TÉCNICO LISBOA

Development of a Multi-Platform Whiteboard Application

Lucas Emanuel Figueiredo Soares

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. João Fernando Peixoto Ferreira
Prof. Alexandra Sofia Ferreira Mendes

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. João Fernando Peixoto Ferreira
Member of the Committee: Prof. Pedro Henrique e Figueiredo Quaresma de Almeida

October 2021

Acknowledgments

First, I would like to thank God for always being with me. I know I can always count on Him. I would like to thank my parents Paula and Emanuel for their unconditional love and support, for the encouragement and caring over all these years, for always being there for me, and for always pushing me a step further when I most needed it. Without them, this Master's degree would not be possible. I am forever grateful for everything you have done for me. I love you both. I want to thank my girlfriend Renata for always being there for me. For the support, for the understanding, for the love, for the laughs. I love you. I want to thank my family for their love and support throughout all these years.

I would also like to acknowledge my dissertation supervisors Prof. João F. Ferreira and Prof. Alexandra Mendes for their insight, support, and sharing of knowledge that has made this Thesis possible.

To all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. A special mention to my best friends Jin Xin and João Bernardo. Thank you.

Last but not least, I want to thank me. I want to thank me for believing in me. I want to thank me for doing all this hard work. I want to thank me for having no days off. I want to thank me for never quitting. I want to thank me for always being a giver and trying to give more than I receive. I want to thank me for trying to do more right than wrong. I want to thank me for just being me at all times.

Abstract

The goal of this project is to develop an innovative whiteboard application supported in multi-platforms that aims to facilitate the presentation and manipulation of handwritten content. The project will use as a starting point the application Xournal++ Mobile [1]. The idea is to extend the editor with features that allow for reliable input and manipulation of handwritten content while supporting remote and self-learning. Given the recent shift to online/remote work and teaching, a whiteboard application that can be used regardless of the device choice can have a massive impact on note-taking and the overall learning experience. The idea is to make a solid whiteboard application to assist the development of the first mathematical structure editor using a cross-platform app software.

Keywords

handwritten mathematics; digital ink; structure editor; tablet PCs; pen-based input; mathematical sketching; mathematical expression recognition; gestures; STEM education; calculational method; educational technology; intelligent tools; sketch recognition

[1] Github repositary for Xournal++ Mobile, https://github.com/xournalpp/xournalpp_mobile

Resumo

O objetivo deste projeto é desenvolver uma aplicação inovadora de whiteboard com suporte em multiplataformas que visa facilitar a apresentação e manipulação de conteúdos manuscritos. O projeto utilizará como ponto de partida a aplicação Xournal++ Mobile [2]. A ideia é estender o editor com recursos que permitem um input confiável e manipulação de conteúdo escrito à mão, ao mesmo tempo que oferece suporte à aprendizagem remota e autoaprendizagem. Dada a recente mudança para trabalho e ensino online/remoto, uma aplicação de whiteboard que pode ser usada independentemente da escolha do dispositivo pode ter um impacto enorme para fazer anotações e também na experiência geral de aprendizagem. A ideia é fazer uma aplicação sólida de whiteboard para auxiliar no desenvolvimento do primeiro editor de estrutura matemática feita através de um software que permite utilização em multiplas plataformas.

Palavras Chave

matemática manuscrita; tinta digital; editor de estrutura; tablet PCs; input baseada em caneta; esboço matemático; reconhecimento de expressões matemáticas; gestos; Educação STEM; método de cálculo; tecnologia educacional; ferramentas inteligentes; reconhecimento de expressões

[2] Github repositary for Xournal++ Mobile, https://github.com/xournalpp/xournalpp_mobile

Contents

List of Figures

1

Introduction

Contents

We live in a digital era where technology is gradually making its way into every sector, and education is no exception. We are experiencing a technological revolution in the pedagogical area, progressively improving communication means and teaching methods. The launch of electronic presentations was a big mark in this process. It allows the instructor to previously prepare classroom presentations and present high-quality content with the benefit of saving time in class and avoiding human errors. Furthermore, it is thought that technology-based presentations can bring major positive changes in the pedagogical area and solve problems related to traditional lecture-based methods [7, 8]. Technology has also shown to be imperative during the COVID pandemic. Since multiple institutions were forced to shut down their installations, education transited to full remote mode, forcing the instructors that were still attached to old teaching methods to make the transition to technology-based ones.

