B2: DrawBuddy

Authors: Lisa Mishra, Ronald Gonzalez, Denise Yang
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A user run P2P application that enables the transfer from physical drawing of diagrams to a computer accessible SVG file that can be modified. Line detecting computer vision algorithms will be used to collect points and lines which will be fed to a vectorization software algorithm. The output SVG file will then be sent to all other users through a server for everyone to be able to modify.**

***Index Terms*—Python, Sockets, Image Vectorization, Computer Vision Line Detection, P2P Communications, SVG files**

1 INTRODUCTION

DrawBuddy is an interactive multi-user whiteboard application that can assist students, or developers in creating diagrams and sharing them to each other without having to go through the hassle of using their mouse to draw diagrams which can be tedious at best, and downright impossible at worst. With COVID-19 having modified many aspects of our lives to an online format there is a growing need for an easier way for students to create these kinds of diagrams online at little to no cost to them. As stated the current best way to create diagrams online and share them with others is to either use some sort of whiteboard software and draw with your mouse and keyboard, or have a drawing pad which can be expensive and take up unnecessary space or use some other software to create the diagrams.

Neither of those two options are very helpful, especially for college students so, we are proposing an application that allows the user to take a picture of their hand drawn diagram and we can convert it into an SVG file that can be transferred to other connected users and from there the users can then modify the file by translating lines or rotating blocks. With this students will be able to form groups of 2-5 and be able to study for classes and have the ability to send properly made diagrams to each other without needing to spend vast amounts of time on some application hand drawing them with a mouse.

Our goal is to have an easy to use application that can set up virtual rooms with which people can take an image of a diagram and have it be converted into a modifiable SVG file. This process is going to use computer vision to detect all of the lines in the file and then vectorize that into an SVG file. That SVG file will be sent to all other users through a socket server being run on the host's computer. Once sent other users will have the option to accept or reject images with the first received image being automatically placed within view to start modification.

2 USE-CASE REQUIREMENTS

We want to ensure the creation of a usable and adaptable application that various groups of students can find a use for. Our use-case requirements have been split into two categories based on the aspect of the project they apply to, those being: communications and ease of access. It is key that the application can support multiple people using it since the main goal of this is to be used by a study group to share diagrams. By nature of the application there are additional required components for the user to have access to, those being a paper, a camera, and some sort of tool to write with. So we want it to be as easy as possible for them to have all of these components readily available.

2.1 Communication Requirements

This application is peer-to-peer by design so that all users can send SVG files to each other and work on the same diagram. For example a class might require a group working on a state transition diagram as a lab and sometimes these diagrams can get complicated so drawing them on a mouse and keyboard is not a realistic option. On the communications side, we need for there to be a good amount of users being able to join a room and an optimal size for a study group is about 4-5 people so we want the max size for the application to be 5 people being able to join a group [1]. Since we can not run a server for people to connect to we're going to have the user who creates the room run the server on their local computer and give them an access code to give out to the other users who will then be able to type that in and connect to the network.

2.2 Usability Requirements

DrawBuddy should be able to recreate the user's diagrams as accurately as possible. Thus we would like to have users rate the produced diagram on a scale of 1 to 10 where 10 is an exact copy of what they intended to create and 1 would mean that the produced diagram is unusable. We also hope of having 90% accuracy in terms of lines outputted compared with the number of lines drawn in order to more quantitatively measure the accuracy of the produced SVG.

2.3 Ease of Access Requirements

One of the primary goals of our project is to make our system accessible, meaning that users should be able to easily use our system with what they currently possess in their work space.The user should be able to draw their diagram

on white letter size printer paper with any writing utensils that uses black ink and has a diameter of anywhere between 0.4-1.0 millimeters. Letter sized papers can frequently be found at home or in academic public settings such as schools and libraries and the range of supported diameters includes common writing utensils from anywhere between a fine tip pen all the way to a sharpie. The user should be able to hold the image up to the camera from anywhere between 1 to 3 feet away since the user should not need to move too much from their current position in front of their computer in order to use our web application. As for the camera we have decided that a laptop camera is enough to work and since we can assume that most of the students are going to have access to a laptop this is best for their ease of access.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

3.1 High-Level Architecture

The main technical components of this project can be broken up into image filtering/vectorizing, the graphical user interface, and communication over a server between users through the usage of sockets.

See Figure 9 for our high-level block diagram (on the next page). The laptop camera will take in the image that the user should be holding up, and give that JPEG or PNG file to an image filtering program. The image filtering process will convert the image to black and white (to make it easier to identify the lines and shapes in the drawing) and eliminate any noise in the image using a Gaussian Blur. The filtered image will then be sent to an image vectorizer program which will vectorize the code using a software called VTracer. VTracer will take the image and create an SVG file that can be rendered to display or sent to other users on the server. On the display will be a graphical user interface that will contain a virtual whiteboard. The vectorized image will be on this whiteboard and the user can use it to modify the diagram and send it to other users within a shared session.

3.2 The Graphical User Interface

The Graphical User Interface should encapsulate the technical components of this project - the user should be interacting with it, the image filtering/vectorizing programs should be behind the GUI to facilitate the "virtual whiteboard" portion of the project, and the socket communication code should also be behind the GUI to allow the different users to communicate. Therefore, the GUI will serve as our frontend and the image filtering and vectorizing process, along with the sockets code for communication among users, will serve as the backend.

