STEREOSCOPIC IMAGE ANALYSIS

Team Members:

Christopher Dowlatram

Steven Calderon

Andres Santamaria

Thomas Whitaker

Sponsors:

Amy Larner Giroux, PhD

National Cemetery Administration

University of Central Florida

December 3rd, 2019

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The goal of this project, which was sponsored by the National Cemetery Administration, was to use computer vision and machine learning in conjunction with epipolar geometry to measure off distances in stereographic Civil War photographs of a St. Augustine burial site. Once these measurements were made, they were compared against the same gravesite in present-day to assist with identifying now-unlabeled headstones of soldiers from that era.

A web-based application was developed in React to accept images as input and allow users to input known and desired measurements. Machine learning on a trained dataset was used to estimate unknown required parameters required to calculate distances (camera focal length and angle of view). Once all parameters were obtained, the application was used to calculate and output the desired measurements to the user.

This project is significant as it will help bring closure to those soldiers whose tombstones are currently unlabeled. Not only that, it will bring closure to their families and descendants as well. It is also important as it properly preserves a piece of American history. Moreover, it will benefit the American Cemetery Administration as it will be a less invasive way to properly identify the fallen soldiers.

Beyond the immediate significance, this project could also be potentially extrapolated to be used on other distance measurement problems which in and of itself could be a great tool to anyone who would use it.

PROJECT OVERVIEW

Twenty men from the 7th Regiment of the New Hampshire Volunteers died in Saint Augustine, Florida during the Civil War. There are old photographs from this era in which some of the wooden headboards clearly show nine of these men's names. However, over time the headboards from these graves rotted away before they could be properly recorded. Since they were all located near a Native American burial ground, when these headboards were replaced with headstones, they were mistakenly labeled in 1905 as unknown Indian War soldiers.

The goal of this project was to use computer vision and machine learning to calculate the distances in the old Civil War photographs and be able to measure those off in the present-day to assist with accurately identifying the graves of the nine men so their names can be placed on their headstones.

The program was required to read in two stereoscopic input images and offer a user interface for placing points and lines to represent known and desired measurements. Since some of the old photographs did not have camera sensor parameters (e.g. focal length, angle of view, etc.), the program's algorithm could not rely on this. Moreover, while there are algorithms that perform depth estimation via heat maps, the program had to be able to do depth estimation in actual units (e.g. feet, meters, etc.).

19th Century Left Stereoscopic Photograph Depicting Civil War Soldiers' Marked Graves (Provided by Dr. Giroux)

In this left of a pair of stereoscopic photographs, the names of nine of the Union soldiers can clearly be read. These soldiers are marked in the next figure.

19th Century Left and Right Stereoscopic Photographs Depicting Civil War Soldiers' Marked Graves with Measurements (Provided by Dr. Giroux)

The nine numbered graves in the left stereographic photograph represent those soldiers whose names can still be read when the photo is enlarged (as seen in the previous figure). In the corresponding right stereographic photograph, the colored lines on the base of the pyramids in the background represent known measurements that are displayed on the right side of the figure. The desired measurements from the headboards in the foreground to the pyramids in the background are marked by the red lines going from the foreground to the background.

21st Century Photograph of Same Burial Site With Now Unmarked Graves (Provided by Dr. Giroux)

This photograph taken in 2018 is of the same scene depicted in the previous two figures. The three pyramids and obelisk in the background are still in the same position and have the same measurements as they had in the 1800s. When the tombstones in the foreground replaced the headboards, instead of being placed directly over the old ones, they were arranged to have a more uniform appearance. In addition, since the names of the soldiers were not properly recorded and because of the nearby Native American pyramids and obelisks, the tombstones were mistakenly labeled as unknown Indian War soldiers.

Motivation

Christopher Dowlatram

I am currently interning on a data science team at the Naval Air Warfare Center Training Systems Division (NAWCTSD), a group within the Naval Air Systems Command (NAVAIR). My work involves the ingestion, manipulation, visualization, and analysis of a lot of data. Sometimes when dealing with such big data, it becomes challenging if not impractical to try to sift through it all to extract any noticeable features or changes in one variable due to some other variable. When faced with such a dilemma, I find myself turning to machine learning.

Machine learning and its derivatives are something that has always piqued my interest in the field of computer science. The idea of writing programs that are trained to solve problems without having to explicitly hard code every aspect of what to do is something that fascinates me. While I have gotten to use it a few times from a data science perspective to train and test data for feature extraction and validation, I still find myself looking for other ways to dabble in it. This past spring semester I had the opportunity to take a robot vision class at UCF. This course gave a brief introduction about some programs and methodologies utilized in the field of robot vision to recognize and capture objects in images. One project involved utilizing the adaptive boosting (AdaBoost) machine learning algorithm for facial recognition. We were tasked to train a program to recognize the difference between a face and non-face then test it on an image containing numerous faces. We knew we were successful if our image recognized all the faces whilst at the same time did not falsely identify too many non-faces as faces.

The AdaBoost assignment was my first actual introduction to using machine learning to solve a visual problem. The final project in the class involved finding a peer-reviewed journal article introducing a novel method in computer vision. The paper I chose was called “Aperture Supervision for Monocular Depth Estimation” by Srinivasan et al [1]. To summarize, the author’s developed a way using machine learning and convolutional neural networks to estimate the depths of objects in images with only the single image as input (previous methods involved at least multiple inputs of the same image at varying angles).

These two assignments really stuck with me as unlike my previous experience with machine learning, it was easier to literally see the result. Because of them, when it came time to choose my senior design project and I saw there was a project involving the use

of computer vision which would give me more exposure to this field, I knew I had to choose it. Moreover, the specific application this project is on—using old photographs and computer vision to assist with retagging modern worn-down tombstones thereby giving closure to those soldiers and their families—is one I feel is important. Not only will this provide more accuracy to a historical site, but it is also an honorable thing to do. In addition, the findings from this project could be used in other similar applications to help others as well (e.g. retagging other previously worn-down graves, etc.).

Steven Calderon

My motivation for doing this project was primarily an interest in undertaking a challenging project using computer vision and machine learning. This project immediately went to the top of my list when it was presented. I have been interested in computer vision and machine learning for some time now and got the chance to take a course in robot vision last semester. Through the course, I was exposed to the basic concepts of computer vision and machine learning, and also got some practice in using those concepts with images and video. This project gives me a great opportunity to use what I learned and build upon it.

Besides the chance to improve my skills and knowledge, I was also drawn by the opportunity to help in reconstructing the past. I have always enjoyed history and like envisioning the stories. Properly memorializing these Civil War soldiers would make the site more immersive and I would take much pride in being a part of that.

Andres Santamaria

Although Dr. Giroux was not available to give us her project pitch, as soon as I saw that the project was related to machine learning and computer vision, I was confident that this was the project that I wanted to work on.

Last fall, I took Computer Vision with Dr. Niels Da Vitoria Lobo and it was one of the most interesting and engaging classes that I have taken at UCF. Not only did we learn about basic computer vision concepts such as convolution matrices and how they can be used to change an image or derive information about an image depending on the values within the matrix, we also learned how to use this data in the context of a neural network to perform classification and validation tasks on sets of images. The knowledge imparted onto us by Dr. Lobo in the beginning of the semester gave us a simple foundational knowledge of computer vision and machine learning that we needed for our final

assignment of the semester: A presentation of a computer vision research paper to the class in which we had to describe the function of the algorithm or process proposed.

When the proposition of continuing to work around and understand computer vision arose with Dr. Giroux's project proposal, I immediately thought back to the experience of parsing through research papers and trying to understand them, and how much I would enjoy having the opportunity to implement some of the great work that has already been done.

Thomas Whitaker

I was immediately interested in this project when I first heard about it due to its use of computer vision and its historical applications. Although I have taken classes in the past for artificial intelligence and machine learning, computer vision is an area I do not have much experience in. I saw this project as an opportunity to immerse myself in a technology that I believe will only become more widespread as time goes on. The problem itself is especially interesting since it is something that almost anyone can understand the basics of and yet the actual implementation will prove to be challenging.

In addition to the technical aspects of this project, I am also interested in historical applications. As someone who has had many family members serve in the US military, with many buried in military cemeteries, I really understand the importance of making sure that they and their burial grounds are properly honored. When Dr. Giroux told us about how some of the soldiers in one of the cemeteries she worked at had been mislabeled as American Indian War soldiers when really they were Civil War Union soldiers, it stuck with me. These are people who fought for our nation and they should be honored by us remembering what it was that they gave their lives for.

Broader Impacts

This project will impact people both through the historical work that it will enable and the potential broader implementation of stereoscopic cameras into the work and lives of people which would allow them to make accurate measurements simply by pointing a camera. By using this technology to determine the locations of graves that have been lost to time, US military veterans will be honored by ensuring that their graves are properly marked and that their service is recognized. It will also help to bring closure to their descendants and educate them on the history of their family.

This technology could also be used with modern stereoscopic cameras which are becoming more common on smartphones and other devices. This could bring visual distance measuring to the masses, allowing for countless opportunities to make measurements that would otherwise be impossible. It could also be useful in industrial settings, allowing workers to make measurements in situations that could be dangerous.

Legal, Ethical, and Privacy Issues

Whilst working on the project, from a legal standpoint, it was decided that any software developed by the team would not be for profit but would be developed solely for the benefit of the National Cemetery Administration. One ethical/privacy concern that came up was what happens to the user's photos that are uploaded to the web interface. This was an easy fix though as the program was developed to not store any user data.

Project Requirements

- Program to calculate distances in historic stereographic photos
    - Algorithm cannot rely on camera sensor parameters as there are none
    - Must be able to calculate depth in measurable distances, not heat maps
- User interface for placing points on a photograph to measure between
    - Place points on known measurements such as the pyramid edges
    - Return distances from headboard in foreground to left-hand pyramid, etc.
- Verifiable present-day ground-truthing of photographs
- Can run on Linux or Windows

Block Diagram

Because of how broad the scope of the problem, it was decided to divide up the tasks into major categories wherein each group member would act as a supervisor for that section. While each team member was in charge of their own section, there was room for collaboration. For example, if a member found that they may have needed assistance, they could ask another for help. In addition, if a member finds they have some bandwidth after completing a task in their own section, they can reach out to help the other members in their sections. These were seen as tentative roles evolved as the two semesters progressed.

Training data

The gathering and development of training data was supervised by Thomas. This included searching for any repositories for existing training data, acquiring/creating for gathering self-made training data, writing programs for training data synthesis, etc.

Computer vision training & processing

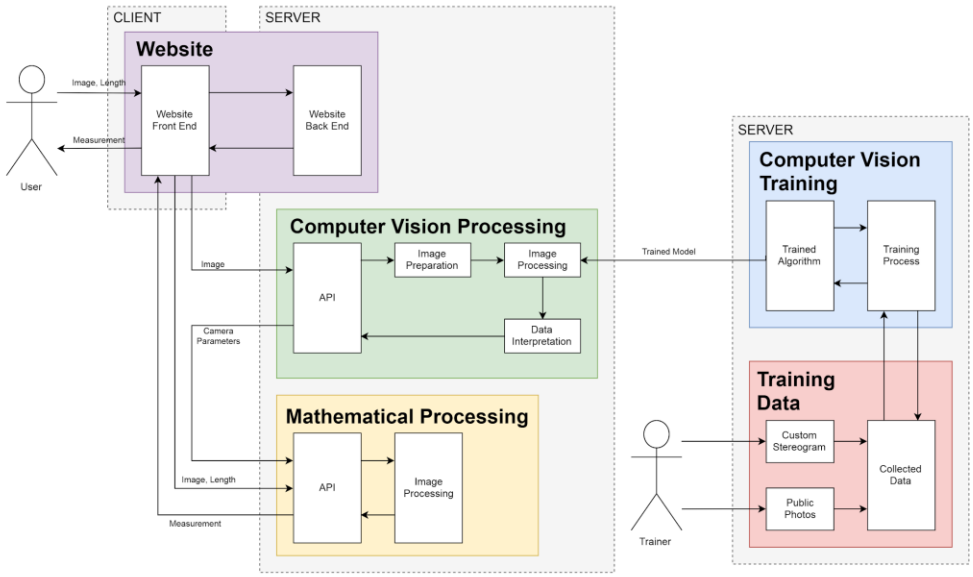
In order to accurately measure distances between points on photographs, other parameters had to be known (e.g. camera focal length, field of view, etc.). The development of a machine learning method using data gathered by Thomas to estimate these unknowns will be handled by Christopher.

Math processing

Once all of the necessary parameters are estimated, distances between points in the photographs could be mathematically determined using epipolar geometry in conjunction with computer vision. The section was overseen by Steven.

Website/User Interface

The web-based user interface in which a user uploads a pair of stereoscopic photos for distance measurements between desired points was overseen by Andres. The UI allows users to upload images with known parameters, estimate any unknown required parameters, then allow the user to place points/lines for desired measurements which it then calculates based on the mathematical portion.



LEGEND

Website
Admin: Andres
Status: Design

A website that allows users to upload a stereoscopic photo to measure the distance of a given length.

CV Processing
Admin: Christopher
Status: Research

An algorithm that uses computer vision to estimate the focal length and camera separation for a stereoscopic image.

Math Processing
Admin: Steven
Status: Design

An algorithm that uses trigonometry to measure the distance of a given length for a stereoscopic image.

CV Training
Admin: Christopher
Status: Research

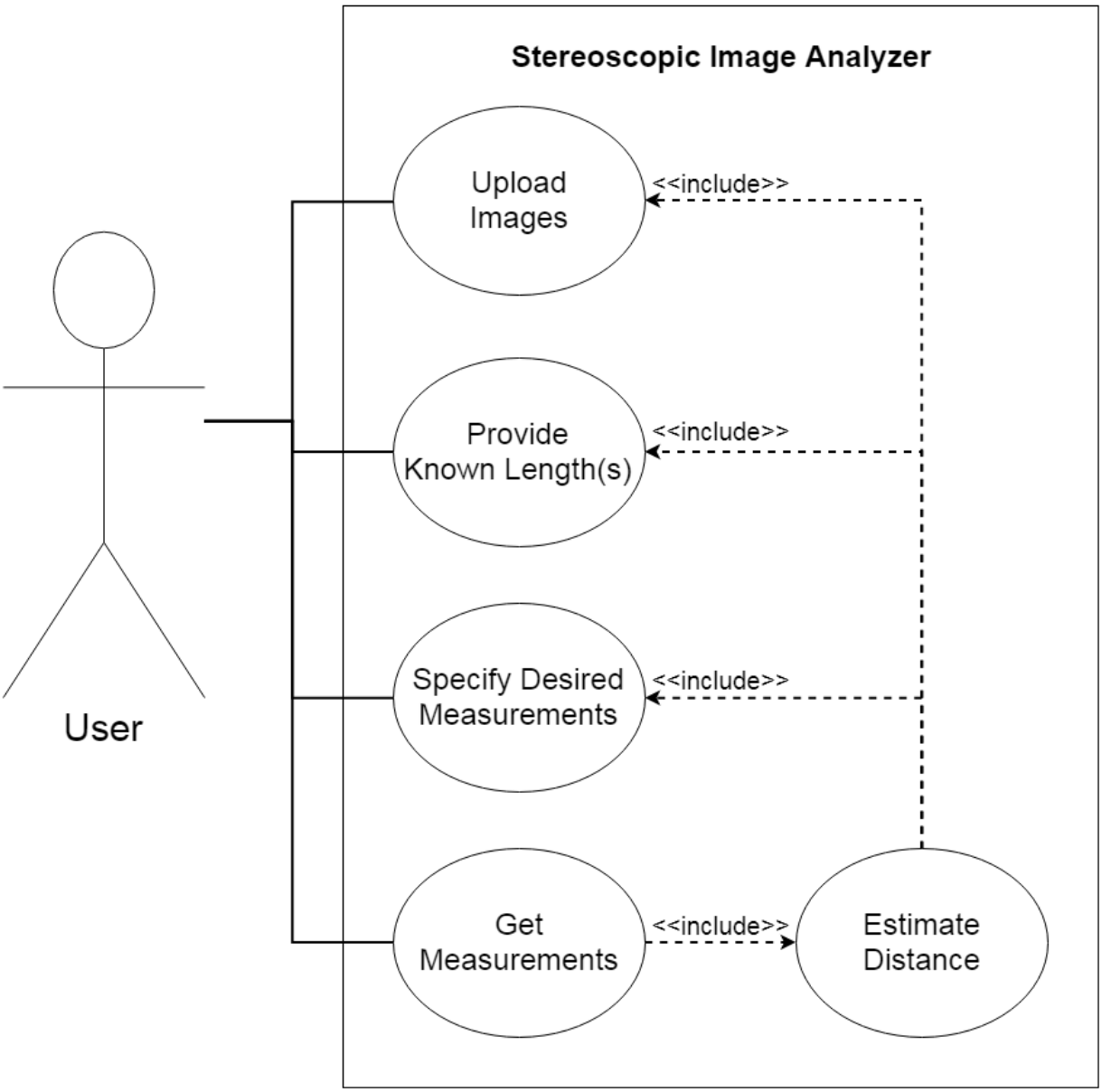
A process that trains a computer vision algorithm to estimate focal length and camera separation.

Training Data
Admin: Thomas
Status: Design

A set of data containing stereoscopic images and measurements of objects within those images.

Use Case Diagram

The user is able to upload an image with known and desired measurements into the user interface which then estimates required parameters if necessary and finally provides the calculated desired measurements back to the user.



TRAINING DATA

For any application that involves training a machine learning algorithm such as computer vision, one must have a sufficiently large set of training data that the algorithm can use to learn how to solve the given problem. In our case, we needed to train our models to be able to figure out the different parameters of the camera that took a stereoscopic image, for example, the focal length or angle of view of the camera. To accomplish this, we decided to make our own custom stereoscopic camera that could automatically take stereographic pictures and add them to a database of training data. EXIF (exchangeable image file format) data is metadata contained within an image that contains information about the camera that took the image. We scraped images and their EXIF data off the internet to bolster our training set. In addition, we also generated our own stereographic images with known measurements to validate the final results of our program.

Technical Objectives

- Investigate various methods for generating training data.
- Determine what training set size is required to sufficiently train our models.
- Determine what data is needed to be collected to train a model for a given parameter.
- Develop robust implementations of data collection devices.
- Implement a database that can store, organize, and retrieve our training data.

Technical Goals

- Understand the various parameters of monoscopic and stereoscopic cameras and how they affect the images they produce.
- Learn about the mechanical properties of a stereoscopic camera and recreate one.
- Use publicly available data on the internet to improve the training of machine learning algorithms.

Technical Specifications

- Training data consists of monoscopic or stereoscopic images associated with certain camera parameters.
- Stereoscopic camera that can be remotely and programmatically activated is constructed and used to create stereographic images with known measurements.
- Program that uses the internet to find images with usable EXIF data and automatically downloads them and their information is developed and used to gather training data for estimating parameters.
- Database that tracks and stores all of the collected data is developed.

Technical Requirements

- Stereoscopic camera, image scraper, and database must all be able to be run on or be controlled from a Windows machine.
- Database must be able to retrieve all selected data within an acceptable time frame (e.g. less than 5 minutes).
- Image scraper must be able to run continuously and download hundreds or thousands of images at a time without interruption.
- Stereoscopic camera must be able to take an image within 1 second of the remote command being given and be ready to take another image within 2 seconds of the previous picture being taken.

Research and Investigations

Initial research into a methodology for determining the size and relative distance between objects within a stereoscopic image has shown that certain parameters are required to have an accurate estimation of those distances. These mathematical-based computer vision approaches primarily require parameters that affect the appearance of the photographed object or scene in the stereoscopic images itself. The mechanics and implications of this are discussed further in later sections. For example, one of the major parameters required to make these distance estimations is the focal length of the camera that took the picture. The focal length affects the angle of view of the image, which is the horizontal range of scenery that the camera is able to capture. This angle of view can greatly impact how an object appears within an image, with a narrower view angle leading to objects appearing larger than they really are and wider view angles causing objects to appear smaller than they really are.

In order to use these computer vision methods to estimate distances within our stereoscopic images, these parameters need to be known to ensure an accurate estimation. In order to accomplish this, it was decided that training models to recognize these parameters given an image would be an ideal solution. The reason for this is primarily the fact that the target stereoscopic images are quite old, and they were taken as physical images rather than digital ones. Because of this, there is not much information about the parameters of the cameras that took these images and there is no EXIF data to use to figure it out. While it might be possible to discover the parameters of the camera used to take the stereoscopic image that Dr. Giroux initially provided to us by engaging in historical research, that would only provide a solution for that one stereoscopic image and it would not generalize. We intended to develop a solution that could be applied to many stereoscopic images.

To train these models for estimating our necessary parameters, a large set of training data need to be collected. This training data consists of both monoscopic images and the parameters of the cameras that took those images (focal length and sensor widths) as well as stereoscopic images and the measurements of known distances within those images. In the following sections, we describe our initial exploration of different methods for finding, generating, and storing these images as well as the methods that were ultimately used.

3D Models

One initial idea for collecting training data was to generate images using 3D models. It was thought that using 3D models would provide a huge benefit by automatically generating thousands of stereoscopic images where every measurement between two points in the scene is already known. There are many engines available that allow for the rendering of 3D scenes so it was believed that this method would be fairly straightforward.

If this method was chosen, it was thought that the Unity engine would have been used to generate the images. Besides being free to use, some of the team members were already familiar with using the Unity engine. Although it would not be the most efficient software for the task, the team's familiarity with the engine could have allowed for more productivity in the limited amount of time than if the 3D software had to have been learned from scratch. With the software for generating the images determined, the task of looking into the feasibility and requirements for using these images as training data would have been next.

Although initial research found that training on 3D models could have similar results compared to real-world samples, certain precautions would need to be taken to ensure their usability. The primary issue was with the texture of the generated images. A computer-generated image does not have the same noise and varied texture that a real-world image has so that would have to be simulated. This issue is further elaborated on in the paper "Learning Deep Object Detectors from 3D" by Peng et al [2].

Because of the time and effort that would be required to make our models accurate and realistic enough to be useful, it was decided that the approach of using 3D models to generate training data was infeasible. Further research into how distances would have been measured within stereoscopic images showed that this kind of training data would not be useful anyways since the machine learning algorithm would not be used to estimate distance itself.

Stereoscopic Camera

It was obvious that the team needed its own stereoscopic camera. Having a stereoscopic camera meant being able to produce stereographic images of scenes with known measurements. This would be exceptionally useful as it would allow for the verification and fine-tuning of the computer vision algorithm that would be developed to estimate distances within those stereoscopic images. As stereoscopic photography is more of a niche activity, there were not a lot of sufficient resources online to provide adequate sample images with known measurements. Because of these reasons, it was decided to develop an affordable, adaptable, and mobile solution that would allow for the creation of stereoscopic images that were needed.

Dr. Giroux initially provided two Nokia Lumia 521 smartphones, which she suggested be used as the individual cameras within the stereoscopic camera. It was agreed that using smartphones was the best solution for creating this camera as purchasing a retail digital stereoscopic camera would cost hundreds of dollars and have limited adjustability in its parameters. In addition to these two smartphones, it was determined that a mount would also be needed for the phones that could fit on a standard tripod that Dr. Giroux provided for us. In addition, it was also determined that a custom control software that would be able to control and coordinate the smartphones would need to be developed. This software would need to be able to remotely activate the cameras on the smartphones and take two pictures within a second of each other and then store those images in a database.

As well as creating a custom stereoscopic camera, it was also determined that a procedure for taking these stereoscopic images and creating useful training data with them would also need to be developed. Dr. Giroux provided a 400-foot-long tape measure that would allow for the measurement and recording of a wide range of distances within an acceptable degree of accuracy. The team ended up taking a trip to the St. Augustine cemetery with the modern stereographic camera to recreate the original stereoscopic image and also to take more photographs and measurements of the area.

One alternative option that was explored in addition to the dual smartphone design was to use a specialized phone attachment that used a set of mirrors to take stereoscopic images with a single smartphone. It was initially thought that such a set up would be superior to the dual smartphone design as it would allow for a much simpler system for collecting stereoscopic images with fewer points of failure and greater flexibility in terms of mobility and compatibility with different kinds of smartphones. Dr. Giroux ordered one of these specialized attachments and some sample photos were taken. It was found that

the stereoscopic photos taken with the attachment had an overlap between the two sides of the image which made them unusable. For this reason, it was decided to abandon the option of using a single camera with a specialized attachment.

Example Image Taken With Mirror Stereoscopic Attachment

The Nokia Lumia 521 smartphones that Dr. Giroux provided were Windows phones which meant they were running the Windows Mobile 8.1 operating system and could only run apps that were on the Microsoft Store. This was initially viewed as an issue since having a Windows phone meant having less control over the phone and fewer options for software to use than having a phone that was running Android. For this reason, the possibility of installing Android on these phones was explored. Although it is generally a straightforward task to install Android on a Windows phone, in this specific case it would be neither possible nor useful to do so. In order to install Android onto a Windows phone, a user would need to edit the read-only memory, or ROM, of the phone to allow a different operating system. This ROM software is unique for different models of phones so whether or not one can find the right ROM software for their particular phone model depends on if enough people are wanting that software for people to develop it. In our case, the phones were old enough that developing ROM software for it was largely abandoned. In addition, the software that was developed had a technical issue where the camera on the phone could not be accessed from an Android environment

which completely defeated the purpose of installing Android in the first place. Because of this, it was decided to abandon this path of research and explore alternative options.

