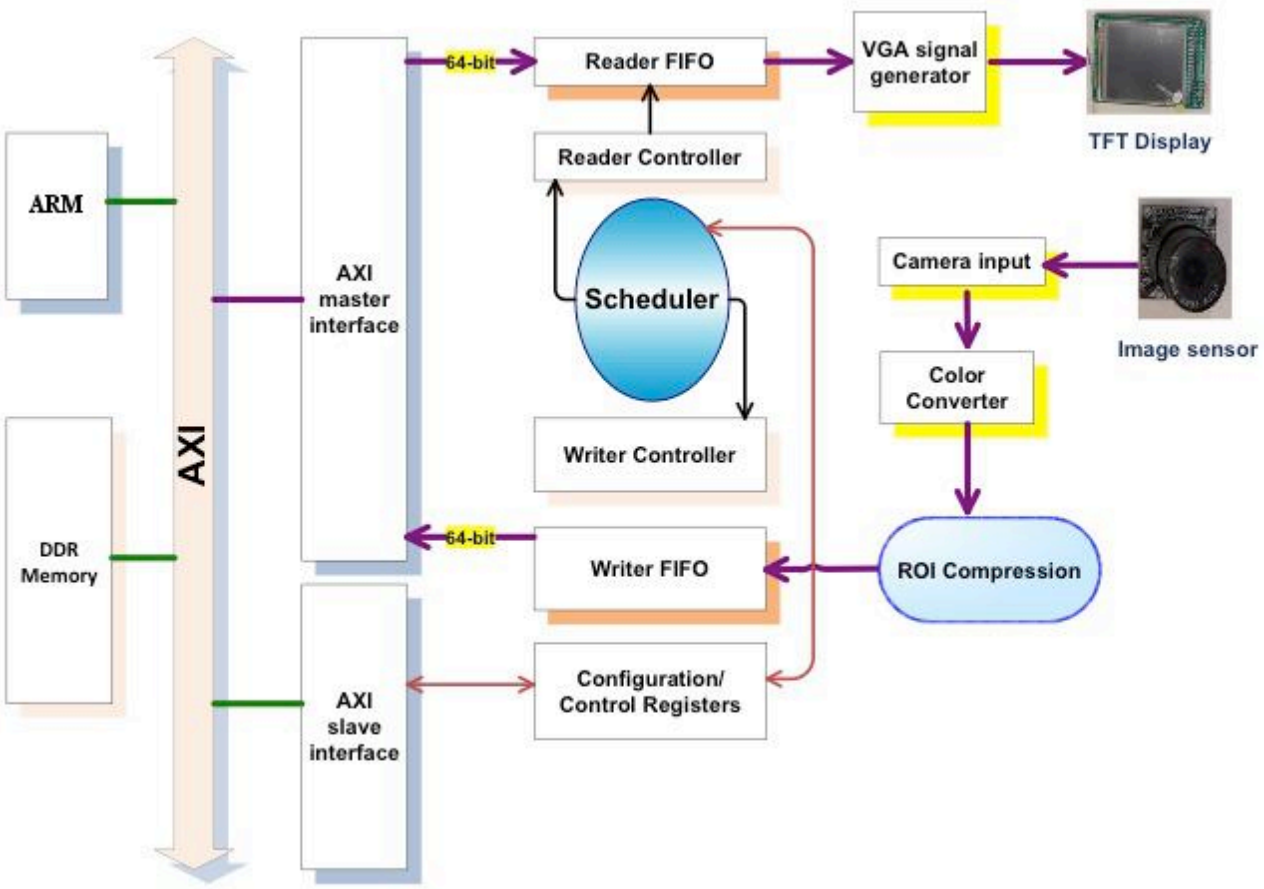

Figure 22: Basic RazorCam Internal FPGA Architecture [36]

3. APPROACH

3.1 High-Level Design

On the following page, seen in (Figure 23), is a dataflow diagram of the final high-level design of my approach. This design and functionality will be explained in depth, step-by-step, in section 3.2. In section 3.3, the development process of the software implementation will be explained, and finally in section 3.4 porting of the software to the RazorCam embedded system on chip will be explained.

