Toward a Compact Underwater Structured Light 3-D Imaging System

by

Geoffrey E. Dawson

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

June 2013

Signature of Author: [illegible]
Department of Mechanical Engineering
May 10, 2013

Certified by: [illegible]
Dr. John J. Leonard
Professor of Mechanical and Ocean Engineering
Thesis Supervisor

Accepted by: [illegible]
Dr. Anette Hosoi
Professor of Mechanical Engineering
Undergraduate Officer

Toward a Compact Underwater Structured Light 3-D Imaging System

by

Geoffrey E. Dawson

Submitted to the Department of Mechanical Engineering
on May 10, 2013 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

A compact underwater 3-D imaging system based on the principles of structured light was created for classroom demonstration and laboratory research purposes. The 3-D scanner design was based on research by the Hackengineer team at Rice University. The system is comprised of a low-power, open-source hardware single-board computer running a modified Linux distribution with OpenCV libraries, a DLP pico projector, camera board, and battery module with advanced power management. The system was designed to be low-cost, compact, and portable, while satisfying requirements for watertightness. Future development and applications may involve navigation systems for an autonomous underwater vehicle (AUV).

An initial study of 3-D imaging methods is presented, and the strengths and drawbacks of each type are discussed. The structured light method was selected for further study for its ability to produce high-resolution 3-D images for a reasonable cost. The build of the 3-D imaging system was documented for reproducibility, and subsequent testing demonstrated its functions and ability to produce 3-D images. An instruction guide for operation of the device is provided for future classroom and laboratory use.

The 3-D imaging system serves as a proof-of-concept for utilizing structured light methods to produce 3-D images underwater. Image resolution was limited by the output resolution of the pico projector and camera module. Further exploration in obtaining ultra high-resolution 3-D images may include use of a more powerful projector and a higher resolution camera board module with autofocus. Satisfactory 3-D scanning validated the performance of structured light scanning above water. However, contaminants in the water hindered accurate rendering by the system while submerged due to light scattering. Future development of a on-the-fly mapmaking system for AUV navigation should include algorithms for filtering light scattering, and hardware should based on an instantaneous structured light system utilizing the Kinect 2-D pattern method. Autofocus and increased projector brightness would also be worthwhile additions.

Thesis Supervisor: Dr. John J. Leonard
Title: Professor of Mechanical and Ocean Engineering

Acknowledgments

I would like to thank my advisor and thesis supervisor Dr. John J. Leonard for his guidance and support during the development of this project. His work on 3-D imaging systems was the inspiration for this thesis, and I hope he will enjoy utilizing the system for years to come in classroom demonstrations and future laboratory explorations.

I would also like to thank Dr. Michael Kaess for his assistance with the programming aspects of this project. Having guidance while setting up the hardware and software systems and help during troubleshooting was indispensable.

Also, I would like to thank the Hackengineer blog team (www.hackengineer.com) for providing a structured light 3-D scanner design and code to base the project on. Their previous work greatly streamlined the imaging system development and allowed for a greater focus to be placed on development for underwater use.

I would not be the engineer I am today without all the fantastic professors, lecturers and assistants I have had the pleasure to learn from and work with here at MIT. Thank you for equipping me with the technical knowledge and intellectual curiosity to pursue a career in mechanical engineering.

Lastly, I owe a huge thank you to my family for always being there for me. Mom, Dad, Phillip and Cooper, I would have to write an entire dissertation to enumerate all the ways you've given me love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivation

3-D imaging is a fast-growing field of interest. Availability of consumer-grade 3-D printing systems (such as the Makerbot Replicator) has exploded the past few years - buyers of these systems can translate CAD files and 3-D images from virtual representations to real-life objects from the comfort of their homes - and, not to be outdone, growth in industry and research applications of 3-D imaging has accelerated as well. Uses for such technology include on-the-fly mapmaking for robots, and metrology methods for a variety of quality control tests. New ways of implementing 3-D scanning into applications are becoming a commonplace occurrence.

Different types of 3-D imaging systems exist, from the infrared projectors used by the Xbox Kinect to the time-of-flight systems used to scan skyscrapers. Systems utilizing these techniques can be used to create models of nearly any desired scene. By taking an image, one can obtain the data needed to replicate or inspect a physical object in a timely manner. The resulting 3-D images contain a point cloud that is rendered by a computer to create a mesh, which is representative of the 3-D scene with respect to the camera lens. It is important to note that most systems are confined to a cone of vision; therefore to recreate an entire object, a programmed reference system is needed to merge the scans.

This project's focus was to create a low-cost, compact, and portable 3-D imaging system for underwater scanning. Current 3-D imaging systems are large, non-portable, and non-waterproof. Adapting these systems for underwater use is prohibited by their power and space needs. By utilizing a design that shrunk the power requirements and footprint, it was possible to create a low-cost, compact, and portable 3-D imaging system that could be used for the basis of a future autonomous underwater vehicle (AUV) navigation system.

1.2 Types of 3-D Imaging Technologies

Three techniques will be discussed herein: time-of-flight, 3-D laser scanning, and structured light. Each technique has its own strengths and weaknesses, and specializes in applications for different ranges and resolutions.

1.2.1 Time-of-Flight

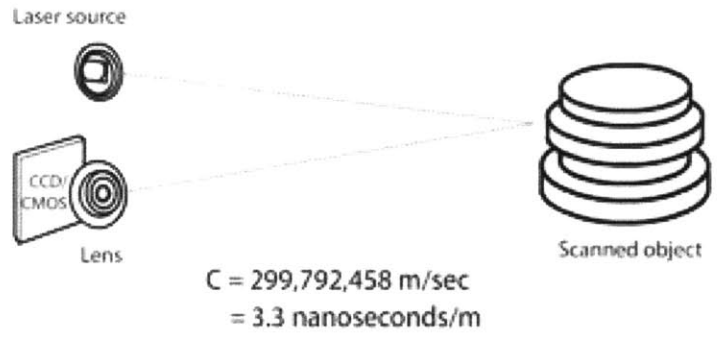

Figure 1-1: Diagram of a time-of-flight 3-D imaging scanner [1].

A time-of-flight camera measures the amount of time it takes for a light signal to reach and reflect off of a subject. A single pulse of light is given, and for each point in the image, the elapsed time is recorded, and distance is calculated relative to the speed of light; thus, a point that records a longer time is farther from the camera. A point cloud is created by matching these distances with the direction from which they were reflected, resulting in a 3-D model of the scene. Time-of-flight is classified as a type of scannerless LIDaR (Light Detection and Ranging) system [2].

The greatest benefit of a time-of-flight system is that its scans are instantaneous – one light pulse records an entire image, and it is possible to capture up to 100 images per second. Systems tend to be efficient and require no large moving subassemblies. Time-of-flight systems can be used for long-range applications on the order of kilometers. However, the resolution of these systems is relatively low; many systems record QVGA resolution (320x240 pixels) [3]. Depending on the distance of the subject from the camera, the resulting image resolution can range from millimeters to meters. Therefore time-of-flight systems are used mainly for purposes in which time sensitivity, not detail sensitivity, is the primary concern – for instance, tracking human movement or building maps on-the-fly with moving robots, or for scanning large scenes, like a building. Contrary to common perception, the Xbox Kinect is not a time-of-flight based system, though it behaves similarly. The Kinect employs a modified form of structured light patterns that scans instantaneously, allowing it to track quick movements over time.

The tradeoff of speed vs. resolution is common between different types of 3-D imaging systems; time-of-flight is favored by applications that emphasize speed and range over resolution.

1.2.2 3-D Laser Scanning

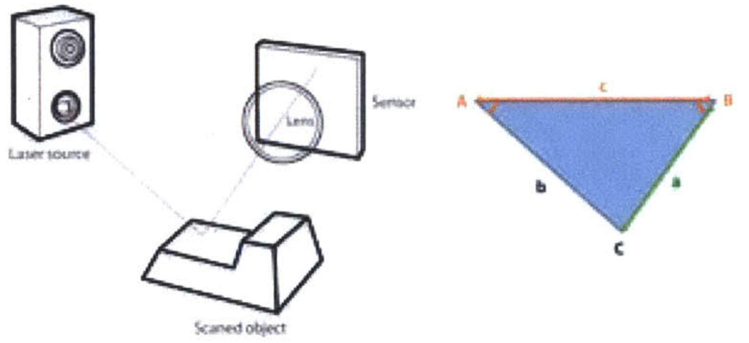

Figure 1-2: Diagram of a laser 3-D imaging scanner [1].

3-D laser scanners triangulate the positions of points relative to the system using laser lines and a camera sensor. By calculating the distance of points relative to the speed of light, and accounting for the geometry between the camera, laser source, and point, a point cloud is formed [4]. As the name suggests, scanning requires time for the laser rangefinder to process all points within the image.

3-D laser scanners cannot be used for long-range applications, such as scanning buildings or natural formations; its range is limited to meters. However, the resolution of 3-D laser scanners is very good. Whereas time-of-flight system resolutions can accurately detect on the level of millimeters, 3-D laser scanners are orders of magnitude better, scanning resolutions on the level of angstroms. But, scanning is not instantaneous, and systems require mechanical moving parts to manipulate the laser rangefinder. Thus, systems can be cost prohibitive for certain purposes.

Laser scanners are used commonly for prototyping and reverse engineering, as well as laboratory and metrology work.

1.2.3 Structured Light

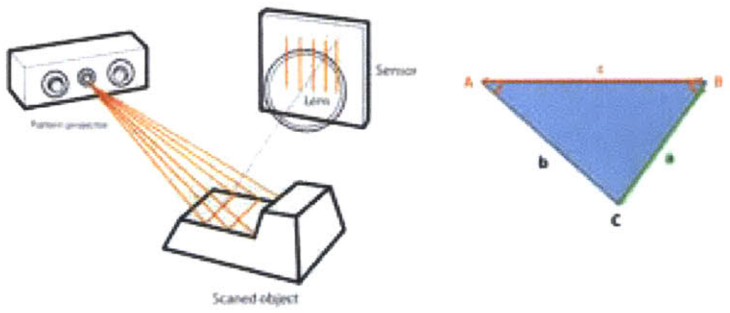

Figure 1-3: Diagram of a structured light 3-D imaging scanner [1].

