University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

QuickScan

Revised Detailed Design

Group 2022.20

Prepared by

Eric Jasin Li - 20669325
Bosco Han - 20651352
Yingjian Bai - 20666928
Michael Han - 20665040
Daivik Goel - 20649169
Lichen Ma - 20649708
Consultant: John Zelek

26 July, 2021

Table of Contents

1. High Level Description

1.1 Motivation

In 2020, over 80 million people of consumers in the United States used AR monthly[1]. In 2026, the 3D rendering market is expected to be worth 9 billion dollars[2]. Demand for 3D objects is rapidly increasing as more developers and creators are looking to create new experiences. We are just scratching the surface of the possibilities with XR and 3D printing while more traditional 3D experiences like animation are seeing a huge wave of new creators. We see that for many people in this space that they use photogrammetry as a means to scan objects into 3D. Although this method is accessible, there are major issues which make it time consuming and prevent its widespread adoption. One of them is that to use photogrammetry, the users have to take many photos at different angles resulting in a lengthy and difficult experience. There are many forums and users we talked to that had these concerns and we thought we had a challenging but reasonable way to help simplify the process.

1.2 Objective

The objective of this project is to design an application which simplifies the process of capturing these photos and their subsequent rendering into 3D objects. We will achieve this by offering a solution that involves the user taking an assisted video on a smartphone and uploading it from where we will process the video and return a 3D object for their use. This gives them a simple method that handles all the complexity in the cloud. We also plan to build a website alongside this where the user can see both their creations and the creations of other people. In that way we allow our users to easily find assets to use in their projects. We believe that our project will reduce the barriers of accessing photogrammetry and aid the growth of the 3D community.

1.3 Block Diagram

The proposed solution is a web application with a front end user interface paired with a backend system for handling database interaction and business logic. Separate systems were introduced for handling large isolated subtasks such as user authentication and 2D/3D object processing. The architecture is outlined in **Figure 1** and is composed of six main subsystems: User authentication subsystem, Website application subsystem, Mobile application subsystem, Backend core server subsystem, 2D image processing subsystem, and 3D object processing subsystem.

1.3.1 Website Application Subsystem

The client system contains the user interface (UI) of our application which handles user inputs and displays user requested information. The user will be able to interact with the front end system through the following actions: They can sign into their account, search and browse 3D objects, and view their own 3D objects. This subsystem is responsible for making API calls to the backend subsystem to fetch user requested information to display on the client side

1.3.2 Mobile Application Subsystem

This subsystem gives a user interface to have the user record an assisted video. This video will then be sent to our backend core system to be processed. To browse the objects we currently plan to have abn

1.3.3 Backend core server subsystem

The core server subsystem is responsible for hosting the external facing APIs of our application. It will link together with the authentication, object processing and image processing subservices and will be the point of contact for all requests from the client side subsystem. The server subsystem is also responsible for interacting with a PostgreSQL database to store user and asset information.

1.3.4 User Authentication

The user authentication subsystem will handle the authentication process for users accessing our web service. We will be building upon the authentication services provided by firebase to support social login as well as username and password login.

1.3.5 3D object processing subsystem

The 3D object processing subsystem is responsible for converting images uploaded by the users into 3D objects. This system will be built upon the photogrammetric open source framework released by AliceVision. The generated 3D objects will be stored by this subsystem and its location will be returned to the backend core server.

1.3.6 2D image processing subsystem

This subsystem is responsible for converting the taken video into frames and removing frames that will be detrimental to processing. It also has image recognition for automated taggs in addition to storing the 2D image assets in an Amazon S3 container and returns its location to the backend core server.

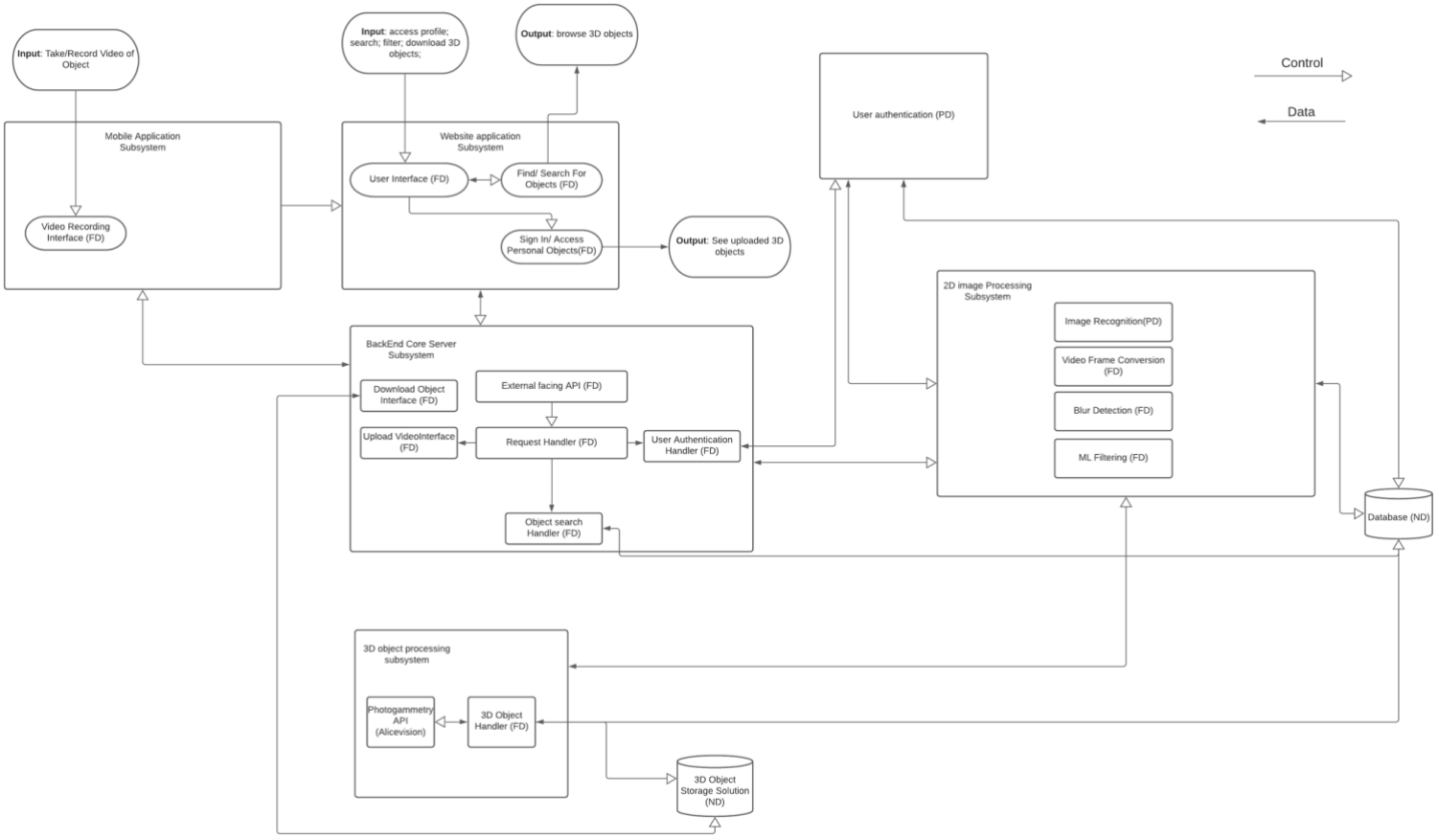

Figure 1: Block diagram breakdown of application software components

2. Project Specifications

2.1 Functional Specifications

Table 1 lists functional specifications including a description, their necessity and an identifiable ID.