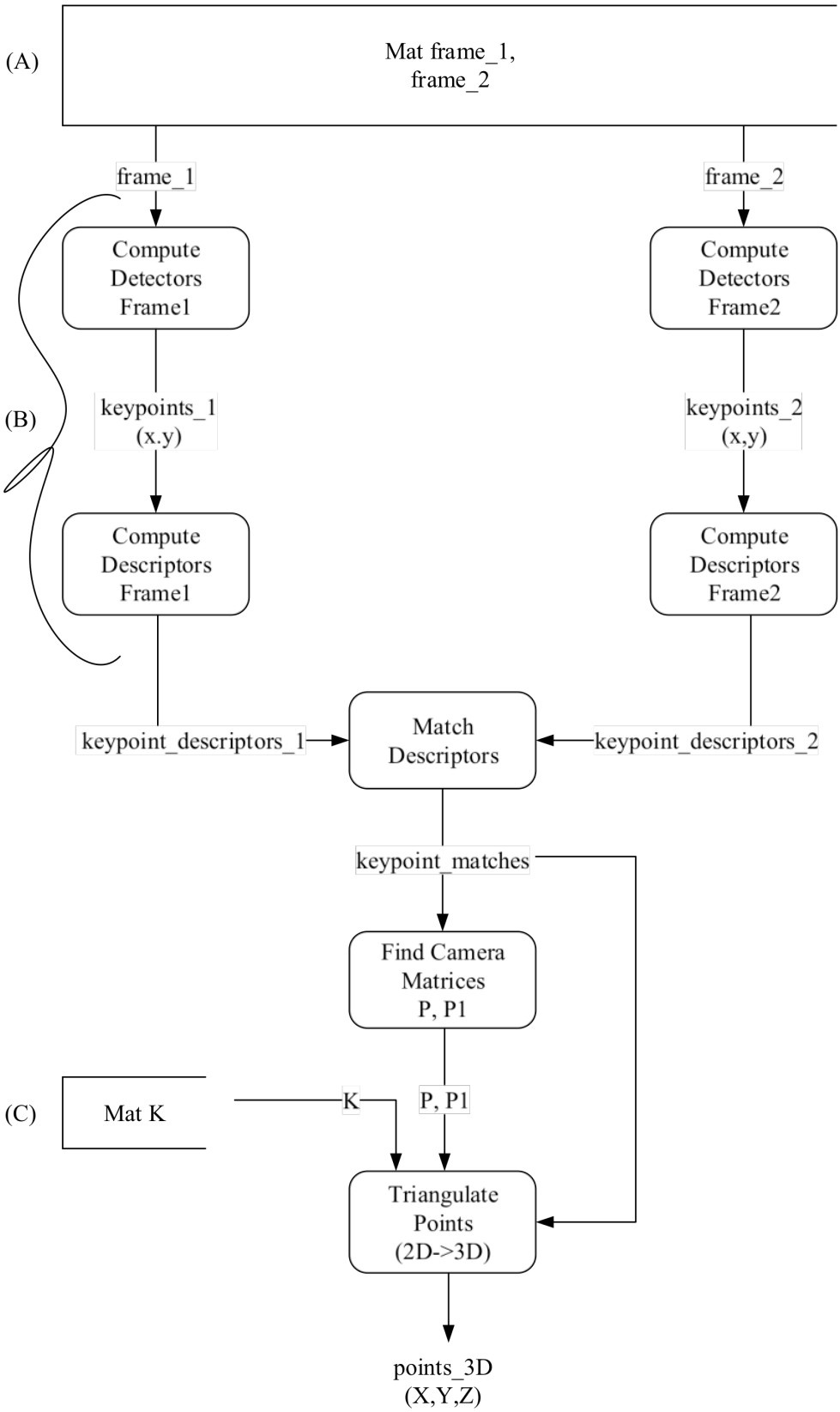

Figure 23: High-Level Design Dataflow Diagram

3.2 Software Implementation

The Software Implementation was coded in C++ using the OpenCV library for reasons explained previously in section 2.1.8. Another reason for choosing OpenCV is its portability to the RazorCam platform, for rapid prototyping, as well as it's relative user-friendliness. The implementation follows an approach for Structure from Motion using sequential Monocular Camera frames for input, as outlined in [11] and [12].

The first step is to read two consecutive frames from the monocular camera device. This was simple; utilizing OpenCV's powerful support for webcams it was as easy as plugging in the device and activating it with the VideoCapture object. After two consecutive frames have been read from the camera, Keypoints are detected for each frame. In this step, either SIFT, SURF, BRISK, or any other Keypoint Detection algorithm can be utilized. BRISK is what was chosen for the final implementation, for reasons outlined in section 2.1.9.1. SIFT and SURF are also viable options so were studied for the Results in section 4.2. Keypoint Detection, seen in part (B) of the diagram in (Figure 23), can be performed in parallel for each frame, increasing overall performance of the Approach. Once Keypoints have been obtained, the Keypoint Descriptors may be calculated using either SIFT, SURF, BRIEF, or Optical Flow descriptors. This step, seen in part (B) of the diagram in (Figure 23), may also be performed in parallel for each keypoint detected in each frame to increase performance. The Keypoint Descriptors are then used to match Keypoints between each frame; this can be done using either brute-force matching or cross-check matching as explained in section 2.1.9.3. Once these matches are found, the best matches can be filtered later, if desired, to obtain stronger results.

Now that a set of optimal matched Keypoints between two consecutive frames has been obtained, the next step is to find the camera matrices P and P1. These matrices essentially allow

the programmer to know the orientation of the camera in each individual frame, which is needed for Triangulation. To obtain P and P1, the Fundamental Matrix must be calculated. This is conveniently implemented in OpenCV with the findFundamentalMat function, which takes as input each set of matched Keypoints. Once the Fundamental Matrix has been found, the Essential Matrix can be calculated allowing P and P1 to be found as per the method outlined by Hartley and Zisserman [12].