Structured light scanners operate similarly to laser scanners, though instead of laser points, a series of light patterns is employed. Whereas laser scanners scan point by point across an object, structured light scanners scan through depth layers. A typical system utilizes a projector to beam a pattern of stripes across an object, and a camera sensor determines distortions in the pattern [5]. As different patterns of stripes are projected, the resulting distortions can be stitched together to determine the distance of points relative to the camera, and a point cloud representation of the object surface is formed.

Projecting many stripes at once allows for fast scanning, though it is not instantaneous like time-of-flight systems. Structured light projection patterns can be sourced from either lasers or projectors, though laser systems are very expensive and remain reserved for industrial and research applications only.

The Xbox Kinect is a special type of structured light scanner that projects an infrared 2-D pattern similar to a QR code on a subject, which allows it to calculate distortions instantaneously, without scanning through different depths over time. Instead, it compares the pattern it detects with the pattern it projects, and calculates depths from the distortions in the 2-D pattern. However, the resolution is limited to the pattern complexity, which is dependent on the infrared projection hardware. Therefore it is good for tracking and low-detail applications, but falls short of applications needing high resolution, such as object model replication and metrology. This thesis uses traditional structured light techniques because it allows for future development that will yield higher resolution models than the Kinect model.

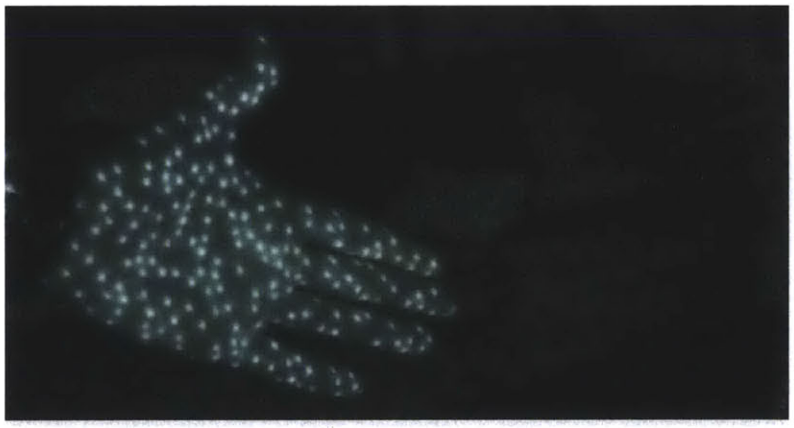
Figure 1-4: Example of Xbox Kinect projection pattern and distortions [6].

Resolution is limited by the complexity of the pattern, which is related to the projector resolution. In the case of lasers, interferometry can be used, resulting in very high resolutions. Structured light scanners employing projectors are a good tradeoff between resolution and cost for consumer applications. Recent advances in projector technology have resulted in a new class of pico projectors – low-cost, low-power projectors that can be used in applications such as structured light 3-D imaging systems.

Structured light imaging systems are adaptable to a wide range of situations and are used in applications ranging from building scans for projection mapping light shows, object replication, and consumer electronics, such as the Kinect.

The goal for this thesis is to recreate a structured light 3-D imaging system designed pioneered by the Hackengineer team, and repurpose it for underwater use. The effects of utilizing structured light techniques underwater have not been widely explored, and recent developments in computer and projector technology have enabled the design of a 3-D imaging system that satisfies power and portability requirements for such research. Observing how such a system would behave underwater is the first step in creating a more advanced system for research purposes. Future applications of such a scanner may involve use with an AUV (autonomous underwater vehicle) for mapping underwater features and robot navigation.

Chapter 2: Theory of Structured Light Techniques

2.1 Displacement of Parallel Light Strips

Structured light principles rest on the phenomena that a strip of light projected onto a non-planar object will appear to be distorted from any angle other than that of 0°, the angle of the strip of light with respect to its projector source. Placing a camera at an offset angle from the projector allows images of the distortions to be captured, and knowing the geometry between the projector, camera, and object of interest results in the ability to calculate the curvature of the object with respect to the appearance of the distortions.

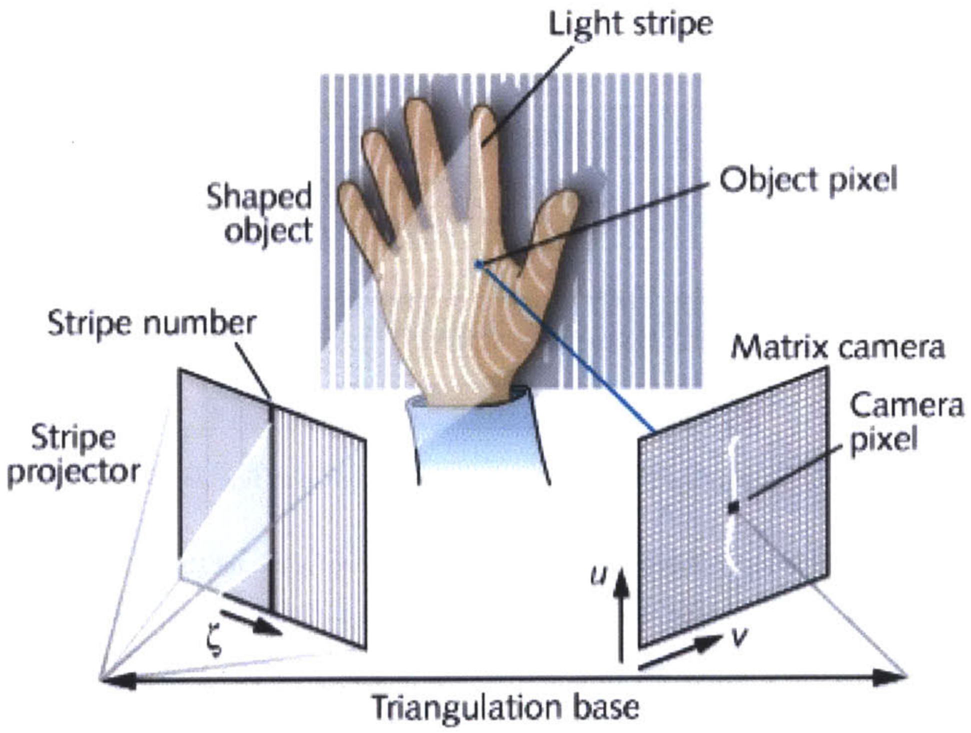

Figure 2-1: Diagram of structured light principles [7].

Projecting a series of light stripes, rather than a single stripe, onto a surface enables a surface area, not a surface line, to be calculated. Therefore it is faster to use this higher sampling method when reconstructing surface curvature. Using a projector with DLP technology results in sharp intensity drop-offs between light and dark stripes, and is utilized here to minimize error.

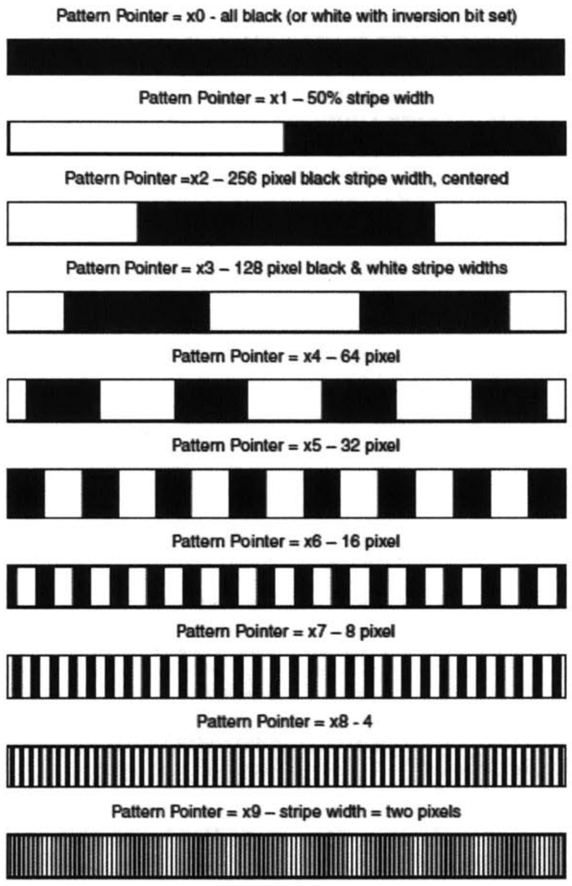
Pattern Pointer = x0 - all black (or white with inversion bit set)
Pattern Pointer = x1 – 50% stripe width
Pattern Pointer =x2 – 256 pixel black stripe width, centered
Pattern Pointer = x3 – 128 pixel black & white stripe widths
Pattern Pointer = x4 – 64 pixel
Pattern Pointer = x5 – 32 pixel
Pattern Pointer = x6 – 16 pixel
Pattern Pointer = x7 – 8 pixel
Pattern Pointer = x8 - 4
Pattern Pointer = x9 – stripe width = two pixels

Figure 2-2: Types of light stripe projections. A series of course to fine projections are shown [8]

Resolution of the 3-D model is correlated to the narrowness of the light stripes. Therefore, projector hardware is usually the limiting factor, because the number of light stripes cannot exceed the number of horizontal pixels projected. Generally the camera component resolution exceeds that of the projector; in this case the camera is VGA (640x480) and the projector is HVGA (480x320). Using more advanced hardware will result in a higher-quality model; however, the primary purpose of this project was to shrink the footprint and power requirements of the system.

It is necessary to project a series of light stripes in succession in order to calculate the object curvature. Each pixel in the camera image is assigned a value of 1, if a white stripe is superimposed, or 0, if a dark stripe is superimposed. As the succession of light stripes progresses, each pixel records a gray code signature of 1s and 0s, which is used to calculate the distance of a point on the model relative to the camera. Computing all pixel signatures results in a 3-D model of the object surface. Increasing the number of projections in the sequence will result in higher resolution and fewer errors, but this requires more time to process and more computing power. Therefore it is a tradeoff between resolution and speed. For static object recreation, this is not an issue, and it is best to project sequences until the projector resolution becomes a limitation.