Table 1: Functional specifications

Subsystem	Specification	Description	Necessity	Identifier
Client Side Application	Download 3D object files	Users shall be able to successfully download any 3D object file available in the database.	Essential	DNLD
	Search and filter 3D object files	Users shall be able to search for any 3D object file as well as filter them by specific tags,, most recent, and creator.	Essential	SRCH
Mobile Application	User log in	Users shall be able to successfully log in to our website within 2 minutes given a valid social login or valid email and password.	Essential	LGN
	Edit 3D object metadata	Users shall be able to edit the tags and description for any 3D object they have uploaded as well as delete the whole object.	Essential	EDIT
	Assisted Video	Users shall be guided through recording a video to be processed into a 3D object. There will be on-screen instructions on how to best take the video in order to improve the quality of the 3D object.	Essential	AVID
2D Image Processing	Automatic Tag Suggestion	A collection of 2D images will be automatically recommended tags using image processing. This will be done through the Google Vision API.	Non-essential	TAG
Database	2D / 3D object store	The database shall store all 2D images and 3D objects along with the description and tags. The database shall delete any 2D image or 3D object at the discretion of the author.	Essential	STOR
3D Object Processing	Create 3D object	A video shall be processed into a 3D object with a .OBJ file extension.	Essential	CREA

2.2 Non-Functional Specifications

Table 2 lists non functional specifications including a description, their necessity and an identifiable ID.

Table 2: Non-Functional specifications

Type	Description	Necessity	Identifier
Performance	The system should process the video input into a 3D object in a reasonable time	Essential	LTCY
Durability	The system should be durable, 3D files should not be lost	Essential	DURA
Availability	The system should be highly available and have fallback plans	Essential	AVAL
Scalability	The system should be able to support a high number of active users	Essential	SCAL
Consistency	Users should not have to wait for a file to load for a long time	Non-essential	CONS
Ease of access	Users should be able to sign in using their google account. The difficulty to record an assisted video should not be high.	Non-essential	EASE
Compatibility	Users should be only able to upload videos of compatible types.	Essential	COMP

3. Detailed Design

3.1 User Authentication Subsystem

The objective of the user authentication subsystem is to meet the application functional requirement of supporting user login. In addition, the system also aims to handle the ease of access non functional specification by supporting login using a google account. For this project, using complete in-app authentication implementation is not an option due to the availability of more reliable and easy to use third party solutions. As a result only secure solutions which integrate with third party solutions are considered. The available choices for third party authentication systems are Firebase, Auth0 and AWS Cognito. The major priorities are to minimize cost, maximize flexibility and a strong emphasis on ease of use.

First, in terms of cost, the objective is to pick a solution which has the largest free tier in order to minimize expenses for this system. All three solutions offer a free tier of which Cognito is the most generous - Auth0 supports free 7,000 monthly logins, compared to 10,000 for Firebase and 50,000 for Cognito [6].

Next, for flexibility, the objective is to pick a solution which supports many authentication methods and has the least number of restrictions on its use. All solutions are compatible with google social login which

addresses the ease of access non function specification for this project. However, in terms of flexibility Cognito lags behind Firebase and Auth0 due to many of its limitations: user attributes to collect from registering users can not be modified, limit to number of custom field attributes.

Finally, in order to devote as much development time as possible to the 3D object handling components of this project, there is a strong emphasis on ease of use for the third party authentication system. In this category Firebase and Auth0 stand out as the most appealing options and Cognito as the most unappealing. The large amount of online documentation and community support for Firebase Auth and Auth0 makes it easy to work with while the opposite is true for Cognito. In addition, many engineering team members have previous experience with Firebase Auth making it a more familiar and appealing framework to work with.

The options are outlined and compared in **Table 3:**

Table 3: Decision Matrix for Third Party Authentication System

Option	Cost	Flexibility	Ease of use	Total
Firebase	7/10	9/10	28/30	44/50
Auth0	5/10	9/10	25/30	39/50
Cognito	10/10	2/10	20/30	32/50

After considering the alternatives, Firebase scores the highest due to its flexibility and ease of use and will be the framework for the user authentication system.

3.2 3D object processing subsystem

The goal of the 3D object processing subsystem was to allow our users to be able to easily take objects and upload them onto the website. As seen in our motivation, we want to enable the next generation of developers/creators to take advantage of open source 3D assets. In order to achieve this we needed a solution that would help us work towards this.

We started research on how we could go about allowing our users to do this. Although there were some experimental methods it seemed as if the most proven and best method were to look at a framework that does photogrammetry for us.

The biggest alternative to Photogrammetry were programs that use LiDAR for object capture. However upon further investigation it became clear that Photogrammetry had some distinct advantages that made it the most compelling. Where LiDAR based object capture typically does give better results, it falters in terms of flexibility and hosting. Currently LiDAR is only equipped on iPhones (from 11 and up) and thus we would be limited to a select group of users that have this. Given our mission to enable more open source assets this would take away the ability from a large percentage of users from uploading their objects. Another major issue is the availability of frameworks/applications that employ the method. Where in Photogrammetry there are many frameworks from reputed developers, LiDAR object capture was limited to a few applications available on the iOS app store. This lack of developer support and hindrance in terms of hosting are a major concern and a takeaway from this method.

Considering all these major aspects we decided to go with Photogrammetry as the method to employ in our 3D object processing subsystem.

3.2.1 Photogrammetry

Photogrammetry is officially defined as "the art and science of extracting 3D information from photographs. The process involves taking overlapping photographs of an object, structure, or space, and converting them into 2D or 3D digital models."[3] Choosing a framework to perform this task would allow our users to upload images to our server and have a 3D object returned for them. This fits into our criteria of something simple to allow our users to easily create and upload 3D objects for open source purposes.

3.2.1.1 Structure from motion technique (SfM)

SfM is a technique that utilizes a series of 2D images to reconstruct the 3D structure of an object. It is based on the same principles as stereoscopic photogrammetry.

To create a 3D reconstruction, many images of an object with a high degree overlap, taken from different angles. SfM algorithms take a set of images and produce two things: camera parameters of every image and a set of 3D points visible in the images which are encoded as tracks. A track is defined as the 3D coordinates of a reconstructed 3D point and the list of corresponding 2D coordinates in a subset of the input images. Most SfM algorithms share the same processing pipeline seen in **Figure 3** [5].