At this point, a set of optimal matched Keypoints has been obtained, as well as the camera matrices P and P1. Since we have calibrated the camera beforehand to obtain K, as seen in part (C) of (Figure 23), everything is now known that is needed for triangulation. Using the Linear Triangulation method discussed in Section 2.1.14, a Z-coordinate is calculated for each set of matching Keypoints. The (x,y) coordinates, in this approach, are taken to be that of the second input frame for the final output. Now that this depth information has been calculated, for the software implementation, a simple Depth-Map can be viewed in real-time. A screenshot of the output can be seen in (Figure 24) below, where each individual dot represents a triangulated three-dimensional point. This information could now be used to implement a UAV Navigation system based off of depth information calculated from the Monocular camera and OpenCV. The performance and accuracy of this method will be discussed in-depth in the Results section 4.3.

Figure 24: Software Implementation 3D Point Cloud Output using SURF Keypoints

3.3 Software Development Process

The software development process of the algorithm described in section 3.2 began by first selecting a method for 3D reconstruction. This process consisted of researching various methods and their application to an embedded system for use on UAVs; the conclusion was reached in section 2 that a monocular camera based system should be utilized due to the SWAP constraints imposed on small quad-rotor based UAVs. Once this design decision was made, the next step was to decide on an algorithm for calculating the 3D depth information needed for 3D reconstruction from 2D images taken using the monocular sensor. From the available methods, some of which were briefly discussed in section 2.2.2, structure from motion (SfM) was chosen to perform the task. Since OpenCV was chosen from the beginning as the implementation language, SfM was perfect because [11] ensured the method was implementable using exclusively OpenCV while [12] ensured the math behind it was sound and provable. The method also utilized sequential images taken using a monocular camera – so it was completely compatible with the design decisions made so far.

Once structure from motion was chosen as the method for 3D depth information calculation, work began on actually implementing this method in software using OpenCV. Following the basic dataflow for SfM, presented in section 2.1.12 (Figure 17), the first step was to detect, describe, and match keypoints in two sequential image frames taken from the monocular camera sensor. First, the frames had to be obtained; this was as simple as using the VideoCapture object built into OpenCV. Once two sequential frames are obtained from the camera, Keypoints are detected using the algorithm of choice; the three specific descriptor/detectors studied are discussed in section 2.1.9. The keypoint descriptors for each frame's set of keypoints are calculated next. These two steps are as easy as using detect and

describe functions built into each detectors OpenCV implementation and can be performed in parallel for each frame. At this point, two sets of described keypoints have been obtained, each corresponding to one image frame; now, they must be matched. This is done using the BFMatcher object built into OpenCV; when matching between SIFT or SURF keypoints the hamming distance BFMatcher function must be used, but with BRISK normal brute force matching is fine. The matches can now optionally be filtered based on a certain matching threshold; this value can be adjusted within the BFMatcher object instantiation.

After a set of optimally matched keypoints was obtained, development began on the next step of SfM: calculation of the fundamental and essential matrices. This was done using the FindFundamentalMat function built into OpenCV, which simply outputs a Mat object containing F, calculated using the matched Keypoints obtained in the previous step. The math behind obtaining F is described in [12] and section 2.1.13.1; E can be obtained directly from F and is done so based on math described in section 2.1.13.2 and [12]. After E is derived from F, the camera matrices P and P1 must be derived from E; the math for this is described in 2.1.13.2 and a full proof can be seen in [12]. Since this functionality was not implemented already in OpenCV it had to be coded by hand, based on examples found in [11]. Verification of the results of this function is admittedly difficult – it is almost impossible to know what the actual values of P and P1 technically "should" be; at this point the math was starting to get a little overwhelming but it was time to press forward into the next step: coding triangulation.

The triangulation function built into OpenCV utilizes simple ray intersection, which [12] stated was almost completely unreliable, so a function had to be hand-coded for this step. After studying different triangulation methods laid out in [12], I chose to implement linear triangulation because [11] stated it was possible to accomplish within the OpenCV framework

even though it was not a standard function. The linear triangulation function essentially consists of constructing one matrix and one vector (corresponding to the inhomogeneous system of equations that must be solved by least squares method). Once these two structures are created, using data from P and P1 and the keypoint matches obtained in a previous step, they are solved using the Solve function built into the C Math library. This triangulation is performed for every matched keypoint pair obtained in the first step; the output is a set of 3D points of the form (x,y,z). This set of 3D points can be visualized in a variety of ways; a 3D point cloud was chosen to display the output within the OpenCV framework and was the final step in development before testing and analysis began. This was accomplished in code by simply drawing each triangulated 3D point onto the second frame used in calculation.

