Real-time Monocular Depth Mapping System
Using Variance of Focal Plane and Pixel Focus Measure

by Paruku Paerhati

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2017

Author: _____
Department of Electrical Engineering and Computer Science
May 22, 2017

Certified by: _____
Kamal Youcef-Toumi, Professor, Thesis Supervisor
May 22, 2017

Accepted by: _____
Christopher Terman, Chairman, Masters of Engineering Thesis Committee

Real-time Monocular Depth Mapping System
Using Variance of Focal Plane and Pixel Focus Measure

by

Paruku Paerhati

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2017, in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Vision is one of the most powerful senses available to creatures. Undoubtedly, many of the fundamental operations of humans, such as the ability to plan paths, avoid obstacles, and recognize objects, depend heavily on their visual perception of the world around them. Although humans have naturally evolved to efficiently use their stereo optical prowess to develop an understanding of their environment, artificial machines and systems in comparison have just begun to utilize computer vision to create awareness of local physical entities. One of the most important sensory skills creatures have is depth perception, which allows them to estimate the relative distance of objects in their vision from many visual cues. Many systems have been developed to aid machines in perceiving the depth map of their environment, and each system has its drawbacks and benefits. In this paper, we introduce the design and implementation of a new system which provides a depth map from the use of a single optical camera with focal plane variation in the images taken. The paper focuses on the methods used to scale the depth from focus algorithm to perform in real-time. The results also showcase a real-time depth mapping system capable of providing rich depth maps of scenes at a high framerate and with advanced noise filtration techniques.

Thesis Supervisor: Kamal Youcef-Toumi
Title: Professor

Acknowledgements

First and foremost, I would like to begin by thanking the three most important individuals who have guided my research and assisted me in bringing this project to fruition. I thank my thesis supervisor Professor Kamal Youcef-Toumi, who is an extraordinarily intelligent, humble, and wonderful professor, in aiding me with this project in many ways. Despite having an extremely demanding schedule as a professor at MIT, he always made himself readily available for any meetings with me to help give feedback and improve my work. I thank Iman Soltani Bozchalooi, a Postdoc within the same Mechatronics Research Laboratory, in always assisting me with the technical aspects of the project. Not only was his previous research in the area of monocular depth-mapping fundamental to my project, he always gave friendly technical assistance and guidance in order to ensure I was on the right path. Last but not least, I would like to thank Mohsen Lakehal-Ayat and those at Ford for providing the funding for this project and again assisting with any difficulties I faced.

Contents

1 Introduction

The primary focus of this research was to design and implement a real-time depth mapping system capable of providing rich depth frames at reasonable frame rates. Although much of real-time depth mapping has been explored in previous works, our work focuses on a unique implementation by which we only utilize a single monocular camera. Throughout the course of this paper, we will begin by giving the context and problem space our depth mapping solution aims to solve. Then we will showcase the mathematical algorithms and previous research which were utilized to generate individual depth maps. We will then delve into the design of both the software architecture of the implementation as well as the hardware components used to achieve the real-time results. Lastly, we will showcase the results of depth mapping tests run on the system in terms of the system's performance, accuracy, resolution and noise filtering capabilities.

1.1 Context

With the advent of high-speed autonomous systems and our increasing reliance on machines and robotics to perform certain tasks, the performance requirements of sensory systems for said machines have also drastically increased. For many areas of robotics, depth perception can provide vital information to the machine about their surrounding environment. For example, self-driving cars can utilize depth perception to detect potential obstacles such as other vehicles or pedestrians [1]. Drones can use depth perception to not only avoid obstacles but to also create a 3D map of their environment for rescue workers to use in rescue missions. One of the most notable uses of depth maps is

Simultaneous Localization and Mapping (SLAM) which aims to recreate a 3-dimensional representation of the environment [2]. These are just a handful of possible applications of depth mapping systems and further improvements in the area of depth mapping can provide vast benefits to not only machines and robotics but to also human life.

However, the different physical constraints of systems such as self-driving vehicles, drones, and robotics force engineers to weigh the benefits and costs of available sensory systems such as their cost, size, weight, resiliency, and accuracy. For example, a large and heavy Lidar system which works for self-driving cars may not be feasible for drones which have limited payload weights. Thus, in our research, we aim to create a satisfactory depth mapping system which can provide real-time depth maps using a single camera in order to be of possible use to the previously mentioned examples.

1.2 Existing Systems

There is a myriad of depth mapping systems which are in use today. They range from Lidar, Sonar, Radar and optical solutions. For the purposes of limiting the scope of this paper, we will analyze only the depth mapping systems which utilize optical cameras to recreate a depth map. The most common of the ocular depth mapping systems include stereo vision, structure from light (SL), and time of flight (ToF).

1.2.1 Stereo Vision

Stereo vision systems work similarly to how human eyes capture depth information. Generally, two cameras are placed a fixed distance apart and an image is captured in each camera. Because there is a horizontal displacement between the two cameras, the images taken will have horizontal disparities. A computer program then compares each point in the images and attempts to match them to extract

depth information about the point. This process is called the correspondence problem in this field [3]. This process is modeled as an optimization problem and is NP-Complete to solve, thus many systems are too slow when solving the correspondence problem [11]. Although new methods have been created to quicken the process, their use of heuristics and estimations often result in a higher rate of errors and low accuracy [11]. Thus, although stereo vision hardware is generally cheaper than other systems, they tend to have low accuracy, high software complexity, and low frame rates [4].

1.2.2 Structured Light

Structured light systems work by emitting a light pattern on an environment (generally in the infrared spectrum) from a projector and then perceiving the resulting reflection of the pattern on a separate camera. The deformation of the original pattern can then be analyzed to obtain depth information. The most famous implementation of this method is Microsoft's Kinect which utilizes a fixed dot pattern and attains an average of about 30 frames per second on modern GPUs, although it is limited to a resolution of 640 x 480 pixels [9]. However, although structured light systems work differently in comparison to the stereo depth mapping systems, structured light systems also face a correspondence problem where they must compute the correspondence of the pattern on the scene via triangulation. Thus, similar to stereo systems, spatial structured light systems incur a high computational match and thus require the system to sacrifice accuracy or precision in order to attain reasonable levels of framerates [9].

1.2.3 Time of Flight (ToF)

Time of Flight (ToF) systems work by emitting a pulse or sinusoid of light from an emitter, then capturing the reflected light wave with a sensor. Because the light will take time to travel to the object and back to the sensor, there will be a phase shift in the perceived light wave. Using the phase shift, the distance to

each point of light can be extracted. Amongst the depth mapping systems that exist today, ToF systems are the newest. Because ToF sensors require a sensor for every pixel in order to map the reflected light, they often suffer from very low resolution [10]. Furthermore, ToF sensors have only recently become affordable at commodity prices from manufacturers and thus users of ToF sensors are often limited by the calibrations set by the manufacturers. However, the benefit of ToF sensors are that they perform relatively well on texture-less surfaces of materials and have minimal computational requirements in comparison to the other systems [10].

1.3 Depth from Focus

The final class of depth mapping systems we will introduce is called depth from focus systems which serves as the basis for how our system generates its depth maps. Depth from focus systems rely on measuring the degree of focus of individual pixels and utilizing the degree of focus to extrapolate information regarding the distance to the object at the pixel. In essence, in order to generate a single depth map, the system will take images of the same scene at varying focal lengths. Then for individual pixels, if we can find the optimal focal length at which the pixel came into the highest degree of focus, then we can estimate its distance from the camera. There are many advantages to a system which relies on depth from focus to generate its depth maps. These systems do not require motion in the scene. Furthermore, the system is a passive sensory system, which means that no form of light is emitted from the system in order to be captured and analyzed unlike the previously mentioned optical solutions. However, due to the system being comprised of only passive cameras, generally lighting conditions will have a significant effect on the accuracy of the depth maps. In addition, depth from focus systems are generally easy to scale and to reduce the runtime on a highly parallelized system. Lastly, the use of a single low resolution camera can greatly reduce cost and size in comparison to other depth mapping

systems. For these reasons, the depth from focus method was chosen as the method by which we generated real-time depth maps of the environment in our system.

