# Real-Time Tele-Medicine in Ophthalmology

## Coding Assignment

## Team 7

CS499

November 30, 2019

| Author | Contribution |
|---|---|
| Sam McCauley | 485 |
| Shelby Stocker | 947 |
| Cole Terrell | 1440 |
| Bryce Kushner | 276 |

## Customers

Paras Vora

Dr. Eric Higgins

# Introduction

The overall goal of our project is to digitize the slit lamp viewing experience for ophthalmologists by attaching two 4k cameras to the slit lamp microscope and displaying this video feed locally and remotely. Slit lamp microscopes are an ophthalmology instrument used to allow doctors to inspect a patient's eyes in great detail. For another person to see what the primary doctor is seeing, however, they have to view it themselves through the lenses. Our product seeks to eliminate this limitation entirely, allowing all the personnel necessary to make accurate diagnoses to view the patient's eye simultaneously in real-time no matter where they are in the world. Locally, the video feed is sent to a display connected to a primary computer. This display can be worn in a VR headset or viewed as a normal PC monitor. In order to retain depth perception the two video feeds are split on the left and right sides of the screen. The video feed is streamed across the internet using Twitch and Open Broadcaster Software (OBS) so that it can be viewed by a remote user such as a secondary physician or students.

The primary software tools that we employed to produce this product were OpenCV --an open source computer vision library for C++ and Python -- Python, and OBS/Twitch. All of these tools combine to produce a high-quality video stream/recording that can be viewed any time at any location.

# Implementation

## Source Code Listing

We are using Git/GitHub as version control for this project. The link to our repository is https://github.com/coleterrell97/SeniorDesign.

## Quality Review

For our quality review we sat down and reviewed the overall functionality of our implementation piece by piece. We went through and talked about how we implemented each element (local feed, stream, changing properties) and how satisfied we were with the way we implemented it. In the end we felt as though the approach we took was the most effective one. Early on we encountered a lot of issues in trying to stream the feed ourselves, so making the move to Twitch to host the stream was extremely effective in limiting latency and connection issues. Furthermore, we did have to make some adjustments to how we adjusted the properties of our cameras. For a large portion of the project we weren't using the actual 4K cameras and so we encountered some issues in the end in pulling all of our code together with the actual cameras being used. This mainly had to do with the camera properties that can be changed, the default values for these properties, and how those changes are actually implemented. After some research we were able to get the necessary functionality for these cameras in an effective way. Overall we are extremely satisfied with the product we have put together and the overall quality of our code.

# User's Manual

Our product consists of multiple modules:

**Module 1 (Local Video Stream):**
The first module of our product, the local video stream, is also the most critical. In order to use the software that we have developed, one must have some basic hardware and several software packages that will be described in the Administrator Manual section.

Once one has the requisite hardware and software and the cameras are plugged in, one can simply run the dual_camera.py script, enter the correct sources for the cameras (an integer value likely between 0-4. Typically 0 and 2 are used, but it can be dependent on your machine). This will produce a window that shows the two camera frames concatenated to produce a stereoscopic view. One can view this window on a regular desktop monitor or through a VR headset.

**Module 2 (Camera Settings UI):**
Our source code also includes a python script called app.py that hosts a web page containing sliders and buttons for manipulating common camera settings such as brightness, contrast, exposure, and zoom.

Once Flask is installed, one can run app.py from the command line, and this will host a web page on localhost:80 (although the '80' does not actually need to be typed since port 80 is the default HTTP port for web browsers). For the host machine, the user only has to go to http://localhost to view the settings page. For a separate machine on the *same* WiFi network as the host, they can go to the internal IP address of the host (ex: http://192.168.1.8). For any other machine, they will have to go to the external IP address of the host.

Once one has accessed the web page, one can manipulate the sliders and buttons to his/her liking, press submit, and see the camera properties change in real time.