3.4 Software Porting to RazorCam

To emulate a production embedded system that could potentially fly on-board a UAV, the RazorCam platform was chosen to prototype the implementation outlined in section 3.2. Since the RazorCam system was designed for OpenCV portability, it made sense to utilize this for a rapid prototype. Once the OpenCV Software Implementation code was ported to the RazorCam, a few minor changes had to be made to the code. Function calls were added to account for the fact that frames, seen in (A) of (Figure 23), would be pulled from the RazorCam FPGA DRAM now instead of the PC webcam the software implementation utilized. Once the right code was added, everything ran but did not perform fast enough for use in a real-time system. It was obvious that some type of hardware/software partitioning was going to be necessary to increase the performance of the system, but this was out of the scope of this Thesis objective so was not pursued. The candidates for hardware partitioning have been found, however, through software profiling that will be discussed next.

Software profiling of the code was performed using Zoom profiling software, the output of which can be seen in (Figure 25). After a minute of profiling a continuous loop of the software implementation, it was reported that the SURFInvoker function was called 46.3% of the time, cv::calcLayerDetandTrace was called 21.5% of the time, ResizeAreaInvoker was called 14.1%, and everything else registered at less than 0.1%! Profiling revealed that the steps using the most execution time was obviously Keypoint detection and description (in the case of this specific profile, SURF was used), seen in (B) of (Figure 23) since all three of these functions serve part of that purpose. So, these functions are the 'critical sections,' prime candidates for a hardware acceleration module for the RazorCam system.

Though a properly functioning Keypoint detection acceleration module has not been implemented, a proposed modification to the RazorCam FPGA internal architecture and dataflow can be viewed in (Figure 26). First, images are acquired by the image sensor, and then sent to the bus (after any optional convolutions, such as grayscale conversion) where they are stored in DRAM. After two successive image frames have been stored to memory, the acceleration module (K) in (Figure 26) can begin detecting the Keypoints in each frame as well as descriptors for each that are then stored in DRAM. Once Keypoint descriptors are detected, matching is performed in the ARM processor, the results of which are stored. Now that keypoint matches have been found, the camera matrices must be calculated; all this info is finally stored in DRAM. Finally the ARM processor can perform triangulation by accessing the stored matches and camera matrices from DRAM, storing the final results back in memory. The triangulation results are then grabbed by the ARM processor, written to the last frame grabbed by the camera sensor, output to the bus and then finally the TFT display for viewing.

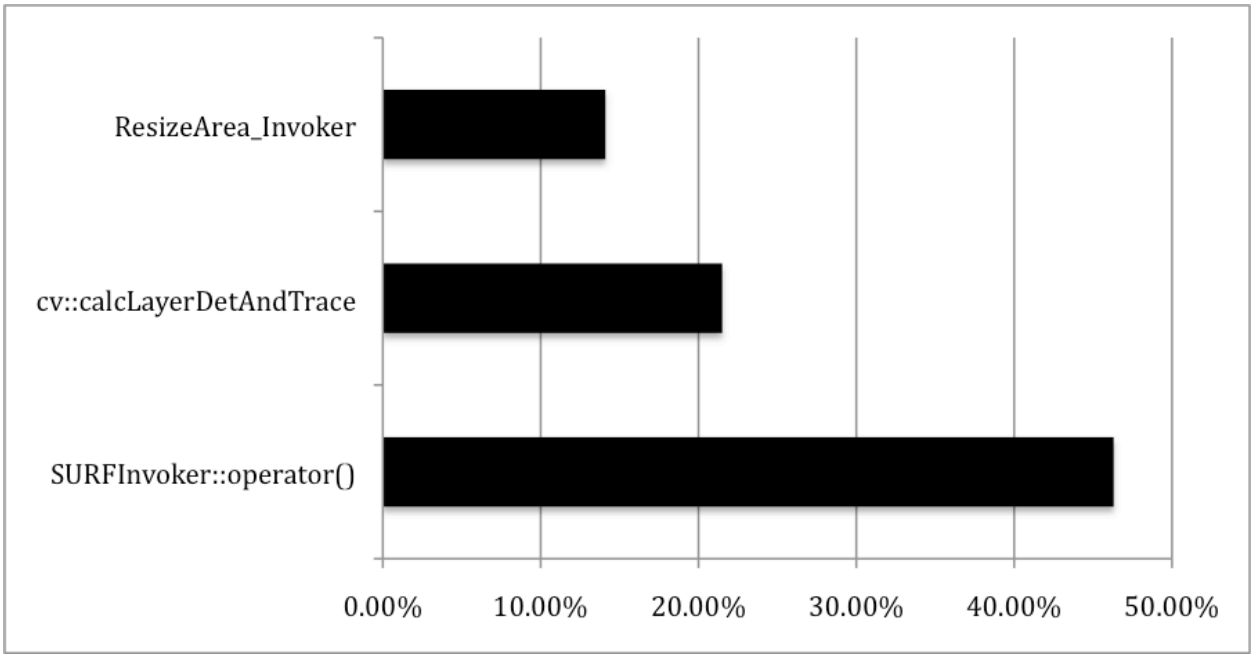

Figure 25: Software Implementation Profiling Results (using SURF Keypoints)

