5-1-2016

FILLING A GAP IN THE MARKET: AN INDUSTRIAL RGB-D CAMERA

Hussain Mucklai

FILLING A GAP IN THE MARKET: AN INDUSTRIAL RGB-D CAMERA

by

HUSSAIN ALI MUCKLAI

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2016

ACKNOWLEDGMENTS

The success of this project belongs to my entire team, and I would be remiss not to acknowledge them first: Mary Kingsbury, Peter Menh, Isaac Ramirez and Nathan Smith. Our faculty mentor, Dr. Christopher McMurrough, also deserves as much of the credit, if not more, for his pivotal help and guidance when the team was at a loss for solutions to the challenges faced. The Department of Computer Science and the College of Engineering must also be acknowledged for funding the project and incorporating the Senior Design experience into their curriculum.

On a more personal note, I would like to thank my family who have made my journey here possible, my friends who have been so supportive of me, and my professors for their dedication to furthering my education.

Finally, all praise is due to God, the most gracious, the most merciful.

May 04, 2016

ABSTRACT

FILLING A GAP IN THE MARKET:

AN INDUSTRIAL RGB-D

CAMERA

Hussain Ali Mucklai, B.S. Software Engineering

The University of Texas at Arlington, 2016

Faculty Mentor: Christopher McMurrough

Computer vision has seen dramatic advances over the past few years, yet most industrial 3D-color cameras still cost many thousands of dollars, and there remains a gap in the market for a mid-range priced product (~$500). This work presents the research done in the design and development of such a device. The industry standards which must be met in order to succeed are in the realm of hardware and software. We show that by taking advantage of inexpensive, mass-produced hardware, as well as 3D printing, hardware costs can be cut down. By leveraging the open-source community to its utmost limit, software costs can be eliminated entirely.

We therefore prove that a functioning 3D-color camera can be produced which is able to withstand harsh environments and comply with most industry-wide standards at a reasonable cost.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

RGB-D cameras (also referred to as 3D-Color cameras) have soared in popularity over the past few years. These cameras provide images with color and depth information for each pixel in the image. This is done by combining information from a standard color camera and one or more infrared cameras paired with an infrared projector.

The importance of RGB-D cameras cannot be understated. Although there are many consumer applications for them, the most notable thus far being gaming [5], they have a far greater purpose to serve in the field of robotics. The added feature RBG-D cameras provide is the depth information they output (the “D” in RGB-D stands for “Depth”). This information is pivotal for any complex robotics project. RGB-D cameras provide much more useful sensory data than any ordinary color camera could. For robots that may be mobile or need to handle dynamic information, depth information of its surroundings is extremely useful. The depth data lets the robot know how close objects in its vicinity are, and allows the robot to act accordingly. A simple example would be a robot tasked with navigating a maze. A robot equipped with an RGB-D camera could much more easily determine the distance between itself and the maze walls and would be more successful in making movements based on its surroundings.

Sample output produced by an RGB-D camera is shown in Figure 1.1. This is the result of placing an RGB-D camera directly in front of the person shown in the figure. It can be seen that the output is not the normal, two-dimensional image produced by standard color cameras. Rather, a three-dimensional model is produced, which can be rotated to illustrate the depth information provided by the camera. Despite the fact that the camera was placed directly in front of the person, we can rotate the model (as shown in the figure) to obtain more detailed facial features. For example, the images in the figure clearly show the man's nose protruding out, his eyes sunk in, etc.