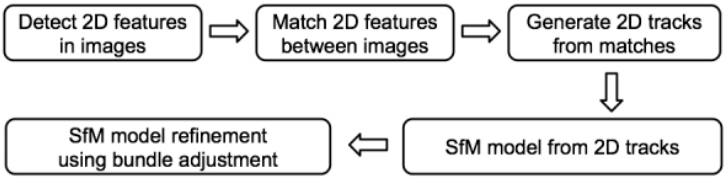

Figure 2: Example SfM algorithm processing pipeline

The advantage of SFM is that Images can be collected with standard consumer-grade cameras, making it a low-cost tool that compliments other 3D technologies, such as lidar. Some of the frameworks listed below such as AliceVision and Colmap use this technique in their pipeline.

3.2.1.2 Multi-Stereo reconstruction (MVS)

The Multi-View Stereo system provides an accurate and efficient dense modeling from unstructured image collections. MVS leverages multiple views to overcome the inherent occlusion problems of two-view approaches. Different applications may use slightly different implementations of MVS, but the general idea is similar: With a collection of images, compute camera parameters and meta information for each image collected. Next, reconstruct the 3D geometry of the scene from the collection of images and their respective camera parameters. In applications like Colmap, MVS takes the output from SfM to compute depth information for each pixel in the image. It is then followed by a blend of the depth and normal maps for multiple images in 3D which produces dense point clouds of the object [8].

3.2.2 Photogrammetric Frameworks

Now that it was decided that we were going to do Photogrammetry, we needed to decide on a framework. Below is a decision matrix highlighting the criteria we had for this.

3.2.3 Decision Matrix

In our research we finally landed on three frameworks to help us do this.

Table 4: Weighted Decision Matrix on Various Photogrammetric Frameworks

Method	Accuracy	Support for CLI	General Performance	Developer Support	Score
MacOS 12 Photogrammetry API	The results we got were a good 3D representation of the and looked very accurate Score: 15/15	We will need to host a VM Mac on the cloud on the latest version of MacOS. It does have support as a CLI granted its running MacOS in a specific configuration The API also releases the object into an OBJ file which means we do not need to convert it. Score: 6/10	The API seems to handle multiple object types in different conditions quite well and give an accurate reconstruction. It also did it without having to tailor the results. Score: 10/10	Although this is still in Beta in the latest MacOS, it is a big feature Apple is touting and many are using it. This should make the framework stable and help us get developer support Score: 15/15	46/50
Alice Vision	With our test photo sample we got a decent result with some erroneous fragments. Tailoring the parameters may help but we want a generalized solution. Score: 15/15	AliceVision does support command line and will be easy to host on the cloud. The API releases the object as a PLY or OBJ files which is desirable Score: 10/10	AliceVision has various nodes that it uses to process the images that you can tailor to your liking. Without doing this however performance with different items can be highly varying Score: 6/10	AliceVision is a widely used open source framework created by reputable academics in Europe. In turn there is a lot of community support and the program is highly supported by the developers Score: 15/15	46/50
Colmap	Colmap is a program created by researchers that includes many highly advanced options that may not be mandatory for simple 3D object generation. For example, extensively defining the type of camera used to capture the imagery isn't necessary. Finally, Colmap only exports a 3D mesh, another program is still needed	The CLI provides access to all of COLMAP's functionality for automated scripting. Score: 10/10	Colmap's pipeline is comprehensive, it takes image input and generates sparse/dense/mesh results. It is better than the others at smoothing out featureless surfaces. However, the time required to process is about the same compared to MVE. Score: 4/10	The colmap website describes every functionality offered in detail and features such as FAQs and tutorials for set up. Although, its online community presence is somewhat lacking Score: 9/15	30/50

	to clean up and refine the mesh. Score: 7/10				
Multi-View Environment	Accuracy for MVE is adequate. It offers SfM, MVS, surface reconstruction and texturing. It allows for reconstruction of large datasets containing some detailed regions with higher resolution than the rest of the scene. Score: 8/10	MVE does support CLI for linux and windows. Thus hosting it on any cloud service provider is no issue. Score: 10/10	Multi-view environment took a long time to render. With 40 high quality images, the render time was around 20 minutes on a personal machine. Hosted on the cloud, this might be quicker, but certainly won't complete renderings instantly. Score: 3/10	The only support we were able to find online were from the github repositoryfor the project.The website contains detailed information regarding the detailed design of this software, not enough support can be found from online forums. Score: 7/15	28/50

3.2.3.1 Decision Matrix Results and Discussion

Based on the decision matrix, we ended up deciding to use Alicevision's Photogrammetry framework with Apple's being a close tie. Their Accuracy, Developer Support and General Performance made them stand out compared to the rest of the matrix. However, AliceVision stood out more to us as it will be easier to host and scale. If we have any difficulties in actual development, we are prepared to switch to a Mac solution if required.

In terms of accuracy, there is a large discrepancy between AliveVision and Apple vs the rest of the options. While some alternatives had acceptable outputs, nothing came close to the level of detail that Alicevision and Apple was able to return us. Apple was the most accurate however Alicevision was quite close.

Apple's support for CLI was there but lacking compared to other options. Given it will have to be run on Mac we will need to specifically choose a Mac Instance that is hosted in the cloud and make scripts that handle receiving an item, processing it and then sending it forward. This Mac Instance will have to be on MacOS Monterrey. This is expensive compared to hosting AliceVision in the cloud which can easily be done with a Linux or Windows Server. It also gives more flexibility in terms of scaling.

The general performance of Apple's API was quite impressive with it handling a range of items and returning fairly detailed 3D objects without high variance depending on the type. This probably is due to the resources Apple has in developing their technology compared to traditional frameworks. AliceVision on the other hand does need more specified tuning for different object ranges. We will try tuning the nodes as possible but it does not seem to be nowhere near the ease of Apple.

AliceVision and specifically Meshroom is used by many creators for their modelling work and thus, there are lots of community support online whether it be how to apply filtering on the UI or how to automate the entire process via a terminal. It also has tons of support online from forum websites, and stackoverflow.

Apple on the other hand is a huge organization, their implementation is relatively bug free and has a lot of

community support. The framework is expected to improve as MacOS Monterrey gets closer to production and more feedback comes to Apple. Both Apple and AliceVision have top tier support which is why we gave both full points for this section.

3.2.4 Code Implementation

Development of the 3D processing subsystem can be automated using a linux terminal. This will allow us to bypass dealing with the UI to initiate each processing. Below is a snippet of the code of meshroom running on jupyter notebooks in Google Colab using Meshroom hosted on linux. The input paths of the photos directory and the output paths simply need to be specified and we would be able to run our custom processing for the photos passed in on colab on any linux environment hosted on cloud.

```
image_folder_path = "" #@param {type:"string"}
output_path = "" #@param {type:"string"}
!wget -N https://www.fosshub.com/Meshroom.html?dwl=Meshroom-2021.1.0-linux-
cuda10.tar.gz
!mkdir meshroom
!tar xzf Meshroom-2021.1.0-linux.tar.gz -C ./meshroom
!mkdir '{output_path}'
!./meshroom/Meshroom-2019.1.0/meshroom_photogrammetry --input '{image_folder_path}' --
output '{output_path}'
```

Figure 3: Current Code Implementation in Google Colab Testing

When developing with Alicevision, we will have to use Meshroom's compiled tar file in a linux environment. Once we install it we will then have access to the Photogrammetry pipeline provided by AliceVision. Once initiated, the pipeline will take the 2D images provided through a series of steps. First, natural feature extraction is applied. In this step, distinctive groups of pixels are extracted if they are invariant to changing camera viewpoints during image acquisition. The SIFT (Scale-invariant feature transform) technique applied here. With a purpose to extract discriminative patches in one image that can be compared to discriminative patches in another image, regardless of rotation, translation, or scale. The extracted patches are centred at stable places of interest since a meaningful feature only exists at a specific scale. The essential idea is that, to some extent, the SIFT invariance may be used to deal with image alterations that occur when views change during image acquisition.