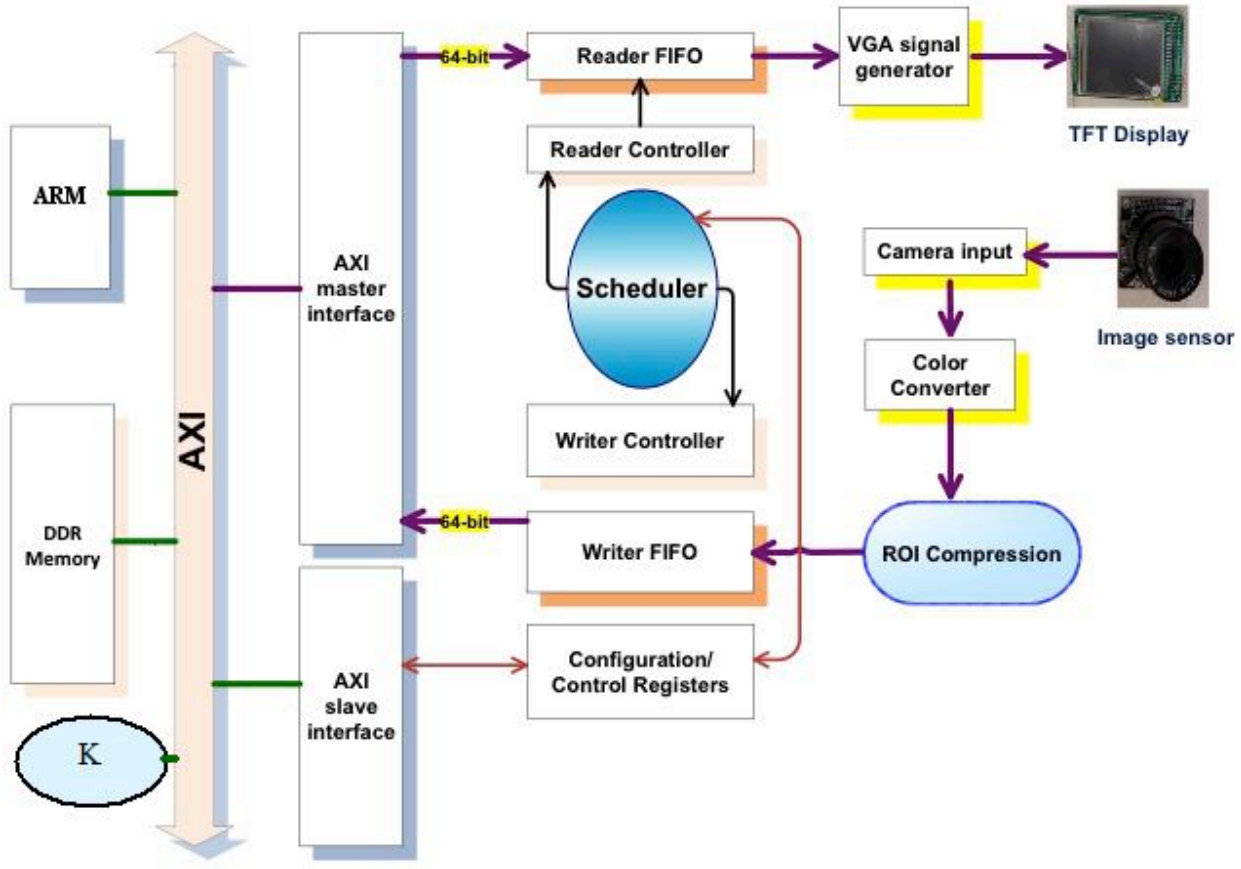

Figure 26: Proposed RazorCam Internal FPGA Architecture for Accelerated Keypoint Detection based on [36]

4. RESULTS AND ANALYSIS

4.1 Methodology

For testing, the software implementation was first evaluated on an Intel 4770K CPU running Ubuntu Linux 12.04 then ported to the RazorCam Embedded Smart Camera running on the Xilinx Zynq-7000's onboard ARM CPU. Each specific platform was tested using every chosen keypoint detection and description algorithm individually: SIFT, SURF, and BRISK. The results of the testing can be seen in the table below in section 4.2 and are analyzed in 4.3.

To test the performance of the software implementation, the total execution time of a single iteration of the algorithm was collected twenty five times then the average of that sample reported as the performance time. To test the accuracy of the software, various depth measurements between the sensor and a simple object were taken, averaged, and compared to the actual measured distance. These depth measurements were also taken with varying lengths of space (moving left in respect to the object being measured) between frames to test whether or not the disparity between camera matrices would have any effect on the accuracy. The accuracy measurements were all taken using SURF keypoints, since they proved to be the best performers in the first round of evaluations, on the same PC hardware running Intel 4770K CPU.

4.2 Results

The results of performance evaluation of the software implementation can be seen in (Figure 27) below and results of accuracy evaluations can be viewed in (Figure 28). In (Figure 27), the table displays the execution time, in seconds, for a single iteration of the algorithm when tested with different keypoint detector/descriptors on each platform (PC and RazorCam). The

table in (Figure 28) reports the distance calculated by the software to a simple object placed in front of the sensor at certain pre-measured lengths.

