UNIVERSITY OF COPENHAGEN

Haptic painting on a smartphone/tablet

Naomi Brandt Madolid Knudsen
&
Sara Selman

Department of Computer Science
Bachelor Thesis
10 Juni 2024

Supervisors:
Waseem Hassan, Postdoc

Abstract

1 page max

Contents

1. Introduction and Motivation

Learning to paint can be an expensive and frustrating experience if someone's painting doesn't turn out how they want. They might need to paint over the canvas completely to get a fresh start or use more money to buy a new canvas. Besides the canvas, they might also need to buy more paint and many different types of brushes depending on their project. Not everyone has the resources to buy all the new materials or the time to paint over a canvas several times so it will be a blank canvas again.
Furthermore, they might not want to use a canvas and paint to practice different paint styles and color combinations because this can seem wasteful material- and money-wise.
As our world becomes more and more digital, the art forms we engage in are also evolving in the same digital direction. People can practice on one of the many drawing apps that are out there, but these only relay the visual feedback of painting. They don't relay the full feedback that a painter receives when painting in real life: the auditory, the visual, and the haptic feedback.
Though digital art is becoming a more significant part of the artistic world, it still can't compete with the real experience. This is where haptic feedback can make a difference. Haptic feedback is a great tool for enhancing the user experience by providing the users with a more tactile and multi-modal experience, adding realism and depth to digital art.

Concept

Our solution to this problem is an Android application, which is designed to simulate the experience of painting on different canvases with various textures using haptic feedback. The app enables the user to feel the textures they are "painting" on, by incorporating tactile feedback, which in this case is vibrations coming from both the stylus and smartphone/tablet. The user also receives auditory feedback as well as visual feedback. This is delivered through respectively through the speakers or headphones and the visuals on the screen.

The app is hosted on a physical object, which is the user's smartphone or tablet. The app can be used everywhere, such as in a classroom or at home, and only requires a stylus for the user to interact with the touchscreen. Although, if you want haptic feedback you have to be connected to an external vibrator through the headphone jack.

The target users of the app are primarily users who enjoy and engage in artistic activities, such as drawing or painting. The app could also be used in education, as it could be a useful tool for teaching artists and students how to work with new mediums, without having the physical materials.

The user is in physical contact and interacts with the app by painting on their device with a stylus. The user chooses the canvas that they wish to draw on, and as they draw

on the screen they will experience haptic, auditory and visual feedback in real-time. The immediate feedback mimics the experience and sensation of painting on the given textured canvas.

The app utilizes an external haptic actuator, which is placed on the stylus, which will mimic the vibrations that would occur in the hand when painting on a textured surface. The actuator provides us with localized vibrations which corresponds to the user's touch input, and dynamically adjusts frequency of the vibrations to simulate the corresponding texture, whether it's rough or smooth.

Haptic feedback is appropriate and useful in this situation since it adds another layer of realism to the painting experience by providing immediate tactile confirmation of their actions and making the interaction more realistic and engaging for users who enjoy artistic activities or students who are learning how to handle different textures and materials, which enhances its educational value.

Ideatation & Experience

When ideating how to provide realistic feedback, we firstly needed to decide which type of actuator we should use, and how it should be placed. Realistic feedback involves a combination of visual, auditory, and haptic elements to create a satisfying and realistic user experience. Since most of the phones and tablets the app will be hosted on has a narrow-band vibration motor with weaker amplitude and limited API control of frequency and amplitude, we needed to use an external one. We used a type of vibrotactile actuator called Voice coil actuator - VCA to simulate the vibrations created when painting on textured surfaces.

We wanted the actuator to be placed on the stylus, since we wanted to create vibration transmissions to the user's hand.
We also implemented auditory feedback alongside the haptic feedback to enhance the user experience further by adding another layer of of realism.

2. Background and Related Work

In this section, we will discuss the purpose of haptic feedback, and the two most common types of models for creating haptic feedback will be explained and compared. We will then briefly discuss related work of digital painting applications.

2.1. Haptics

The point of haptic feedback is to provide stimulation and feedback through our sense of touch. There are two different areas of haptics, which is kinaesthetic perception and tactile perception (HapticLab, n.d.). Kinaesthetic perception allows us to assess our bodies in space and makes us able to move as intended even if we can't see. An example is if you are holding a cup with something and it is dark, your body still knows how to lift the cup to the intended target - your mouth (HapticLab, n.d.). It is the receptors in muscles, joints, tendons and (partly) skin that makes this perception possible.
Tactile perception allows us to feel texture, vibration, pressure and temperature. Looking at the cup example again, using tactile perception, we can feel the texture of the cup and the heat from a hot beverage in the cup. (HapticLab, n.d.). It is solely the receptors in the skin that makes this perception possible.

The two most common methods for implementing haptics are parametric and data-driven methods. These methods are different in their approach of creating haptic feedback.
In the parametric method, the haptic content is created using predefined equations, with tunable parameters. This allows more control over the haptic feedback, since these parameters can be manipulated manually to tune the properties of the created haptic feedback. For haptic feedback that simulates a texture, a simple approach would be to create a pure sine wave as a vibration to imitate the texture (Kern, Hatzfeld, & Abbasimoshaei, 2023).

The data-driven method is based on data recorded from the interaction of a sensorized tool with a real object, which makes the feedback more realistic. For haptic feedback that simulates a texture, the approach would be to record the acceleration signals resulting from interaction with the texture. These acceleration signals can then be replayed to create haptic feedback (Kern et al., 2023).

We used both models in our project since we adjusted our expectations to our project the further along we came to completion. Note that we shifted from parametric haptics to data-driven haptics as the project progressed.
Data-driven haptics was the preferred model since this would give the closest simulation of a real-life painting/drawing experience. While parametric haptics is an easier model to implement, but it doesn't yield the same close simulation as data-driven

haptics do.

2.1.1. Parametric haptics

One of the simplest models in parametric haptic texture simulation is to create a pure sine wave using the constant parameters: amplitude, frequency, and sample rate. These parameters should be changed depending on the background. Information for the parameters would be collected from other studies where these factors were studied using different materials. This approach makes it harder to make the texture simulations realistic and look at other factors that impact the haptic feedback such as the type of brush.

other studies = literature?

After our data collection (see section 3.3), we will be able to use our values for amplitude, frequency, and sample rate instead of the values from other studies. To do this, we would have to calculate the frequency and the amplitude from the raw data from the accelerometer for all the different canvases.

In section (3.5.3 PlaySound), one can read about the implementation of this model.

To understand this specific model, it is essential to understand the sine wave, its parameters and the sample rate.

2.1.1.1. Sine wave

A sine wave is a mathematical curve that that represents a smooth and repetitive oscillation, that oscillates around a central axis, graphically this would be depicted as two semi-circular curves that alternate above and below the central axis. The sine wave is characterized mathematically in terms of its amplitude and frequency, and is defined by $y = sinx$.

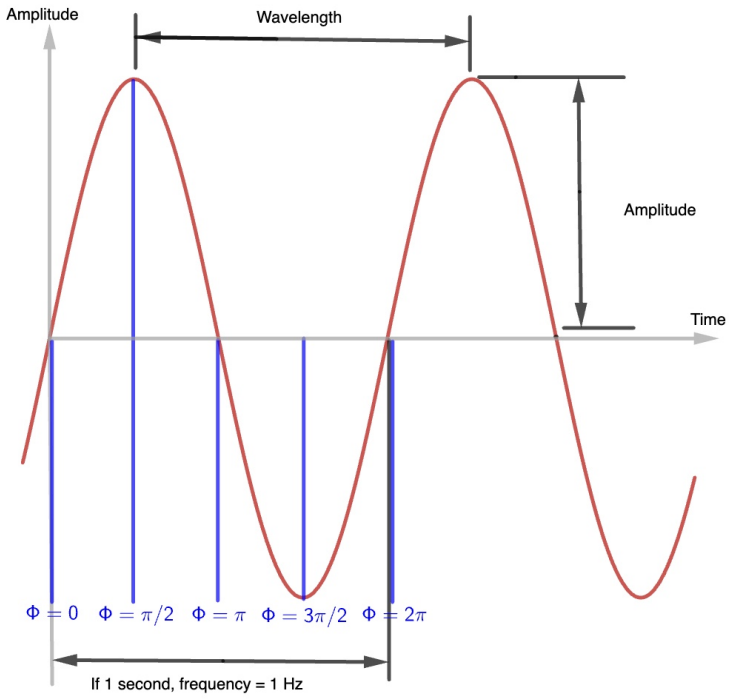


Figure 2.1.: Sine wave waveform showing the main attributes

The formula for a sine wave is mathematically described as a function of time (t):

$$y(t) = Asin(2\pi ft + \Phi) = Asin(\omega t + \Phi)$$

Where:

- **A - Amplitude :** This represents the peak value of the wave.
  It indicates how far the wave oscillates from its central axis. This means that the wave oscillates between +A and -A, since it determines the maximum and minimum values of the wave.

- **f - Frequency:** This is the number of complete oscillations (cycles) that occur per second. It is measured in Hertz (Hz).
  A higher frequency means more cycles per second and a shorter wavelength. In contrast, a lower frequency means fewer cycles per second and a longer wavelength.

- ω **- Angular Frequency:** - This is the rate of change of the phase of the sine wave, measured in radians per second. It is related to the frequency by the equation = 2πf.

- Φ **- Phase:** - This specifies (in radians) where in its cycle the oscillation is at t=0.

2.1.1.2. Sample rate

When we want to convert an analog audio signal, such as a sound wave, into a digital signal, it must be sampled at intervals. The sample rate is the number of samples taken per second, to represent and create the continuous signal digitally.
The higher the sample rate, the more "snapshots" you capture of the audio signal, and the more closely the final digital file will resemble the original. The audio sample rate is measured in kilohertz (kHz).
The standard sample rate for consumer audio is 44.1 kHz, which means that every second, 44,100 samples were taken.

The same process goes for creating vibration signals. With vibrations the sample rate determines how accurately the digital representation captures the nuances of the vibration, here the standard sample rate for slower vibration or movement is ∼1 kHz, meaning that 1,000 samples are taken per second.

2.1.2. Data-driven haptics

In our project, we used the data-driven method for creating both haptic and auditory models. The haptic models are created by collecting data from moving a sensorized tool (a brush with a accelerometer attached) at different speeds along different types of canvas types and recording the sounds of these movements. From the movement data, we can get hand and texture acceleration, and velocity that can be used to create models for the haptics based on the velocity of the user's hand across the screen.

The same process is used for data-driven audio feedback as the recorded sound files are used for auditory feedback at different speeds. Thus, this model makes it possible to adjust the sound depending on the hand velocity.

In section ?? ??, one can read about the implementation of this model.

2.1.3. Related work

There are numerous different digital painting applications on the market. Most digital painting applications are usually only focused on visual feedback to replicate the visual aspects and simulate the experience of drawing or painting in real life.
If we take a look at one of the most popular drawing apps "Procreate", we see that they focus on creating a realistic experience by providing the user with realistic brush simulations, and brush strokes that replicates real paint.

However, the the addition of haptic feedback to stimulate the tactile sensations created by painting on different surfaces remains unaddressed. By incorporating haptic feedback, we can provide the users with a more engaging and realistic experience, by replicating the tacticle sensations a person would feel when painting on different canvases.

Different studies, exploring haptics in both education and games, have shown that haptic feedback can enhance user experience. Research indicates that haptic and tactile feedback improves user engagement and satisfaction, by providing additional

sensory feedback that complements the visual and auditory feedback.

In a study where haptic feedback in digital games were explored the participants in the study reported that haptic feedback makes digital games more convincing and realistic (Kirginas, 2022). And in another study where haptic feedback in education the study reported that haptic feedback can make the learning more immersive and engaging (Fouad, Mansour, & Nabil, 2023) As our application can be used for both education and entertainment, both these studies are relevant.

3. Methods

In this section, we will provide an overview of our painting application. Here we will discuss the design, hardware and key components. We'll cover the app's interface and different functionalities, the hardware, and implementation process. In addition to that, we'll explain the haptic and audio feedback mechanisms, which includes data collection, segmentation and haptic models.

3.1. Design

The general idea of the project was to develop a mobile application interface that is intuitive and user-friendly. To better understand the steps that had to be taken to create the final app, we divided the tasks into two visual categories, the canvas and the pen/brush, and one haptic category the vibrations, and one auditory, the sound.

To visualize the general idea of the app, we created a detailed sketch of the interface. This sketch serves as a guideline, illustrating the interface of the app. It includes the key components such as the canvas area, tool/canvas selection menus and color palettes. The process of creating a sketch helped us to conceptualize what interface the user would be provided with, and give a clear direction for further development. The sketch of the interface can be seen below:

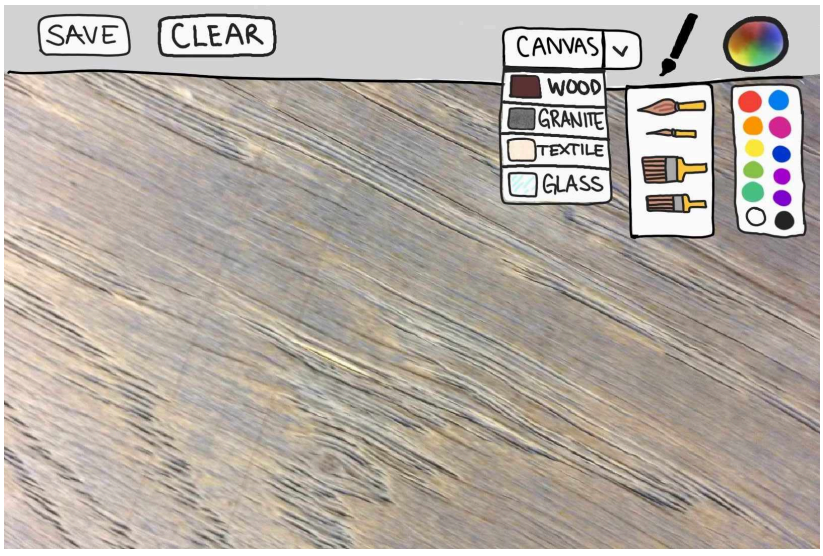


Figure 3.1.: Sketch of app interface

3.1.1. Canvas

The canvas has to have a few functionalities. We have chosen some functionalities that are necessary to create the multi-modal experience.
These are the must-have features that are currently implemented in our application:

- The user must be able to change the background so they can choose different canvases.
- The user must be able to draw/paint on this canvas, and the painting will not disappear.
- The user must be able to clear their painting so they have a fresh canvas.

We have 4 different backgrounds to choose from: Wood, Canvas, Silk and Glass.