Exploring the available options in the Microsoft Store, an app called “Win IP Camera” developed by Biyee SciTech Inc was found. This app would essentially turn the Windows phone into an IP accessible webcam. A live MJPEG stream which is basically a constant stream of new JPEG images over a local network would become accessible. While this would make remote access to the smartphone camera quite simple, it had one major restriction in that it would require operating in an area where there was a wireless network where we could access local devices. While this would not be an issue on the UCF campus, it could have been a problem at the St. Augustine Cemetery. The possibility of using a wireless hotspot generated by a third mobile phone to act as the local wireless network however was explored but another problem emerged.

The IP webcam app for the smartphones was unstable over extended periods of time. After a few minutes, the live MJPEG stream from the camera would freeze and would have to be restarted from within the app. There were a few occasions where the app would freeze completely, locking up the phone and requiring a full restart. It was believed that the cause of this issue was the phone’s limited hardware capabilities, primarily because the app provided a warning when it started up that the phone’s hardware may have been insufficient for reliable streaming. In addition, the phone’s RAM usage was nearing 100% while the app was in use. It was determined that the phones that Dr. Giroux provided would not be able to support the live streaming of photos regardless of the software being used.

As an alternative to streaming photos over a local network, the possibility of accessing the phone’s camera via a USB connection was explored. An application developed by Microsoft called “Project My Screen” which allowed for the streaming of a phone’s screen to a Windows machine via a USB cable was discovered. This video stream proved to be of high quality, low latency, and extreme stability. However, the software only supported streaming from one smartphone at a time and there was no way to directly access the smartphone camera via USB. It was determined that using the Windows phones would have issues regardless of how they were accessed, so another model of smartphone was looked into.

The options for what model of smartphone to use for the camera was limited by the phones available to the team. A request was sent to the University of Central Florida Center for Distributed Learning asking to borrow any of the smartphones they use for development. They were gracious enough to accept the request and upon review of the

smartphones the team personally had available to use, it was decided to use the iPhone 8 since one of our team members already had one and UCF CDL had one to spare. (see Appendix)

The iOS app store was searched for apps that would allow for the remote access of the iPhone cameras. It was eventually decided to try the iVCam Webcam app by e2eSoft because it was free, allowed for connection via network and USB, and supported streaming from multiple smartphones simultaneously. The app was tested, and it worked exactly as expected. It gave the option of adjusting the stream frame rate, image quality, and image resolution. After some testing, it was decided to have a stream with high image quality, a 1920 x 1080 pixel resolution, and a 15 frames-per-second frame rate. There was an option to stream an even higher resolution, however, it was found that the stream was unstable at that resolution. A low frame rate was chosen to minimize the system load on the iPhone and the computer running the camera control software to maximize the stability of the stereoscopic camera.

The final unanswered question was how the camera would be programmatically accessed using Python. It was found that the Python OpenCV library had functions specifically designed to capture frames from a live camera. One feature of the iVCam software that was especially beneficial is the fact that it registers the smartphone camera as an actual webcam on the receiving system rather than simply providing an IP accessible MJPEG stream like other software that was tried. Because of this, accessing the live stream from the camera was trivial with OpenCV. A prototype of the camera control software was created and tested using this method and the team was satisfied with the results.

Image Scraping

Although having a stereoscopic camera is useful for collecting sample stereoscopic images, it was not suitable or adequate to collect the amount of training data needed to train the models. During initial brainstorming, the idea to use publicly available images from the internet as training data for the models came up. It was decided to create a program that would be able to automatically gather pictures from the internet, process and extract their relevant information, and store them in a database.

One public image repository website that immediately came to mind was Flickr. Flickr has millions of images that are publicly available to download and they have a REST API that makes it easy to automatically download images and their EXIF data. The team began reading Flickr's API documentation and developing a Python script to access it and download images. However, a problem came up when it came time to request an API key. To receive a Flickr API key, one must create a Flickr account and request the key through the developer portal. Since none of our team members had a Flickr account, it was attempted to create one. It was found that no matter what email address was used to try and register an account with, Flickr would reject it saying there was already an account associated with that email. This was confusing since none of the team members had used any of their emails to make Flickr accounts previously. The team emailed Flickr's support to try and resolve the issue. Flickr's support informed the team that they were in the process of migrating their login and registration system and that new accounts could be created once this migration was complete. Given this response, it was decided to search for alternative websites to Flickr to gather images from. The team did eventually receive another email from Flickr saying that new accounts could be made, and the team was able to do so successfully and request an API key. However, at that point, the team had already found other sources for training data. (see Appendix)

Photobucket was considered to be used as a source of photographs, but it was learned that they remove all EXIF data from the images uploaded to their website. Without this data, there would be no way to determine the relevant parameters of the camera that took a given image making the image useless for the purpose of training. Many online photo sharing services that were examined also removed the EXIF data from images uploaded to their sites.

One potential alternative to Flickr was a website called SmugMug. SmugMug, which interestingly enough owns Flickr, has a similarly robust API that would make it easy for a user to collect images and their metadata from their website. Although SmugMug does

not have as many photos as Flickr, their selection was still large enough to be sufficient for the team's needs. Unfortunately, SmugMug was also similar to Flickr in that an account is required to request an API key. While the team was able to successfully make a SmugMug account and request an API key, it was discovered that SmugMug is a paid service that only offers a short trial membership. That would mean that the team would need to either continually make new accounts and request new API keys or purchase a SmugMug subscription. Neither of these options was desirable, so it was decided to continue looking for alternative sources of photographs.

While exploring alternatives to Flickr for finding images online, the team had the idea to use the publicly available images on Wikipedia. It was found that Wikimedia, Wikipedia's media host, has an API that allowed for the downloading of images and their EXIF data in a way very similar to the Flickr API with the added benefit of not having a rate limit (although the API maintainers request that users not submit excessive API requests.) There was also the benefit of being able to select specific pages/topics for images, giving finer control over the content of the images that were downloaded.

During research into various methods for analysis stereoscopic images, a dataset used by Cornell for developing 3D models from multiple pictures (1DSfM) was discovered. This data set contained thousands of monoscopic photographs, with a large majority of them having the focal length of the camera that took them contained within their EXIF data. This data set was exactly what was needed and provided a solution to the problem of not being able to collect images with EXIF data from online image repositories.

In addition to the Cornell dataset, another much larger data set from a research team in China that also had images and their corresponding focal lengths and sensor widths (FocaLens) was discovered. The team ultimately ended up using the Cornell and FocaLens datasets alone for the training data as together they contributed over 200,000 labeled images.

After figuring out ways of acquiring images, the next step was figuring out how to retrieve the parameters of the cameras that took the images. The found APIs had endpoints for retrieving that metadata, so the collection of images from those APIs was easy. However, it was still needed to find a way to get that information for local images. A few different Python libraries for extracting EXIF data from jpeg images was explored.

The first library that was examined was Pyexif. The library was appealing since it was being actively maintained, however, upon testing it was found that it had difficulty extracting the EXIF data from most of the images it was tested on. PIL, the Python Image

Library, was looked into next to extract the EXIF data. PIL had a lot more success than Pyexif, being able to extract the EXIF data from images about 90% of the time. However, PIL is a general-purpose image library and as such, was not optimized for this task. As a result of this, it was found that the extraction process was slow, so it was looked into seeing if there were any other alternatives.

The final library that was looked at was ExifRead. Although it had not been updated since 2015, it was able to extract EXIF data from 87% of the sample images and process a set of 4000 images in about a minute. Because it had a comparable success rate to PIL and ran much faster, the team decided to use ExifRead.

Getting the focal length from an image's EXIF data was straightforward. ExifRead represents the focal length as a numerator and a denominator, so all that had to be done was divide the numerator by the denominator and round to the nearest decimal place. Unfortunately, an image's EXIF data does not explicitly state the sensor width of the camera. However, a formula that allowed for the calculation of the sensor width based on information that is available in the EXIF data was discovered [3]. The user also recommended referencing the DxOMark camera database for more accurate measurements.

Database

With so much data being collected, a system for storing and organizing all of it was needed. There was not a lot of complexity, so it was decided that a relational storage system would be sufficient. The team looked into two primary methods of storing the data.

The first method looked into was using a simple XML or CSV file to keep track of the data. While such a method would be very simple to implement since there are no built-in methods for accessing the data efficiently any lookup times would be linearly related to the size of the data set. This meant that as the data set grew larger, accessing the data would become unwieldy and unacceptably slow. For this reason, it was decided to take a more technically-involved approach and use a database.

Since it was decided to use a database, MySQL was immediately considered since it was relational which fit the format and usage of the data well and because it was well documented and multiple team members were familiar with using it. The team found the Python library PyMySQL which allowed for the easy execution of queries on the database using a Python script which worked perfectly since both the stereoscopic camera control software and image scraping program used Python.

Originally, the team considered using a simple database with a single table that would contain an image and its associated parameter in each row. While such a database configuration would be usable, it was nowhere near-optimal. During the initial project status presentation in Senior Design 1, Dr. Heinrich suggested the team separate the image parameters into separate tables linked by a foreign key. Such an arrangement would allow for easily associating multiple image parameters with a single image, greatly increasing the flexibility of the database. In addition, his suggestion also gave the team the idea to create different tables for different image parameter types. This arrangement would allow for easier and faster lookups for different types of parameters which would come in useful while retrieving training data for a specific model.

Design Summary

Stereoscopic Camera

The stereoscopic camera is designed to quickly and easily take large amounts of photos with variable parameters. It consists of two smartphones mounted on a tripod with an adjustable distance and angle between them. The smartphones are connected via USB to a laptop. A Python script accesses the smartphones and automatically takes pictures using their cameras then saves those images to a database.

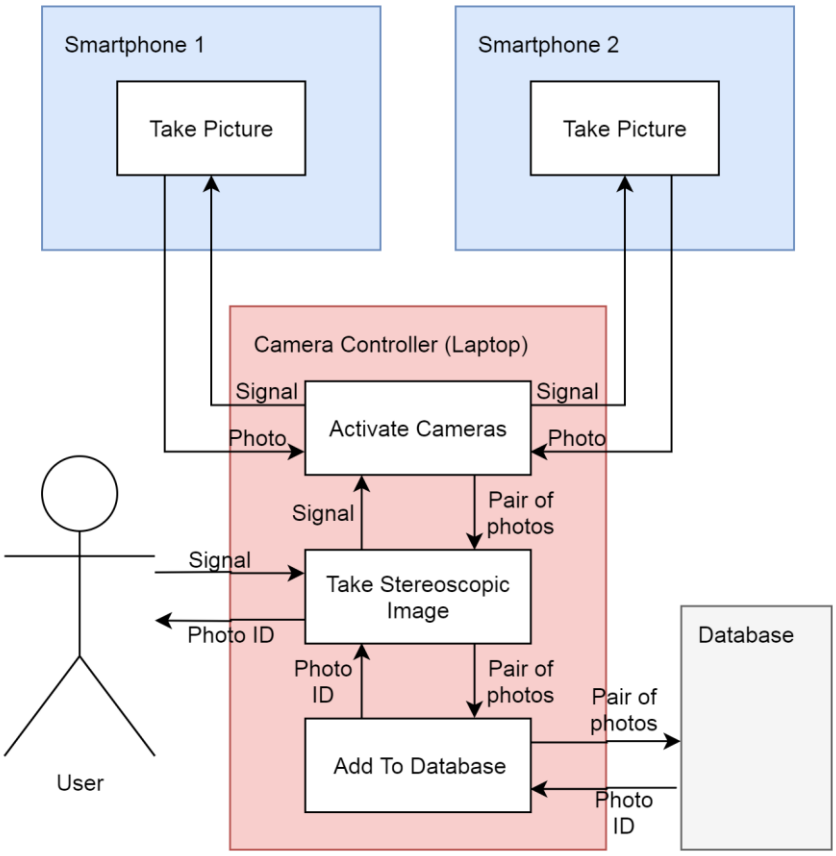
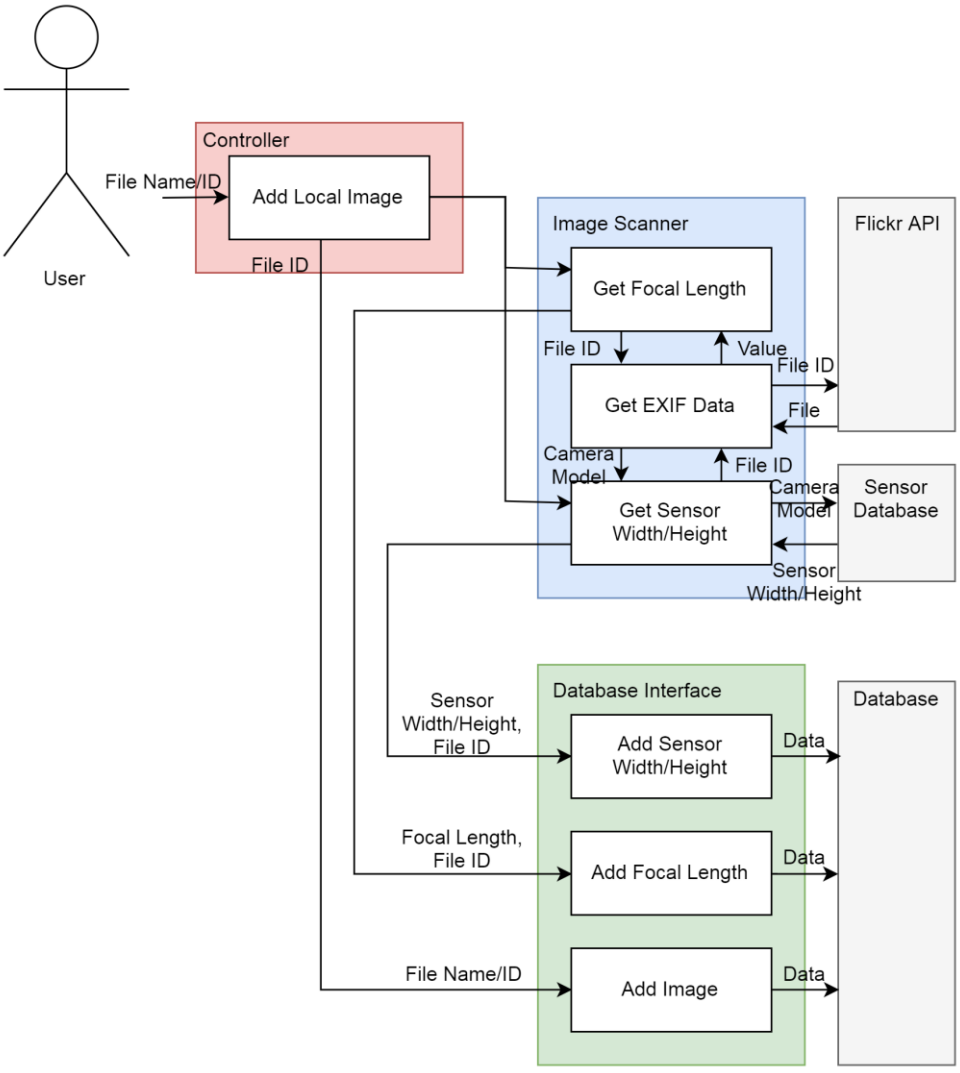


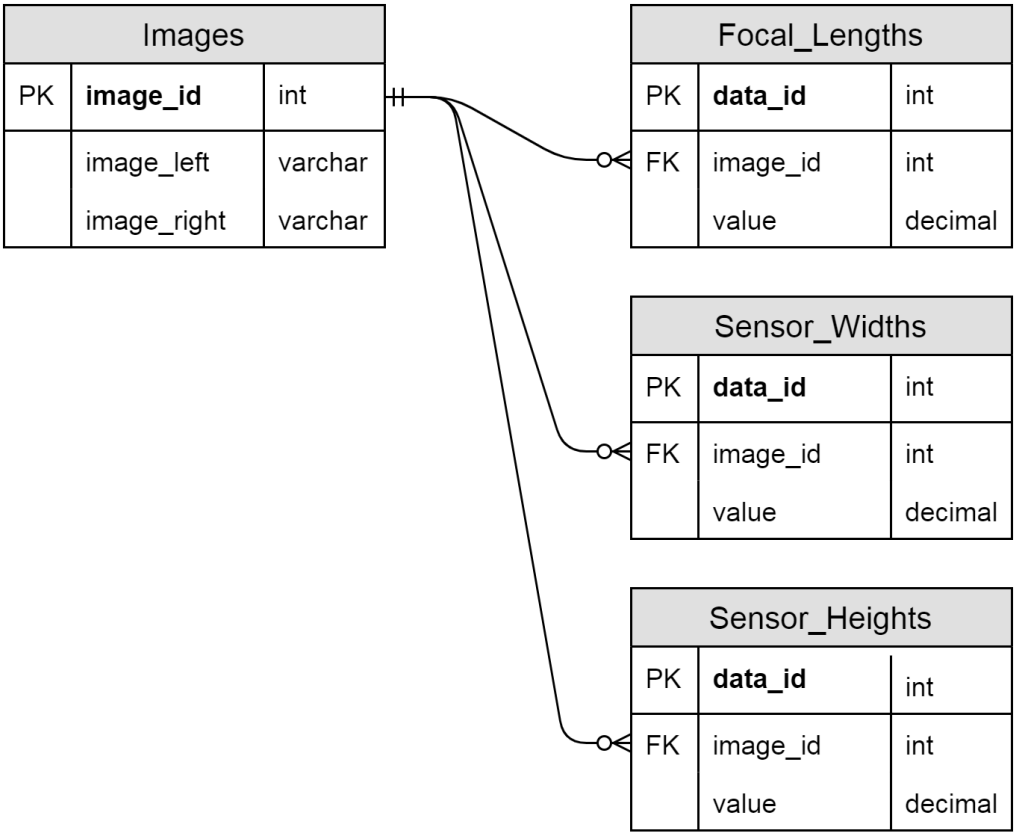
Image Scraper

The image scraper allows for the automatic collection and collation of images and their parameters from downloaded datasets or websites with supported APIs, greatly bolstering the size of our training data set and increasing the accuracy of our image parameter estimation models. It uses a Python script to access publicly available images from the internet. It then downloads those images and adds them to our database along with the relevant parameters that have been extracted from their EXIF data.



Database

The database stores, organizes, and retrieves the training data upon request. It consists of a single MySQL database with three tables. An image table keeps track of all images added to the database, both monoscopic and stereoscopic, and associates them with an image ID. A focal lengths table associates a focal length with a specific image. A sensor widths and a sensor heights table associates a sensor width/height with a specific image.



Design Content

Stereoscopic Camera

The stereoscopic camera consists of three primary components: The camera tripod, the camera smartphones, and the camera control software. These components are designed to allow for maximum portability and ease of use. Most of these components have been purchased or donated, however, the team did develop the software that controls and coordinates the cameras and interfaces with the database.

The camera tripod is built using a regular tripod that was given by Dr. Giroux. A Neewer 8” dual camera tripod mount was used to have two mounts on a single tripod with an adjustable distance between them. Two Neewer smartphone tripod mounts were also used to hold the smartphones in place. Neewer equipment were chosen both for both the dual mount and smartphone mounts to ensure maximum compatibility between the components. USB cables were also used to charge and access the phones.

Mounted on the tripod are two iPhone 8s. These smartphones act as the individual lenses of the stereoscopic camera. They are running the app iVCam Webcam, which converts the smartphone camera into a USB accessible webcam. The app is set to stream a high-quality image of 1920 x 1080 pixels, with the lowest selectable frame rate of 15 frames per second. Although the app provides the option of streaming via a wireless network, it was chosen to stream via a USB cable to ensure low latency and operability regardless of the availability of a wireless network.

The camera control software is written in Python and runs on a Windows machine. The control software works in conjunction with the iVCam Webcam desktop client which provides the necessary interface for accessing the smartphone cameras. In addition to the desktop client, a special driver provided by e2eSoft is installed twice to allow access to two smartphone streams simultaneously. iTunes must also be installed to provide the proper drivers to access the iPhones via USB.

The control software instructs the user to first activate the left camera and then the right camera. It then displays the live stream from both of the cameras so the user can see what the cameras see before taking a picture. The user may press the spacebar to take a stereoscopic image or press the E key to exit the program. The control software then uses the VideoCapture function from Python OpenCV to capture the frames being streamed by the two cameras. When the spacebar is pressed, the most recent frame from each camera

is captured and saved locally. The path to these images is added to our database using the PyMySQL library.

The Team Setting Up and Testing the Stereoscopic Camera Apparatus

Stereoscopic Image Pair Taken With Apparatus at St. Augustine Cemetery

Image Scraper

The image scraper is designed to be modular meaning that a module can be added for different sources of images to make integration and expansion of sources easy. There is a primary control interface module that allows the user to easily add new photos and their data to the database. It allows the user to either find images from the internet or directly add a local file.

Local images are analyzed using the ExifRead library for Python. The user provides a path to a set of images, and ExifRead attempts to extract the EXIF data from those images. If at least one of the necessary parameters is successfully extracted, the image is added to the database and is assigned an image ID. The focal length is represented as a ration with the unit being millimeters. If a valid focal length value is extracted, i.e. nonzero and less than 1000mm, then it is added to the database and associated with the image by its image ID.

The sensor width is not stored in the EXIF data, but it can be either looked up or calculated based on other given parameters. If the EXIF data contains the model of the camera that took the picture, a publicly available database of cameras can be used to find

the sensor width. Otherwise, the horizontal resolution in pixels can be divided by the focal plane resolution in pixels per inch to determine the sensor width in inches.

Database

The database is implemented using MySQL 8.0. It consists of five tables, one of them storing the image ID and the other four storing information about parameters or measurements associated with those images. The team also has a few prepared SQL queries that allow for the simple adding and retrieval of information to and from the database.

The primary table is the images table. It is designed to store the image IDs. The image ID acts as the primary key for the table and it consists of an int that is not null and is automatically incremented by one on new additions to the table. Associated with this image ID are two columns for storing images: the left image and the right image. For a stereoscopic image, the name of the image on each side of the stereoscopic images is saved in its respective column. For a monoscopic image, the name of the image is saved in the left image column and the right image column is set to be null. It was created using the following SQL query:

```
CREATE TABLE Images (
    image_id INT NOT NULL AUTO_INCREMENT,
    image_left VARCHAR(40) NOT NULL,
    image_right VARCHAR(40) DEFAULT NULL,
    PRIMARY KEY(image_id)
);
```

The three tables used for storing image parameters associated with monoscopic images are identical in structure (focal length, sensor width, and sensor height). They are separated purely to speed up the retrieval of those parameters. Each table has a data ID column that acts as the primary key. This data ID is not used for our training. It is used purely to differentiate rows within the tables. Each table also has an image ID column. This acts as a foreign key referencing the images table and acts as the associative link between images and their parameter values. There is a one mandatory to many optional relationship between the images and their parameters. Finally, in each table, there is a value column that records the value of that specific parameter for that image. It is a decimal data type storing with an accuracy of one decimal place ranging between -999.9

and 999.9. Both tables were created using the following SQL query with the name changed respectively for each one:

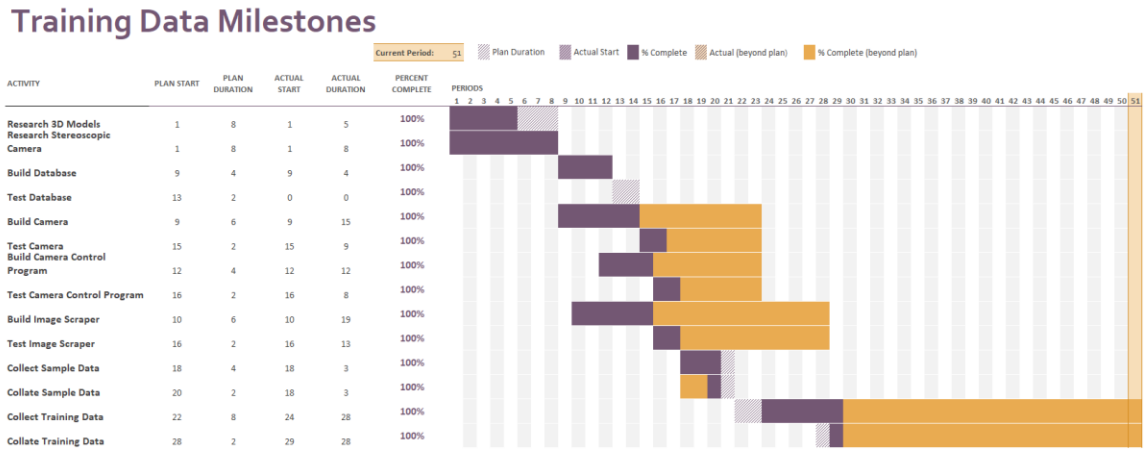
```
CREATE TABLE Focal_Lengths (
   data_id INT NOT NULL AUTO_INCREMENT,
   image_id INT NOT NULL,
   value DECIMAL(4,1) NOT NULL,
   PRIMARY KEY(data_id),
   FOREIGN KEY (image_id) REFERENCES Images(image_id)
);
```

Adding information to the database is simple. The first step is to add the image id to the images table. The user provides the path(s) to the image(s) and then retrieves the image ID for the image they just added. They then add information about the image to the measurements, focal lengths, or sensor width tables. They must provide the ID of the image that the data is associated with, the decimal value of the parameter, and a pair of two points if they are adding a distance measurement to a stereoscopic image.