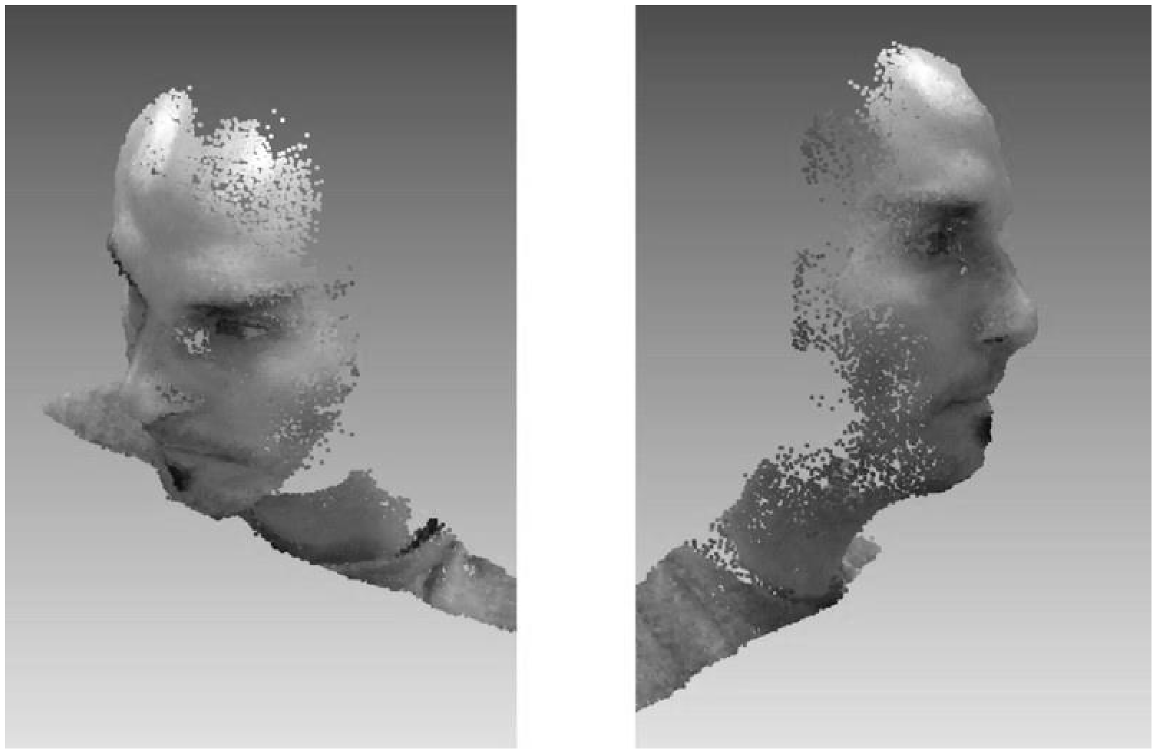

Figure 1.1: Sample RGB-D Camera Output [6]

This simple example highlights the utility of depth information. Despite this incredible advantage RGB-D cameras provide, there still remains a gap in the market for an industrial camera at a lower price point. The work done for this project designs and develops a product to fill this gap. The next chapter will expand upon this gap, and subsequent chapters will walk through the design and development process. Lastly, the final chapter will discuss the outcome of our endeavors.

CHAPTER 2

THE INSPIRATION

2.1 The Market

As mentioned in the preceding chapter, there remains a gap in the market for an industrial RGB-D camera at a lower price point. This section will discuss this market in greater detail and identify the gap which this product aims to fill.

Figure 2.1: The Microsoft Kinect [7]

When shopping for an RGB-D camera, one finds products that can be categorized as either reasonably cheap or outlandishly expensive, and nothing in between. Furthermore, one finds that the cameras on the cheaper side are only suitable for consumer applications, and would not adapt well to more industrial settings. Figure 2.1 shows a Microsoft Kinect, which belongs to the former of the two categories, and Figure 2.2 shows a Sick RANGER-C40412, a member of the latter.

The Kinect was a revolutionary innovation in the RGB-D world. At a retail price of $150, it expanded the use of RGB-D cameras to lower-budget projects. Indeed, at the time of its launch, the Kinect was “an order of magnitude cheaper than similar sensors that had existed before it” [5].

Figure 2.2: The Sick RANGER-C40412 [10]

However, despite this attractive price point, the Kinect (and other products of its kind launched subsequently) still posed many inconveniences to the industrial world. These consumer RGB-D cameras were not as robust as required for use in industrial applications, nor did they meet industrial standards in terms of the cables used to power the devices and get data from them. Furthermore, there were many ways for these RGB-D cameras to format their output. Some manufacturers opted for one open-source format, some opted for another, while some decided to develop their own proprietary format altogether. This made it very risky for industrial applications to use these cameras, as their projects would have to be tied to a specific camera and its format, and changing cameras would mean a lot of rework would have to be done.

Engineers in the above situation had little choice but to look at the other end of the RGB-D market spectrum. At this end, they found products such as the Sick RANGER-

C40412, with prices ranging from as low as $5,000, going all the way up to $15,000. The RANGER-C40412, shown in the figure, costs approximately $9,000 [12]. This is a massive elevation in price compared to cameras of the Kinect's ilk. These products may well have the features to justify their astronomical prices, but many industrial projects neither have use of these extended features nor the budget for them. Ideally, these projects call for a product such as the Kinect, but adapted to make it more suitable for the industrial stage. Such a product would only cost a few hundred dollars more than the Kinect, but meet all the needs of the industry at only a fraction of the cost of the more expensive models. This ideal product represents the industrial RGB-D camera that is the focus of this paper.

