# 3D Reconstruction in Real Time using Stereo Cameras (December 2019)

Team 4DR — Kyle Bautista, *Team Member*, Alan Huynh, *Team Member*, Brandon Luong, *Team Member*, Jesse Angelo Pangilinan, *Team Member*, and Professor Glenn Healey, *Academic Advisor*

***Abstract*— The focus for the project by the end of both quarters is to create a stereo camera setup, that is able to create a 3D Reconstruction of an object or a person in real time. We would also want the apparatus to be scalable, so that it is easy to attach another camera, or use more computing power to receive results with higher resolution and better accuracy. The goals for Fall quarter was to have a stereo camera setup that can at least create a 3D point cloud from a set of two images rather than running at real time. The Fall quarter goal was almost reached; the cameras can currently run a lightweight program that creates a sparse 3D point cloud. 5The outcomes of the Fall quarter were a result of inexperience in working in a new topic and using inefficient testing methods. The goal was achieved by finding the internal and external parameters, which were then used to create a rectified image with the newly found epipolar geometry. Through the rectified images found, the program attempts to find corresponding points between the images, and finally triangulates its position. A side goal of the project was to also have a hardware-based control system, using individual buttons, making it easy to control the cameras with just a few buttons. Buttons have been implemented fully and can be easily remapped to control different functions of this system.**

## I. Introduction

THE current state of the project is to create an accurate 3D point cloud from two cameras, other complications must be solved beforehand. The goals and methods are in the following three sentences. First, the cameras need to be controlled so that they can take pictures at the same time. Secondly, these cameras need the ability to detect the corners of a checkerboard pattern which will give some real-world references and it will help find the internal and external camera parameters. Finally, these parameters are used to rectify the image, and point correspondences are found and triangulated, leading to a 3D point cloud. From the background, all of the solutions are considered basic photogrammetry and computer vision, which has been studied for quite a while (Hartley). The focus is to apply the linear algebra studied in these textbooks and papers, into a simple and scalable project that will create 3D reconstructions in real time. Epipolar geometry, linear algebra, and photogrammetry have all been well studied for a while now.

## II. Main Body

### A. Material Used

The main materials used in the project were:

1. *Stereo Cameras*

We used two webcams which were the Logitech C270. Webcam were the most affordable and best fit our project as it provided a decent amount of pixel density. Each camera was 720p which allows images to come out high definition but also minimize processing time that a higher resolution might increase.

*2. Raspberry Pi*

The Raspberry Pi has the ability to run our software standalone and also used for scalability and usability. In terms of scalability, we can expand on our hardware with additional sensors (i.e. Ultrasonic) to make our system more redundant and robust. In addition, the Pi allows us to appeal to non-technical users where we have added a button interface that simply executes the program when pressing certain buttons.

*3. Apparatus: Tripod and Polyethylene Foam*

We needed a rig to hold the two cameras. Therefore, we implemented a tripod to be able to get an adjustable height that we need to achieve to get a picture at the correct levels. Furthermore, we added a polyethylene shaped tube of around five feet length and six inch diameter to hold the two cameras. The tube is aligned perpendicular to the tripod to get the width and distance we need between the cameras. See Figure 1 for a visual representation.

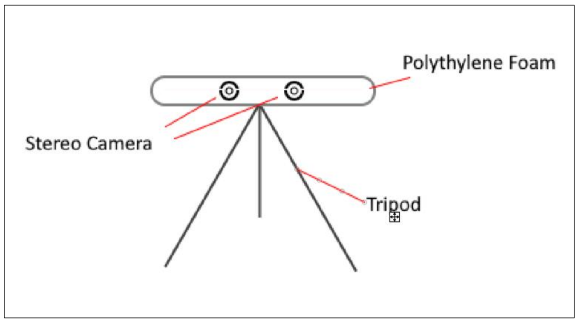

*Figure 1. Diagram of Stereo Camera Rig*

Kyle Bautista, Alan Huynh, Brandon Luong, Jesse Angelo Pangilinan are the group members of this work and are students with the Department of Electrical Engineering and Computer Science, University of California, Irvine. (email respectively: kvbautis@uci.edu, alanh7@uci.edu, jpangil1@uci.edu, btluong1@uci.edu)

Professor Glenn Healey is the academic advisor for this work and is also with the Department of Electrical Engineering and Computer Science, University of California, Irvine. (email: ghealey@uci.edu)

## B. *High Level Hardware and Software System*

### *1. Software - Edge Detection and Checkerboard Detection*

The software analyzes the images inputted, and finds points where the change in intensity values is the highest from all directions. By finding all the high intensity points, the software will initially find all the edges within the scene. Later these points are all tested against each other to find the points the are all linearly spaced from each other, which creates the checkerboard.