2 Depth from Focus

The depth from focus algorithms used in this system is based heavily on the previous work performed by the Mechatronics Research Laboratory at MIT [5,6]. Nevertheless, in this section we will briefly describe the algorithm used in order to convert raw images to depth maps.

In comparison to the other optical solutions, the depth from focus method is a more parallelizable and scalable algorithm [6]. However, there are key components which must be discussed before attempting to implement such a method. How does one measure the degree of focus of an individual pixel? In this section, we will discuss in detail the mathematical algorithms used in generating depth information from a number of images taken at varying focal lengths.

2.1 Focus Measure

A vital step in performing the depth from focus method is determining the degree of focus of individual pixels in the raw image. As we take images at varying focal lengths, the goal of the depth from focus method is to determine for each pixel, at which focal length the pixel came into the highest degree of focus. Thus a reliable and performance scalable method of scoring the degree of focus is necessary for the degree of focus system to succeed. Many different algorithms were tested for their performance and accuracy for the focus measure of individual pixels in the past research performed at the Mechatronics Research Laboratory at MIT [5, 6]. In the end, the most reliable methods found were to either perform a type of Laplacian based operator or to utilize a form of wavelet decomposition on raw images using one of the Daubechies wavelets [6].

2.1.1 Wavelet Decomposition

The method known as wavelet decomposition has been researched thoroughly in academia. The most common and poignant use case of wavelet decomposition in industry has been in the field of image compression. One of the emerging contenders for image compression standards, the JPEG 2000 format, utilizes wavelet decomposition to compress images to a lower size while minimizing loss in the image details [7]. However, wavelet decomposition can also be utilized to find and highlight details and areas of high frequency in the signal and thus was found to be also useful for representing regions of high frequency of focus in the images [5,6]. In this section, we briefly describe how wavelet decomposition works.

Wavelet decomposition is the method of analyzing a signal and turning it into a form of an orthonormal wavelet series representation based on an overall mother wavelet. These wavelets are similar to sinusoidal waves used in Fourier transforms, and form the basis of the signal representation. However, in contrast to a Fourier transform, the wavelets used in the wavelet transform or decomposition are finite signals with finite lengths. Theses individual wavelets can be created by scaling and shifting the original mother wavelet. Thus, because the individual wavelets are transient signals, the wavelet decomposition method generally performs better in capturing the transient and localized behavior of the signal than Fourier-transforms [6]. Specifically, the Daubechies wavelets were chosen as the mother wavelet for the wavelet decomposition due to their ability to better represent localized frequencies in the signal.

In order to apply wavelet decomposition to a discrete signal, we use a technique called the discrete wavelet transform (DWT). The discrete wavelet transform performs essentially a cascade of band pass filters on the signal in order to separate the signal into its frequency sub-bands [6]. In order to perform

the filtration, the high-pass and low-pass band filter's coefficients are given by the mother wavelet that was chosen. Figure 2-1 illustrates how we take the original discretized image and perform a cascade of low-pass and high-pass filters on the 2-dimensional signal in varying orders of the rows and columns of the signal in order to obtain the final details of the image.

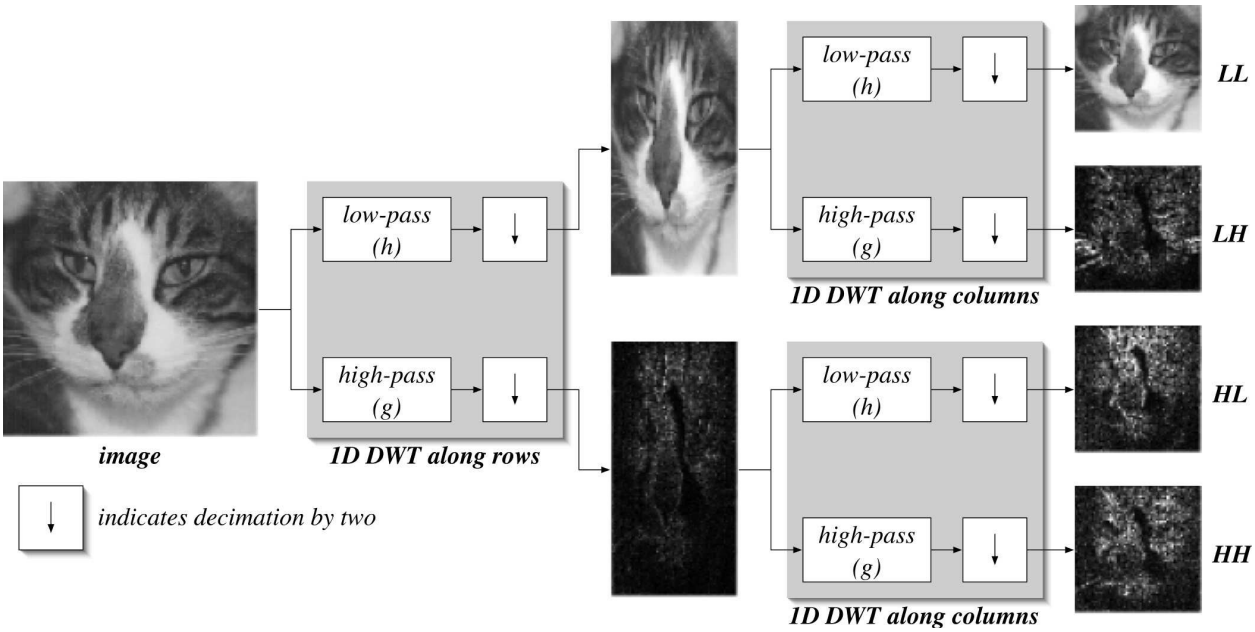


Figure 2-1: An illustration of the discrete wavelet transform on an image of a cat.

2.1.2 Focus Measure Calculation

Treating the raw image as a 2-dimensional signal, the result of performing a wavelet decomposition on the image can be represented as 4 different components as seen in the right of Figure 2-1. The wavelet decomposition returns different details of the raw image. Each "detail" is a result of performing the wavelet transform on the signal in different orders in regards to the rows and columns of the images. The LL details represent a downscaled approximation of the original image. The HL detail represents variations in the x axis (or rows) and thus is referred to as the vertical details of the image. The LH detail represents variations in the y axis (or columns) and thus is referred to as horizontal details. Lastly, the

HH details are the diagonal details in the images. Higher levels of detail in the individual LH, HH, and HL represent locations of high frequency within the signal and thus in our case, sharp contrast in the image.

Discarding the approximate detail (LL) and utilizing the horizontal, vertical, and diagonal details allows us to measure the sharpness or focus of individual pixels in the signal. In order to formulate our degree of focus measure FM matrix for an image of size M pixels, we need simply take the sum of the squares of the horizontal, vertical, and diagonal details of each pixel p given by the wavelet decomposition as shown in equation (2.1).

$$FM[p] = W_{LH}(p)^2 + W_{HL}(p)^2 + W_{HH}(p)^2 \tag{2.1}$$
$$p \in [0, M-1]$$

2.2 Depth Estimation

Now that we have a method of obtaining the focus measurements for individual pixels for raw images, now we must find a way to use that information to obtain a depth map. In the design of our Depth from Focus system, we take N images of the same scene, where each image i is taken at a different focal length f_i and we increase the focal length after each image by f_{step}. We also include an offset parameter f_{offset} to set the total focal range to be within $[f_{offset}, f_{offset} + (n-1)f_{step})]$. In the actual system, N, f_{offset}, and f_{step} are configurations the user can choose. In the simplest way, the focal length for an image i can be calculated as in equation (2.2).

$$f_i = i * f_{step} + f_{offset} \tag{2.2}$$
$$i \in [0, N-1]$$
$$f_i \in [f_{offset}, f_{offset} + (n-1)f_{step}]$$

After each image is taken at a given focal length, we obtain the focus measure matrix of the degree of focus score of individual pixels for said image via the methods described in (2.1). Thus, as a result we have a list of focus measure scores of the pixels of the varying images as represented by (2.3).

$$FM_i, i \in [0, N-1] \tag{2.3}$$

Lastly, we wish to generate a matrix $MaxFM$ which, for each pixel p, stores the image index at which the maximum FM value was found for the pixel as shown below (2.4).

$$MaxFM[p] = argmax_i \, FM_i[p] \tag{2.4}$$
$$p \in [0, M-1]$$

Thus, the above process illustrates how one can choose the parameters in order to take N images, each at varying focal lengths in $[f_{offset}, f_{offset} + (N-1)f_{step}]$. Then compute the focus measure matrices FM_i for each of the N images. Lastly, we must also note that objects positioned closer to the camera will come into focus earlier in our images since for our earlier images we have lower focal lengths. This means that for the closer objects, their maximum focus measure scores FM_i will be higher in earlier images. We can use this notion to extrapolate that the matrix $MaxFM$ can then be used to estimate the distance to the pixel and thus the matrix itself can be used to represent a relational depth map of the scene.