The next step in the pipeline is known as image matching. With an objective to find images that are looking at the same areas of the scene. The way this works is to simplify each image in a compact image descriptor that computes the distance between all image descriptors efficiently. A vocabulary tree approach is a common method for generating this picture descriptor. By feeding it all of the retrieved feature descriptors, it creates a classification by comparing them to the descriptors on each node of the tree. Each descriptor is stored in a single leaf that can be indexed. And the collection of used leaf indices represents the picture description.

Step 3 in the pipeline is features matching. The collection of descriptors from the two input photos are matched photometrically. With candidate features in picture B for each feature in image A. We can't rely on absolute distance measurements to determine whether or not a match is genuine because the descriptor space isn't a linear and well-defined space (we can only have an absolute higher bound distance). We assume

there is only one legitimate match in the other image to eliminate bad choices. Thus the two closest feature descriptors for each feature descriptor on the first image and use a relative threshold between them for each feature descriptor is checked. This assumption eliminates repeating structure features, and has proven to be reliable. Random Sample Consensus is then used to build a geometric filtering.

After features matching, structure from motion (SFM). In SFM, corresponding features are matched, and distances between them measured on the camera image plane. When matching locations of multiple points on two or more photos are found, individual camera positions (x,y,z), (x',y',z'), orientations i, i', focal lengths f, f' and relative positions of corresponding features are calculated, this is the bundle adjustment. 'Structure' refers to all these parameters and 'motion' refers to the movement of the camera. Next, a dense point cloud and 3D surface is determined using the camera parameters and using SFM points.

After SFM, the next step in the AliceVision pipeline handles depth maps estimation. From all the camera points gathered, a depth value of each pixel is retrieved. The algorithm chooses the N best/closest in the area for each image. The intersection of the axis with the pixels of the selected camera is used to determine fronto parallel planes. This results in a volume W, H, and Z with a large number of depth possibilities per pixel. The degree of similarity for each of the pixels is calculated with a method known as the zero mean normalized cross-correlation (ZNCC). This creates a volume of similarities. For each neighboring image, similarities are accumulated into this volume.

The last few steps that are part of the AliceVision pipeline are meshing, texturing and localization. It is in theses steps that makes the scanned object look quite identical to their real life counterparts. Since these steps are responsible for creating a dense geometric surface representation of the scene.

Finally, we can obtain the result or make updates to the request by directly accessing the output directories we had allocated before initiating the process to access the 3D objects.

3.2.5 Hosting Implementation

We plan to have a hosted Meshroom instance in the cloud in order to send our transactions. In looking for a host we considered various options but decided on going with AWS to keep uniformity with the rest of our systems.

AWS has EC2 instances available for purchase as dedicated hosts. From the AWS website, pricing information is shown. On-demand, 1 year and 3 year savings plans are offered. With on demand pricing as low as $1.083 by the hour [7]. AWS also has GPU accelerated instances which will be what we need for quick processing of the files. This makes it very appealing to us.

The actual photogrammetry pipeline will be in the form of a UNIX executable.
Thus we will need to build a REST API to get the images from our server, save them locally, execute this code and then take the result to transmit back. To keep consistency we will use Node.js to interface between the different subsystems and embed a python script in order to execute the command.

Pseudo Code for this would look something like the following:

```
router.get('/', function(req, res, next) {
//get the images location from other services
Imagefolderlocation = req.query.imagelocation
//take images location and run the Photogammetry pipeline
Python.run(Photogrammetry.py, imagefolderlocation)
        poll(outputfolder){
        //check if an obj file was created
        if(file.ending == .obj){
                3Dobject = file
}
//take the 3D object and upload it to AWS, so other services can get
AWSLocation = uploadtoAWS(3Dobject)
//post this location to the other services
router.post(AWSLocation)
 });
```

Figure 4: Pseudocode for Python script

This subsystem is quite complex but will play a vital role in our project. The photogrammetry pipeline hosted on a Linux instance with a Node.js REST API interfacing will be a challenge but one that seems necessary given the benefits that this pipeline offers us when creating 3D objects.

3.4 Video processing subsystem

The purpose of the 2D image processing subsystem is to address the application functional requirements of the conversion of video to images, 2D image storage and automatic tag suggestion support. This section is further divided into the video to image converter, image recognition and classification system as well as the 2D image database to handle each of the above functional requirements. In addition, the system also aims to provide these functionalities while considering the non-functional specifications of performance and scalability.

3.4.1 Video to Image Converter

The goal of this subsection is to take the uploaded assisted video and convert them into frames that we shall use for our photogrammetric pipeline. After consideration, we decided to achieve this through using a python library to convert the video into frames and then using a combination of libraries and a machine learning model to delete erroneous frames from the result.

3.4.1.1 Converting Video into Frames

After considering various options we decided to go with a solution that involves the Python Library OpenCV as our way of converting the video into frames. The reason we decided to choose OpenCV is because, beyond it being an easy, reliable framework to do this, it has other functionality such as Blur Detection that we plan to leverage.

Using OpenCV our code of converting the video into frames would look something like this.

```
import cv2
uploadedvideo = cv2.VideoCapture('uploadedvideo.mp4')


def createframe():
    framearray = []
    while uploadedvideo != finished:
        framearray.append(uploadedvideo.createframes(0.5)) # create a frame every 0.5 seconds\

    cv2.write(framearray, .jpg, ./writefolder) # write frames into jpg images in output folder
```

Figure 5: PseudoCode for writing video into frames

This would give us a folder of frames at different time intervals at a constant time apart. We would then need to do some processing on the frames to get rid of non viable candidates.

3.4.1.2 Blur Detection

After the video has been converted into frames we will then do blur detection to get rid of frames which will skew our result. As a result we ventured to find a method to do this and ended up finding the Laplacian Method.

Although it is typically used for edge detection, in our case, as determined with the following research paper [12], we can actually use it to detect blurriness. To do this we take the greyscale version of our images and convolve it with the Laplacian Kernel. The Laplacian Kernel is used to find rapid changes in our image by taking the second derivative of the image on a higher plane.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 6: Laplacian Kernel [12]

We then use the variance of the response to determine the blurriness. Blurry images will fall below a certain threshold value which will be our evaluator. We will go through each image in the folder, evaluate it based on this criteria and then delete or keep it according to the threshold we determine is best from manual testing. From there the images will be selected using our machine learning model.