Hardware/CPU Used in Testing	Keypoints Used for Calculation		
	SIFT	SURF	BRISK
PC - Intel 4770K	0.261 sec	0.225 sec	0.301 sec
RazorCam – ARM Cortex-A9	4.6 sec	4.4 sec	5.2 sec

Figure 27: Software Performance Evaluation Results (in Seconds)

Real Distance Between Object and Camera Sensor	Disparity Between Frames					
	0 m	0.05 m	0.1 m	0.15 m	0.25 m	0.3 m
0.10 m	0.357	0.345	0.332	0.327	0.321	0.338
0.20 m	0.685	0.676	0.668	0.659	0.654	0.623
0.33 m	1.170	1.122	1.080	1.031	0.992	0.924
0.50 m	1.532	1.455	1.420	1.377	1.286	1.267
0.75 m	2.080	2.012	1.936	1.877	1.754	1.721
1.00 m	2.591	2.534	2.476	2.421	2.335	2.277
2.50 m	7.082	7.035	6.987	6.953	6.895	6.879
5.00 m	12.070	12.044	12.012	11.987	11.910	11.882
10.0 m	21.074	21.055	21.037	21.022	20.990	20.966
15.0 m	34.534	34.511	34.497	34.472	34.438	34.425

Figure 28: Software Accuracy Evaluation Results (in Meters)

4.3 Analysis

The performance results show that it is possible to implement a 3D reconstruction algorithm using OpenCV on a desktop system with a general purpose CPU as well as an embedded system on chip using an ARM CPU. The software implementation running on a general purpose CPU (Intel 4770K) works great in real time. The problem here is, this computer cannot be attached to the UAV with SWAP constraints. That is why it is important to port this

code to the RazorCam, because that is a computer that could actually fit on the UAV However, the performance results are dismal for this method running on the RazorCam; not even close to real-time performance. Both of these implementations are okay for proof-of-concept, but for a real-life system on chip it would be optimal for the algorithm to perform in real-time without so much delay in execution time.

If a hardware accelerator for keypoint detection could be fully implemented, based on the software profiling discussed in section 3.4, the software implementation could potentially run in real-time on the RazorCam system on chip, making it a viable solution for the backbone of a 3D reconstruction system. At the very least, these results demonstrate the proof-of-concept and that a rapid prototype can be deployed using OpenCV and the RazorCam. Although the system is not fully accelerated yet, continued research can be done to pursue a hardware acceleration module for the keypoint detection and description functions.

The accuracy results show that this may not be the most optimal method for 3D reconstruction. Although increasing the disparity between frames and increasing the distance the object is away from the sensor both slightly improve the accuracy, the percentage of error is still quite high either way. Increasing the disparity between frames improves the results because it allows for a more accurate calculation of F and therefore E, P, and P1 (similar to stereo vision methods that require a certain disparity between the camera sensors to function properly). Increasing the distance between the object and sensor increases the accuracy to a point, but will eventually decrease the accuracy as it passes a certain threshold (obviously, as the object would be less and less visible as it moved away). This also has to do with the specific camera lens being used, as the exact field of view and intrinsic parameters vary from camera to camera. Upon going back through the algorithm to find possible accuracy problems caused by bad data or

miscalculation, no problems could be found within any of the operations. The only doubts I have about the actual implementation lie within the Triangulation function. It is stated in [12] that linear triangulation is not their preferred method; another method coined ‘optimal triangulation’ is presented in [12] that is much more complex and is not easily implementable within the OpenCV framework. Implementing this method in future work has a good possibility of fixing these accuracy problems and bringing the percent error down to an acceptable value for a real-time system. The results demonstrate the proof-of-concept, though, and that a 3D reconstruction algorithm can be developed in OpenCV and ported to an embedded system on chip for potential real-time use.

5. CONCLUSIONS

5.1 Summary

In this Thesis, a method was presented to calculate depth information from keypoints in two consecutive image frames using a monocular camera sensor as input and the OpenCV library for implementation. This method was first implemented in software and run on a general-purpose CPU, and then ported to the RazorCam Embedded Smart-Camera System and run on an ARM CPU. After profiling was performed on the software implementation, critical sections of the algorithm were identified for potential hardware acceleration and a proposed hardware partitioning for the RazorCam Embedded Smart Camera system on chip was presented. The results of performance and accuracy testing of the software implementation were not promising, however, demonstrating slow speed in pure software on the RazorCam and an unacceptable percentage of error.

5.2 Potential Impact

The potential impacts of this work will be seen through the continuation of this work in the Smart ES lab at University of Arkansas. This method provides the basic depth information which would be the backbone of a UAV navigation system, as well as a way to obtain it in real-time on a small, low-power embedded system. Building upon this work, many potential impacts could be realized beyond just UAV navigation; this technology could easily be applied to other vehicles such as cars, trains, or boats. This technology is much easier to integrate into existing products than other solutions, such as LiDAR, which must have a 360-degree view around the vehicle, is bulky, and requires very high power. If pursued further, the work presented here could

potentially be commercialized in a few years time with enough manpower and investment; the field of UAV is exploding with increasing use being seen in the civilian sector.