The user will have to start the application on their laptop, and the homepage of the GUI should look like the below picture. There will be two buttons on the home page, one that says "host" and one that says "attendee."

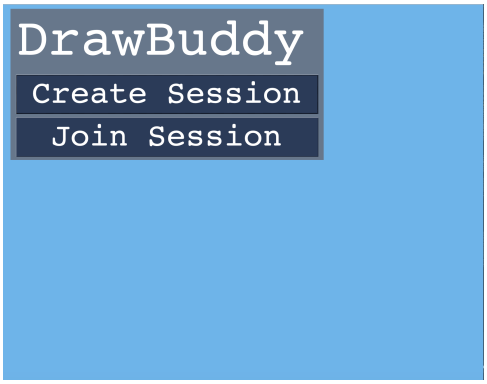
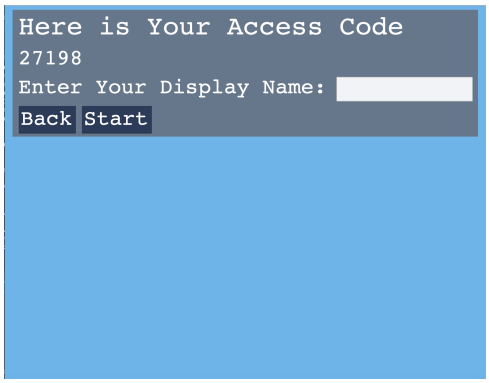
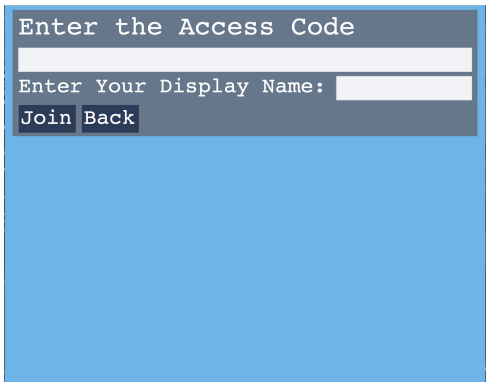


Figure 1: The home page of the graphical user interface



(a)



(b)

Figure 2: User interface (a) the creator's view. (b) the joiner's view

There will be two buttons there, one that says "Create Session" and one that says "Join Session." If you click "Create Session", you will start a session and create a 5-digit access code for that session, as shown in Fig. 2(a)

The host will then have to give the access code to their peers. If you're an "attendee", you will have to use that

access code to join the server. Fig. 2(b) depicts what the attendee will see on their end of the user interface once they click on the "attendee" button.

Each user will have their own whiteboard that will look like Fig. 3, with a camera in the top right corner that captures the user's drawing and the whiteboard that has the vectorized version of it.

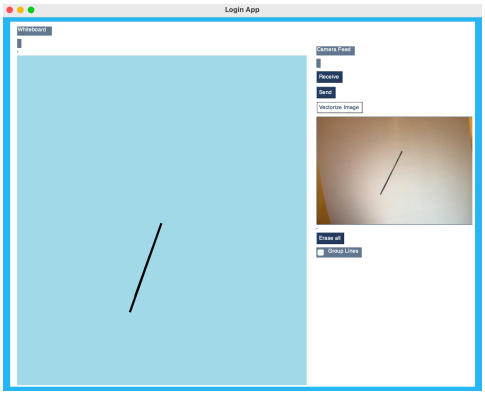


Figure 3: The virtual whiteboard in the GUI

All attendees of the session can send and receive images. The bottom right corner of the virtual whiteboard will have a "send" and "receive" button for all users. Note that the host is not the only one that can send messages. We decided to use the names "host" and "attendee" to mimic the terminology that many people are already familiar with due to their using Zoom for virtual learning. For example, in the context of using Zoom, even though there is typically only one host, all attendees in the zoom meeting can share their screen. Similarly, the host in our app is only responsible for starting the session so that other connected users can communicate within the shared server.

3.3 Image Filtering and Vectorizing

A high level depiction of the order of programs through which the image travels before being rendered to display and ready to send is shown below.

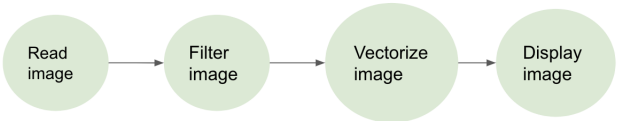


Figure 4: A high level summary of the image filtering and vectorizing process

Once the image is read through the laptop camera, the backend should begin filtering and vectorizing the image before ultimately displaying the vectorized image onto the whiteboard. The image filtering portion will use computer vision algorithms to read the image, convert it to black and white, and then apply a Gaussian Blur on the image. The Gaussian Blur helps to eliminate the noise and assist the vectorizer in being better able to detect the lines and shapes in the drawing.

Once the image filtering is completed, the image vectorizing program will start. It will vectorize the image using an open-source software called VTracer. We had also considered other options instead of VTracer, which are noted in the 'Design Trade Studies' section. The vectorized image will be an SVG file that can be rendered to the display and sent to other users.

Should the user choose to modify the vectorized image, they would have to click on the component that they want to vectorize. The program will use the mouse click and release points to determine whether the user is attempting to translate or scale and then carry it out. If the user wants to translate the image, they will have to select the image as a whole, by clicking near the center of the image (a border like the one shown in the [figure] should be displayed if successful). Then the user can click and drag the image to translate. If the user wants to scale the image, then they would have to click on one of the endpoints of the polygon or line and drag the mouse click to scale it out.