2.2 The Sponsor

The need for the market-gap-filling camera described in the previous subsection was brought to our attention by Wynright Corporation, a technology solutions firm dealing heavily in robotic systems.

Our faculty mentor, Dr. McMurrough, is a former employee of Wynright, and through his former colleagues still at the company learned about a challenge they were facing at work. The engineers at Wynright were developing a sophisticated autonomous machine to haul shipping containers at a dock and move them to their respective locations based on markings on the containers themselves. Such a machine would require an industrial RGB-D camera in order to identify the shipping containers correctly and be able to accurately move its mechanical arms to lift the containers safely. Dr. McMurrough learned that the project currently made use of the Microsoft Kinect as the system's camera in a suboptimal arrangement. As mentioned previously, this is in no way an ideal solution for this situation, and the Wynright engineers vented their frustrations to our faculty mentor. It is through this exchange that the idea for our project was born.

CHAPTER 3

THE VISION

Now that the need for such a camera has been established, we can move on to specifying exactly what we are trying to build and how we are going to build it.

The aim of this project is to build an industrial grade, RGB-D camera priced at $500 or less. There are many and more strategies which could be used to accomplish this goal. The specific method we opted for involved selecting suitable commercial-off-the-shelf products to piece together to achieve the desired result. This strategy was chosen because of a few key advantages it offered. The development time was naturally trimmed by reducing the number of components that needed to be constructed by the team from scratch. Furthermore, given the limited experience of the team in the 3D vision field, using completely developed and tested components reduced the possibility of introducing defects in the product. Lastly, many of the software components needed for our project were available at no cost to us as open-source projects.

Our plan to build the finished product we desired can be outlined in a few simple steps. We would first need to select a pre-existing RGB-D camera from those available on the market. We would then work on streaming data from this camera and converting it into an industry-standard format. To perform this processing, we would need to purchase a single-board computer (SBC). Once we worked this out, we would configure the SBC to work as a server and send out the formatted image data to requesting clients. Successfully accomplishing this task would represent completing the majority of the development work.

Most industrial devices receive power and data through a single Ethernet cable. To comply with this standard, we would connect our SBC to a Power over Ethernet (PoE) Injector. Finally, we would need to encase our components into a single housing unit to give our product a unified appearance, but more importantly, to make our camera more robust. Once this step was complete, the camera itself would be finished. As a last act, we would need to write software for the client side to know how to interact with our camera and request information from it.

The above paragraph glosses over numerous details to summarize our development plan as succinctly as possible. As the Project Owner, I had a responsibility to be involved in all aspects of the venture; however, as the Software Lead (and a Software Engineering student), I was more heavily invested in the software side of the product than the hardware. Therefore, the details of our software implementation will be discussed in detail in the next chapter. The remainder of this chapter will introduce the components we will use in the implementation, starting with the hardware and then moving on to the software.

3.1 The Hardware

Because the software for our product will either be leveraged from the open-source community (for free) or developed internally by the team, the project's cost (and thus, price) will be entirely determined by the hardware we use. Table 3.1 summarizes the costs of the hardware components discussed in this subsection. It can be deduced from the table that the total cost of all hardware used in the project is $275.43, which is well under the targeted price point of $500. Additionally, mass producing the camera would push costs down further due to the added benefit of bulk purchasing and the bargaining power that comes with it.

Table 3.1: The Hardware Cost Summary

Component	Model Details	Price ($)
Commercial RGB-D Camera	Intel RealSense R200	99.00
Single-Board Computer	MinnowBoard Turbot	139.00
Power over Ethernet	TP-LINK Gigabit PoE Injector Adapter	17.43
Casing	3D Printed	20.00

3.1.1 The Camera

The commercial RGB-D camera selected for the project was the Intel RealSense R200. This may appear to be an odd decision based on the heavy praise of the Microsoft Kinect earlier in the piece, and the recent release of its successor, the Kinect 2. However, we chose the R200 (shown in Figure 3.1) because of several advantages it possessed over the Kinect series.