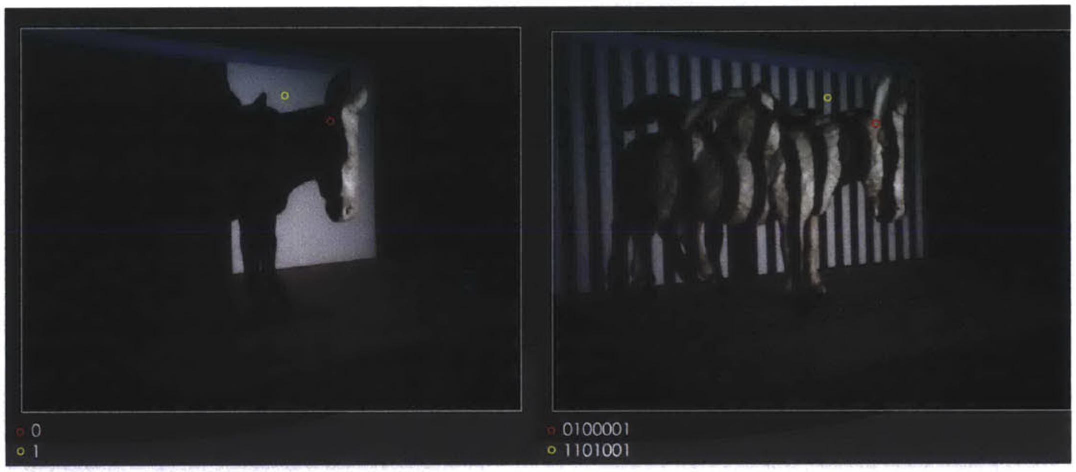

Figure 2.3: Example of light stripe sequence progression. Each pixel is assigned a gray code signature of 1s and 0s, which is used to calculate the distance of that pixel from the camera [9].

The series of projections depicted in Figure 2-2 are taken from two rows of those shown in Figure 1-1. Locations of two different pixels are shown in this example, denoted by red and yellow, and the 1s and 0s in the legend shows the resulting signatures. The picture on the left is the first projection in the series; the picture on the right is the seventh. From these signatures, each pixel's distance can be calculated from the camera when compared to the expected signature, calculated from the undistorted stripe patterns. As a result of analyzing the deformations and calculating distances, a point cloud can be formed, representative of the object being scanned. This .xyz point cloud file can then be viewed using MeshLab, and exported for further manipulation and 3-D printing.

It is important to note that the resulting 3-D model is from the single-cone perspective of the camera; to yield a fully defined model from all perspectives requires scans from multiple positions. Stitching together these scans requires creating a register linking the scans relative to one another. This is beyond the scope of this thesis, however, register algorithms for this exist and it is possible to use the compact structured light 3-D imaging system presented here for such purposes if desired.

2.2 Creating Parallel Light Stripes

Light stripes can be created two different ways. The first involves a projector beaming an image onto an object. The second uses laser interference to create a light stripe pattern. The use of one or the other depends on the application.

Projector methods are low-cost and the equipment is widely available. Using a projector with DLP technology, as opposed to LCD, results in a cleaner projection. Because it operates on the principle of reflection across millions of mirrors, light intensity is very good and drop-off at the fringes is low. This results in linear gray value production. However, the resolution of the 3-D model depends on the pixel resolution of the projector and pixel sub-processing techniques. Therefore, 3-D models using projector methods can be accurate on the level of microns, though advanced projection systems utilizing post-processing cost orders of magnitude higher than simple ones.

Laser interferometry methods are accurate down to the angstrom level, by the nature of the electromagnetic wavelengths used. However, laser systems are complex and costly, and are subject to problems not found in projector methods, like speckle noise [10]. Applications for laser interference systems are mainly found in research laboratories, due to the highly specialized nature of the equipment and the skillset needed to maintain and operate the machinery. This is not feasible for consumer applications.

2.3 Precision and Accuracy

Precision and accuracy of the structured light imaging system depend heavily upon the depth of field, camera resolution, and projector resolution. In theory, it is possible to achieve wavelength-level precision with an optical projector, but in practice, this is infeasible. Methods to deduce additional accuracy include phase shifting, such that sub-portions of a stripe can be resolved, and interpolation over a range of images such that accuracy of 1/50 pixel is possible. Examples of applications in which structured light can be applied are shown below in Table 2-1.

Table 2.1: Accuracy of structured light applications [11]

Application	**Accuracy**
Planarity of 60cm-wide surface	10μm
Shape of engine combustion chamber	2μm
Shape of 2" object	1μm
Radius of knife edge	0.4μm

Precision and accuracy also depends on the surface of the object being scanned. Translucent or reflective materials introduce problems and are difficult or impossible to scan, depending on the severity. These can include organ tissue, wax, clear plastics, and plant matter. More advanced computer processing is needed to interpret the signals received and deduce what is noise and what is not [11].

2.4 Current Applications

Structured light imaging systems are used in a variety of fields for 3-D object model measurement and recreation, though it is more prominent on an industry, not consumer, level.

The most recognized consumer-level application of structured light remains the Microsoft Kinect peripheral for the Xbox 360, which substitutes the player's movements for the traditional gaming controller. Industry and research applications include automotive and medical purposes, such as car seat measurement or combustion chamber volume measurement, and organ scanning and metrology.

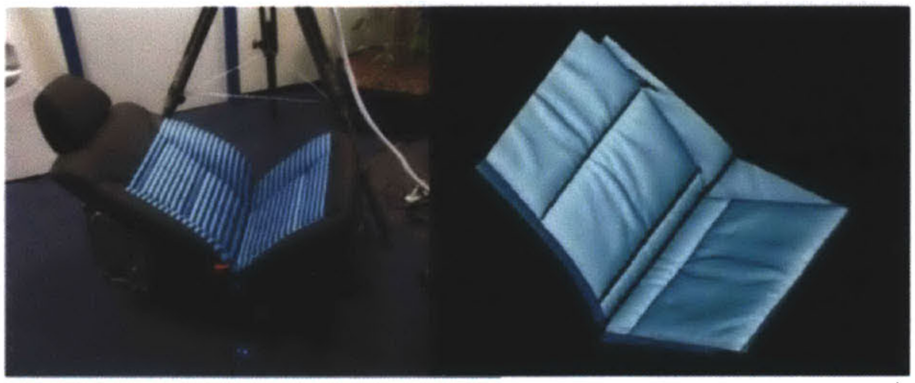

Figure 2.4: Example of structured light used to model automotive applications [12].

Fast fashion clothing companies, such as H&M and Zara, utilize 3-D scanning to reduce template-to-shelf time to a matter of days. Forensics labs utilize scanning for investigative purposes, and toolmakers use it to inspect quality of grinds and polishes, just to mention a few more.

The cost of obtaining and running a 3-D scanning system has fallen such that its adoption into new industries is to be expected. Because of these lower costs and improved hardware, the focus of this thesis was to modify a structured light 3-D imaging system such that it could one day be used for autonomous underwater vehicle navigation systems.

Chapter 3: Construction of the 3-D Imaging System

3.1 Component Selection

This thesis aimed to create a low-cost, low-power, compact and portable underwater 3-D imaging system using the principles of structured light. In order to minimize the footprint and power requirements of the system, electronic components had to be well integrated. The Hackengineer team at Rice University previously developed a 3-D imaging system with a BeagleBoard-xM, pico projector, and camera board. This system basis was then modified to include better battery management and a specialized enclosure for watertightness. The prototype cost was approximately $750, and it can be operated for four hours on battery power.

3.1.1 BeagleBoard-xM Single-Board Computer

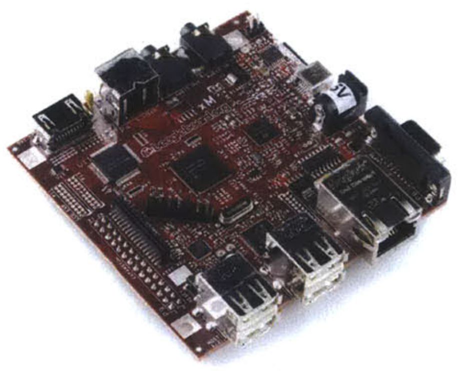

Figure 3-1: Picture of a BeagleBoard-xM [13].

The BeagleBoard-xM is a $150 ARM-processor based open source single board computer chosen for its low cost, low power requirements and multitude of ports. The board measures 3.25"x3.25" and features laptop-like performance and expandability, while keeping hand-held power levels.

Features include a super-scalar 1 GHz ARM Cortex A8 processor, 512MB LPDDR ram, 4 USB 2.0 ports, 10/100 Ethernet, DVI-D output, S-video output, stereo audio out/in, microSD card slot, camera header, RS-232 serial port, and HDMI-out. Other options explored included the Raspberry Pi and BeagleBone. The BeagleBoard-xM was chosen by the Hackengineer team for its HDMI-out port, which could connect to a pico projector.

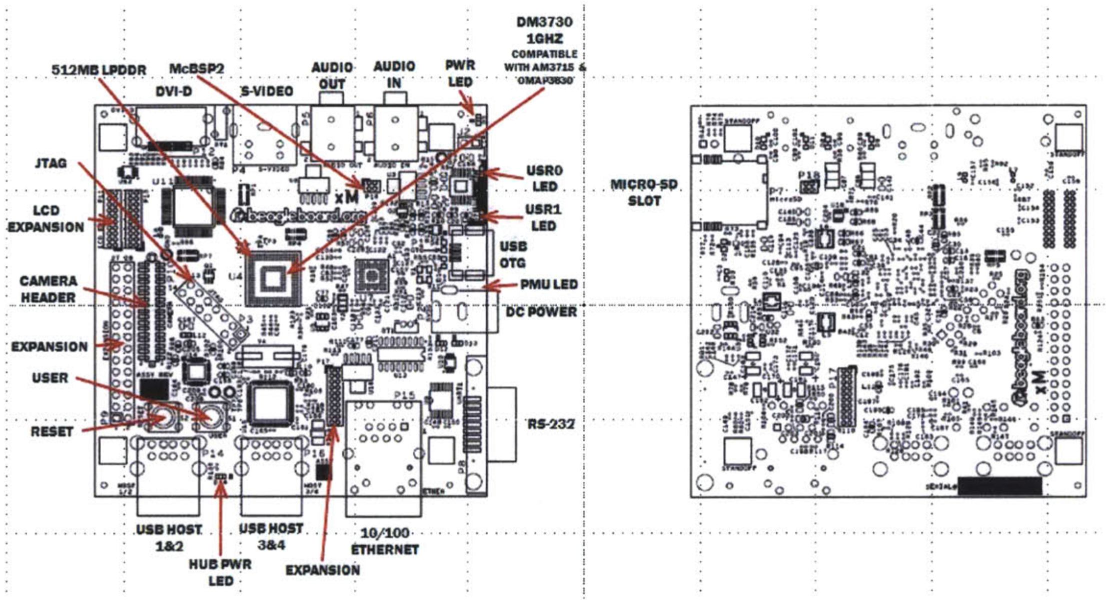

Figure 3-2: Schematic of BeagleBoard-xM [14].

