JJEE
Jordan Journal of Electrical Engineering
Volume 7, Number 3, September 2021
Pages 179-201
ISSN (Print): 2409-9600, ISSN (Online): 2409-9619

EyePaint: A Gaze Estimation-Based System for Hands-Free Painting

Dean J. Lawrence[1], Hannah G. Imboden[2], Hussein K. Chebli[3], Maya S. Kabbash[4], Adnan Shaout[5*]

[1, 2, 3, 4, 5] Department of Electrical and Computer Engineering, University of Michigan, Dearborn, Michigan, USA
E-mail: shaout@umich.edu

Received: March 02, 2021 *Revised: April 06, 2021* *Accepted: April 12, 2021*

***Abstract*—** This paper presents EyePaint, a system that enables hands-free painting on canvas via webcam-based gaze estimation software, a custom graphical user interface (GUI), and a computer numerical control (CNC) modified for painting. The proposed system's primary use is to assist a user with a physical disability in painting on canvas. Gaze estimation is administered using only a webcam through a unique combination of object detection algorithms, like Haar cascades, to localize facial features and learned linear regression models to translate facial features to gaze location. The custom GUI allows a user to draw a line or a circle in four different paint colors on a virtual canvas before committing their shape to a physical canvas solely through using the low-accuracy webcam-based gaze estimation. The system and its prototype are found to have utility through basic acceptance and system testing, but the chosen gaze estimation solution is generally too low accuracy for a positive user experience.

***Keywords*—** Gaze estimation; Eye tracking; Computer numerical control; Assistive technology; Painting; Graphical user interface.

1. INTRODUCTION

People with upper limb disabilities are, for the most part, unable to express themselves through painting. It can be extremely challenging for them to learn other means of painting, and the assistive technology that currently exists is uncommon. Lowering this barrier could allow new voices into contemporary art that were previously unheard.

Perera et al. [1] described existing assistive technology for artists with upper limb disabilities and the issues with those solutions. Technology for traditional painting is generally a simple tool, such as a mouth stick or head wand, which requires repetitive unnatural movement that can result in chronic pain and damage to the mouth. Digital painting assistive technology can use different inputs such as facial movements or voice control in the creation of art. This is an extremely time consuming and tedious process, and there are no off-the-shelf options available.

A work by Creed investigated a similar concept [2]. They ran a small scale study with ten disabled artists using gaze estimation and two physical buttons to complete a series of simple digital art tasks. The participants found it very frustrating to use due to the significant time each task took and gaze estimation sensitivity making fine movements and selecting small icons difficult. Some of the participants struggled to use the buttons without looking away from the screen, causing issues with the gaze estimation. Some also felt physical discomfort from positioning themselves for gaze estimation and straining their eyes. The paper found that "there has been no work, however, that has investigated the potential of the technology to support the creative process of professional disabled artists."

* Corresponding author
Article's DOI: 10.5455/jjee.204-1614652154

The system proposed in this paper is intended to avoid the issues with existing digital art assistive technologies, while still resulting in a physical painting. The design seeks to meet two simultaneous goals: i) to decouple the acts of creating art from physically touching a canvas and ii) to not be cost prohibitive. The first is achieved by designing around gaze estimation as the only input mechanism with a physical machine to replicate inputs and the second by requiring only a universal serial bus (USB) webcam to predict where a user is looking. In its ideal form, this could serve as a fast way to digitally input a rough painting and then, if the artist desires more detail, they could finish the painting using mouth or foot. This would be less tedious and require them to spend significantly less time physically painting and incurring physical strain.

Some inspiration for the proposed system came from a yearly competition called RobotArt, where contestants created robots capable of making paintings in unique ways [3]. A common solution is to attach a drawing utensil to a computer numerical control (CNC) to create art from human inputs on a computer, similar to what is seen from Li et al. [4]. The closest existing product to the CNC portion of the proposed system is the WaterColorBot!, launched via Kickstarter, a commercial product that produces a watercolor painting from a scalable vector graphics (SVG) input file [5].

There have been several attempts at creating a webcam-based gaze estimation software, both in the literature as well as commercial products. Zheng et al. [6], Kannan [7], and Falke et al. [8] have previously published works on the topic. WebGazer.js [9] and GazeRecorder [10] are existing products that claim to accomplish this goal. These works and products show that the approach can be made to work with a variety of methods and that it is viable to produce a custom implementation as a component of this system.

The contributions of this paper include an implementation of a webcam-only gaze estimation software, a novel user interface for painting that is scalable and usable with a low accuracy gaze estimation solution and specific modifications to an existing 3D printer design that enables it for this use case. Section 2 details the approach taken towards creating and evaluating a solution to the problem. Section 3 describes the design of the system using block diagrams, flowcharts, user interface mockups and CAD drawings. Section 4 displays the results of the prototype including images of paintings created with the system and data summaries from system testing. Section 5 provides an analysis of the prototype, along with further research and development suggestions. Finally, section 6 offers a conclusion and closing thoughts.

2. RESEARCH METHODOLOGY

This paper presents EyePaint, an end-to-end system that produces a physical acrylic-on-canvas painting while using webcam-based gaze estimation as its only input mechanism. The novel work provided in this paper is not only the custom implementation of gaze estimation, but the user experience work provided into how these pieces fit together into a coherent system.

The prototype of the system, concept, and analysis presented in this paper is produced by following a slightly modified waterfall design methodology. The process begins with a survey of existing products and research to study previous solutions. The learnings are applied in the design process to synthesize a set of requirements and determine which

approaches to the problem may be viable to achieve the goal. The design is then taken and iteratively prototyped to a complete implemented system.

The effectiveness of the final system is evaluated using a set of system and acceptance tests. The functionality of each system module is independently, quantitatively verified using custom testing software that collects data samples over the course of specific test cases and provides summary statistics for each case. These summary statistics are directly compared to internal requirements and expectations from existing research. Acceptance testing is performed qualitatively by the authors to determine whether the system broadly meets its goal of being human usable without hand input.

This approach in addressing the problem statement is both time consuming and difficult to experiment with, but it is necessary because of the large amount of engineering that must be done to produce a system prototype with complex hardware and software that can be evaluated.

3. SYSTEM ARCHITECTURE AND DESIGN

The system architecture is broken down into three main modules: the gaze estimation software, the graphical user interface (GUI) and the CNC. The gaze estimation software and GUI are both part of the same software package that runs on a user's personal computer. That computer is connected to the motherboard of a physical 3D printer-like CNC over a USB cable. The design of each of these three modules is enumerated in the following subsections. Fig. 1 shows a component breakdown of the system.

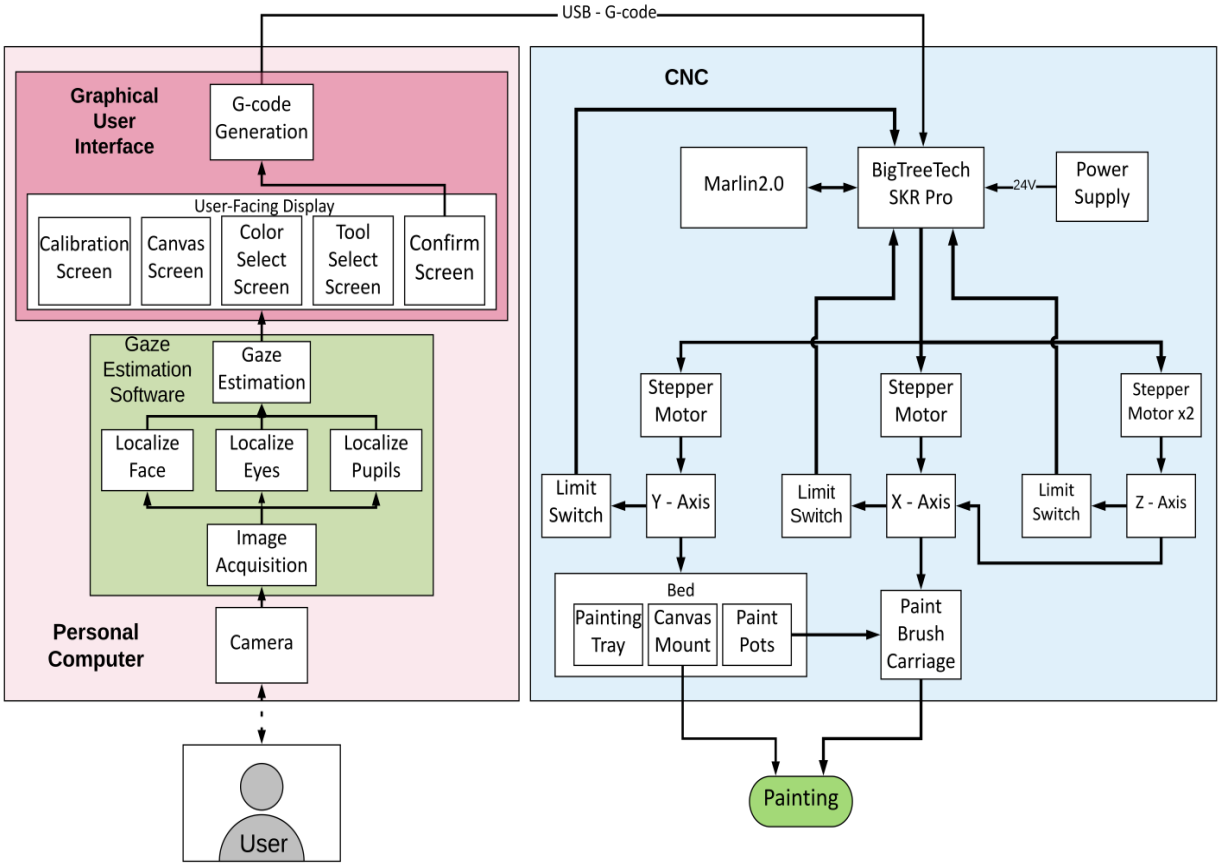

Fig. 1. System block diagram.