Figure 3.1: The Intel RealSense R200 [3]

The most apparent advantage of the R200 camera is its compact size, a feat which warrants its inclusion as the in-built camera in some tablets and 2-in-1 computers starting in late 2015 [1]. Furthermore, unlike the Kinect which was built primarily to identify humans and track their movements, one of the Intel camera's main functional areas is to build "real-time digital representation of the 3D Scene observed by the camera" [1]. This is the functionality required of our product, and our internal tests confirmed that the R200 was more accurate at doing this than the Kinect.

3.1.2 The Single-Board Computer

There were three main selection criteria for the single-board computer (SBC) to be used in our product. The first two requirements are to use Intel's chipset in its architecture and have at least one USB 3.0 port. These requirements are for the SBC to be compatible with the Intel R200. The third requirement is for the SBC to be powerful enough to be able to handle converting and streaming images in real time.

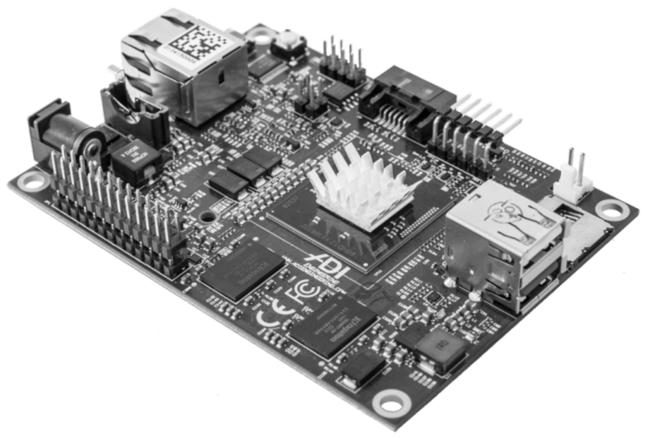

Figure 3.2: The MinnowBoard Turbot [8]

Our three criteria eliminated many of the lower-tier SBCs. The one we decided on, which satisfied all of our needs at a reasonable price point, was the MinnowBoard Turbot (shown in Figure 3.2). Its specification checked all the requirements, and as the most economical choice to do so, it was the primary contender.

3.1.3 The Power over Ethernet

As industry standard dictates, our device needs to both be powered by, and transmit data over, an Ethernet cable. The MinnowBoard, like many SBCs, is able to transmit data over an Ethernet cable, but requires a dedicated power input. To work around this, we planned to connect the MinnowBoard to a Power over Ethernet (PoE) Injector, specifically, the TP-Link PoE Injector shown in Figure 3.3. On one side of the Injector (the left side of the figure) we connect the data and the power cables directly from the MinnowBoard. The Ethernet port on the other side of the Injector will be exposed through the casing. The user of our device can connect their data cable directly into our Injector. The PoE device will then split the power and the data signals and output them separately to the MinnowBoard, and this setup effectively allows our product to receive power from, and transmit data through, a singular Ethernet cable.

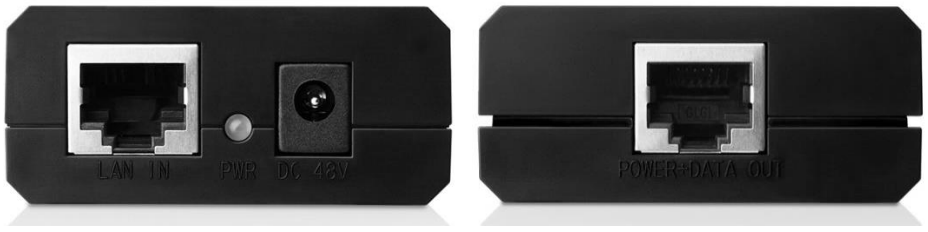

Figure 3.3: The TP-Link PoE Injector [11]

3.1.4 The Casing

The casing for our device would need to be custom made to meet our exact specifications. We require mounts for the R200, the MinnowBoard and the PoE Injector. We would also like to add our own design and branding to the case.

Requesting a tailor-made case of the type we need may have cost time and money in the past, but due to the advances in 3D printing technology, and the availability of this technology on the University campus, we were able to 3D print a case to our exact specifications at low cost.

3.2 The Software

This subsection will introduce the third-party software components that are essential to our product's operations. As mentioned previously, these components are completely open-source and are available for use per their respective licenses.