Figure 3.2.: Wood

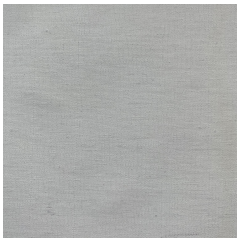


Figure 3.3.: Canvas

Figure 3.4.: Silk

Figure 3.5.: Glass

3.1.2. Pen/brush style

We had to think about some different style choices for the pen/brush to decide on the functionality. Should the user be able to both use a pen and a brush? What color should it be? What type of brush type should the user be able to use? Should the user be able to mix user-designed colors?
These are the must-have features that are currently implemented in our application:

- The user must be able to choose between different brushes.
- The user must be able to choose between different colors.

We have 3 different brushes to choose from: Thin, Round and Square.

Figure 3.6.: Thin

Figure 3.7.: Round

Figure 3.8.: Square

3.1.3. Vibration

We had to make a decision when choosing which type of actuator for vibration we wanted to use. We could use a device's built-in actuator for vibration, or we could use an external vibration motor as an actuator. Since the device we had access to didn't have a built-in vibrator, we had to use an external one. We used the HapCoil-One Voice Coil Actuator (3.2.2 External vibrator (Voice coil actuator - VCA)) as our external vibrating motor. We also had to decide when and how this external vibrator

should vibrate, and where it should be placed. Below we have written how we should be able to control the vibration.
These are the must-have features that are currently implemented in our application:

- The vibration should start when the user touches the canvas.
- The vibration should stop when the user stops touching the canvas.
- The vibration should be different from canvas to canvas.
- The vibration should be different from brush to brush.
- The vibration should be different depending on the velocity of the pen. There should be at least three different types: slow movement, medium movement, and fast movement.
- The vibration should be different when the user keeps the brush in one place while touching the screen and when the user moves the brush while touching the screen.

3.1.4. Sound

Just as for the other design choices, we had to look at what functionalities should be implemented to give the user the best auditory feedback.
These are the must-have features that are currently implemented in our application:

- The sound should start when the user touches the canvas, but only if the user is connected to a Bluetooth headset.
- The sound should stop when the user stops touching the canvas.
- The sound should be different from canvas to canvas.
- The sound should be different from brush to brush.
- The sound should be different depending on the velocity of the pen. There should be at least three different types: slow movement, medium movement, and fast movement.

3.2. Hardware

The hardware used for the haptic painting app was an android tablet along with a stylus. We attached a voice coil actuator to the stylus for providing vibrations. The actuator was attached to the stylus using a custom housing that we 3D printed. The details of these components is in the following subsections.

3.2.1. Device & Stylus

The device we used to host the application on is the Samsung Galaxy Tab S6 Lite. This tablet provides and is accompanied by a stylus, which we utilized for our project. By attaching the actuator to the stylus, we were able to use the provided stylus to mimic the sensation of a painting brush.

The screen specifications of the tablet are as follows:

- Resolution: 1920 x 1200 pixels
- Screen Size: 0.225 x 0.135 meters

The stylus specifications are as follows:

- Dimensions: 7,7 x 144 x 7,14 mm
- Weight: 7,03 g

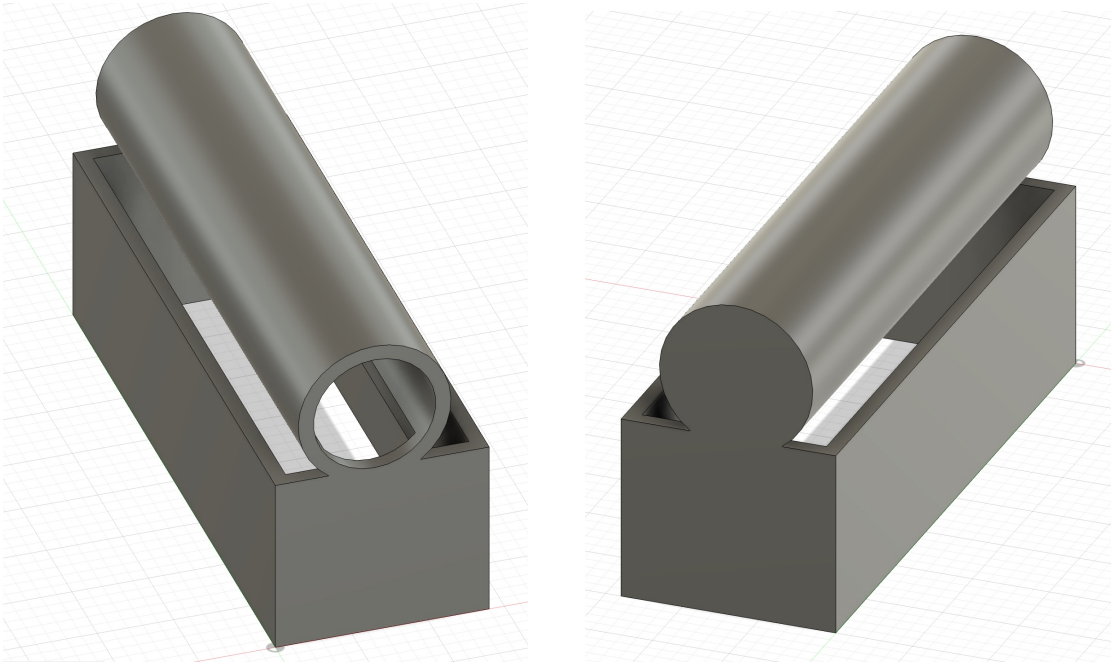
3.2.2. External vibrator (Voice coil actuator - VCA)

To mimic the subtle vibrations and tactile sensation that the user would feel when painting on a textures surface, we attached a HapCoil-One Voice Coil Actuator (VCA) on the stylus.
Voice coil actuators (VCAs) works by utilizing the interaction between a generated magnetic field and a coil of wire. When an electrical current is applied to the coil, it generates a magnetic field that interacts with the permanent magnetic field, causing the coil to move.

3.2.3. 3D printed actuator holder

When ideating how the actuator should be placed on the stylus, we shortly considered creating a silicone holder for the actuator that the user would easily be able to slide on, but since silicone rubber had great compression reliance and is a vibration isolator, the silicone rubber would reduce the vibrations, so we came to the conclusion that the best material for the actuator holder would be a plastic. We wanted to create a plastic holder for the stylus that the user would be able the slide on the stylus.

We 3D sketched how we imagined the design should look, for the external vibrator to stay in touch with the stylus, but not fall off in case of movement. You can see this in Figure 3.9. The sketch was made in AutoDesk Fusion 360.



(a) The bottom part of the pen holder

(b) The top part of the pen holder

Figure 3.9.: First 3D sketch of the design for the holder

We obtained the measurements for the holder by measuring the external vibrator (see 3.2.2) and the stylus (see 3.2.1) and adjusting the original sketch's measurements as we printed several prototypes until the size was right and the holder fit perfectly.
We tested the effectiveness of the holder by ensuring that it kept the external vibrator attached to the stylus without falling off.
We also had to make sure that the external vibrator fit tightly in the holder, so that were no unnecessary vibrations (noise) created due to the mold colliding with the stylus.

The dimensions for the 3D printed holder are as follows:

- Length = 40.0 mm
- Width = 14.2 mm
- Height = 21 mm

3.2.4. Stylus and external vibrator set-up

In Figure 3.10 you can see the complete set-up of the holder connecting the stylus to the external vibrator. This set-up ensures that the user of the pen is disturbed the least by the wires.

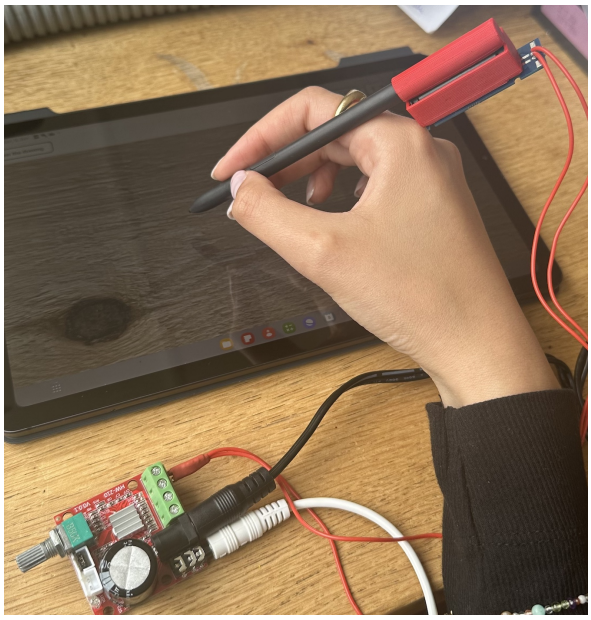


Figure 3.10.: The whole set-up of 3D printed holder connecting stylus and external vibrator

3.3. Data Collection

In this section, we explain the processes and methods for data collection, processing, and modeling that we used in our project.

A setup of how we collected data can be seen in Figure 3.11. In the specific setup in Figure 3.11, we collected data about the wood texture with a thin paintbrush.

Figure 3.11.: Setup for Data Collection

We used code in both the Arduino IDE and MATLAB for data collection.
Firstly, we had to use some code in the Arduino IDE app, which made communication with the MPU6050 accelerometer and gyroscope sensor possible.
The details for the code can be seen in Appendix A.0.8. In the code, we specified the baud rate as 115200, which means that the serial data transmission is 115200 bits per second. We also had to specify that we were using the board described in Section 3.3.1 and which port was connected to the board.

Secondly, we used parallel processing to run two different files in Matlab at the same time: one for collecting the XYZ data from the MPU6050 accelerometer in a `.csv` file (called `arduinoDataRead.m` - see Appendix A.0.1) and one for saving the sound recording in a `.wav` file (called `sound.m` - see Appendix A.0.2).
In `arduinoDataRead.m`, we specified the baud rate and the port to be the same as in the Arduino code. We also specified the number of seconds that data from the accelerometer should be recorded, which was 4 seconds.
In `sound.m`, we specified the sample rate of the sound recording to be 44100 Hz, and the bits per sample to be 16. We also specified the recording length to 4 seconds as we did in `arduinoDataRead.m`.

We then created a file called `parallel.m` which was used to run the arduinoDataRead and sound function concurrently. It is also in this file that we specify the name of the `.csv` and `.wav` files that are created in respectively `arduinoDataRead.m` and `sound.m`.

The accelerometer sent the XYZ data to the arduino, but we only used the Z-axis data because it was the direction we needed for our analysis, since we are interested in the horizontal motion of the paintbrush as it moved across the different textures.

3.3.1. Arduino board

We first installed the Arduino IDE software on our computer so we could communicate with the Adafruit Metro M0 Express microcontroller board that we were using.
We then connected an accelerometer `MPU6050` to the Arduino board to record the velocity of the strokes made by different brushes on different textures and speeds. We connected the pins from the accelerometer to the board respectively as follows: the GND to GND, the VCC to 5V, and SCL and SDA to the digital pins SCL and SDA. To measure the acceleration we used the code which can be seen in Appendix A.0.8.

3.3.2. Accelerometer

The Accelerometer we used when recording the velocity, is a MPU-6050 Accelerometer. The MPU-6050 contains a 3-axis gyroscope and a 3-axis accelerometer.
The accelerometer features a user-programmable gyro full-scale range of ± 250, ± 500, ± 1000, and ± 2000 °/sec (dps), and a user-programmable accelerometer full-scale range of ± 2g, ± 4g, ± 8g, and ± 16g, which inhances the precision tracking of both fast and slow motions (TDK, 2016), in our project we only used the accelerometer part of the MPU.

3.3.3. Results

In our data collection process, we collected acceleration data and audio recordings across multiple conditions: The data was collected for four different canvas textures and three different brushes. We recorded the acceleration data and audio simultaneously for a duration of 4 seconds, and did this for all three different velocities: fast, medium, and slow, where:

- Fast Speed: 24 cm in 4 seconds (0.06 m/s)

- Medium Speed: 12 cm in 4 seconds (0.03 m/s)
- Slow Speed: 6 cm in 4 seconds (0.015 m/s)

The processing of this data will be discussed in the next sections.

3.4. Modeling

After recording our data, our next step was to find out if our data fell within the desired intervals. These intervals were determined based on calculations of the expected velocity, which we calculate in 3.4.2 Data Segmentation.
This process ensures that our recorded data aligns with our expected ranges, which verifies the accuracy and consistency of our data collection. When looking though our files, we founds some recordings that deviated outside our expected intervals. This could be because of different external factors as misalignment of the accelerometer during the recording. We then needed to re-record these segments to ensure the accuracy and consistency of our dataset.

It is important to fine-tune the feedback to ensure that the haptic experience aligns with the real texture. The process of fine-tuning involves adjusting different parameters to tune the haptic feedback delivered to the user, to ensure that the user receives the most realistic experience.

3.4.1. Pre-processing

After having recorded the sound into `.wav` files and acceleration data into `.csv` files, we used `MATLAB` to convert the raw values collected to acceleration data in gram force (gs) and then converted this to meters per second squared. We were then able to separate this acceleration data into hand acceleration, texture acceleration, and the velocity computed from the hand acceleration. In Figure 3.12, one can see an example of this data visualized as graphs.

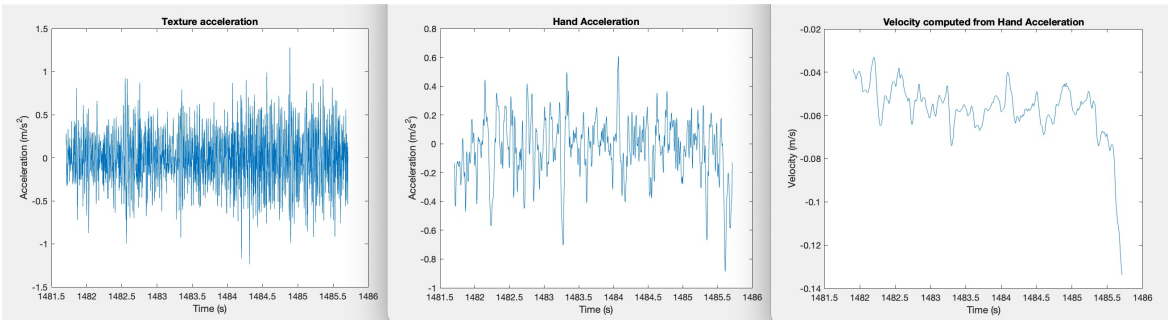


Figure 3.12.: Graphs of (a) texture acceleration, (b) hand acceleration, and (c) velocity computed from the raw data of the round brush on canvas

To convert the data, we used the code which can be seen in Appendix A.0.4. Here we used a high pass filter with a cut-off frequency of 20 Hz to isolate the texture frequency and get the texture acceleration. We used a band pass filter with a high cut-off frequency of 20 Hz and a low cut-off frequency of 1 Hz to isolate the frequency of hand motion and get the hand acceleration. We then used a numerical method to

integrate the hand acceleration data so we could calculate velocity.
We needed to use the texture acceleration data to create the haptic `.wav` files that should play in the app.