### *2. Software - Internal and External Camera Parameters*

For each camera, the software takes all the locations found in the checkerboard pattern, and maps them to real world locations. The software then attempts to find the closest matrix, that creates the linear transform between real world locations and pixel locations found in the checkerboard pattern. Using up to 50 samples of this checkerboard pattern in different positions ends up giving the best results to find the internal parameters, the focal length and pixel/meter ratio. Then, the camera is then fed pictures that were taken at the same time, and both have the checkerboard pattern showing. These pixel locations correspond to each other, and similar to finding the internal parameters, the matrix between the two pixel locations are found. Using another 50 samples of these image pairs gives the final external camera parameters.

### *3. Software - Epipolar Geometry*

These internal and external camera parameters can help determine the epipolar geometry of the cameras. The basic idea behind epipolar geometry is that there is a line within the first camera, where all the pixels correspond to another line in the second camera. There is also image rectification which makes every line in this, have a slope 0 and parallel to the horizontal axis. This helps with the correspondence problem speed up from being a N^4 problem to an N^3 problem.

### *4. Hardware - Cameras and Button Control*

We are utilizing a Raspberry Pi 4 for our main processing unit to execute the 3D reconstruction software and the Pi might transition as an intermediary unit to transfer images to a computer with more processing power. The stereo cameras are connected to the Raspberry Pi, button system, and a touchscreen LCD. The button system connected to Raspberry Pi will be used for controlling the integral functions of our software, thus creating an easy user interface that non-technical users can even use. The touchscreen display will be used to preview any picture data or can be used to create a graphical user interface in conjunction with the buttons for any more specific functions.

## C. *Methodology*

The method that we used to stay on track include having a timeline (i.e. Gantt Chart), version control, budgeting, weekly meetings, and reporting. The Gantt Chart alongside the weekly meeting with the teaching assistants gave us a gist of where progress on a wider scope but put pressure to stay on track. As seen by the figure below, the figure is a partial view of our Gantt Chart around week eight and nine where green is on tracked, yellow is planned, and orange is delayed.

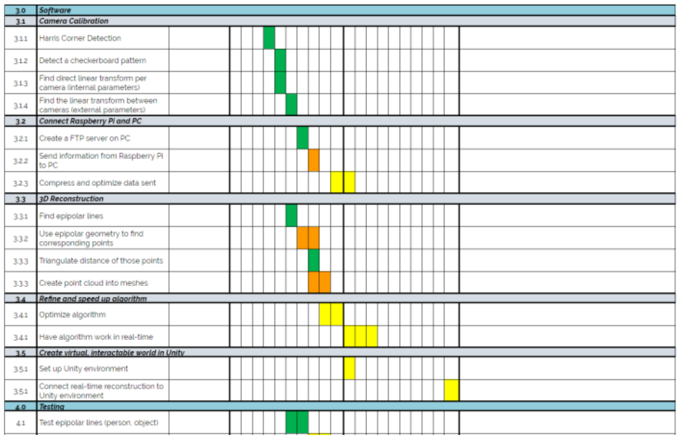

*Figure 2. Gantt Chart*

Having a distributed version control for the software development contributed to keeping us organized. We had a system where every person forked from a master branch and made pull requests when they wanted to make changes. The system allowed us to have a version that was working completely and separated each team members work so that they do not interfere with each other. The figure below provided by the University of Washington demonstrates a visual representation of the version control.

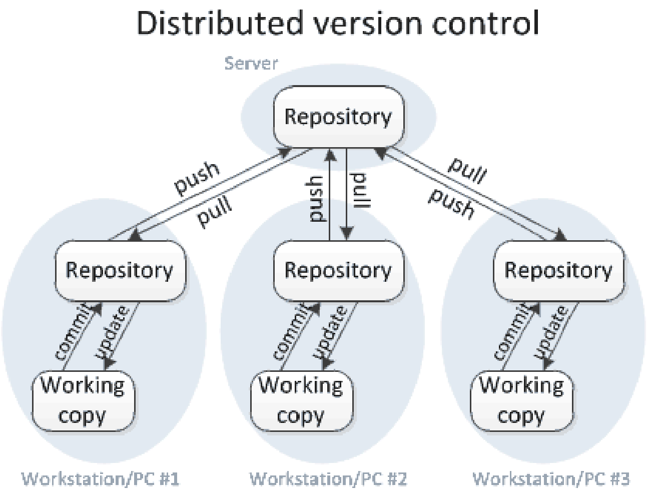

*Figure 3. Version Control*

We also found that communicating amongst the team and having weekly meetings contributed to staying on track with the timeline. Our team had weekly meetings in person to establish action items that needed to be completed or work collaboratively on items that needed to be worked on in person. In addition to physical interaction, we also created a Discord server to communicate when all team members were not able to meet in person or needed assistance outside of our scheduled meeting. We also emailed our academic advisor when needed to get information on concepts we were not knowledgeable on.