**Module 3 (Streaming to Remote Viewers via Twitch):**
Once the local video stream and camera settings UI are operational, the user can optionally choose to send the local stream to Twitch for remote viewing. This is done using the open-source encoding tool Open Broadcaster Software (OBS).

Once OBS is installed and the settings are configured (see Administrator's Manual), one will need to configure the scene (that is the image that will be streamed to twitch). This is relatively simple, as one only needs to create a new Display Capture source by pressing the plus button in the Sources pane and selecting the Display Capture option. The following dialogue box will allow the user to select the correct display (if more than one display is connected to the computer). Once this is done, the scene view should mirror the user's selected display. Ideally, the user would have a second monitor dedicated to the local stream and a primary monitor to run OBS and the settings UI.

Lastly, in order to stream to Twitch, one must have a Twitch account. The administrator will provide the Twitch account and the stream key which must be entered in the OBS settings page under Stream. It should be noted that the stream key must be kept private, or else anyone could stream to the account to which the key is attached any time. Once the stream key is entered, the user can press the Start Streaming button, and the stream can then be viewed by anyone via Twitch.

# Administrator's Manual

**Hardware Requirements:** the hardware requirements for our product are two See3CAM 4K USB cameras from Econ-Systems. Our software is developed using an OpenCV patch provided by Econ-Systems, so using other cameras would likely not be stable.

**Software Requirements:** the software requirements for our product are OpenCV v3.4.1 that uses the videoio module that can be found in the GitHub repository at the following link: https://github.com/econsystems/opencv, the Flask microframework, and OBS. This repository also includes documentation for installing OpenCV and building it from its source files. It should also be noted that the most recent commit of this custom patch does not support Python, thus one must use the build from November 4, as this is the latest commit known to support Python. Because we are using a custom OpenCV build, OpenCV must be built from source (rather than using pre-built binaries). The source files can be found at https://github.com/opencv/opencv. Both of these GitHub repositories are excellent resources for determining the requisite software for running Python scripts that use OpenCV. All of our Python software uses Python v3.6 and later. Python v2.x is not supported at all.

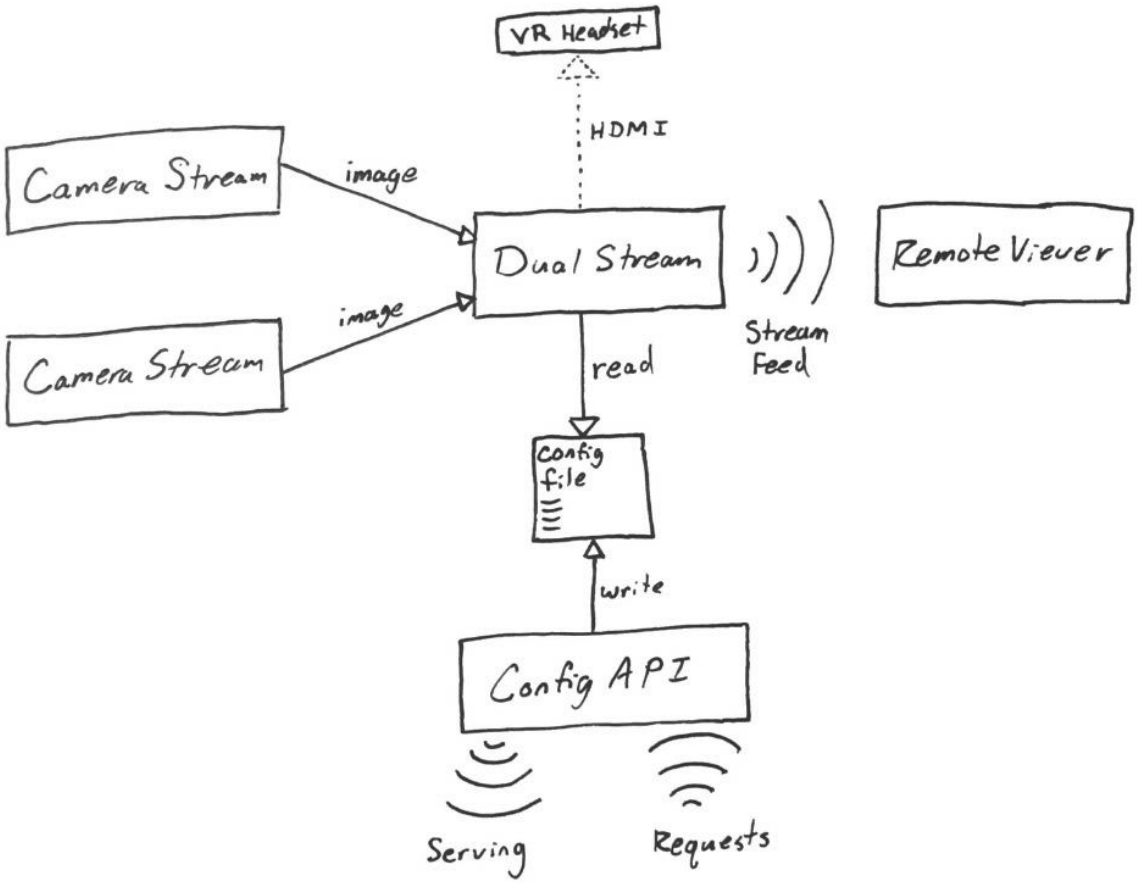
**OBS:** to configure OBS, one must first install the software. OBS can be downloaded from https://obsproject.com, and the install wizard requires only that you supply an install location. After successfully installing OBS, one can use OBS's settings configuration wizard to optimize the settings for streaming given the available. No other changes need to be made to OBS's streaming settings to get a basic stream running, but to further optimize the encoding process (GPU encoding, varying audio sampling rates, video buffering, etc) one can visit the OBS wiki for detailed explanations and guides: https://obsproject.com/wiki/.