Both primary software modules are implemented as a single application intended to run on a personal computer. Fig. 2 shows a UML class diagram that describes the architecture of the total PC software package. The software is broken down into 14 different classes. App is the main class that contains most of the program code. The Python script runs the 'execute' method upon launch. The three main user interface components inherit from a Button superclass because of their similar needs regarding reactivity and interactivity. Tool and Color enums are used by multiple classes to standardize the methods when a color or tool is referred to by different parts of the code. The main program runs in a superloop structure where the code runs the 'loop' method, followed by the 'render' method, and then repeating ad infinitum.

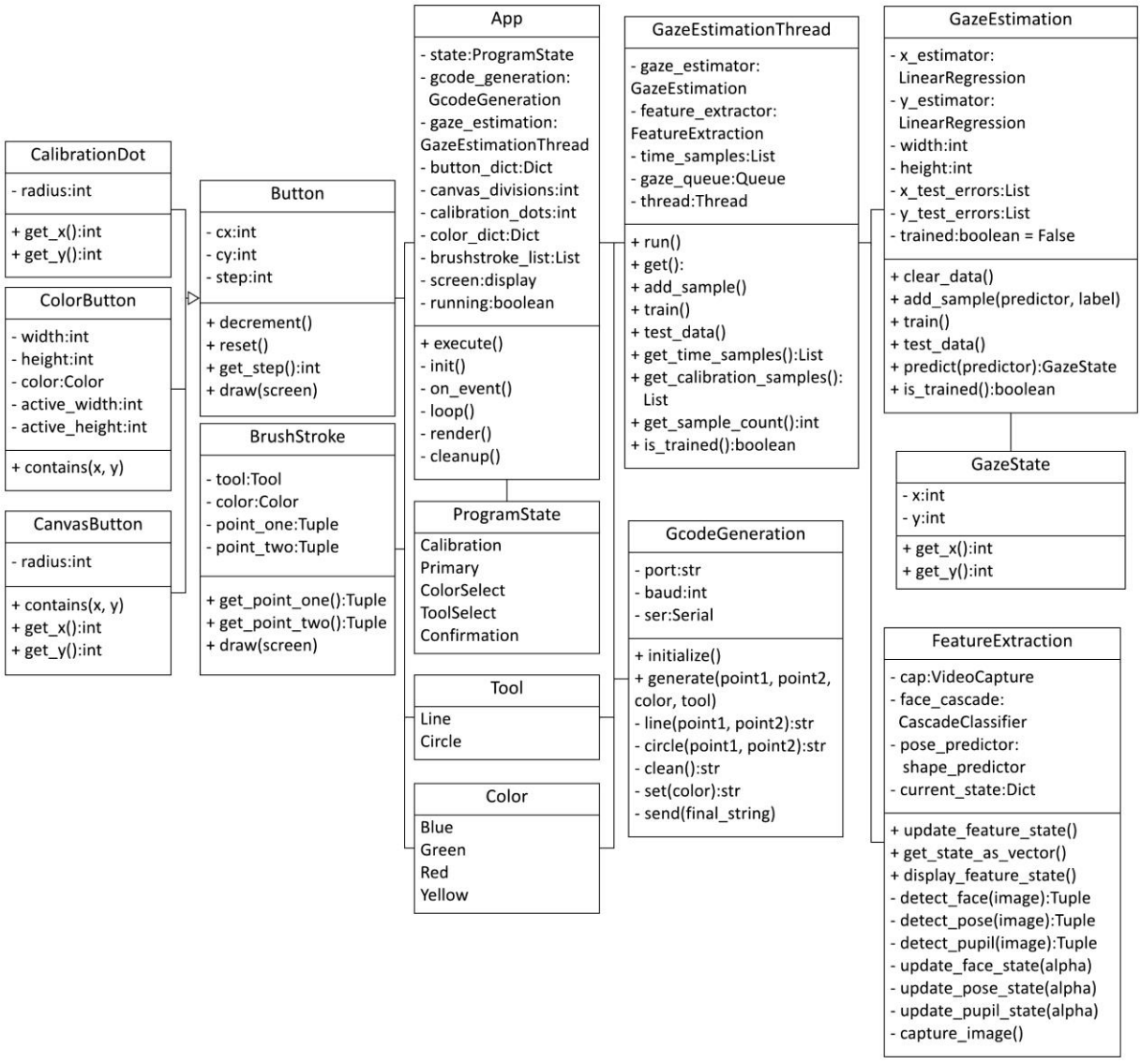

Fig. 2. Software implementation class diagram.

The gaze estimation code is contained within the GazeEstimationThread, GazeEstimation, and FeatureExtraction classes. The FeatureExtraction class provides code for taking in an image from the user's webcam and all computer vision algorithms applied to it until the resulting feature vector. The GazeEstimation class provides calibration by

maintaining an internal dataset, and contains two Scikit-Learn LinearRegressors to provide the predictions. The GazeEstimationThread class wraps both classes in a separate processing thread so that the user interface can run independently. Running the much slower gaze estimation code in a separate thread allows the user interface's loop to run as quickly as it can without being directly impacted.

The GcodeGeneration class enables the main interface for communicating with the CNC. The software calls the 'init' method immediately upon opening. Shapes are drawn simply by calling the 'generate' method that takes in two-point tuples and Tool and Color objects. This method constructs a complete string out of the recipes discussed in Table 1 and then sends that string over USB to the motherboard of the CNC using the PySerial package.

All of the code - for the design in this research - was written in Python because of the ease of prototyping and access to the existing computer vision and machine learning libraries to accelerate development. It makes use of the OpenCV, dlib, and Scikit-Learn libraries to implement the gaze estimation portions. The software uses the Python library PyGame to implement the GUI. Because of the lack of common existing user interface (UI) elements in its design, it makes sense to implement each display from scratch using basic shapes and text. The library also lends itself well to creating displays that animate and react to the user's input.

3.1. Gaze Estimation Software

In the existing literature, there appear to be two general approaches to creating webcam based gaze estimation software. The first involves classical computer vision algorithms [3], while the other is to produce an end-to-end deep convolutional neural network to perform the task [4]. Based on the lack of the hardware necessary to train such a network, as well as the difficulty involved with building a dataset necessary to train it on, the classical perspective approach to the problem is taken with the design.

The general flow is to first apply computer vision algorithms to create a vector of feature locations within the image, followed by mapping that feature vector to a Cartesian coordinate on the display. This approach is carried forward in this paper, while no specific implementation was followed as a set of instructions.

3.1.1. Feature Extraction

There are three primary steps and computer vision (CV) algorithms used in the Feature Extraction process. First, the user's face is detected and localized within the image. Second, both eyes can be localized within the face. Third, the pupils must be localized within the eyes. Fig. 3 describes the Feature Extraction process as designed at a high level; note that each of the aforesaid three sub processes is broken down in more detail.

Fig. 4 describes the designed process to localize the user's face within the entire captured image. Face localization is provided by OpenCV's pre-trained Haar cascade sliding window object detection model [11]. It is fast and accurate enough for the purpose. If multiple detections are made, only the largest box is used to filter out false positives or background faces. To help account for the high noise in detections, a time-weighted average with alpha value of 0.15 is applied to the detected bounding box.

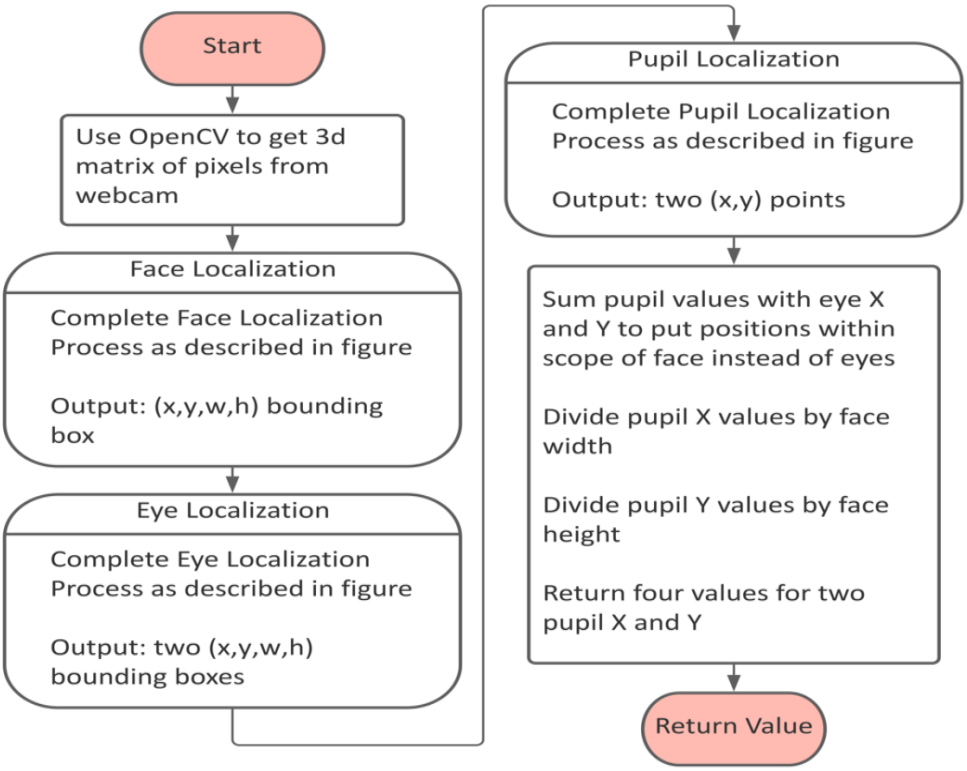

Fig. 3. Flowchart of the feature extraction process.