3.4.1.3 Machine Learning Model to Delete Frames

Finally, once our video has been converted into frames and blur detection has been performed to remove low quality frames, the system will utilize machine learning to remove frames which are not useful for 3D object processing. These frames include ones where the target object is not completely present and not enough features are available to find overlap with other images. To perform this function, the machine learning model performs two small subtasks.

The first task is to generate a score which represents our model's estimate for how useful a frame is. This score will primarily be based on feature overlap - if there is not enough feature overlap between a frame and its adjacent frames, we can remove it since it would not help with processing in the 3D object

subsystem. We will be determining feature overlap using the SIFT (Scale Invariant Feature Transform) technique for feature extraction and FLANN (Fast Library for Approximate Nearest Neighbors) for feature matching. SIFT was determined to be our approach for feature extraction since features extracted using this technique are affected by scale or orientation. This is particularly useful for our project since we will be dealing with frames containing images of one object in many different orientations. There are many other approaches aside from SIFT such as Haris Corner and Shi-Tomasi but none of them possess the scale and orientation invariant properties that we need for our application. The one notable alternative is SURF (Speeded-Up Robust Features) which offers the same benefits as SIFT with an improved runtime but is currently still patented. In order to avoid legal and pricing concerns, SIFT was chosen. For feature matching, FLANN was the approach selected due to its quick execution time compared to other techniques such as brute force matching.

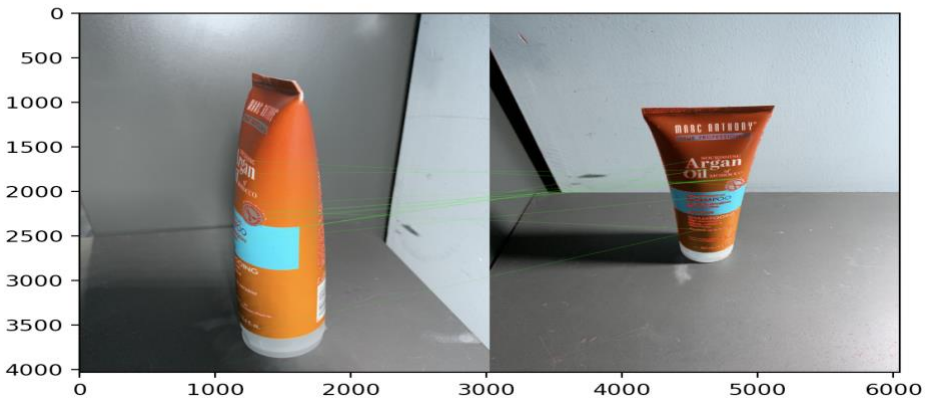

Figure 7: Feature Matching using SIFT and FLANN

The second task, once we have obtained a usefulness score for a frame, is to set a threshold for when we should remove the frame. To determine how low the score should be before we can be confident that it can be removed, we will train our system by trying out different thresholds and evaluating the 3D object returned.

SIFT is quite an involved algorithm. There are mainly five steps involved in the SIFT algorithm:

- Scale-space peak selection: Potential location for finding features.
- Keypoint Localization: Accurately locating the feature keypoints.
- Orientation Assignment: Assigning orientation to keypoints.
- Keypoint descriptor: Describing the keypoints as a high dimensional vector.
- Keypoint Matching

```
sift = cv2.xfeatures2d.SIFT_create()

train_keypoints, train_descriptor = sift.detectAndCompute(training_gra
y, None)
test_keypoints, test_descriptor = sift.detectAndCompute(test_gray, Non
e)

keypoints_without_size = np.copy(training_image)
keypoints_with_size = np.copy(training_image)

cv2.drawKeypoints(training_image, train_keypoints, keypoints_without_s
ize, color = (0, 255, 0))

cv2.drawKeypoints(training_image, train_keypoints, keypoints_with_siz
e, flags = cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

Figure 8: Code for Detect keypoints and Create Descriptor

3.4.2 Image Recognition & Classification

The goal of this subsection was to allow the website to be able to recognize the 2D images uploaded by the users and provide them multiple tags based on the content of the images for selection. We wish to appropriately label those images so that even after converting them into 3D objects, we would still find them conveniently. After consideration, we decided to achieve this through integrating a third party API to automatically label the images as well as allow users to add tags manually.

3.4.2.1 Automatic Tags

One advantage of designing automatic tags is to assist the users to label the images properly since images may consist of many different elements. The other one is to save time because users may need to upload various amounts of pictures from different aspects of different objects. Automatic tags could help to classify and label them quickly.

There are several challenges in designing the automatic tags. For example, how accurate the tags are that third party APIs could provide. We started some investigation and found out the two APIs that could fit our design purpose, which are Google Vision [10] and Clarifai [11], respectively. The following is a decision matrix highlighting the criteria on both APIs.

3.4.2.1.1 Decision Matrix

Table 5: Weighted Decision Matrix on Google Vision API vs. Clarifai API

Criterion	Google Vision API	Clarifai API
Accuracy [1] (how well for different APIs recognizing the images and providing labels)	The tags we got looked very accurate describing the image Million of predefined categories have defined in Vision API Score: 9/10	The tags we got looked very accurate describing the image Score: 9/10
Supported Formats [1] (image formats the APIs support)	Vision API supports almost all image types: JPEG, PNG8, PNG24, GIF, Animated GIF, BMP, WEBP, RAW, ICO, PDF, TIFF Score: 10/10	Clarifai works with the most popular image formats including JPEG, PNG, TIFF, BMP, WEBP Score: 8/10

Image Support [0.5] (different types of image the APIs specified)	Google cloud provided more specific detections including but no limits to images of faces, landmarks, logos, multiple objects etc. It will cover almost every aspects of the image and satisfies what we need Score: 10/10	Clarifai supported general detections on the images It did not specify image in more details Score: 5/10
Financial Cost [0.5] (cost for using APIs to process images)	Free for first 1000 units per month, $1.5 per 1000 units after Score: 7/10	$2 per 1000 units on average Score: 5/10
Evaluation Score	27.5/30	22/30

Evaluation Score for Google Vision: 1x9 + 1x10 + 0.5x10 + 0.5x7 = 27.5
Evaluation Score for Clarifai: 1x9 + 1x8 + 0.5x5 + 0.5x5 = 22

3.4.2.1.2 Decision Matrix Results and Discussion

Based on the decision matrix, we ended up deciding to use Google Vision API. Its combination of Accuracy, Support Formats, Image Support, Financial Cost made it stand out compared to the Clarifai.

Both APIs have very good accuracy on labeling the images. However, Google Vision API is cheaper, supports more image formats and could specify images from faces to landmarks. There is no surprise to this result since Google has strong strength on machine learning and supports Vision API by offering powerful pre-trained machine learning models through REST and RPC APIs. It assigns labels to images and quickly classifies them into millions of predefined categories.

3.4.1.1.3 Code Implementation

When developing this processing subsystem we will have multiple ways since Vision API supports several languages such as GO, Java, and Python. The following code demonstrates how to use Vision API to request information from an image, such as label detection.