Retrieving information from the database is also simple. It is designed so that data can be retrieved by the parameter type. To retrieve all of the data collected about focal lengths, the user would execute the following query:

```
SELECT Images.image_left,
       Focal_Lengths.value
FROM Images
INNER JOIN Focal_Lengths
ON Images.image_id = Focal_Lengths.image_id;
```

This would retrieve the path to the image as well as the decimal value of the focal length. To retrieve sensor widths, the user would execute the same query where Focal_Lengths is replaced with Sensor_Widths.



Development Timeline

Research 3D Models

May 22nd - June 8th
Determine the feasibility of using 3D images to automatically generate stereoscopic images with known measurements. It was determined that the generated images would be insufficiently realistic to train a model, so the idea was abandoned.

Research Stereoscopic Camera

May 22nd - June 18th
Determine the feasibility and methodology for creating our own stereogram. It was concluded that a stereogram using smartphones that are remotely accessible would be created and that it would be sufficient for producing training data.

Build Database

June 20th - July 13th
Create a database that is able to receive, store, and retrieve large amounts of training data collected from multiple sources. The team used MySQL to build a database that can store both monoscopic and stereoscopic images, as well as the relevant EXIF data and measurements of distances within the images.

Test Database

July 14th - July 28th
Verify that the database fulfills all requirements, is stable, and can handle large amounts of data without issue.

Build Camera

June 20th - July 28th
Construct the stereoscopic camera that we used to collect training data. This included setting up the smartphones and installing the necessary software, setting up the mount, and preparing any other equipment necessary for the usage of the stereogram.

Test Camera

July 29th - August 3rd
Verify that the stereogram meets requirements and is usable in the places we will go to collect training data.

Build Camera Control Program

July 7th - July 28th
Create a program that is able to remotely access our stereogram, take pictures, and automatically store them and their associated data in our training data database.

Test Camera Control Program

July 29th - August 3rd
Verify that the program meets requirements and works as expected.

Build Image Scraper

June 24th - July 20th
Create a program that is able to crawl supported websites and download images and their EXIF data, process those images as necessary, and automatically add them to our database. It is implemented in Python, with support for adding local files, Flickr photos, and Wikimedia photos.

Test Image Scraper

July 21st - July 27th
Verify that the program meets requirements and runs as expected.

Collect Sample Data

July 21st - August 10th
Begin collecting a small sample set of training data to be used to verify the efficacy of the collection methods and have data to work with as the models are being trained. This involved going to St. Augustine and taking the photos firsthand, as well as scraping images from the internet. The set consisted of a few thousand photos.

Collate Sample Data

August 6th - August 10th
Organize the collected data and prepare it to be used for its various functions.

Collect Training Data

August 14th - November 25th
Collect more training data to increase the accuracy of the trained models. The training data set was expanded to around two hundred thousand photos.

Collate Training Data

August 25th - November 25th
Organize the collected data and prepare it to be used for its various functions.

Build Plan

1. Build stereoscopic camera.
   The camera consists of two smartphones mounted on a camera tripod. The smartphones are mounted in such a way that the distance between them is adjustable. It can take two photos with each smartphone within a second of each other. It is mobile so that it can take photos in many locations. It is controllable via software on a laptop, and this software also automatically adds the images taken to a database.
2. Build image scraper.
   This software can find and analyze photos to extract parameters that we need to train our models. It can add that information to a database. It can also accept images that we provide to it locally and it is also modular in a way that we can easily add support for different websites. The program can access these supported websites and download more images for us to use as training data.
3. Build database.
   The database can accept both monoscopic and stereoscopic image ids. It also accepts parameters that are associated with a specific monoscopic image or a set of points and a distance associated with a specific stereoscopic image. It is configured in a way that is optimized for retrieving data by the type of parameter being requested.
4. Collect data.
   The image scraper and our stereoscopic camera are used to collect data and populate the database. We aimed to have a few hundred measured distances associated with some stereoscopic images as well as several thousand monoscopic images along with their associated camera parameter data.

Prototype Plan

1. Ensure cameras are remotely accessible.
   Install the necessary software on smartphones and attempt to remotely take a picture with it from a laptop. This is the essential component of the camera and determines the implementation of the stereoscopic camera and its control software.
2. Download and process sample set with image scraper.
   Use the image scraper to download a few images using the supported APIs to make sure that more images can be downloaded. Also, test the extraction of EXIF data from those images to make sure the method is robust enough to analyze images from many different sources.
3. Evaluate lookup times with database.
   Ensure that the configuration of the database is such that looking up images by a parameter type is easy and is performed within a reasonable amount of time.

Test Plan

1. Camera integration tests.
   Test the camera system as a whole to make sure it works as expected. Use it in a controlled environment to ensure compatibility between all parts.
   Components to test include:
   - Photos indoors
   - Photos outdoors
   - Environment with Wi-Fi network
   - Environment with mobile hotspot
   - Photos automatically added to database

Evaluation Plan

1. Evaluate stereoscopic camera.
   Determine if the camera meets the specifications we started out with. The camera should have an adjustable distance between the smartphones, and it should be mounted on a mobile tripod. The camera should be able to be accessed remotely, and both be instructed to take pictures within a second of each other. This ability will be tested by taking multiple pictures of a running timer so the total delay can be calculated. The camera control software must also be tested to make sure that the stereoscopic photos are indeed added to the database as specified.
   Ultimately, setting up the camera apparatus was straightforward, and it worked in a local setting with Wi-Fi. However, when the team went out to the cemetery, the software was malfunctioning (the Bluetooth and hotspot were not stable, and the USB connection was faulty). As a result, the pictures were manually taken on both iPhones while they were mounted to the tripod. The end result was the same.
2. Evaluate image scraper.
   Determine if the image scraper fulfills the needs that necessitated its development. The team should be able to add locally available images to the database, with the relevant camera parameters being added automatically. The program should also be able to download images from supported websites. The program should run in a reasonable amount of time, with the time to add a single photo being a second or less.
   The image scraper worked perfectly. With it, over two hundred thousand images and their associated metadata was acquired.
3. Evaluate database.
   Determine that the database is able to accept and store both monoscopic and stereoscopic images IDs and is able to associate relevant parameters with that data. The database should be configured in a way that images labels and parameter values can be retrieved by a specified parameter.
   The database worked perfectly. It was able to store all information about the images in a clean, queryable way.

Technologies and Equipment

LANGUAGES

Python

Creator: Guido van Rossum
Use: General purpose programming language
Features:

- Interpreted and high level
- Cross-system compatibility
- Rich library ecosystem

Python was selected as the programming language with which the team would implement the stereoscopic camera control software and our image scraper. It was chosen because it is extremely flexible, with a wide range of libraries to choose from and giving multiple options on systems to run the code on and interface with. Python is also the language that the team members working on this portion of the project had the most expertise in.

SQL

Creators: Donald D. Chamberlin and Raymond F. Boyce
Use: Database manipulation and query language
Features:

- Relational
- Statically typed
- Cross-platform

SQL, specifically MySQL, was chosen for the database because its relational nature is well suited for the kind of data to be stored. SQL is well supported and documented, so the team has a greater ability to create a database that fulfills our requirements by using SQL. It is also cross-platform, giving flexibility in how to host the database.

LIBRARIES

OpenCV

Creator: Intel Corporation
Use: Webcam stream frame extraction and manipulation
Features:

- Direct webcam stream access
- Frame extraction from video stream
- Frame cropping and combination

OpenCV is a library with a wide array of tools for working with images and other visual media. The team is using a Python version of the library to access the cameras on the two smartphones for the stereoscopic camera. This library also allows for the cropping and manipulation of frames from the camera feeds then the saving of those frames locally.

PyMySQL

Creator: PyMySQL Community
Use: Access a MySQL database using the Python programming language
Features:

- Execution of queries on database using Python
- Access to database information like last inserted row ID

PyMySQL is a Python library that allows for the execution of queries on a MySQL database using a Python script. It is used in the camera control software and the image scraper to interface with the database so new information can be added to it.

ExifRead

Creators: Gene Cash and Thierry Bousch
Use: Read EXIF metadata from tiff and jpeg files
Features:

- Easy to process images
- High rate of successful metadata extraction
- Well organized metadata dictionary

Although ExifRead is older and not as well supported as other EXIF Python libraries, it was found to be the most reliable and easy to use. With one function call, a dictionary can be generated containing all the EXIF metadata contained within the collected images. This library is being used to extract information about the cameras that took the images the team collects from the internet so that this can be used to train the models.

SOFTWARE

iVCam Webcam

Creator: e2eSoft
Use: Allows a smartphone to be used as a webcam
Features:

- Phones can be accessed via network or USB
- Stream parameters such as frames per second and resolution are adjustable
- Support for multiple simultaneous camera streams

The iVCam Webcam software is what allows the team to remotely activate the cameras on the two smartphones. It allows the team to access the cameras as if they were webcams, thereby enabling the team to save frames to take stereoscopic images with the two smartphones.

EQUIPMENT

2 x Neewer Smartphone Holder Vertical Bracket

Creator: Neewer
Use: Allows smartphones to be mounted on a standard tripod mount
Features:

- Adjustable bracket width allows for the use of several smartphone models
- Rotatable mount allows for pictures to be taken in a portrait or landscape orientation

These smartphone holders allow the team to securely mount smartphones on the tripod. This is important to ensure that the smartphones don’t shift position in between pictures and become misaligned.

Neewer Dual Camera Mount Tripod Bracket

Creator: Neewer
Use: Allows for two items to be mounted in parallel on a single tripod mount
Features:

- Dual mounts are aligned in parallel
- Adjustable base width allows for control over the distance between cameras

This dual camera mount allows the team to have two smartphones mounted in parallel on the tripod which is an essential component of the stereoscopic camera. Since both the dual camera mount and the smartphone brackets are manufactured by Neewer, the team can ensure maximum compatibility between the components.

Camera Tripod

Use: Keeps the cameras level and at a fixed height above the ground
Features:

- Adjustable height
- Portable

This camera tripod is what holds the dual smartphone mount above the ground.

2 x iPhone 8

Creator: Apple
Use: Acts as the cameras for our stereoscopic camera
Features:

- iOS operating system allows for apps to be run
- High-quality camera
    - 12 Megapixels
    - f/1.8 aperture

These smartphones are used as the individual cameras in our stereoscopic camera. One of the phones is owned by one of the team members, and the other phone is being borrowed from the UCF Center for Distributed Learning.

2 x Lightning to USB 2.0 Type-A Cables

Creator: Apple
Use: Connects and charges smartphones to the laptop running the control software
Features:

- Provides both data and power transfer
- Length allows for flexibility in camera and laptop placement

These cables allow the team to access the smartphones via USB and keep them charged during usage. Connecting to the phones via USB instead of using a local wireless network is preferable because it allows the team to use the stereoscopic camera anywhere rather than just in an area where there is a local wireless network that allows for peer to peer connections.

COMPUTER VISION ANALYSIS

There exist numerous mathematical-based computer vision programs to estimate the size of objects, the distance between objects in images, and the distance of objects from the camera lens. However, said algorithms require certain known parameters before they can attempt these calculations (e.g. focal length, angle of view, etc.). The focal length (inversely proportional to field of view) and angle of view of the Civil War-era photographs were unknown. Before any mathematical processing could be done, these had to be estimated. It was believed that these unknown parameters could be estimated using machine learning techniques which are explained in this section.

Technical Objectives

- Investigate different machine learning frameworks/algorithms for computer vision analysis.
- Determine which framework is able to solve the parameter estimation problem.
- Determine the training requirements/dependencies for each framework.
- Determine which frameworks are compatible with the stacks that the group is planning on implementing in the user interface.
- Implement chosen framework.
- Fine-tune, debug, and test accuracy of framework.
- Pass output of framework to mathematical section for distance estimation.

Technical Goals

- Learn about different machine learning algorithms (e.g. convolutional neural networks, supervised deep learning, etc.).
- Learn about different computer vision programs (e.g. OpenCV, etc.).
- Use machine learning in conjunction with computer vision to assist with and improve efficiency/accuracy of image-based distance measurement algorithms.

Technical Specifications

- Frameworks will be trained on a broad dataset of images in which the target parameter of said images are known.
- Frameworks will accept an image in which the target parameter is unknown then using the results of the training data, estimate this.
- Framework will either output parameter or transfer it for use in another program (e.g. distance calculation).

Technical Requirements

- Framework (CNN) must be Windows/Linux compatible.
- Framework must be compatible with the chosen stack used for the user interface.
- Framework must be fast (i.e. able to calculate parameter in seconds, thereby not keeping the user waiting too long).
- Framework must be able to accurately estimate measurements with a low margin of error (e.g. 10%).

Research and Investigations

Potential Solutions

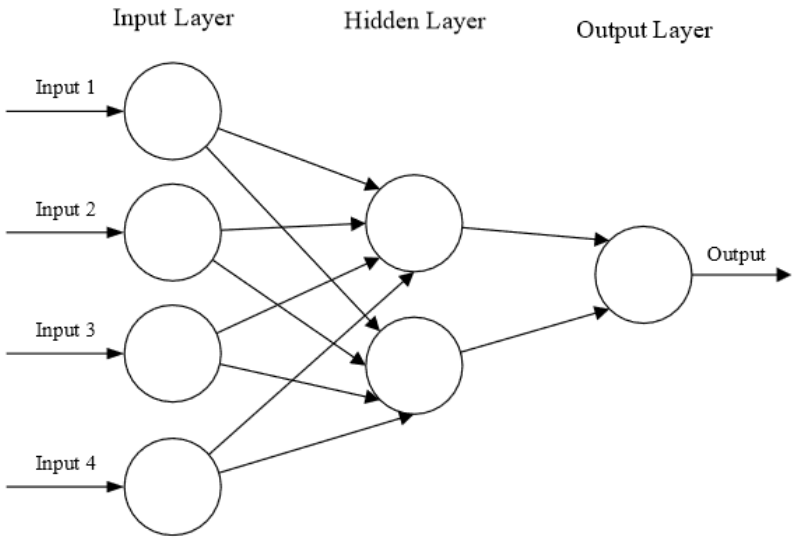
- Use computer vision to differentiate different depths then use reference measurements to calculate distances between points in images.
- Train a model to predict distances:
    - Train on wide data set (e.g. self-taken images, procedurally-generated models, etc.) to create a convolutional neural network that will be able to find distances between objects and cameras simply by passing image into trained model.
- Train a model to predict camera parameter (e.g. focal length, angle of view, etc.) to use with mathematical algorithm for distance estimation:
    - A purely mathematical approach was attempted by Dr. Giroux and colleagues, but this was found to be extremely difficult. As such, it was surmised that an approach combining both mathematics and CV may be more fruitful.
    - Train a convolutional neural network to predict the camera parameter (e.g. focal length, angle of view, etc.).
    - Use a mathematical formula to find the distance of objects from cameras as well as from each other based on calculated camera parameters (formula found in “Distance measuring based on stereoscopic pictures” by Mrovlje et al [4]).

After initial research, it was found that the third of these solutions (training a convolutional neural network on images with known measurements of a particular parameter (e.g. focal length, angle of view, etc.) to estimate said parameter in a photograph in which it is unknown) was the most feasible approach and further research was done to determine the technologies (e.g. programs, dependencies, materials, etc.) required to achieve this.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are a form of artificial neural networks (ANNs), a highly useful computational model in the field of machine learning. As their names suggest, these algorithms are heavily inspired by biological nervous systems. ANNs are formed by numerous interconnected computational nodes (neurons) that work together to learn from an input (e.g. an image) to generate a final output. A user would load input into an input layer of neurons that feed into hidden layers which would then, in turn, make decisions based on prior layers to improve the final output, a process known as learning (multiple hidden layers is referred to deep learning). Supervised learning is learning with pre-labeled inputs and one or more designated outputs with the goal being to minimize the error of calculating known outputs. Unsupervised learning does not have any labeled inputs and the general goal is to modify some associated cost function [5].

Three Layered Feed-Forward ANN [5]



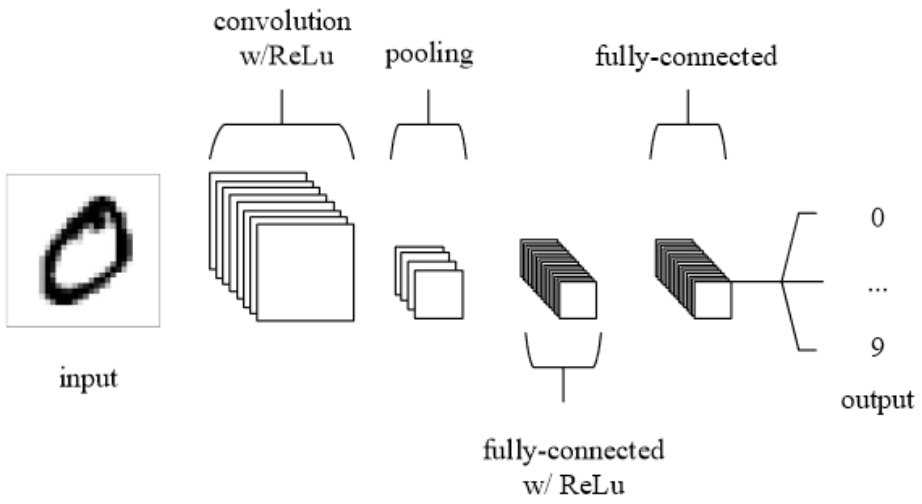
CNNs are like most ANNs in that they too are comprised of neurons that receive input, perform operations (such as scalar multiplications), and self-optimize through learning. The major difference though is that CNNs are generally used for image pattern recognition. The structure of a CNN can be divided into five main layers as seen in the figure below:

1. The input layer: Stores the input image's pixel values
2. The convolutional layer: Determines the output of the neurons connected to the input layer by calculating a scalar product between the weights of said

neurons and the input's volume. A rectified linear unit (ReLu) adds an activation function to the activation output of the previous layer.

3. The pooling layer: Performs down-sampling on the input thereby reducing its parameters within that activation.
4. The fully-connected layers: Produces class scores from the activations for classification (can use ReLu to improve performance).
5. The output layer [5].

CNN Architecture [5]



Based on this research and the desired goal of the project, a supervised learning approach involving the training and testing of a CNN on images with known focal lengths to then be used to predict the focal lengths of images with unknown focal lengths originally seemed the most viable route to determine these required parameters.

It was eventually decided during the implementation phase, that angle of view which is a function of focal length and sensor width (Angle of view (in degrees) = 2 Arctan(sensor width / (2 X focal length)) * (180/π)) would be a better parameter to train and predict so this was done.

DeepFocal: A CNN Method for Focal Length Estimation

There were existing methods for determining the focal length of an image however, a majority required specific geometric calibration objects to be in the field of view of the image. A method using a CNN was tested and proposed by a team at the University of Kentucky to estimate the focal length of a given photograph. Their method directly estimates the focal length from raw pixels of an input image using a deep convolutional neural network without the need of calibration objects.

The team at the University of Kentucky utilized one of the most commonly used CNN architectures at the time known as AlexNet[1]. AlexNet was originally created for multi-class object classification and has been adapted to other tasks such as image style recognition and scene characterization.

The goal of the researchers was to estimate the horizontal field of view, H_θ, which has a one-to-one mapping with focal length:

$H_\theta = 2tan^{-1}(w/2f)$ where w is a given image width and f is the focal length [6]

Mathematically, if they could estimate H_θ and w was known, then f could be easily calculated from this equation.

Their method to achieve this was modifying AlexNet to estimate the field of view in an input image. They saw this as a regression problem in which H_θ was the label. In order for this to work, the Alexnet architecture which was comprised of five convolutional layers and three fully-connected layers had to be modified such that the final fully-connected layer had a single output node corresponding to H_θ. They named this modified Alexnet CNN DeepFocal. Their networks were trained and implemented using an open-source deep learning framework known as Caffe.

Instead of training their network from the beginning, they decided to use transfer learning in which the learned features from a previously trained base network were transferred to a newer target network that will be retrained for a new task. This meant that the first n layers of the target network were initialized with the weights from the first n layers of the

[1] AlexNet is a large, deep convolutional neural network which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton of the University of Toronto in Canada and named after the first researcher, Alex [7].

base network while the weights of the remaining layers were randomly initialized. The network was then trained for the new task of determining H_θ using the target dataset.

The accuracy of the resulting model was found to be impacted by the aspect ratio of the inputted image. Running a portrait image using a portrait only trained model or running a landscape image in a landscape only trained model was more accurate than running an image through a combined trained model (portrait and landscape) [6].

After reading through the paper, the DeepFocal algorithm had a lot of promise with respect to being able to estimate the focal length parameter and served as a basis for moving forward. Further research had to be done on Caffe to determine how it is used for deep learning in vision as opposed to other well-known machine learning libraries such as PyTorch, TensorFlow, and Keras.

DEEP LEARNING FRAMEWORKS

Caffe

Caffe is a machine vision library with a Python API that ported MATLAB's implementation of fast convolutional nets to C and C++. It is great for feedforward networks, image processing, and training models without code. However, it is not good for recurrent networks, slower for larger networks, and has no commercial support. In comparison to a lot of other frameworks, Caffe has been established for quite a while and is still in moderate use to this day (as seen by its use in numerous repositories on the web). Despite this though, official support for Caffe has come to a halt. Caffe itself is also only officially supported on Ubuntu, Red Hat, and OS X. While there exist community-based versions of Caffe for Windows, these are (according to Caffe's GitHub page) "experimental." Our group did reach out to the DeepFocal team and were told that while their method which used Caffe did work for their particular problem, it might not work exactly for our problem (See Appendix). They also stated that since their work was done in 2014, the framework at the time might be dated by today's standards and encouraged us to look into other frameworks in addition to Caffe [8].

PyTorch

Caffe went on to have a second updated version called Caffe2. However, Caffe2 was merged into another framework developed by Facebook known as PyTorch. PyTorch offers dynamic computation graphs that allow users to process variable-length inputs and outputs (useful for recurrent neural networks). PyTorch has a low-level API focused on work with array expressions. However, it has gained a lot of popularity in the last few years in the field of academic research, especially when it comes to developing and optimizing deep learning frameworks with a lot of custom parameters. Since it is written in C with a Python API, it is very fast in comparison to other modern deep learning frameworks. It also has great debugging capabilities. However, its architecture is relatively complex, and it can be challenging to read at first. As opposed to Caffe which is normally used for handling small datasets quickly, PyTorch is used for models and larger datasets that require faster execution. Since it is newer, it might not have as many coding examples compared to Caffe. However, PyTorch's speed, use of Python, and compatibility with other OS's such as Windows make it a stronger contender for a machine learning framework [9].

TensorFlow

TensorFlow, developed by Google, is one of the more well-known deep learning frameworks today. Like Caffe and PyTorch it is also written with a Python API over a C/C++ engine, thus making it faster in comparison to other frameworks. Unlike PyTorch, TensorFlow supports both high and low APIs. However, like PyTorch, it is still pretty challenging to use at first. It is also much more difficult to perform debugging. Despite this, it is still capable of being used for high-performance models and large datasets where speed matters. Because of its popularity and its compatibility with the major modern OS's, there exist quite a lot of online repositories that utilize TensorFlow and it is already heavily used in computer vision and object detection [9].

Keras

Keras is a deep learning library written purely in Python that sits on top of TensorFlow. Keras is a high-level API that has gained recent popularity for its easy use and syntax, and the ability to facilitate fast development. Since it sits on top of another framework, Keras is slow in comparison to other frameworks. Its architecture is much simpler and more concise though, leading to less need for debugging. Because of its overhead, Keras is suitable primarily for smaller datasets. It is compatible with all OS's that TensorFlow is compatible with. Based on its pros, Keras would be useful to pick up and use on a small dataset but if this project involves a larger dataset, then it might take a while to set up and train. Moreover, Keras is relatively newer though since it has gained a lot of popularity due to its ease of use, there are probably a few tutorials that demonstrate how to use it for computer vision problems [9].

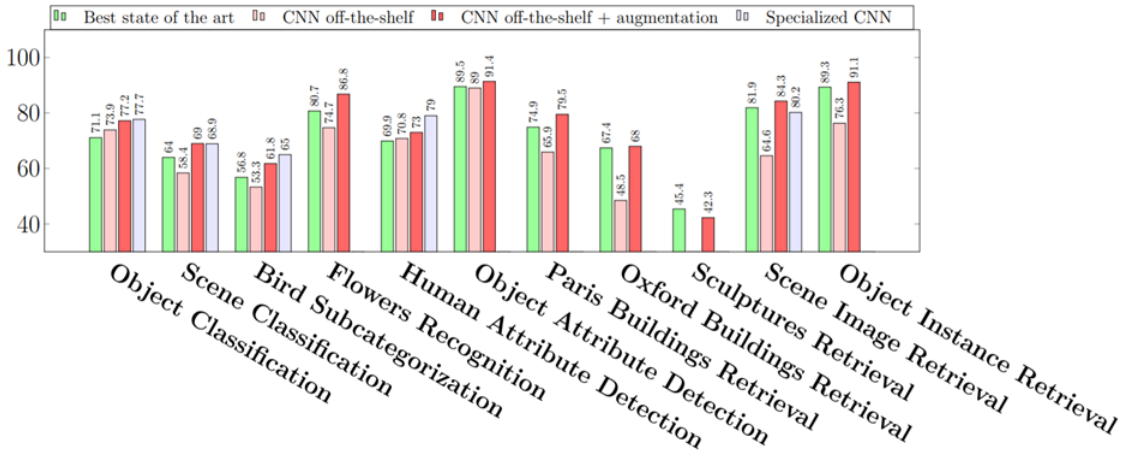
TRADITIONAL LEARNING VS. TRANSFER LEARNING

TensorFlow with a Keras API was the framework used to develop the model as explained at the start of the Design Summary below. However, prior to designing the model, it had to be considered if creating it purely from scratch in a traditional machine learning process would be viable or if using a previously-trained model adapted for the problem at hand would be fruitful. The latter is a process known as transfer learning which was mentioned in the DeepFocal paper above. One major benefit of transfer learning is it overcomes a lack of data by building off a model that has already been itself trained on a vast amount of labeled data. Moreover, features and weights from the previous model can be leveraged when training the newer model.