TI developed the BeagleBoard-xM primarily as an open-source, open-hardware educational tool. It can run any Linux distribution, though it was discovered that Ubuntu running Unity GUI consumed 100% of the CPU and was very slow. A modified Angstrom distribution running GNOME was discovered to be more efficient and was chosen for this project.

3.1.2 Liquidware Beaglejuice 2 Hi-Capacity Battery Pack

The Liquidware Beaglejuice 2 Hi-Capacity Battery Pack is an 8,400mAh lithium-ion battery module engineered for use with the BeagleBoard-xM. It is comprised of six battery modules and a power management chip. Its biggest advantage is that it allows the BeagleBoard to be powered in a portable manner. The pack includes 2-pin barrel jacks for each of its four power output ports, and can supply 5V up to 4A sustained. A 19V power supply with 2.1A current rating can recharge the pack in six hours, and LED indicators on the board display what proportion of charge remains.

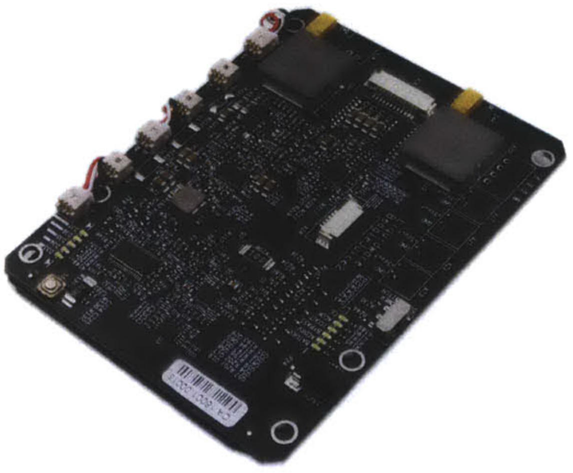

Figure 3-3: Picture of a Beaglejuice 2 Hi-Capacity Battery Pack [15].

The Beaglejuice 2 can supply power for a single BeagleBoard-xM up to 20 hours. For this project, a pico projector was modified to accept power from the Beaglejuice. Though this reduces total battery life for the imaging system to four hours, it allows the entire system to be portable and self-contained.

Standoff holes engineered at key locations on the board allow the Beaglejuice 2 to be mounted underneath the BeagleBone-xM, creating a compact, portable computing unit.

3.1.3 Leopard Imaging LI-LIBCMVGA Camera Module

The Leopard Imaging LI-LIBCMVGA Camera Module is a standard VGA resolution camera with a camera board featuring a 34-pin connection that is compatible with the BeagleBoard-xM. Other options, including higher resolution camera modules, were explored, however, the pico projector chosen features HVGA resolution, which is lower than VGA, so increasing the resolution of the camera module would not increase the resolution of the 3-D model output.

The camera module can be interchanged at a later time with minimal code modification should new technology develop a pico projector with higher resolution. The project can be expanded through interchangeability of modules with minimal modifications to the code.

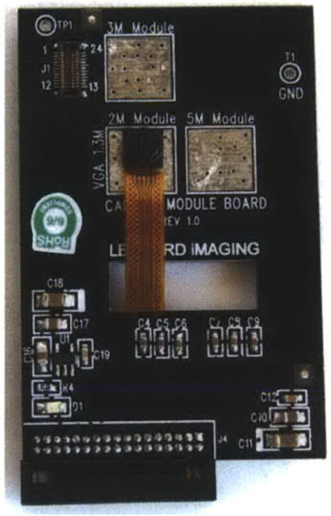

Figure 3-4: Picture of a Leopard Imaging LI-LIBCMVGA Camera Module [16].

3.1.4 TI Pico Projector Development Kit

The TI Pico Projector Development Kit is an HVGA resolution DLP pico projector. It allows for portable projection and structured light to be engineered into mobile applications; its compact 44.8x67.4x14.2mm dimensions are key to the overall small footprint of the imaging system. At the heart of the projector is DLP technology, which incorporates micromirrors that create binary light patterns with speed, precision and efficiency when controlled by the integrated DLP1700 spatial light modulator [8].

Technical specifications include HVGA resolution (480x320), 7 lumen brightness output, 1000:1 contrast ratio, and HDMI input.

Modifications to its power supply cable enabled the projector to be connected to the Beaglejuice 2, because the ultra-low power requirement of the pico projector draws only 5V.

Figure 3-5: Picture of a TI Pico Projector Development Kit [8].

3.1.5 Polycarbonate Waterproof Enclosure

Though the imaging system can operate without an enclosure, it is necessary to waterproof the system for underwater testing. Development toward a AUV navigation system would not be able to occur if the system cannot withstand being submerged in a tank of water.

A Rose-Bopla IP-67 rated polycarbonate enclosure # 331624120 was employed as the casing for the project. The IP-67 is broken down as the following: IP (ingress protection rating) 6 (dust-tight) 7 (waterproof when immersed 1m at 30 minutes). The clear polycarbonate lid allows for the transmission of light from the pico projector through the casing.

The exterior dimensions of the box are 9.45x6.40x4.72" [17].

Preliminary testing of the enclosure suggested that it would perform satisfactorily while submerged, and that the contained electronics would remain dry.

3.2 Initializing Software and Code

This section explains how the BeagleBoard-xM was initialized, and how C++ code was compiled onto the system in order to control the projector and camera, and how an output point cloud file was created. The C++ code was developed by the Hackengineer team at Rice University.

3.2.1 Installing Angstrom and OpenCV Libraries

The Angstrom distribution of Linux was used for its efficiency when running on the BeagleBoard-xM. Ubuntu was tested and installed, but eventually abandoned in favor of Angstrom when it was discovered that the CPU consistently ran at 100%, even after the resource heavy Unity GUI was replaced with GNOME.

The BeagleBoard-xM has no NAND memory, so it runs the operating system off a microSD that the user supplies. Therefore a Linux image must be created, downloaded, and packaged onto the card. Fortunately, a tool called Narcissus exists online, which can be used to create rootfs images for BeagleBoard applications.

At narcissus.angstrom-distribution.org, the following steps were taken to create a tar.gz image of an Angstrom Linux distribution with OpenCV libraries and native toolchain ability for code compilation directly on the BeagleBoard-xM [18].

Machine: beagleboard
Image name: thesis
Complexity: advanced
Release (default): 2011.03
Base system (default): regular (task-base)
/dev manager (default): udev
Type of image: tar.gz
Software manifest: no
SDK type: simple toolchain
SDK hostsystem (default): 32bit Intel
User environment selection: X11
X11 desktop environment: blank

Additional packages selections:
Development packages: OpenCV headers and libs
Additional console packages: All kernel modules, OpenCV
Network related packages: NetworkManager, NetworkManager GUI applet

Platform specific packages: OMAP Display Sub System (DSS)
Documentation: BeagleBoard validation GUI extras, BeagleBoard validation GNOME image

The tar.gz image was then downloaded onto a Linux workstation. A microSD card was inserted into the media reader slot and discovered to have path `/dev/sdb` after running the `–df` command.

The microSD card was formatted into two partitions for a FAT boot and ext3 file system with the following commands [19]:

```
mkdir angstrom-wrk
cd ~/angstrom-wrk
wget
http://cgit.openembedded.org/cgit.cgi/openembedded/plain/contrib
/angstrom/omap3-mkcard.sh
chmod +x omap3-mkcard.sh
sudo ./omap3-mkcard.sh /dev/sdb
sync
```

This resulted in the creation of two mounted directories:

```
/media/boot
/media/Angstrom
```

Copying the MLO, uImage, and u-boot files within the tar.gz image to the boot partition was executed with the following command:

```
tar --wildcards -xzvf thesis.tar.gz ./boot/*
cp boot/MLO-* /media/boot/MLO
cp boot/uImage-* /media/boot/uImage
cp boot/u-boot-*.bin /media/boot/u-boot.bin
sync
```

Copying the file system onto the Angstrom partition was achieved with the following code:

```
sudo tar -xvj -C /media/Angstrom –f thesis.tar.gz
sync
```

The microSD card was inserted into the BeagleBoard-xM, and left alone for an hour to configure itself.

The opkg command was used to ensure that all libraries, including OpenCV were up to date. For example, updating OpenCV included the following commands:

```
opkg update
opkg install opencv opencv-dev opencv-samples opencv-dev-samples
```

It was later discovered while attempting to run the structured light executable file that the OpenCV library files were not linked, so it was necessary to create symbolic links between .2.2 and .2.2.0 files using the `ln -s` command.

A uEnv.txt modification was necessary to recognize the camera board module within the u-boot environment for the 2.6.32 Angstrom build; adding the following to the file within the boot directory made the correction:

```
camera=lbcmvga
```

A driver module for the kernel was inserted and initialized with the following code:

```
opkg install kernel-module-mt9v113
setenv camera lbcmvga
```

At this point the BeagleBoard-xM was configured for supporting the C++ files that would run the scan and create the point cloud. The next section discusses the compilation of C++ files to run the structured light sequence between the pico projector and camera, and capture the images, analyze the pixels, and output a .xyz point cloud file.

3.2.2 Preparing Structured Light Images and Makefile

To create a structured light scan, two things were needed: a series of graycoded stripe images to project, like those in Figure 2-2, and code files to run the sequence of images, capture the camera output, analyze the pixels, and output a point cloud .xyz file.