3.2.1 The Camera Drivers

To read from the Intel RealSense R200 camera itself, we require its drivers. Unfortunately, upon release, Intel had only developed and made public drivers for the Windows platform. This was a problem for the team, as our MinnowBoard was running Linux, a free operating system, and switching to Windows would add an unnecessary licensing fee to our product.

Fortunately, the open-source community (in all its vastness) was working on a Linux version of the drivers [4]. The R200 was a highly anticipated product which many developers wanted to test out, and a good number of developers happen to use Linux. Through this stroke of luck we were able to move past this hurdle by including this open-source component into our project.

3.2.2 The Industry Standard Format

As mentioned previously, the RGB-D camera's output would need to be converted to a format that was industry standard. Of the formats to choose from, the Point Cloud Library [9] format was the most preferred. In this format, RGB-D images would be presented as a collection of points in a 3D space, (also known as a cloud). Each point in the cloud would have RGB values for its color, and XYZ values for its position.

3.2.3 The Server

The single-board computer (SBC) would need to be able to stream point clouds in real time to clients requesting the data. Using the Transmission Control Protocol (TCP) to do this made sense. TCP is the standard by which much of the communication over the Internet takes place and most (if not all) machines in an industrial setting would be able to use this protocol.

Implementing TCP in our application would be a very tedious and error-prone exercise. There are, of course, open-source library applications that have already implemented networking features (including TCP) and have done so much more efficiently than we have done. The specific open-source library we chose was ZeroMQ [2]. We chose this library because of its wealth of documentation, its superior performance, and the functions it implemented which were well suited to our specific needs.

CHAPTER 4

THE IMPLEMENTATION

As the Software Lead for this project, my main role was to ensure the functionality of the product's software components. This penultimate chapter will detail the work done in the implementation of these software components.