There are two major strategies when it comes to deep transfer learning. The first is utilizing off-the-shelf pre-trained models as feature extractors. As explained earlier, deep learning models contain layers that each learn different features. These layers connect to a final layer (usually a fully connected layer) to get a final output. This architecture can be modified keeping the original layers the same but removing the final layer as a fixed feature extractor for another task.

As an example, AlexNet, when used without its final layer, allows for the transformation of images from a new domain task into a 4096-dimensional vector thus enabling the extraction of new features from this new task by building off knowledge from the original model. This was the basis for how the DeepFocal model was created from AlexNet [10].

Performance of Task-Focused Models vs. Pre-Trained Models [10]
The figure shows that features from the pre-trained models outperform features from task-focused models.



The second major strategy is fine-tuning off-the-shelf pre-trained models. In this method, not only is the final layer replaced but some of the prior layers feeding into it are retrained. While more involved, it is achievable as deep neural networks have configurable architectures. Generally, in these networks, the initial layers recognize generic features while the later ones capture specific features, applicable to the desired task. An example is seen in CNNs used for facial recognition in which the first hidden layers may recognize edges, the middle hidden layers may recognize ears or eyes, and the final hidden layers may recognize the entire face. The idea is that certain layers would be frozen (have fixed weights) while others would be fine-tuned for the new task at hand [10].

There exist many open-source pre-trained deep learning models available for import in Keras including VGG-16, VGG-19, Inception V3, and ResNet. AlexNet can be seen as the modern groundbreaking model that inspired the development of all of these other ones. AlexNet was developed in 2012 though so it can be seen as a little dated (at the time, TensorFlow and Keras were not available). Moreover, it is not available for import in Keras. While DeepFocal was built off of a variant of it (CaffeNet), since it is not open-source and since training it from scratch would require a lot of time and additional resources (GPUs), it was determined that it would be more feasible to look into using one of the other pre-trained models and monitor the accuracy/loss of the custom model to see which one would be best to predict focal length.

Pre-Trained Models Available in Keras [11]

Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
ResNeXt50	96 MB	0.777	0.938	25,097,128	-
ResNeXt101	170 MB	0.787	0.943	44,315,560	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

CNN CASE STUDIES

LeNet

LeNet was one of the first successful applications of a CNN. Developed by Yann LeCun in the 1990s, it was used to read numbers (e.g. zip codes, digits, etc.). It went on to inspire many other CNNs.

AlexNet

As stated previously, AlexNet, developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton, can be seen as the first major CNN for computer vision analysis. It was submitted to the ImageNet Large Scale Visual Recognition Competition (ILSVRC) challenge in 2012 where it won and significantly outperformed the rest of the competition (It had a top-5 error of 16% compared to the second best being a 26%). AlexNet had a similar architecture to LeNet. However, it was much deeper and was one of the first to feature convolutional layers stacked on top of one another. AlexNet inspired many newer models that were able to build off of its architecture by modifying its layers and parameters to decrease error rates. It is possible to replicate AlexNet in more modern frameworks such as TensorFlow and Keras, but the time spent doing that could be saved on just importing and fine-tuning one of its successors that are already included with Keras.

GoogLeNet/Inception

GoogLeNet, a CNN developed by a team at Google (and whose name is obviously a reference to the original LeNet CNN), was the winner of ILSVRC 2014. Its main feature was the development of an Inception Module which reduced the number of parameters in the network (4M compared to AlexNet's 60M). In addition, GoogLeNet uses average pooling instead of fully-connected layers on the top of the convolutional network which eliminates a lot of superfluous parameters. GoogLeNet has a lot of follow up versions one of which (InceptionV3) is included in Keras as an import. Inception-V3 has a size of 92MB with a top-1 and top-5 accuracy of 0.779 and 0.937, respectively.

VGGNet

VGGNet, a network developed by Karen Simonyan and Andrew Zisserman, was the runner-up at ILSVRC 2014. VGGNet showed that the depth of a network is a main component for good performance. The final version at the time contained 16 convolutional/fully-connected layers and features a homogenous architecture that performs 3 x 3 convolutions and 2 x 2 pooling from beginning to end. There are now two pre-trained versions available for use in Keras: VGG16 (16 layers deep; 528 MB; top-1 accuracy 0.713; top-5 accuracy 0.901) and VGG19 (19 layers deep; 549 MB; top-1 accuracy 0.713; top-5 accuracy 0.900). A con of VGGNet is that it is much more expensive to evaluate and uses a lot of memory and parameters compared to other networks. It was found that most of these parameters were in the first fully connected layer and that these could be removed without any noticeable downgrade in performance, thus also removing any superfluous parameters.

ResNet

Residual Network (ResNet), a network developed by Kaiming He, was the winner of ILSVRC 2015. Its main features are skip connections and a high use of batch normalizations. Unlike the other CNNs, ResNet does not have fully connected layers at the end. Because of its size and top-5 accuracy (>0.9), ResNet is considered a front-runner when developing and training models. Six versions of ResNet are available for use in Keras (3 ResNet and 3 ResNet-V2) with sizes ranging from 98 to 232 MB and top-5 accuracies all above 0.9 [12].

ResNeXt

ResNeXt, a network developed by UC San Diego and Facebook AI Research (FAIR), was the runner-up for ILSVRC 2016. The model is described as adding an additional “cardinality” dimension on top of ResNet which controls the number of more complex transformations. ResNeXt has two versions both of which have higher top-5 accuracies then ResNet but are comparable in power to ResNet-V2 [13].

Xception

Xception, a network developed by Google in 2017, stands for the Extreme version of Inception. A modified depthwise separable convolution, Xception was designed to be better than Inception-V3 for both ImageNet ILSVRC and JFT datasets. Xception was found to outperform VGG-16, ResNet-152, and Inception-V3 in terms of both top-1 (0.79) and top-5 (0.945) accuracy with a lower error rate as well. Like the others, it is available for import as a pre-trained model in Keras and it has the added benefit of being quite small (88 MB) [14].

Inception-ResNet-V2

Inception-ResNet-V2 is a model consisting of Inception-V4 (the successor to Inception-V3) merged with a ResNet. This combination is able to outperform the similar base Inception network without residual connections. Training is faster compared to the regular Inception-V4 and it has one of the highest top-5 accuracy (0.953) among all of the pre-trained models available in Keras. Despite this though, it is still a relatively large model (215 MB) [15].

MobileNet

MobileNet is another network developed by Google in 2017. The model's size and complexity are reduced using depthwise separable convolution. This makes it useful for mobile and embedded vision applications. The model's two main features are its smaller size due to its fewer parameters and its smaller complexity due to the fewer number of operations it performs. There exist two versions available for import as pre-trained models in Keras: MobileNet and MobileNetV2. While both have relatively lower top-5 accuracies when compared to the other models (0.895 for MobileNet and 0.901 for MobileNetV2), they both have the smallest size out of all of the other pre-trained models (16 MB and 14 MB, respectively). If it is found that the other models are too large to be in a client-side web application, MobileNetV2 may end up being the front-runner for the focal length problem [16].

DenseNet

DenseNet (Dense Convolutional Network) is a network jointly invented by Cornwell University, Tsinghua University, and FAIR and the paper for it in 2017 CVPR won the Best Paper Award. DenseNet has high accuracy and as its name suggests, dense connections. DenseNet's advantages include a strong gradient flow, parameter/computational efficiency, more diversified features, and maintenance of low complexity features. There are three versions available as pre-trained models in Keras, all with relatively small sizes and high top-5 accuracies (>0.92) [17].

NASNet

NASNet (Neural Architecture Search Network), a network developed by Google Brain, was published in 2018 CVPR. In NASNet, while the overall architecture of the model is predefined, the blocks/cells in the model are not but are instead searched for in a small dataset and transferred to a larger dataset by reinforcement learning. To better explain, the best convolutional layer/cell is searched for on the CIFAR-10 dataset (Canadian Institute For Advanced Research) then applied to the ImageNet by stacking more copies of said cell. An additional regularization technique also helps improve the generalization of the model. NASNet achieves great results with smaller datasets and lower complexity. It is available in two versions in Keras: NASNetLarge which is 343 MB and has the highest top-5 accuracies of all the current models (0.96) and NASNetMobile which is a measly 23 MB but still has a relatively high top-5 accuracy (0.919). Given the size and accuracy and the necessary requirements. NASNetMobile may end up being the best mobile-compatible front-runner out of all the Keras models [18].

Design Summary

Since the machine learning framework is used in a web-based application, TensorFlow with Keras was chosen as the framework as TensorFlow.js allows for networks to be ported to JavaScript for use in a web environment and despite its overhead, Keras greatly simplifies the development process. Moreover, Keras comes with many pre-trained models which alleviated the need to train a model from scratch which in itself would require a lot of time and resources (e.g. wealth of training data and storage, GPUs, etc.). The goal was to adapt and fine-tune one of these models for the parameter estimation problem by modifying the final output layer for a new desired variable: the estimated focal length at first and then the angle of view in the final design.

Transfer learning from one of the pre-trained Keras models was essential. There are many included models that could be used to attempt the focal length estimation problem. Indeed, quite a few have top-5 accuracies over 0.900 which would give a high level of confidence. Despite this, some of the included models are quite large (100 to 500 MB), have parameters ranging from tens to hundreds of millions, and are quite deep. While it is found that the larger models tend to have the higher top-5 accuracies (>0.94), a smaller, more mobile-friendly model could achieve a similar though slightly less result (~0.91).

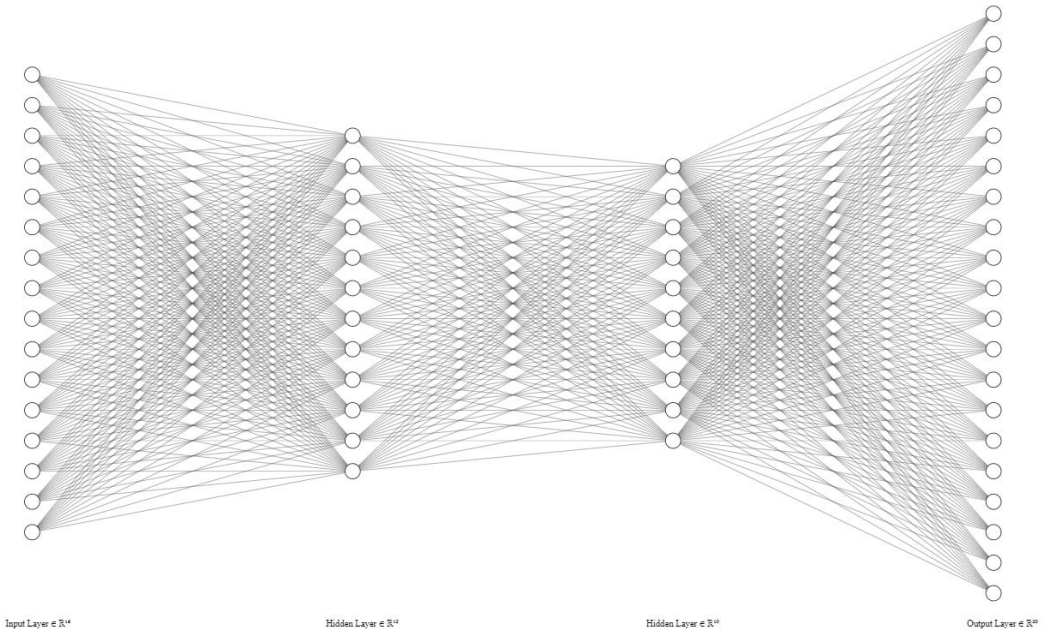
That being said, given the resources at hand, it was initially planned to train the model on MobileNetV2 (14 MB and 88 layers with 0.901 top-5 accuracy), NASNetMobile (23 MB and 0 layers with 0.919 top-5 accuracy), and Xception (88 MB and 126 layers with 0.945 top-5 accuracy). Hopefully, the models trained, validated, and tested would be sufficient enough for the focal length estimation problem. If they were not, it would be seen if doing transfer learning with larger, more powerful pre-trained models are possible such as either InceptionResNetV2 (215 MB and 572 layers with 0.953 top-5 accuracy) or NASNetLarge (343 MB and 0 layers with 0.960 top-5 accuracy). If more resources were required, than the high-powered computers at the University of Central Florida’s Senior Design Lab could be sufficient but access and permission would need to be obtained first. The chosen pre-trained network ended up being the Xception model as this garnered the highest accuracies compared to the others.

The problem was treated as a standard image classification problem in which the final output layer of the designed network would consist of a finite number of nodes representing the potential parameters (e.g. 20.0 mm or 30.0 mm for the original focal length model and 25 or 100 degrees for the updated angle of view model). Similar to other classification problems, in the end, the node with the highest calculated probability

represented the parameter from the input image (e.g. 99% on the 25 degree node would mean the input image's angle of view is, with a certain level of confidence, 25 degrees).

Simplified Architecture of Target CNN

The input and initial hidden layers of the imported Keras model were imported as is. However, the final few hidden layers (corresponding to the second to last layer in this image) were unfrozen to fine-tune. In addition, the final layer (corresponding to the 20 final nodes in this image) were stripped and modified such that there was one node for the most common angle of view measurements in the training dataset.



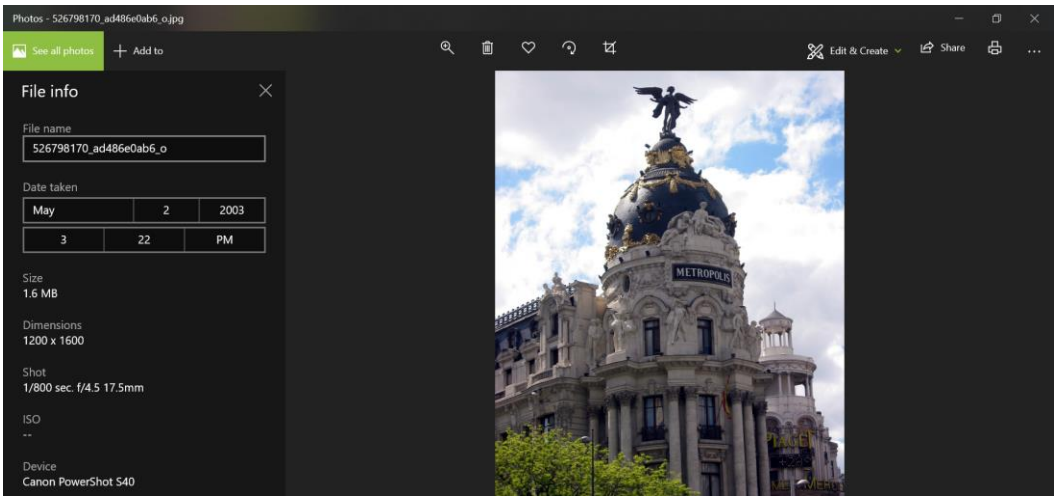
The model, was written in Python 3 and trained, validated, and tested on images with known focal lengths or angle of views. Once it was able to predict these with a certain level of accuracy in images where they are already known, it was then used to determine the angle of view in the old Civil War photographs. Once it was able to do this with a high level of confidence, the model was be ported to JavaScript using the TensorFlow.js module to be implemented in the client-side portion of the website.

Design Content

PREPARING THE DATASET

The images needed to train, test, and validate the model that was designed in this section had to have both focal length and sensor widths available as information that could be extracted from their Exif data.

Madrid Metropolis Image Acquired from 1DSfM Dataset
In this figure, the resolution (1200 x 1600) and focal length (17.5 mm) are available for the source image and can be extracted as metadata.



Once a sizeable dataset of images with Exif data was acquired (explained earlier in the Training Data section by Thomas Whitaker), a script was written to easily parse the focal lengths and sensor widths needed to calculate angle of views for training. Fortunately, there exist many different Python libraries to extract Exif data from images (exif, ExifRead, piexif, etc.). The extracted focal lengths will then be used to label each image for the model.

The ratios of images used to test, train, and validate the model were the standard 60% to train, 10% to validate, and 30% to test (e.g. if there were 10000 images, then 6000 would be used for training, 1000 would be used for validation, and 3000 would be used for testing).

Depending on the specific pre-trained model, it was necessary to preprocess the input images prior to training. As an example, some pre-trained Keras models have precise

input image resolution sizes (e.g. Xception which was the chosen model required input images be 299 x 299). Fortunately, Sci-kit learn, another Python library had a preprocessing module.

DESIGNING THE CNN MODEL

As stated before, multiple mobile-friendly Keras models were used at first to create the best model (Xception ended up being the). This chosen model was imported and the final few layers of it were unfrozen such that they could be retrained for the focal length classification problem. From here, additional layers were added leading into a final output layer with one node for every angle of view. Once the model was properly prepared, the training and validation data were fed into it to go through a set number of epochs (30). At the end of this, the training and validation accuracy were examined to ensure they were of a tolerable level.

EVALUATING THE MODEL ON TESTING DATA

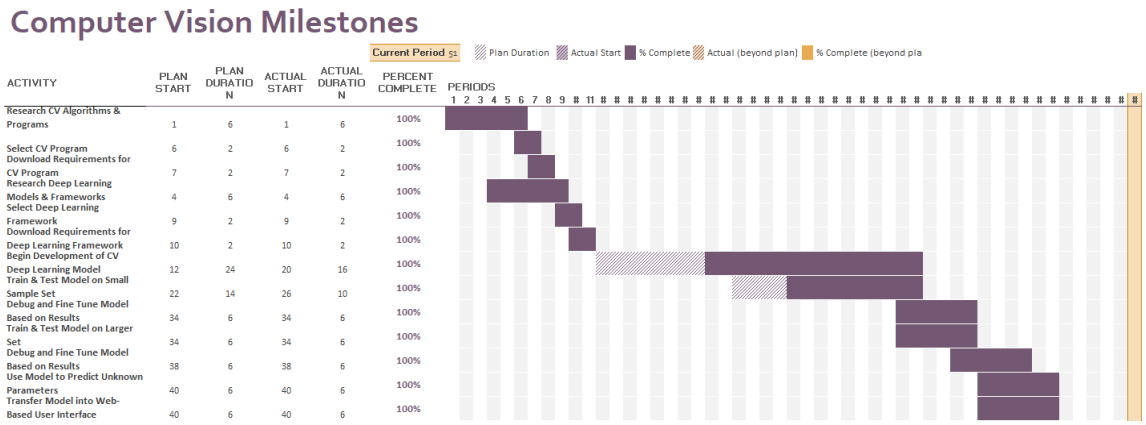
Once the model's training and validation accuracy was of a tolerable level, it was tested on the images with known focal lengths and evaluated using a custom model_evaluation_utils library in Python.

USING THE MODEL FOR PREDICTIONS

Once the Keras model was properly trained, validated, and tested, it was used to predict focal lengths in images in which they are unknown by calling upon the `predict` function in Keras.

PORTING THE MODEL

Once the Keras model was complete, it was saved as an HDF5 file and an API was written to run it in the website's backend.



Development Timeline

Research CV Algorithms & Programs

May 22nd - June 11th
Research different computer vision software and programs used for object detection, size determination, and distance measurements.

Select CV Program

June 7th - June 13th
Narrow down on a specific program based on its capabilities (e.g. speed, etc.) and OS/web compatibilities.

Download Requirements for CV Program

June 12th - June 18th
Once CV program is selected, download its software and dependencies and begin experimenting with it.

Research Deep Learning Models and Frameworks

May 31st - June 20th
Research different machine learning models and frameworks used in computer vision for measurement predictions.

Select Deep Learning Framework

June 18th - June 24th
Narrow down on a specific framework based on its capabilities (e.g. data limit, speed, etc.) and OS/web compatibilities.

Download Requirements for Deep Learning Framework

June 21st - June 27th
Once framework is selected, download all necessary software and dependencies and begin experimenting with it.

Begin Development of CV Deep Learning Model

June 28th - August 8th
Research and begin developing CV model to do parameter estimation.

Train and Test Model on Small Sample Set

August 4th - August 17th
Feed initial training data into the model then use to predict known parameter measurements.

Debug and Fine Tune Model Based on Results

August 11th - September 1st
Debug and fine-tune initial model to get better results.

Train and Test Model on Larger Set

September 2nd - September 15th
Feed final training data into model then use it to predict known focal lengths with a higher level of confidence.

Debug and Fine Tune Model Based on Results

September 12th - September 25th
Make sure the model is properly debugged and fine-tuned to get accurate results.

Use Model to Predict Unknown Parameters

September 26th - October 2nd
Use model to obtain the unknown parameter calculations which will be used in the mathematical portion.

Transfer Model Into Web-Based Interface

October 3rd - October 16th
Once the model is completed, transition it into the web-based application.

Build Plan

1. Preprocess data.
   The acquired images are preprocessed prior to use for training, validating, and testing the model. They are varied and sectioned off such that 60% is used for training, 10% is used for validating, and 30% is used for testing. The images are also resized to 299x299 for the chosen pre-trained model Xception, using Sci-kit learn.
2. Design model.
   A pre-trained Keras model is modified such that its final few layers are unfrozen to be retrained and its final layer will consist of one node per each possible focal length in mm. The images with known focal lengths are fed into the model which is trained and validated over 30 epochs.
3. Test model.
   Once the model has sufficient training and validation accuracy, it is tested on the test set, using the custom model_evaluation_utils library.
4. Predict focal length using model.
   Once the model has a high testing accuracy, it is used to predict the focal lengths in images where it is unknown using the `predict` function in Keras.
5. Port model to web application.
   The model is saved and ported to the web application as explained above under the PORTING THE MODEL section using the official documentation available under the readme in TensorFlow.js official GitHub page.

Prototype Plan

1. Ensure proper image specifications.
   The pre-trained models in Keras only accept square images of size 299 x 299 pixels. While Sci-kit learn's preprocessing module can handle image resizing, care is taken such that the original image quality is retained (e.g. no stretching vertically/horizontally, etc.).
2. Fine-tune model based on training and validation accuracy.
   The accuracy of the initial models may be quite low, so additional fine-tuning of the final layers and variation of the input images is necessary.
3. Fine-tune model based on testing accuracy.
   The accuracy of the model when calling upon the `predict` function may have had low accuracy initially so again, care is taken to ensure the test data was representative of the training/validation data, and if it is, the model had to be fine-tuned.
4. Ensure model compatibility with front-end.
   The TensorFlow model, written in Python through Keras is compatible with the MERN stack used to develop the web application. It is necessary for some additional debugging to ensure it is adapted into the front-end and works as intended.

Test Plan

Integration Tests

1. Test to ensure model is properly integrated into front-end
2. Test to ensure model works as intended within front-end

Evaluation Plan

Evaluating output

The output of the model for the desired historical image is examined to determine its feasibility. If the estimated angle of view is too low (indicating a zoomed out image) or too high (indicating a zoomed-in image), then it needs to be redone. If however, the angle of view of the image is the same as that in other similarly-scaled images, then it is most likely the correct one and can be used for other calculations.

Ultimately, upon completing the project, it was found that since the model was trained on outdoor images since it was designed to work on the cemetery photograph which was an outdoor image, it fared better on this and other similar scenes. However, in more local settings (e.g. indoors, etc.) the model did not do as well. After discussing this with the sponsor, Dr. Giroux, it was found that for the sake of this problem, this would be satisfactory.

Technologies and Equipment

TOOLS

Google Colab

Use: Machine learning research tool in a Jupyter notebook environment
Features:

- Built off Jupyter notebook so comes with all of its features (e.g. Python 3, live environment, etc.)
- Supports in-line Linux commands
- Allows for cloud-based development with CPU, TPU, or GPU

Google Colab was chosen as the primary tool of development for the computer vision/machine learning portion as there are already numerous online examples of notebooks used for computer vision neural network development. In addition, a few of the team members already have experience with Jupyter Notebook (which Colab is based off of) and its live environment is very useful for rapid prototyping. Moreover, code developed in Colab can be exported to PY files for use in other programs or shared with others. Finally, Colab is optimized for training deep learning models with its GPU support.

LIBRARIES

TensorFlow

Developer: Google Brain Team
Use: Machine Learning Framework
Features:

- Has many different APIs which will be of great use in this project
    - Keras: Supports easy model building
    - TensorFlow.js: Allows for frameworks to be developed for web/mobile compatibility
    - TensorFlow Graphics: Supports deep learning for computer graphics problems (e.g. object classification, etc.)
- Has thorough documentation for setup and required dependencies (e.g. required libraries and drivers for GPU support, etc.)

- Supports Python, JavaScript, etc.

TensorFlow is one of the most well-known and developed machine learning frameworks out there. There already exist numerous tutorials and notebooks online with examples on how to do image classification using it in Python. In addition, its TensorFlow.js library will be essential when porting the model from the development environment (written primarily in Python) to the client-side web environment (written primarily with JavaScript). Finally, some of the group members already have some experience using TensorFlow.

Keras

Author: François Chollet
Use: Neural Network Library
Features:

- Runs on top of TensorFlow
- Allows for easy model building through its user-friendliness and modularity
- Runs on CPU and GPU
- Written in Python

Since Keras sits on top of TensorFlow and thus introduces overhead, it was initially debated whether or not to use it in the development of the model. However, after further research and seeing how it greatly improves code readability as well as development, it was decided that the benefits outweigh the risks.