3 Design

In this chapter, we will describe the hardware and software architecture of the system. Specific details will be given regarding the software implementation of the algorithms and the specifications of the hardware, as well as their configurations, that was used in the system. Furthermore, we will analyze the cost and benefit of individual pieces of the hardware that were chosen for the system. The goal of this chapter is to give enough details on the system's design such that interested individuals have the ability to recreate the system on their own.

In previous experiments with the depth from focus algorithm [5, 6], reasonable depth maps were able to be generated from images of a scene taken at various focal lengths. However, the experiments in the past research did not have the system's latency as a constraint. In our system, the primary focus is to ensure a low latency generation of depth maps at high framerates of the scene such that they are capable of running in real-time. Thus the goal of the design focused primarily on maximizing the speed of both the algorithms and hardware in generating depth maps while still maintaining adequate levels of richness and accuracy within the depth maps. First, we will discuss which hardware components were chosen in order to ensure a high-speed parallelized generation of the depth maps.

3.1 Hardware Architecture

The primary constraints in choosing the hardware for this system was based on ensuring a high-speed computation of the wavelet decomposition algorithm as well as other methods that are used in generating individual depth maps. Furthermore, cost of the components were also factored in the design.

The overall design consists of four components. The first is a type of programmable lens called an Active Lens which aids us in adjusting the focal length of the optical configuration prior to taking images. The second component is an offset lens used to offset the overall focal length of the optical system. The third component is the high speed gray-scale camera which allows us to take images up to a maximum framerate of 751 fps. The final component is the NVidia TX1 which serves as the computational and image processing hub, whose onboard GPU allows us to perform massively parallel depth map calculations in real-time.

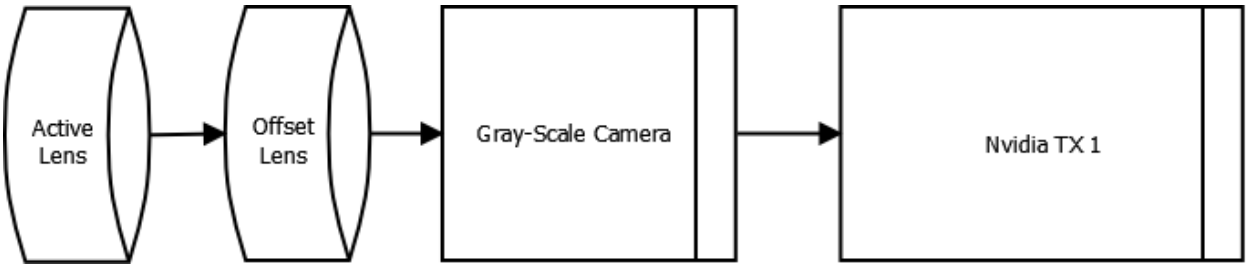


Figure 3-1: A diagram showcasing the hardware architecture of the system

Figure 3-1 diagrams the layout of the system where the signal travels through the lens first, becomes capture by the camera, then the digital signal is processed by the NVidia Tx1. First, we will discuss the Active Lens which serves as both an optical lens and as the primary means by which we configure the focal length of the optical system.

3.1.1 Active Lens

Active lens systems are special type of ocular lens which allow for programmatically setting the focal length of the lens. Within our system, a key constraint is the requirement of being able to take many images at varying focal lengths. Thus the ability to vary the focal length accurately and with low latency is crucial to the success of the overall depth mapping system.

The specific version of the lens used was the Optotune EL-10-30-C which was chosen specifically for its focal range, latency, and accuracy. Within the Active Lens is housed a thin membrane which contains a highly refractive liquid. The lens also has a USB interface with which current may be sent to the lens from a device. Lastly, the conductivity of the membrane itself in conjunction with coils surrounding the membrane allow current passed to the lens to generate an electric field and in turn apply pressure to the membrane. The produced electric field allows the membrane to constrict or contract and thus changing the focal length of the camera system.

3.1.2 Offset Lens

We also use an offset lens that is attached behind the active lens which has a static focal length that is manually set prior to running the system. This static lens is necessary due to the limited focal range of the active lens. Thus, by choosing the correct focal length of the static lens, one can manipulate the overall focal range of the system to either focus on the closer objects of the scene or objects farther away. The resulting conjunction of lens allow us to send commands to the lens and in turn adjust the focal length of the system all with a total latency of under 1 millisecond [8]. After the light signal from the scene passes through both the active lens and the offset lens, it is then captured by our high speed camera.

3.1.3 Camera

The requirements the system demanded from the camera were much more stringent than the lens component. For example, in order to generate a single depth map of the scene, N images must be taken, where each image is at a different focal length. Increasing N allows for the system to have better noise filtering capabilities and more depth resolution. This is due to the fact that when we calculate our $MaxFM$ matrix for this set of FM images, the depth values of the individual pixels are their respective indices where the FM was the highest. Thus with a wider range of image indices, we can be more accurate in finding which focal setting provided us with the highest focus. However, doing so also places higher demand on both the camera system and the image processing system. Hence if a user would want better depth mapping accuracy, they should increase N. However, if the user requires higher framerates, they should limit N. If we set $N = 100$ images, then in order to obtain 10 depth maps per second, we would need $fps * N = 10 * 100 = 1000$ images taken, each at a different focal length, per second. Thus, to ensure that the camera does not become a bottleneck in our system, we must place utmost priority in selecting a camera with the ability to capture images at a very high framerate.

We placed higher significance to the speed of the system (real-time) to maximize the framerate of the depth mapping. Thus, we decided that it was unnecessary to select a camera with a high resolution, nor a camera that is able to capture colored images. A camera with a high resolution and color would not only increase the number of pixels we have to analyze in order to perform our wavelet decomposition, but 3-channelled colored pixels means we would need to perform 3x the amount of computation as well. In the end, we decided to choose the Basler acA640 camera for its maximum framerate of 751 fps, USB interface, small size, 640x480 resolution, and cost.

By selecting a high-speed programmable Active Lens and a high-speed gray-scale camera to use in our system, we reduced the chances of the camera or the focal length configuration becoming a bottleneck in our system. However, without a doubt the most important component and bottleneck of the system is the image processing unit and computer which translates the raw images into depth maps by running a multitude of algorithms.

3.1.4 NVIDIA Jetson TX1

Arguably, the most important part of the system is the embedded component which houses the processing units that run our algorithms. Continuing our previous example, if we wish to create depth maps where each depth map is generated from the processing of 100 images, then the best we can do is to generate roughly 7 depth maps per second since our camera cannot take images at a higher framerate. This means that we must process roughly 700 images, each at a resolution of 640x480 pixels, in a single second. The minimum requirement algorithm required to generate inaccurate depth maps, without any sort of noise filtration, is the wavelet decomposition of the images as well as the $MaxFM$ matrix generation. Without any noise filtration, the depth maps would be extremely inaccurate and unusable for any client applications. Even then however, just the raw unfiltered depth map alone is guaranteed to place enormous load on the processing units of the system due to the large amount of FM matrices we must generate through our wavelet decomposition.

