Preliminary Project Design

Team Number

14

Team Members

Ross Copeland, Aaron Aylor, Christopher Ariagno, Jack Lunceford, Noah Mohabbat

Project Name

TrackMe.Live

Project Synthesis

A web application to either manually control or use an ML model to track an individual while speaking with an external camera.

Project Description

This project is being undertaken in order to make the process of filming speakers in rooms such as classrooms and churches completely remote. Right now, a problem with filming a speaker in a room is that when the speaker goes out of frame, someone has to physically change the camera angle to continue filming the speaker. To address this problem, we will create a web application called TrackMe.Live that will enable the user to control the camera angle from a computer to keep the speaker in frame, as opposed to having to physically change the camera angle. That way, the camera angle can be changed remotely. As a stretch goal for this project, we would like to automate the process of filming a speaker by controlling the camera via a machine learning model. With a machine learning model in place, if the speaker goes out of frame, there would be no need for a human to control the camera angle to continue filming the speaker. Instead, the camera would automatically change its angle to continue tracking the speaker. Thus, the filming of a speaker would be fully automated. The end result of the project would be a web application that controls the camera angle remotely, as well as possibly a machine learning model built into the web application that would fully automate the process of filming a speaker.

Project Milestones

First Semester:

UI with Video Streaming (Oct. 14, 2021)

VISCA commands to server (Nov. 14, 2021)
UI for video and VISCA control (Nov. 30 2021)
Python and Flask backend (Nov. 30, 2021)

Second Semester:

Full-scale command line program (Feb. 26, 2022)
Deep Learning model (Mar. 1, 2022)
Integration of UI & Tracking Systems (Apr. 13, 2022)
System Testing & Documentation (Apr. 20, 2022)

Project Budget

Item	Cost	Vendor	
Sony VISCA Camera	$195	eBay	
Camera Wires	$30	Amazon	
HDMI Capture Card	$30	Amazon	
		Total:	$255

Overview

For our senior design project, we are looking to create a control system for a special format of cameras called VISCA cameras. The VISCA Protocol is a camera system used with PTZ cameras. This format is relatively standardized across cameras, and used widely across non-profits, universities, and companies. Our goal with the TrackMe.Live software is to create a software package where users can easily control VISCA cameras using an easy to use web interface. This piece of software should be run locally on a user's computer so they can control VISCA cameras that are attached to their computer. We believe this software will hit a pretty broad segment of users who have VISCA cameras but don't have the money to buy expensive controller software.

Our software package is split up into three different components. The components for the TrackMe.Live software package include:

1. Angular Frontend
2. Flask Backend
3. VISCA Camera API

The Angular Frontend will be the actual web interface where users can actually control the camera. This will be built using the typescript framework called Angular. Our backend is built with Python using the Flask web framework. Finally, the VISCA Camera API is built out using

Python as it will work seamlessly with our Flask Python backend. **Figure 1** shows how the three components will work with each other.

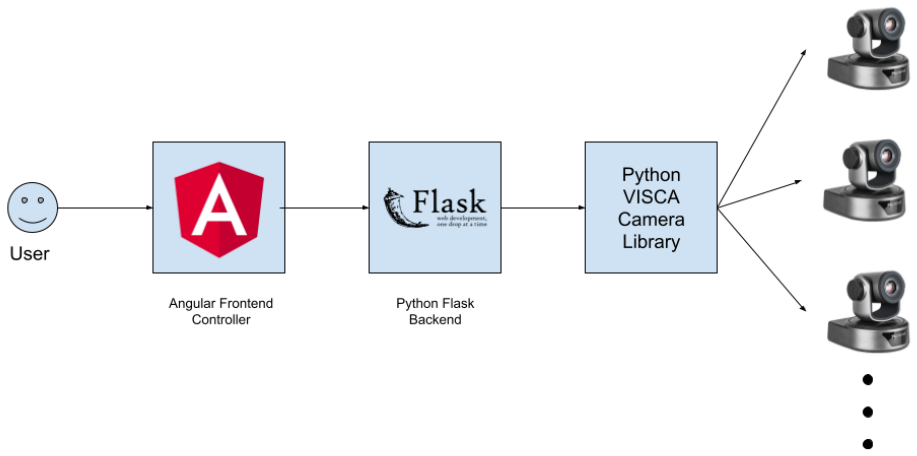

Figure 1: Architecture diagram for Trackme.Live

Angular Frontend

Like discussed in the Overview, the Angular Frontend will be used by the user in order to control the actual VISCA camera(s). We can split the front end UI into a few components:

1. Camera Live Feed
2. Software Buttons to control the camera
3. Instructions for an optional keyboard interface to control the camera
4. Settings Menu

We are planning on making these four components part of the MVP for our project, and extend out to different features as we see fit as we get farther into the project. **Figure 2** shows a mockup of our frontend interface in Angular.

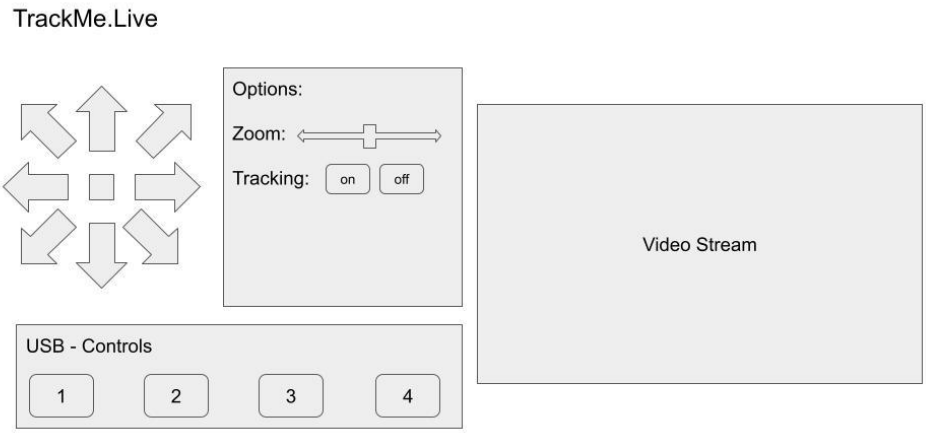

Figure 2: A mock up design of the Angular interface

In order to quickly iterate and create our software, we will utilize a few frontend Angular libraries. Our current plan is to utilize the following frontend libraries: Bootstrap, Font Awesome, NGX-Joystick. Using these libraries will provide easy access to things like pre-designed HTML components, icons, and joysticks for the user to interact with. This will save us time in the long run as we can focus more on the user experience and actual features rather than building out HTML components from scratch.

In order to interact with the backend, we will be using web sockets. Web sockets are very similar to the Unix-style sockets as we have a two way communication with frontend and backend that allows for pretty seamless integration with one another. We will be using web sockets for anything that needs low-latency interaction with the backend. This means we will be using web sockets for all of the actual VISCA commands needed to control the camera. To implement the web sockets, we will use a library called socket.io. Socket.io is a handy library to implement web sockets in the frontend and backend. Specifically in Angular, we will use their library to interact with our Flask backend. This library acts as a wrapper to the actual web socket interface so we can work easily with the socket.io backend in Flask.

Flask Backend

The Python and Flask Backend will be used as a "middleware" between the frontend UI and the VISCA camera API. Our backend will be responsible for a few different things including:

1. Hosting our frontend UI
2. Streaming the camera video
3. Listening to socket requests from the frontend

Like discussed in the Angular Frontend section, we will be using socket.io to let the frontend and backend communicate. We are using socket.io as it is low latency, easy to implement, and reliable. Socket.io is relatively easy to implement as we can just grab the socket.io official Flask library, and add some annotations onto our existing Flask functions. **Figure 3** shows a code snippet of how to implement a socket in Flask.

```python
@socketio.on('connect', namespace='/web')
def connect_web():
    print('[INFO] Web client connected: {}'.format(request.sid))

@socketio.on('test', namespace='/web')
def test():
    print('[INFO] Web client {} said TEST'.format(request.sid))

@socketio.on('disconnect', namespace='/web')
def disconnect_web():
    print('[INFO] Web client disconnected: {}'.format(request.sid))
```