Scikit-learn

Author: David Cournapeau
Use: Machine Learning Library
Features:

- Offers many classification, regression, and clustering algorithms
- Offers many pre-processing functions
- Written in Python with core methods written in Cython for performance

Sci-kit learn is one of the original machine learning libraries offered in Python. Its sufficient documentation makes it both easy to pick up and use. Moreover, while it does offer many algorithms for ML analysis, another major feature is its pre-processing module that allows users to prepare data for use not only with its own libraries but other

libraries as well (e.g. TensorFlow, Keras, etc.). This will be of great use as a lot of the images need to be resized and reformatted prior to being fed into the model.

MATHEMATICAL ANALYSIS

To calculate the distance between two points in an image of a stereographic pair, mathematical analysis is required to determine these values. Epipolar geometry, or the geometry of stereo vision, provides the basis for converting projections into different coordinates.

Technical Objectives

- Investigate various mathematical computer vision methods to calculate distances in images, both monocular and binocular.
- Determine essential parameters to calculate distances using mathematical computer vision methods.
- Implement chosen method.
- Fine-tune, debug, and testing method.

Technical Goals

- Understand how the two images in stereographs work together.
- Learn how to create real-world measurements from stereographs.

Technical Specifications

- Program accepts two images (left and right of a stereograph), two image coordinates made on the left image, focal length, and sensor width.
- Program outputs calculated distances between two inputted image coordinates.

Technical Requirements

- Must be able to accurately estimate measurements with a low margin of error (e.g. 10%).

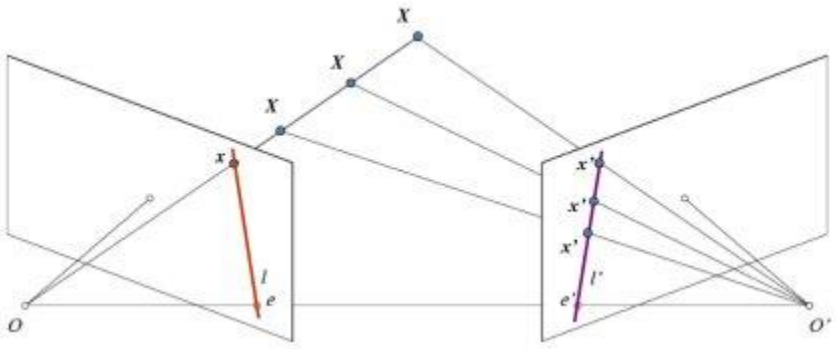
Research and Investigations

Epipolar Geometry

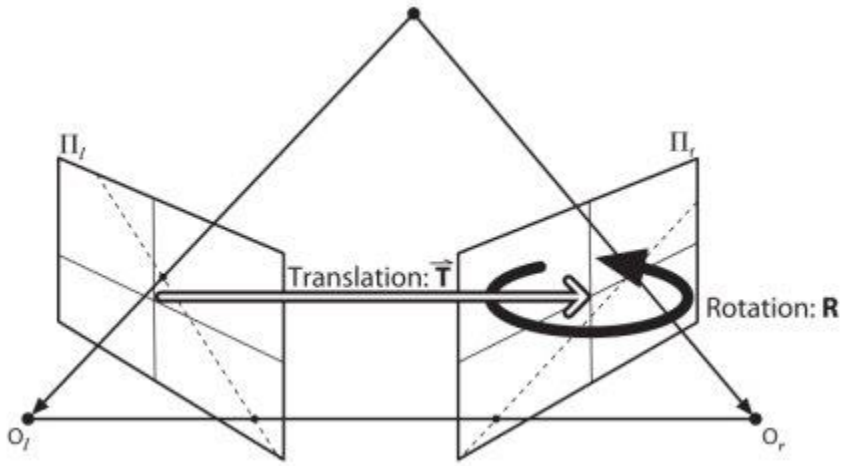
Epipolar geometry is the geometry of stereo vision. When two cameras view a 3D object from different positions, there are a number of geometric relations between the 3D points and the 2D projections. Using epipolar geometry it is possible to reconstruct a 3D scene with the 2D projections in stereographs.

Using only one image it is not possible to convert points in the image to 3D space because it is only on one plane. By adding a second image one can add the third dimension coordinates to the points of the first image [19].

Two images and their point correspondence [19]



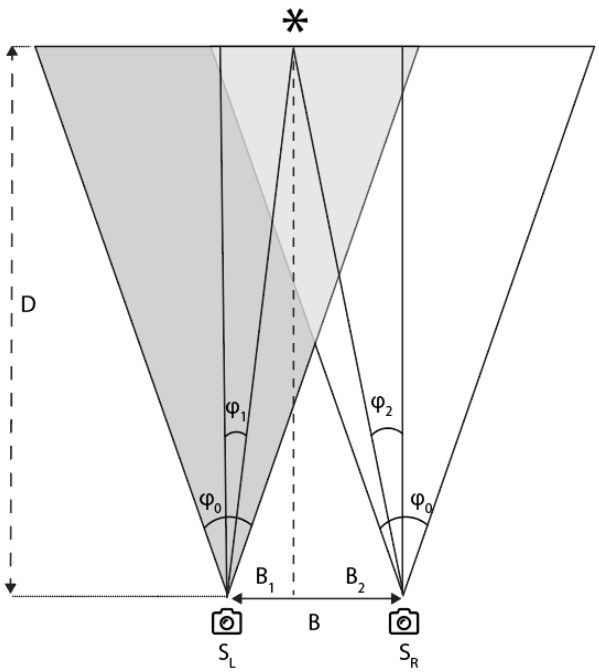
Having two images of the same scene allows one to perform this conversion [19]

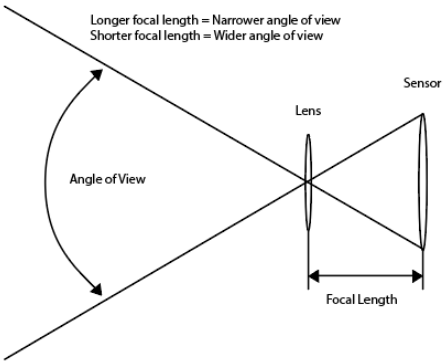


Calculating Depth from Stereoscopic picture

Stereoscopic pictures allow for the calculation of the distance between the camera(s) and the chosen object within the picture. An object's position (distance D) can be calculated by doing some geometrical derivations. Accuracy depends on picture resolution, optical distortions, and distance between cameras [4].

Basic geometry of stereographic cameras, angle of view and focal length [4].





B = Distance between cameras

$B = B_1 + B_2 = D\tan\varphi_1 + D\tan\varphi_2$

D = Distance from center point of cameras to target

$D = B / (\tan\varphi_1 + \tan\varphi_2)$

$x_1 / (x_0 / 2) = \tan\varphi_1 / \tan(\varphi_0 / 2)$

$-x_2 / (x_0 / 2) = \tan\varphi_2 / \tan(\varphi_0 / 2)$

φ_0 = Angle of view of camera

$D = Bx_0 / (2\tan(\varphi_0 / 2)(x_L - x_D)$

$(x_L - x_D)$ = disparity of target between left and right camera

$\varphi_0 = 2\arctan(x_0 / 2 * f)$

f = Focal Length [4]

Calculating Depth using a Single Camera

Utilizing triangle similarity, we can determine the distance from the camera to an object. The dimensions of the object need to be known.

A formula for obtaining the focal length of the camera.

$F = (P \times D) / W$

Where,
F = Focal Length
P = Perceived pixel width of object
D = Distance from camera to object
W = Actual width of object

Reworking the formula, we can get a new formula that determines the distance from the camera to the known object.

$D = (W \times F)/P$ [20]

Calculating depth with a single camera [20]

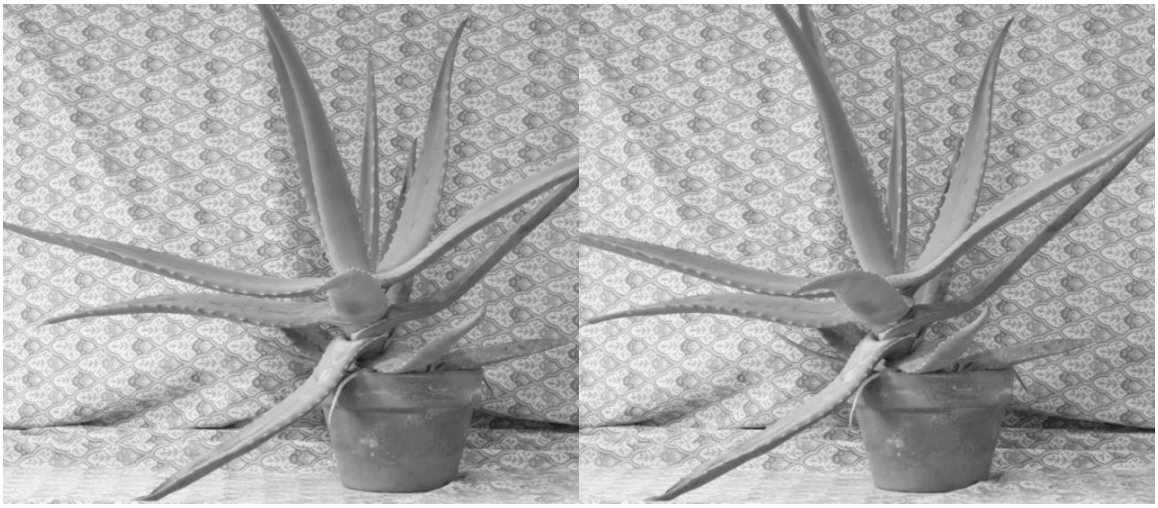
Stereoscopic Reconstruction

OpenCV comes with stereographic functions that allow for feeding in images, tweaking input parameters, and getting a 3d reconstruction of the stereograph.

OpenCV has a class called StereoSGBM. This class implements a modified H. Hirshmuller algorithm. This is a stereo method for Semi-Global Matching (SGM) used to create 3D reconstructions. This is one of the currently top ranked algorithms. The complexity is linear to the number of pixels and the disparity range. The runtime is typically 1 to 2 seconds [21].

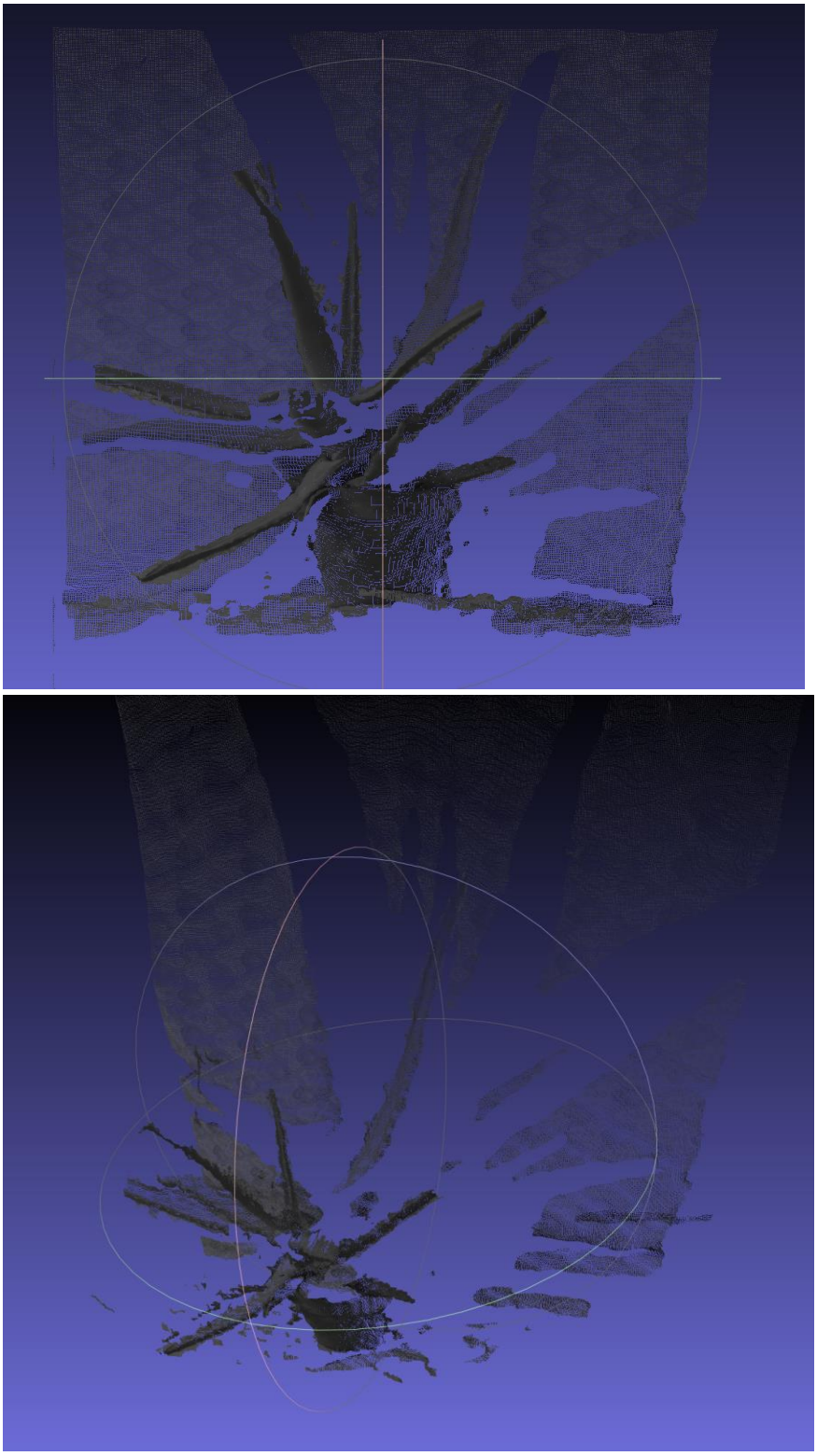
OpenCV uses a modified version of the algorithm. By default it runs in a single pass, meaning it will go only 5 directions instead of the original 8. The algorithm also only matches blocks instead of pixels. Some pre- and post-processing are also included into the algorithm. The class can be set to run similar and more accurate like the original algorithm but will consume a lot of memory.

Stereograph of an aloe plant



Disparity map calculated from the two images

3D reconstruction using depth map and stereograph



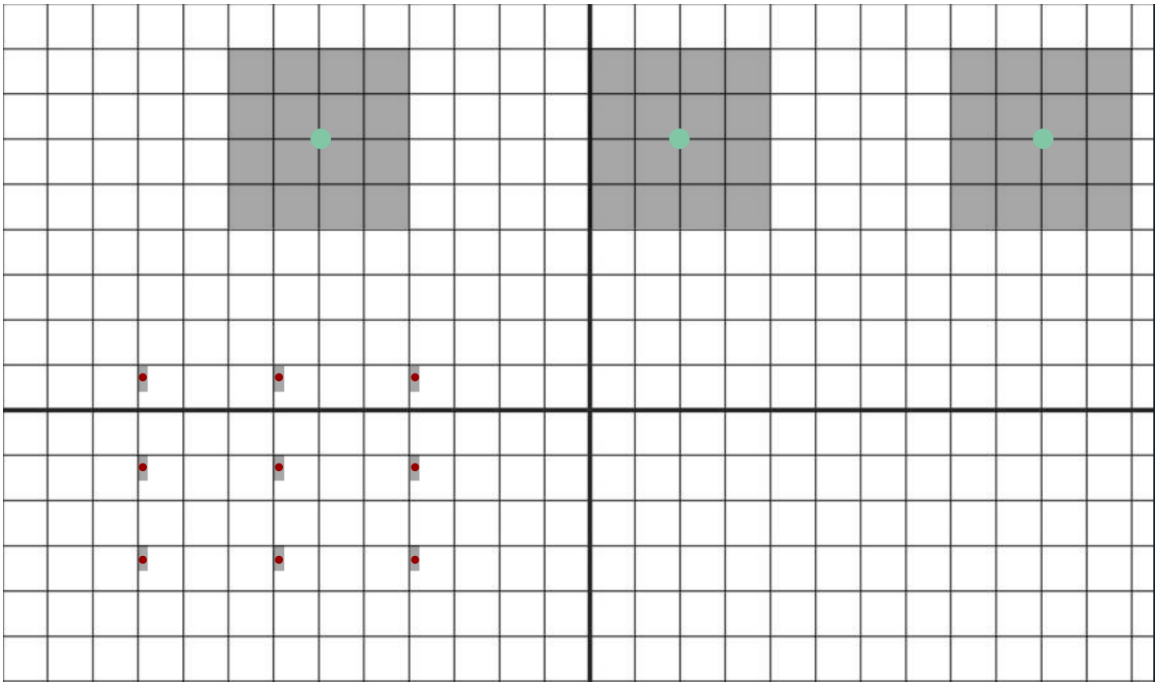
Design Summary

Program receives left and right image of a stereograph pair, 2 points to be measured, and the image angle of view.

Program then converts the image pixel coordinates into world coordinates.

19th Century Left and Right Stereoscopic Photographs Depicting Civil War Soldiers' Marked Graves with Measurements (Provided by Dr. Giroux)

Image points transformed into a 2D coordinate system.

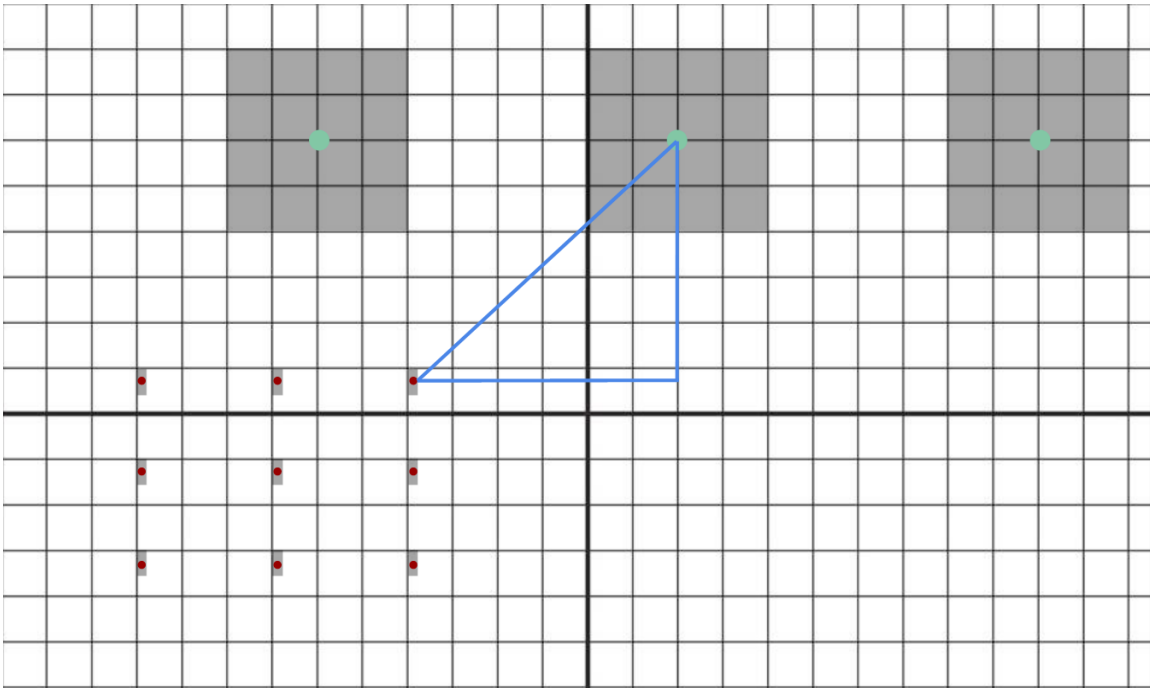


Program then calculates the distance between points using the distance formula.

Distance formula

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Using distance formula on two points



Design Content

There are many different methods to gain the same result. Some methods may be easier but would require data that may not be available.

Input images and preprocess

Images need to be properly processed so algorithms can function properly. Stereo images are:

- Set to the size/dimensions of the smallest of the two images
- Converted to 8-bit single-channel
- Downscaled if resolution is too high and would slow computation

Obtaining a proper disparity map

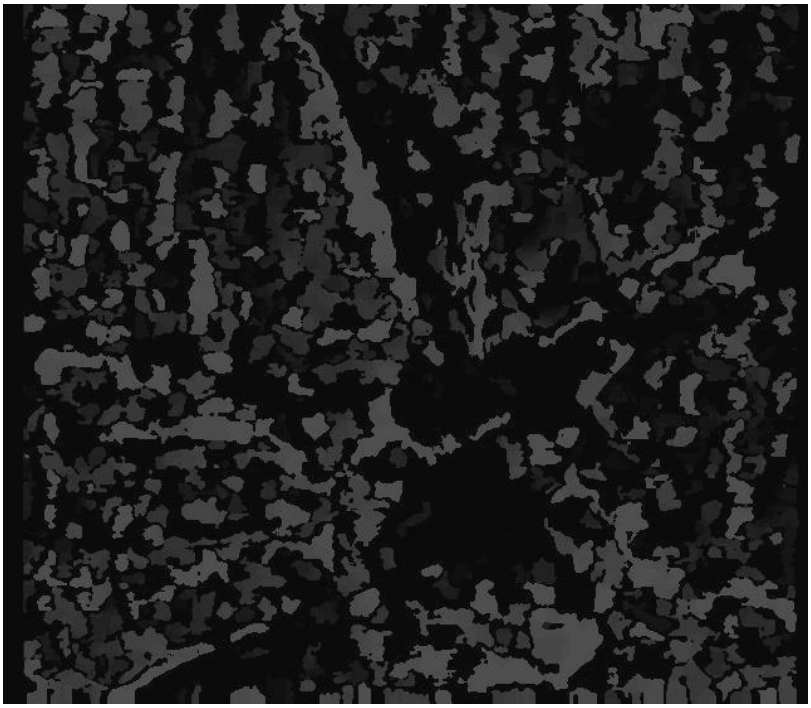
A better disparity map means a better 3D reconstruction. All the methods for creating a disparity map need to be tuned for every pair of stereographic images.

Two solutions for getting a properly tuned disparity map are as follows:

Creating an automatic disparity tuner
Create a function that finds the best parameters for the SBGM class to use to get a disparity map that matches the original image as much as possible.

Allow the user to find the proper SBGM parameters for a good disparity map
The user would be allowed to send in the parameters, along with the images. Multiple maps are outputted, and the user can select the disparity map on the front-end that matches the image as close as possible.

Disparity Map without Tuning



Disparity Map with Tuning

Creating 3D reconstruction

OpenCV's library contains the SBGM class, mentioned previously. This class has a function that converts the stereographs into a 3D point cloud. The function is called `reprojectImageTo3D()`. It takes in a disparity map obtained by running the SBGM's `compute()` method beforehand, this `compute` method takes the left and right image of the stereograph.

Applying scale to 3D reconstruction

The raw 3D reconstruction does not use real world units. The points will need a scale factor to be applied to get correct distances.

Obtaining scale: First need to obtain a real world measurement between points. These are 2 different methods to obtain a real world measurement.

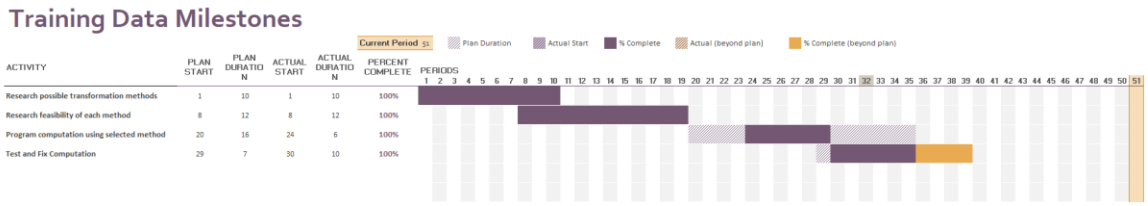
Method 1: Asking the user for a measurement in the stereograph.

This is the most accurate method and would require the front-end to allow the user to input the distance between two points.

Method 2: Using the depth calculation for images formulas to find the distance between two points.
This method is less accurate than method 1. The program in this instance would need to know the angle of view (obtained by machine learning model explained earlier) and the baseline between the two cameras.

Returning calculation

The front-end receives the calculated points from the program and needs them in the appropriate format. This will be the world coordinates for each pixel in the image or the calculated real world distance between the points.



Development Timeline

Research Possible Transformation Methods

May 22nd - June 28th
Research different methods for transforming images into world coordinates.

Research Feasibility of Each Method

June 28th - August 2nd
Compare requirements for each method and select the most accurate method that can be used under our constraints.

Program Computation Using Selected Method

August 16th - September 6th
Write a program that takes in images and parameters and returns the distance between two points given.

Testing 3D Reconstruction and Computation

September 6th - November 11th
Testing program reconstruction. Run on multiple stereographs.

Build Plan

1. Build input and return function.
   This method receives images and parameters and then sends back data in the appropriate format. This data is received by the front-end to use appropriately. At first, this sends back placeholder data, so the frontend can continue to work concurrently.
2. Create preprocessing function.
   Images are set to the same dimensions.
3. Create the SBGM stereo reconstruction object creator.
   The SBGM object creator function receives the images and any relevant parameters. This object can compute the 3D reconstruction points. These are set to output on the return function. At this point, the points are not accurate at all, because the disparity checker is not tuned, and the scale is not added. The user may run the program disparity map tuner that is created in the next step.
4. Create disparity map tuner.
   If a user disparity map tuner is used, the SBGM accepts the correct parameters from the front-end and this is not needed. If the user does not use a disparity map tuner, the SBGM creator runs the automatic disparity map tuner. The automatic tuner runs different parameters in a logical manner to find optimal parameters that match the stereograph pairs.
5. Create scale applier.
   Scale applier applies a scale to the 3D grid that SBGM constructs.