3.4.2. Data Segmentation

We needed specific data points that corresponded to the data points where the hand velocity was close to the expected velocity (values within 0.0005 of the expected velocity).

The expected velocity was calculated by using the formula for calculating velocity:

$$\bar{v} = \frac{\Delta x}{\Delta t}$$
$$\bar{v} = \text{ average velocity}$$
$$\Delta x = \text{ displacement}$$
$$\Delta t = \text{ change in time}$$

By inserting our own values in the formula, we were able to calculate the expected velocities:

Fast speed (4-second recording of a length of 24 cm)

$$v_f = \frac{0.24m}{4s} = 0.06m/s$$

Medium speed (4-second recording of a length of 12 cm)

$$v_m = \frac{0.12m}{4s} = 0.03m/s$$

Slow speed (4-second recording of a length of 6 cm)

$$v_s = \frac{0.06m}{4s} = 0.015m/s$$

Now that we knew the expected velocities, we used some code to read the velocity data and find data points that fit these criteria. The code for reading the velocity data of slow speed can be seen in Appendix A.0.5 which is similar to the code for medium and fast speed which can be seen in respectively Appendix A.0.6 and Appendix A.0.7.
No matter the code, we first found out at which indexes the velocity was within 0.0005 of the expected velocity. For the data points to be useful, we required that the data points were next to each other and that there were at least two data points.
We saved the indexes of the data points in a one-dimensional array. We then found the texture acceleration data at these indexes and repeated this array until we had 80 samples. From these 80 samples, we then created a `.wav` file that should be played as the haptics (vibration).
For each of these `.wav` files we had to manually adjust the gain so it felt as close to the real feeling of a certain brush on a certain canvas as possible, which will be explained further in Section 3.4.

3.4.3. Empirical segmentation

After the `.wav` was recorded during data collection (see Section 3.3), we used the audio editing app Audacity to apply a high-pass filter to the sound and to cut the them into a shorter files.
At first, we imported the fast `.wav` files into Audacity, and then we applied a high-pass filter at 100 Hz with a roll off of 6 db, which reduced all frequencies below 100 Hz. This minimizes unnecessary low frequencies, such as background noise that could affect the auditory feedback.
We then empirically selected the best 100 ms sound signal in the file. For the medium speed, we empirically selected the best 50 ms sound signal from the chosen fast sound signal, and for the slow speed, we empirically selected the best 25 ms sound signal from the chosen fast sound signal. We did this to create new variations for medium and slow feedback so they could be slowed down and be the same length as the fast sound signal. We explain why this is important to do in (3.4.4 Fine tuning).

3.4.4. Fine tuning

As mentioned, we wanted to manually adjust the gain of the data-driven signals for each of our `.wav` files, to make sure that the haptic feedback feels as realistic as possible.
For the haptic feedback, we tuned the feedback using Audacity, which is an audio editing app. Earlier in the pre-processing (see Section 3.4.1), we applied a high pass filter to our raw data from our recorded csv. file, this gave us data for the texture acceleration, which we then analyzed to find the places where the data lied in our wished interval, here we needed a minimum of two values that lied next to each other. We then adjusted the amplitude of the texture acceleration after if had been through our data segmentation process (3.4.2 Data Segmentation). We did this to ensure that the haptic feedback accurately represented the real texture, to create the most realistic haptic feedback.

omskriv?

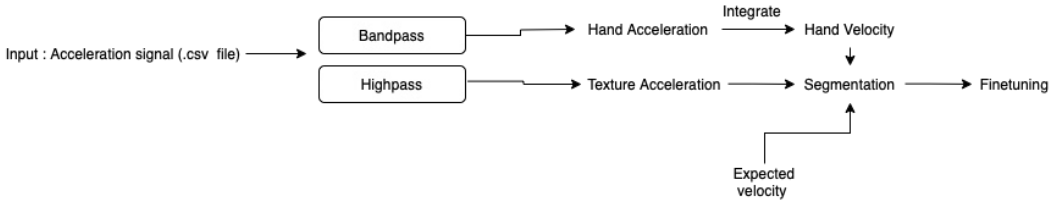
For the auditory feedback `.wav` files, we wanted to create a seamless loop, so the sound mimics the sound of moving a brush continuously on a surface as close as possible. We did this by copying the 100 ms sound signal chosen in 3.4.3 Empirical segmentation and reversing it.
After doing this for the fast velocity feedback, we wanted to fine-tune the feedback for medium and slow, here we used the 50 ms and 25 ms cuts of the fast feedback. We slowed these down proportionately to respectively the fast/medium velocity ratio and the fast/slow velocity ratio, i.e. to create the medium sound file we reduced the speed by 50%, and to create the slow sound file we reduced the speed by 75%.
For all three speeds, we adjusted the volume of the sound, to make sure that the sound wasn't too high or low. We did this to ensure that the different speeds maintained consistency to create the most realistic auditory feedback.

During the fine-tuning process, we found out that using the original medium and slow recordings resulted in significantly different sound for each speed. This would mean that the auditory feedback during transitions between speeds would be very disrupted and the continuity and realism would be compromised.
To solve this issue, we decided to use the fast recording for both the medium and slow

speeds. By slowing down the fast recording instead of using their respective recordings, we maintained a more consistent audio across all speeds, which ensured smooth transitions.

3.4.5. Haptic rendering

(imput) user touch and user velocity look at chosen bush and canvas
rendering pipeline , loop uo the models and then play
An overall algorithm on how the haptic rendering is done, can be read about in 3.5.8. The more technical information about the haptic rendering is explained in 3.5.3 PlayHaptics, 3.5.4 DrawingView, and 3.5.5 ControlFeedback.



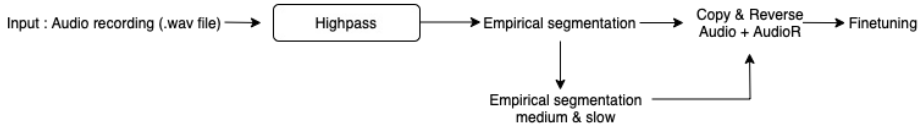


Figure 3.13.: Modelling Pipelines

NOTE: The pipelines should be in finetuning?

3.5. Implementation

To follow the `"Single Responsibility Principle"` SOLID design principle for object-oriented software development stating that each class is only responsible for one part of the software's functionality and must only solve one problem (Roca, n.d.), we created 9 classes to implement the desired functionality: `CustomAdapter`, `CustomColorAdapter`, `CustomBrushAdapter`, `PlaySound`, `PlayMusic`, `PlayHaptics`, `DrawingView`, `MainActivity` and `ControlFeedback`.

3.5.1. Layout XML files

Not finished yet

3.5.1.1. Toolbar

Canvas spinner
Brush spinner
Color spinner

Toolbar buttons

3.5.1.2. Main activity layout

3.5.2. Dropdown menus in Toolbar

The toolbar of our app interface features three dropdown menus, each serving its own purpose.

CustomAdapter

The `CustomAdapter` class is used to define how the canvas dropdown menu/canvas spinner is inflated using the `custom_canvas_items` XML layout. The class inherits from the `BaseAdapter` class and besides its constructor it has four methods which overrides the methods of the `BaseAdapter` class: `getCount`, `getItem`, `getItemId` and `getView`.

- The method `getCount` returns the number of canvas items that are in the canvas picture array. This method is not used, but is required to override because `CustomAdapter` inherits from `BaseAdapter`.
- The method `getItem` returns the canvas picture at the position, i, in array. This method is not used, but is required to override because `CustomAdapter` inherits from `BaseAdapter`.
- The method `getItemId` returns the position of the item within the adapter's data set whose row id we want. This method is not used, but is required to override because `CustomAdapter` inherits from `BaseAdapter`.
- The method `getView` gets a View that displays and sets the canvas picture and canvas name in the `custom_spinner_items` layout. This presents the canvases as options of a descriptive name with a corresponding picture.

This class is used in `MainActivity`'s method `onCreateOptionsMenu`.

CustomColorAdapter

The `CustomColorAdapter` class is used to define how the color dropdown menu/color spinner is inflated using the `custom_color_items` XML layout. The class inherits from the `BaseAdapter` class and besides its constructor it has the same four methods as the `CustomAdapter` class. The methods `getItem` and `getItemId` are exactly the same.

- The method `getCount` returns the number of color items that are in the color array. This method is not used, but is required to override because `CustomColorAdapter` inherits from `BaseAdapter`.
- The method `getView` gets a View that displays and sets the background color of the spinner items in the `custom_color_items` layout. This displays the color options as squares in the spinner.

This class is also used in `MainActivity`'s method `onCreateOptionsMenu`.

CustomBrushAdapter

The `CustomBrushAdapter` class is responsible for making a View for each of the brush items in the brush dropdown menu/brush spinner. These are inflated using the `custom_brush_items` XML layout. The class inherits from the `BaseAdapter` class and besides its constructor it has the same four methods as the `CustomAdapter` and `CustomColorAdapter` classes. The methods `getItem` and `getItemId` are exactly the same.

- The method `getCount` returns the number of brush items that are in the brush picture array. This method is not used, but is required to override because `CustomBrushAdapter` inherits from `BaseAdapter`.
- The method `getView` gets a View that displays and sets the brush pictures in the `custom_brush_items` layout. This displays the brush options as pictures in the spinner which provides the user with options for selecting different brush sizes and shapes to paint with.

This class is used in `MainActivity`'s method `onCreateOptionsMenu` as well.

3.5.3. Sound and vibration

PlaySound

The class `PlaySound` was the first attempt at making the external vibrator vibrate through sound. We accomplished this by creating a pure sine wave that would be played through the external vibrator. The class has four methods that are essential to play the sine wave: `initTrack`, `startPlaying`, `playback`, and `stopPlaying`.

- The method `initTrack` creates an `AudioTrack` instance and initializes it with a specified sample rate and buffer length.
- The method `startPlaying` starts playing the `AudioTrack` and also sets the boolean `isPlaying` to true, which is used in other functions.
- The method `playback` generates a simple sine wave from a specified amplitude, frequency, sample rate, and buffer length. Note that the sample rate and buffer length should be the same as in `initTrack`.
- The method `stopPlaying` stops and releases the `AudioTrack` if it is playing and sets `isPlaying` to false.

To use this class we had to manually choose the sample rate, amplitude, and frequency of the sound for each canvas. The buffer length was determined from a function `AudioTrack.getMinBufferSize` which uses the specified sample rate. This was done through a function called `playParametricHaptics` (see Appendix B.0.1) in the class `ControlFeedback`, which uses the functions described above to play a pure sine wave based on the specified values if no other sound is currently playing. A new thread is created to play the audio each time the user touches the screen anew.
We used `DrawingView`'s function `onTouchEvent` to run the `playParametricHaptics` function when the user touches the screen, and to run the `stopPlaying` function when the user stops touching the screen.

This class is not used in the final product, as we use the `PlayMusic` class for playing the auditory feedback and the `PlayHaptics` class for playing the haptic feedback.

PlayMusic

The `PlayMusic` class is used to play the auditory feedback and direct this to a headset if one is connected through a USB-C port otherwise it will be directed to the tablet's internal speakers. This will only happen if the tablet isn't connected to an aux line.
The class has seven methods: `init`, `startPlaying`, `setVolume`, `pausePlaying`, `stopPlaying` and `findAudioDevice`.

- The method `init` initializes a SoundPool instance with different parameters. It also loads the fast, medium an slow sound from the raw resource files and initializes an `AudioManager`.
- The method `startPlaying` uses the `AudioManager` to direct the sound to either the internal speakers or the connected Bluetooth headphones. It then starts and sets the drifferent `SoundPool` streams to be looping if they are done looping. It also sets the `PlayMusic` attribute `isPlaying` to true.
- The method `setVolume` sets the left and right volume of the the three streams (fast, medium and slow sound) based on a given velocity. This is used to mute the streams we don't want to hear and unmute when we want to hear the streams again.
- The method `pausePlaying` pauses and unloads the `SoundPool` streams and sets the attribute `isPLaying` to false.
- The method `stopPlaying` stops the `SoundPool` streams and sets the attribute `isPLaying` to false. It also releases the `SoundPool` resources so the `SoundPool` has to be initialized again.
- The method `findAudioDevice` is used to find out if it is possible to connect to a specified device and, in the case that it is, return the specified device. This method is used in `startPlaying`.

This class is used in the `DrawingView` class and the `ControlFeedback` class to initialize, play and pause the players.

PlayHaptics

The `PlayHaptics` class is used to play the sounds for the haptic feedback and direct this to the aux line if one is connected otherwise nothing will be played.
The class has seven methods: `init`, `startPlaying`, `setVolume`, `pausePlaying`, `stopPlaying` and `findAudioDevice`.

- The method `init` initializes a MediaPlayer instance with different parameters. It also sets the data resource of the media to be played and initializes an `AudioManager`.

- The method `startPlaying` uses the `AudioManager` to direct the sound to either the internal speakers or the connected Bluetooth headphones. It then starts and sets the `MediaPlayer` looping if it is done preparing. It also sets the `PlayHaptics` attribute `isPlaying` to true.
- The method `setVolume` sets the left and right volume of the `MediaPlayer`. This is used to mute the players we don't want to hear and unmute when we want to hear the players again.
- The method `pausePlaying` pauses the MediaPlayer instance and sets the attribute `isPlaying` to false.
- The method `stopPlaying` stops the `MediaPlayer` instance and sets the attribute `isPLaying` to false. It also releases the `MediaPlayer` resources so the `MediaPlayer` has to be initialized again.
- The method `findAudioDevice` is used to find out if it is possible to connect to a specified device and, in the case that it is, return the specified device. This method is used in `startPlaying`.

This class is used in the `DrawingView` class and the `ControlFeedback` class to initialize, play and pause the players.

3.5.4. DrawingView

The `DrawingView` class creates a canvas that can be drawn on and handles the style of the paint. This class is the essential class for showing visual feedback. It inherits from the `View` class and therefore has three constructors. Besides those, the class has five methods: `setupDrawing`, `changeColor`, `changeBrush`, `clearView` and `getVelocity`. It also has three other methods, which override the `View` class methods: `onSizeChanged`, `onDraw` and `onTouchEvent`.

- The method `setupDrawing` initializes a `Path`, a `Paint` and a `Canvas`. It also sets the properties for the `Paint`, such as the brush color, type, and thickness. This method is called in the constructors of the `DrawingView` class.
- The method `changeCanvas` changes canvas background and clears anything that was drawn on the canvas before. This is done by changing the value of the `chosenBackground` variable. It then calls upon the `clearView` function.
- The method `changeColor` changes the color being painted with. This is done by adding a new `Path` to the `Path` list, a brush type to the list of brush types, and a paint color to the list of colors. It then calls `invalidate()`, which calls upon the `onDraw` function so the new color can be used to draw with.
- The method `changeBrush` changes the brush - a brush is defined as a width and a `Paint.Cap` - being painted with. This is done by adding a new `Path` to the `Path` list, a brush type to the list of brush types, and a paint color to the list of colors. It then calls `invalidate()`, which calls upon the `onDraw` function so the new brush can be used to draw with.