This will all be done on the graphical user interface. In order for the change to be reflected in the vectorized diagram on the whiteboard, the underlying SVG file will have to be modified. For example, if a user's vectorized diagram has a line, the SVG file will have a line in it that indicates that there is a "line" with the associated endpoints written out in the SVG file as well. Therefore, if we're modifying a line using the GUI, we also have to modify the SVG file behind it, specifically those endpoints. We'll be using the python library SVGUtils for that. Fig. 5 depicts a high-level summary of this process.

3.4 Communication Among Users

Once the image is vectorized as an SVG file, it is ready to be displayed onto the whiteboard and sent to other users in the session. The host and attendees can now send and receive images to each other (recall that the only functional difference between a host and an attendee is that the host started the session and provided the access code to the attendees). In order to receive an image, the receiver has to actually click on "receive. The reason for this decision was to make sure that the sender can't just inundate any receiver with images. Until the user hits "receive", that SVG file will be stored in a queue for the receiver to eventually take the file off of. The receiver will also see, on their whiteboard, the number of total received messages in the bottom left corner, so that the sender will not have to notify the receiver every time they decide to send a message. If the receiver wants to look at the received images, they can open the inbox, which will look like Fig. 6.

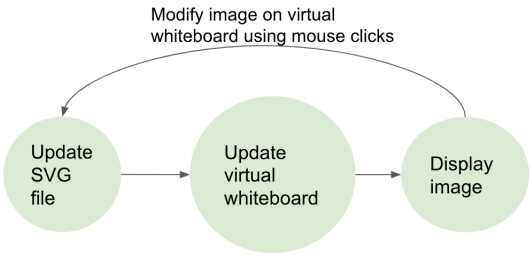


Figure 5: A high level summary of the image filtering and vectorizing process

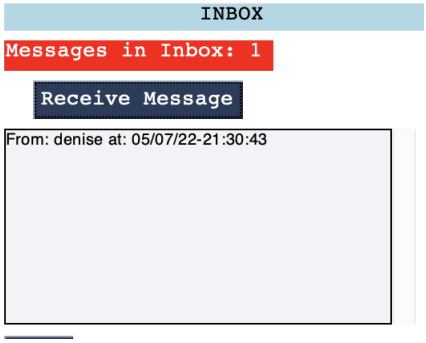


Figure 6: Example of user inbox. If the receiver wants to view an image, they can click on the button that is labeled "Receive Message", which will show the sender and time the message was sent.

4 DESIGN REQUIREMENTS

4.1 Accessibility Requirements

Our system must also support camera resolutions of 720p and 1080p because these are the most common resolutions of the built-in cameras on laptops. The image will be captured as either a PNG or a JPEG before sent over to image processing to remove any noise.

4.2 User Experience Requirements

The result will then be sent over to the image vectorizer where the JPEG/PNG will be converted to a SVG file that will then be sent over to the display. The overall latency from capturing the image from the laptop camera to rendering it on the user's display should take no more than 2 minutes for a simple image. From experimentation, to create a diagram consisting of roughly 50 primitives digitally with a mouse took about 4 minutes, so we are aiming to not only be more efficient than drawing the diagrams digitally but also provide a 2 times speed up with our application.

The components within the file can be translated or scaled digitally. In order to provide a smooth user experience, we require the latencies to modify a component of the digitized diagram to be 150 milliseconds. This latency is based on Dabrowski's research on The Effects of Latency on Player Performance and Experience in a Cloud Gaming System, where he added network latencies to a PC game (Crazy Taxi) and measured how they affected the users' quality of experience (QoE)[2]. Users were asked to rate their QoE on a scale of 1-5[2]. There was a linear decrease in QoE as the latencies increased however after 150ms the ratings dropped below 3[2]. This is a much stricter bound than necessary since in gaming users need to be able to react immediately or else their character could lose; however with our use case users may only experience a slight annoyance if there is a small amount of lag.

4.3 Communication Design Requirements

The server will be set up and run by the first user and they will give the access code to all the other users who will then join. The latency to join the server is almost negligible since there is going to be a screen where we will have the host of the room start the room and they will be able to see who has joined the room. Once they see that the room is full with five people they will then select the option to begin communications. The SVG file produced from the vectorizer will also be broadcasted to all connected users. We require sending the SVG file to users in a joint session to take less than 500 milliseconds since once again we can add more leeway than the gaming latency bound since our use case does not require immediate responses; furthermore this latency is likely to be masked by the user interaction of clicking the receive button to actually receive the image.

5 DESIGN TRADE STUDIES

The area with the most variability of our project is image processing and vectorization. For these steps within our design we have considered several approaches.

5.1 Pure CV Vectorization

Our initial idea involved using computer vision to process the image and produce the primitives that we would then output to an SVG file; however the main flaw with this approach is that the line detection algorithm broke our lines down into multiple overlapping lines especially as we increased the thickness of the lines.