```
import io
import os

# Imports the Google Cloud client library
from google.cloud import vision

# Instantiates a client
client = vision.ImageAnnotatorClient()

# The name of the image file to annotate
file_name = os.path.abspath('resources/wakeupcat.jpg')

# Loads the image into memory
with io.open(file_name, 'rb') as image_file:
    content = image_file.read()

image = vision.Image(content=content)

# Performs label detection on the image file
response = client.label_detection(image=image)
labels = response.label_annotations

print('Labels:')
for label in labels:
    print(label.description)
```

Figure 9: Pseudocode for Vision API call

3.4.2.2 Manual Tags

Although the Google Vision API could automatically give users tags precisely, we still implement this manual tags feature for users to manipulate the tags manually. Users should be able to add, delete, change, and check out their images' tags anytime on the website. In this way, the tags could be more precise and creative.

3.4.3 2D Image Database

The 2D image database component is responsible for storing and retrieving user provided 2D images in an efficient and cost effective manner. For this project, storing images in an object format inside our backend application database is not an option due to scalability concerns and the availability of reliable and cheap cloud solutions. As a result only solutions which secure solutions which store images in an easy to access cloud container are considered. The available choices for cloud image storage systems are AWS S3, and Azure Blob storage. The major priorities are to minimize cost and a strong emphasis on ease of use.

First, in terms of cost, the objective is to pick a solution which has the largest free tier in order to minimize expenses for this system. In this category, AWS and Azure are similar with both providing a free tier for twelve months offering five gigabytes of storage and twenty thousand get requests. Azure comes slightly out on top as it provides ten thousand put requests as opposed to two thousand by AWS.

Finally, in terms of ease of use, AWS stands out on top due to its status as one of the oldest and most reputable cloud storage providers. When added to the wide availability of documentation, online support and team member experience with the solution, AWS S3 would be the easiest system for our team to work with.

Given the small amount of options and criteria to consider, a decision matrix was omitted for this section. The final decision was to pursue AWS S3 as our cloud object storage solution due to its ease of use and our

familiarity with the solution. The system will aim to store images in our main PostgreSQL database by linking an S3 container where we can fetch the image assets.

3.5 Client and UI

3.5.1 Front-end Framework

The choice of the framework is very important for our application as it will determine how fast, scalable and responsive it will be. Our framework of choice is React for our web application and React Native for our mobile application. These are open-source Javascript libraries which are widely used in production level code. React abstracts away the document object model (DOM) and provides a simple programming library for developers. By using React, we can create a single-page application which means that the content on the page can be changed without having to reload the page. This provides several performance advantages as well as cutting costs for the server as a new page does not need to be fetched from the server to reflect a change. React Native allows us to create native applications for both android and iOS to achieve a cross-platform solution. Rather than manipulating the DOM, React Native works by interpreting the Javascript written by developers and translating it to native code on the phone, all done seamlessly in the background. This means that the application can be coded in Javascript and it can be built to run on Android and iOS, significantly cutting down on development time.

3.5.2 Front-end language

The language of choice will be Typescript. The choice between Typescript and Javascript boils down to the developer experience as Typescript gets transpiled to Javascript when built. Typescript is a superset of Javascript and it allows for developers to add type information to code. This improves the developers ability to debug as the IDE and build process are able to use type information to check for errors. On the other hand, developers will need to declare types and if a node package we are using does not support types, they will need to be manually declared or an alternative package will have to be used. In the end, Typescript would be a very useful tool as there will be many group members working within the same repository and Typescript can significantly reduce the number of errors caused by working with another person's code.

3.5.3 Client Application Architecture

As previously determined in 3.7.1, React and React Native are our frameworks of choice. The architecture of React and React Native are the same, revolving around components, this also means it is easy to create reusable UI components on both web and mobile platforms. Data gets passed down to child components while actions get passed up to parent components through the use of callbacks. Therefore, our application can be designed in such a way that all necessary data gets fetched from our server as soon as the page loads, that data is then broken up and sent to their respective components. The QuickScan application will have four main components split between the web and mobile application. The first and second component is the catalog page which is also the landing page, and the detailed item page, both found on the web application. The other two components are on the mobile app: the upload object page, and a user login page. The catalog page will contain a list of 3D objects which have been uploaded by other users. These objects will be initially sorted based on how popular they are within a one week period. Additionally, other sorting options will be available as well as the ability to filter by tags. A user can record a video and upload it to be processed into a 3D object on the mobile app. Once the 3D object is available, they will be directed to the upload object page. Here they can enter a title, a description and add tags. Once a 3D object is clicked

on the catalog page, the user will be directed to the detailed item page which will contain more information about the object such as when it was created, and who the author is. Finally, the user login page will allow users to login using a registered email and password or their facebook or google accounts. These four components will be built with Typescript and styled using Cascading Style Sheet (CSS).

There are two main node packages we will be using to help speed up development. One of them is react-router-dom, their API provides several useful functions related to manipulating the uniform resource locator (URL) and browser session history. This allows for smooth transitions between our main components and a better user experience. Another useful package is the aws-sdk package. Their software development kit (SDK) can interface with the Amazon Web Service (AWS) API without the need to manually set headers and authorization. This package will be used to retrieve objects stored in our Amazon Simple Storage Service (S3) such as the 3D objects. By using a filename, a temporary url that expires in a given time can be generated to access these objects safely, in AWS this url is known as a signed url. Another package is the react native camera package. This package is essential for our mobile app as it will allow us to develop the assisted video functionality. Not only are we able to overlay components on top of the camera to provide an interactive user interface, but we can also receive data on what the user is currently recording and use that to assist them in recording a higher quality video, allowing us to produce a higher quality 3D object.

3.6 Back End

3.6.1 Back-end Infrastructure

3.6.1.1 Definition

Behind the Front-end of the website and mobile application, there exists the web server component that serves the user facing component to the user. Since we are using React and React Native as the front-end frameworks, we will also be using the same frameworks and database design for the corresponding back-ends.

It will provide all of the computations required to serve a dynamic and interesting user experience to the user. Things this might include are:

- User Identity (login and authentication).
- CRUD based logic (user navigation throughout the website)
- Connection to databases

We keep in mind scaling to a large amount of traffic and data, and will implement the solution using modern design and architecture.

3.6.2 Back-end Framework

From our team's background and experiences, we have narrowed down the possible back-end frameworks to Spring(Java), Django(Python), and Node.js(JS/TS). Although we all have industry experience in any of the above, we have decided that Node.js would be the easiest and most efficient way of building full-stack apps that are serving the frontend in React and React Native.

The first reason is that by having the entire full stack application being built in the same language, it will lessen the learning curve of our stack. Although we have members with experience in the three backend

frameworks listed beforehand, not all of us have experience in each and every one, and our experience also varies. Therefore, only having to deal with one main programming language is a very large benefit.

The second reason is that some React packages like ReactDOM have specific components that are designed to work with Node.js. This will make it easier to implement abstraction and reduce lines of code. As Node.js is the most popular backend for React apps, it will likely also have the most existing documentation. This will make developing features with existing solutions a much more smooth experience.

Another reason is that Node.js runs on an optimized V8 engine which makes it able to process large quantities of requests asynchronously. Examples of high traffic applications running on Node.js include Uber and Linkedin and PayPal.

3.6.3 Web Server Architecture

The web server is responsible for taking user requests from the front end, and converting them into useful responses. For example, this might include taking a 3D file that a user has selected from our repository, getting that data from a database, and serving it to the user.

There are some options of selecting a virtual private server to host the static content of the website. Some choices are Amazon EC2, Amazon Lightsail, and Digital Ocean. We can take a look at key statistics like http request response time, maximum http load, and cpu performance when comparing the three.

Table 6: Weighted Decision Matrix on EC2 vs Lightsail vs Digital Ocean

VPS	CPU Performance(operations/second)	Network speed(Mbps)	Http request response time(ms)	Maximum http load(request/second)
Lightsail ($20 monthly)	660	770	57	30
EC2 ($28 monthly)	2190	1779	47	25
Digital Ocean ($20 monthly)	710	384	62	20

As the cost of the services are similar, the other metrics should be taken into account when deciding the VPS. Amazon EC2 is generally the winner when it comes to performance.

Users will access the website using a domain, “quickscan.com” routed by a DNS service. Our static content will be hosted in an EC2 instance. The backend of the website and mobile app will be connected to PostgresQL databases that store information like 3D file popularity, user saved images and files, and 3D files. Blob resources such as the input videos and actual 3D files will be stored in Amazon’s Simple Storage Service, S3.

By distributing workload through multiple EC2 instances, it is possible to route user requests by geographical location, reducing latency and response time. The added redundancy also increases fault tolerance and reduces risk of website downtime. The servers will automatically adjust their load

capacities to handle increases or decreases in traffic, real time. This will help maintain efficient performance and reduce unnecessary workload.

Amazon services employ implicit scalability. By adjusting server rates, we can increase server capacity and server quantity. This will allow us to scale horizontally and vertically depending on the growth of the user base and real time traffic.

3.6.4 Database Design

3.6.4.1 Definition

The main product that QuickScan provides is the output data in the form of a 3D object file. Similarly to a web app like Instagram, we have to facilitate the storing of large quantities of information and blob files. We need a query based database that can accurately and quickly return simple information such as file tags, file popularity and file id. Then we will have an Amazon S3 blob storage that will hold the actual .obj and similar files.

3.6.4.2 Query based database design

In modern software infrastructure, databases are typically categorized as relational and non-relational. Relational databases have the advantage of containing certain attributes that relate sets of information. An example of a popular relational database is SQL, which has been adopted into a variety of different servers such as PostgresQL.

Non-relational databases refers to the database design where information is stored in forms, where each column of information doesn't necessarily contain specific attributes. For example, two forms in the same database might contain a disjoint number of rows of information, with completely different contents. Popular "NoSQL" platforms include MongoDB and Cassandra.

For QuickScan's purposes, we need a database with a fixed schema that will make querying data for 3D files and users quick. The benefit to relational databases is that it is known exactly the structure of the information stored in certain tables. This matches the types of data we are storing. The key information our backend requires such as user schemas, files schemas, will have a static structure regardless of possible changes in scaling and scope.

The advantage that non-relational databases bring, primarily the ability to dynamically add differing structures of data to an existing database, does not benefit QuickScan, as the information stored will always comply with the existing database schema.

3.6.4.3 Blob storage

When choosing between blob storages, the options round down to Microsoft Azure, Amazon AWS, and Google Cloud. We are choosing Amazon S3 storage since we are also using Amazon EC2 to host our servers. Amazon services have more seamless integration with each other than with external hosts like Azure or Cloud.

4. Discussion and Project Timeline

4.1 Evaluation of Final Design

The objective of this project is to design a set of applications that facilitates video capturing to simplify the process of rendering 3D objects with photogrammetry. The proposed design in this report meets this objective by implementing a responsive mobile application that assists users in capturing a video along with a server side API endpoint for accessing photogrammetry processing pipeline hosted on the cloud, a database to store the objects, and a desktop website to browse and download processed 3D assets. Overall, the design meets all of the original design specifications.

4.2 Use of Advanced ECE Knowledge

This project uses upper--year ECE knowledge from the courses ECE 356: Databases, ECE 416: Advanced Topics in Networking, ECE 452: Software Architecture, ECE 417: Image Processing and ECE 454 Distributed Systems. Databases are essential in our project as we need them to store various types of information such as the user id, password, video, 2D image locations, 3D object locations, tags and other metadata for our objects. To connect all of our subsystems we need to tap into our knowledge on networking. We will be using HTTP requests all over the network. Some of our APIs such as the Vision API uses our knowledge from Distributed Computing and Networking to communicate through RPC's. Our systems have various architecture styles and we will use our knowledge from Software Architecture to ensure they are structured in the best way. Our 3D and 2D image processing subsystems utilize knowledge from our Image Processing course in order to do various actions on the set of images generated from the video.

4.3 Creativity, Novelty and Elegance

This project is novel because currently, despite many other 3D scanners being available, there are not any platforms that easily allow for someone to shoot a video for easy photogrammetry. The current application uses various cutting edge technologies scattered around and integrates them to create a creative, elegant and unified product. Should our product go to plan, the elegance of simply being able to take an assisted video and then getting a 3D object back without having to take a variety of photos at different angles is unprecedented and can truly be groundbreaking.