However, due to the nature of the problem, many of our algorithms, such as the wavelet decomposition, can be massively parallelized. Not only can we have fine-grained parallelization within individual algorithms (which run on the rows and columns of the image), we can also have coarse-grained parallelization via processing different images in parallel. Furthermore, much of our algorithms such as the discrete wavelet transform, require floating point operations. Unfortunately, traditional

general processing units would fail to give us the performance we require in our image processing as they are designed for limited parallelization of very generalized and often more complicated instructions rather than computing with a high degree of parallelization of much simple instruction sets. Thus, it was clear that the type of processing unit we needed was a Graphical Processing Unit (GPU) which is known for its high degree of parallelization, high speed vector operations, and high speed floating point operations.

Amongst the different available options, we decided to choose the NVidia Jetson TX 1 as our main computational unit due to a number of reasons. The first and most important is its on-board graphical processing unit which houses 256 CUDA cores. Different from traditional cores of general processors, each CUDA core excels at performing simple floating point, integer, or vector operations much faster than traditional cores. However, the CUDA cores generally cannot perform more advances instructions that general processor cores have been designed to do. Furthermore, amongst GPU programming languages, NVidia's CUDA hails as the most well documented and developer friendly language for industry standards in GPU programming. Given the fact that CUDA's syntax is nearly identical to traditional C++ and that native C++ code can be written alongside CUDA, it would be much easier to write highly parallelized algorithms in a CUDA supported system in comparison to other languages. For these reasons, the NVidia Jetson TX1 was chosen as the main computational unit by which we process raw images and compute depth maps.

3.2 Software Architecture

In designing the software architecture, great care must be taken to ensure the maximum performance capabilities of the overall depth map generation. Furthermore, due to the nature of the architecture of the NVidia Jetson TX1, we must carefully choose which processes to designate to be run on the general

ARM processor and which processes to perform on the GPU. Although many instructions can be executed correctly on both types of processors, there is often significant difference in the performance of the execution of the instructions. Furthermore, despite the powerful capabilities of the NVidia Jetson TX 1's graphical processing unit, its general ARM processor on the other hand is a smaller and much slower processor in comparison to modern desktop general processors. The overall architecture is represented in the following diagram figure 3-2.

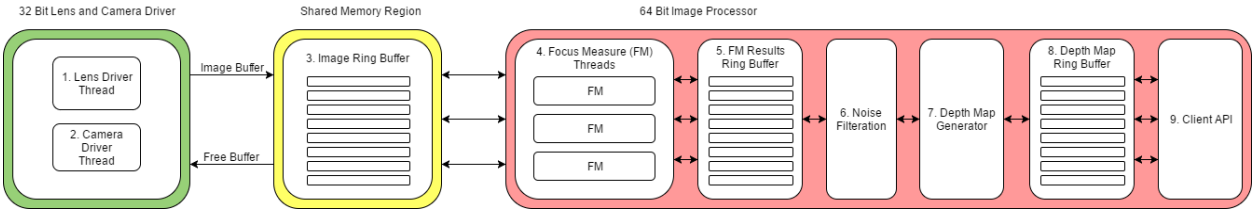


Figure 3-2: Layout of the overall software architecture. Arrows indicate passing of data.

Overall, the software architecture can be split into three major categories of processes. The first group of processes are the Active Lens driver process and the Camera Driver process. The lens driver effectively sends a designated current to the active lens hardware and sets the focal length for the next image to be taken. The Camera driver process actively captures images and once it captures an image, it obtains a free buffer from the Image ring buffer (figure 3-2) and fills it with the image data. Both processes run on the general processing unit and not on the GPU. Due to limitations in the camera manufacturer's available driver software, for which they only currently support x32 ARM processors, we were required to compile and execute these threads in a 32-bit environment.

Therein lied a major challenge in the software since the image processing threads often require parallelization, use the GPU, and are written in CUDA. This entails that the image processing threads are

required to be compiled and run from a 64-bit environment. Thus, in order to bridge the gap in the communication of the 32 Bit and 64 Bit processes, particularly the passing of image data between the camera driver and the Focus Measure threads, we initialize a static shared memory region where both 32-bit and 64-bit processes can read and write to via message passing. Thus the second group of processes is the Image ring buffer data structure which is housed in the shared memory region and orchestrates the process of passing images from the camera driver to the focus measure threads.

The third major group of processes are the various 64-bit image processing threads. The first set of processes in this group are the focus measure threads which request a filled image buffer from the shared memory region and perform the focus measure algorithm from section 2.1. Afterwards, the focus measure results are stored in another ring buffer (figure 3-2). Once all the focus measure scores are available for a full iteration of focal lengths, then the noise filtration processes are called which process the focus measure matrices from the focus measure ring buffer and apply a number of different noise filtration techniques in order to attempt to eliminate as much as noise as possible from the signal. After the noise filtration process, the filtered focus measure matrices are passed to the depth map generator which effectively computes the $maxFM$ matrix as described in section 2.2. Lastly, the depth maps are stored in a depth map buffer which is accessible to clients via an outward facing API.

3.2.1 Active Lens Driver

The active lens driver is a process which runs on the ARM general processor and sends current signals to the active lens unit in order to set the focal length of the optical system. The process itself waits for a message from the camera process to increment or change the focal length and then sets the focal length. A simple Checksum on the current configuration message is used when sending a serial signal to the active lens unit to ensure that the message is not corrupted.

3.2.2 Camera Driver

The camera driver is responsible for capturing a raw gray-scale 640x480 digital image from the Basler camera and passing it to the shared memory region. When the shared memory region has a free buffer, which indicates that a focus measure thread has already processed the previous image that was in the same buffer, the camera process indicates to the lens driver thread to increment the focal length and then captures an image. The number of images which have been written to the image ring buffer is updated and the camera drive thread waits until another buffer is free.

3.2.3 Shared Memory Region

The shared memory region is a region in memory where we house image data from the camera driver thread and allow the focus measure threads to access the image data. The size of the shared memory region itself can be configured by the user based on how many images they wish to store in memory that are waiting to be processed. A larger buffer size would allow more images to wait processing but it would also result in more stale images. On the other hand, using fewer buffers means the camera driver will toss out images until a buffer is free and thus it results in ensuring that we only use the most recent images as possible. Due to the fact that we are running the focus measure threads in parallel, we require a thread-safe data structure to allow the concurrent focus measure to ask for images and process them independently. Thus a ring buffer design was used where the camera process asks for free buffers and returns filled buffers, and the concurrent focus measure threads can ask for filled buffers to process concurrently. A concurrent thread-safe data structure is crucial due to the fact that multiple images should be processed concurrently to minimize the image processing bottleneck of the overall system which is the primary limitation in the depth maps per second or framerate of the system.

3.2.4 Focus Measure Threads

Out of all the processes in the software of the system, the generation of a focus measure matrix via the wavelet decomposition on the image is the most expensive and performance intensive process. Thus great care was taken to minimize the framerate bottleneck that is caused by the focus measure threads. Each thread obtains an image from the shared memory region, performs the focus measure, and returns a finished buffer that can be later overwritten by the camera driver. There were multiple performance optimizations taken in designing the focus measure threads. Firstly, each thread is executed on the GPU to fully utilize the 256 cores of the graphical processing unit. In order to maximize the parallelization and scalability of the wavelet decomposition, and hence the focus measure matrix generation, we automatically split the original image into smaller blocks where each CUDA core can process and generate a focus measure score independently. The block sizes are chosen based on the resolution of the original images and the number of cores the hardware has. In order to utilize all cores, we would need to split the image into at least $\frac{resolution}{numcores}$ blocks. This allows us to ensure fine-grained parallelization of the focus measure computation. Furthermore, we have multiple overall focus measure threads running concurrently, each processing different images at the same time. The results of each focus measure thread are placed in another ring buffer afterwards.