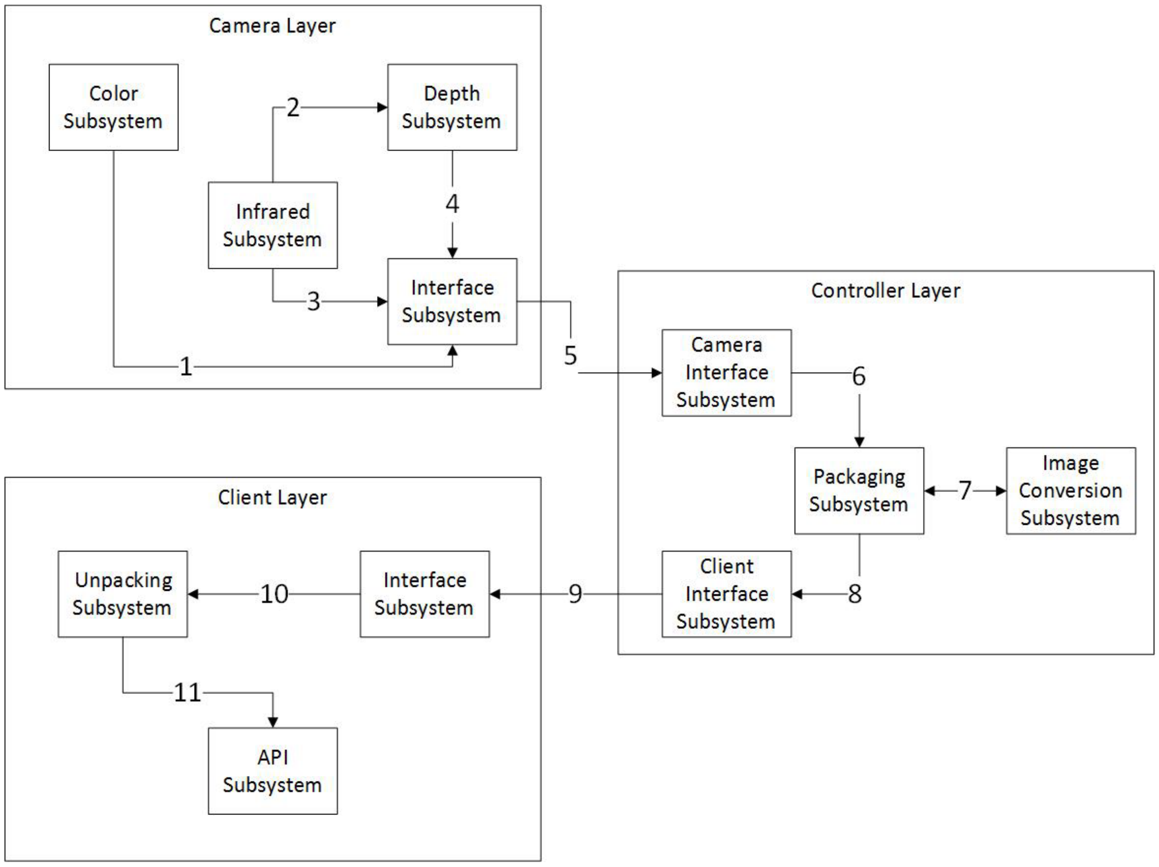

Figure 4.1: The Software Architecture

Figure 4.1 shows the software architecture of our RGB-D camera. The architecture depicts the software components employed in delivering the camera's functionality, as well as the data flow between these components.

The Camera Layer represents the commercial RGB-D camera used in our device (the Intel RealSense R200). The components of this layer came pre-installed on the camera and were not implemented at all by the team. The Controller Layer represents the single-board computer (SBC) we have used (the MinnowBoard Turbot), and the Client Layer represents a client machine attempting to receive image streams from our device. The components of these two layers were the primary focus of our implementation efforts. The following subsections will walk through the main challenges faced during this process.

4.1 The Capture

The first task we had was to read data from the R200. The Camera Interface Subsystem would be responsible for this, and as mentioned in section 3.2.1, the open-source community had an answer for this challenge. The problem, however, was that we were attempting to use the driver while it was actively in its first stage of development. As a result, the team had its first taste of actively engaging in the development of an open-source project. This was a valuable experience for most of us as young computer scientists. We were able to work through all the problems we encountered in the driver by corresponding with the principal contributors of the project.

Once we had cleared this hurdle, we were able to successfully read image streams from the R200. The next step was to convert the streams to an acceptable format.

4.2 The Conversion

We had elected the Point Cloud Library (PCL) [9] format to be the standard for the camera's output stream. The Camera Interface Subsystem was able to read data from the R200 in the form of two streams, one for color images and one for depth maps. A capture of these streams can be seen in Figure 4.2. From our discussion of point clouds in section

3.2.2, we knew that we needed to combine these two streams to generate a stream of point clouds.

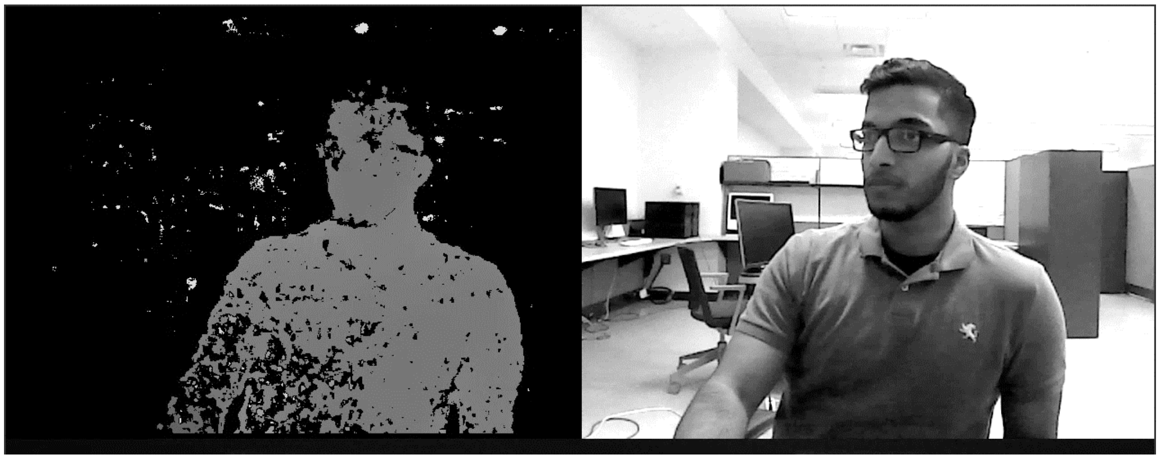

Figure 4.2: The Depth and Color Streams

This conversion consumed the bulk of the processing done by the system. To generate the point cloud, our algorithm would parse through each pixel in the color image and obtain its RGB value, obtain the depth value from the corresponding pixel in the depth map, and use this value and the position of the pixel to calculate the XYZ value for the specific pixel. Using these RGB and XYZ values, we are able to create a point and add it to the point cloud. A capture of a point cloud is shown in Figure 4.3. The figure shows the point cloud rotated to view the cloud from the left and the right hand sides.