- The method `onDraw` is used to draw the canvas background and the strokes. It draws the strokes by iterating through the colors, brushes, and `Path`s lists.
- The method `onSizeChanged` dictates what happens if the size of the `View` changes, such as if the screen is rotated.
- The method `clearView` clears the canvas of any paint by resetting all the `Path`s in the `Path`s list. It then calls invalidate(), which calls upon the onDraw function.
- The method `getVelocity` computes the velocity of the user's movements on the screen in m/s. In Java, we can only calculate the x-velocity and y-velocity in pixels per second so to get it in meters per second, we first calculate the total velocity using Pythagoras theorem: $v_{total} = \sqrt{(v_x^2 + v_y^2)}$. We then used the pixel information of the tablet's screen and the screen's width and length to calculate the total velocity from pixels per second to meters per second. The length is 1920 pixels and 0.225 meters. Which gives us this equation: $\sqrt{(v_x^2 + v_y^2)} * \frac{0.225}{1920}$.
- The method `onTouchEvent` dictates what should happen with the different touch screen motion events. If the screen is touched, it starts playing the sound for the haptic and tactile feedback, and when the screen is no longer touched, it stops the feedback. It also adjusts the haptic and tactile feedback being played based on the velocity.

This class is used in the `MainActivity` and the `activity_main` XML layout file.

3.5.5. ControlFeedback

The class `ControlFeedback` is used to control the auditory and haptic feedback. It has five methods: `initializeFeedback`, `velocityFeedback`, and `playParametricHaptics`.

- The method `initializeMusic` initializes and starts the `SoundPool` in the `PlayMusic` instance.
- The method `initializeFeedback` initializes and starts the `MediaPlayer` in the instances of the `PlayHaptics` classes. This also mutes the fast and medium haptics, while keeping the slow haptics unmuted.
- The method `velocityFeedback` changes the volume of the `MediaPlayer` in the instances of the `PlayHaptics` classes depending on the velocity of the user's movements.
- The method `playParametricHaptics` plays the frequency, amplitude, and sample rate that matches the canvas. This is only used with the `PlaySound` class, and we don't use this method because it is used for the parametric method (see Section 2.1.1).

This class is used in the `DrawingView` class.

3.5.6. MainActivity

The class `MainActivity` initiates the code and launches the app's first screen: the DrawingView with the toolbar. It inherits from the `AppCompatActivity` class and has two methods, which override the `AppCompatActivity` class methods: `onCreate` and `onCreateOptionsMenu`. It also has three helper functions which are used in the `onCreateOptionsMenu` method: `canvasSpinnerSelection`, `colorSpinnerSelection`, and `brushSpinnerSelection`.

- The method `onCreate` initializes the app by inflating the toolbar's left side and setting the view. It also requests Bluetooth access, if the user hasn't already given permission.
- The method `onCreateOptionsMenu` inflates the rest of the toolbar and creates the canvas, brush, and color spinner. It calls upon the `canvasSpinnerSelection`, `colorSpinnerSelection`, and `brushSpinnerSelection`.
- The method `canvasSpinnerSelection` is a helper function to the `onCreateOptionsMenu` method so it is more readable. It dictates what should be done when a canvas is selected in the spinner.
- The method `colorSpinnerSelection` is a helper function to the `onCreateOptionsMenu` method so it is more readable. It dictates what should be done when a color is selected in the spinner.
- The method `brushSpinnerSelection` is a helper function to the `onCreateOptionsMenu` method so it is more readable. It dictates what should be done when a brush is selected in the spinner.

3.5.7. UML diagram

Using a UML diagram, we show a better overview of the relationship between the different classes. Note that the "..." indicates that there are more attributes than written. We have only written the most important attributes for understanding the relationships and the classes. We haven't written the constructors of the classes.

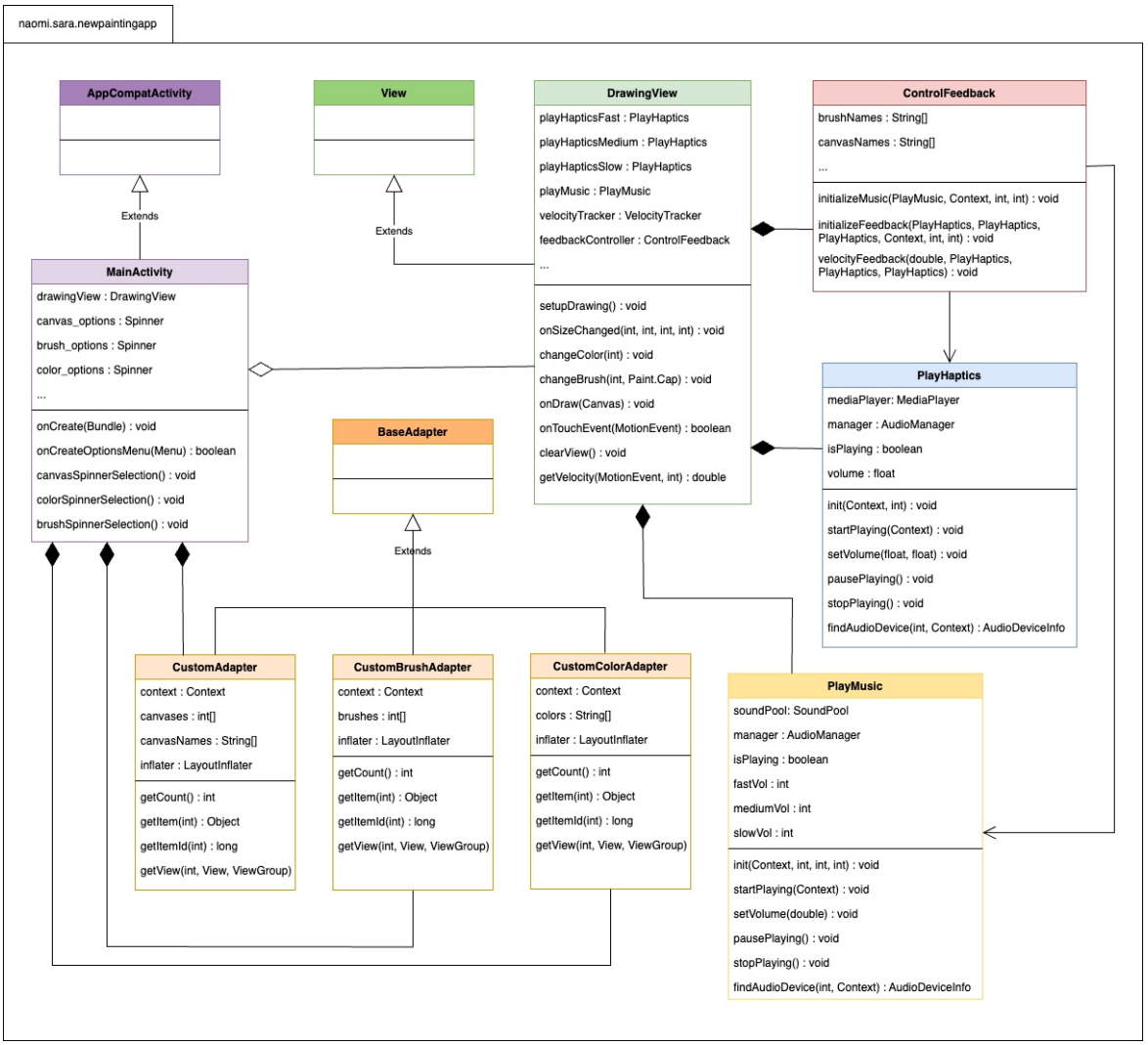


Figure 3.14.: UML diagram of the classes in our app

3.5.8. Pseudocode

We have written three algorithms in pseudocode to provide an overview of how the core functionalities and implementation of our application work, one for visualization, one for auditory feedback and one haptic feedback.

Algorithm 1 Visualization

Input: user's touch
Step 1: Toolbar and background
Create a wood background and toolbar with a clear button and drop-down menu for canvas, brush, and color.
Step 2: Drop-down menues
Register that the user is touching a drop-down menu
Show the options of the chosen drop-down menu
if new option is chosen in drop-down menu **then**
    The drop-down menu closes
    Change background, brush, or color depending on the chosen drop-down menu.
else
    The drop-down menu closes and nothing is changed
end if
Step 3: The clear button
if clear button is clicked **then**
    All the painting on the background is removed
end if
Finish

Algorithm 2 Auditory feedback

Input: User's touch
Step 1: Playing auditory feedback
Register that the user is touching the screen
Determine what the playback should be based on the chosen brush{B} and canvas{C} type : sound{B C}
if tablet is connected to nothing or USB-C port to headphones **then**
    Play auditory feedback
    Step 2: Playing auditory based on movement
    Keep checking the velocity of the user's movement
    Change auditory feedback playback based on the velocity, canvas, and brush
    Step 3: Stop feedback
    When the user stops touching the screen, pause all feedback
else
    Auditory feedback will not play
end if
Finish

Algorithm 3 Haptic feedback

1: **Input:** User's touch
2: **Step 1:** Playing haptic feedback
3: Register that the user is touching the screen
4: Determine what the playback should be based on the chosen brush{B} and canvas{C} type : haptic{B C}
5: **if** tablet is connected to aux line **then**
6: Haptics will begin playing to the external vibrator if aux line is connected.
7: **Step 2:** Playing haptic feedback based on movement
8: Keep checking the velocity of the user's movement
9: Change haptic feedback playback based on the velocity, canvas, and brush
10: **Step 3:** Stop feedback
11: When the user stops touching the screen, pause all feedback
12: **else**
13: Haptic feedback will not play
14: **end if**
15: **Finish**

Together, these algorithm makes up the workings of the painting app.

3.6. Test

We made instrumented tests for the classes: `ControlFeedback`, `DrawingView`, `PlayHaptics` and `PlayMusic`. Instrumented tests are run on a hardware device or emulator unlike unit tests, which are run on a machine's local Java Virtual Machine (JVM) (AndroidDevelopers, n.d.).
Unit tests were irrelevant for testing of the app since a context was needed to access resources in the app which most classes' methods use. Contexts can only be accessed with instrumented tests.
We couldn't test a lot of `PlayHaptics`' functionality since an emulator doesn't have an emulated aux line, and we were using an emulator and not a hardware device to run these tests.

3.6.1. DrawingView tests

We needed to add some methods to the `DrawingView` class, in order for us to access some of the private attributes that we needed in order to test the methods. We added `getPaint`, `getPath` and `getCanvas` in order to get respectively the `Paint`, `Path` and the chosen background.
We tested the methods `changeCanvas`, `changeColor`, `changeBrush` and `onDraw`:

- We tested `changeCanvas` by asserting that if you use the function to change the canvas to the third canvas, then the chosen background in the `DrawingView` class should also be specified to the third canvas.
- We tested `changeColor` by checking if the `Paint` color is changed to red, after we called `changeColor(red)`.

- We tested `changeBrush` by calling on the method with the values width 10 and `Cap.BUTT` and asserting that the `Paint`s stroke width and stroke cap is now the same as these.
- We tested `onDraw` by drawing on known pixels, and asserting that these pixels are black as this is the color that was painted with.

3.6.2. PlayMusic tests

We also needed to add some methods to the `PlayMusic` class. We added `isPlaying`, `getVolume` and `getSoundPool` in order to get respectively the `isPlaying` attribute and the volume of a fast, medium or slow audio, and the SoundPool instance.
We tested the methods `startPlaying`, `setVolume`, `pausePlaying` and `stopPlaying`:

- We tested `startPlaying` by asserting that the `isPlaying` variable was changed to true (it is initially false) after we called on the method.
- We tested `setVolume` by asserting that the variable for checking the volume of the fast audio was 1 (unmuted) after calling the method with a velocity of 0.06.
- We tested `pausePlaying` by first calling `startPlaying`. We then called `pausePlaying` and asserted that the `isPlaying` variable was changed to false.
- We tested `stopPlaying` by first calling `startPlaying` and saving the `SoundPool` instance. We the then called `stopPlaying` and asserted that the saved `SoundPool` instance wasn't the same as the current `SoundPool` instance.

3.6.3. PlayHaptics tests

Since `PlayHaptics` will only play sound if an aux line is connected, we can only test the initialization (the method `init`) and volume (the method `setVolume`). Again, we added some new methods in order to do this: `getVolume` and `getMediaPlayer`.

- We tested `init` by asserting that the `MediaPlayer` in the class wasn't null.
- We tested `setVolume` by asserting if the volume attribute was 1 (unmuted), after we called `setVolume(1,1)`.

3.6.4. ControlFeedback tests

As with `PlayHaptics` we cannot test all the methods in this class, because it initializes the haptics which need a connected aux line to play.

- We tested `initializeMusic` by asserting that the `PlayMusic` instance's `isPlaying` variable is true.
- We tested `velocityFeedback` by initializing three `PlayHaptics` instances and setting the velocity to 0.06. We then check that the fast haptics volume is umuted and the rets is muted.

3.7. User's Guide

The app uses a single-canvas interface. When the user opens the app they are presented with the default canvas, which is wood. The user is also presented with a toolbar placed at the top which provides the user access to the different canvas, brushes and colors, and a button for clearing the painting.