3.2.5 Focus Measure Ring Buffer

This ring buffer is similar in structure and function to the ring buffer in section 3.2.3 used for raw image data. Again, at this stage we are required to use a ring buffer structure because we have multiple concurrent focus measure threads attempting to store their focus measure results into the same data structure and the ring buffer can provide us with efficient thread-safe storage. Again, the size of the

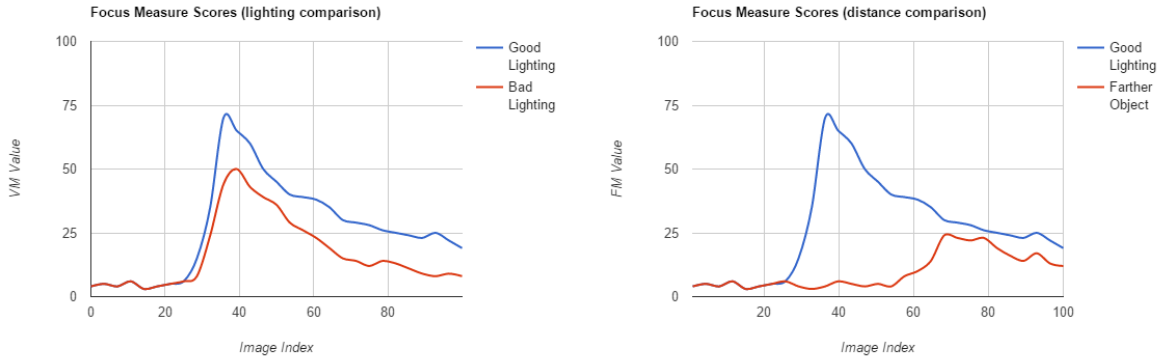
buffer is chosen by the user. A larger buffer means that images waiting to be processed will have a longer latency. A smaller buffer size means that images will be thrown away until the buffer has room and thus we will have less stale images. The focus measure results are then provided to the next stage which is the noise filtration process.

3.2.6 Noise Filtration Process

The thread responsible for removing noise filtration is integral to the success of the depth mapping system. This is because we need to find what distinguishes pixels which have a clear peak at a certain focal length from those pixels which relatively do not peak at all and thus whose depth is not detectable by our system. A comparison of the focus measure evaluations of noise and a relevant signal is shown below in Figure 3-3 [6]. The chart shows how the focus measure score of a signal reaches a peak before tapering off unlike the noise samples.

Figure 3-3: Focus Measure score of noise vs. signal data. Source: [6]

One important aspect of signal data in comparison to noise is that the focus measure for signals steeply climb to a peak before tapering off, whereas noise data often remains closer to its mean and results in minimal to no single peaks of focus measures. However, there are other factors to consider as well. For instance, under poor lighting conditions, the height of the focus measure peak of the signal is greatly dampened due to the general blurriness of the raw images from the camera. This also occurs to signals as we attempt to obtain the focus measure scores for objects farther in the distance in comparison to those which are closer to the camera. Figure 3-4 depicts the original signal under worse lighting conditions and Figure 3-5 compares a signal of the focus measure scores of a pixel on an object that is farther away.



Left: Figure 3-4 comparison of lighting conditions. Right: Figure 3-5 effect of distance on focus measure

In order to adequately filter out the noise from the sample data we tested a number of various techniques within our system. In the following subsections, we will talk about the different noise methods and how they work. We compare the depth map results of the noise filtration methods in the Results chapter later in this paper.

3.2.6.1 Minimum Thresholding

The first and most naïve method one can attempt is to simply set a minimum focus measure threshold for the data. Simply put, if a focus measure calculation for a pixel for a given image is below the threshold, it is counted as noise and ignored. Being the simplest of the filtration methods, the minimum thresholding method incurs negligible performance cost to the system. The threshold itself is manually selected by the user. Statement (3.1) shows the behavior of this filter.

$$if\ FM[p] < minThreshold, FM[p] = 0 \tag{3.1}$$

3.2.6.2 Kurtosis Thresholding

The next method we implemented is a thresholding based on the kurtosis value of the signal. Indeed, often the kurtosis value for a signal can be used to represent how "peaky" a signal is. Figure 3-6 showcases the different kurtosis values of different types of distributions.

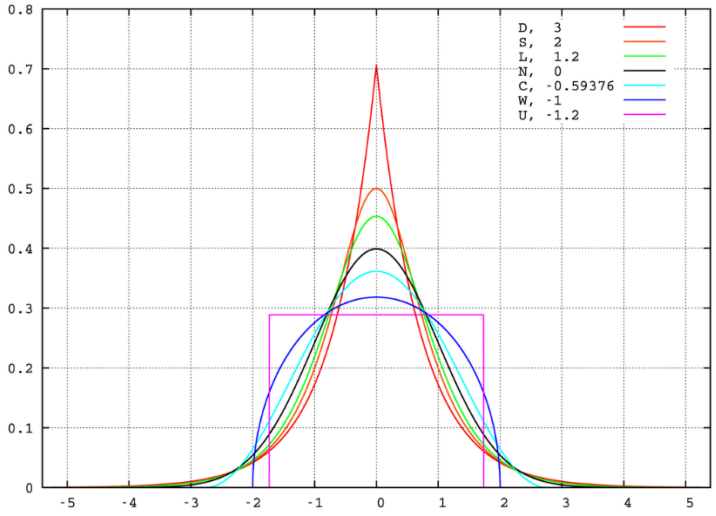


Figure 3-6: A comparison of different kurtosis values

The kurtosis can be calculated by taking the ratio of the fourth moment and the variance squared. It can be represented in the following notation in equation (3.1).

$$FM_{mean}[p] = \frac{\sum_{i=0}^{N-1} FM_i[p]}{N} \tag{3.2}$$

$$k[p] = N \frac{\sum_{i=0}^{N-1}(FM_i[p] - FM_{mean}[p])^4}{(\sum_{i=0}^{N-1}(FM_i[p] - FM_{mean}[p])^2)^2}$$

As can be noted from the equation, the calculation of kurtosis for a given pixel requires at least iterating over all the focus measure scores for said pixels twice. The first to calculate the mean and the second to calculate the kurtosis value. Again, if the kurtosis value for the pixel is below the threshold we have set for the system, then it is discarded as noise. Similarly, the Kurtosis threshold is set manually by the user.

3.2.6.3 Variance Thresholding

Similar to the kurtosis thresholding method, the variance method is also a statistical analysis of the signal sample points that can be used to determine if the signal is noise or not. The variance can be calculated in a similar manner in equation (3.2).

$$FM_{mean}[p] = \frac{\sum_{i=0}^{N-1} FM_i[p]}{N} \tag{1.3}$$

$$v[p] = \frac{\sum_{i=0}^{N-1}(FM_i[p] - FM_{mean}[p])^2}{N}$$

3.2.6.4 Maximum Mean Thresholding

After analyzing and comparing the noise signals to data signals, we created our own filtration criteria. Often than not, what determined whether or not samples were signal or noise was the fact that the difference between the peak and the mean was also quite noticeable. Thus we defined another metric by taking the difference between the maximum focus value and subtracting it from the remaining mean. The value can be calculated using the formula in (3.4).

$$FM_{max}[p] = max_{i=0}^{N-1} FM_i[p] \tag{3.4}$$

$$maxmean[p] = FM_{max} - \frac{(\sum_{i=0}^{N-1} FM_i[p]) - FM_{max}}{N-1}$$

Each of the three noise filtration methods performed differently and incurred different runtime costs. In the Results section, we discuss and compare the results of the different filtration methods. Regardless, when starting the system, the user is able to give command line arguments to the process to decide whether to use any of the noise filtration methods, or any combination of them, and what their thresholds should be.

3.2.7 Depth Map Generator

The depth map generator thread is simple and incurs minimal performance cost. The generator process takes the filtered focus measure matrices and simply computes the maxFM matrix as discussed in section 2.2. It then places the computed depth map into the depth map ring buffer.

3.2.8 Depth Map Ring Buffer

The depth map ring buffer is a data structure that receives completed depth maps from the generator process and in turn exposes the finished depth maps to the client API.

3.2.9 Client API

The last and final component of the software is the client API where client processes can retrieve completed depth maps in order. The interface is listed in the appendix.

3.2.10 Tessellation

A final new configuration was added to the software architecture which allowed us to down-sample the focus measure matrices and thus speed up computation by reducing the resolution of the depth maps. While sacrificing the depth map resolution, tessellation results in a faster depth map and more responsiveness.