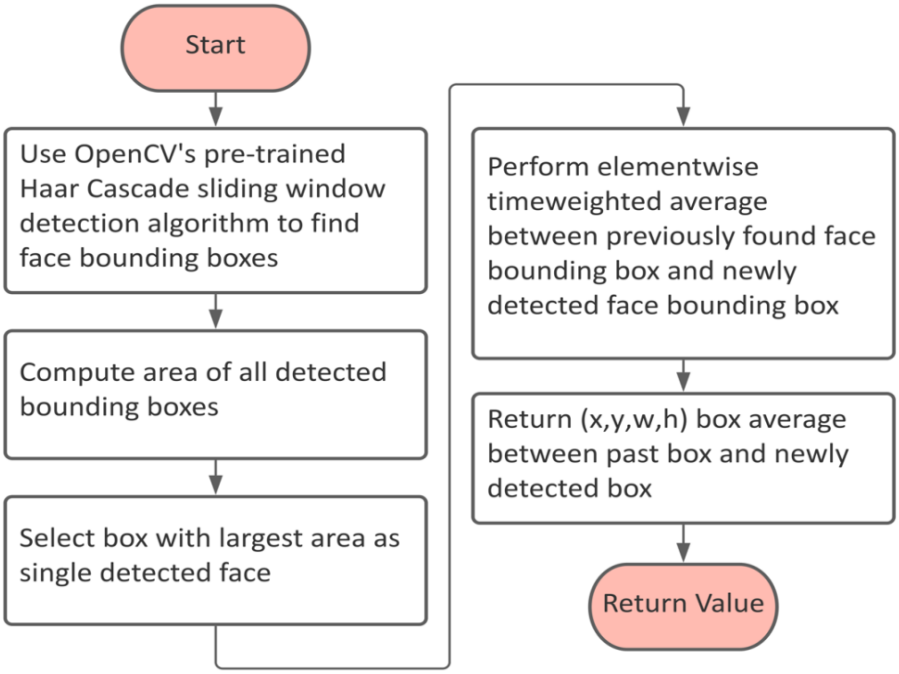

Fig. 4. Flowchart of the face localization process.

Fig. 5 describes the process to find the user's eyes, applied following the face localization. After the face region-of-interest has been cropped from the image, a pre-trained pose detection model provided by dlib is used. This algorithm fits a set of 68 landmark points with unique identifiers to the face region of interest (ROI) [12]. Haar cascades were also experimented with, but the chosen method was found to fit both eyes much more consistently with much less noise. The points numbered 37 and 40 are used to bound the

width of one eye, and the points numbered 43 and 46 are used to bound the other. These points each appear at one of the corners of the eye, and are used to calculate the width of a bounding box. The same value is used for the height so that each bounding box is square.

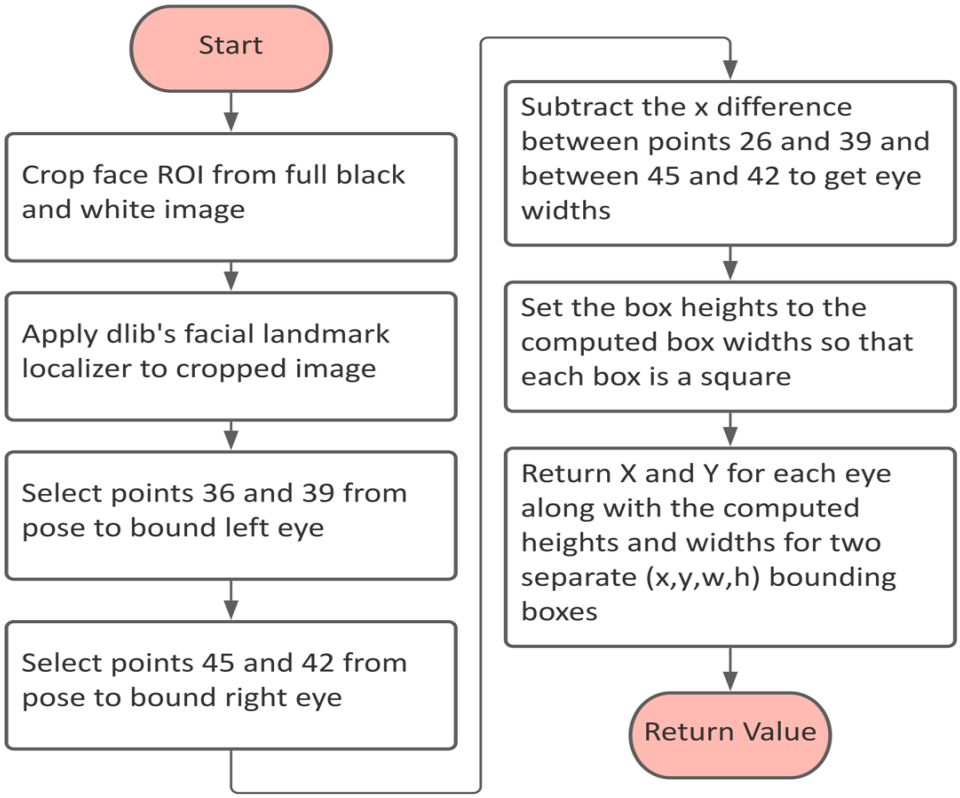

Fig. 5. Flowchart of the eye localization process.

Once each eye has a bounding box around it, those ROIs are cropped, and an eye center localization algorithm is applied to them. The algorithm is implemented according to the specifications in "Accurate Eye Centre Localization by Means of Gradients", by Timm et al. [13]. Fig. 6 describes the algorithm at a high level. The algorithm determines the pixel in the image that maximizes a function which aims to find the center of a circular region using the image's gradients. This proposal's implementation works decently well but can struggle at points when the user is looking too far downwards or upwards. In order to meet the real-time threshold defined early in the design of a frequency of 5 Hz, each eye ROI is scaled down to a 20-by-20-pixel image.

Once the three CV algorithms are complete, the pupil locations are post-processed and passed to the gaze estimation stage. Because the pupil locations are found within the eye ROI, the X and Y values from each eye bounding box are added to the respective X and Y values for each pupil. This places the pupils within the face's bounding box. The pupil locations are kept with respect to the face bounding box instead of the full image in an attempt to be invariant to changes in the location of the user's face within the image.

The pupil X and Y positions are then normalized by dividing them by the width and height of the face bounding box. This ensures that the values are always between zero and one and is an attempt to be invariant to changes in the distance from the camera.

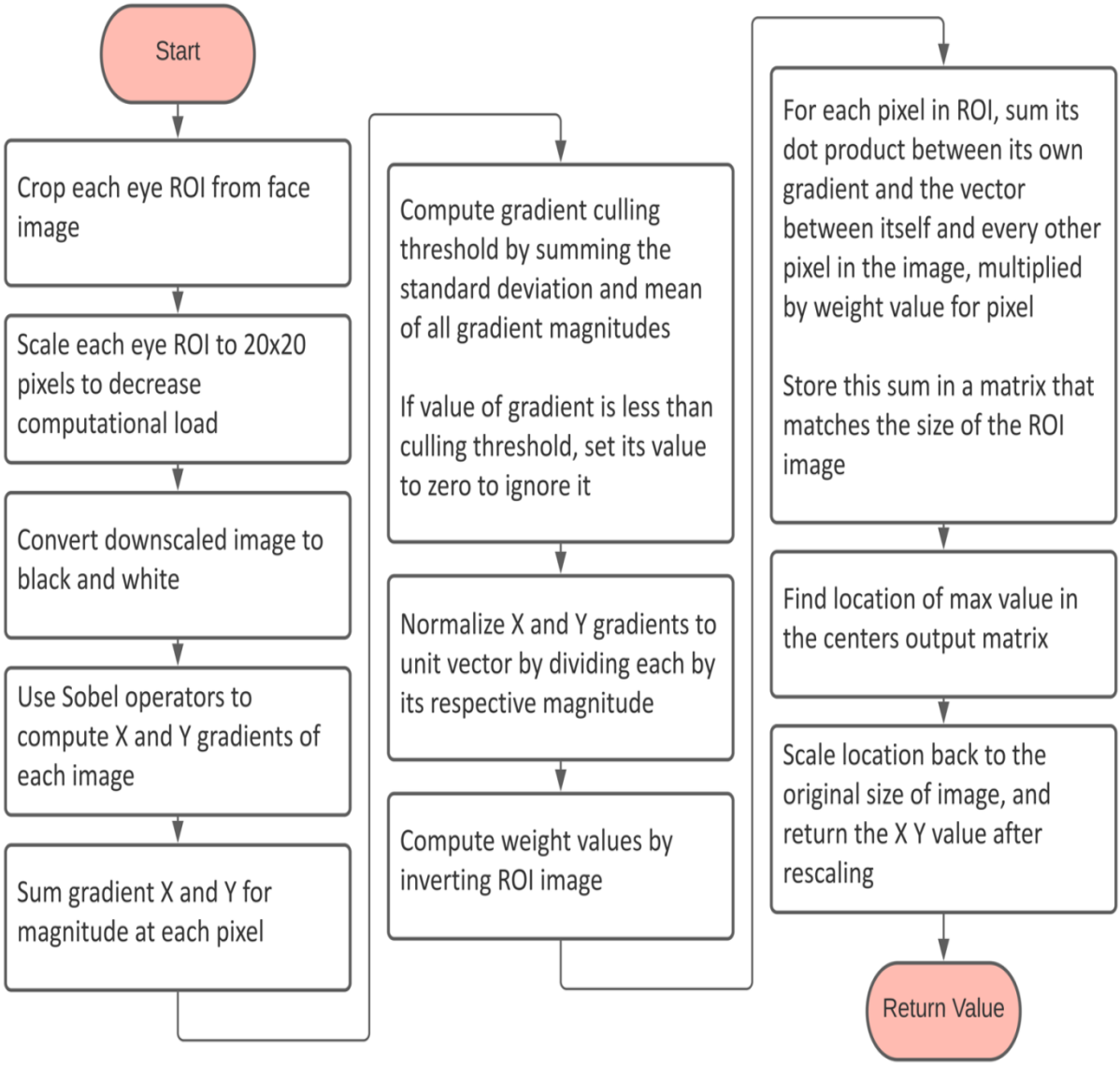

Fig. 6. Flowchart of the pupil localization process.

3.1.2. Gaze Estimation

The mapping of pupil locations to gaze estimation is provided by two linear regression models. One takes the x-coordinate of both pupils and returns the predicted x-gaze, and the other takes the y-coordinate of both pupils and returns the predicted y-gaze. Their parameters are learned from a required calibration process upon software start-up. Training the regressors - from scratch with each use - accounts for differences between environments, such as location of the webcam in relation to the display, or implicit physical differences between unique users.