4.4 Student Hours

Table 7: Hours each student has worked so far

Team Member Name	Hours each student has worked so far
Daivik Goel	125
Michael Han	120
Bosco Han	122
Eric Li	121
Lichen Ma	128

Yingjian Bai	123

4.5 Potential Safety Hazards

Given that our project is software based, we do not have many (if any) safety hazards. From the perspective of each team member, sitting and coding can be strenuous to our eyes and a lack of physical activity can hurt our wellbeing. So working on this project can present a hazard to us like that. Our way of mitigating this is to make sure we all take 15 minute breaks every hour and get physical exercise whenever possible.

4.6 Project Timeline

Below is our estimated project timeline. Please note that these are subject to change and the dates for 4B are estimated.

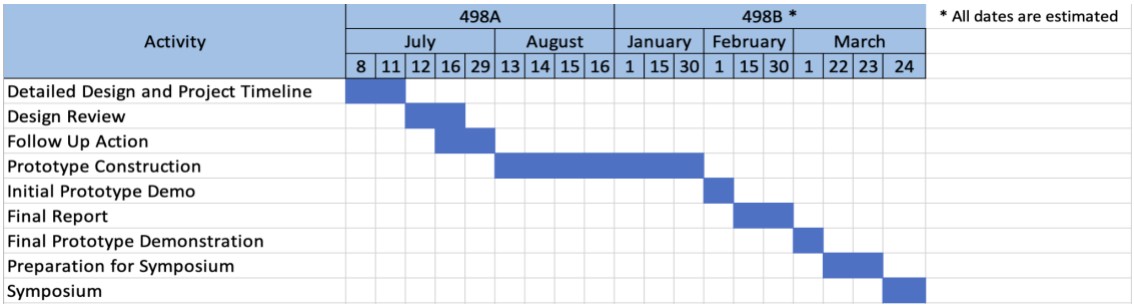

Figure 10: Project Timeline

References

[1] “20 Augmented Reality Statistics You Should Know in 2021,” *Product Configurator and Product Visualization*. [Online]. Available: https://www.threekit.com/20-augmented-reality-statistics-you-should-know-in-2020. [Accessed: 04-Jun-2021].

[2] “3D Rendering Market 2020-2026: Global Statistics Report,” *Global Market Insights, Inc.* [Online]. Available: https://www.gminsights.com/industry-analysis/3D-rendering-market. [Accessed: 04-Jun-2021].

[3] “Photogrammetry Software: Photos to 3D Scans,” *Autodesk*. [Online]. Available: https://www.autodesk.com/solutions/photogrammetry-software. [Accessed: 05-Jul-2021].

[4] N. Lievendag, “Photogrammetry Software,” *3D Scan Expert*. [Online]. Available: https://3Dscanexpert.com/review/photogrammetry-software/. [Accessed: 06-Jul-2021].

[5] Nowpublishers.com. 2021. [online] Available at: <https://www.nowpublishers.com/article/DownloadSummary/CGV-052> [Accessed 6 July 2021].

[6] Google Developers, “Firebase Authentication” [Online]. Available: <https://firebase.google.com/products/auth#:~:text=Firebase%20Authentication%20aims%20to%20make,and%20GitHub%20login%2C%20and%20more.> [Accessed 6 July 2021]

[7] D. J. Daly and D. J. Daly, “Economics 2: EC2,” *Amazon*, 1987. [Online]. Available: https://aws.amazon.com/ec2/instance-types/mac/. [Accessed: 06-Jul-2021].

[8] F. Eickeler, “Modification of Parameters in Image-based Automated 3D-Reconstruction for Fine Structures.” .

[9] “Quickstart: Setup the Vision API,” *Cloud Vision API Documentation.* [Online]. Available: https://cloud.google.com/vision/docs/setup.

[10] “Clarifai Guide,” *Clarifai Documentation*. [Online]. Available: https://docs.clarifai.com/

[11] “VPS Benchmark,” *Compare VPS Proviers*. [Online]. Available: https://www.vpsbenchmarks.com/compare/lightsail_vs_docean

[12] Tbs, A. Rosebrock, “Blur detection with OpenCV,” *PyImageSearch*, 17-Apr-2021. [Online]. Available: https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/. [Accessed: 24-Jul-2021].

Marking Rubric for Detailed Design & Related Discussion					Group: 2022.
	Outstanding (93-100 points)	**Excellent (80-92 points)**	**Good (68-79 points)**	**Satisfactory (50-67 points)**	**Unsatisfactory (0-49 points)**
Complexity of design task*	Project involves the open-ended design of 4 or more subsystems	Project involves the open-ended design of 3 subsystems	Project involves the open-ended design of 2 subsystems	Project involves the open-ended design of 1 subsystem	Project involves no open-ended design
Quality of final design	Design is convincingly shown to meet project objective and satisfy all specifications; highly novel, elegant, and creative; a truly great accomplishment	Design is shown to meet project objective and satisfy all essential and most non-essential specs; novel, elegant, and creative; an excellent design; demanding project	Design is shown to meet project objective and satisfy all essential design specs; some novelty, elegance, or creativity; a good design; moderately challenging project	Design arguably meets project objective and satisfies all essential design specs; some key details are missing; adequate design; relatively simple project	At least one essential spec is not shown to be met or the design is not shown to meet the project objective; or the design is incomprehensible or inadequately described
Use of engineering design process†	A reasonable solution is proposed; thorough investigation of serious alternatives and/or iterations; design process is clearly systematic, comprehensive, and substantial	A reasonable solution is proposed; investigation of serious alternatives and/or iterations; design process is systematic, comprehensive, and arguably substantial	A reasonable solution is proposed; investigation of serious alternatives and/or iterations; design process is systematic and comprehensive	A reasonable solution is proposed; brief mention of alternatives, and/or iterations; some indication of a meaningful design process	Minimal attempt to explain design alternatives or an iterative design process; or the process/project is simplistic, trivial, superficial, irrelevant, poorly explained, or very vague
Quality of justifications†	Arguments are logical, clear, complete, substantial, and sophisticated; supporting citations are complete and authoritative	Arguments are logical, clear, and complete with some degree of sophistication; effective use of citations to support almost all arguments	Arguments are logical and clear with no significant gaps; effective use of citations to support some arguments	Credible effort to justify decisions, but arguments are difficult to follow, unpersuasive, or have many gaps; credible effort to use citations to support arguments	Arguments are missing, minimal, incomprehensible, illogical, faulty, simplistic, superficial, trivial, or vague; citations are missing or poorly used
Level of knowledge used†	Analysis or design clearly uses substantial fourth-year ECE technical knowledge	Analysis or design clearly uses substantial third-year ECE technical knowledge	Analysis or design arguably uses substantial third-year ECE technical knowledge	Analysis or design clearly uses substantial second-year ECE technical knowledge	Analysis and design use little technical knowledge beyond first-year engineering
Breadth and depth of quantitative technical analysis (QTA)†	Detailed and substantial QTA is applied convincingly to all aspects of the design; extensive use of engineering theory, simulation tools, and/or data analysis tools	Detailed and substantial QTA is applied correctly to all important and most minor aspects of the design; major use of engineering theory, simulation, and/or data analysis tools	QTA is applied to multiple important aspects of the design; at least one aspect is detailed; good use of engineering theory, simulation, and/or data analysis tools; possible slight errors	QTA is applied to one important aspect or several minor aspects; credible effort to be detailed and to use engineering theory, simulation, &/or data analysis tools; possibly minor errors/gaps	QTA is missing, minimal, incomprehensible, superficial, trivial, or irrelevant or has major errors/gaps; or the design decisions and design are not amenable to QTA

*For 5-group teams, add one to each of the numbers in this row.

†These components will be assessed for each subsection in Section 3, and then the scores combined.

ECE498A: Marking Sheet for Detailed Design and Project Timeline Document		
Group number: 2022.		**Marks**
Presentation and layout	Follows required formatting, layout, structure, file name: up to 4 marks Correct spelling, grammar, captioning, referencing, units: up to 4 marks Readability, professionalism of language, flow, graphics: up to 4 marks	**/ 12**
Detailed design and related discussion	Score from rubric on page 21 and spreadsheet: up to 60 marks Reasonable identification of potential safety hazards: up to 2 marks	**/ 62**
Project timeline	Realistic plans for design revision, if deemed necessary: up to 3 marks Realistic plans for ECE498B deliverables (Final Report, final prototype demonstration, symposium preparation): up to 3 marks	**/ 6**
Deduction for late submission: (-10 marks if miss deadline, and -10 marks off for every additional 24 hours)		
Deduction for violating file naming, page limit, font size, line spacing, or margins: (-2 marks off for each page or partial page beyond the limit stated on page 18) **Note:** If line spacing is too tight, margins too narrow, or font sizes too small, the equivalent number of excess pages will be computed and marks deducted accordingly.		
Final Mark:		**/ 80**
Additional comments from course instructor:		