4 Results

After implementation of the entire system as describe in section 3, the results of the depth mapping system were quite promising. Much of the system, such as the number of frames required per single depth map and noise filtering thresholds, is configurable via command line arguments which allowed for easy experimentation and testing of different configurations. Of course, our utmost primary concern is the speed of the system to ensure it is real-time reactive. Secondly, we wish to have as accurate a depth map as possible without sacrificing too much of the framerate of the depth maps.

The test environment used in this paper involves 3 signs spaced approximately 200 mm apart from each other. The signs were designed to increase the contrast in the resulting raw images as much as possible and thus involve a black background and white text. Figure 4-1 shows the layout of the three signs.

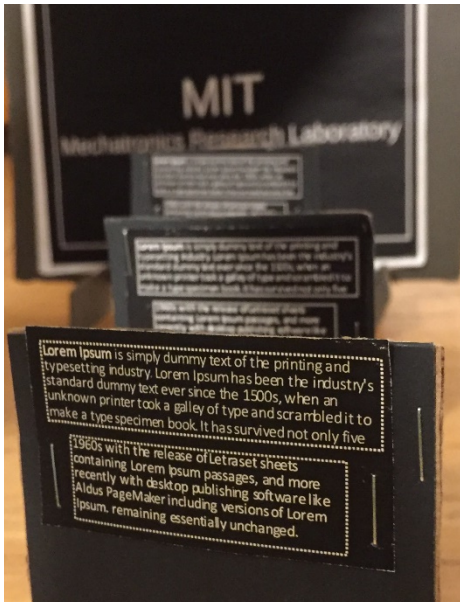


Figure 4-1: The layout of the three signs used in the experiments

4.1 Raw Depth Map Image

In this section, we discuss the results of attempting to create a depth map where we perform no depth mapping but instead vary the number of images required to produce a single depth map. Varying the number of images that each depth map requires serves to increase the accuracy of the depth map's granularity. Each pixel is mapped to an image index and thus if we require 80 images per depth map, there are 80 possible values of depth per pixel. On the other hand, if we reduce the number images required per depth map, we end up with a coarser depth map.

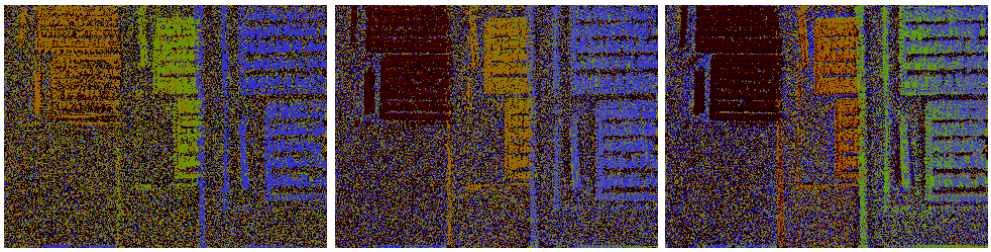


Figure 4-2: Depth maps generated from varying N. Left: $N = 5$. Center: $\mathrm{N} = 10$. Right: $\mathrm{N} = 80$.

The visualization of the depth in the depth map uses color to indicate the depth values of individual pixels. Note that the color transition occurs in the following order from closest to farthest: blue, green, yellow, red. Thus in our depth maps, we can clearly visualize the closest sign which is a mix of blue and green, afterwards the next farthest sign which is a yellowish red, then finally the farthest sign which is red. Note the change in colors of the depth as we increase the number of frames per depth map. This is due to the fact that as we increase the number of frames, the possible range of depth values increases and thus our pixels take on the hue of color that is most accurate to their depth. The depth map in figure 4-2 with $N = 5$ only has 5 possible colors for each pixel and the depth map with $N = 80$ has 80 possible colors because we have a larger possibility of image indices in our MaxFM.

4.2 Noise Filtration

Although in the above depth maps in figure 4-2 the depth of the signs is clearly visible, there still exists quite a bit of noise amongst the depth of the individual pixels. This is because no noise filtration is applied on those depth images. In this section, we explore the application of the various depth map noise filtration techniques discussed in section 3.2.6 and compare their resulting depth maps. It should be noted that all the following depth maps in this section are generate using $N = 80$ images per single depth map frame and we explore conditions for noise filtration in both good lighting and bad lighting.

4.2.1 Well Lit Environment

All depth maps in figure 4-2 and the depth maps in this section are taken under well-lit conditions. In the well let environment, a lamp is placed in front of the signs where as in a badly lit environment the lamp is turned off. Good lighting highlights the contrast in objects within the view of the camera and allows us to more accurately calculate the focus measure values for each pixel. First, let us display the unfiltered depth map under the same conditions so we can compare it later on with the filtered versions.

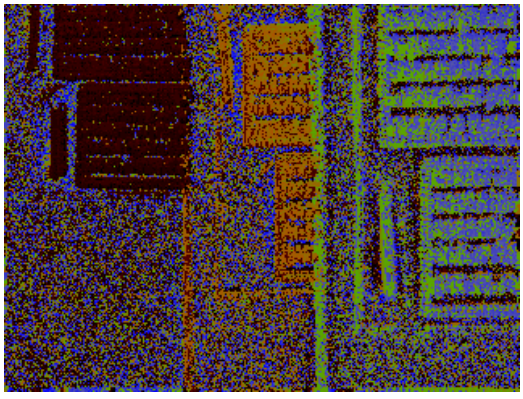


Figure 4-3: Unfiltered depth map taken in good lighting

4.2.1.1 Minimum Thresholding

The below depth maps are the result of applying the minimum thresholding method, discussed in section 3.2.6.1 to the original depth maps in a well-lit environment.

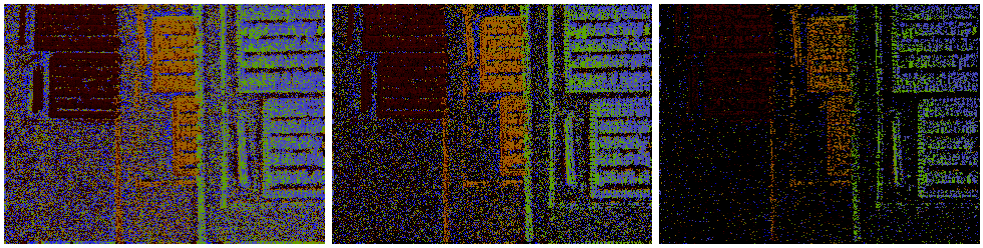
$threshold = 5$ $threshold = 10$ $threshold = 30$

Figure 4-4: Minimum thresholding applied to image with $N = 80$

It should be noted that the minimum thresholding does quite a good job qualitatively to reduce the noise found in the original unfiltered depth maps. The noise reduction is noticeable particularly in regions outside the text of the signs.

4.2.1.2 Kurtosis Thresholding

Here we apply the kurtosis thresholding method discussed in section 3.2.6.2 to the original depth map.



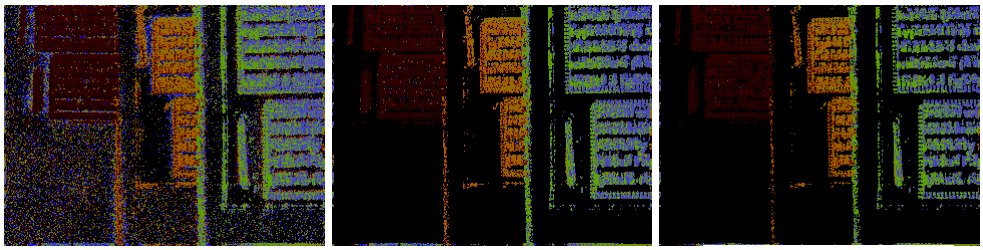
$threshold = 1$ $threshold = 5$ $threshold = 10$

Figure 4-5: Kurtosis thresholding applied to image with $N = 80$

Despite our initial hypothesis that Kurtosis should have given a good indicator of whether samples were noise or signal, the results indicated that kurtosis was the worse out of the four filters we tested in our system. Most likely this can be attributed to the fact that although the signal's focus measure values do indeed reach a single peak, they are skewed and the noise can sometimes peak as well. Notice that increasing the threshold uniformly reduces both noise and signal strength in the depth maps.