The calibration process consists of tasking the user with looking at a sequence of dots spread across the screen. Only one dot appears at a time, with a continuously closing circle surrounding it that lasts for a period of three seconds. At the time that the dot closes, a snapshot of the pupil locations is taken and paired with the location of the dot to serve as a ground-truth for where the user should have been looking. This predictor-label pair is added to an internal dataset. The next dot in the sequence is then displayed and the process is repeated. This process is described as a flowchart in Fig. 7. No optimum total number of dots

has been determined but increasing the number of dots will result in a larger training set for the regressors and will presumably yield the effects associated with that. The final configuration chosen for the design is 16 dots set in a 4-by-4 evenly spaced grid across the entire display. This configuration works well enough for this purpose.

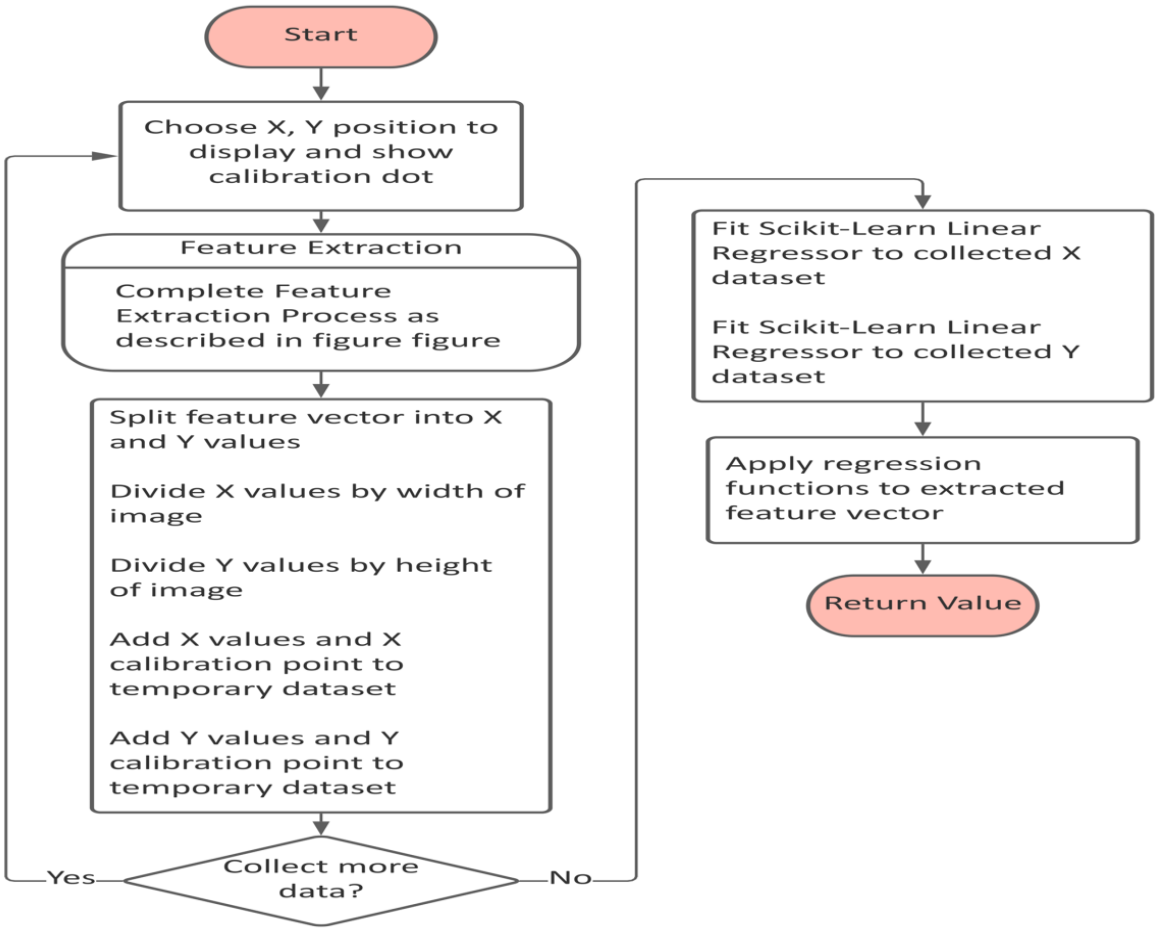

Fig. 7. Flowchart of the calibration process.

It should be noted that noise in the calibration process is a potential problem that should be addressed. There can be significant variations in how well the gaze estimation works for the end user depending on how well the data collection process went. If poor data is collected during calibration, the regression model will fit to that bad data and will not make predictions like the user is expecting. Solutions to this problem were not heavily investigated during development, but it should be possible to remove outliers from the dataset to keep the data clean or train a regression model with regularization to prevent overfitting to outliers.

3.2. The GUI

Designing the GUI around known constraints provided some of the key challenges. This module of the system is also the main piece that enables the gaze estimation and CNC to communicate and is how the user interacts with the product. Thus, every facet must be designed with gaze estimation in mind, with the additional knowledge that the gaze estimation is not particularly accurate.

The design also attempts to consider previously documented issues with user interface where gaze estimation is the only input mechanism. First, the Midas touch problem of gaze estimation, where each action the user takes is inherently input [14], makes it difficult to discern whether a user intends to take an action or not. There is not an easily differentiated click, so everywhere the user looks can be read as clicking on that button. This becomes an added concern when considering how a user should pick up or set down a brush with only a two-dimensional input. Secondly, the eye has a tendency to wander, so giving a user as small a button as possible to focus on during a 'click' is a priority.

Because of the issues mentioned, the design constrains the user to only discrete shapes instead of freeform lines, with these shapes only being drawn to the canvas once a user is happy with the result. Li et al. [4] found that users preferred digital to physical painting systems where the user draws a line in two separate steps: by first inputting it on a display and then committing it to the canvas. They also found that users preferred this input system over one where the user's input mapped live from digital to physical.

The design intentionally limits the user to four possible colors, and two tools, a line and a circle. The number of colors is limited by the size of the painting tray and the accuracy of the gaze estimation. Although the choice of tools and colors is limited, there is no inherent limitation from adding more.

The user interface consists of five distinct screens that will be individually described in the following section:

a) Calibration
b) Canvas
c) Color Select
d) Tool Select
e) Confirmation

Each screen is governed by the same set of design principles. All buttons have a hidden active area significantly larger than the visible button. Each starts with a full shape of varying size depending on type that proceeds to get smaller as the gaze prediction is held within that button's active area. This draws the user's eye in while he focuses on the button. Upon the shape hitting the point at which it has zero area, the button is thereby clicked, the screen updates and the user can move on to their next action.

In more simple terms, a user is interpreted to have clicked on a button if their gaze is held within the same active area for three continuous seconds. If the user's gaze leaves a particular active area before three seconds have elapsed, the timer is reset and nothing is clicked. This is how the software is able to distinguish between the user's intention while they are looking around.

3.2.1. User-Facing Display

Fig. 8 shows the design mockup for the calibration screen of the GUI. The calibration screen implements the process described in the previous section. It is the first screen shown to the user and runs on its own to completion. The software pauses for a few seconds upon start-up to give the gaze estimation algorithms time to get their bearings, as well as allow the user to orient themselves before the process begins.

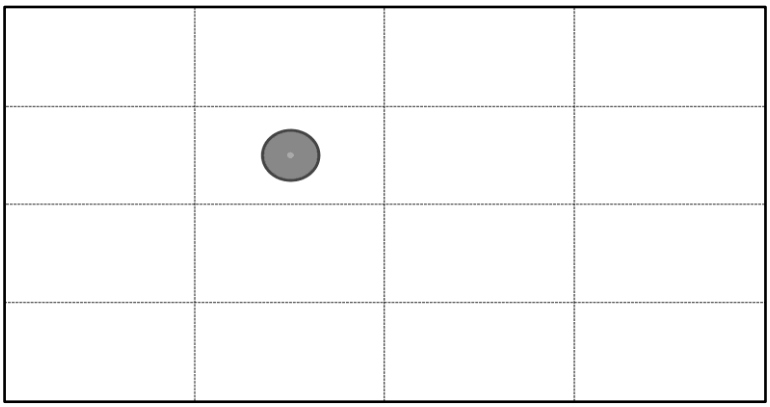

Fig. 8. The calibration screen.

Fig. 9 shows the design mockup for the canvas screen of the GUI. The canvas screen is the main display that each of the following three screens returns to. It contains two large buttons on the right and left sides of the screen. The left button is used to move to the color select screen, and the right button is used to move to the tool select screen. By selecting either button as described earlier, the screen immediately switches to the color select screen or tool select screen. After a selection has been made on either of the selection screens, the interface automatically returns to this display for the user to use the color or tool they just selected.

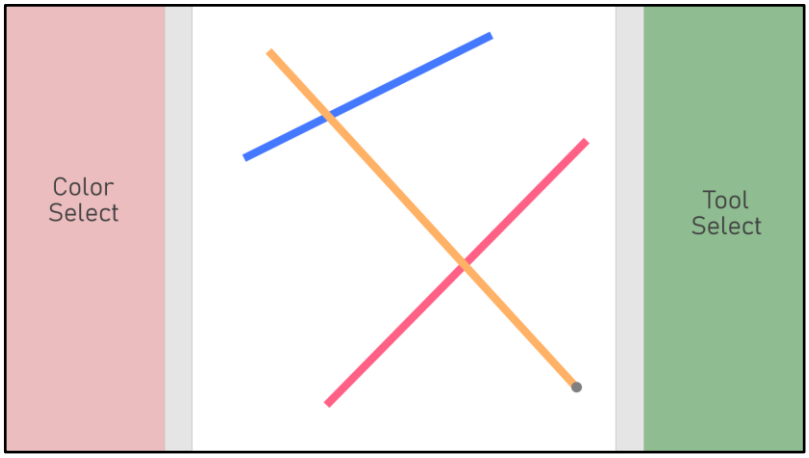

Fig. 9. The canvas screen.