A sequence of 20 stripe images and a C++ structured light algorithm sl.cpp were downloaded from the Hackengineer Google project repository. Another C++ file clGPIO.cpp configuring the capture button located on the enclosure was downloaded to control the GPIO139 pin out [20].

The sl.cpp file is broken into three sections. Part one initializes the camera using OpenCV libraries and compiles an image sequence of increasingly fine stripe projections. The images are projected through the pico projector and superimposed over the object being scanned, while the camera captures one image per stripe projection.

Part two includes the decoding and depth-rendering algorithm, determining whether a pixel is assigned a 1 (light) or 0 (dark) value for each respective stripe image projection overlay. The discrepancy between the original stripe image output and the camera capture files is computed, and used to determine the depth of the object relative to the camera for each pixel. A point cloud is rendered using the resulting data, and stored in the Results folder within the 3DScanner repository, or onto a thumb drive if detected by the code.

Part three cleans up possible errors by eliminating outlier points caused by noise and other causes.

The slGPIO.cpp file is used to execute the sl.cpp file when a physical button on the exterior of the casing is pressed. It is assigned to trigger the executable when voltage to the 139 GPIO pin is interrupted by the button action.

A makefile combining the sl.cpp and slGPIO.cpp files into an executable was run using the `make` command within the 3DScanner directory containing the sl.cpp, slGPIO.cpp, images, and makefile.

3.3 Assembling the System

The core module of the imaging system is comprised of the BeagleBoard-xM, Beaglejuice 2, and camera module. This module was then mounted inside the enclosure with a custom plastic bracket, and connected to the pico projector, on its own separate bracket. Securing the components to brackets with velcro allowed for ease of removal and insertion, which is important, as the module must be removed for charging and downloading output .xyz files. A waterproof pushbutton for initiating the 3-D image capture executable was installed into the top of the enclosure and wired to the core module.

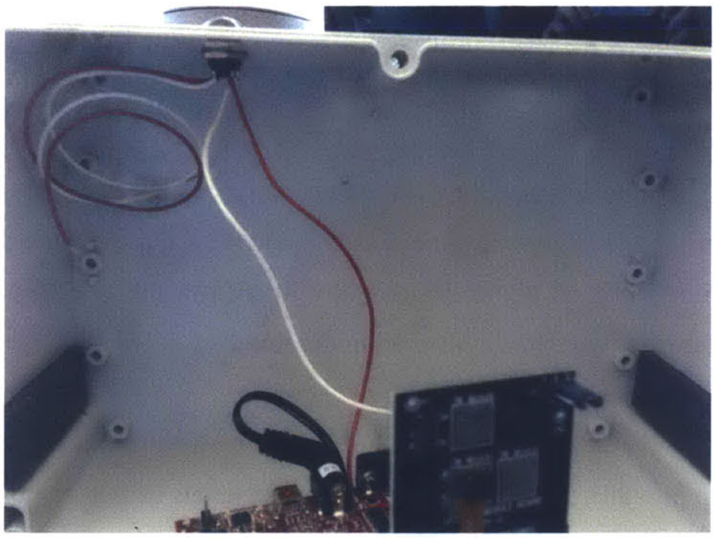

Figure 3-6: Wiring the pushbutton to the core module.

Before mounting the BeagleBoard-xM to the Beaglejuice 2, GPIO pin 139 and ground were prewired in preparation to connect with the waterproof pushbutton. Wiring the pin first was necessary, because the header exists on the underside of the BeagleBoard-xM and was inaccessible after mating with the Beaglejuice 2.

Plastic standoffs were then used to attach the BeagleBoard-xM to the Beaglejuice 2 at four overlapping locations on the PCBs. The BeagleBoard-xM was oriented above the Beaglejuice 2 such that power input to the BeagleBoard was situated directly above power output from the Beaglejuice 2. A 2-barrel connecter was used to supply 5V power to the BeagleBoard-xM.

The camera module was then inserted into the 34-pin camera header on the BeagleBoard-xM, completing the construction of the core module. A magnetic 2x telephoto lens was attached to the camera using a washer so that its frame of vision would match the projector throw. Without this lens, the stripe projections would only fill a quarter of the camera's field of vision.

The pico projector, prior to connection with the BeagleBoard-xM, had to have its power cord modified. Originally, the device sourced power from a wall plug that connected to a transformer brick. Cutting away the brick, stripping the wire, and soldering the leads to a salvaged 2-barrel connector yielded a cord that could be plugged into the Beaglejuice 2 for power.

Two brackets were made within the enclosure, one to support the core module and one to support the pico projector. Mounting the pico projector and camera module at the same height was critical for accurate scanning, as the algorithm is set up for minimal homography.

Soldering the leads from the waterproof pushbutton to the wires prepared on the BeagleBoard-xM, and then mounting the core module and pico projector to their respective brackets completed the build. The polycarbonate cover was then reattached with screws.

The following picture shows the completed 3-D imaging system turned on. Notice the push button on top to activate image capture, placed such that it is analogous to a normal 2-D camera.

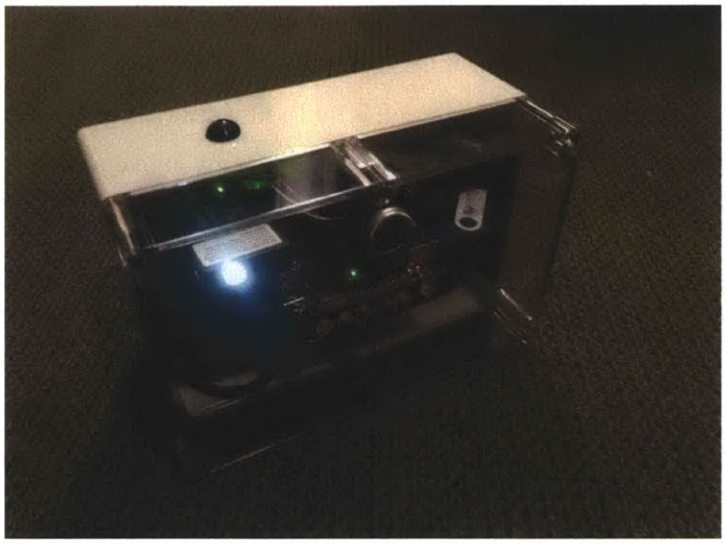

Figure 3-7: Completed underwater structured light 3-D imaging system.

Chapter 4: Instructions for Operation

4.1 Using the Structured Light 3-D Imaging System

It is first necessary to charge the system. Remove polycarbonate cover from the enclosure by removing the six screws accessible from the top, and remove the core module from the bracket by pulling up and away from the velcro. If necessary, unplug the pico projector, or remove it as part of the core module. Be careful to not pull the module out forcefully, because it remains tethered by the wire leads connected to the pushbutton switch mounted on top of the enclosure. Insert the 19V barrel pin into the DC port located on the Beaglejuice 2. Charging may take up to 6 hours; status can be checked upon pressing the white indicator nub on the Beaglejuice 2. A series of LEDs like those on the Macbook Pro will indicate charge status.

When charging is complete, turn on the power to the Beaglejuice 2 by operating the switch located next to its power outputs. Insert the module in the reverse order of which it came out, and re-secure the pico projector. Do not reattach the cover yet. Press the power button on the pico projector to turn it on. Ensure that the projector focus is correct for the desired distance from the object of interest by using the focus scroll wheel on the side of the projector. The BeagleBoard-xM will automatically power on once voltage is detected, and the system will auto-login. Use password "admin" if needed.

After plugging in a keyboard and mouse to the USB slots, navigate to the 3DScanner folder with terminal using commands `ls` and `cd`. Insert a USB thumb drive for storing output .xyz point cloud files into a USB port. Run the slGPIO executable with command `./slGPIO`. This will activate the first scan sequence, which is sacrificial. Remove the mouse and keyboard, and reattach the cover tightly while leaving the USB thumb drive attached in the USB port.

Now the system is ready to use. Press the pushbutton on top to begin capturing a 3-D image. The sequence of stripes will be projected from the pico projector, and the camera will capture images for each projection. A point cloud file will automatically be written onto the USB thumb drive.

Each rendering will take approximately 10 seconds. Press the pushbutton again to capture another scan, and repeat as desired. The system will last approximately four hours per charge.

4.2 Rendering the Output Files with MeshLab

Postprocessing of each .xyz file is required to obtain a 3-D model. This can be done with MeshLab, a free open-source point cloud renderer.

To begin, open the enclosure cover and retrieve the USB thumb drive. The .xyz point file clouds will be arranged chronologically as result_0.xyz, result_1.xyz, etc.

Install MeshLab onto the workstation if it is not already present. Import the desired .xyz mesh file into MeshLab. The program will render the point cloud. Eliminate any erroneous points or undesired sections, if need be, using the delete vector point command.

Compute normal with Filters > Point Set > Compute Normals For Point Sets and set the number of neighbors to 100. Check the “flip normals with respect to viewpoint” box, and make sure Render > Lighting > Light On is enabled. Continue to Filters > Color Creation and Processing > Per Vertex Color Function and modify the values to best fit the application.

Export the file in whatever desired format is available, such as .3df, .obj, or .stl. MeshLab supports a wide variety of file formats.

Chapter 5: Example 3-D Images and Discussion

5.1 Above Water Scanning

The system was first tested dry to verify the performance of the structured light system.

A small plastic dinosaur toy approximately 1" tall was chosen to be scanned because of its size and detail. The dinosaur had four legs, a long neck and a head with a bump on top. Its skin was ribbed with a rough texture.

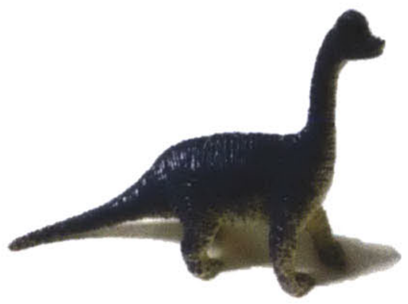

Figure 5-1: Image of a dinosaur toy.

Scanning a toy of such detail would test the limitations of the 3-D imaging system. Would the system detect the toy at all, because of its small size? How well would it be able to trace the shape of the body? Would it be able to capture details, such as the ribbing on the skin and the bump on top of the head?