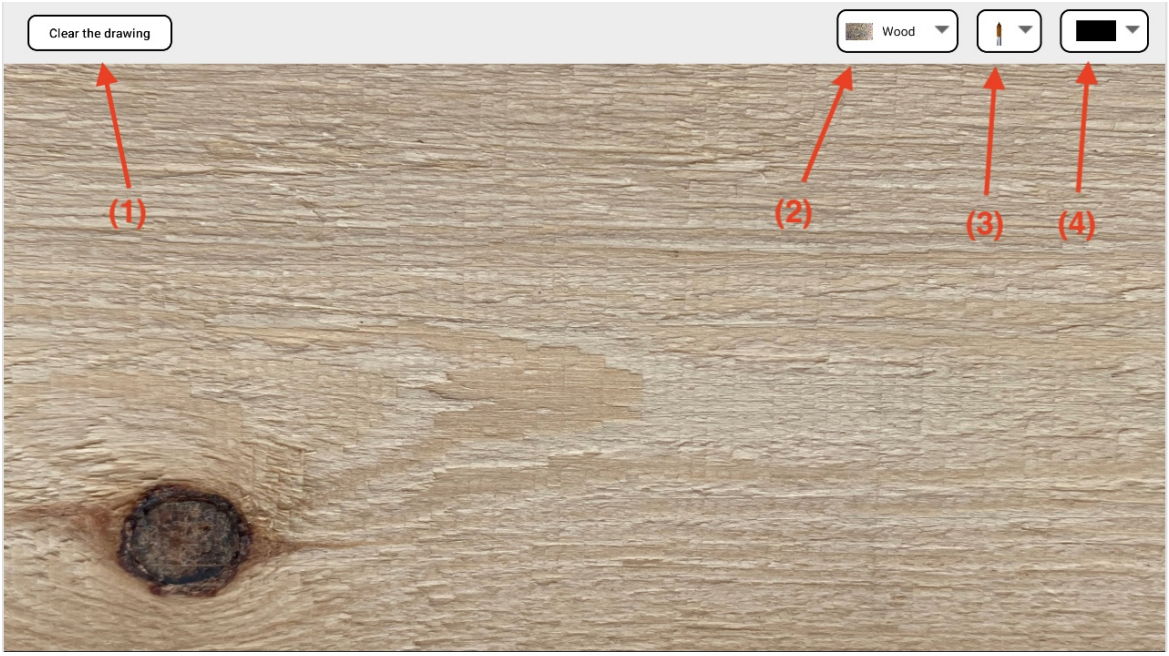


Figure 3.15.: Main Interface:
1) Clears the canvas, this button allows you to reset your canvas and start fresh.
2) Drop down menu for the canvases, this menu lets you select from a variety of canvas types
3) Drop down menu for the brushes, this menu offers different brush styles
4) Drop down menu for the colors, choose from a palette of colors to apply to your brush.

Upon pressing the different drop down menus, the user will be presented with the different choices.

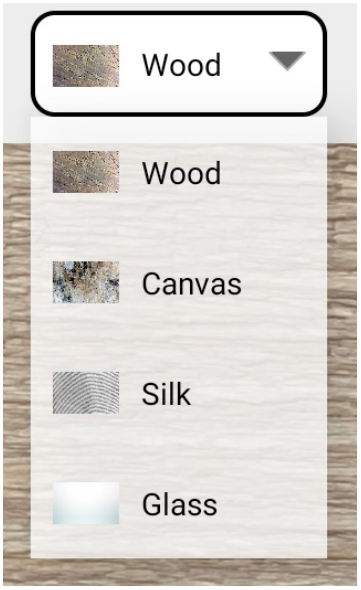


Figure 3.16.: Canvas options
1) Click the canvas drop down menu
2) Select the desired canvas type
3) The canvas will change to the selected option

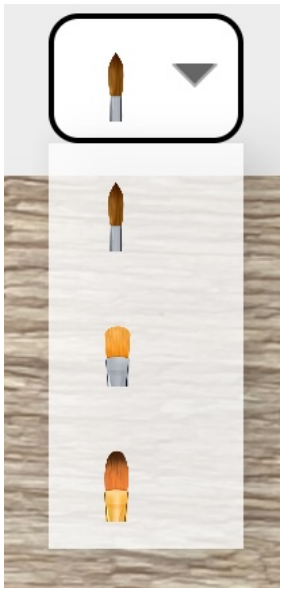


Figure 3.17.: Brush options
1) Click the Brush drop down menu
2) Choose the desired brush style
3) The brush will update to the selected option

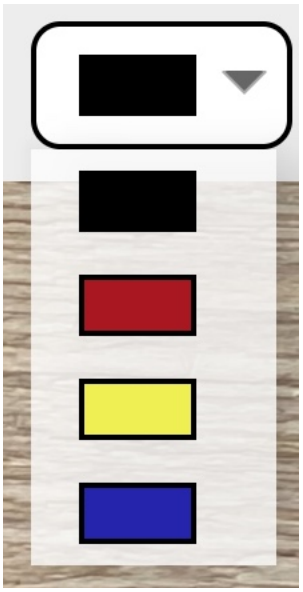


Figure 3.18.: Color palette
1) Click the color drop down menu
2) Pick the desired color from the palette
3) The color will switch to the selected option

Once the users have selected their preferred canvas, brush, and color, they can begin painting directly on the canvas using the stylus, and clear the painting when they want to start over.

4. User Studies

In this section, we conduct different types of user studies to evaluate the system, user experience and test our product. We conduct user studies, to obtain insights into user preferences, satisfaction, and overall interaction with the application. We conducted two different user test, one to test the realism and one to test the usability.
For both of out studies we used a within-subjects design by having all participants test all the conditions and scenarios of the user interface.

4.1. Realism user study

In this user study, the haptic feedback from the real-life canvases and brushes will be compared with the virtually created canvases and brushes. Due to limited time, and what can actually be expected of the test participants, we do not look at all the possible comparisons that can be made. Instead, we choose some selected canvases and brushes to compare. To make sure that the order of the different comparisons wouldn't bias the results the order was randomized.
SKRIV OM HVOROFR DER ER 7
The seven participants ware asked to rate the comparisons on a scale from 0 to 10, where 0 is zero difference and 10 is completely different.

We first wanted to test the way different canvases influence haptic feedback. Here we will keep the brush type constant and compare the sensations of using this brush (both real-life and virtual) on the different canvases and ask how different the sensation is. For this test, we chose to use the square brush on all four canvases since the square brush is the brush that creates the most sensation.
Each virtual canvas with the constant brush is then compared to the real-life ones, and the user is asked to rate the difference.
A table of the different comparisons for the real-real canvases alone, and the virtual-virtual canvases looks like:

Canvas Test **Real vs. Real**
Wood vs. Glass
Wood vs. Canvas
Wood vs. Silk
Canvas vs. Silk
Canvas vs. Glas
Silk vs. Glas

Canvas Test **Virtual vs. Virtual**
Wood vs. Glass
Wood vs. Canvas
Wood vs. Silk
Canvas vs. Silk
Canvas vs. Glas
Silk vs. Glas

A table of the different virtual canvases and real canvases compared looks like:

Canvas Test **Virtual vs. Real**
Virtual Wood vs. Real Wood
Virtual Wood vs. Real Glass
Virtual Wood vs. Real Canvas
Virtual Wood vs. Real Silk
Virtual Canvas vs. Real Wood
Virtual Canvas vs. Real Glass
Virtual Canvas vs. Real Canvas
Virtual Canvas vs. Real Silk
Virtual Silk vs. Real Wood
Virtual Silk vs. Real Glass
Virtual Silk vs. Real Canvas
Virtual Silk vs. Real Silk
Virtual Glass vs. Real Wood
Virtual Glass vs. Real Glass
Virtual Glass vs. Real Canvas
Virtual Glass vs. Real Silk

In the second test we wanted to test the way different brushes influence haptic feedback. Here we will keep the canvas type constant and compare the sensations of using different brushes on this canvas. This is done with the virtual brushes on the virtual constant canvas and the real-life brushes on the real-life constant canvas. We will ask how different the sensation is and the user is asked to rate the difference.
Each virtual brush with the constant tree is then compared to the real-life ones, and the user is asked to rate the difference.
A table of the different virtual and real brushes compared looks like:

Brush Test **Virtual vs. Real**
Virtual Thin vs. Real Thin
Virtual Thin vs. Real Square
Virtual Thin vs. Real Round
Virtual Square vs. Real Thin
Virtual Square vs. Real Square
Virtual Square vs. Real Round
Virtual Round vs. Real Thin
Virtual Round vs. Real Square
Virtual Round vs. Real Round

For both the tests the participants were blindfolded and wore noise-cancelling headphones, to ensure that they weren't influenced by any visual or auditory feedback.

4.2. Usability user study

We want to evaluate and test the usability of the application. Here it is important to note that usability is not a single property of a user interface, but has many different components (Nielsen, 1994).
Nielsen associates the definition of usability with these five usability attributes (Nielsen, 1994):

- **Learnability:** The system should be easy to learn so that the user can rapidly start getting some work done with the system.
- **Efficiency:** The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible.
- **Memorability:** The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
- **Errors:** The system should have a low error rate, so that users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.
- **Satisfaction:** The system should be pleasant to use, so that users are subjectively satisfied when using it; they like it.

The primary objective of our user studies was to evaluate whether the experience felt realistic. Our goal is to create a convincing and realistic simulation using haptic and auditory feedback. To address this, we added a sixth component to Nielsen's five key components: Realism.

- **Realism**: The degree to which the experience feels realistic and convincing to the user.

The realism attribute was tested in our first user study (see 4.1 Realism user study). To test the five other usability attributes, we designed a series of tasks involving different scenarios for our three participants. Nielsen states that 5 is the best number of participants for user studies (Nielsen, 1994). But since realism was a critical aspect of our project and given the small user-interface, we only found it relevant to involve 3 test participants.

During the testing process, we presented the three participants with our application under three different conditions:

1. Without any haptic or auditory feedback:
2. With only haptic feedback: This allowed us to isolate the effect of the haptic feedback.
3. With only auditory feedback: This allowed us to isolate the effect of the auditory feedback.

For each of the scenarios, we asked the participants the same set of questions, with some scenario-specific questions added, to consistently gain insight into their perception and experience.
The participants will then answer each question using a Likert scale ranging from 1 to 5, with 1 being the worst rating (Strongly disagree),and 5 being the best rating (Strongly agree) and 3 is the neutral (Nielsen, 1994)
Using this scale will help us understand their perceptions and experiences, and make it easier to compare and analyze the data collected from the conducted user study. We encourage participants to avoid selecting 3 to ensure more definitive feedback and that they take a clear stance on their feedback.

By using the method of testing the application under four different conditions, we were able to evaluate and compare how the haptic and auditory feedback each influenced the realism of the user experience.

For each of the scenarios, we gave the participants a set of instructions:

1. Change the background to your desired texture
2. Change the brush to your desired brush
3. Change the color to your desired color
4. Draw a triangle
5. Clear the drawing

We gave them these instructions to make sure that we created an authentic user experience, where all of the functionalities were used.

We then asked the participants a list of scenario-specific questions and then some questions based on the six stated usability components, which they answered using the 1-5 scale.

Scenario 1: Without any haptic or auditory feedback
1. How engaging did you find the painting experience?

Scenario 2: With only haptic feedback
1. How much did the haptic feedback enhance your painting experience?

Scenario 3: With only auditory feedback
3. How much did the auditory feedback enhance your painting experience?

To ensure that the order of the given scenarios did not bias the results, we used a Latin square to randomize the sequence in which each participant experienced the scenarios:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

The scenario based questions helped us understand the users overall experience and, most importantly, how realistic the feedback felt. These questions also allowed us to see how different types of feedback affected the user experience.

We were then interested in questions regarding the overall usability of the application:

Learnability:
1. How easy was it to understand how to use the application for the first time?
2. Was the application navigation as expected?

Efficiency:
3. How quickly were you able to efficiently complete the task?
4. Do you think that the application allows you to complete the task efficiency.

Memorability:
5. If you were to complete the same task again after a long time, do you think you would be able to remember how to use it?

Errors:
6. If you encountered any errors during the task, how easily were you able you recover from them?
7. Did you encounter any difficulties or errors when learning how to use the application?

Satisfaction:
8. How satisfied are you with the overall experience?

By conducting this user study, we ensure that our application is intuitive, effective and provides the user with a satisfying user experience. The feedback we gain from our user studies will be useful and valuable in identifying improvements and is used when refining the application, since we are able to tailor it to meet the needs and preferences of our target users.

5. Results

In this section, we will present the results of the different user studies and the implementation tests.

5.1. Realism user study results

From the realism user study, we were able to visualize the average results necessary for us to evaluate the virtual textures' similarity to textures in real life.

In Figure 5.1, the results of the average rating of the real texture comparisons can be seen. An empty spot in the figure is a comparison with the texture itself.

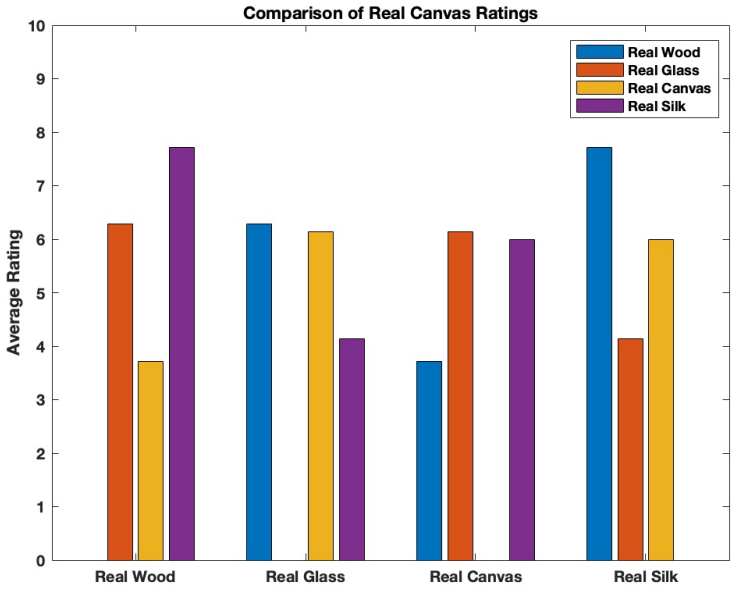


Figure 5.1.: Results of the average rating when the participants compared real textures with the real square brush

We note that the participants experienced that the haptic feedback from the wood and canvas textures felt the least different and that the haptic feedback from silk and glass textures felt the lest different.
We also note that the participants felt a certain difference between all the textures.

In Figure 5.2, the results of the average rating of the virtual texture comparisons can be seen. Again, the empty spot in the figure is a comparison with the texture itself.

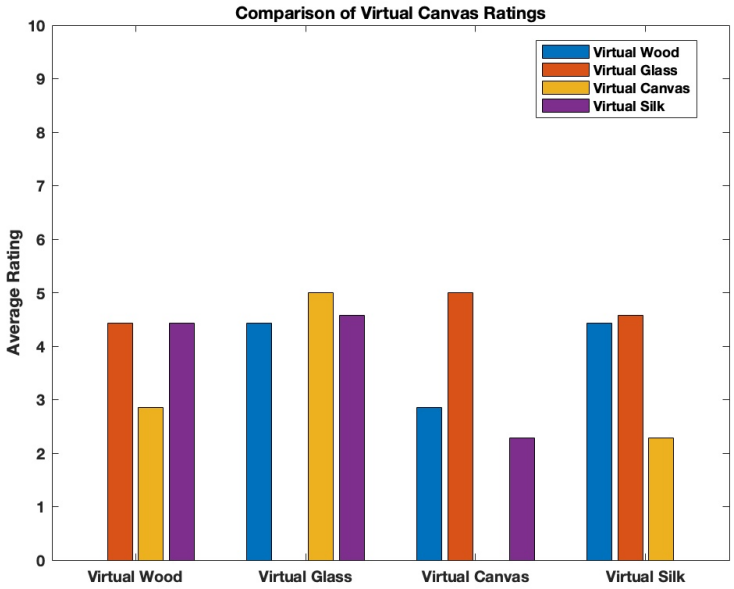


Figure 5.2.: Results of the average rating when the participants compared virtual textures with the virtual square brush

We note that the participants experienced the haptic feedback from the virtual canvas and silk textures as feeling the least different. The haptic feedback from the virtual wood texture felt the least different from the virtual canvas texture. Also, the difference between the glass texture and the rest of the textures was felt almost equally different, but the wood texture was slightly less different.
We also note that the difference between the virtual textures was felt less by the participants than the difference between the real textures.