4.2.1.3 Variance Thresholding

Below is the application of the variance thresholding method as discussed in section 3.2.6.3.



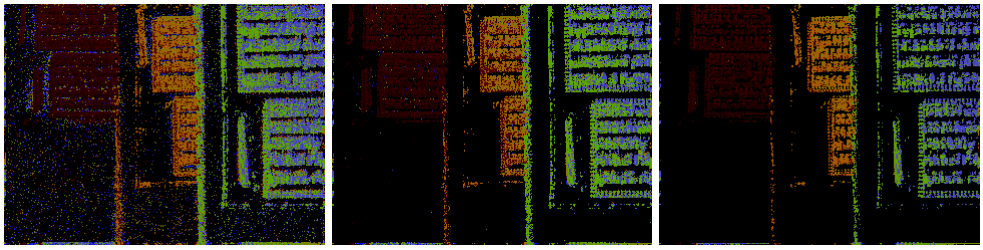
$threshold = 1$ $threshold = 5$ $threshold = 10$

Figure 4-6: Variance thresholding applied to image with $N = 80$

The results of performing the variance thresholding indicate that under well-lit environments and at a relatively high number of images per depth map, the variance thresholding fares much better than the kurtosis thresholding method and slightly better than the minimum thresholding.

4.2.1.4 Maximum Mean Thresholding

Lastly, we show the results of applying the maximum mean thresholding as discussed in section 3.6.2.4.



$threshold = 5$ $threshold = 10$ $threshold = 20$

Figure 4-7: Maximum mean thresholding applied to image with $N = 80$

The maximum mean thresholding method's ability to reduce noise in the final depth map is very similar to the Variance method under well-lit conditions.

4.2.2 Badly Lit Environment

Although the results of running the noise filtration methods resulted in there being negligible difference between the minimum thresholding, variance, and maximum mean noise filtration capabilities, in this section we show that the differences in the filtration methods become more clear as the lighting of the scene worsens. Below is the unfiltered depth map of the environment when lighting conditions are minimal and 80 images are used per depth map.

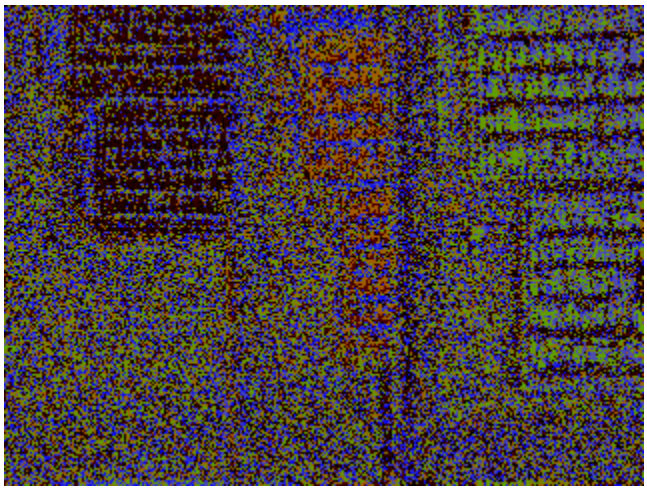


Figure 4-8: Unfiltered depth map of the badly lit environment

Note the effect of worse lighting conditions. The amount of noise is greatly increased and the number of signal pixels are lessened. Below are the best results from applying the four noise filtration techniques to the badly lit environment. All depth maps are generated with $N = 80$

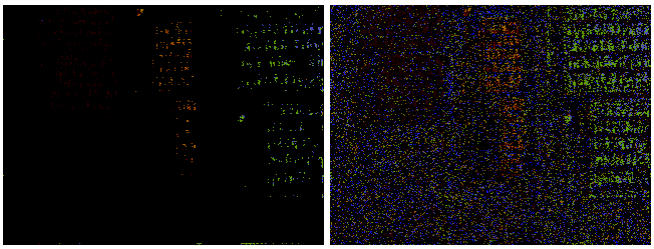


Figure 4-9: Left: Minimum thresholding set to 10. Right: Kurtosis thresholding set to 7

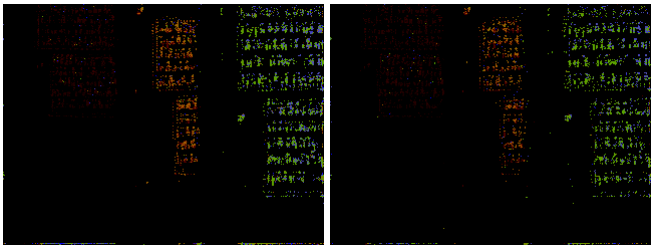


Figure 4-10: Left: Variance thresholding set to 1. Right: Max Average thresholding set to 5

In comparing the various results of the application of the noise filtration methods on a badly lit environment, we can note major discrepancies. Firstly, we confirm our notion that utilizing the kurtosis value for thresholding performs poorly in reducing noise in the signal. Secondly, we can now note a difference that was difficult to detect in the well-lit scenario. The fact that the depth of the third and farthest sign is far less visible in the minimum thresholding depth map in comparison to both the variance and max average depth maps. The reason is that as the lighting condition worsens, the height of the maximum value of the focus measure scores of individual pixels also become dampened and decrease. Combining the reduction in the maximum focus measure from the bad lighting and the fact that the third sign is farther means that although the signal will have a peak focus measure, it will be very low and hence why our minimum thresholding incorrectly believed it was noise. In contrast, our variance and max average methods utilize the mean of the signal and not just the maximum. This allows the two methods to still detect the farthest sign even in worse lighting conditions.

As a result of comparing the three noise filtration methods, it can be concluded that in well-lit environments the minimum thresholding, variance thresholding, and max average thresholding perform

similarly in reducing the noise of the signal. However, as the lighting condition worsens in the environment, the variance and max average methods perform better.

4.3 Depth Map Framerate

Arguably, this section is the most important result out of the previous sections. This is due to the fact that the primary goal of this research is to develop a real-time depth mapping system and hence the rate at which depth maps are generated, or the depth map frame rate, must be maximized in order to ensure real-time generation. In this section, we compare multiple configurations mentioned in Section 4.1 and 4.2 and their resultant effects on the frame rate of our depth maps.

Here, we will compare the effect of increasing the number of images required to produce a single depth map on the framerate at which depth maps are generated per second. Below is a graph of the different configurations.

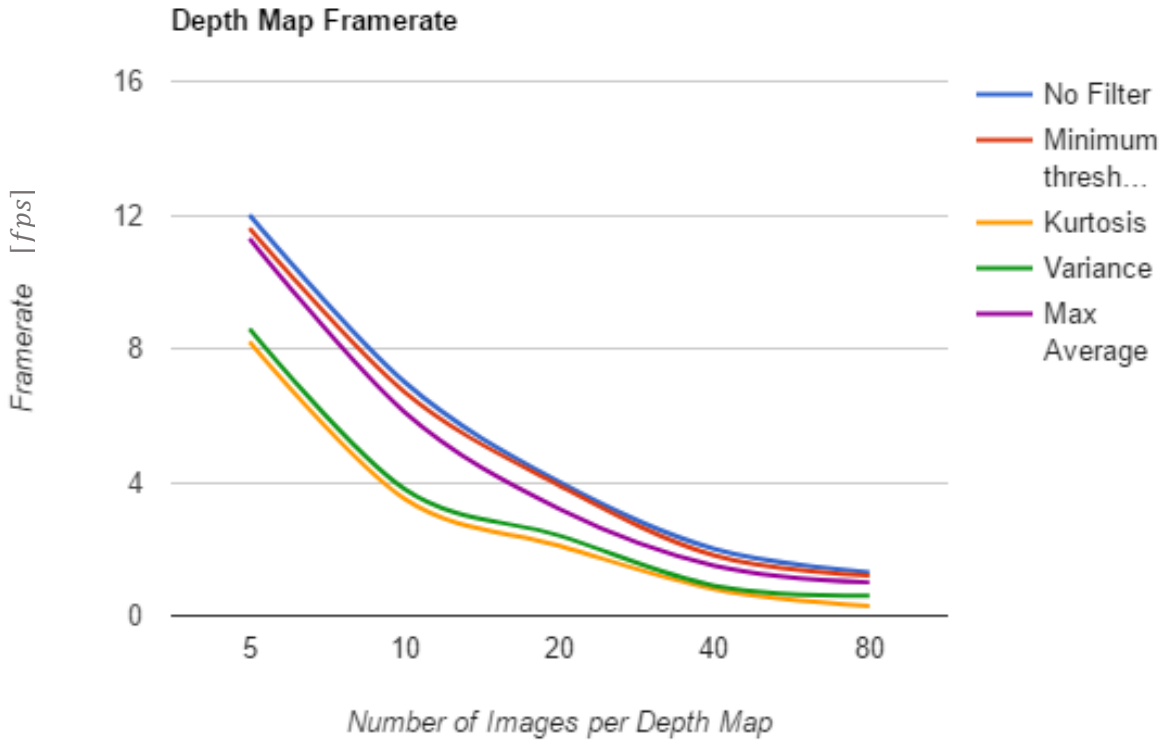


Figure 4-11: Plot of the effect of increasing the number of images per depth map for various configurations on the framerate of the depth map generation