Using the 3-D imaging system yielded the following model, after cleaning away erroneous points in MeshLab:

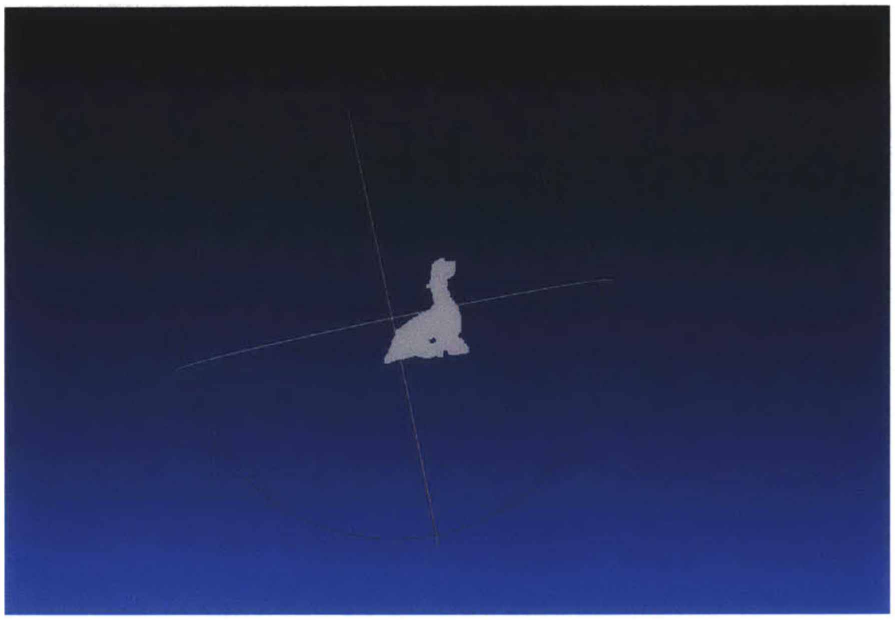
Figure 5-2: 3-D scan of a dinosaur toy.

The structured light 3-D imaging system successfully capture the shape of the dinosaur toy, and was able to recreate features such as the bump on its head and spacing in between legs from a scan distance of one foot. However, the resolution was not fine enough to detect the ribbing on the skin. This was not surprising, due to the lack of resolution in the pico projector and camera board module. At a scanning distance of one foot, the calculated resolution was 0.635mm. The ribbing was finer than this, so it could not be recreated. Four cavities in the surface from bad graycode signatures due to misread stripe patterns were discovered, but not deemed to be significant. Modifying the system in future iterations with high-resolution hardware would fix detail and cavity issues.

5.2 Submerged Underwater Scanning

After successful verification of the system above water, the system was submerged in a tank for underwater testing.

The 30 gallon tank was filled with tap water and placed in a dark room. One cupful of dirt was added to the mixture to help simulate conditions in an outdoor body of water. The system was placed one foot away from the area where objects would be submersed for scanning.

The dinosaur was not detected during its initial underwater scan, due to the addition of suspended particles scattering light in all directions, so a larger object had to be chosen. A mug, due to its large and simple shape, was chosen to be the basis of underwater testing.

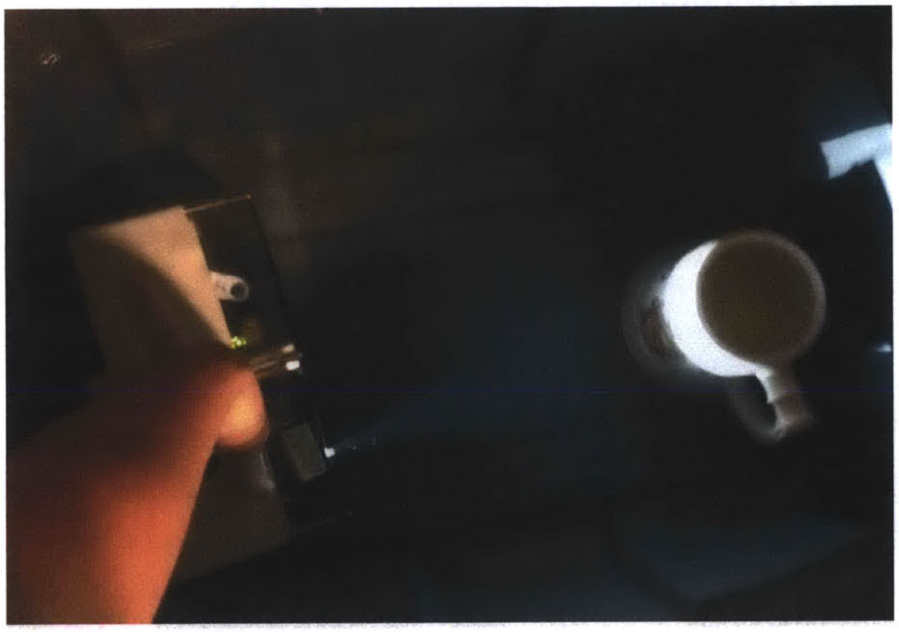
Figure 5-3: Image of scanning a mug underwater.

The mug featured a large, cylindrical body with an ornamented handle, and measured approximately 7" in height and 6" across.

Scanning underwater introduced new variables that would challenge the system's ability to scan an object with good fidelity. Three main concerns were that the system would take on water due to insufficient seals or a leaking pushbutton, the projector brightness would be insufficient, and that the introduced particles suspended in the water tank would create movement and scatter light, thereby introducing error into the scans. Because structured light scanning takes time to complete, it was postulated that movement of obstructing particles would introduce errors into the pixel graycode signatures, which would then result in miscalculations when determining distances for the point cloud rendering.

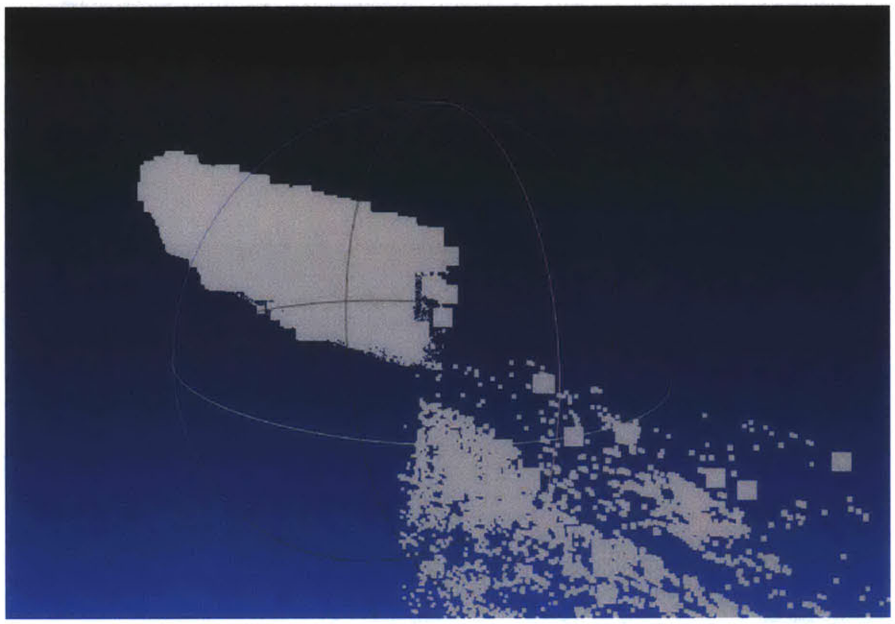
Figure 5-4: 3-D scan of a mug underwater.

A resulting scan of the mug is shown above. As suspected, the addition of suspended particles caused severe scattering of light. This image shows the point cloud from a shifted angle – not straight on, from the camera perspective. One can see that the system recorded the suspended particles, and therefore was unable to "see" parts of the mug behind the dirt. However, the mug was detected, as the large body shown. There appears to be a gap between the body and the handle. This scan is unusable for any metrology or recreation purposes, but it shows that the system can work in terms of object detection.

The test was repeated to verify this phenomenon, and all yielded similar results. However, the system was sufficiently watertight and did not draw on any water during the testing period, being able to withstand 5-minute intervals submerged underwater, and the pico projector brightness was sufficient at close range, alleviating two of the three concerns enumerated earlier.

An algorithm for eliminating scattering, and using the Kinect style of 2-D structured light patterns for instantaneous processing would significantly improve scanning in future iterations. Additionally, autofocus for scanning a large range of distances should be implemented in subsequent builds.

Taking these suggestions into account during the next build iteration would result in the next step toward an on-the-fly mapmaking navigation unit for a AUV.

Chapter 6: Summary and Conclusions

A waterproof structured light 3-D imaging system was created as a proof-of-concept for classroom and laboratory research purposes. The primary objective was to construct a functioning underwater system that satisfied footprint and power requirements, yielding a low-cost, compact, and portable unit. The result was an imaging system that could be explored further one day to create an autonomous underwater vehicle (AUV) navigation system.

The proof-of-concept was built for approximately $750, and was comprised of a low-power, open-source hardware single-board computer running a modified Linux distribution with OpenCV libraries, a DLP pico projector, camera board, and battery module with advanced power management. Hardware component selection and code was based off of previous work by the Hackengineer team at Rice University. A watertight enclosure enabled testing of the system in rugged environments that would normally be inaccessible to other 3-D imaging systems.

An instruction guide was included for operation of the system. Future improvements to the proof-of-concept system may include automatic execution of the slGPIO script so that a mouse and keyboard are not necessary to initialize the system, and an external power port.

Rendering low-detail objects was successful after post-processing for above-water scans. The system can reproduce models with accuracy on the order of millimeters, although noise from the low-resolution camera introduces cavities and errors. Range was limited by manual focus on the projector, and image resolution was limited by the output resolution of the pico projector and camera. Further exploration in obtaining ultra high-resolution 3-D images may include use of a higher resolution projector and a higher resolution camera board module.

During underwater scanning, backscattering caused by reflective particles suspended in the water and hindered performance severely. The system was able to detect objects, but the output point cloud was accompanied by noise and unacceptably low levels of detail for purposes other than object detection. However, watertightness of the system was satisfactory, and the imaging system was able to stay submerged underwater in multiple five-minute increments during testing without taking on water. Pico projector brightness was sufficient for close-range testing, although it would be insufficient for scanning farther beyond 6-7 feet.