In Figure 5.3, the results of the average rating of the virtual and real texture comparisons can be seen.

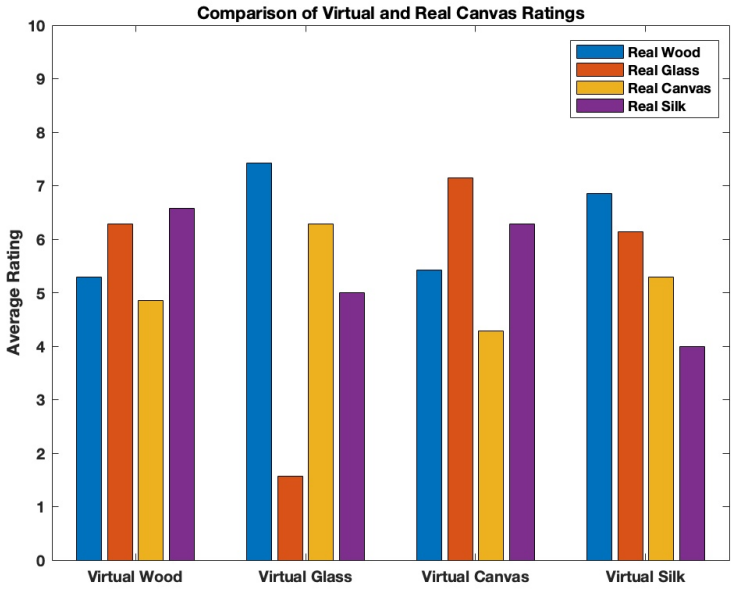


Figure 5.3.: Results of the average rating when the participants compared virtual textures and real-life textures with respectively the virtual and real square brush

We note that the virtual glass, virtual canvas, and virtual silk haptic feedback are experienced as being the least different from their real counterpart textures. The virtual glass texture was especially dissimilar from the textures that weren't glass. Virtual wood is felt as being the least different from the real canvas texture, where real wood is a close second.

In Figure 5.4, the results of the average rating of the virtual and real brush comparisons can be seen.

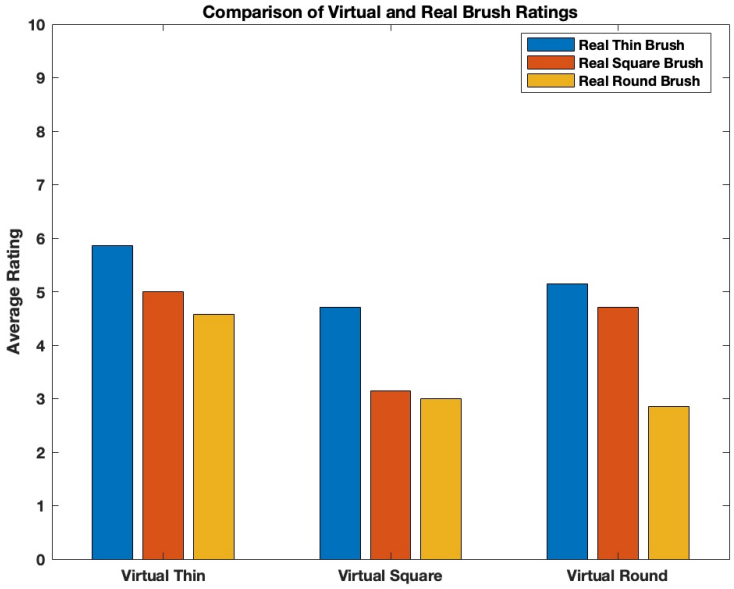


Figure 5.4.: Results of the average rating when the participants compared virtual and real-life brushes on respectively virtual and real wood

The virtual round brush is the least different from its real counterpart. We note that the participants experienced the real square brush and the real round brush as being close in dissimilarity to the virtual square brush, but the real round brush is slightly less dissimilar. We also note that the thin virtual brush was experienced to be the most dissimilar to the real thin brush.

5.1.1. Comments made by participants during the realism study

During the realism user study, the participants often commented on why they scored as they did, or what they think could have been done to enhance the haptic feedback. All participants agreed they didn't feel much from the virtual glass texture haptic feedback. However, some people compared feeling a little haptic feedback in one texture and nothing in the virtual glass texture as them being not that different from each other, whereas others experienced this as being completely different.
The older participants couldn't feel any haptic feedback at all unless they held on to the actuator directly or the actuator with the holder away from the pen. The younger participants also described the haptic feedback as being too low, and that they would prefer it to be stronger, even if this would make the feedback more distant from the real haptic feedback.
Some people also described having difficulty comparing the haptic feedback from the virtual and real textures, although for several reasons. One reason was the weight of the real brushes and the stylus. The weight of the stylus made it more difficult to compare the haptic feedback from the virtual and real textures because the real brushes' weight were lighter.
Another reason was the different direction from which the vibration came. Some participants became bewildered because the haptic feedback from the virtual textures traveled from above and down into the fingers, whereas the haptic feedback from the real canvases traveled from below and up into the fingers.
A third reason was that some participants experienced the haptic feedback from the virtual textures as "resistance" whereas the real texture haptic feedback was described more as vibrations.

When it came to the different sensations from the haptic feedback of the textures, there were also several comments to explain the score.
Many described the wood texture as bumpy, whereas silk, canvas, and glass felt more smooth. One participant could also clearly feel the difference between two textures, but would still not describe it as that different (above 5) despite this.

5.2. Usability user study results

We visualized the results from the usability study as the average results for the scenario-based questions. This can be seen in Figure 5.5.

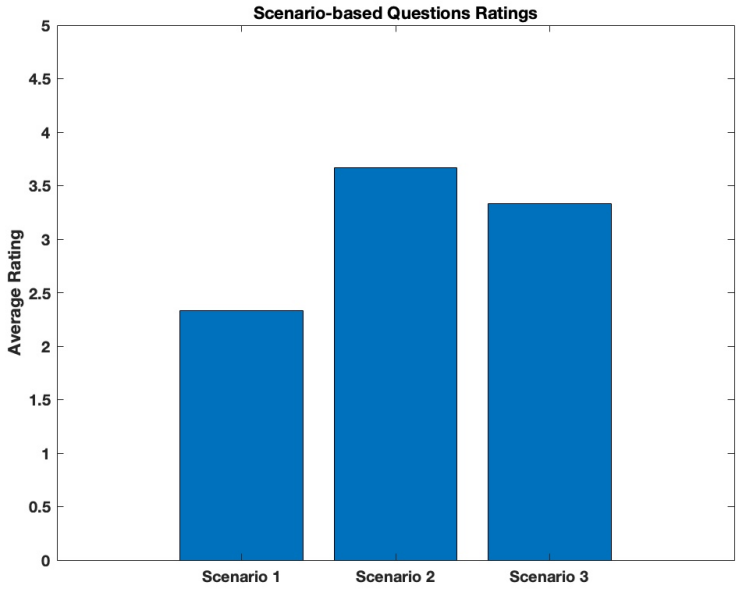


Figure 5.5.: Results of the rating of the scenario-based questions

We note that the engagement was highest for the scenario with only haptic feedback and the lowest for the scenario without any haptic or auditory feedback.

We were also able to visualize the average results from each of the usability questions. This can be seen in Figure 5.6.

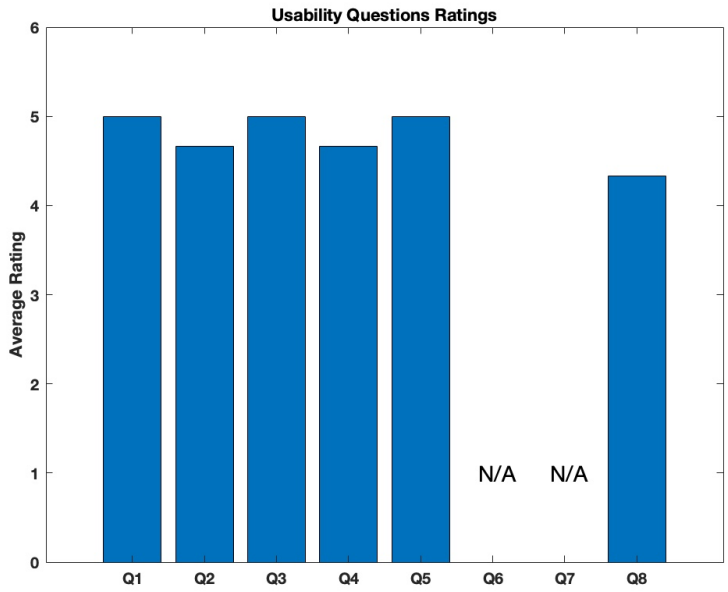


Figure 5.6.: Results of the rating of the usability questions

There are no ratings from questions 6 and 7 since none of the test participants encountered any errors or difficulties (this is shown as N\A in Figure 5.6).
We note that the overall rating in all the questions is above four. In question 2, the rating has a slightly lower average than the rest because one of the participants was

confused about which brush was square and round when looking at the pictures. In question 4, the average rating was slightly lower than the rest of the questions' because one participant thought the dropdown menus didn't show fast enough for him to be efficient.
In question 8, the lowest average rating was given since one of the participants had considerable experience with other drawing/painting apps, and they found the drawing interface to be too simple.

5.2.1. Comments made by participants during the usability study

During the usability user study, the participants often commented on why they scored as they did, or what they think could have been done to enhance the user experience. One participant noted that the reaction time of the different drop down menus was too slow, which disturbed the interaction flow.
Another participant suggested having an explanatory test for the different brushes, as they found it hard to see the difference in the small icons.
The same participant suggested exaggerating the haptic feedback, even if it's not entirely realistic, to make the interaction more noticeable and satisfying.

5.2.2. Instrumented test results

All the instrumented tests explained in section 3.6 Test passed successfully, which showed consistent and reliable functionality across all tested classes.

5.2.3. noter

As_mag for frequency accz_g for at se om det er godt (jo mere stabil jo bedre) interpolation Write some code to look after the correct velocity in the velocity (maybe look at the velocity calcula-

6. Discussion

QUESTION: SHOULD WE HAVE A TEST DISCUSSION SUBSECTION AND A IMPLEMENTATION DISCUSSION SUBSECTION???

The realism user study showed that even though the participants didn't feel as big a difference between the haptic feedback from the virtual textures as they did between the haptic feedback from the real textures, they still experienced the virtual glass, canvas, and silk as being the least different from their real counterparts. This shows that the participants can feel the difference between the virtual textures when compared to real textures. However, the virtual wood texture felt the least different from the real canvas texture, and this average rating was higher than the average rating of the other virtual textures and their corresponding least different real canvas. This indicates that at least the virtual wood haptic feedback should be fine-tuned again.

Looking more closely at the individual virtual textures, we see that the participants on average felt that the virtual wood texture was least different from the virtual canvas texture and that the virtual glass and silk texture approximately had the same dissimilarity. The participants had the same opinions when comparing the real textures, although here they felt that the silk texture was even more dissimilar than the glass texture. When comparing the real textures and the virtual textures, the dissimilarity to the virtual wood was from least dissimilar to most: real canvas, real wood, real glass, and real silk. Despite these results following the same dissimilarity as with the virtual texture comparisons and real texture comparisons, the real wood is not experienced as the least dissimilar to the virtual wood. Thus, the virtual wood haptic feedback should be adjusted so it feels more similar to the real wood texture.

Furthermore, the study showed that the participants on average felt that the virtual glass texture was most different from the virtual canvas texture and that the virtual wood and silk texture approximately had the same dissimilarity. The participants' perception was the same when looking at dissimilarities between the real textures (that weren't glass) and the virtual glass texture. However, when comparing the real textures, the participants felt that the glass texture was least dissimilar to the silk texture and that the wood and canvas texture had approximately the same dissimilarity. That is, the virtual glass texture is least dissimilar to the real glass, but the dissimilarity between the virtual wood texture and the virtual glass texture should be bigger. This could be because the virtual wood texture was not ranked as the least dissimilar to the real wood texture.

Additionally, the study showed that the participants on average felt that the virtual canvas texture was most different from the virtual glass texture and that the virtual silk texture was the least dissimilar. The participants' perceptions were almost identical when looking at the dissimilarities between the real textures and the virtual canvas

texture. Here the real glass texture felt the most dissimilar, and then silk, wood, and canvas. This is the exact dissimilarity scale for the real texture comparisons. Again, the difference in the dissimilarity scale between virtual canvas and wood texture and the real canvas and wood texture is most likely due to the virtual wood texture not being ranked as the least dissimilar to the real wood texture.

Moreover, the study showed that the participants on average felt that the virtual silk texture was most different from the virtual glass and wood texture and that the virtual canvas texture was the least dissimilar. The participants' perceptions were very close when looking at the dissimilarities between the real textures and the virtual silk texture. Here the dissimilarity to the virtual silk texture was from least dissimilar to most: real silk, real canvas, real glass, and real wood, while the dissimilarity to the real silk texture is: real silk, real glass, real canvas, and real wood. This indicates that despite the virtual silk texture being the least different from the real silk texture, it might need to be fine-tuned again, such that the dissimilarity will be smaller to virtual glass and bigger to virtual canvas.

Here we're going to write about the brush part of the realism user study similar to the canvas part written above

From the user comments during the realism user study, we also found that all participants felt the haptic feedback was too low. Thus, it is clear that we would have to up the gain for each of the haptic feedback `.wav` files, even if this would make the haptic feedback less realistic. We should especially consider this if we intend for an older demographic to be part of the intended user group since they had the most difficulty feeling the haptic feedback.

The usability user study showed that although people found that the app was more engaging with either haptic feedback or auditory, compared to other painting apps, the visualization was a bit disappointing.
The usability study showed that most of the participants wished for a more exaggerated haptic experience. Exaggerating the haptic feedback would deviate from the real haptic feedback, and make it less true to reality, but would result in higher satisfaction since it would provide a more noticeable haptic experience. This could be solved by exaggerating and tuning the haptic feedback by adjusting the gain, but still maintaining a realistic vibration.

Though the implementation of the application has been mostly successful, several challenges have also risen during the process.