Figure 3: Code snippet for Python and Socket.io

VISCA Camera API

In order to control the VISCA camera, we will need to create some form of API to actually interface with the camera. Luckily, there is a lot of different documentation on actually sending VISCA commands to cameras. VISCA is a very standardized format for cameras, meaning that if a camera is labeled as "VISCA", we can send the exact same byte code commands to any camera and get the same response. To connect a VISCA camera, we will need to plug in both the video and serial interface to the computer. Some VISCA devices have both video and serial combined like this one, and others require multiple cables like this one. Our library will need to read video and also send / receive commands from the camera. In order to do this, we will need to establish a base connection with the camera. Once we detect and create a connection with the camera, we can then send over known byte code over to the camera. **Figure 4** shows a snippet of the commands we will be sending to the camera.

Control commands

Command	Function	Command Packet	Note
AddressSet	Broadcast	88 30 01 FF	Address setting
IF_Clear	Broadcast	88 01 00 01 FF	I/F Clear
CommandCancel		8x 21 FF	
CAM_Power	On	8x 01 04 00 02 FF	Power ON/OFF
	Off	8x 01 04 00 03 FF	
CAM_Zoom	Stop	8x 01 04 07 00 FF	
	Tele(Standard)	8x 01 04 07 02 FF	
	Wide(Standard)	8x 01 04 07 03 FF	
	Tele(Variable)	8x 01 04 07 2p FF	p = 0(low)~7(high)
	Wide(Variable)	8x 01 04 07 3p FF	
	Direct	8x 01 04 47 0p 0q 0r 0s FF	pqrs: Zoom Position (0(wide) ~0x4000(tele))
CAM_Focus	Stop	8x 01 04 08 00 FF	
	Far(Standard)	8x 01 04 08 02 FF	
	Near(Standard)	8x 01 04 08 03 FF	
	Direct	8x 01 04 48 0p 0q 0r 0s FF	pqrs: Focus Position
	One Push AF	8x 01 04 18 01 FF	

Figure 4: Snippet of VISCA commands being implemented

Constraints

One of the biggest constraints to our piece of software is making sure the software is free or relatively cheap. We were approached by St. Lawrence last semester with a request for this piece of software as they felt that many of the other alternatives are very expensive. Making sure this software stays cheap or free is important as it stays within the original request of the stakeholders of the project. Additionally, we will need to make sure this piece of software is operating system agnostic to support the widest variety of use cases in the real world. Finally, we would like to use Python as one of our main programming languages as it is the easiest and most friendly to interface a web server and USB interface with one another.

Stretch Goals

Additionally, we have some stretch goals that we would like to complete with this project. If there is additional time in this project, the team would like to achieve a few different tasks including:

1. Implementing a machine learning library to track certain objects in the camera feed
2. Support for multiple camera control in the frontend and backend
3. Support for VISCA commands over IP

Ethical Issues

One of the ethical issues that we have to keep in mind is to ensure that the work of others is respected. For example, there are already many existing solutions to parts of our project on the internet (e.g. using a third-party video player, a joystick button, etc.). Any third-party solutions must be credited in some form. If we directly use any source code of anyone in our project, then we should indicate the authors of the source code in our project appropriately. Additionally, the dependencies and frameworks used in our project should be apparent in our project source.

Another ethical issue is the potential harm of our project. A potential harm of our project is damaging the camera of the user of our program. Our program will be working with cameras that use the VISCA command protocol. This command protocol interacts with the hardware of our camera. The risk of damaging the camera via a command is negligible. However, we have to ensure that we do not create any unintended consequences in our program when integrating these commands into our program. For example, if our program unintentionally and repeatedly sends commands to the camera far beyond its normal use, this may lead to damaging of the camera.

Intellectual Property Issues

Our project should not run into any intellectual property issues from the code we are using. We are using Angular to design our frontend and Flask for the backend which are both open source. As long as we credit the original authors and use the libraries as intended then we will not face any issues there. One of the intellectual property issues we could face is creating a program that is similar in design to another that is currently under copyright. So our team will strive to respect the current copyrights and patents to create a product that is uniquely ours.

Change Log

Nothing has been changed from our initial project description. With the additions of the Preliminary Project Design, Ethical Issues, Intellectual Property issues, and the Change Log.