5.2 CV Vectorization with Temporal Data

A second approach to resolve this challenge involved processing the image temporally i.e. the user would draw a single primitive, and capture it with the camera before drawing the next primitive. With every frame, we could reduce the additional lines generated into a single primitive and create a mask using the previous frames to subtract the previous primitives, such that we only vectorize the newly added primitive onto our virtual whiteboard. The primary concerns with this approach was that it would require additional accuracy on the users part to present their drawing in a similar position between frames such that we can compare two frames to remove the previously drawn primitives. For example if the mask from the previous frame was a straight line in the middle of the paper, but then the user held up their next image such that their next primitive was slightly in the center; this would result in the risk of the next primitive being masked out. This method would add additional latency than some of the other approaches since the user would have to draw the primitives in their diagrams one by one which would not scale well. Furthermore from a user interaction perspective, this approach increases the difficulty of use such that customers may not see the benefit of using DrawBuddy when compared to other virtual collaborative platforms like Zoom Whiteboard.

5.3 Potrace vs VTracer

We then pivoted to perform the vectorization process on the software side. We considered two different software libraries to use: Potrace and VTracer. Both of the algorithms used for Potrace and VTracer involve using a path walking algorithm to determine the contour of the primitives within the image file. Then they smooth the image to remove staircase artifacts termed "jaggies" from the path tracing. Potrace does this by identifying the directions of subpaths and creating a line on the straight subpaths, whereas VTracer uses the signed area of right triangles to determine whether to fill the jaggies in or remove the corners of the staircase that create these jaggies. Both algorithms then try to optimize the created lines to simplify the shape before applying a Bézier curve fitting in order to smooth it out.

Potrace can convert bitmaps into SVG files and works well for low-resolution images, which would satisfy our requirements for vectorizing diagrams because our use-case is for displaying diagrams in a collaborative setting. In other words, we are assuming the user would not spend a considerable amount of time (less than 2 minutes) to create their diagrams live. This method has lower CPU and memory footprints than other bitmap tracing libraries like Autotrace. However with Potrace we would need an additional step to convert the PNG/JPEG files produced from our computer vision software into a bitmap. With VTracer, we do not need this extra step since it can convert JPEGs and PNG into SVG files. This also means that we could extend DrawBuddy to be able to process colored input as well. Furthermore when Potracer performs it's optimizations, it analyzes the entire shape whereas VTracer runs a linear algorithm on clusters of the shape. As a result VTracer has a smaller CPU and memory requirement than Potrace making it more favorable for minimizing latency [3].

5.4 Sockets

We made the decision to end up using Sockets as opposed to other programs or applications. Initially we had it so that sending the SVG file to all users would take 150 milliseconds but we after some initial testing we realized it would not be possible to reach that mark using Sockets since the code was as optimized as it could be and the files just needed that amount of time to be sent and saved. For that reason we did consider using software other than Sockets to send the image but after some consideration we came to the conclusion that since instant messaging takes about 5 seconds to send and receive 500 ms would be an adequate update to our requirement for vector sending latency.

One of the options we had considered was using Amazon Web Services Elastic Compute Cloud or EC2. We decided against this for a few reasons, the main one being none of us were very experienced using AWS whereas we had one member who was very familiar with using the Python Sockets library. Using Sockets would be a much faster endeavor than using EC2 since there would be no additional documentation to learn. Another reason why we chose to use Python Sockets is for integration, all of our software is being written in Python so it would make for a much easier integration process if the server code was also done in a similar fashion.

6 SYSTEM IMPLEMENTATION

6.1 The Graphical User Interface

The graphical user interface will be implemented in Python, using the PyGUI library. The PyGUI library will allow us to develop a GUI API that is designed specifically for Python. We chose this because our image filtering and vectorizing code, as well as our sockets code, was all written in Python; therefore, in order to make our integration smoother, we decided to write our user interface in Python as well. . Our GUI code will also have to tie into the image vectorizing code and communication code. See Figure 1, 2, 3, 6 for our designs for our user interface.

6.2 Image Filtering and Vectorizing

We will be using the laptop camera as the means for taking in the image. A user will hold up their drawing to their laptop, and the laptop camera should be able to take the image and send it to the image reading and filtering software. Refer to Fig. 4 for the high-level overview of the image filtering and vectorizing component of the project. Image filtering will be done in Python, using the OpenCV library, a library that provides the tools to facilitate the use

Figure 7: The crop image algorithm. The green box represents the bounding box.

of computer vision in our program. We implemented an image cropping feature in order to remove the background of the image to ensure that when the user holds up the paper from various distances, we do not get unwanted artifacts due to objects in the background. We used OpenCV to convert the image to gray scale, and apply a binary threshold to convert the image black and white, before performing a morphological erosion and dilation to remove noise. After that we find the biggest contour within the image and create a bounding box around it. Then we scale the bounding box down to remove the hand that's holding the paper as shown in Figure 7.

Image vectorization will be done using VTracer, a raster image (a JPEG or PNG image) to vector graphics converter. VTracer stores the objects that it identified as a path object within the SVG file which is very versatile for creating various sorts of filled shapes and lines, but for our purposes we only wanted to display lines, so we wrote our own SVG parser that iterates through the lines of the file and extracts the end points of each path object and out stores it into a 2D array. Then we read each line within this 2D array and draw it in our whiteboard GUI.