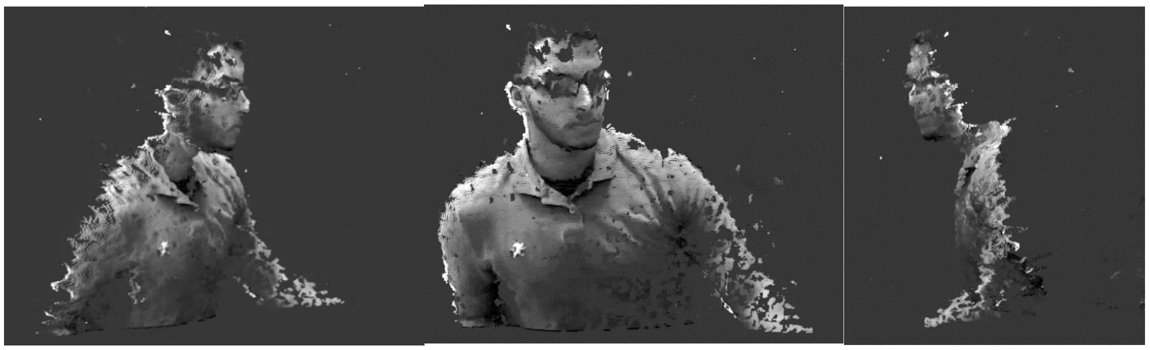

Figure 4.3: The Point Cloud

Our camera would output images which are 480 pixels high and 640 pixels wide. This means that the above algorithm would have to generate 307,200 points to create just one point cloud. Attempting to output at 15-20 frames per second (FPS) would mean our single-board computer (SBC) would have to churn out 4,608,000-6,144,000 points per second. To reliably reach this performance target, the team needed to implement the above algorithm as efficiently as possible. The technique that yielded the best results was to utilize multithreading in the algorithm. Using both of the cores on the MinnowBoard allowed the generation of points, and therefore frames, twice as fast.

4.3 The Transmission

One we were able to convert our R200 images into point clouds, we were ready to transmit these clouds to the client. The transmission involves four subsystems in total, namely the Packaging and Client Interface Subsystems in the Controller Layer and the Interface and Unpacking Subsystems in the Client Layer. These four subsystems made heavy use of the ZeroMQ library introduced in section 3.2.3. However, the point clouds were large and complex data structures, and ideally the data to be transmitted would be minimized to reduce any latency experienced on the client's side in receiving the data. We accomplished this by realizing that sending the entire point cloud was inefficient, as the cloud itself contained metadata which was either constant between point clouds or irrelevant for our purposes. We therefore only transmitted the essential data, which was in effect the list of all the points used to generate the cloud. We collected this data on the client side and added the metadata fields to produce an identical point cloud to the one captured by the server.

Using the ZeroMQ library in our implementation also gave our product the added feature of being able to stream its output to multiple clients simultaneously in a highly scalable fashion. This is in contrast to commercial cameras like the R200 and Kinect on their own, which only permit one client to read from them at a time. This allows our camera to be shared by numerous applications in a networked environment, which is not an uncommon occurrence in an industry setting.

4.4 The API

Our final task in the implementation of the device's software was to develop an API (Application Programming Interface) to allow clients to be able to communicate with our camera. The API would allow the developers to read from the camera's stream with just one or two simple lines of code.

We decided to implement two API calls for clients to use. One was to capture a single point cloud from the camera's stream. The other was to start and stop reading the entire stream of point clouds from the camera. This distinction in functionality is very important from a resources standpoint. Streaming point clouds consumes a lot of memory due to the size of the data structure and the fast pace of arrival. It would be very easy to flood the client's network buffer by continuously streaming the clouds and only delivering a cloud to the client application when one is requested. However, this would be extremely inefficient, as many thousands of bytes are being thrown away unused by the client every second. The implementation the team opted for, which only streamed images when requested, is much more apt for the specific situation, and uses far fewer of the client's resources.