Prototype Plan

1. Build command line program that accepts the core program parameters to test without the use of the frontend.
2. If user disparity map tuner is used, have a simple GUI with sliders to set the best disparity parameters or allow user to select the best disparity map from a set of pre-generated maps.
3. Have a simple GUI that allows selecting points to be measured.
4. Outputs on simple GUI or command line what the computed distance was calculated.

Test Plan

Integration Tests

1. Check if images are properly being received from front-end.
2. Check if parameters are being sent in properly and set to correct types.
3. Check if data is being sent back in proper JSON format.

Evaluation Plan

Evaluating the distance between points calculation

The accuracy of the distance measurement is examined to determine its accuracy at different distances. The program is run on a stereograph with known distances and measurements. The accuracy is evaluated with objects close to the cameras. Then with objects spaced further behind. These calculated measurements are compared to the recorded known distances. The calculated distances should fall within 2 feet of the actual distances.

Technologies and Equipment

LIBRARIES

OpenCV

Developer: Originally Intel, then Willow Garage, then Itseez (later acquired by Intel)
Use: Computer Vision and Machine Learning software library
Features:

- Has many different modules which will be of great use in this project, such as:
    - Image Processing (imgproc) - image processing module that includes linear and non-linear image filtering, geometrical image transformations, color space conversion, histograms and so on.
    - Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, stereo camera calibration, stereo correspondence algorithms, and elements of 3D reconstruction
    - 2D Features Framework (features2d) - feature detectors
    - Object Detection (obdetect) - detection of object and instances of some predefined classes (faces, eyes, mugs, people, cars and so on)
- Has thorough documentation for setup and required dependencies
- Supports Python, JavaScript, etc.

OpenCV is an open-source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, both classic and state-of-the-art computer vision and machine learning algorithms. There are a great number of tutorials online with examples of using OpenCV for computer vision projects. Some of the group members already have some experience using OpenCV.

NumPy

Author: Travis Oliphant (originally, now more a community project)
Use: Adds support for large, multi-dimensional arrays and matrices, along with high-level mathematical functions to operate on these arrays.
Features:

- Powerful N-dimensional array object
- Facilitates advanced mathematical operations on large numbers of data

NumPy is the fundamental package for scientific computing with Python. Images in computer vision are treated as 2-Dimensional (grayscale images) or 3-Dimensional (RGB images) matrices. NumPy is an essential package to manipulate these arrays with less effort.

WEBSITE

Technical Objectives

- Research different stacks and the technologies they use.
- Research TensorFlow and how to translate machine learning models from Python to TensorFlow models.
- Research Node.js and the Django Framework and evaluate which is more viable as a server-side application environment.
- Research React and learn how it works, what advantages it provides, and its viability for our application.
- Research Express and understand how to handle routes with Node.js.
- Evaluate pros and cons of different stacks and technologies.
- Research Docker and how it is used to package and deploy software.

Technical Goals

- Understand and learn to use Node.js and determine its strengths and weaknesses as an environment.
- Understand and learn to use Express.js and why it is the library of choice to use alongside the Node.js environment.
- Understand and learn to create user interfaces in React.js and be able to describe the difference between using React and conventional HTML + CSS templating engines.

Technical Specifications

- The application must work on both Windows and Linux operating systems. Since cross-platform functionality is essential, the application is being developed to accommodate those needs (Docker).
- The application must accept image inputs from the user to feed into the aforementioned models. There must also be user interface generated from these uploads from which the user can specify points that they wish to approximate the distance between.

Technical Requirements

- The website will need a server-side environment to handle requests from the user (Python-based models).
- The website will need dynamically updating user interface components to efficiently update and provide measurement approximations.
- The website needs to process user input with the machine learning models to estimate the parameters necessary for the depth calculation.

Research and Investigations

Potential Stacks

Web development stacks refer to the software that makes up a site's back-end. This includes but is not limited to programming frameworks, databases, application programming interfaces (APIs), and the operating system of choice. Web applications typically have four levels: the client level, the web server, the application server, and the database server. The client level refers to what the user or client can see on their browser. The web or HTTP server can handle simple requests such as requests for graphics and HTML files. The application server houses the server-side scripts that handle file requests and other more complex requests including redirecting potential database requests to the database server. The database server is the database of choice for a web application. If database information is queried, the application server requests the database server for any necessary data.

The most important qualities for the back-end stack in this project are speed, reliability, and easy integration between the machine learning model and the user input. The possible stacks were narrowed to a variant of a MERN stack (MySQL, Express.js, React.js, Node.js) and a Django stack (the group ultimately settled on the MERN stack). Because of the server costs and ethical questions associated with storing user-uploaded images on the website, the group opted to have a website with no database server. Per the requirements, the website must accept two stereoscopic images as input, include a front-end interface for the user to place points on the images they upload and then return the calculated distance between the points or sets of points.

It was thought that using a MERN stack or any JavaScript-based stack would integrate seamlessly with the TensorFlow implementation of our algorithm. The TensorFlow.js

library is a useful tool that would have allowed the group to run, build, and train new machine learning models in the same language as the rest of the site and offers the ability to convert models built using Python, such as a Keras model, into TensorFlow without having to recreate the model. TensorFlow.js also has a Python package that can be used to directly translate a Keras model into an equivalent model in JavaScript.

Alternatively, the group could have used a Python-based Django stack. For the development of our final algorithm, it was anticipated using one or many of the algorithms and methods provided in OpenCV, a free open-source library for computer vision and machine learning software, to build and train the final model. OpenCV has Python packages and modules that integrate natively with any Python app including a Python web app built using the Django framework. Python code is slower on average than code built using C / C++. However, the OpenCV Python packages are wrappers that call the underlying C / C++ code which eliminates the expected speed limits on the OpenCV Python code. These time savers are only true for the OpenCV Python wrapper packages, however, so any custom code used in the final deployment, as well as the Django based back-end, would have run slightly slower than an equivalent build using a JavaScript-based stack.

Research into Potential Stacks

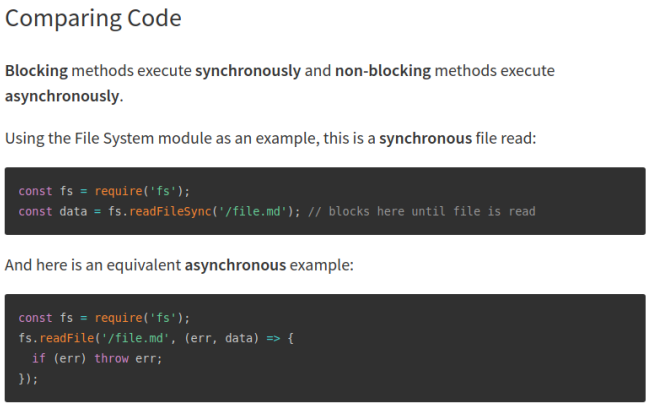
For the purposes of this project, the most important considerations when building the website were speed, reliability, ease of use, and the ability to communicate with the machine learning models. After initial research, the possible stacks were narrowed to either a MERN stack with TensorFlow to adapt the Python-based models or a Django stack.

The main differences between these implementations is the back end API and specifically the languages and frameworks used to develop them: Node.js, a JavaScript runtime environment used to create server-side code, and Python (Django), an interpreted language that is very easy to use, very well documented, and has incredibly useful libraries and packages for creating machine learning models and performing mathematical computations.

Node.js

Node.js differs from the Django framework in several ways, most notably how it handles requests. Node.js has an event-driven, non-blocking model that utilizes asynchronous programming. To be “event-driven” means that everything that happens in Node.js is a reaction to an event [22].

Code snippets demonstrating synchronized and asynchronous implementations [23]



Comparing Code

Blocking methods execute **synchronously** and **non-blocking** methods execute **asynchronously**.

Using the File System module as an example, this is a **synchronous** file read:

```
const fs = require('fs');
const data = fs.readFileSync('/file.md'); // blocks here until file is read
```

And here is an equivalent **asynchronous** example:

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
});
```

Blocking refers to when JavaScript code execution is blocked as it waits on a non-JavaScript operation such as I/O or requests for files from the server. The synchronized methods and operations provided by the Node.js standard library are the most commonly used blocking operations. All of the synchronized I/O operations in Node.js also have matching asynchronous, or non-blocking, methods and accept callback functions.

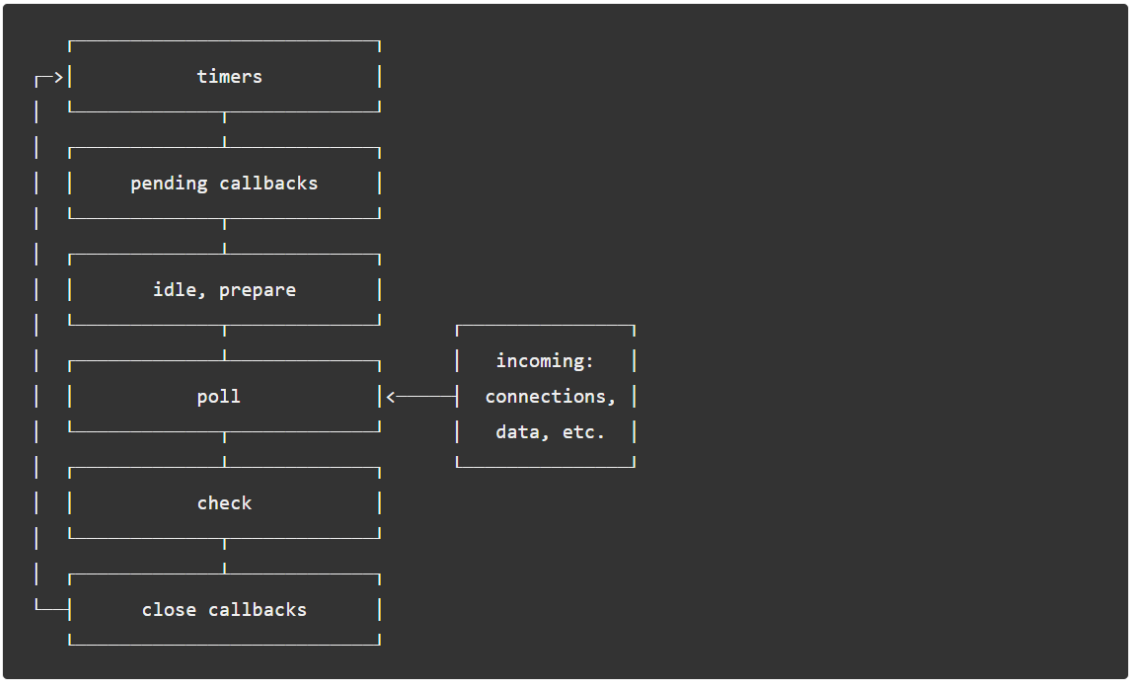
The first code segment is a JavaScript snippet that does not use an asynchronous, non-blocking scheme. The program is simpler to understand since it looks and acts like traditional code, however, in the second line blocks the execution of the code and requires that the file read, an I/O action complete before it resumes. The second code snippet does the same thing but utilizes Node’s asynchronous method instead. The advantage of using the asynchronous method is that the event loop does not have to wait for the I/O action, the `fileRead()`, to finish. Because of this, the event loop can continue to react to user events instead of waiting on slow I/O and file requests [23].

The Node.js Event Loop

The Node.js event loop is not a library, but rather a runtime construct that defines how and when Node responds to events. Typically, server behavior is defined through callbacks at the beginning of a script and then the event loop is initiated at the end of the script by a blocking call. Node, on the other hand, begins the event loop after the execution of the initial script [24].

Node.js boasts superior performance and throughput than its competitors due to its asynchronous nature and event loop construct. The Node.js event loop is broken down into separate sections called phases which make up its order of operations. Callbacks are a term used in Node.js jargon that refers to asynchronous functions. Callbacks allow the developer to continue to process events and take requests from the user without having to wait for all of its pending requests to process. This allows Node.js to handle large amounts of I/O requests or any other requests that require OS resources since it does not have to wait for a response [25].

Simplified version of the event loop [24]



Each phase in the event loop has a queue that holds its callbacks. When the event loop enters a phase, it performs any necessary operations and then dequeues callbacks and executes them until there or none left, or it reaches its execution limit, whichever comes first. Once either of these conditions has been met, the event loop will continue on to the next phase.

Phase Overview

- Timers
    - Timers specify the amount of clock time that must pass before a callback can be executed. As soon as callbacks are available in the timer phase, they will be executed.
- Pending callbacks
    - Certain system operations such as error logging are queued here.
- Idle / Prepare
    - Abstracted from the user. This phase is used internally by Node.
- Poll
    - Two main functions
        - Determining how long to block and poll for I/O.
        - Process events in the poll queue.
- Check
    - Callbacks can be set to this queue with the `SetImmediate()` function
    - If the poll phase is clogged waiting for certain requests and is idle, the event loop will continue on to this phase and execute events in this queue rather than wait on the poll phase.
- Close Callbacks
    - ‘Close’ events are emitted in this phase when sockets are destroyed.

Installing Node.js

Node Version Managers

A node version manager is a command-line program that allows users to install and use multiple different versions of Node.js. With a node version manager, developers can verify the functionality of their code and various common versions of Node.js. This tool is critical for ensuring the compatibility of the source code.

Installing a Node Version Manager

Depending on the OS, a different node version manager is needed. There are other viable node version managers however, herein the group describes how to install the ones we used.

- OS X and Linux Node Version Managers
  - nvm
    - To install nvm, run the following command in the terminal to download and run the installation script [26].
    - `$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash`
- Windows
  - nodist
    - nodist is designed to replace an existing Node.js installation. If one already has Node.js installed, it must be uninstalled before proceeding.
    - To install nodist, navigate to the nodist git repository and download the latest installer. The installer can be found here [27].

After installing a node version manager for the operating system, it must be ensured that the latest version of npm is installed. npm is the node package manager and will be essential later for installing node packages for building and testing our code. The following code segment shows how to install it via the command line [28].

```
$ [sudo] npm install npm -g
```

The Django Framework

Django is a high-level web-development framework for Python that encourages rapid development by having an opinionated, monolithic framework. The appeal of the out-of-the-box solutions provided by Django is the reduced development time of large, complex web applications with common use cases. Another useful trait about Django is that it integrates with the mathematical calculations, image processing functions, and machine learning models written in Python for other parts of the project. Django is highly scalable which is good for a larger-scale application but less suited to the limited use case of our application.

Django has the advantage of a smoother learning curve for the use cases that it supports out of the box. The large community support surrounding the framework also aids this learning curve, with extensive documentation and examples for Django. Django, however, does require that its users be very familiar with its way of doing things. Since Django is a monolithic framework, it will not work as cleanly with the other technologies we are using, such as React for the application side, compared to an all JavaScript environment [29].

Installing Django

Before installing Django, ensure that the latest version of Python3 is installed and that the Python package manager `pip` is installed. The following command installs Django [30].

```
$ pip install Django==2.2.3
```

Express.js

Express advertises itself as a “fast, un-opinionated, minimalist server-side web framework for Node.js.” The advantage that an un-opinionated framework provides to a developer is control.

Essentially, what Express allows the team to do is make the most out of the environment that Node provides. Node itself does little to set up the web application outside of providing the asynchronous event loop model and access to npm’s extensive module library. Express provides the capability to handle and resolve incoming and outgoing traffic to build a web server and can connect to any database the developer likes. Because

of its less restrictive nature, it allows the team to work with virtually any other technology alongside it [31].

Installing Express

Navigate to the application's root directory or create it with

```
$ mkdir (app_name) && cd (app_name)
```

Once the directory has been created, the following command is ran:

```
$ npm init
```

This command creates a package.json file for the application. The package.json file for an application is essentially a manifest that provides npm with important information regarding the project, including but not limited to its name, the version of the application, the dependencies it requires, user-defined scripts and much more. Every node application needs a package.json file, and this command creates a bare bone one for the user. Follow the prompts and fill out the basic information for the application. Most of the prompts can be disregarded until the entry point is prompted for. When this point is reached, input the desired application name.

```
$ entry point: (index.js)
```

The default name is index.js, however, this can be named as one sees fit. The chosen name will automatically be added to the application's package.json file as the main entry point.

Now the package.json manifest file has been created for the application, Express can be installed by running this command in the application's root directory.

```
$ npm install express --save
```

The `--save` flag automatically saves express to the application's dependencies list. To prevent Express being saved to this list, use the `--no-save` flag [32].

React.js

React.js is a declarative, component-based library for building user interfaces. React allows the developer to define and reproduce components that will compose the overall interface. The power of React lies in JSX, the syntax used to describe the components the group renders. JSX looks and acts similarly to HTML, however, JSX comes with the full capabilities of JavaScript for anything written within curly braces. All components are JavaScript objects and their data are accessible throughout the scope of the project. React allows the developer to create stateful components that efficiently re-render when their states are updated. Most importantly, React works seamlessly with any JavaScript technology we want to use alongside it. This functionality is crucial since the team will be working with machine learning models built in TensorFlow to make our distance approximations [33].

MySQL

MySQL is among the best Relational Database Management Systems and is also the most common. MySQL has a large community of users and developers which makes it easy to find help even with niche issues. It is incredibly powerful and can handle the datasets that we intend to use it with and uses a standard language (SQL) that most developers are familiar with. Most importantly, MySQL is open-source and available for use by the public free of charge.

As mentioned in the data collection portion of the report, the database must store the training data and possibly other miscellaneous pieces of data. The accuracy of the models depends on the images that it trains on. MySQL provides a way to access images as necessary to ensure the reliability of the model [34].

Docker

A docker container is an executable software package that includes all of the code, runtime, settings, system tools and libraries. Downloads for the Docker Engine are available for Windows, MacOS, and various common Linux distributions including Debian, centOS and Ubuntu [35].

Docker containers are built according to the instructions provided in the Dockerfile in the root context. Dockerfiles contain instructions with the commands that a user should run to assemble the image. The commands provided in the Dockerfile will only work if they

are syntactically correct, ensuring that environments are built error free. Using the CMD instruction, the user tells the container what process to run. Individual docker containers are only restricted by their context. When running an application with multiple modules such as a MERN application, it is useful to isolate the environments of each procedure for added security and performance. For this application, the API code is built in its own docker container, and the client code is as well. The Dockerfiles in the root of both directories provide instructions for preparing each server and are combined by using `docker-compose` CLI [36].

Docker Compose is a tool for defining and multi container docker application. Docker Compose reads in a `docker-compose.yml` in the parent directory of the dockerized code modules and defines the docker images as services that can be run in parallel. In this case, the services are the server, which provides the applications API code, and the client, which renders the front end and gives real-time requests to the server. The docker images run separately from one another, however, the communication ports used for interacting with the localhost and with each container remain open and functional.
This command builds the image:

```
$ docker-compose up --build
```
[37]

Once the application image is built, the docker image can be uploaded to the Docker Hub where it is available for public or private use. Docker Hub also provides a feature where it auto-builds new commits to a designated repository. The dockerized version of the application also functions as an isolated testing environment. By using the `up` command, future developers can run and destroy the application as testing environments with minimal latency [37].

Once the full application docker image is built and hosted on the Docker Hub, the application will be available for use on any machine with the Docker Engine installed. Open a terminal or command prompt in the respective operating system and run the command:

```
$ docker run {imageName}
```
[38]

If the image does not already exist on the user's machine, the docker daemon will fetch the latest release of the specified docker image. This can also be done using `docker pull`.

Running the docker image will launch the docker containers and run the current development build of the application [39].

Docker empowers the developer with tools to ease debugging and testing, while simultaneously providing self-updating deployments of our latest application code.

Design Summary

The project consists of a program written to receive two stereoscopic input images. The images should be of the same object but with varied angles. The program scans and identifies the objects in both images. The user first inputs a reference measurement and then they are able to draw lines for measurements they wish to determine (e.g. the distance of one object relative to another). Afterward, the image and measurements are fed into a program and the estimated measurement is outputted to the user.

Design Content

Back End Architecture

The web server and API endpoints for the website are designed to ensure simplicity, reliability, and efficiency. During development, several technologies were considered to extend the functionality of our web server, however, it was decided to stick to a more reasonable stack in order to avoid scope creep and maintain quick server response times. The server consists of a Node.js web server that handles the serving of webpages and providing an API for the client-side user interface to process images with our algorithm. Node was decided as our framework of choice due to its strong documentation and large community of developers, its reliability and strong performance, the availability of useful packages for both Node and JavaScript in general such as Express and TensorFlow, and the development benefits of having an all JavaScript environment.

NGINX, a web server commonly used with Node.js servers, is not used for several reasons. The primary reason is that the benefits provided by NGINX are not significant enough to justify the increase in complexity and development time that would be necessitated by the usage of NGINX. Because of the simplicity of the Node.js server and the API, there is not a sufficient security risk to require a reverse proxy to protect the Node.js server from direct internet contact. Such a reverse proxy would only act as a hindrance to server response times. Based on the available resources and expected user

quantities, it is expected that there would be only one instance of the Node.js required to meet user demands. As such, there would be no benefit from using NGINX as a load balancer. In addition, since the server would only be serving one static web page with dynamic content being loaded after the page has been served to the client, the HTTP cache provided by NGINX would provide a very limited benefit. Finally, using NGINX increases the overall complexity of the back-end architecture meaning that there are more moving parts, more opportunities for things to go wrong, greater maintenance requirements, and more involved training required for any future maintainers of the server. For these reasons, the costs of using NGINX is considered to be greater than the benefits that it would provide.

Multiple templating engines were considered for our Node.js server, especially Pug (formerly known as Jade) and EJS (Embedded JavaScript templates). Although there were not any technical specifications that favored the usage of one templating engine over another, it was determined that EJS would be the preferable templating engine since it has a syntax that is similar to PHP, and would, therefore, be easier to learn and use than another templating engine. With further consideration, however, it was eventually determined that a templating engine would not be necessary at all since our web server would only be serving one page with no required preprocessing. By having client-side JavaScript access the server API and load any necessary content as needed, load times for the website's sole page would increase. In addition, the complexity of the web server would be decreased dramatically, increasing stability and decreasing maintenance requirements.

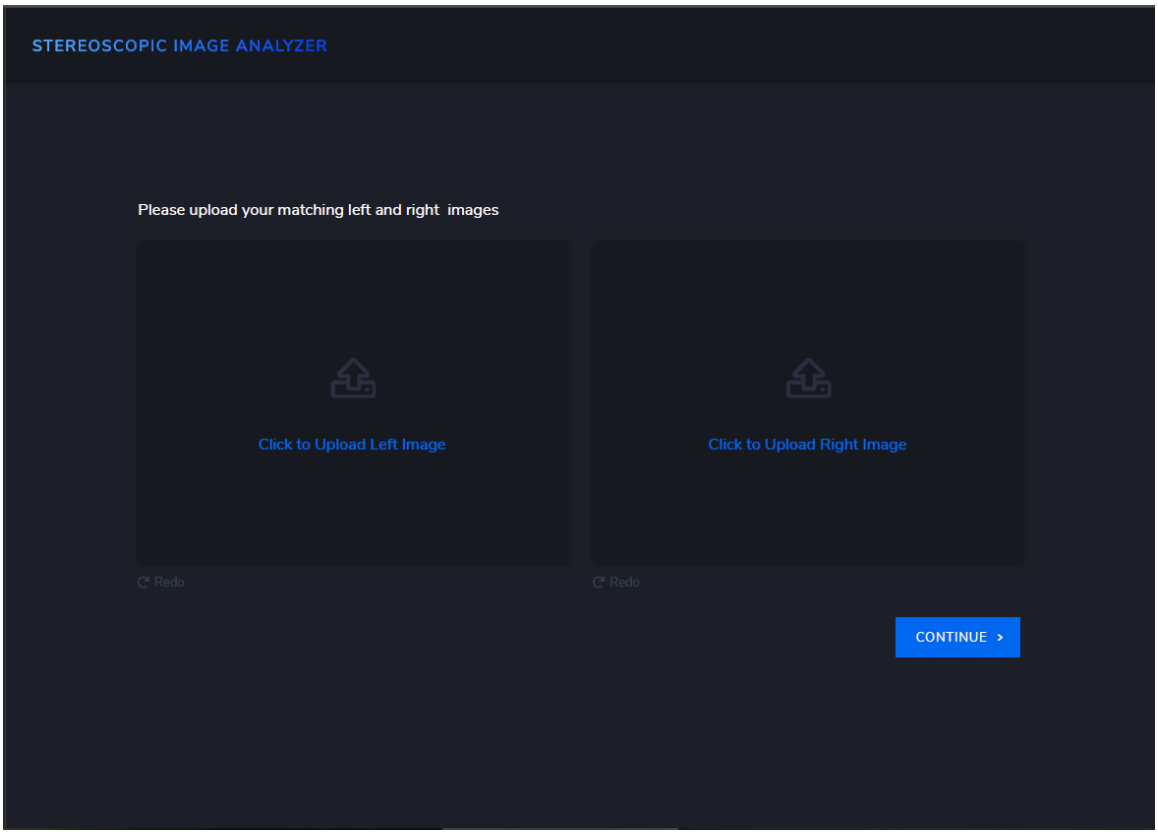
Front End Architecture

The front end of the application was built using React.js and its dynamically updating components. React provides the front-end with several useful properties that make it an ideal choice for our project. One of the goals of this project is to ensure that the final project is usable and maintainable by our sponsor and potentially by another senior design group. The separation of application logic from the front-end layer built in React makes our codebase more readable.

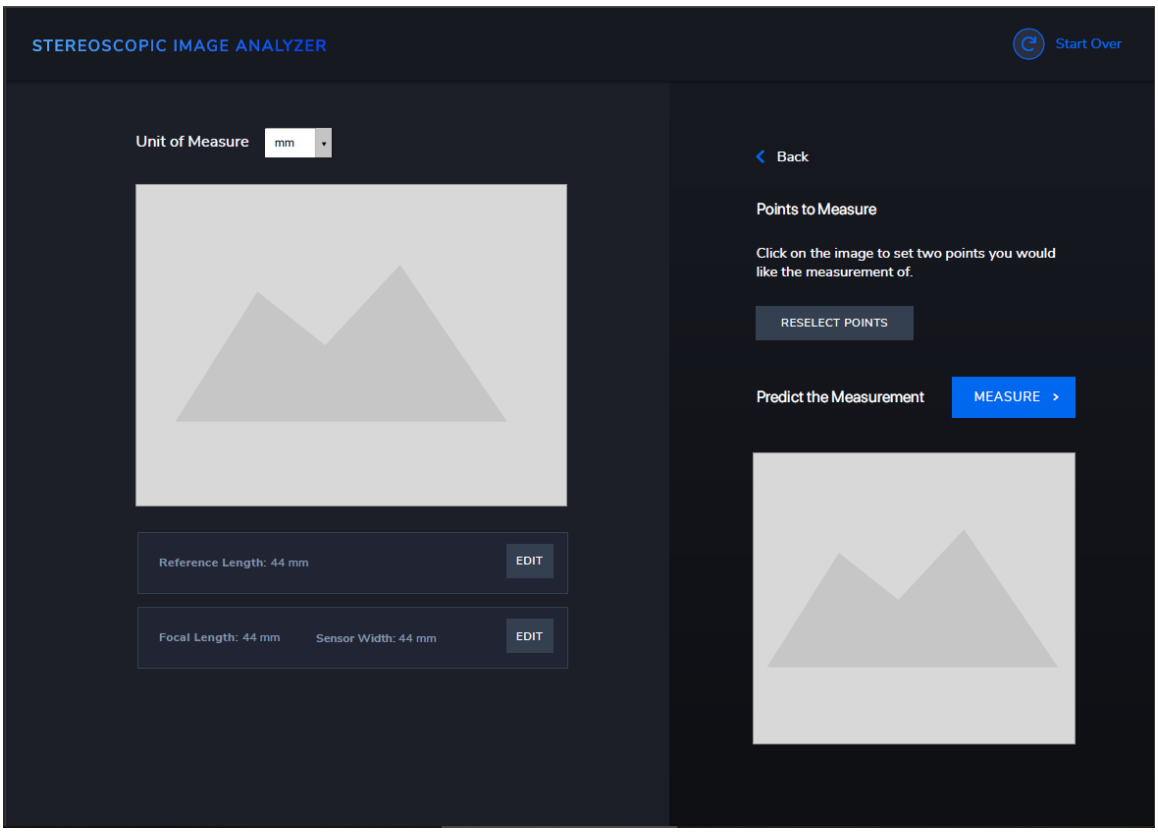
React's component-based model allows front-end code to be easily modified in the future and also allows bugs and other issues to be isolated to specific functions and data structures. React gives our users a way to provide sanitized inputs and parameters for both the back-end and TensorFlow to work with while removing itself from any application logic.

User Interface Wireframes

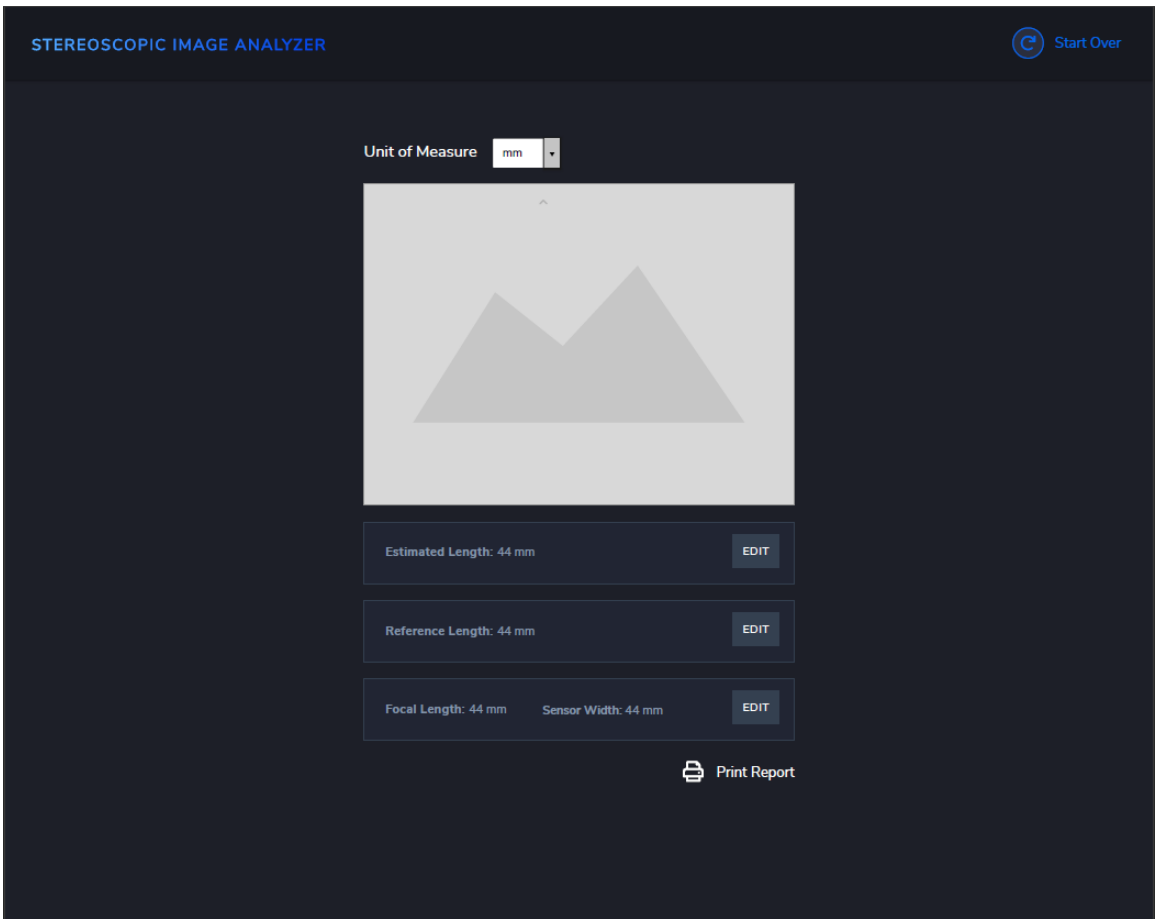
User uploads two stereoscopic images into the home page of the user interface.

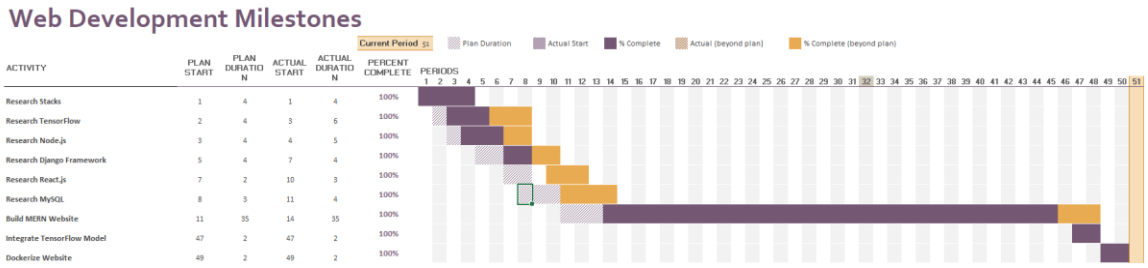


The program will scan both images for objects. The user will then input any known measurements and the measurements they wish to know.



The program will then determine and display the unknown measurements based on the known passed measurements.





Development Timeline

Research Stacks and Related Technologies

May 22nd - July 22nd

Various web development stacks were examined, and the pros and cons of each language and/or framework were learned. After surveying the most common options and the requirements of this project, it was decided that the two most viable stack options were MERN, an all-JavaScript full stack development solution, and a Django stack, a robust, easy to learn web framework with the added advantage of being written in Python, allowing for simple integration with the data science and image processing modules included in the rest of the application.

Research TensorFlow

June 1st - June 8th

TensorFlow was researched to find out what compatibility issues and fixes it presented. It was discovered that most common models trained with Python libraries were directly translatable to TensorFlow.js via a Python package called `tensorflowjs`. This revelation was essentially the last nail in the coffin for Django as a possible framework. This module allowed the entire web application to be written in JavaScript since the models could be integrated easily on the client-side with the help of TensorFlow. (It was eventually decided that the model would be called in a back-end Python API in the MERN stack).

Research Node.js

June 8th - June 18th
Node.js was researched. The group vaguely understood Node to be a JavaScript alternative to a Python or Ruby server-side implementation, but that was the extent of it. After reading into the subject, the group learned of Node's event loop and asynchronous, non-blocking I/O model and the advantages that the environment provided, such as eliminating thread-based networking and deadlocks. The group learned about Node Version Managers and how they help in the development of enterprise software. With an nvm installed on the team's machines, the team could easily stay up to date with node, npm and other associated modules globally while also being able to revert to other versions of these technologies to ensure backwards compatibility.

Research Django Framework

June 18th - June 29th
The Django framework and what benefits it could provide the application as the web framework of choice was looked into. Django comes with out-of-the-box solutions for a variety of common use cases and expedites code development for projects with well-defined needs. Django is also a Python-based web framework, meaning that the API would be written in the same language as the data science and image processing programs. For the image processing section, this could have be a boon thanks to the potentially intensive server-side calls for image uploading and processing.

However, Django requires the developer to be intimately familiar with the details of the entire framework and work with its "opinions," such as the Model Template View (MTV) architecture. Django excels for projects where the task is not to reinvent the wheel, but rather take advantage of the known best practices for web development as decided by web developers. The group's solution was too small to make most of the benefits that Django would have provided as a framework, such as its highly scalable architecture style, its support for rudimentary web server features such as user authentication and RSS feed integration, and would also introduce a new language into an otherwise all JavaScript environment. As such it was abandoned as a potential framework.

Research Express.js

June 29th - July 13th
After learning about Node.js and what it brought to the table as a technology, the team was curious to see what Express.js brought to the table. It was discovered that what Express essentially does is provide all of the functionality an application could need while taking advantage of the environment created internally by Node. The team learned how to install Express and add it to the project's dependencies list for later usage and development.

Research React.js

July 13th - July 20th
After looking into the possible back-end technology solutions for the project, the next phase was to look at front-end solutions. React.js is declarative, component-based library for building user interfaces. React allows front-end developers to create dynamically updating, stateful components that can make up larger interfaces. Because of the modular nature of React, it allows developers to more easily develop and debug features for their applications since all functionality can be narrowed down to a single culprit. This makes React a natural choice since it is highly maintainable and allows the team to use the power of JavaScript to create interactive and robust user interfaces. Since React is a JavaScript library, it integrates perfectly with other JavaScript libraries, most notably TensorFlow.js. This integration is crucial since the TensorFlow models will rely on the input supplied to them via application code written in React.

Research MySQL

July 20th - July 24th
MySQL is a robust, open source RDBMS with extensive documentation and user community that can handle datasets both large and small. MySQL has drivers for all common operating systems and for several different flavors of back-end technology, such as Flask, Django and Node.js. The database requirements, although small, are at the heart of the solution we are trying to provide. The database provides access to the training data the team gathered for our neural networks and informs the network on how best to estimate certain characteristics of the image, namely, focal length and sensor width.

Build MERN Website

July 24th - November 22nd
The website is built and consists of a MySQL database on a remote database, an Express-based API operating in a Node.js environment, and a front-end user interface built with React to provide a rich feel for the client. It accepts user-inputted images, allows the user to select points on the images that they input, and returns the distance approximations.

Integrate TensorFlow Model

November 18th - November 22nd
The model is integrated and ran via a Python-API on the website's backend.

Dockerize Website

November 18th - November 22nd
The website is transferred to Docker so it will be easier for the sponsor to deploy and use without having to worry about downloading dependencies or being on a certain OS.

Build Plan

1. Build a web server that supports an API for the application.
   The finished product must include an API to handle user requests including back-end calls that require an image as input for preprocessing and depth estimation, validation of stereoscopic images, and other parameters from the user that will be needed.
2. Build a dynamic front-end to work in conjunction with the API.
   The front-end has many jobs and its foremost duty is to provide the user with a rich, interactive feel. The front-end must update efficiently and dynamically to changing user input and be able to reliably capture user input as parameters modules in other layers of the application.
3. Create a MySQL database.
   The MySQL database will hold the training data for our neural networks. By using a remote database server, an additional tier is added to our application architecture. Isolating the main responsibilities of the application provides many benefits, including increased security, modularization of the code base, pinpoint scalability needs on a tier by tier basis, and add functionality without disrupting existing code. The database server also ensures consistent availability of unadulterated data for the neural networks to train and retrain on.
4. Integrate the machine learning network into the application.
   The backend API must be able to run the machine learning network.
5. Dockerize website
   The website will be dockerized thereby making it easier for the sponsor to deploy and use.

Prototype Plan

The team plans on building a prototype using the MERN Stack. This plan will describe the procedure for the prototype.

1. The MERN Prototype
    a. Create a web server using the Node.js environment and Express.js web framework.
    b. Create a dynamic user interface in React to service the application's requirements from the user.
    c. Create a remote MySQL database to house the training data.

Test Plan

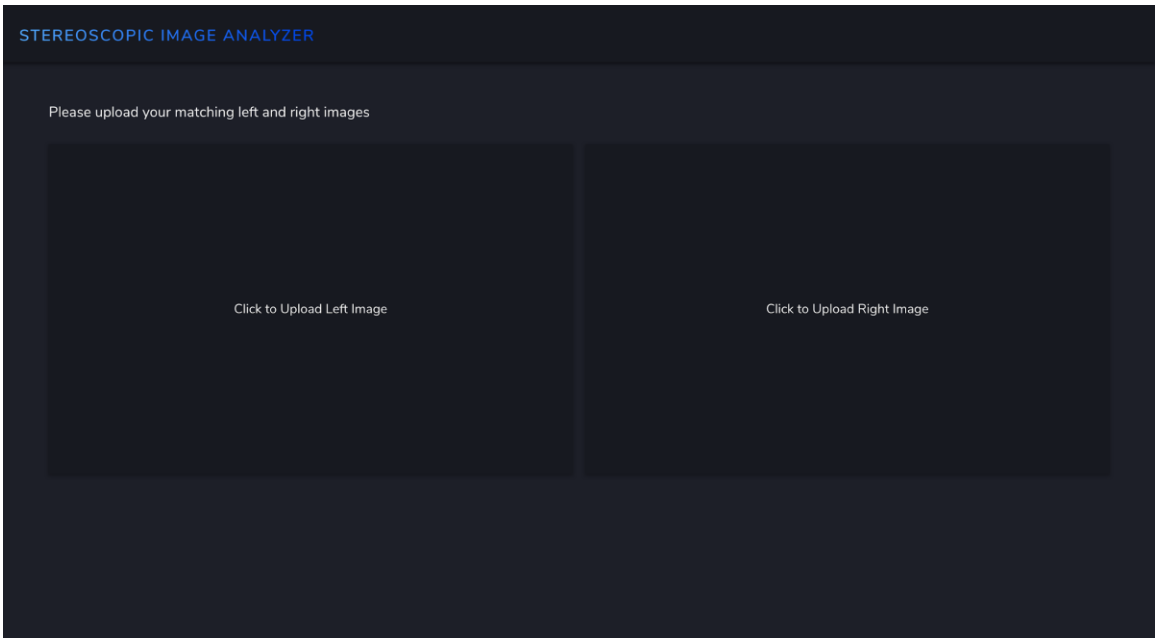
1. Test the application with real users and adjust the design of the application based on user feedback. User feedback allows the team to take outside opinions into consideration about the functionality of the application, including speed, user-friendliness, and reliability.

Evaluation Plan

- Define criteria with which to judge the application
  The application must work on both Windows and Linux machines, it must accept user input, process the user input and then return back estimated distances between points selected by the user. The application must be scalable and must be beginner-friendly. All of the computation and parameter estimation should be abstracted away from the user.

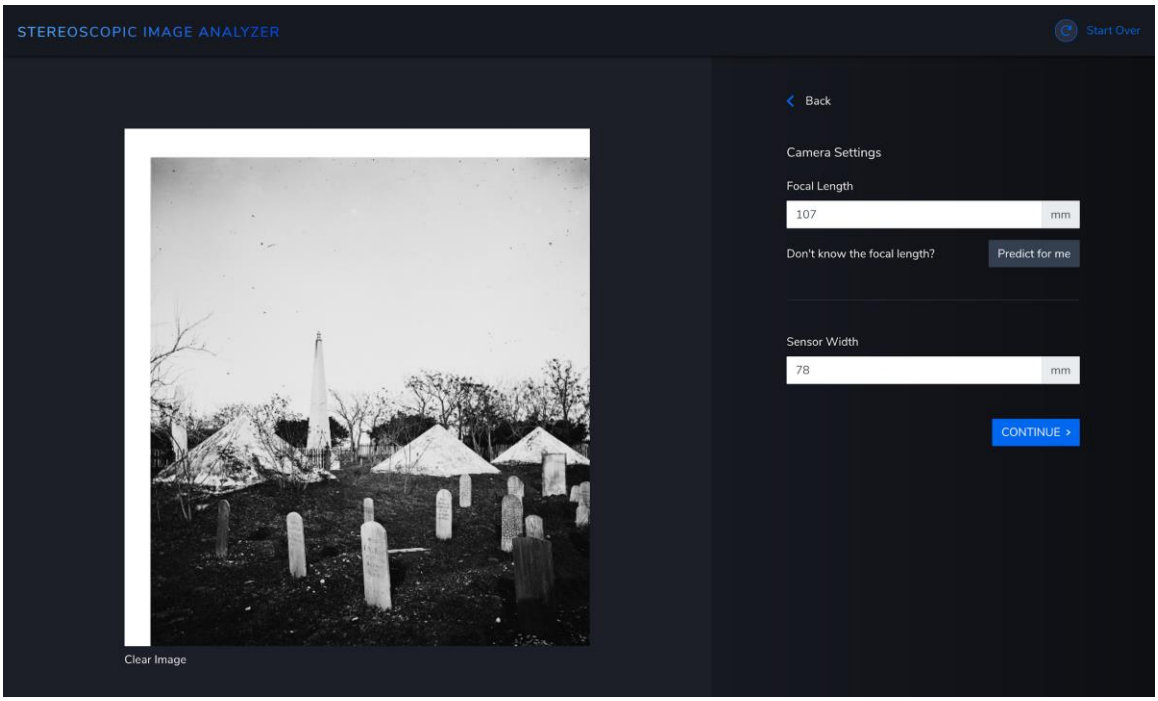
Screenshots of Final Product

On start, application prompts user to upload the matching Left and Right stereograph images.

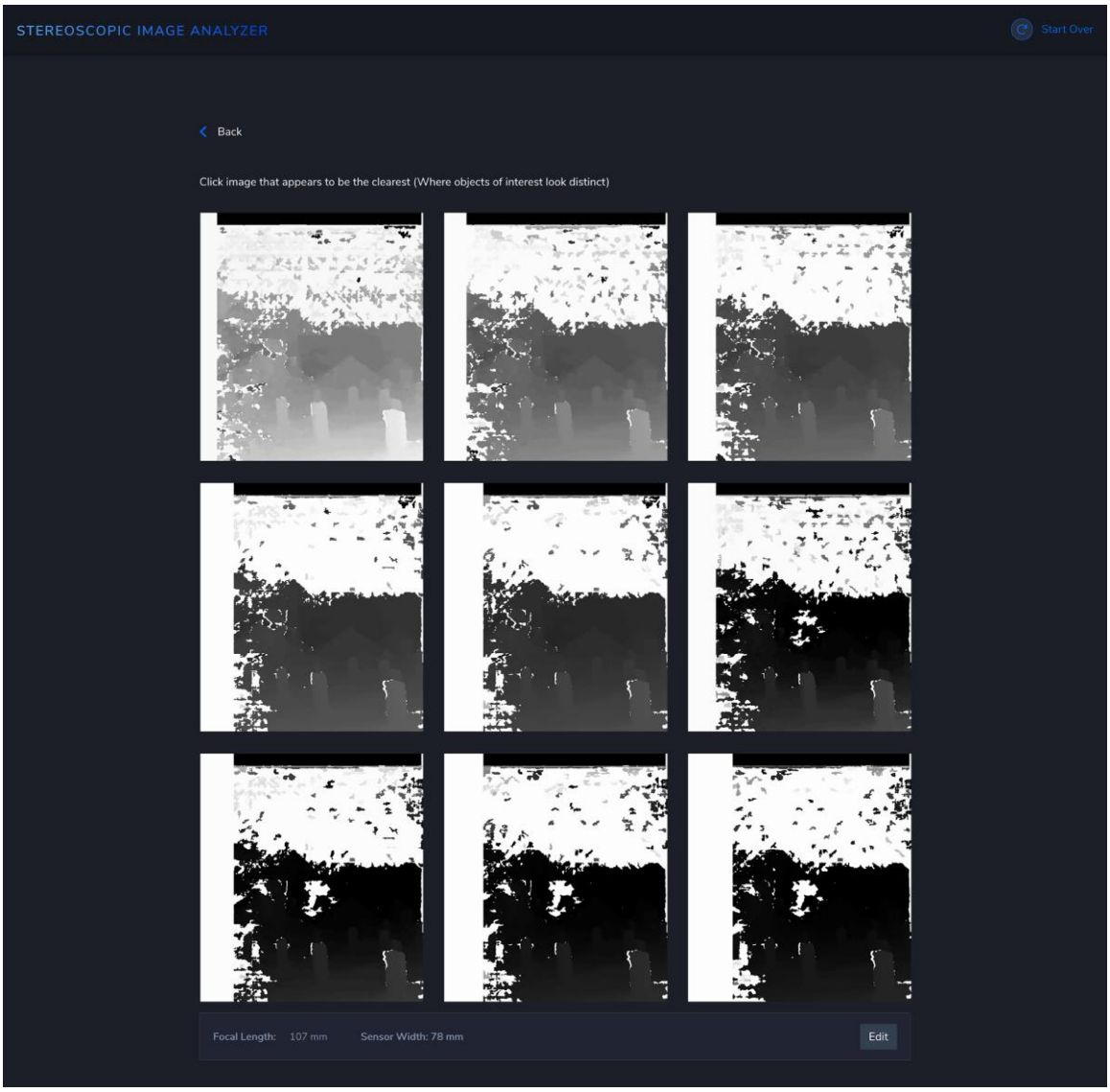


After upload, images are shown, and user is allowed to continue to the next step.

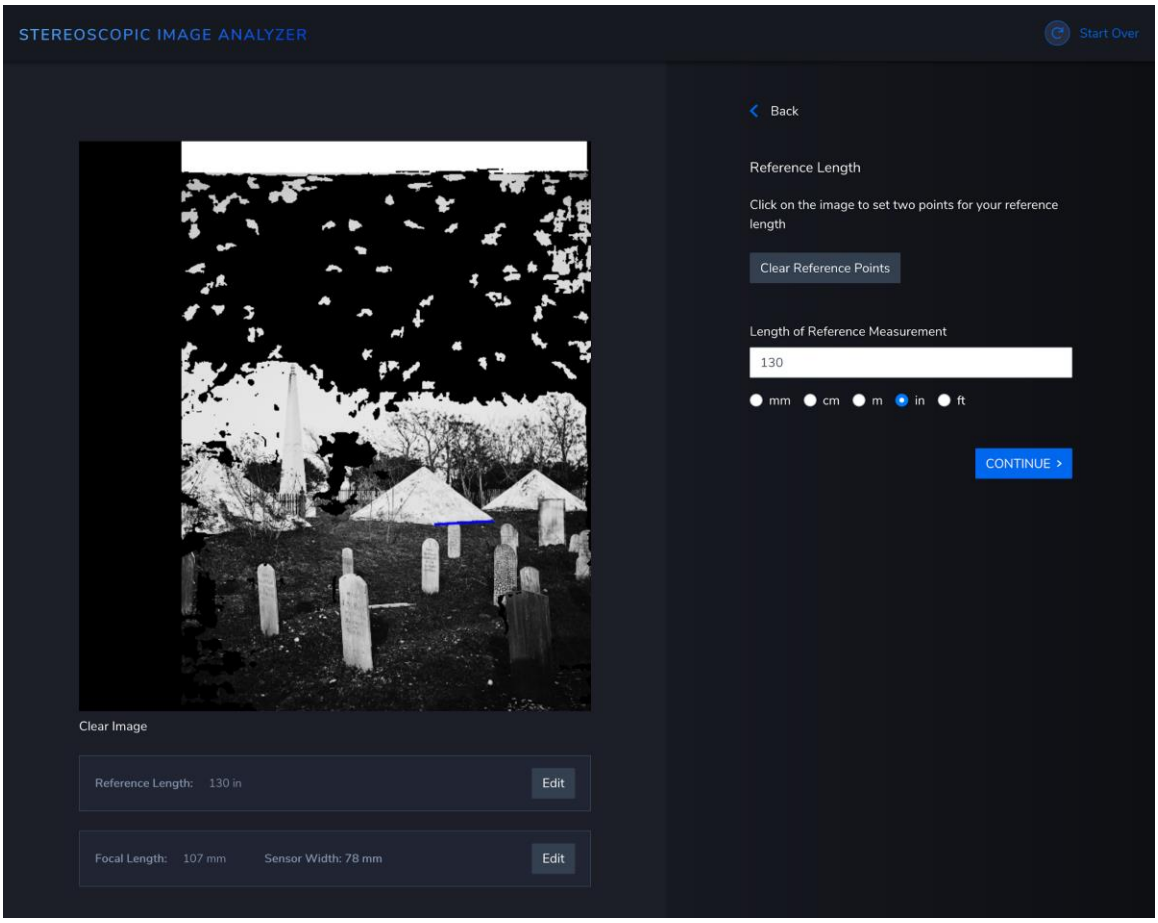
Application prompts user to input the known focal length and sensor width. If user does not know focal length, user can click “Predict for me” button that runs the left image through the machine learning model trained for predicting the focal length.



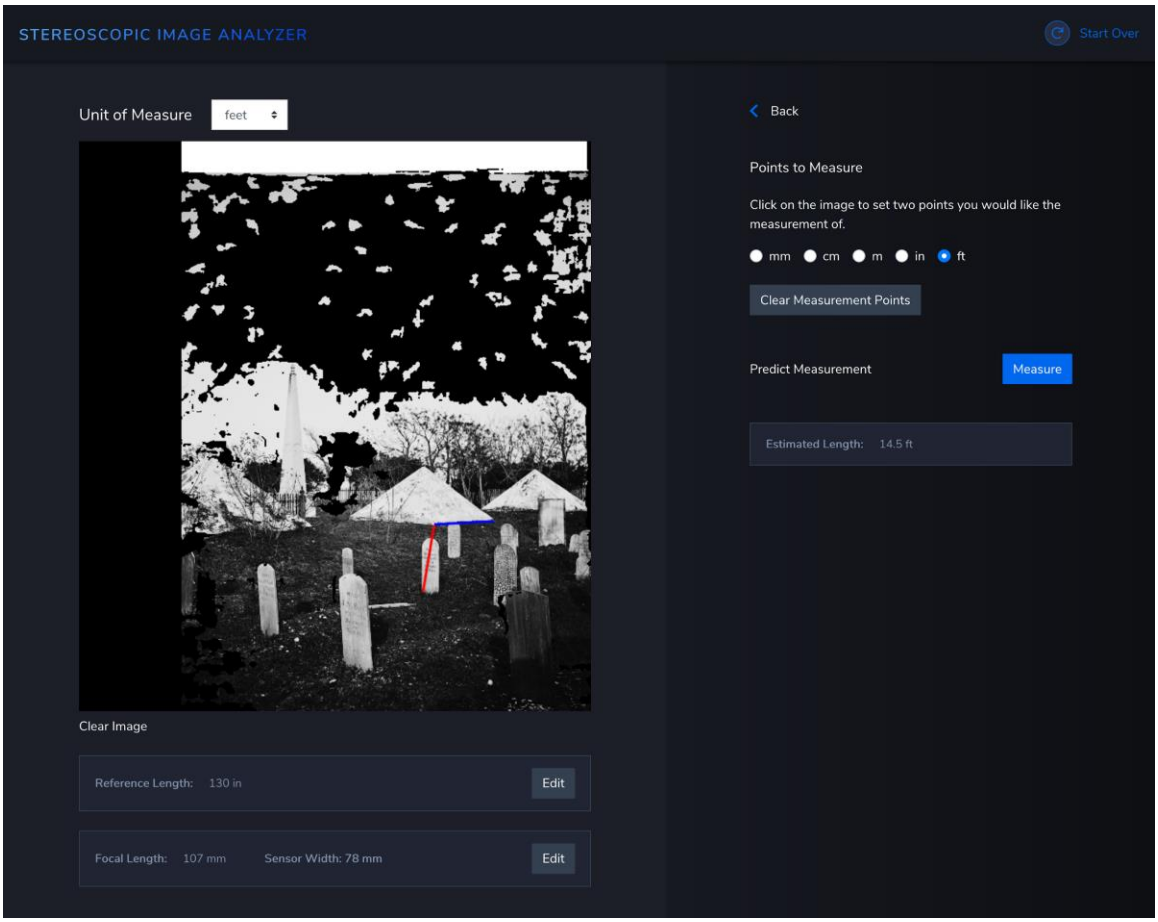
On the next step, parameters are run through model and disparity maps are generated. The application then prompts the user to click on the “best” disparity map that was generated. The images are tied to the settings needed for the 3D reconstruction.



After choosing the best disparity map, the application runs the settings and blocks out the pixels on the image without valid data. The application then prompts the user to select two points and their real world length.



At the final step, user is prompted to select two points to measure. Once points are selected, user clicks “Measure” button and all the setting will be sent to model to be calculated. Once model calculates a value, the front end is updated to display the measurement with the measurement unit the user selected.



Technologies and Equipment

The installation and technology descriptions provided in this section assume that the user is on a Linux-distribution as that is what the Docker container the app is deployed with is based in. The technology used in this application is exclusively open-source and all development will be done locally.

BUDGET

The project was self-funded and relied on free open-source software. Our sponsor Dr. Giroux was contacted for external costs which were limited.

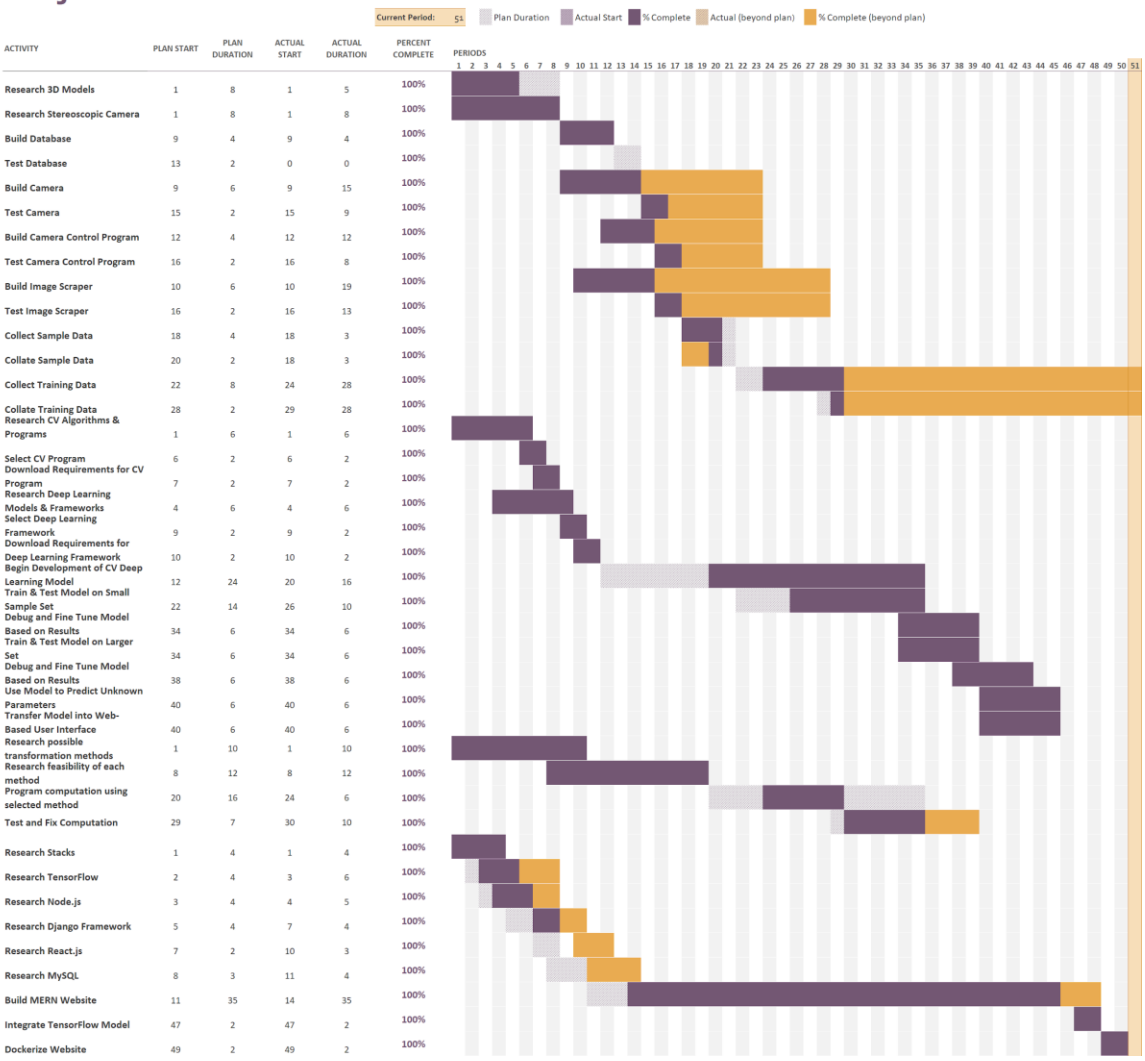
The development of our stereoscopic camera incurred some costs. Although the smartphones and camera tripod have been provided for us free of charge, we needed to spend money to have a mount for those phones. We decided that the optimal use of our money was to find equipment online, rather than spend the time and effort to try and potentially fail to create our own mount. We looked on Amazon and found some equipment that was reasonably priced and fulfilled all the requirements that we had. This equipment was ordered from Amazon and paid for by Dr. Giroux. Below is an overview of the equipment costs:

2 x Neewer Smartphone Holder Vertical Bracket : $20
1 x Neewer Dual Camera Mount Tripod Bracket : $10

TOTAL **$30**

MILESTONES

Project Milestones



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Research 3D Models	1	8	1	5	100%
Research Stereoscopic Camera	1	8	1	8	100%
Build Database	9	4	9	4	100%
Test Database	13	2	0	0	100%
Build Camera	9	6	9	15	100%
Test Camera	15	2	15	9	100%
Build Camera Control Program	12	4	12	12	100%
Test Camera Control Program	16	2	16	8	100%
Build Image Scraper	10	6	10	19	100%
Test Image Scraper	16	2	16	13	100%
Collect Sample Data	18	4	18	3	100%
Collate Sample Data	20	2	18	3	100%
Collect Training Data	22	8	24	28	100%
Collate Training Data	28	2	29	28	100%
Research CV Algorithms & Programs	1	6	1	6	100%
Select CV Program	6	2	6	2	100%
Download Requirements for CV Program	7	2	7	2	100%
Research Deep Learning Models & Frameworks	4	6	4	6	100%
Select Deep Learning Framework	9	2	9	2	100%
Download Requirements for Deep Learning Framework	10	2	10	2	100%
Begin Development of CV Deep Learning Model	12	24	20	16	100%
Train & Test Model on Small Sample Set	22	14	26	10	100%
Debug and Fine Tune Model Based on Results	34	6	34	6	100%
Train & Test Model on Larger Set	34	6	34	6	100%
Debug and Fine Tune Model Based on Results	38	6	38	6	100%
Use Model to Predict Unknown Parameters	40	6	40	6	100%
Transfer Model into Web-Based User Interface	40	6	40	6	100%
Research possible transformation methods	1	10	1	10	100%
Research feasibility of each method	8	12	8	12	100%
Program computation using selected method	20	16	24	6	100%
Test and Fix Computation	29	7	30	10	100%
Research Stacks	1	4	1	4	100%
Research TensorFlow	2	4	3	6	100%
Research Node.js	3	4	4	5	100%
Research Django Framework	5	4	7	4	100%
Research React.js	7	2	10	3	100%
Research MySQL	8	3	11	4	100%
Build MERN Website	11	35	14	35	100%
Integrate TensorFlow Model	47	2	47	2	100%
Dockerize Website	49	2	49	2	100%

PROJECT SUMMARY

The CNNs developed to predict the focal lengths and angle of view had high training accuracies (>0.7). When tested on the Civil War stereoscopic images with a reference measurement on the base of the middle pyramid, the focal length model returned 105mm which was plausible given that there existed cameras from that time period with that focal length. The mathematical algorithm when using the focal length model and tested on a modern scene of the St. Augustine cemetery, returned measurements that were slightly larger than real-life measurements (off by 3 to 4 feet). When tested on the Civil War stereoscopic photographs, the results were plausible in some cases but too large in others.

When tested on the Civil War stereoscopic images, the angle of view model returned 19 degrees which was also plausible. The mathematical algorithm when using the angle of view model and tested on a modern scene of the St. Augustine cemetery, returned measurements that were closer to the expected measurements for some desired measurements but still slightly off for others.

Discrepancies in calculating the distance in the Civil War photographs were attributed to noise in the original photographs, lack of contrast (e.g. warping near the end of the original glass plates), and the choice of initial reference points (for the previous calculations, the reference measurement was on the middle pyramid). After trying out the CNN models, it was decided to manually input some of the parameters to determine how this would affect the results. The sensor width measurement in the Civil War stereoscopic photo was set to 78mm and the focal length was varied from 50mm to 100mm to 150mm. In addition, the reference measurement was moved to the leftmost pyramid where both the leftmost desired measurements were originating from. It was found that this greatly improved both the leftmost measurements in the 50mm example. After discussing this with our sponsor, it was concluded that the initial reference measurement played a critical role in acquiring accurate results and now knowing this in conjunction with the models and the known measurements, the GUI should be very beneficial in assisting with the retagging problem.

CONCLUSIONS

We have presented a novel way to use machine learning, epipolar geometry, and computer vision to predict distances in stereoscopic photographs. We developed two models to predict two camera parameters needed for distance estimation (focal length and angle of view) and compared the end result of both. Our end application did well in measuring distances in a modern scene of the target cemetery when the initial reference measurement was near the desired measurement. Likewise, our application did well on the same cemetery from the 1800s in similar situations. Both did well when the desired measurements were within a nearby vicinity to the original reference measurement. With this application and these findings, we anticipate that our sponsor will be able to properly retag the nine men whose names are visible in the old stereoscopic photographs.

REFERENCES

[1] Srinivasan, Pratul P., et al. “Aperture Supervision for Monocular Depth Estimation.” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, doi:10.1109/cvpr.2018.00669.

[2] Peng, Xingchao, et al. “Learning Deep Object Detectors from 3D Models.” *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, doi:10.1109/iccv.2015.151.

[3] C, Michael. “How Can I Get the Image/Sensor Dimensions in Mm to Get Circle of Confusion from EXIF?” *Photography Stack Exchange*, 13 July 2013, photo.stackexchange.com/a/40870.

[4] Mrovlje, Jernej and Damir Vrančić. “Distance measuring based on stereoscopic pictures.” *9th International PhD Workshop on Systems and Control: Young Generation Viewpoint*, 2008.

[5] O'Shea, Keiron, and Ryan Nash. “An Introduction to Convolutional Neural Networks.” *Neural and Evolutionary Computing*, 2 Dec. 2015.

[6] Workman, Scott, et al. “DEEPFOCAL: A Method for Direct Focal Length Estimation.” *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, doi:10.1109/icip.2015.7351024.

[7] Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” *Communications of the ACM*, vol. 60, no. 6, 2017, pp. 84–90., doi:10.1145/3065386.

[8] “Comparison of AI Frameworks.” *Skymind*, skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch.

[9] “Keras vs TensorFlow vs PyTorch | Deep Learning Frameworks.” *Edureka*, 22 May 2019, www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/.

[10] Sarkar, Dipanjan. “A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning.” *Medium*, Towards Data Science, 14 Nov. 2018, towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[11] “Applications.” *Keras*, keras.io/applications/.

[12] “CS231n Convolutional Neural Networks for Visual Recognition.” *GitHub*, cs231n.github.io/convolutional-networks/.

[13] Tsang, Sik-Ho. “Review: ResNeXt - 1st Runner Up in ILSVRC 2016 (Image Classification).” *Medium*, Towards Data Science, 9 Dec. 2018, towardsdatascience.com/review-resnext-1st-runner-up-of-ilsvrc-2016-image-classification-15d7f17b42ac.

[14] Tsang, Sik-Ho. “Review: Xception - With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification).” *Medium*, Towards Data Science, 25 Sept. 2018, towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568.
[15] Tsang, Sik-Ho. “Review: Inception-v4 - Evolved From GoogLeNet, Merged with ResNet Idea (Image Classification).” *Medium*, Towards Data Science, 27 Sept. 2018, towardsdatascience.com/review-inception-v4-evolved-from-googlenet-merged-with-resnet-idea-image-classification-5e8c339d18bc.
[16] Tsang, Sik-Ho. “Review: MobileNetV1 - Depthwise Separable Convolution (Light Weight Model).” *Medium*, Towards Data Science, 14 Oct. 2018, towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69.
[17] Tsang, Sik-Ho. “Review: DenseNet - Dense Convolutional Network (Image Classification).” *Medium*, Towards Data Science, 25 Nov. 2018, towardsdatascience.com/review-densenet-image-classification-b6631a8ef803.
[18] Tsang, Sik-Ho. “Review: NASNet — Neural Architecture Search Network (Image Classification).” *Medium*, Medium, 18 May 2019, https://medium.com/@sh.tsang/review-nasnet-neural-architecture-search-network-image-classification-23139ea0425d
[19] “Epipolar Geometry.” *OpenCV*, docs.opencv.org/master/da/de9/tutorial_py_epipolar_geometry.html.
[20] Rosebrock, Adrian. “Find Distance from Camera to Object Using Python and OpenCV.” *PyImageSearch*, 19 Jan. 2015, www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv.
[21] Hirschmuller, Heiko. “Stereo Processing by Semi-Global Matching and Mutual Information.” *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 16 Apr. 2007.
[22] “About.” *Node.js*, nodejs.org/en/about/.
[23] “Overview of Blocking vs Non-Blocking.” Node.js, nodejs.org/en/docs/guides/blocking-vs-non-blocking/.
[24] “The Node.js Event Loop, Timers, and Process.nextTick().” *Node.js*, nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/.
[25] “What Are Callbacks?” *Node.js*, nodejs.org/en/knowledge/getting-started/control-flow/what-are-callbacks/.
[26] “Nvm-Sh/Nvm.” *GitHub*, 17 July 2019, github.com/nvm-sh/nvm.
[27] “Nullivex/Nodist.” *GitHub*, 29 Mar. 2019, github.com/nullivex/nodist.
[28] “Downloading and Installing Node.js and Npm.” *Npm Documentation*, docs.npmjs.com/downloading-and-installing-node-js-and-npm.

[29] Shaleynikov, Anton. “Node.js vs. Django: Is JavaScript Better Than Python? - DZone Web Dev.” *Dzone.com*, 30 Nov. 2018, dzone.com/articles/nodejs-vs-djangois-javascript-better-than-python.
[30] “Download.” *Django*, www.djangoproject.com/download/.
[31] Stanton, Brendan. “What Is Express.js and Why Does It Matter?” *ProgrammableWeb*, 5 May 2017, www.programmableweb.com/news/what-expressjs-and-why-does-it-matter/analysis/2017/05/05.
[32] “Installing.” *Express*, expressjs.com/en/starter/installing.html.
[33] “React – A JavaScript Library for Building User Interfaces.” *React*, reactjs.org/.
[34] “MySQL Introduction.” *Tutorialspoint*, www.tutorialspoint.com/mysql/mysql-introduction.htm.
[35] “About Docker Engine - Community.” *Docker Documentation*, https://docs.docker.com/install/.
[36] “Dockerfile Reference.” *Docker Documentation*, https://docs.docker.com/engine/reference/builder/.
[37] “Overview of Docker Compose.” *Docker Documentation*, https://docs.docker.com/compose/.
[38] “Docker Run.” *Docker Documentation*, https://docs.docker.com/engine/reference/commandline/run/.
[39] “Docker Pull.” *Docker Documentation*, https://docs.docker.com/engine/reference/commandline/pull/.

APPENDICES

Emails to the DeepFocal Team

Christopher Dowlatram
Sat 6/22, 10:35 PM

Good afternoon,

My name is Chris Dowlatram. I am an undergraduate computer science student at the University of Central Florida. My classmates (whom I've CC'ed in this email) and I are currently working on a project for our senior design class involving distance measurements between objects in old photographs (Civil War era). We already have the known size of some objects in the photographs but are lacking other parameters that are needed to do accurate distance estimation. One of those parameters is focal length. We came across your DEEPFOCAL paper and were greatly intrigued by it. We were interested in applying that to our project (utilizing a CNN trained on images with known focal lengths to estimate unknown focal lengths in other images). We were wondering if you had any advice on how to approach this in terms of code, resources, etc.? In addition, we were wondering if you would be willing to share the code on how you created, trained, and tested the CNN? We would obviously credit you all in our design document. Thank you.

V/R,

Chris Dowlatram

Jacobs, Nathan <nathan.jacobs@uky.edu>
Sun 6/23, 9:04 AM
Christopher Dowlatram; Andres Santamaria; Steven Calderon; Thomas Whitaker

Chris,

To my knowledge, we haven't released any code for this project. Let me check w/ my co-authors to see how hard it would be to release the inference code (which was probably PyCaffe or MatCaffe). I don't think we will want to put the energy into preparing all the code for sharing.

Regardless, I don't think the results will be of much use to you given that the imagery you are working with is so different from the training data. The focal length error could be very large! Also, a lot has changed since 2014... so the results could probably be easily improved using newer networks and training strategies.

Cheers,

- Nathan

Christopher Dowlatram
Sun 6/23, 1:06 PM

Hi Nathan,

Thank you so much for taking the time to respond. I had a feeling in the end my team and I would most likely have to acquire the training data and develop the network ourselves. We're dealing with stereographic images so we were thinking of creating our own set of stereographic images from scratch with known measurements then using this to train a model which would hopefully be able to predict focal lengths as you guys did. You said that newer networks and training strategies since 2014 could be used. Is Caffe a little dated? Would Keras, Tensorflow, or another framework be more applicable to what we are trying to do?

Chris

Jacobs, Nathan <nathan.jacobs@uky.edu>
Mon 6/24, 8:41 AM

Most CV researchers working on deep learning methods have moved on to PyTorch or Tensorflow (maybe w/ Keras).

There is beautiful geometry in the problem you are working on. Just throwing a deep network at it might be ok as a prior but will likely not work as well as geometric constraints if they are available in the image. In case you haven't seen it, you should read the Single View Metrology paper and work out from there.

Good luck!

- nathan

Emails to Flickr Support

Flickr Support <help@flickr.com>
Fri 7/12, 3:11 PM
Thomas Whitaker

Thanks for reaching out, and sorry for the trouble. We're hard at work building the ability for folks to sign up for a new Flickr account without needing to create a Yahoo account, and we're close to completion. We're investigating an issue with Yahoo Login that's currently affecting new members such as yourself, but for the moment I would suggest that you hold off on trying to create a new account on our current system. Very soon, you'll be able to sign up for Flickr without a Yahoo account.

I'll update you as soon as I hear that this is out, and you'll be able to sign up again with your email address.

Best,
Katherine

Flickr Support <help@flickr.com>
Fri 7/19, 11:45 PM
Thomas Whitaker

Hi there,

Thanks for your patience. New sign-ups are now live, and you can now sign up for an account at https://identity.flickr.com/sign-up. You should be able to use the email you originally used when you tried to create an account.

Please let us know if you have any other questions or issues, we'll be happy to help.

Best,
Flickr Support

Emails to UCF Center for Distributed Learning

Thomas Whitaker
Mon 7/22, 3:55 PM
Ryan Seilhamer

Hey Ryan,

About a month ago I asked you if I could use some of the CDL phones for my Senior Design project, and you said that would be okay. Since the time that we would need those phones is approaching, I wanted to give you some information about how we would be using those phones and ask some questions about our limitations and responsibilities.

We would be using the phones as part of a custom stereoscopic camera, where each phone would act as one of the cameras in the device. They would be connected via USB or network to a laptop that would send a command to take a picture on each phone simultaneously, which would then compose a stereographic image. The phones themselves would be mounted using a standard phone tripod mount, so there would be no worry for physical damage. We might need to install an app from the app store, this part is still being researched and I'll try to get back to you on that soon.

I expect that we would only be borrowing one phone that matches one of the phones we already have. We wouldn't need to borrow it for long, most likely it would be for a few afternoons for development/testing, and then for a day trip while we take our photos.

If this is all acceptable, we would like to use the phones starting in August. Please don't feel obligated to lend us these phones if that usage causes you undue liability, this equipment isn't mission-critical for our project and we have alternatives. However, access to one of the phones would be especially beneficial. If you have limitations on our usage of the phone, such as when we could borrow it and where we could take it, please let me know.

Thank you for your offer to help us with our senior design project!

Thomas Whitaker

Ryan Seilhamer
Tue 7/23, 3:59 PM
Thomas Whitaker

HI Thomas,

Sure, this sounds good. What kind of phone do you need?

We currently have:

- iPhone 6s Plus
- iPhone 8
- Nexus 6p
- Samsung S9

We also have some iPads, if that helps.

Thanks,

--
Ryan Seilhamer
Asst. Director of Mobile Strategy & Innovation
University of Central Florida
digitallearning.ucf.edu/msi
ucfmobile.ucf.edu