The center of the screen is used as a virtual canvas. The locations that a user can select are discretized into a grid of anchor points that can be selected as buttons when drawing a shape. This choice constrains the user to a discrete set of points that can be used when drawing to allow the user to focus on a concrete point instead of a hypothetical continuous position on the canvas. Both shape types can be drawn by selecting two of the anchor points. For the line shape, the first point selected is the start of the line and the second point selected is the end of the line. For the circle shape, the first point selected is the circle's center and the second point selected sets its radius. Based on the final accuracy of the gaze estimation a 4-by-4 grid of points was chosen, but the software is implemented to allow for the size of the anchor point grid to be configurable at startup.

Fig. 10 shows the design mockup for the color select screen of the GUI. This screen serves the task of allowing the user to choose which color their stroke will be drawn with. The system is designed around four color options, so this screen can be split into quadrants.

Each button rests in the center of a quadrant, and its active area takes up the entirety of its own quadrant. The borders of the active areas are depicted by the vertical and horizontal dashed lines shown. The color that will be selected is naturally depicted by the color of the button on screen. Once a color has been selected, the display can be returned to the canvas screen.

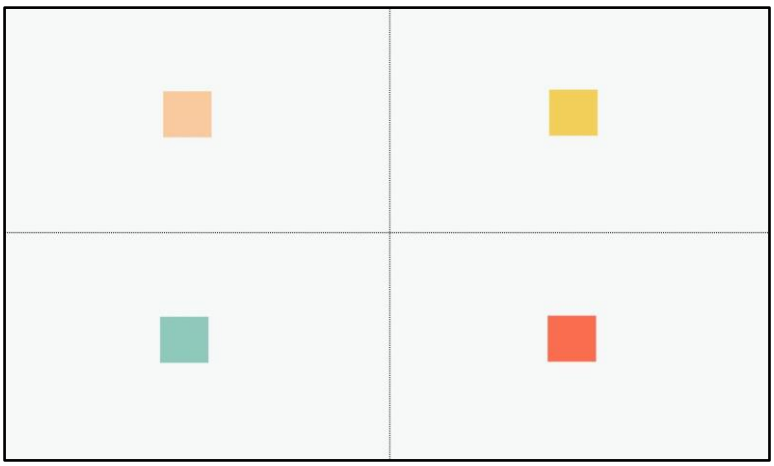

Fig. 10. Color select screen.

Fig. 11 shows the design mockup for the tool select screen of the GUI. The tool select screen allows the user to choose between the two tool options. The design is limited to just a line and circle tool due to their simplicity, but more can be added. Because there are only two, the screen is split into two equal halves vertically down the middle. Each entire half of the screen is used as each button's active area.

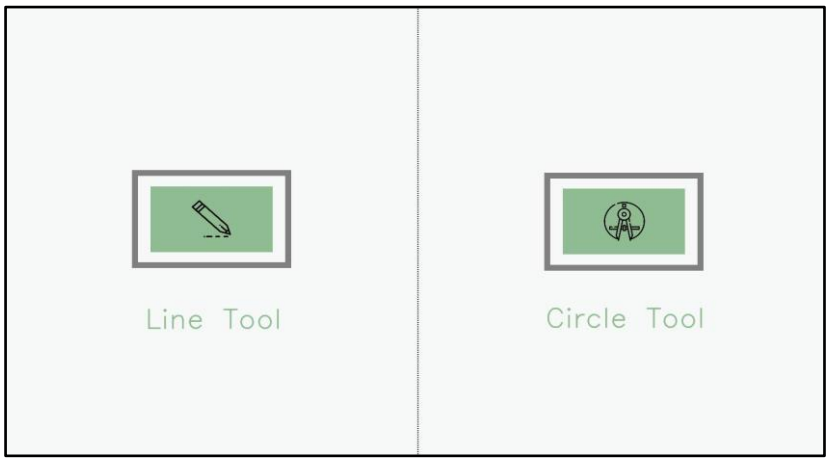

Fig. 11. Tool select screen

Fig. 12 shows the design mockup for the confirmation screen of the GUI. The confirmation screen appears after a complete shape has been drawn on the canvas. It is represented by an overlay of the canvas screen with two options for the user to select. A small region in the center of the screen is kept as a neutral area for the user to look while making a decision. The left option cancels the stroke and deletes it from the canvas. The right option firmly commits the stroke to the canvas by initiating the G-code generation process that sends a set of commands to the CNC.

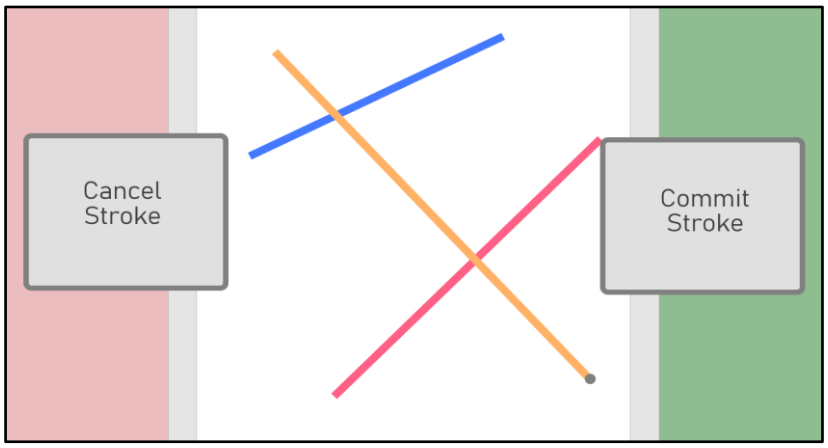

Fig. 12. The confirmation screen.

3.2.2. G-code Generation

The GUI controls the CNC by transmitting a G-code string over USB upon a stroke being committed. The G-code generation process, shown in Fig. 13, receives the currently active tool and color, and the two points entered on the canvas.

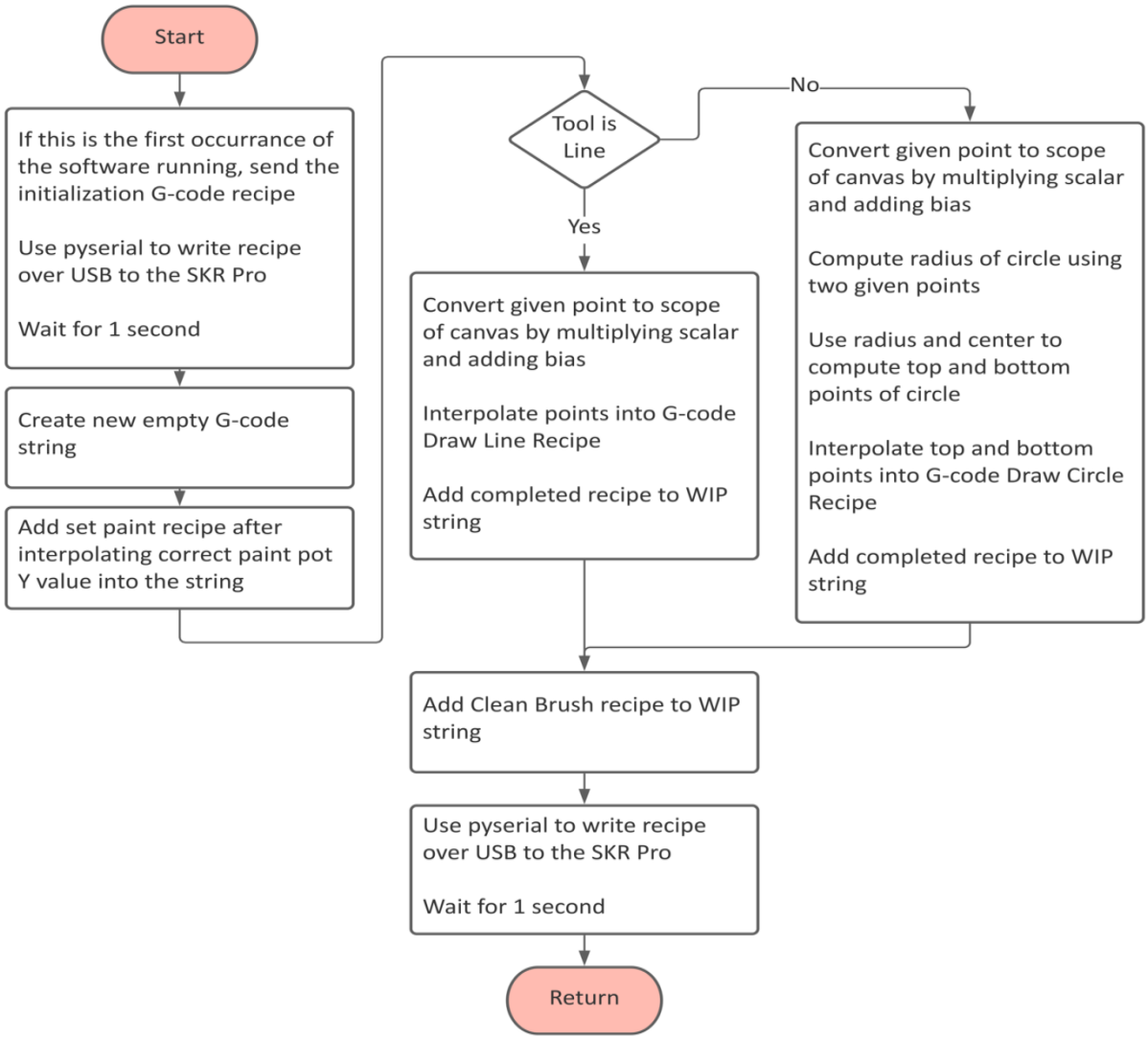

Fig. 13. Flowchart of the G-code generation process.

These inputs are used to populate values into empty pre-written G-code recipes. A final string consists of the recipes for loading the brush with paint from a paint pot, drawing a circle or line depending on the tool, and cleaning the brush on the cleaning rag appended together. The points used for the draw recipe have a scale and bias applied to them to convert from the location on the virtual canvas to the location on the physical canvas.

Table 1 shows the final G-code recipes that were used in the design. The speeds and locations for each routine are determined through some calculations and trial and error. The coordinates are measured in millimeters, and each motion is made with absolute positioning so that the software on the PC does not need to keep track of the position of the carriage. In the table, a <> denotes where a value is interpolated into the string during the G-code generation process. Any X, Y, or Z value that has a defined number in it is not modified at all. The setup string is sent as soon as the program begins to run, and it sends basic configuration commands to the motherboard.