### D. Results and Performance

The following are visual representation and results of our software described earlier in "high level hardware and software system." In terms of performance, the software heavily depends on whether the program is run on the Raspberry Pi or an external unit. On an external unit like a laptop, the software runs seamlessly and smoothly. On the other hand, when run on the Raspberry Pi, the software works properly but does not have the processing power as compared to a computer; therefore, when using the Pi, there are some delays in the process.

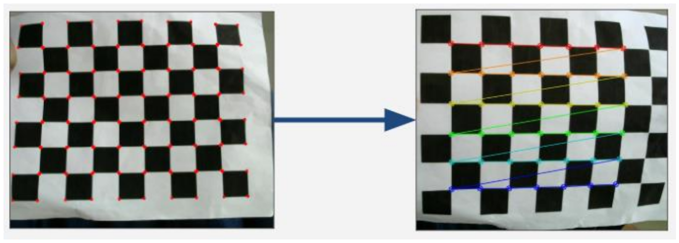
*Figure 4. Edge Detection and Checkerboard Detection*

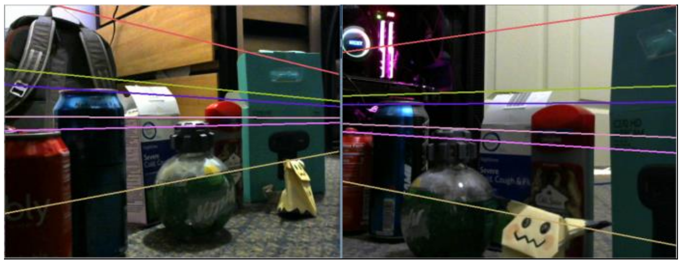
*Figure 5. Epipolar Geometry and Epipolar Lines*

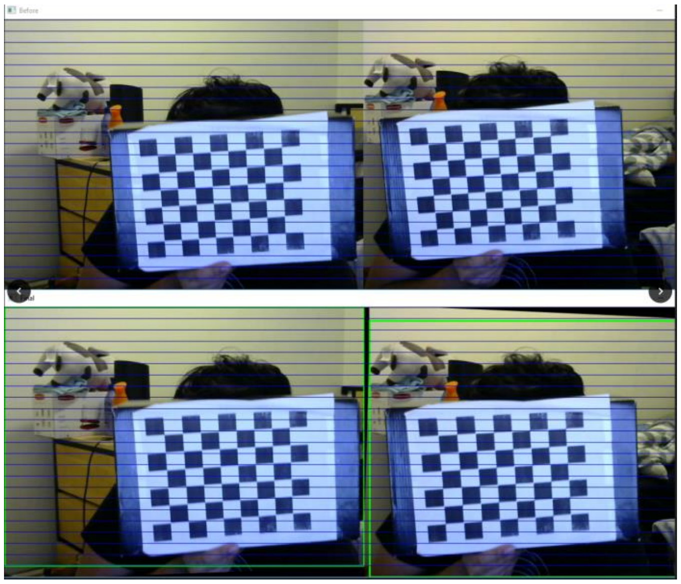
*Figure 6. Image Rectification*

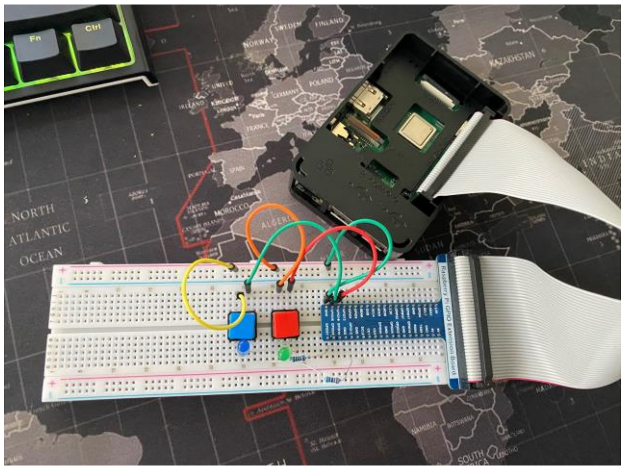
*Figure 7. Simple Button Control System*

## III. Summary

In terms of summarization of the project, we have achieved a sparse 3D point cloud, by using edge/checkerboard detection, finding internal/external parameters, and finally image rectification. The combination of OpenCV library and the linear algebra from previous studies, we were able to achieve all these different problems. We are able to now have a simple set of controls that help us also improve the internal and external parameters by having quick access buttons to have controls.