From the results, it is clear that despite our optimizations and design in parallelizing the depth frame generation process, increasing the number of images required per depth map greatly reduces the framerate of our system. Furthermore, the calculation of individual kurtosis and variance methods take significantly longer in comparison to using no filter, the minimum thresholding method, or the max average method.

4.4 Tessellation

Another method was explored in order to reduce the runtime of the depth map generation systems. It must be considered that not all devices which wish to make use of the depth maps generated by us require the maximum depth map resolution. Thus, in an effort to reduce the runtime of the depth maps, we attempted to tessellate and reduce the resolution of the depth map while performing the focus measure calculations. After the HH, LH, and HL matrices are calculated by the wavelet decomposition, we average neighboring pixel blocks of the details into scaled down versions. Afterwards, we calculate the reduced sized FM matrix which in the end leads to a reduced depth map size.

Within the system, the user is able to feed arguments to the image processor in regard to the tessellation factor. A tessellation factor of 0 reflects the original imager, a factor 1 down scales the depth maps by 4x and a factor of 2 downscales by 16x. Formula (4.1) illustrates the relationship between the original resolution and the new tessellated resolution given by the tessellation factor T.

$$\text{newResolution} = \frac{originalResolution}{4^T} \tag{4.1}$$

Below are the results of tessellating the original scene using various tessellation factors.

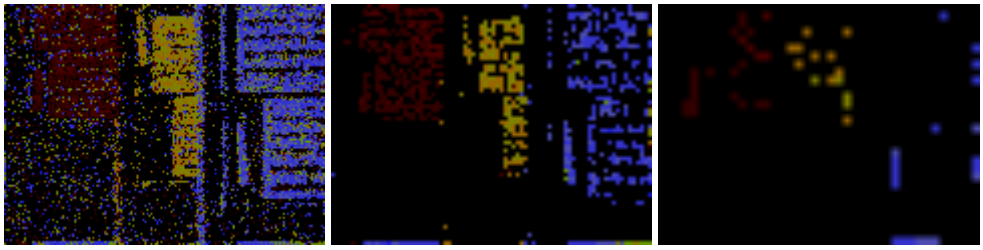


Figure 4-14: Tessellation of the original depth map scene by factors of 1, 2, and 3. $N = 80$

Unexpectedly, because we use averaging the details from the wavelet decomposition in our tessellation process, the increase in the tessellation factors resulted in noise reduction. This is suspected to be primarily due to the fact that because of the random nature of noise, averaging noise samples results in cancellation of the noise by each other. Indeed, the tessellation successfully reduces the scale of the depth map and results in a more pixelated and low resolution depth map.

5 Conclusions & Recommendations

5.1 Conclusions

In conclusion, after experimentation of different configurations we have shown that with the methods and designs discussed in this paper, one can implement a real-time depth mapping system. Furthermore, our tests indicate that the best noise filtration method is the max averaging as it performs well in poorly lit conditions and has the minimum performance impact on the overall framerate of the depth maps generation. We hope that this research is continued forward to further improve the said system in as many ways as possible. Lastly, we hope that the research conducted in this paper was able to aid the progress of improvements in the field of depth mapping systems. As technology progresses, so do the requirements on the robustness of systems and their sensory capabilities. Thus the goal of aiding depth mapping systems is to help improve existing devices and in turn benefit the quality of human life.

5.2 Recommendations

Despite the fact that we were able to implement a real-time depth mapping system along with adequate noise filtration of the results, there are many ways in which this research can be continued in order to perfect the depth map generation system. The first of which are further software optimizations and changes to the software design in such ways to further reduce bottlenecks in the code and minimize both the software complexity of the system and the runtime such that our depth map system is made

more reactive. For example, the implementations of the wavelet decomposition or the various noise filtration methods can be further optimized to reduce their individual runtime complexities. Second, the system should be tested with different hardware to see the effect on the depth map generation. This includes using different cameras, lens, and a computational unit. In fact, it is possible that FPGAs have the potential to speed up the system better than the GPU and thus we would like to test implementing portions of the software modules in FPGAs as well.

Another major feature which could greatly aid the depth map generation system is the use of modeling in both noise filtration and depth map estimation. The model would take as input the samples of calculated focus measure scores and its output would be the distance it estimates the pixel is from the camera. Currently, we naively use the image index as an estimate of the depth to the object.

However, if an adequate model was constructed of the focus measure signal, then with constraint optimization we can fit our samples to our model. Not only would this allow us to have higher depth map granularity with less images per depth map, it would also allow us to simultaneously perform noise filtration as samples that do not fit the model well can be ignored as noise.

Lastly, we would like to explore applications and example use cases of this system in the real world. In the future, we would like to utilize the generated depth maps to solve simple depth problems in constrained real-world scenarios such as on robots and other devices.

Bibliography

[1] Stavens, David Michael, et al. Learning to drive: perception for autonomous cars. Stanford University, 2011.

[2] Dissanayake, MWM Gamini, et al. "A solution to the simultaneous localization and map building (SLAM) problem." IEEE Transactions on robotics and automation 17.3 (2001): 229-241.

[3] Kamencay, Patrik, et al. "Improved depth map estimation from stereo images based on hybrid method." Radioengineering (2012).

[4] Li, Larry. "Time-of-flight camera–an introduction." Technical White Paper, May (2014).

[5] Iman Soltani Bozchalooi, Kamal. Youcef-Toumi, Mohsen. Lakehal-Ayat. Depth mapping using active lenses. Internal Report: Mechatronics Research Laboratory, Oct - Dec, 2014 Jan - Sep, 2015.

[6] JooHun Kim, High-Speed Optics based Depth Mapping for Automated Active Safety System, Master of Engineering Thesis: Massachusetts Institute of Technology, May 18, 2016.

[7] Marcellin, Michael W., et al. "An overview of JPEG-2000." Data Compression Conference, 2000. Proceedings. DCC 2000. IEEE, 2000.

[8] Optotune. Fast Electrically Tunable Lens EL-10-30 Series.

[9] Ryan Fanello, Sean, et al. "Hyperdepth: Learning depth from structured light without matching." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[10] Zhu, Jiejie, et al. "Reliability fusion of time-of-flight depth and stereo geometry for high quality depth maps." IEEE transactions on pattern analysis and machine intelligence 33.7 (2011): 1400-1414.

[11] Lucas, Bruce D., and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." (1981): 674-679.

Appendix

NVidia Jetson TX1: An embedded computer created by NVidia with the following statistics

GPU: NVIDIA Maxwell ™, 256 CUDA cores

CPU: Quad ARM® A57/2 MB L2

Memory: 4 GB 64 bit LPDDR4 @ 25.6 GB/s

Client API: Method to obtain depth map images

$bool\ getDepthMap(float * depthMapBuffer)$:

Argument: $\boldsymbol{depthMapBuffer}$ – A float pointer to store the results of the depth map in.

returns False if there were no depth maps which were already processed in the buffer.

returns True if there was a depth map readily processed, and it fills $depthMapBuffer$ with the results.