CHAPTER 5

CONCLUSION

In conclusion, we developed a product which satisfied all of the requirements it had set out to. Our RGB-D camera is able to stream point clouds in real time at an acceptable frame rate. This would allow for immediate decisions to be made by real time systems, such as robots, based on the data provided by the camera. The camera has also been built well enough to be rated IP 52, which means the device is completely protected against contact and partially resistant to dust, as well as being unharmed by water drops. Our device would fit well into an industrial environment due in part to this IP rating, and in part to being powered by an Ethernet cable, which is also used for data transmission. Lastly, our camera was built well within its targeted budget, which is an important factor in relation to its commercial success.

The practical applications of our RGB-D camera stretch far beyond the need of our sponsors. Our product can assist any application which could make use of reliable depth and color data of its surroundings in a harsh environment. Examples of these applications range from large-scale welding to quality control in industrial ovens. Furthermore, as robotic technology advances and the world moves further towards automation, these applications will only increase in number. The device developed by the team would allow projects which may not have a large budget to make use of an industrial-grade RGB-D camera. Even projects which had afforded a more expensive camera before may make use of our product to alleviate their own production costs.

APPENDIX A

GLOSSARY

Application Program Interface (API): A set of routines, protocols, and tools that specify how software components should interact.

Algorithm: A process or set of rules to be followed in calculations or other problem-solving operations.

Client: A piece of computer hardware or software that accesses a service made available by a server.

Core: A core is the processing unit which receives instructions and performs calculations, or actions, based on those instructions.

Driver: A program that controls the operation of a device.

Ethernet: A system for connecting a number of computer systems to form a local area network, with protocols to control the passing of information and to avoid simultaneous transmission by two or more systems.

Ingress Protection (IP) rating: An international standard used to define levels of sealing effectiveness of electrical enclosures against intrusion from foreign bodies and moisture.

Multithreading: A technique by which a single set of code can be used by several processors at different stages of execution.

Server: A computer or computer program that manages access to a centralized resource or service in a network.

Single-Board Computer (SBC): A computer which is a complete computer in which a single circuit board comprises memory, input/output, a microprocessor and all other necessary features.

REFERENCES

[1] C. Culbertson, "Introducing the Intel® RealSense™ R200 Camera (world facing)", Intel Developer Zone, 2015. [Online]. Available: https://software.intel.com/en-us/articles/realsense-r200-camera. [Accessed: 28- Apr- 2016].

[2] "Distributed Messaging - zeromq", Zeromq.org, 2016. [Online]. Available: http://zeromq.org/. [Accessed: 28- Apr- 2016].

[3] Intel, Intel R200 Camera. 2016.

[4] "IntelRealSense/librealsense", GitHub, 2016. [Online]. Available: https://github.com/IntelRealSense/librealsense. [Accessed: 28- Apr- 2016].

[5] K. Litomisky, "Consumer RGB - D Cameras and their Applications", University of California, Riverside, 2016. [Online]. Available: http://alumni.cs.ucr.edu/~klitomis/files/RGBD-intro.pdf. [Accessed: 28- Apr- 2016].

[6] MacRumors, 3D Depth Mapping. 2016.

[7] Microsoft, Kinect Sensor. 2016.

[8] MinnowBoard, MinnowBoard Turbot. 2016.

[9] "PCL - Point Cloud Library (PCL)", Pointclouds.org, 2016. [Online]. Available: http://pointclouds.org/. [Accessed: 28- Apr- 2016].

[10] Sick, Sick RANGER-C40412. 2016.

[11] TP-Link, TP-Link PoE Injector. 2016.

[12] Vision-supplies.com, "Sick RANGER-C40412 (1014218) | Vision-Supplies.com - Vision, Sensors & Automation Components Distributor", 2016. [Online]. Available: https://www.vision-supplies.com/p/87467/sick-ranger-c40412. [Accessed: 28- Apr- 2016].

BIOGRAPHICAL INFORMATION

Hussain was born in Karachi, Pakistan in December of 1993. He completed his Cambridge International A Levels at Simba International School in Ndola, Zambia in 2012, after which he travelled to Arlington, Texas to pursue his Honors Bachelor of Science degree in Software Engineering. Upon graduation, Hussain plans to move to Boston, Massachusetts to begin his first full-time position as a Software Engineer. From his year as a Research Assistant in the Information Security Lab at the University of Texas at Arlington, Hussain developed an interest in research and academia, and hopes to one day return to school to earn a postgraduate degree.