5.3 Future Work

This thesis provides the basic framework to begin creating a UAV navigation system. Basic depth information is calculated using a Monocular Camera in real-time, but the information is very basic. In future work, this depth information's accuracy could be improved by potentially using a better triangulation algorithm. The work presented here could also be improved by further hardware/software partitioning of the algorithm; perhaps hardware acceleration of Keypoint Detection or Descriptor calculation is realizable. Any future work would essentially center on improving the accuracy and performance of the approach outlined in this thesis, as well as developing it into a more full-fledged navigation system besides just providing basic depth information for certain points.

REFERENCES

[1] C. Harris and M. Stephens. "A Combined Corner and Edge Detector."*Alvey Vision Conference*. 15 (1988).

[2] D. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints." *International Journal of Computer Vision* 60, no. 2 (2004): 91-110.

[3] S. Leutenegger, M. Chli, and R. Y. Siegwart. "BRISK: Binary Robust Invariant Scalable Keypoints." *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011.

[4] I. Barandiaran, M. Graña, and M. Nieto. "An Empirical Evaluation of Interest Point Detectors." *Cybernetics and Systems* 44.2-3 (2013): pp. 98-117.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. "Speeded-Up Robust Features (SURF)." *Computer Vision and Image Understanding* 110, no. 3 (2008): 346-359.

[6] M. Trajković and M. Hedley. "Fast Corner Detection." *Image and Vision Computing,* 16.2 (1998): pp. 75-87.

[7] J. Matas *et al.* "Robust Wide-Baseline Stereo from Maximally Stable Extremal Regions." *Image and vision computing* 22.10 (2004), pp. 761-767.

[8] M. Agrawal, K. Konolige, and M. R. Blas. "Censure: Center surround extremas for realtime feature detection and matching." *Computer Vision–ECCV 2008*. Springer Berlin Heidelberg, 2008, pp. 102-115.

[9] E. Rublee *et al.* "ORB: an efficient alternative to SIFT or SURF." *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011.

[10] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. "KAZE Features." *Computer Vision–ECCV 2012*. Springer Berlin Heidelberg, 2012, pp. 214-227.

[11] D.L. Baggio *et al*. “Exploring Structure from Motion Using OpenCV,” in *Mastering OpenCV with Practical Computer Vision Projects.* Birmingham, UK: Packt Publishing Ltd, 2012, ch. 4.

[12] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[13] D. Yoo, D. Won, and M. Tahk. "Optical Flow Based Collision Avoidance of Multi-Rotor UAVs in Urban Environments." *International Journal of Aeronautical and Space Sciences* 12, no. 3 (2011): pp. 252-259.

[14] D. Howden and T. Hendtlass. "Collective Intelligence and Bush Fire Spotting." In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 41-48. ACM, 2008.

[15] H. Bendea, *et al.* "Mapping of Archaeological Areas Using a Low-Cost UAV. The Augusta Bagiennorum Test Site." In *XXI International CIPA Symposium*, pp. 01-06. 2007.

[16] A. Samad *et al.* "The Potential of Unmanned Aerial Vehicle (UAV) for Civilian and Mapping Application." In *System Engineering and Technology (ICSET), 2013 IEEE 3rd International Conference*, pp. 313-318.

[17] F. Caballero *et al.* "A Visual Odometer Without 3D Reconstruction for Aerial Vehicles. Applications to Building Inspection." In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference*, pp. 4673-4678.

[18] N. Metni and T. Hamel. "A UAV for Bridge Inspection: Visual Servoing Control Law with Orientation Limits." *Automation in Construction* 17, no. 1 (2007): pp. 3-10.

[19] L. Zhengrong *et al*. "Knowledge-Based Power Line Detection for UAV Surveillance and Inspection Systems." In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, pp. 1-6. IEEE, 2008.

[20] D. Hausamann *et al*. "Monitoring of Gas Pipelines–a Civil UAV Application." *Aircraft Engineering and Aerospace Technology* 77, no. 5 (2005): pp. 352-360.

[21] P. Doherty and P. Rudol. "A UAV Search and Rescue Scenario with Human Body Detection and Geolocalization." In *AI 2007: Advances in Artificial Intelligence*, pp. 1-13. Springer Berlin Heidelberg, 2007.

[22] T. Tomic *et al.* "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue." *Robotics & Automation Magazine, IEEE* 19, no. 3 (2012): pp. 46-56.

[23] A. Puri. "A Survey of Unmanned Aerial Vehicles (UAV) for Traffic Surveillance." *Department of Computer Science and Engineering, University of South Florida,* 2005.

[24] A. Puri, K. P. Valavanis, and M. Kontitsis. "Statistical Profile Generation for Traffic Monitoring Using Real-Time UAV Based Video Data." In *Control & Automation, 2007. MED'07. Mediterranean Conference on*, pp. 1-6. IEEE, 2007.