Table 1. The G-code recipes.

Routine number	Routine name	G-code recipe
1	Set-up	G21 ; Set to measure in mm M92 X40.0 Y40.0 Z400.0 ; Set steps per unit G90 ; Set to absolute positioning G28 ; Rehome motors G01 Z34 ; move up to main height
2	Draw straight line	G01 X<> Y<> Z34 ; move over first point of line G01 X<> Y<> Z14 ; move down to canvas G01 X<> Y<> Z14 ; move to end point of line G01 X<> Y<> Z34 ; move back up from canvas
3	Draw circle	G01 X<> Y<> Z34 ; move over beginning of circle G01 X<> Y<> Z14 ; move down to canvas G02 X<> Y<> R<> ; draw first half of circle G02 X<> Y<> R<> ; draw second half of circle G01 X<> Y<> Z34 ; move back up from canvas
4	Clean brush	G01 X295 Y54 Z34 ; move over water pot G01 X295 Y54 Z14 ; move down to water pot G01 X295 Y54 Z34 ; move back up from water pot G01 X270 Y20 Z34 ; move over cleaning rag G01 X270 Y20 Z14 ; move down to cleaning rag G01 X320 Y20 Z14 G01 X320 Y0 Z14 G01 X270 Y0 Z14 G01 X270 Y20 Z14 G01 X320 Y20 Z14 G01 X295 Y20 Z34 ; move back up from canvas
5	Load brush	G01 X295 Y<> Z34 ; move over paint pot G01 X295 Y<> Z4 ; move down to paint level G01 X295 Y<> Z34 ; move back up to main level

A complete G-code string consists of a load brush recipe, followed by either a line or circle recipe, and finished with a clean brush recipe. Each time a shape is drawn, the CNC will go through the motions of reapplying paint to the brush and then cleaning the brush. While time consuming, this process guarantees that there is enough paint on the brush for each line, and it allows the color to be changed between every stroke.

3.3. CNC

The CNC module, as shown in Fig. 14, physically creates the painting with acrylic paint on canvas. It takes in G-code over USB and performs the given movements to create the final product, the painting. For the sake of simplicity, this design is heavily based on an open-source RepRap 3-D printer. The main source is the C201 by Makers Mashup which takes from other printers itself. The C201 design is altered slightly with some modifications, mainly to the bed, to facilitate painting.

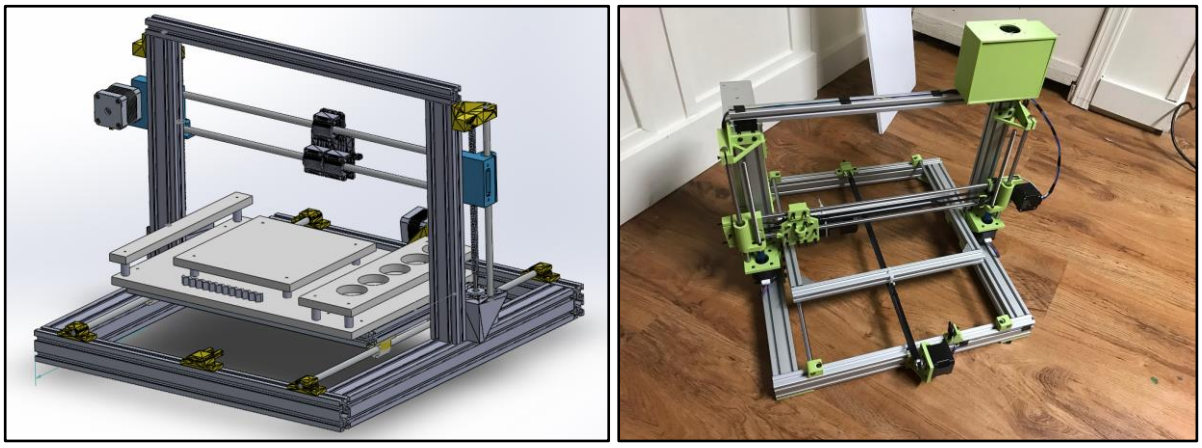

Fig. 14. Side-by-side CNC CAD assembly and WIP build.

The frame is altered to allow a 10x10″ canvas and a paint tray next to it, rather than the previous 8.4x8.4″ print bed. In order to keep the CNC frame small, the paint tray plate overhangs its frame to allow movement over both sides of the CNC frame. The frame is 19.7x18.3x14.6″ which is small enough to fit on most tables. A simple alteration is the paintbrush mount which is 3D printed and uses two set screws to hold the brush. This replaces the extruder of a printer, and has the same back plate so it can be swapped in. The drawing of this custom part is shown in Fig. 15.

The bed is the bulk of the unique mechanical design work. The main goal of the bed is to hold the canvas, paints, water and cloth in a way that makes swapping them out uninvolved. The secondary goal is to make the paint tray and canvas vertically level to eliminate unnecessary slow vertical movements of the CNC carriage. As shown in Fig. 16, it consists of four 10 mm coroplast plates and four 3D printed standoffs. Coroplast is cleanable, lightweight, and easy to machine, so it suits the application. The four plates connect to a large baseplate for mounting.

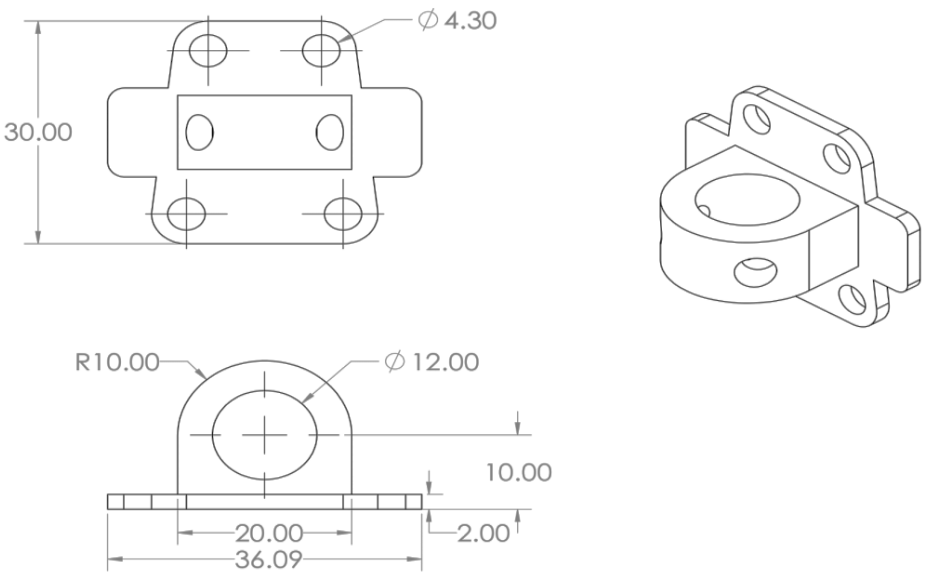

Fig. 15. Paintbrush mount CAD drawing.

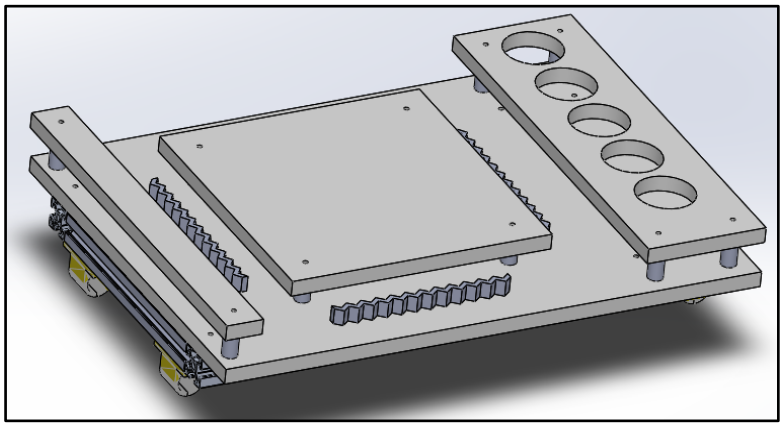

Fig. 16. Complete bed CAD assembly.

The paint tray has a total of five holes, four for paints and one for water, and an open space for a cleaning cloth. These holes hold small paper cups, like the ones found to hold condiments at a fast-food restaurant, allowing the user to swap them out as they wish. The tray stands off the baseplate far enough to allow the cups to hang down through the tray.

The canvas plate holds the wooden frame of the canvas from the inside, and the side guard and paint tray hold it from the outside. This boxes in the canvas enough to keep it from sliding but allows the user to simply set it in place without any complex mounting system. The canvas is lifted with standoffs to be at the same height as the paint tray.

The main motherboard used in the CNC is the SKR Pro V1.3 by BigTreeTech. The stepper motor drivers used are BigTreeTech TMC2208. The board and stepper motors are powered by a 24 V power supply, capable of supplying a current of 20 A.

The board is flashed with the Marlin2.0 firmware. Some parameters were modified by the authors to serve the purpose of the system. These include increasing the size of the CNC's bed, tweaking the number of steps per unit for the Z-axis, and disabling additional motor drivers that are not used by the design.

4. RESULTS AND EVALUATION

The final prototype came together almost entirely to the authors' expectations. Fig. 17 shows a sample of paintings made using the full EyePaint system. These examples show a good variety of the shapes it is possible to draw with the system prototype.

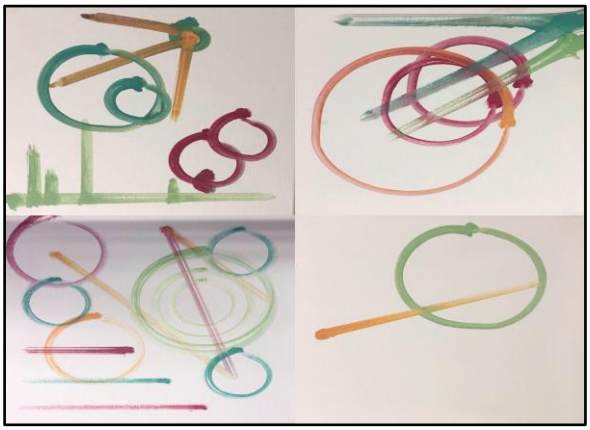

Fig. 17. Example paintings created with the system.