The results were useful in determining that structured light scanning using a sequence of stripes is not ideal for underwater usage. Factors introduced by submerging the system in water limited its performance by preventing clear, detailed scanning. As such, it is recommended that advances in filtering algorithms that can compensate for reflective suspended particles and fast-response 3-D scanning systems, such as the Kinect's QR code-like structured light pattern or time-of-flight, be used in future development of a AUV navigation system.

Chapter 7: Appendices

Appendix A List of Linux Packages

toolchain
initscripts
opencv-dev
opencv-samples
opencv-samples-dev
python-core
python-modules
sysvinit
sysvinit-pidof
task-native-sdk

Appendix B Camera Module Installation Code [20]

```
camera=lbcmvga
vram=16M


opkg install kernel-module-mt9v113
```

Appendix C GPIO Button Shutter Code slGPIO.cpp [20]

```
/* Copyright (c) 2011, RidgeRun
 * All rights reserved.
 *
From https://www.ridgerun.com/developer/wiki/index.php/Gpio-int-test.c
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    This product includes software developed by the RidgeRun.
 * 4. Neither the name of the RidgeRun nor the
 *    names of its contributors may be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY RIDGERUN ''AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL RIDGERUN BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>
#include <signal.h>     // Defines signal-handling functions (i.e. trap Ctrl-
C)
#include "sl.cpp"

 /****************************************************************
 * Constants
 ****************************************************************/

#define SYSFS_GPIO_DIR "/sys/class/gpio"
#define POLL_TIMEOUT (3 * 1000) /* 3 seconds */
#define MAX_BUF 64

/****************************************************************
 * Global variables
 ****************************************************************/

int keepgoing = 1;      // Set to 0 when ctrl-c is pressed

/****************************************************************
 * signal_handler
 ****************************************************************/
// Callback called when SIGINT is sent to the process (Ctrl-C)
void signal_handler(int sig)
{
      printf( "Ctrl-C pressed, cleaning up and exiting..\n" );
      keepgoing = 0;
}

/****************************************************************
 * gpio_export
 ****************************************************************/

int gpio_export(unsigned int gpio)
{
      int fd, len;
      char buf[MAX_BUF];
      fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
      if (fd < 0) {
            perror("gpio/export");
            return fd;
      }
      len = snprintf(buf, sizeof(buf), "%d", gpio);
      write(fd, buf, len);
      close(fd);
      return 0;
}

/****************************************************************
 * gpio_unexport
 ****************************************************************/

int gpio_unexport(unsigned int gpio)
{
      int fd, len;
      char buf[MAX_BUF];
      fd = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
      if (fd < 0) {
            perror("gpio/export");
            return fd;
      }
      len = snprintf(buf, sizeof(buf), "%d", gpio);
      write(fd, buf, len);
      close(fd);
      return 0;
}

/****************************************************************
 * gpio_set_dir
 ****************************************************************/

int gpio_set_dir(unsigned int gpio, unsigned int out_flag)
{
      int fd, len;
      char buf[MAX_BUF];
      len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR  "/gpio%d/direction",
gpio);
      fd = open(buf, O_WRONLY);
      if (fd < 0) {
            perror("gpio/direction");
            return fd;
      }
      if (out_flag)
            write(fd, "out", 4);
      else
            write(fd, "in", 3);
      close(fd);
      return 0;
}

/****************************************************************
 * gpio_set_value
 ****************************************************************/
```

```
int gpio_set_value(unsigned int gpio, unsigned int value)
{
	int fd, len;
	char buf[MAX_BUF];
	len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);
	fd = open(buf, O_WRONLY);
	if (fd < 0) {
		perror("gpio/set-value");
		return fd;
	}
	if (value)
		write(fd, "1", 2);
	else
		write(fd, "0", 2);
	close(fd);
	return 0;
}

/****************************************************************
 * gpio_get_value
 ****************************************************************/

int gpio_get_value(unsigned int gpio, unsigned int *value)
{
	int fd, len;
	char buf[MAX_BUF];
	char ch;
	len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);
	fd = open(buf, O_RDONLY);
	if (fd < 0) {
		perror("gpio/get-value");
		return fd;
	}
	read(fd, &ch, 1);
	if (ch != '0') {
		*value = 1;
	} else {
		*value = 0;
	}
	close(fd);
	return 0;
}


/****************************************************************
 * gpio_set_edge
 ****************************************************************/

int gpio_set_edge(unsigned int gpio, char *edge)
{
	int fd, len;
	char buf[MAX_BUF];
	len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/edge", gpio);
	fd = open(buf, O_WRONLY);
	if (fd < 0) {
		perror("gpio/set-edge");
		return fd;
```

```
        }
        write(fd, edge, strlen(edge) + 1);
        close(fd);
        return 0;
}

/****************************************************************
 * gpio_fd_open
 ****************************************************************/

int gpio_fd_open(unsigned int gpio)
{
        int fd, len;
        char buf[MAX_BUF];
        len = snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);
        fd = open(buf, O_RDONLY | O_NONBLOCK );
        if (fd < 0) {
                perror("gpio/fd_open");
        }
        return fd;
}

/****************************************************************
 * gpio_fd_close
 ****************************************************************/

int gpio_fd_close(int fd)
{
        return close(fd);
}

/****************************************************************
 * Main
 ****************************************************************/

int main(int argc, char **argv, char **envp)
{
        struct pollfd fdset[2];
        int nfds = 2;
        int gpio_fd, timeout, rc;
        char *buf[MAX_BUF];
        unsigned int gpio;
        int len;
        //if (argc < 2) {
        //      printf("Usage: gpio-int <gpio-pin>\n\n");
        //      printf("Waits for a change in the GPIO pin voltage level or input
on stdin\n");
        //      exit(-1);
        //}
        // Set the signal callback for Ctrl-C
        signal(SIGINT, signal_handler);
        //gpio = atoi(argv[1]);
        gpio = 139;
        gpio_export(gpio);
        gpio_set_dir(gpio, 0);
        gpio_set_edge(gpio, "falling");  // Can be rising, falling or both
        gpio_fd = gpio_fd_open(gpio);
        timeout = POLL_TIMEOUT;
```

```
      while (keepgoing) {
            memset((void*)fdset, 0, sizeof(fdset));
            fdset[0].fd = STDIN_FILENO;
            fdset[0].events = POLLIN;

            fdset[1].fd = gpio_fd;
            fdset[1].events = POLLPRI;
            rc = poll(fdset, nfds, timeout);
            if (rc < 0) {
                  printf("\npoll() failed!\n");
                  return -1;
            }
            if (rc == 0) {
                  printf(".");
            }
            if (fdset[1].revents & POLLPRI) {
                  lseek(fdset[1].fd, 0, SEEK_SET);  // Read from the start of
the file
                  len = read(fdset[1].fd, buf, MAX_BUF);
                  //printf("\npoll() GPIO %d interrupt occurred, value=%c,
len=%d\n",gpio, buf[0], len);
                  printf("Starting Scan\n");
                  slScanner();
                  printf("Scan Complete\n");
                  printf("Press button to scan again or Ctrl C to quit");
            }
            if (fdset[0].revents & POLLIN) {
                  (void)read(fdset[0].fd, buf, 1);
                  printf("\npoll() stdin read 0x%2.2X\n", (unsigned int)
buf[0]);
            }
            fflush(stdout);
      }
      gpio_fd_close(gpio_fd);
      return 0;
}
```

Appendix D Structured Light Algorithm for BeagleBoard-xM [20]

A Makefile is used to call and run scripts sl.cpp and slCPIO.cpp. This outputs a .xyz file, which can then be exported to MATLAB for processing. This code was created by the Hackengineer team at Rice University.

D.1 Makefile [20]

```
# 'make'        build executable file 'mycc'
# 'make clean'  removes all .o and executable files

.PHONY: depend clean

SCAN = slGPIO
DISPLAY = display

CC = gcc
CFLAGS := -g -Wall -DLINUX $(shell pkg-config opencv --cflags)
LIBs := $(shell pkg-config opencv --libs)
LFLAGS =

# Source files
SRCs := \
      slGPIO.cpp
      #sl.cpp \


# List of object files to compile
OBJs = $(SRCs:.cpp=.o)

all: $(SCAN)

$(DISPLAY): display.o
      gcc -g -Wall

$(SCAN): $(OBJs)
      $(CC) $(CFLAGS) $(INCLUDES) -o $(SCAN) $(OBJs) $(LFLAGS) $(LIBs)


#     cp $(SCAN) Release/

%.o: %.cpp %.h
      $(CC) $(CFLAGS) $(INCLUDES) -c $<  -o $@

clean:
      $(RM) *.o *~ $(SCAN)
```