[25] L. Li *et al*. "Multi-Objective Optimization Model And Evolutionary Algorithm To Plan UAV Cruise Route For Road Traffic Surveillance." In *Transportation Research Board 92nd Annual Meeting*, no. 13-0735. 2013.

[26] L. Wallace *et al*. "Development of a UAV-LiDAR System with Application to Forest Inventory."*Remote Sensing* vol. 4, no. 6 (2012): pp. 1519-1543.

[27] Y. Lin, J. Hyyppa, and A. Jaakkola. "Mini-UAV-Borne LiDAR for Fine-Scale Mapping." *Geoscience and Remote Sensing Letters, IEEE* 8, no. 3 (2011): pp. 426-430.

[28] J. Park and Y. Kim. "3D Shape Mapping of Obstacle Using Stereo Vision Sensor on Quadrotor UAV." *AIAA Guidance, Navigation, and Control Conference,* January 2014. Seoul National University, 2014.

[29] K. Schmid *et al.* "Stereo Vision Based Indoor/Outdoor Navigation for Flying Robots." In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3955-3962. IEEE, 2013.

[30] D. Magree, J. G. Mooney, and E. N. Johnson. "Monocular Visual Mapping for Obstacle Avoidance on UAVs." *Journal of Intelligent & Robotic Systems* 74, no. 1-2 (2014): pp. 17-26.

[31] T. Mori and S. Scherer. "First Results in Detecting and Avoiding Frontal Obstacles from a Monocular Camera for Micro Unmanned Aerial Vehicles." In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1750-1757. IEEE, 2013.

[32] J. Lee *et al*. "Obstacle Avoidance for Small UAVs Using Monocular Vision." *Aircraft Engineering and Aerospace Technology* 83, no. 6 (2011): pp. 397-406.

[33] M. Brown, R. Szeliski, and S. Winder. "Multi-Image Matching Using Multi-Scale Oriented Patches." In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 510-517. IEEE, 2005.

[34] H. Yu and R. Beard. "A Vision-Based Collision Avoidance Technique for Micro Air Vehicles Using Local-Level Frame Mapping and Path Planning."*Autonomous Robots* vol. 34, no. 1-2 (2013): pp. 93-109.

[35] M. Mefenza, F. Yonga, and C. Bobda. "Razorcam: An Embedded Platform for Image Processing." ASEE Midwest Conference, 2013.

[36] M. Mefenza, F. Yonga, and C. Bobda. “RazorCam: A Prototyping Environment for Video Communication,” ACM HotMobile 2013 Poster. *SIGMOBILE Mobile Computing Commun. Rev*., 17(3): pp. 13–14, November 2013.

[37] M. Mefenza. “Design and Verification Environment for High-Performance and Secure Video-Based Embedded Systems.” Smart Embedded Systems Laboratory, Computer Science and Computer Engineering Department, University of Arkansas, December 2013.

[38] B. Rinner et al. "The Evolution from Single to Pervasive Smart Cameras." In *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, pp. 1-10. IEEE, 2008.

[39] J. Schlessman et al. "Hardware/Software Co-Design of an FPGA-Based Embedded Tracking System." In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pp. 123-123. IEEE, 2006.

[40] M. Bramberger et al. "Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance." In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pp. 174-181. IEEE, 2004.

[41] F. Dias et al. "Hardware, Design and Implementation Issues on a FPGA-Based Smart Camera." In *Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on*, pp. 20-26. IEEE, 2007.

[42] H. C. Longuet-Higgins. "The Reconstruction of a Plane Surface from Two Perspective Projections." *Proceedings of the Royal Society of London. Series B. Biological Sciences* 227, no. 1249 (1986): 399-410.

[43] Q. T. Luong and O. D. Faugeras. "The Fundamental Matrix: Theory, Algorithms, and Stability Analysis." *International Journal of Computer Vision* 17, no. 1 (1996): 43-75.

[44] Z. Zhang and G. Xu. "A General Expression of the Fundamental Matrix for Both Perspective and Affine Cameras." In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence-Volume 2*, pp. 1502-1507. Morgan Kaufmann Publishers Inc., 1997.

[45] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.

[46] C Bobda. *RazorCopter.* Digital Photograph. Used with Permission <smartes.uark.edu>

[47] R. Best. *MQ-9 Returns from Winter Training Mission*. March 6, 2012. Digital Photograph. United States Air Force, Public Domain. <http://www.hancockfield.ang.af.mil/photos/mediagallery.asp?galleryID=5774&page=1>

[48] T. Tschida. *NASA Altair Predator B with Payloads for NOAA-NASA Flight Demonstration*. April 20, 2005. Photograph. National Aeronautics and Space Administration, Public Domain. <http://www.dfrc.nasa.gov/Gallery/Photo/Altair_PredatorB/HTML/EC05-0090-19.html>

[49] AeroVironment, Inc. *Nano Hummingbird.* 2011. Digital Photograph. Used Under Non-Commercial License Courtesy of AeroVironment, Inc. < http://www.avinc.com/media_gallery/>