Fig. 18 depicts an image of the GUI as well as the physical painting that results from it. This example makes use of all four colors available, as well as a variety of examples of the two shape tools. It can be clearly seen how the lines on the digital canvas map to the physical version.

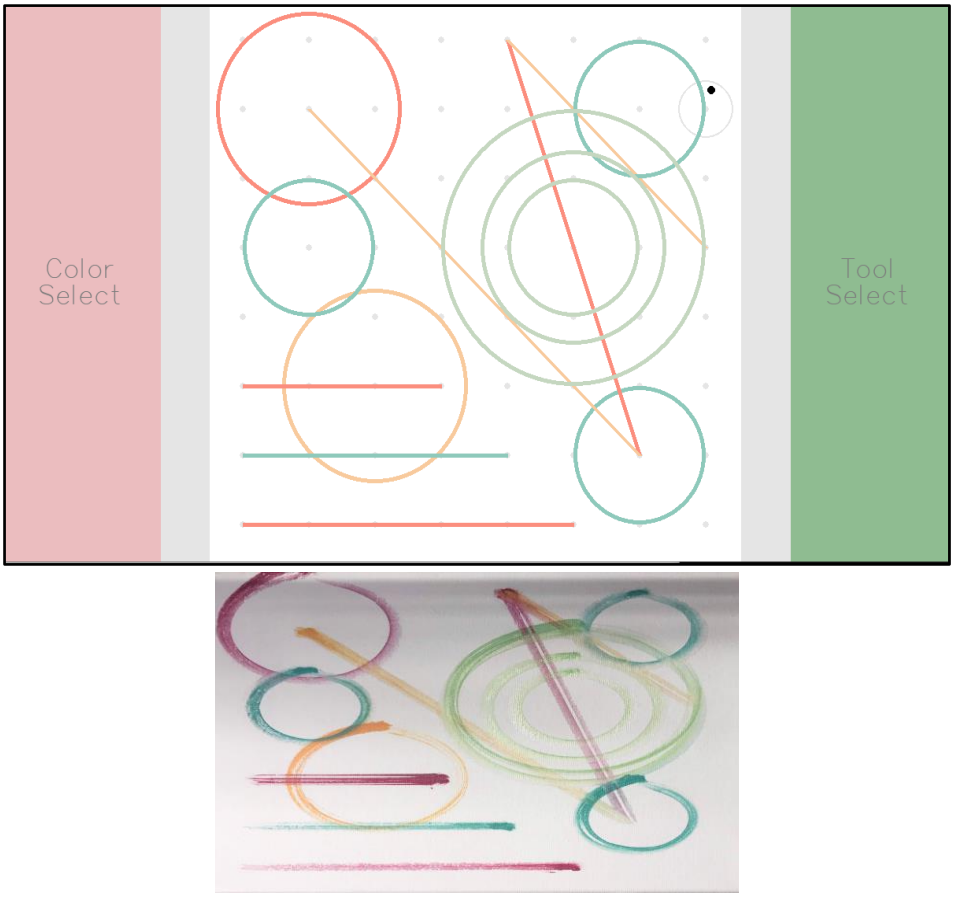

Fig. 18. Side-by-side of how GUI input results in painting output.

Fig. 19 shows the complete bed of the CNC. The image on the left is prior to any paint or canvas being added. The stand-offs and canvas mount are clearly visible, as well as the five holes for paint pots and Velcro for attaching a cleaning rag. The image on the right shows the bed after paint pots, a cleaning rag and a canvas have been placed inside.

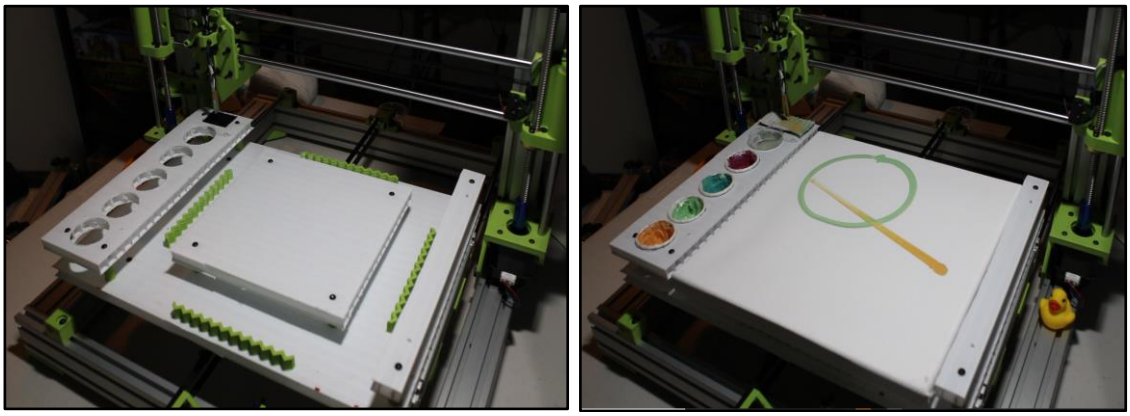

Fig. 19. The CNC painting bed with and without materials.

Each of the system modules are partially evaluated through a set of system tests. The computer used for testing and demoing the system contains a Ryzen 5 1600, six-core 3.6 GHz CPU, 32 GB of DDR4-3200 RAM, and a 1080p 24" monitor. The software does not make use of GPU acceleration. The webcam is a Wansview 101JD, with a 1080p image resolution and a capture rate of 60 FPS. The procedures and results of these tests are summarized in the following subsections.

4.1. Gaze Estimation Testing

Gaze estimation tests are performed using a special custom testing program that performs experiments and collects corresponding data samples. The program begins with the same calibration process as the main GUI, but instead of displaying screens for user input, the software will place a new calibration-like dot at a random position on the display. Behind the scenes, the software splits the display into a grid of rectangles of a size specified upon startup. After a dot has fully closed, the gaze estimation prediction will be recorded, and the software determines which grid square the gaze prediction lies within. The Manhattan distance is calculated between the predicted rectangle and the ground truth, and a new dot is shown and the process continues. For each trial during these tests, a total of 50 dots are shown. Because of the potential variation in the quality of calibration data, it is very difficult to standardize the environment between each test. It may be a better idea to record samples of different grid sizes using the same calibration parameters, or switch between grid sizes in between samples, but that is not the approach taken.

Table 2 shows the precision measurements taken over the course of six different trials of the testing software. The data are collected in a room lit primarily from the ceiling. They naturally show that as the grid size increases, the ability of the software to correctly predict the specific square the user should be looking at decreases. The second line shows the precision measurements taken on a very similar six trials, but in a dark room where the user's face is lit only by the display. This change in environment massively increases the

performance of the software. In this environment, generally only the face is visible in the image, and it is possible that this allows for a decrease in noise in the face and pupil detections.

Table 2. Results of the gaze estimation precision test.

Environment	2x2 Grid	3x3 Grid	4x4 Grid	5x5 Grid	6x6 Grid	7x7 Grid
Lit room	90%	82%	42%	22%	16%	14%
Dark room	96%	68%	78%	36%	32%	46%

Additionally, the accuracy of the gaze estimation software can be measured in an average error as a percentage of screen width or height that the prediction is off by. This metric is also recorded on the same testing samples as the precisions detailed previously, but it is calculated differently by finding the absolute value of the center of the grid rectangle subtracted by the raw gaze position. It is then divided by the width or height of the display in pixels to obtain a percentage for either the X or Y axis respectively. The mean error on the X-axis is found to be 8.75%, and the mean error on the Y-axis is found to be 14.64%. A boxplot of the 50 samples collected during a trial is shown in Fig. 20. These results show that the gaze estimation performs significantly more strongly on the X-axis than it does on the Y-axis.

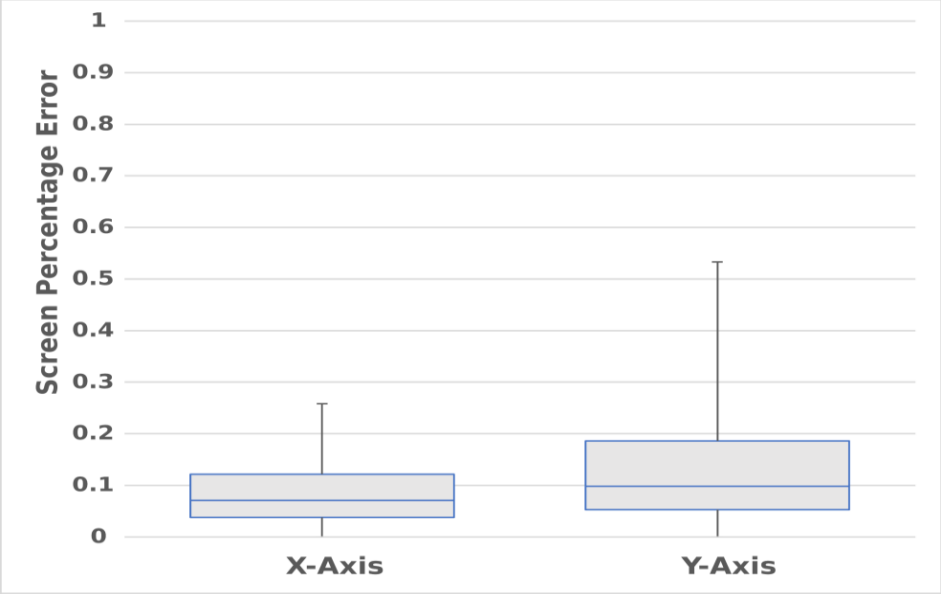

Fig. 20. Boxplots of raw gaze estimation error test samples.

On average, gaze estimation predictions update at 5.18 Hz, with the histogram in Fig. 21 showing a set of period samples collected during a trial of the software. The software is tested on a computer with a Ryzen 5 3600 6-core 3.6 GHz CPU with no GPU acceleration. This test is performed to determine whether the software met an internally set real-timeness requirement of a 5 Hz update frequency.