6.3 Modifying Vectors

Every line on the whiteboard has a unique figure index to help distinguish it from other lines, used particularly when the user tries to move, delete, rotate, or scale a line. All lines are stored in a dictionary with the key being the figure index and the value being the endpoint values (the format being a list containing (x0, y0) and (x1, y1)). Whenever a line is modified or deleted, the program obtains the figure index (using the graph element of pygui), and uses that figure index as a key into the dictionary to modify the endpoints that is stored for that key in the dictionary. Whenever a line is deleted, the entry in the dictionary is also deleted.

We added a feature to be able to delete lines. Sometimes, when a user's drawing is vectorizing, the computer vision line-detection algorithm that we use can add extra lines that aren't in the user's diagram. So, we added a delete lines feature so that the user can check the "Delete Lines" box, and click on whichever lines that the user wants to delete (and then uncheck the "Delete Lines" box once they are done). There is also an "erase all" button that clears the entire whiteboard. One use case scenario in which this feature could be used is if the user vectorizes an image, but decides to add or remove several things rom their diagram and vectorize again. Before re-vectorizing the image, they can hit the "Erase all" button. We had also implemented a "group lines" feature that could be used to combine several smaller lines into one big line, but we realized that the same effect could be achieved by the user by deleting lines and then scaling one - this second solution was picked because it could allow the user to be more accurate and not confuse the two features.

6.4 Communication

A program will be written in Python to facilitate the communication between users. The Python sockets library will allow setting up a server that the users can use (and that the host can set up with an access code). A 5 digit access code will be randomly generated, this access code is what users will need to input before they can join the room, but they double as the port number for joining the server. The Python threading library will allow concurrent communication between the users, meaning that more than one image can be sent at the same time. When the server is created it will create a thread for handling user messages and a thread for the server which is what sends messages back to other users. When a user joins the server they will create a thread to handle incoming server messages so that they can always receive incoming server messages. The users can then generate vectors and when they hit send the vector will be converted into a string format and then encoded using UTF-8 so that we can get it in the bytes python format so that it can be compressed to make sending faster. After compression we encode it using base64 encoding to get it into a valid string format so that it can then be encoded using UTF-8 again to get it back into bytes so that it can be sent across the server again. The users will then receive the message in an inbox where they can select the vector they want to use and undo all the encoding and compression and rebuild the vector on their side. See Figure 6 for a layout for how a receiver's inbox would look in the GUI.

7 TEST & VALIDATION

7.1 Tests for Image Vectorization

7.1.1 Image Accuracy Results

Testing vectorization will involve submitting PNG files with 1, 5, 10, 20, and 50 lines drawn on them and assess-

ing how many lines are outputted onto the whiteboard in the GUI. As shown in Figure 8, DrawBuddy is relatively consistent with it's accuracy (roughly around 83.7%) in terms of outputted lines versus lines actually drawn. We were close but did not meet our quantitative use case requirement for accuracy of 90%. This is likely due to the uneven lighting conditions while testing which we will expand upon more in the future work section. However this failure is not a big concern since the overall diagram is fairly true to the original drawing. Furthermore the user vectorize the image again which will overlay the new image over the existing one, and then the user can manually translate, resize, and rotate the new lines to create the diagram they originally intended.

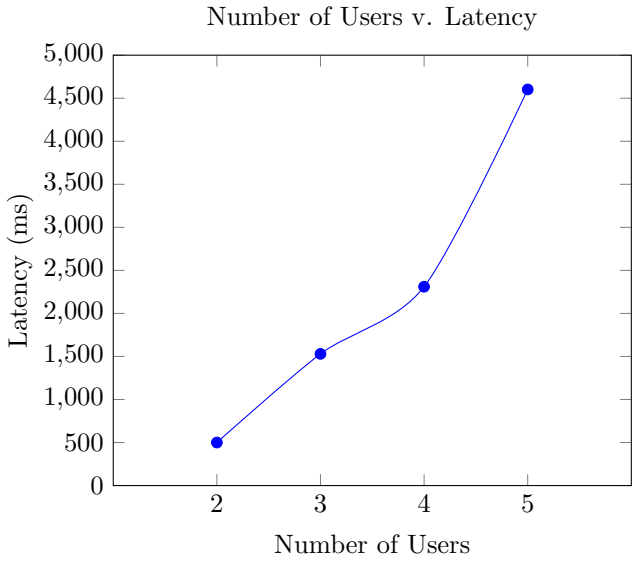
Overall our results were In order to judge accuracy we will ask several users to create PNG files using an online drawing software and have them rate on a scale of 1-10 how well the outputted SVG file reflects their original drawing. Originally 10 users rated the outputted images an average of 5.3 but after we tuned the line detection parameters and included image cropping, we asked 10 more users and received an average of 9.1 thus meeting our use case requirement for usability of 9 out of 10.

7.1.2 Vectorization Latency Results

Furthermore in order to measure the latency we will track the length of time it takes for the image processing steps and VTracer to convert the PNG once it's been uploaded to when a SVG file is actually outputted onto the GUI. For an image of 50 lines, it took around .1 seconds thus performing 1200x better than our use case requirement of 120 seconds When modifying the diagrams we will be timing the latency from when the mouse clicks the bounding box to the mouse release, and how long it takes for updates to be reflected within our display.

7.2 Tests for Communication Specification

Testing for communications is going to require running the server on one computer and then having a varying amount of users join and then sending SVG files to all of them. We tested this with 2 users and that is where we got our updated vector latency requirement of 500 ms, when testing with more than two users we saw an increase in latency based on the total number of users that had joined and this was due to the fact that the server software sends the vector to one user at a time so an increase in users leads to more latency.