We found a flaw that was not exposed in the usability user study, but that was sometimes experienced in the realism user study: The haptic feedback would sometimes not stop if the user stopped touching the screen as the actuator would keep vibrating if several clicks were made within one second. This meant the vibration felt stronger if the user touched the screen again because two vibrations would vibrate simultaneously. This disturbed the realism of the application.
This happened because the application couldn't keep up with the speed of the user's touching and removal of contact with the screen. Although a solution to this problem

is to restart the app or turn off the amplifier completely, something should be done in the future to solve this problem.

In addition to this, we encountered several difficulties During the implementation of haptic and auditory feedback. We found that in some cases the haptic or auditory output sounded more bumpy, which was better for the haptic feedback, but not for the auditory feedback. This implementation made us able to route the haptic feedback to the aux line and the auditory feedback to a Bluetooth headset. Since we weren't satisfied with the the bumpiness of the auditory feedback, we tried changing the implementations, so it was smoother, but doing this removed our ability to route the auditory feedback.
We then considered other options like playing both the haptic and auditory feedback through the aux line but splitting it, so one out would be connected to headphones and another to the amplifier. This was however not possible due to the different sample rates of the feedback.
That is, through this experimentation of the auditory and haptic implementation, we found that it was not easily possible to play haptic and auditory feedback simultaneously without the auditory feedback being less than ideal.
In general, we learned that there are certain hardware constraints, which make it impossible to play something through the aux line and internal speakers simultaneously unless it is the same file.

7. Conclusion and future work

In conclusion, we explored and developed the idea of creating a painting application with integrated haptic feedback to enhance the user experience by simulating the tactile sensations of traditional painting, and thereby create a more engaging experience. The development process involved implementing a application for the rendering system containing various functionalities, and setting up the hardware for interaction with the application.
Our aim was to create realistic vibration feedback, that reacts to the users interactions in real-time, which we achieved by modulating the haptic feedback by undergoing a data segmentation process, and using audio processing tools to adjust the amplitude. By adjusting the amplitude and high-pass filters, we ensured that the feedback was as realistic as possible.
Additionally, we also implemented auditory feedback to further enhance the user experience, this made it possible for the users to experience real-time auditory feedback as they paint.

Throughout the implementation and development phases, we encounted several challenges, such as the implementation of combined haptic and audio feedback, that we were not able to impelement due to hardware limitations. This limitation restricted our ability to combine the haptic and auditory feedback, which would have created a more engaging and realistic user experience.
Despite this challenge, we successfully demonstrated the potential of combining haptic and audio feedback to enhance digital painting applications, since we were still able to implement each of the parts and test them separately.
skriv om andre fejl/challenges (NOT FINISHED)

The user studies we conducted as part of this thesis provided us with valuable feedback about the realism of the implemented features.**skriv lidt om hvad vi fandt ud af (NOT FINISHED)**

Future work could focus on how to combine the auditory and haptic feedback

We also considered other implementations that would improve the application's realism, satisfaction, and overall user experience. These could have been implemented if given more time and resources. We describe these implementations as "nice-to-have"s since they are beyond the app's needed core functionalities. For the canvas the "nice-to-have" functionalities would be that we would want to implement that the user should be able to save their painting and revisit it at any point even if they have started on a different painting. This would provide the user would greater flexibility, and is more true to the real world. Also the user should be able to save their painting in their camera roll, which would enable the user to not only share, but also save their paintings somewhere safe. Another "nice-to-have" would be visual indentation, the

canvas needs to have visual indentation when drawn on. this would add another layer of realism, the tactile feedback could mimic the the physical sensation of painting on a canvas with a given texture.
For the brush, we would have adjustable brush thickness, The user should be able to choose the thickness of the chosen brush, which would give the user more control over their artwork and mimic the different paint brushes an artist would want to use.
Another way to make the application more immersive and engage the user would be personalized mixed colors, the user should be able to mix user-designed colors and paint with these. By allowing the users to mix their own colors and paint with them, we would closely mimic the traditional painting process and enhance the realism of the painting experience.

References

AndroidDevelopers. (n.d.). *Test in android studio.* Retrieved 03.06.2024, from https://developer.android.com/studio/test/test-in-android-studio

Fouad, M., Mansour, T., & Nabil, T. (2023). *Use of haptic devices in education: A review* (Tech. Rep.). Mechanical engineering Dept, Engineering, Ismailia, Egypt.

HapticLab. (n.d.). *Haptics 1x1.* Retrieved 12.05.2024, from https://www.hapticlabs.io/haptics1x1

Kern, T. A., Hatzfeld, C., & Abbasimoshaei, A. (2023). *Engineering haptic devices - third edition.* Springer International Publishing.

Kirginas, S. (2022). *Exploring players' perceptions of the haptic feedback in haptic digital games* (Tech. Rep.). National and Kapodistrian University of Athens, Greece.

Nielsen, J. (1994). Usability engineering.

Roca, C. (n.d.). *What are the 5 solid principles? what is solid and what is it for?* Retrieved 22.05.2024, from https://www.thepowermba.com/en/blog/what-are-the-solid-principles

TDK. (2016). *Mpu-6050, six-axis (gyro + accelerometer) mems motiontracking™ device.* Retrieved 22.05.2024, from https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/#:~:text=The%20MPU%2D6050%20devices%20combine,complex%206%2Daxis%20MotionFusion%20algorithms

A. Code snippets used for data-driven haptics

A.0.1. MATLAB code `arduinoiDataRead.m`

This MATLAB code saves the data from the accelerometer into a `.csv` file and decides the baud rate (which should be the same as in the Arduino IDE app), and the data recording length.

```
function arduinoDataRead(filenameData)
port = '/dev/tty.usbmodem11101';
baudRate = 115200;
s = serialport(port, baudRate);
recDuration = 4;
% Open a file to save the data
fileID = fopen([filenameData,'.csv'], 'w');

tic; % Start timer
while toc < recDuration
    data = readline(s);
    fprintf(fileID, '%s\n', data);
end

fclose(fileID);
delete(s);
clear s;

end
```

A.0.2. MATLAB code `sound.m`

This MATLAB code saves the microphone-recorded audio into a `.wav` file and decides the sample rate, bits per sample, and the recording length.

```
function sound(filenameSound)
disp(audiodevinfo);

% Set parameters for recording
Fs = 44100;  % Sample rate in Hz
nBits = 16;  % Bits per sample
nChannels = 1;  % Number of audio channels

% Select device ID from the output of audiodevinfo if needed
```

```
deviceID = 0;

% Create an audiorecorder object
recObj = audiorecorder(Fs, nBits, nChannels, deviceID);

% Duration of recording in seconds
recDuration = 4;
recordblocking(recObj, recDuration);
audioData = getaudiodata(recObj);

%audiowrite(['testet','.wav'], audioData, Fs);

audiowrite([filenameSound,'.wav'], audioData, Fs);
end
```

A.0.3. MATLAB code `parallel.m`

This MATLAB code makes it possible to runt he code of two different function in different files at the same time.

```
pool = gcp('nocreate');
if isempty(pool)
    pool = parpool;
end

filenameSound = (['round_glas_d_1'])
filenameData = filenameSound
f = parfeval(pool, @sound, 0, filenameSound);
f2 = parfeval(pool, @arduinoDataRead, 0, filenameData);
wait(f);
```

A.0.4. MATLAB code `VelocityAcceleration.m`

This MATLAB code converts raw acceleration values to texture acceleration, hand acceleration, and velocity. Note that the `filename` variable is changed depending on the file we want to get data from.

```
clear;clc;

filename = 'thin_wood_s_4';
% read data from file
data = readmatrix([filename,'.csv']);

% Extract Z-axis acceleration data
accZ = data(:, 4);

% Time and acceleration in SI units
time = data(:, 1) / 1000;  % Convert milliseconds to seconds
```

```
% Convert raw values to acceleration in g's
% ±8g full-scale range. change this to match sensitivity of your accelerometer
full_scale = 8;
adc_resolution = 2^16; % 16-bit ADC. this is same for all ADCs
g_conversion_factor = full_scale / (adc_resolution / 2);

% convert acceleration from bits to g
accZ_g = accZ * g_conversion_factor;

% Convert acceleration from gs to m/s2
acceleration = accZ_g*9.8;

% maximum acceleration in the signal
peak_acceleration = max(abs(real(acceleration)));

Fs = 1000;  %  sample rate

% High pass filter to isoltate the texture frequency
%Fc = 20;  % Cut-off frequency
Fc = 20;  % Cut-off frequency
[bh, ah] = butter(1, Fc/(Fs/2), 'high');
% 1st order Butterworth high-pass filter
filtered_accelerationHigh = filtfilt(bh, ah, acceleration);


% Band pass filter to isolate the frequency of hand motion and
%remove low freq noise
%Fh = 20; % upper cut off frequency
Fh = 20; % upper cut off frequency
Fl = 1; % lower cut off frequency
[b, a] = butter(1, [Fl/(Fs/2) Fh/(Fs/2)], 'bandpass');
% 1st order Butterworth high-pass filter
%filtered_accelerationBandPass = filtfilt(b, a, acceleration);


% YOU DO NOT NEED THIS PART FOR NOW
% Frequency analysis using FFT
% Fs = 190;  % Sampling frequency in Hz
%(adjust based on your data acquisition rate)
% L = length(filtered_acceleration);  % Length of signal
% f = Fs*(0:(L/2))/L;  % Frequency vector
% FFT_accZ = fft(filtered_acceleration);  % Compute FFT
% P2 = abs(FFT_accZ/L);  % Two-sided spectrum
% P1 = P2(1:L/2+1);  % Single-sided spectrum
% P1(2:end-1) = 2*P1(2:end-1);  % Correct the amplitude
%
%
% % plot the frequency spectrum
```

```
% figure;
% plot(f, P1)
% xlabel('Vibration Frequency', 'FontSize', 12);
% ylabel('Magnitude', 'FontSize', 12);
% grid on;
% box on;


% Integrate the filtered acceleration data to calculate velocity
filtered_velocity = zeros(size(filtered_accelerationBandPass));
for i = 2:length(filtered_accelerationBandPass)
    dt = time(i) - time(i - 1);
    filtered_velocity(i) = filtered_velocity(i - 1) +
    filtered_accelerationBandPass(i) * dt;
    %filtered_velocity(i) = filtered_accelerationBandPass(i) * dt;
end

% Plot the filtered velocity
figure;
plot(time, filtered_velocity);
title('Velocity computed from Hand Acceleration');
xlabel('Time (s)');
ylabel('Velocity (m/s)');

% Plot the acceleration of hand
figure;
plot(time, filtered_accelerationBandPass);
title('Hand Acceleration');
xlabel('Time (s)');
ylabel('Acceleration (m/s^2)');

% Plot the acceleration due to texture
figure;
plot(time, filtered_accelerationHigh);
title('Texture acceleration');
xlabel('Time (s)');
ylabel('Acceleration (m/s^2)');
```

A.0.5. MATLAB code `slowAcceleration.m`

This MATLAB code extracts the wanted acceleration data for slow velocity. This function is added to the code in Appendix A.0.4 when needed.

```
function slowAcceleration(filtered_velocity, filtered_accelerationHigh, filename)
Fs = 1000;  %  sample rate

% the indexes for where the velocity is the right value
velocity_indexes = zeros(size(filtered_velocity));
```

```
% how many indexes of correct velocity value is there
counter = 0;
for i = 1:length(filtered_velocity)
    if (filtered_velocity(i) > 0.0145 && filtered_velocity(i) < 0.0155) ||...
       (filtered_velocity(i) < -0.0145 && filtered_velocity(i) > -0.0155) && ...
       (counter == 0 || velocity_indexes(counter) == i-1)
        counter = counter + 1;
        % save index of velocity at the next empty index
        velocity_indexes(counter) = i;
    end
end
% trim data so it is only the length of actual indexes
velocity_indexes = trimdata(velocity_indexes, counter);

% get the data at the velocity indexes from filtered_accelerationHigh
texture_acceleration = zeros(size(velocity_indexes));
for i = 1:length(velocity_indexes)
    texture_acceleration(i) = filtered_accelerationHigh(velocity_indexes(i));
end

% repeat the data of acceleration
texture_acceleration = ...
    repmat(texture_acceleration,ceil(80/length(texture_acceleration)),1);

% create audiofile from accelerationdata
audiowrite([filename,'_haptics.wav'], texture_acceleration, Fs);
end
```

A.0.6. MATLAB code `mediumAcceleration.m`

This MATLAB code extracts the wanted acceleration data for medium velocity. This function is added to the code in Appendix A.0.4 when needed.

```
function slowAcceleration(filtered_velocity, filtered_accelerationHigh, filename)
Fs = 1000;  %  sample rate

% the indexes for where the velocity is the right value
velocity_indexes = zeros(size(filtered_velocity));

% how many indexes of correct velocity value is there
counter = 0;
for i = 1:length(filtered_velocity)
    if (filtered_velocity(i) > 0.0295 && filtered_velocity(i) < 0.0305) ||...
       (filtered_velocity(i) < -0.0295 && filtered_velocity(i) > -0.0305) && ...
       (counter == 0 || velocity_indexes(counter) == i-1)
        counter = counter + 1;
        % save index of velocity at the next empty index
        velocity_indexes(counter) = i;
    end
```

```
end
% trim data so it is only the length of actual indexes
velocity_indexes = trimdata(velocity_indexes, counter);

% get the data at the velocity indexes from filtered_accelerationHigh
texture_acceleration = zeros(size(velocity_indexes));
for i = 1:length(velocity_indexes)
    texture_acceleration(i) = filtered_accelerationHigh(velocity_indexes(i));
end

% repeat the data of acceleration
texture_acceleration = ...
    repmat(texture_acceleration,ceil(80/length(texture_acceleration)),1);

% create audiofile from accelerationdata
audiowrite([filename,'_haptics.wav'], texture_acceleration, Fs);
end
```

A.0.7. MATLAB code `fastAcceleration.m`

This MATLAB code extracts the wanted acceleration data for fast velocity. This function is added to the code in Appendix A.0.4 when needed.

```
function slowAcceleration(filtered_velocity, filtered_accelerationHigh, filename)
Fs = 1000;  %  sample rate

% the indexes for where the velocity is the right value
velocity_indexes = zeros(size(filtered_velocity));

% how many indexes of correct velocity value is there
counter = 0;
for i = 1:length(filtered_velocity)
    if (filtered_velocity(i) > 0.0595 && filtered_velocity(i) < 0.0605) ||...
       (filtered_velocity(i) < -0.0595 && filtered_velocity(i) > -0.0605) && ...
       (counter == 0 || velocity_indexes(counter) == i-1)
        counter = counter + 1;
        % save index of velocity at the next empty index
        velocity_indexes(counter) = i;
    end
end
% trim data so it is only the length of actual indexes
velocity_indexes = trimdata(velocity_indexes, counter);

% get the data at the velocity indexes from filtered_accelerationHigh
texture_acceleration = zeros(size(velocity_indexes));
for i = 1:length(velocity_indexes)
    texture_acceleration(i) = filtered_accelerationHigh(velocity_indexes(i));
end
```

```
% repeat the data of acceleration
texture_acceleration = ...
    repmat(texture_acceleration,ceil(80/length(texture_acceleration)),1);

% create audiofile from accelerationdata
audiowrite([filename,'_haptics.wav'], texture_acceleration, Fs);
end
```