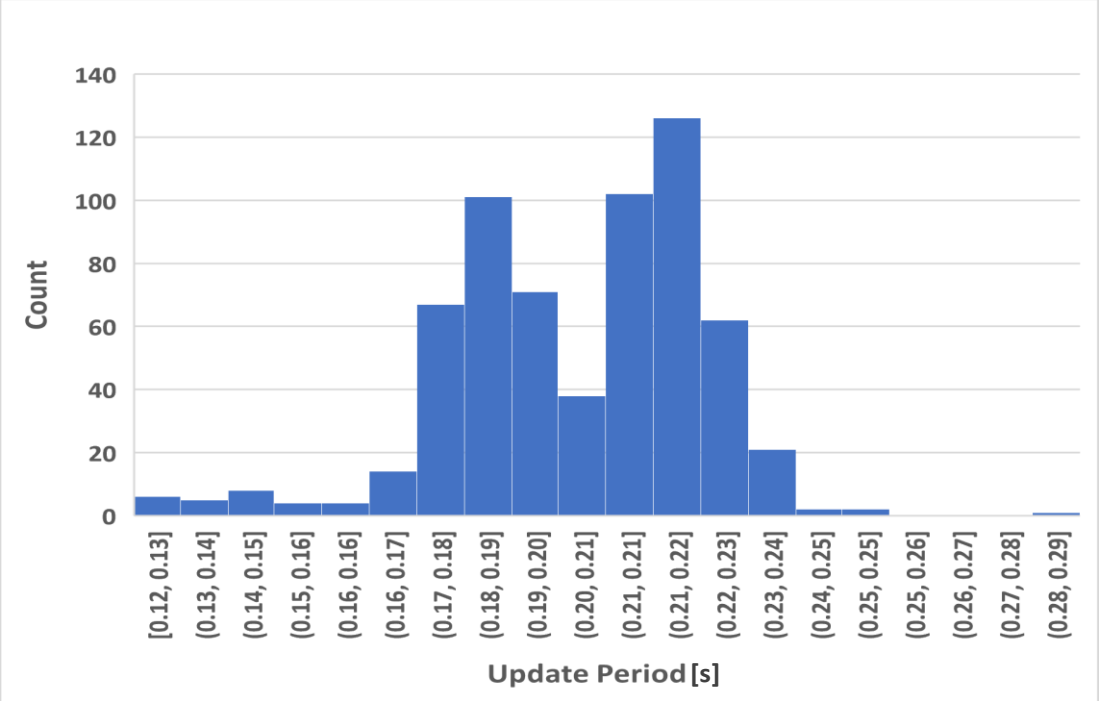

Fig. 21. Histogram of gaze estimation update speed samples.

4.2. CNC Motion Testing

The speeds and accuracies of each of the three main CNC motion routines are tested by creating a small test set of inputs, sending those inputs to the CNC, and timing each routine and measuring how far from the specified point on the digital canvas the carriage ended up from on the physical canvas while drawing the line. The results from each of these tests are then averaged.

The paint routine has an average completion time of 4.22 s, the load brush routine has an average completion time of 10.37 s, and the clean brush routine has an average completion time of 16.72 s. Overall, between the CNC receiving a string and motion ceasing, the routines average 31.31 s. On the paint routine, the average error measure between where a stroke is specified to end on the digital canvas versus where the CNC carriage ends up on the physical canvas is 0.64 inches.

Some of these motions are intentionally kept slow to avoid splattering paint and maintaining clear strokes. The set paint routine and clean brush routine are clearly longer than the paint routine. This is a result of those two motions containing actions on the Z-axis. The design's threaded rods that move the CNC's Z-axis of motion have a hard limit on how fast they can be rotated. This mandates that motion on that axis is slow and most of the time spent in motion is spent on moving the brush up and down.

5. DISCUSSION AND ANALYSIS

5.1 Further Development

The framework - provided in this paper - has multiple points at which it is easily extensible. The boundary between each of the three main modules is sufficiently abstracted away so that modifications to one can be made without impacting the others. With a high

accuracy gaze estimation solution, the resolution of the canvas display's discrete anchor grid can be increased by a configuration variable for greater variety of control by the user. Adding an additional shape tool should be a matter of creating a new button on the tool select display, writing the code to render that shape on the virtual canvas, and formulating a new G-code recipe to be filled in and sent over USB to the CNC. Changing the set of four colors available in the palette can be easily configured with the color select screen. On the other hand, increasing the number of paint colors that can be used simultaneously would be significantly more difficult due to the physical nature of requiring room on the CNC's bed to fit more paint pots.

The linear nature of development on the prototype system gave little room for revision if a specific approach was not feasible. Given an opportunity to redesign the premise from scratch, the main points the authors would focus on would be first the quality of the gaze estimation, and second the specific way gaze estimation interacts with the CNC.

A more 'smart' calibration process is one of the key priorities, as well as reevaluating and fine-tuning the used computer vision algorithms. More formal low-level testing should also be performed between different sets of possible inputs to the linear regression models, as well as trying a larger variety of more complex models to determine whether the approach used is actually the best. Additionally, it may be worth discarding the webcam-based gaze estimation in favor of a more precise commercial product that may work better as an input to the system.

With an improved gaze estimation solution, it may be worthwhile to explore other options for allowing a user to control the CNC using gaze, such as opening it up for free-form shapes or potentially more real-time input tracking. These are interesting ideas that were not explored during this work due to time constraints.

5.2 Comparison to Existing Research

Comparisons between this system and others may be difficult to draw because it appears to be the first end-to-end system of its kind. In addition, clinical trials were not performed to evaluate its true efficacy as an assistive technology. While these reasons make the entire system difficult to evaluate, comparisons can be drawn between an individual module of this system and those like it.

The accuracy of gaze estimations solutions with similar approaches can be directly compared. Authors in [6] provided testing results that were performed with a similar procedure to what is used in this work. Their design produced an accuracy of 94% on a 3x3 grid and 78% on a 5x5 grid. Authors in [7, 8] used different testing procedures that are not as easy to directly compare. In [7], authors found an average error of 1.66 cm on a 13.94 cm iPhone display, making the total error 11.9% of the screen size. In [8], authors found an average error of 262 pixels on the X-axis and -487 pixels on the Y-axis on a 14" 1080p display, making the error 13.64% of the total screen on the X-axis and 45.09% on the Y-axis.

As described in section 4.1, the solution designed and implemented for this system provided results that were either more poor or in line with those found in these works. As previously stated, the best result obtained by this work's gaze estimation is 82% on a 3x3 grid and 36% on a 5x5 grid. Both are well below those described in the previous paragraph. By the second recorded metric, the average percentage error of 8.75% on the X-axis and

14.64% on the Y-axis places this solution roughly in-line or above the other two webcam-based gaze estimations referenced. In the end, it appears that the implementation created could use a good deal more tuning or a reassessment of the design, and could hint that some of the negative user experience aspects could be averted with a better implementation of the same concept.

6. CONCLUSIONS

Overall, the final system satisfies the majority of the specifications and provides a potential solution to the problem posed. The webcam-based gaze estimation does not reach the original accuracy expectations, but still results in a product that is acceptable. This method of gaze estimation is overall the wrong choice for this concept, and it is not suited to be used as an input mechanism in its current state, but the work done in the user interface will still hold true with higher accuracy solutions. In general, any of the three main modules of the system can be swapped out with a different implementation and still achieve the same goal.

As an independent research work, gaze estimation with a simple webcam is still an interesting concept that is worth pursuing, but does not currently have a clear state-of-the-art. It is still very possible that there are innovations that have not been discovered during development of this work, and it is possible to produce gaze estimation software that is more accurate. The low-cost webcam only approach has application as a research tool when there's a need to record approximate gaze.

Aside from the issues produced by the choice in gaze estimation solution, the concept of the system proves to be strong given some potential changes in the design. It is clear that the technical aspects of the concept can work, although it still remains to be acceptance tested with persons with disability to evaluate the true viability of its intended application.

REFERENCES

[1] D. Perera, R. Jim Eales, K. Blashki, "Supporting the creative drive: investigating paralinguistic voice as a mode of interaction for artists with upper limb disabilities," *Universal Access in the Information Society*, vol. 8, no. 2, pp. 77–88, 2008.

[2] C. Creed, "Eye gaze interaction for supporting creative work with disabled artists," *Proceedings of the 30th International BCS Human Computer Interaction Conference*, pp. 1-3, 2016.

[3] *The Robotic Art Competition: RobotArt*, 2021. <http://robotart.org/>

[4] J. Li, J. Jacobs, M. Chang, B. Hartmann, "Direct and immediate drawing with CNC machines," *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, pp. 1-2, 2017.

[5] *WaterColorBot!*, 2016. <https://watercolorbot.com/>

[6] C. Zheng, T. Usagawa, "A rapid webcam-based eye tracking method for human computer interaction," *2018 International Conference on Control, Automation and Information Sciences*, pp. 133-136, 2018.

[7] H. Kannan, *Eye Tracking for the IPhone Using Deep Learning*, Massachusetts Institute of Technology, 2017.

[8] M. Falke, L. Höglund, *Implementing Gaze Tracking with a Simple Web Camera*, KTH Royal Institute of Technology, 2019.

[9] A. Papoutsaki, P. Sangkloy, J. Laskey, N. Daskalova, J. Huang, J. Hays, "Webgazer: scalable webcam eye tracking using user interactions," *AAAI Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pp. 3839-3845, 2016.

[10] *GazeRecorder*, 2021. <https://gazerecorder.com/>

[11] P. Viola, M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. l-l, 2001.

[12] V. Kazemi, J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1867-1874, 2014.

[13] F. Timm, E. Barth, "Accurate eye centre localisation by means of gradients," *Proceedings of the International Conference on Computer Vision Theory and Applications*, vol. 11, pp. 125–30, 2011.

[14] R. Jacob, "Eye tracking in advanced interface design," *Virtual Environments and Advanced Interface Design*, vol. 258, pp. 288, 1995.