As can be seen on the graph above it is not a linear change in latency like how we would expect and this is due to many factors including that testing was done on one computer with the server and all users being run on the same laptop which led to an increase in latency as the computer might not have been able to handle loading and running all those users. Another factor could be high variability in vector generating which could increase the total number of primitives in the SVG file that was being sent.

8 PROJECT MANAGEMENT

8.1 Schedule

Our schedule has been broken up into multiple parts to create a separation of components that will then need to be integrated together. These components are based mostly on which Python library they are mainly focused on. The parts are the OpenCV line detection software, the PyGUI software for creating an application that looks appealing, VTracer for vectorizing the OpenCV output, Sockets for server creation, testing and verification time, and finally some integration time. The full schedule can be found in figure 10.

8.2 Team Member Responsibilities

The project has been divided such that each member is working mainly on parts that they are most familiar with. Ronald will be working on the Python Sockets software as he has the most experience with it and has previously worked on a project that included it. Similarly Lisa will be working on the OpenCV. Both Lisa and Ronald will be working on the frontend for the application using PyGUI. Denise will then be the one working on the image vectorization using VTracer. Everyone will then test their individual components before the final integration that everyone will be a part of.

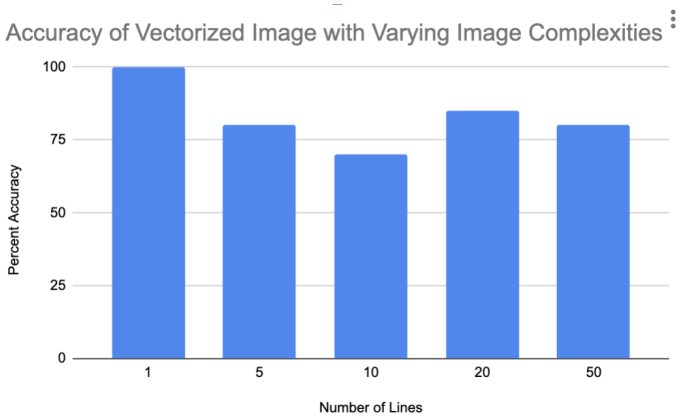


Figure 8: Vectorization accuracy for various image complexities measured by number of lines outputted on the whiteboard versus actual number of lines drawn. The average accuracy is 83.7%.

Table 1: Bill of materials

Description	Manufacturer	Quantity	Cost @	Total
Laptop	Apple	3	$0	$0
Pencil	Bic	1	$0	$0
Pen	Pilot	1	$0	$0
Permanent Marker	Sharpie	1	$0	$0
Paper	GP	20	$0	$0
				$0

8.3 Bill of Materials and Budget

Our project will be done entirely on software so we don't really have the need to purchase anything. We will need some things for testing and verification that won't be coming out of our budgets as they are extremely to get a hold of, those being a laptop with a camera that every team member has access to, some writing implements (Sharpie permanent marker, pencils, and pens), and some 8.5"x11" white paper. The table for cost can be seen in table 1.

8.4 Risk Management

The biggest risk that jeopardizes our project involves accurately converting the image to uniform lines and polygons within our latency requirement. There are many lines generated by our computer vision software but in order to improve the usability of our project we need to reduce the lines such that we only have a single line for each connected component that the user drew. This would also make rendering more efficient since the software will not need to process as many lines. In order to mitigate this we changed our approach to separate vectorization from the computer vision processing by utilizing VTracer, a PNG to SVG converter. Another risk was that VTracer would often convert other objects in the background of the image so we sought to improve the image quality by introducing a image cropping feature to remove the background as well as tune the line detection parameters.

9 ETHICAL ISSUES

While DrawBuddy is functional it is not quite ready for public use, since there are some ethical concerns with security and privacy. DrawBuddy does not currently use any method of encryption such that malicious users can snoop on the messages that are being sent between users. Furthermore while each session is passcode protected, if a malicious user gains access to the passcode then they can join the session and will have access to all the files sent after they have joined. There is no way for the host to remove them except to close the entire session. Another privacy concern is that DrawBuddy will ask users to allow camera access on the very first use, but not with any launches after that, so it is likely that the user may forget that they gave DrawBuddy those permissions such that DrawBuddy now has those permissions even if the user does not want that.

10 RELATED WORK

Based on what we have searched there is nothing online that converts hand drawn diagrams to images but there are some somewhat similar products out there.

10.1 Zoom

Zoom allows for the user to get in a video call with a large number of other users. Zoom can be free but the issue is calls will be limited to 40 minutes and the next pricing is

$150 a year. There is a draw option so someone can share their screen and create diagrams with their mouse but that is an extremely tedious option and exactly what we are trying to avoid.

10.2 Online Whiteboard Programs

There are plenty of online multi-user whiteboard programs but they face the same issue that Zoom does which is that this requires the user to use their mouse to create diagrams, the user could have a drawing tablet but that could incur a higher cost if they do not own one already.

11 SUMMARY

Virtual platforms have proven to be an effective mode of collaboration not only under COVID but also during poor weather conditions and for connecting peers across different timezones. DrawBuddy aims to provide an accessible and intuitive interface for virtual collaboration by enabling users to effortlessly transcribe their physical black and white diagrams onto a digital platform.