D.2 Structured Light Sequence sl.cpp [20]

```
////////////////////////////////////////////////////////////////////////
// sl.cpp
//
//*********PART1 - Project SL and Cap Imgs************
//Sequentially project and capture 20 images
//
//*********PART2 - Decode captured images************
//
//*********PART3 - Project 3D object with openGL********
//
////////////////////////////////////////////////////////////////////////

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <assert.h> // For my own paranoia's sake
#include <opencv/cv.h>
#include <opencv/highgui.h>
#define WHITE_THRESHOLD 5
#define BLACK_THRESHOLD -5
#define NUM_IMAGES 18
#define CAM_WIDTH 640
#define PROJ_WIDTH 640
#define Z_SCALE 500   //250 works
//#define TEST_RECOGNIZER
int gray2bin(int gray);
bool file_exists(const char * filename);
int slScanner(void);
int slScanner(void)
{
      //*********PART1 - Project SL and Cap Imgs************
      //Initialize
      IplImage* slPat[NUM_IMAGES] = {0};
      IplImage* capImg[NUM_IMAGES + 1] = {0};
//    int height,width,step,channels;
      uchar *data;
      char s[128];
      int i,outFileNum=0;
      //Usage
      //if(argc>1){
      //    printf("Usage: sl images are in same folder named grayIMGx.png
and range from 0-19\n");
      //    exit(0);
      //}
      //Load sl images
#ifndef TEST_RECOGNIZER
      for(int i=0;i<NUM_IMAGES;i++){
            sprintf(s,"%d.png",i+1);
            slPat[i]=cvLoadImage(s);
            if(!slPat[i]){
                  printf("Could not load image file: %s\n",s);
                  exit(0);
            }
      }
      // create a window
      cvStartWindowThread();
      cvNamedWindow("mainWin", CV_WINDOW_NORMAL);
      cvSetWindowProperty("mainWin", CV_WND_PROP_FULLSCREEN,
CV_WINDOW_FULLSCREEN);
      // pause
      cvWaitKey(500);
      // set up camera
      CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
      if (!capture){
            fprintf( stderr, "Camera not found \n");
            getchar();
            exit(0);
      }
      cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 480);
      cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 640);
      //cvSetCaptureProperty(capture, CV_CAP_PROP_FPS, 5);
      //cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_COUNT, 1);
      //Maybe this will fix the frame delay?
      //Display images and capture
      for(int i=0;i<NUM_IMAGES;i++){
            // show the image
            cvShowImage("mainWin", slPat[i] );
            // pause
            cvWaitKey(5);
            // capture the image
            cvGrabFrame(capture);
            cvGrabFrame(capture);
            cvGrabFrame(capture);
            cvGrabFrame(capture);
            cvGrabFrame(capture); //Get to last frame in buffer
            capImg[NUM_IMAGES + 1] = cvRetrieveFrame(capture);
            // write the image
            //cvWriteFrame(writer, capImg[i]);
            capImg[i] = cvCloneImage(capImg[NUM_IMAGES + 1]);
            sprintf(s,"results/%d.png",i+1);
            cvSaveImage(s,capImg[i]);
      }
      cvReleaseCapture(&capture);
      //Display captured images for debug
      for(int i=0;i<NUM_IMAGES;i++){
            // show the image (for debug)
            cvShowImage("mainWin", capImg[i] );
            // pause
            cvWaitKey(100);
      }
#else // TEST_RECOGNIZER
      for(int i=0;i<NUM_IMAGES;i++) {
            sprintf(s,"setupp/%d.png",i+1);
            capImg[i]=cvLoadImage(s);
            if(!capImg[i]){
                  printf("Could not load image file: %s\n",s);
                  exit(0);
            }
      }
#endif
      //*********PART2 - Decode captured images************
      // First initialize a matrix for showing the decoded values, needs 10
bits,
	// so could move to short* if memory becomes an issue.
	int *columnmap = (int*)calloc(capImg[0]->imageSize, sizeof(int));
	if(columnmap == NULL) {
		printf("Allocation for column map failed! Aborting\n");
		return 1;
	}
	// We're going to step over matching inverse pairs of projected
patterns, and
	// perform a threshold operation to determine whether a given pixel is
white
	// or not.
	for(int i=0;i<NUM_IMAGES;i+=2){
		printf("Processing scan patterns %d, %d\n",i,i+1);
		uchar* data1 = (uchar*)capImg[i]->imageData;
		uchar* data0 = (uchar*)capImg[i+1]->imageData;
		for(int y=0;y<capImg[i]->height;y++){
			for(int x=0;x<capImg[i]->width;x++) {
				int u = y*capImg[i]->widthStep+x*capImg[i]-
>nChannels;
				if(columnmap[u] == -1){
					// We have discarded this pixel
					continue;
				}
				int gray1 = data1[u] + data1[u+1] + data1[u+2];
				int gray0 = data0[u] + data0[u+1] + data0[u+2];
				// Shift our Gray code bit vector
				columnmap[u] = columnmap[u] << 1;
				if(gray1 - gray0 > WHITE_THRESHOLD){
					// We have a white pixel!
					assert((columnmap[u] & 1) == 0);
					columnmap[u] |= 1;
				} else if(gray1 - gray0 < BLACK_THRESHOLD){
					// A black pixel
					assert((columnmap[u] & 1) == 0);
					columnmap[u] |= 0;
				} else {
					// Not quite sure about this pixel: Punt!
					columnmap[u] = -1;
				}
			}
		}
		// Avoid memory leaks!
//		printf("Releasing images %d, %d\n", i,i+1);
//		cvReleaseImage(&capImg[i]);
//		cvReleaseImage(&capImg[i+1]);
	}
	// Convert Gray code to decimal
	for(i = 0; i < capImg[0]->imageSize; i++){
//		int tmp = columnmap[i];
		columnmap[i] = gray2bin(columnmap[i]);
//		printf("Gray: %4x -> Binary %4x\n",tmp,columnmap[i]);
	}
	// TODO: Test against projector patterns -> 0 disparity
	// TODO: Test with SL images
    //Find available filename on jumpdrive
    sprintf(s,"/media/sda1/result%d.xyz",outFileNum);
    while (file_exists(s)) {
        outFileNum++;
        sprintf(s,"/media/sda1/result%d.xyz",outFileNum);
    }
      // Write the pointcloud
      FILE *f = fopen(s,"w");
      if(f == NULL) {
        printf("Error opening %s:  Trying local directory...\n",s);
        FILE *f = fopen("result.xyz","w");
        if(f == NULL) {
            printf("Error opening result.xyz:  Aborting!\n");
            return 1;
        }
      }
      printf("Writing pointcloud...\n");
      for(int y = 0;y < capImg[0]->height; y++){
            for(int x = 0;x < capImg[0]->width; x++) {
                  int u = y*capImg[0]->widthStep+x*capImg[0]->nChannels;
                  if(columnmap[u] == -1) continue; // Discarded!
                  float disp = (float)columnmap[u]/(PROJ_WIDTH) -
(float)x/CAM_WIDTH;
//                printf("%f\n",disp);
                  if(disp == 0.0) continue;
                  float z = (float)Z_SCALE/(disp);
                  fprintf(f,"%d %d %f\n",x,y,z);
            }
      }
      fclose(f);
      //*******Part3 - Project 3D object with openGL*********
      for(int i = 0; i < NUM_IMAGES; i++){
            cvReleaseImage(&capImg[i]);
            //cvReleaseImage(&slPat[i]);
      }
      //Exit
      cvDestroyWindow("mainWin");
      free(columnmap);
      return 0;
}
// Convert a 10-bit Gray code value into a binary integer representation
// Thanks to http://www.dspguru.com/dsp/tricks/gray-code-conversion
int gray2bin(int gray){
      unsigned int temp = gray ^ (gray>>8);
      // Our error condition should propagate
      if(gray == -1) return -1;
      temp ^= (temp>>4);
      temp ^= (temp>>2);
      temp ^= (temp>>1);
      return temp;
}
bool file_exists(const char * filename)
{
    if (FILE * file = fopen(filename, "r"))
    {
        fclose(file);
        return true;
    }
    return false;
}
```

Chapter 8: Bibliography

[1] *3D Scanners: A Guide to 3D Scanner Technology.* Photograph. rapidform.com. Web. 16 April 2013. <http://www.rapidform.com/3d-scanners/>.

[2] Iddan, Gavriel, and Giora Yahav. "3D Imaging in the Studio and Elsewhere." *Proceedings of SPIE.* 4298 (2003): 48.

[3] Schuon, Sebastian, Christian Theobalt, James Davis, and Sebastian Thrun. "High-Quality Scanning Using Time-of-Flight Depth Superresolution." *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops.* (2009): 1-7.

[4] Marshall, Gerald. *Handbook of Optical and Laser Scanning.* Marcel Dekker, 2004.

[5] Furht, Borko. *Encyclopedia of Multimedia.* 2nd. Springer, 2008. 222.

[6] Reza, Andres, dir. *Kinect With Nightshot.* Youtube, 2010. Film. 17 April 2013. <http://www.youtube.com/watch?v=nvvQJxgykcU>.

[7] Hecht, Jeff. *Lasers Bring Gesture Recognition to the Home.* 2011. Photograph. Laser Focus WorldWeb. 18 April 2013. <http://www.laserfocusworld.com/articles/2011/01/lasers-bring-gesture-recognition-to-the-home.html>.

[8] *Using the DLP Pico 2.0 Kit for Structured Light Applications.* 2011. Photograph. Texas Instruments. Web. 18 April 2013. <http://www.ti.com/lit/an/dlpa021a/dlpa021a.pdf>.

[9] *Structured Light vs. Microsoft Kinect.* 2011. Photograph. Hackengineer, Houston. Web. 18 April 2013. <http://www.hackengineer.com/structured-light-vs-microsoft-kinect/>.

[10] Dainty, C. *Laser Speckle and Related Phenomena.* Sprinter Verlag, 1984.

[11] Gupta, Mohit, Amit Agrawal, Ashok Veeraraghavan, and Srinivasa Narasimhan. *Measuring Shape in the Presence of Inter-Reflections, Sub-Surface Scattering and Defocus*. 2011.

[12] *Structured Light for 3D Scanning*. Brown University, 2007.

[13] Unknown. Photograph. Web. 25 April 2013. <http://robotstore.cc/images/detailed/0/tmp_drOUz4.jpg>.

[14] Unknown. Photograph. Web. 25 April 2013. <http://www.linuxfordevices.com/images/stories/beagleboard_xm_schematics.jpg>.

[15] Unknown. Photograph. Web. 26 April 2013. <http://antipastohw.pbworks.com/f/1343068375/BeagleJuice2 Front.jpg>.

[16] Unknown. Photograph. Web. 26 April 2013. <http://www.mouser.com/images/leopardimaging/lrg/LI-LBCMVGA.jpg>.

[17] Unknown. Photograph. Web. 28 April 2013. <http://www.rose-bopla.com/Prod_Pgs/Basic_Boxes/Prod_Polycarb_33.htm>.

[18] Caicedo, Jorge. "How to Install OpenCV on a BeagleBoard-xM with Angstrom OS." *Jorge Engineer*. 21 Jan 2012. Web. 29 April 2013. <http://installangstromonbeagleboarxm.blogspot.com/2012/01/how-to-install-opencv-on-beagleboard-xm.html>.

[19] Weaver, Trey. "Installing Angstrom on the BeagleBoard-xM." 31 Oct 2010. Web. 29 April 2013. <http://treyweaver.blogspot.com/2010/10/installing-angstrom-on-beagleboard-xm.html>.

[20] "Kinect-Like-3D Camera." *Hackengineer*. 7 Feb 2012. Web. 4 May 2013. <http://www.hackengineer.com/3dcam/>.