**Flask:** app.py uses the Flask microframework; thus, to run app.py, one must first install Flask using pip. The command to do this is either *pip install flask* or *pip3 install flask* depending on which version of pip came with your particular python version.

**Port Forwarding and IP Addresses:** if the administrator wants the settings page to be externally visible to remote users, they will have to configure their router's port forwarding capabilities to allow external traffic to either port 80 or another port that the settings page is being served on. This can be done by going to your router's GUI (commonly at http://192.168.1.1) and forwarding your machine's IP address to the external IP address of the router. Furthermore, the administrator can find the internal address of the host by typing *ifconfig* (or *ipconfig* for Windows) into the host's terminal. To find the external IP address of the router go to https://www.whatismyip.com/.

**Twitch Account:** an additional concern of the administrator is to ensure that the Twitch account that is to be used for streaming is available to the user, and that the stream key is kept private.

**Adding Features:** our Real-Time Tele-Medicine Kit was built with expansion of capabilities in mind. All aspects of our project are very modular, depicted in the following class diagram.




Each part of the ecosystem has its own task with room for additional capabilities and interfacing. Currently, the setup is as follows:

**CameraStream:** The CameraStream class is a threaded OpenCV class that takes a camera source and continually reads the frames in a subprocess, thus allowing other applications to simultaneously do work. Other classes are able to efficiently read the frames that the CameraStream subprocess fetches.

**dual_stream.py (Dual Stream):** The dual_stream.py file's main task is to concatenate two stereoscopic images and display them locally on the screen. Currently, this is done by reading the two threaded instances of the CameraStream class. It also reads the local config file to control the camera settings like zoom, brightness, saturation and perform the necessary image modifications.

**app.py (Config API):** The app.py file is responsible for hosting the UI webpage that allows the operators to make changes to the camera streams. It writes the users desired configurations to a local .json file.

**OBS (RemoteViewer):** The remote viewer's job is to take the camera stream and allow remote users to view it. Currently this is done by doing a screen capture using OBS and streaming it over Twitch.