DrawBuddy utilizes computer vision for noise filtering and then VTracer to convert the image into an SVG file for the user to interact with and forward to peers via sockets.

Despite being widely used during COVID, virtual academia is still a relatively new and growing field. We hope to continue to help this field grow through DrawBuddy.

11.1 Future work

While we did achieve our MVP and a few stretch goals, our image processing stages produced poorer results in Wiegand Gymnasium compared to when we tested it in the lab and at home. This was likely due to the different lighting that also illuminated the objects in the background. Furthermore when there is uneven lighting DrawBuddy tends to omit portions of the drawn diagram when vectorizing. To make our system more robust, we could utilize a HSL (hue, saturation, light) line detection algorithm to better identify objects within the diagram instead of our current greyscale implementation. We could also improve our security by encrypting messages shared between peers via encryption algorithms like AES and also ask the user for camera permission upon every launch of the application.

11.2 Lessons Learned

The most important portion we learned is to give ourselves plenty of slack time, because We realized that integrating different pieces of code into a working application is bound to introduce new bugs. Furthermore as we worked on this project we identified new features that we needed to add that we didn't previous take into account like image cropping as well as adding helpful GUI features like a back button. Fortunately we heeded our mentors advice and did give ourselves plenty of slack in order to accomplish the additional work that we discovered and were able to integrate everything together to produce our MVP and meet a few stretch goals like rotation as well.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- AWS – Amazon Web Services
- CPU - Central Processing Unit
- CV - Computer Vision
- EC2 – Elastic Compute Cloud
- GUI - Graphical User Interface
- JPEG - Joint Photographics Expert Group
- P2P – Peer-to-peer
- PC - Personal Computer
- PNG - Portable Network Grpahic
- QoE - Quality of Experience
- SVG - Scalable Vector Graphic
- UTF-8 - Unicode Transformation Format 8-bit

References

[1] "5 tips for an Effective Study Group," The David Eccles School of Business, 23-Apr-2015. [Online]. Available: https://eccles.utah.edu/news/5-tips-for-an-effective-study-group/. [Accessed: 04-Feb-2022].

[2] Dabrowski, Robert, et al. WORCESTER POLYTECHNIC INSTITUTE, 2014, The Effects of Latency on Player Performance and Experience in a Cloud Gaming System, https://web.wpi.edu/Pubs/E-project/Available/E-project050514-142618unrestricted/The_Effects_of_Latency_on_Player_Performane_and_Experience_in_a_Cloud_Gaming_System.pdf. Accessed 6 Feb. 2022.

[3] Pun, Sanford. "VTracer." Vision Cortex, 1 Nov. 2020, https://www.visioncortex.org/vtracer-docs.

[4] Selinger, Peter. 2003, Potrace: a Polygon-Based Tracing Algorithm, http://potrace.sourceforge.net/potrace.pdf. Accessed 2 Mar. 2022.