A.0.8. Arduino IDE code for collecting accelerometer data

```
#include <Wire.h>

// MPU6050 I2C address
const int MPU = 0x68;

void setup() {
  Serial.begin(115200); // Start the serial communication with a baud rate of 11520
  Wire.begin(); // Initialize I2C communication

  // Wake up MPU-6050
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);
  Wire.write(0); // Wake up MPU-6050 from sleep mode
  Wire.endTransmission(true);

  // Set sample rate to 1kHz by setting the sample rate divider to 0
  Wire.beginTransmission(MPU);
  Wire.write(0x19);
  Wire.write(0); // SMPLRT_DIV = 0
  Wire.endTransmission(true);

  // Set DLPF to a value for 1kHz sample rate (e.g., DLPF_CFG = 2 for ~92Hz bandwi
  Wire.beginTransmission(MPU);
  Wire.write(0x1A);
  Wire.write(2); // DLPF_CFG = 2
  Wire.endTransmission(true);

  // Set accelerometer to ±8g sensitivity
  Wire.beginTransmission(MPU);
  Wire.write(0x1C); // Accelerometer Configuration register
  Wire.write(0x10); // AFS_SEL = 2 for ±8g range (0x10 = 00010000)
  Wire.endTransmission(true);

  Serial.println("Time (ms),AccelX,AccelY,AccelZ"); // Print CSV header to serial
}

void loop() {
  // Request accelerometer data
```

```
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with ACCEL_XOUT_H
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Request 6 bytes from the accelerometer registe

  // Read the accelerometer data
  int16_t AcX = Wire.read() << 8 | Wire.read();
  int16_t AcY = Wire.read() << 8 | Wire.read();
  int16_t AcZ = Wire.read() << 8 | Wire.read();

  // Print data to Serial
  Serial.print(millis());
  Serial.print(",");
  Serial.print(AcX);
  Serial.print(",");
  Serial.print(AcY);
  Serial.print(",");
  Serial.println(AcZ);
}
```

B. Code snippets from application implementation

B.0.1. The function `playParametricHaptics` in the `DrawingView` class

```
private void playParametricHaptics() {
    if (!playSound.isPlaying) {
        // Create a new thread to play the audio.
        // Performing intensive operations and computations on the main UI thread,
        // app slow. That is, it is a bad idea to do intensive computations on mai
        // so it is recommended to create a new thread to do computations in the b
        (new Thread(() -> {
            switch (chosenBackground) {
                case 0:
                    sampleRate = 41000;
                    amplitude = 3000;
                    frequency = 140;
                    break;
                case 1:
                    sampleRate = 41000;
                    amplitude = 3000;
                    frequency = 90;
                    break;
                case 2:
                    sampleRate = 41000;
                    amplitude = 3000;
                    frequency = 300;
                    break;
                case 3:
                    sampleRate = 41000;
                    amplitude = 3000;
                    frequency = 30;
                    break;
                default:
            }
            buffLength = AudioTrack.getMinBufferSize(sampleRate, AudioFormat.CHANNEL
            playSound.initTrack(sampleRate, buffLength);
            playSound.startPlaying();
            playSound.playback(amplitude, frequency, sampleRate, buffLength);
        })).start();
    }
```

```
}
```

C. Pseudocode of the application implementation

C.1. PlayMusic class

C.1.1. Method init

Algorithm 4 init in PlayMusic

1: **Input:** Context and sound file name
2: **Step 1:** Initialize MediaPlayer
3: Reset MediaPlayer and set its audio attributes
4: Try setting the data source of the MediaPlayer
5: Try preparing the MediaPlayer for playback asynchronously
6: **Step 2:** Initialize AudioManager
7: **Finish**

C.1.2. Method startPlaying

Algorithm 5 startPlaying in PlayMusic

1: **Input:** Context
2: **Step 1:** Direct sound output
3: **if** Bluetooth headset isn't connected **then**
4: Set the device's internal speakers on
5: **else**
6: Set MediaPlayer's device output as Bluetooth headset
7: **end if**
8: **Step 2:** Start the playback of the MediaPlayer
9: **if** MediaPlayer isn't playing **then**
10: Set the MediaPlayer to loop the playback
11: Start or resume the MediaPlayer playback
12: Set isPlaying variable to true
13: Set volume variable to 1
14: **end if**
15: **Finish**

C.1.3. Method setVolume

Algorithm 6 setVolume in PlayMusic

```
1: Input: Values for left volume and right volume
2: Step 1: Set MediaPlayer volume
3: if MediaPlayer isn't null then
4:     Set MediaPlayer volume to the input          ▷ left volume = right volume
5:     Set volume variable to left volume           ▷ left volume is either 0 or 1
6: end if
7: Finish
```

C.1.4. Method pausePlaying

Algorithm 7 pausePlaying in PlayMusic

```
1: Input:
2: Step 1: Pause MediaPlayer playback
3: if isPlaying variable is true then
4:     Set isPlaying variable to false
5:     Pause MediaPlayer playback
6: end if
7: Finish
```

C.1.5. Method stopPlaying

Algorithm 8 stopPlaying in PlayMusic

```
1: Input:
2: Step 1: Stop MediaPlayer playback
3: if isPlaying variable is true then
4:     Set isPlaying variable to false
5:     Stop MediaPlayer playback
6:     Release MediaPlayer resources
7: end if
8: Finish
```

C.1.6. Method findAudioDevice

Algorithm 9 findAudioDevice in PlayMusic

```
1: Input: Device type and Context
2: Step 1: Check if the device type can be connected to
3: if Context isn't null then
4:     Get info about all possible devices to connect to using AudioManager
5:     for Every device info do
6:         if device type = device type input then
7:             return device info
8:         end if
9:     end for
10: end if
11: return null
12: Finish
```

C.2. PlayHaptics class

C.2.1. Method startPlaying

Algorithm 10 startPlaying in PlayHaptics

```
1: Input: Context
2: Step 1: Direct sound output and start the playback of the MediaPlayer
3: if Aux line is connected then
4:     Set MediaPlayer's device output as the aux line
5:     Set the MediaPlayer to loop the playback
6:     Start or resume the MediaPlayer playback
7:     Set isPlaying variable to true
8:     Set volume variable to 1
9: end if
10: Finish
```

C.3. DrawingView class

C.3.1. Method changeCanvas

Algorithm 11 changeCanvas in DrawingView

```
1: Input: Canvas index
2: Step 1: Set new canvas background
3: Set chosenBackground ad the canvas index input
4: Call clearView
5: Finish
```

C.3.2. Method changeColor

Algorithm 12 changeColor in DrawingView

```
1: Input: Color
2: Step 1: Add to variable lists to change color
3: Set currentColor variable to input color
4: Create new Path
5: Add currentColor to color list, currentBrush to brush list, and the new path to
   Path list
6: Call on OnDraw function using invalidate()
7: Finish
```

C.3.3. Method changeBrush

Algorithm 13 changeBrush in DrawingView

```
1: Input: Brush width and Paint.Cap
2: Step 1: Add to variable lists to change brush
3: Set currentBrush variable to as a pair of the input width and input Paint.Cap
4: Create new Path
5: Add currentBrush to brush list, currentColor to color list, and the new path to
   Path list
6: Call on OnDraw function using invalidate()
7: Finish
```

C.3.4. Method onDraw

Algorithm 14 onDraw in DrawingView

```
1: Input: Canvas
2: Step 1: Paint background on canvas
3: Specify background bounds and draw the background based on the chosen canvas
4: Step 2: Paint the strokes on canvas
5: if Path list is empty then
6:     Paint using the drawPath and drawPaint variables
7: else
8:     for i = 0 to i < path list size do
9:         Sets the brush color, brush width, and Paint.Cap of the drawPaint
10:        Paint using paths from the path list and the drawPaint
11:    end for
12: end if
13: Finish
```

C.3.5. Method onSizeChanged

Algorithm 15 onSizeChanged in DrawingView

1: **Input:** Current width, current height, old width, and old length of DrawingView
2: **Step 1:** When size of view is changed
3: Calls on the functionality of Views onSizeChanged function
4: **Finish**

C.3.6. Method clearView

Algorithm 16 clearView in DrawingView

1: **Input:**
2: **Step 1:** Clear the view of any painting
3: **for** i = 0 **to** i < path list size **do**
4: Reset Path in path list
5: **end for**
6: **Step 2:** Call on onDraw function
7: Use invalidate() function
8: **Finish**

C.3.7. Method getVelocity

Algorithm 17 getVelocity in DrawingView

1: **Input:** MotionEvent and pointer identifier
2: **Step 1:** Get velocity in pixels per second
3: Compute the velocity per second of the MotionEvent
4: Get the x- and y-velocity
5: **Step 2:** Convert velocity to a total velocity in meters per second
6: **return** $\sqrt{(v_x^2 + v_y^2)} * \frac{0.225}{1920}$ ▷ screen length is 0.225 meters and 1920 pixels
7: **Finish**

C.3.8. Method onTouchEvent

Algorithm 18 onTouchEvent in DrawingView

```
1: Input: MotionEvent
2: Step 1: Get information about MotionEvent
3: Get the last point
4: Get the index associated with the MotionEvent and pointer identifier associated
   with the index
5: Step 2: Switch statement for different MotionEvents
6: switch (type of MotionEvent)
7:     case ACTION_DOWN
8:         if velocity tracker is null then
9:             Retrieve new velocity tracker
10:        end if
11:        Set starting point of the stroke
12:        Initialize and start the auditory and haptic feedback
13:        Call on OnDraw function using invalidate()
14:    case ACTION_MOVE
15:        Add a line from the last point in the Path to a new point
16:        Get velocity every 10 milliseconds
17:        Update the haptic and auditory feedback based on the velocity
18:        Call on OnDraw function using invalidate()
19:    case ACTION_UP
20:        Pause all auditory and haptic feedback
21:        Call on OnDraw function using invalidate()
22:    case ACTION_CANCEL
23:        Recycle the velocity tracker
24: Finish
```

C.4. ControlFeedback class

C.4.1. Method initializeFeedback

Algorithm 19 initializeFeedback in ControlFeedback

```
1: Input: 3 PlayInterface instances, context, and chosen brush and background index
2: Step 1: Get .wav files based on brush and background index
3: Save a fast, medium, and slow .wav file in three different variables
4: Step 2: Play the files saved in the variables
5: Initialize, start three MediaPlayers's playback using the saved files
6: Set the volume of the playbacks. Only the slow sound/haptics is unmuted
7: Finish
```

C.4.2. Method velocityFeedback

Algorithm 20 velocityFeedback in ControlFeedback

```
1: Input: 3 PlayInterface instances, and the velocity of the user's movement
2: Step 1: Play sound based on velocity
3: if velocity > 0.045 then                ▷ halfway between fast and medium velocity
4:     Unmute fast sound/haptics and mute medium and slow sound/haptics
5: else if velocity > 0.025 then           ▷ halfway between medium and slow velocity
6:     Unmute medium sound/haptics and mute fast and slow sound/haptics
7: else
8:     Unmute slow sound/haptics and mute fast and medium sound/haptics
9: end if
10: Finish
```

C.5. MainActivity class

C.5.1. Method canvasSpinnerSelection

Algorithm 21 canvasSpinnerSelection in MainActivity

```
1: Input:
2: Step 1: What happens when an item in the Spinner is selected
3: Change canvas index to the selected canvas' position and clear the view.
4: Step 2: What happens when no items in the Spinner is selected
5: Change canvas index to 0
6: Finish
```

C.5.2. Method colorSpinnerSelection

Algorithm 22 colorSpinnerSelection in MainActivity

```
1: Input:
2: Step 1: What happens when an item in the Spinner is selected
3: Change color of brush to the color selected in the spinner
4: Step 2: What happens when no items in the Spinner is selected
5: Change color of brush to black
6: Finish
```

C.5.3. Method brushSpinnerSelection

Algorithm 23 brushSpinnerSelection in MainActivity

```
1: Input:
2: Step 1: What happens when an item in the Spinner is selected
3: switch (chosen brush position)
4:     case 0
5:         Set chosen brush index as 0 (thin brush)
6:         Change brush with the width and Paint.Cap of the selected brush
7:     case 1
8:         Set chosen brush index as 1 (square brush)
9:         Change brush with the width and Paint.Cap of the selected brush
10:    case 2
11:        Set chosen brush index as 2 (round brush)
12:        Change brush with the width and Paint.Cap of the selected brush
13: Step 2: What happens when no items in the Spinner is selected
14: Set brush as standard brush (thin brush)
15: Finish
```

D. User study results

D.1. Comments from usability user study

for langsom reaktionstid, forklarende test til ikoner. Overdriv heller haptisk feedback, selvom det ikke er virkelighedstro.

D.2. Comments from the realism user study

We collected the comments from the realism user study in a table.

	Comments
M 23	• The haptic feedback is very low • If you turned up the vibrations a bit, it would feel exactly like the real canvas • The wood texture feels bumpier, whereas silk, canvas, and glass feel more smooth
M 23	• The haptic feedback is very low • I can't feel anything with the virtual glass, and I can feel a little with the other texture, so they are not that different • It is hard to really compare the different feedbacks because the feedback from the real textures comes from the bottom, and the feedback from the virtual textures comes from the top
M 51	• I can't feel the haptic feedback when I'm just holding on to the pen • The vibration should be stronger • The wood texture feels bumpier, whereas silk, canvas, and glass feel more smooth
F 52	• I can't feel the haptic feedback when I'm just holding on to the pen • The vibration should be stronger • I can't feel anything with the virtual glass, and I can feel a little with the other texture, so they are completely different • The haptic feedback on the virtual textures feels more like resistance, whereas the haptic feedback on the real textures feels more like vibrations. This makes it hard to compare
F 23	• The haptic feedback is very low • The haptic feedback on the virtual textures feels more like resistance, whereas the haptic feedback on the real textures feels more like vibrations. This makes it hard to compare • The textures feel distinctly different, but still not that different
F 23	• The haptic feedback is very low • The different weight of the pen and the brushes makes it harder to compare the haptic feedback
M 26	• The haptic feedback is very low • I feel like it would be better if the haptics were stronger, even if this would make the feedback more distant from the real haptic feedback

Table D.1.: Caption