Since each element is so independent, both in terms of interfacing and computationally, there are many places that will gracefully allow for additional functionality. Beyond that, our code is meticulously documented to aid in the modifications.

# Testing

## Test Plan

## Test Cases and Results - Acceptance Testing

| Case Number | Test Case | Pass Conditions | Fail Conditions | Result |
|---|---|---|---|---|
| 1 | The settings web page is viewable | When app.py is ran, visiting localhost:80 through a web browser brings up the settings web page | When app.py is ran, visiting localhost:80 through a web browser does not bring up the settings web page | **PASS:** There is a settings page with sliders for brightness, contrast, etc. |
| 2 | The settings web page is functional | When the settings web page is pulled up, adjusting the sliders impacts the image accordingly | When the settings web page is pulled up, adjusting the sliders does not impact the image accordingly | **PASS:** The settings web page affects the video feed |
| 3 | Video feed displayed locally | The video feed from two 4k cameras being attached to the slit lamp is able to be seen on a local screen | The video feed from two 4k cameras being attached to the slit lamp is not able to be seen on a local screen | **PASS:** We are consistently able to produce a local feed from the two cameras. |
| 4 | Video feed displayed | The video feed from | The video feed | **PASS** |

| | | | | |
|---|---|---|---|---|
| | remotely | two 4k cameras being attached to the slit lamp is able to be seen on a remote screen | from two 4k cameras being attached to the slit lamp is not able to be seen on a remote screen | |
| 5 | Video stream able to be downloaded | When a stream is finished, it is able to be downloaded | When a stream is finished, it is not able to be downloaded | **PASS** |

# Test Cases and Results - Unit Testing

| **Case Number** | **Test Case** | **Pass Conditions** | **Fail Conditions** | **Result** |
|---|---|---|---|---|
| 1 | index.html | The settings page is able to be viewed when app.py is running | The settings page is able to be viewed when app.py is running | Pass |
| 2 | app.py & index.html<br><br>Slider values don't reset | When a slider value is adjusted and submitted, it does not reset | When a slider value is adjusted and submitted, the value resets to what it originally was | Pass |
| 3 | app.py & index.html<br><br>Settings sliders adjust video feed image | Changing the values of the settings sliders actually affects the video feed image | Changing the values of the settings sliders has no impact on the video feed image | Pass |
| 4 | dual_camera.py<br><br>Script asks for user input | When the file dual_camera.py is run, it asks the user for the camera input source | When the file dual_camera.py is ran, it does not ask the user for the camera input source | Pass |

| | | | | |
|---|---|---|---|---|
| 5 | dual_camera.py<br><br>Video feed displayed in stereoscopic view | When dual_camera.py is ran, a window that shows the two camera frames concatenated to produce a stereoscopic view is produced | When dual_camera.py is ran, a window that shows the two camera frames concatenated to produce a stereoscopic view is not produced | Pass |
| 6 | Twitch<br><br>Video feed is streamed | When the stream key is entered, the user can press the Start Streaming button, and the stream can then be viewed | When the stream key is entered, the user can press the Start Streaming button, but the stream cannot be viewed | Pass |
| 7 | app.py func update_values() | Dictionary values of CONFIG_SETTINGS are updated based on the values from the html page | Dictionary values of CONFIG_SETTINGS are not updated based on the values from the html page | Pass |
| 8 | app.py func update_values() | The updated CONFIG_SETTINGS dictionary is written to the config file | The updated CONFIG_SETTINGS dictionary is not written to the config file | Pass |
| 9 | app.py func send_values() | The dictionary CONFIG_SETTINGS is properly sent in json formatting via a GET http request | The dictionary CONFIG_SETTINGS is not properly sent in json formatting via a GET http request | Pass |
| 10 | index.html SWAP_CAMERAS | The value 1 is sent if checked, otherwise 0 is sent | The correct values are not sent depending on whether it is | Pass |

| | | | | |
|---|---|---|---|---|
| | | | checked or not | |
| 11 | index.html VERTICAL_FLIP | The value 1 is sent if checked, otherwise 0 is sent | The correct values are not sent depending on whether it is | Pass |

## Testing Quality Review

Upon review of our test cases, we found that the level of testing that we performed was sufficient for ensuring proper operation of our product in a real-world scenario. As we completed our demonstration of our product in the ophthalmology clinic at UK, we found that each of the aspects of our software that were tested were critical to the success of the product, and we did not find any other serious limitations that required additional testing to guarantee a viable product.