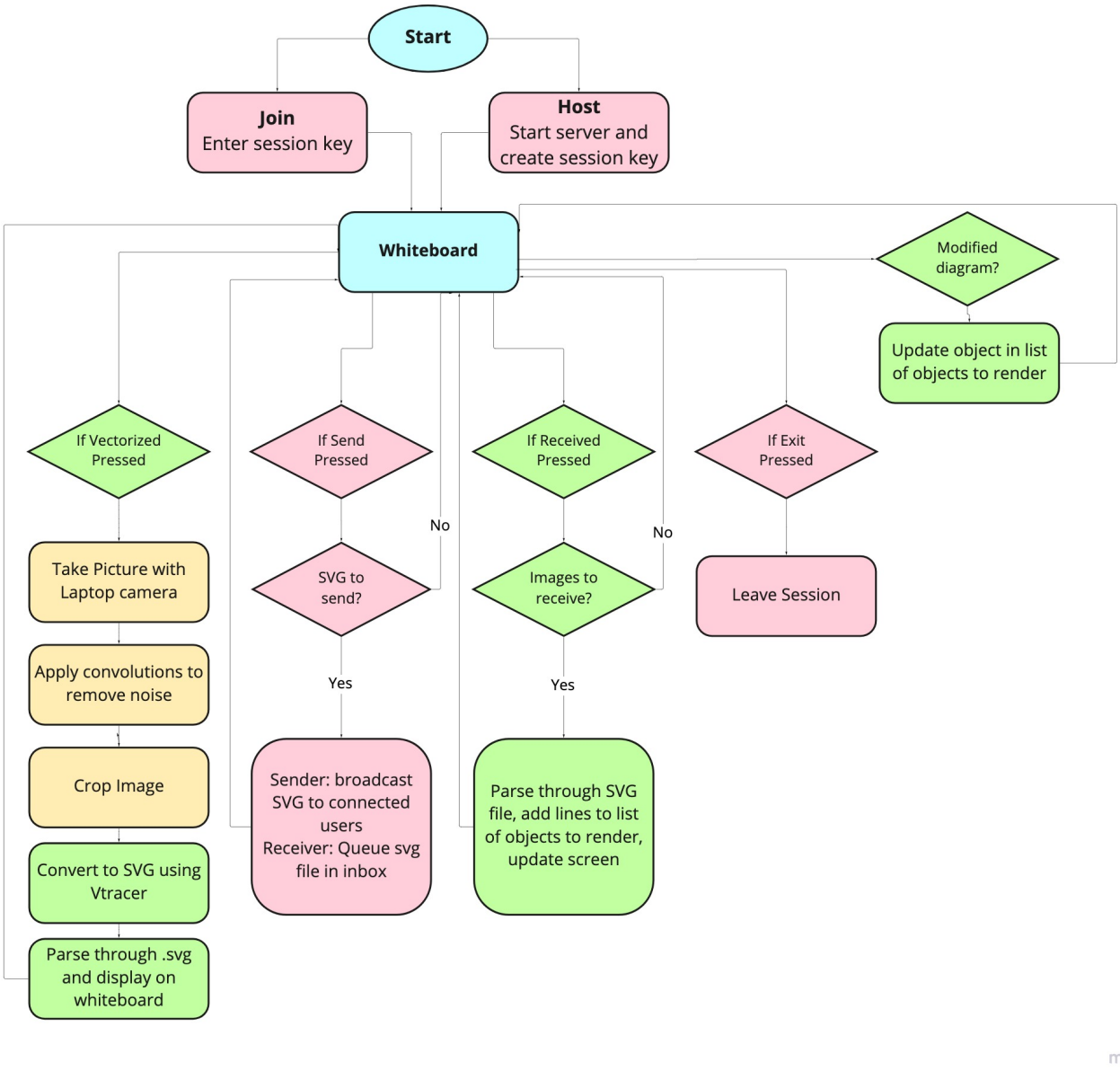
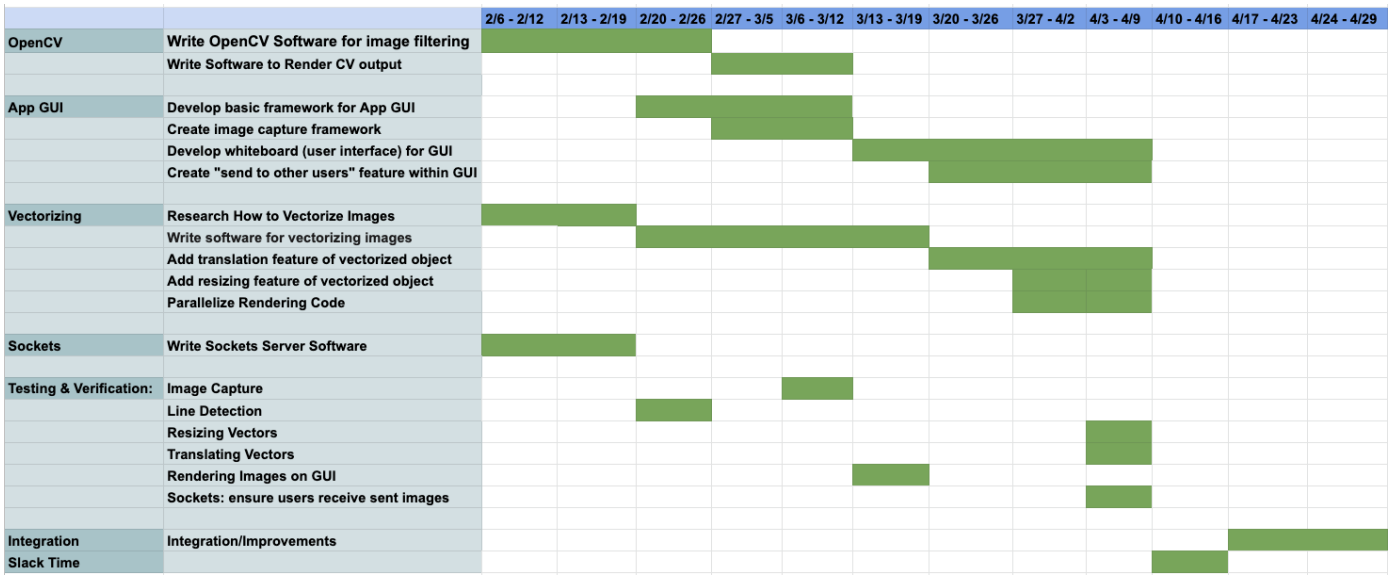


Figure 9: Software flowchart for DrawBuddy. The red components represent communication tasks, the yellow portions are the image processing components, and the green portions are vectorization steps. The bolded font represents our web app states.



		2/6 - 2/12	2/13 - 2/19	2/20 - 2/26	2/27 - 3/5	3/6 - 3/12	3/13 - 3/19	3/20 - 3/26	3/27 - 4/2	4/3 - 4/9	4/10 - 4/16	4/17 - 4/23	4/24 - 4/29
OpenCV	Write OpenCV Software for image filtering												
	Write Software to Render CV output												
App GUI	Develop basic framework for App GUI												
	Create image capture framework												
	Develop whiteboard (user interface) for GUI												
	Create "send to other users" feature within GUI												
Vectorizing	Research How to Vectorize Images												
	Write software for vectorizing images												
	Add translation feature of vectorized object												
	Add resizing feature of vectorized object												
	Parallelize Rendering Code												
Sockets	Write Sockets Server Software												
Testing & Verification:	Image Capture												
	Line Detection												
	Resizing Vectors												
	Translating Vectors												
	Rendering Images on GUI												
	Sockets: ensure users receive sent images												
Integration	Integration/Improvements												
Slack Time													

Figure 10: The current scheduled plan.