## IV. Conclusion

For the future, we are looking to improve the current system by moving the computing power of a real-time system from a Pi to a powerful computer with a GPU. We will also want to showcase this real-time model on software that can run such models, so we need to send this data to another computer anyways. We also don't want the computer we are running on to lose memory from real-time data, so we won't be able to save the data outputted by this device. Overall, we have some catching up to do, but we are ready to tackle more of the other problems coming next quarter. Some of these new problems we will need to solve is the ability to have the performance of the code be fast enough so that it can run at real-time. We also want scalability, so the code should be able to add more cameras and improve the quality of the final image.

## Appendix 1 – Technical Standards

The few standards we need to worry about when it comes to 3D reconstruction, is the data output for the point clouds and 3D models. There are many 3D model formats available, but we are most likely going to use neutral 3D file formats like STL or COLLADA. These are used so that they can be used to convert into other proprietary formats that are found in more proprietary codes. Most likely, we will be using COLLADA, because it is known to be compatible with Blender, which is the software we are looking to use to test these 3D models. For point clouds, we will most likely XYZ files since those are also testable in Blender. Other than these standards, the rest of the images will be in PNG format so that we have a lossless compression file format. It will keep the images we take and save for references and for recalculations at the highest quality available.

## APPENDIX 2 – CONSTRAINTS

There were 3 different challenges and constraints that this project has posed when trying to develop this system of cameras and programs. The first constraint of using a Raspberry Pi 4 run the code is that it is difficult to show the code running when on the go because it would be difficult to carry a monitor everywhere. To solve this problem, we decided to use separate button controls and use a smaller monitor that can be taken on the go. This has helped make this project be more mobile so that we can show our progress during labs. The second problem is that the computing power on the Raspberry Pi 4 will not be powerful enough to handle the real-time part of the 3D reconstruction. In order to have the code run in real-time, the code must be able to reconstruct an object in 1/x amount, where x is the frames per second that we want to achieve. 10 frames per second seems like a good medium between performance and quality, but the Raspberry Pi 4 lacks the GPU to handle lots of linear algebraic equations and processes. The most likely solution to this will be having the Raspberry Pi do image rectification, while a stronger GPU can handle point correspondences. This will remove the load from the Pi itself and move the constraint to how fast the data can be sent from the Pi to a computer with a GPU. The final constraint comes from the choice in cameras. As a group, we had to constantly balance between performance speed, cost of the hardware, and quality of the final 3D model. For each of these cases, we decided on the goldilocks choice, right in the middle. We decided on a cheap camera that gives HD quality at 720p and 30 frames per second. If we decided on a camera more expensive and higher quality, it would have required much more computing power to handle the many more pixels. If we decided on any lower quality of a camera, the quality of the final 3D model would be poor.

## APPENDIX 3 – HARDWARE AND SOFTWARE SECURITY ISSUES

The Raspberry Pi that is being utilized is currently not connected to the Internet. Therefore, the Pi is not susceptible to attacks that originate from the Internet. At first, we wanted to create a system where picture data from the stereo cameras are sent to a SQL server through the Raspberry Pi. Then, we transfer the pictures from the server to the main computer where the main computation for the point cloud and mesh creation are executed. However, we realized that having such a server leaves our data vulnerable to attacks. Taking that into account, we decided to try two solutions. One is to use a local server where we would do the transferring of files all locally to the main computer. The local server will mitigate the possibility of an online attack because only those connected to or on the local server has access. However, we would need to bring the main computer with us wherever we setup, bringing down any aspect of an easily portable rig. The second option is to save our data straight onto the Pi and create the point clouds and meshes there. Creating the point clouds and meshes on the Pi will be slower than doing those actions on a computer with a dedicated GPU, but bringing the Pi completely offline solves the problem of an online attack.

## ACKNOWLEDGMENT

1. The library that helps with coding cameras and computer vision is OpenCV.
2. This work was supported in part by the EECS 159 funding.
3. The equipment sources include but were not limited to Newark, Best Buy, and Lowes.
4. Kyle Bautista, Alan Huynh, Brandon Luong, Jesse Angelo Pangilinan are the group members of this work and are students with the Department of Electrical Engineering and Computer Science, University of California, Irvine. (email respectively: kvbautis@uci.edu, alanh7@uci.edu, jpangil1@uci.edu, btluong1@uci.edu)
5. Professor Glenn Healey is the academic advisor for this work and is also with the Department of Electrical Engineering and Computer Science, University of California, Irvine. (email: ghealey@uci.edu)

## REFERENCES

[1] Ma, Yi, et al. *An Invitation to 3-D Vision: from Images to Geometric Models*. Springer, 2001.
[2] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2017.
[3] Trucco, Emanuele & Verri, Alessandro. (1998). Introductory techniques for 3-D computer vision.
[4] Szeliski, Richard. *Computer Vision: Algorithms and Applications*. Springer, 2010.