Specifically, with regard to our acceptance tests, each of these tests corresponds to a user story. Given that all of our user stories were tested and deemed to have passed those tests, we can say with confidence we delivered on each of the goals that we set with our customer.

# Metrics

## Story Points

The scale will range from 1-5, with 1 signifying a task that is simple and will require little effort and 5 signifying a task that is complex and will require a great effort.

A.) The most critical function of our product is the transfer of a high-resolution video stream from cameras mounted to an ophthalmologist's slit lamp to a portable viewing device as well as a virtual reality viewing device.

**Story Points:** 3, while this is the most critical function of our product, we already have some code to work from, and we are not as concerned with potential latency issues for the local stream as we are with the remote video stream. Overall, this task is moderately complex and will require a moderate effort to complete.

B.) The next function that our product shall have is the ability to stream the high-resolution video over a network connection to a remote viewer (with VR capabilities also). This stream must have low-to-moderate latency, as the stream may be used for applications such as guiding a resident through a procedure in real-time or acquiring a second opinion from a remote-located physician.

**Story Points:** 5, this function is not as critical as the local stream, but our customers have demonstrated the immense potential that such a function could have to revolutionize remote medicine; thus, it is still of great importance to our team. Minimizing the latency over the network connection will likely be a challenge in addition to the fact that our customers require ultra-high resolution video. Overall, this task will likely take the majority of our time and effort to complete, and it will almost certainly be the most challenging part of our project.

C.) Another aspect of this project is to create a settings page so that camera properties such as brightness, contrast, saturation, etc. are able to be updated by the user of the system.

**Story Points:** 3, this capability is not necessarily required but it is useful for the ophthalmologist to be able to adjust the image so that they have the best possible view of what they are trying to see.

D.) The last function of our product will be to record and store the video stream that we capture for later use. This function may present storage challenges, as the 4k resolution video will likely be very large. We may need to learn about video compression or invest in large amounts of external storage.

**Story Points:** 2, this function will not be as challenging to implement, as saving the video stream should be quite simple to implement on any of the machines that are receiving the stream. The tricky part will be the efficient storage of the video so that it is portable and scalable, but there are simple solutions in existence already.

# Product Size

User Stories:

1. As an ophthalmologist, I want to be able to change visual settings of the image, so that I can adjust the way the image looks to better suit my needs.
2. As an ophthalmologist, I want to be able to view on a virtual reality headset a video feed of my patient's eyes through a slit lamp, so that I have a digitized experience.
3. As a secondary ophthalmologist, I want to be able to view remotely a video feed of the primary ophthalmologist's patient's eyes through a slit lamp, so that I can assist the primary physician.
4. As an ophthalmologist, I want to be able to record the video stream of my patient's eyes, so that I can look at it later.

# Product Effort

As a whole, around 50 hours were spent implementing this project. The most difficult aspect was ensuring we had the proper software installed. Across different group members own devices and implementations there were several issues with software compatibility. This became even more prevalent when implementing the camera property controls as certain versions of Opencv were required to function properly with the cameras we were using.

# Developer's Notes

Our developers notes are found on our Github at the link https://github.com/coleterrell97/SeniorDesign. Cole's notes are under the file "cole_dev_notes", Shelby's notes are under the folder "shelby_dev_notes", Sam's notes are in the file "sam_dev_notes.md", and Bryce's notes are in the file "bryce_dev_notes.docx".