Although electronic presentations are very reliable for most areas, they have significant downsides in areas that involve writing mathematical formulae and expressions such as in STEM education. The problem lies in the complications that involve writing and presenting mathematical content. Commonly used input devices like keyboards and mice do not support most mathematical symbols, involve high cognitive load, and are very time-consuming compared to paper and pen for problem-solving [9]. Besides, slide-based presentations work mainly as one-to-many communication, which limits interaction with the audience, and in STEM education, there's often the need to adapt content during class to comprehend students' level of understanding [10]. Blackboards solve this kind of drawback but imply that the presenter writes everything in presentation time, which consumes a high amount of time and can lead to errors.

A potential solution is to use pen-based devices to write and display mathematical content through digital ink, which has the potential to play an important role in the future of the classroom [11, 12]. Instructors can now write and present without the additional effort that common input devices add to the process while taking advantage of the benefits that technology offers. In addition to solving problems related to mathematical content, digital ink lets students express themselves in innumerous ways, offering instructors insight into the student thought process and generating interesting artifacts for discussion [1]. It lowers barriers so that more students feel comfortable participating in class and reduce the high cognitive load associated with common input methods. Digital ink also facilitates active learning, where students are directly engaged in the learning process [13].

However, because manipulating mathematical content involves a large number of syntactic manipulations, users often find themselves overloaded with tasks that could be optimized through computer software used in tablet PCs. For example, having to manually rewrite expressions for minor changes and using overly complex menus to trigger actions often lead to loss of thought, wasting of time, and

error making. For this reason, there is the need for an interface able to effectively and reliably input and manipulate mathematical content to improve the overall experience of writing and make the transition to technology-based devices as smooth as possible [6].

1.1 Work Objectives

The main goal of this thesis is to enrich the state of the art by developing the most advanced whiteboard application built with Flutter, supported in multi-platforms. The idea is to construct a solid starting point for the development of the first structure editor supported in multi-platforms.

Since the main objective is to bring improvements in the pedagogical area, instructors, students and researchers are the main beneficiaries. It not only brings advantages to lectures and in-class presentations but also brings significant benefits for personal use. Although this software was developed to make the most out of pen-based devices, users are free to use it on any kind of smartphone, tablet, or computer.

This application was developed to support and improve the presentation and manipulation of any written content through features that make the process more interactive and intuitive, less time-consuming, and prevent human errors. The system provides flexible and reliable tools that assist the user in writing and displaying content. By combining digital ink with the benefits of computer software, the idea is to improve the overall note-taking experience in multiple devices to build the starting point for the first structure editor using cross-platform software.

- Development of a cross-platform whiteboard application

It is discussed which cross-platform frameworks available are the best to use and some advantages and disadvantages of this kind of software compared with native development. Finally, since Flutter is the toolkit used, some other used toolkits are surveyed in section 2.

- Survey of the best cross-platform frameworks available
- Survey of already implemented software using the Flutter toolkit

1.2 Thesis Outline

This section presents a detailed breakdown of this document.

- Section 2 (Background and Related Work) explores and explains some of the concepts used in the project as well as ideas and state-of-the-art technologies similar to the project that were sources of inspiration.
- Section 3 (Development of the Whiteboard Application) describes the process of developing the project that intends to solve the problem presented above, the approach to take as well as the architecture behind it.
- Section 4 (Evaluation) describes the evaluation methodology used in the project
- Section 5 (Conclusion and Future Work) Sums up the project and the possible future work

2

Background and Related Work

Contents

Topics introduced in the previous section will now be explored in further detail. To better understand this work's objectives and thereafter the proposed solution, software that contributed to the state of the art will be analysed in order to build up a concrete set of ideas that will be helpful to the development of this project.

2.1 Pen-based devices and Digital Ink

Instructors are increasingly relying on digitally projected slides rather than using blackboards and whiteboards to write and display content. It allows the preparation of high-quality content in advance without requiring the instructor to write everything during the presentation. While allowing easy sharing and reuse of materials, it also facilitates distance learning.

This kind of tool lacks the flexibility to adjust the materials in lecture time so the natural response is to integrate digital ink, giving instructors the flexibility they need to adjust previously prepared materials [14].

2.1.1 Overview

Digital ink refers to the technology that represents handwritten content in a digital kind of way. The majority of these systems use some kind of pen, stylus, or even the user's finger over a digitizer laid under an LCD screen to record what is being written. The effect depends on the system but in the pen and stylus case, it normally resembles to writing on paper with a pen. Systems using these technologies support note taking and sharing, real-time distributed conversation and meetings, and classroom presentation and capture. Ink can change colors, be moved and resized, be transformed into standardized text, among other things. Inking systems can record time, pressure, context, and other types of information for every stroke drawn [14]. It can then be saved as handwritten content or converted through handwriting recognition technology to standardized the text.

2.1.2 Role in education

Digital Ink systems are becoming more popular over the years across a wide variety of domains. With its many advantages, these have the potential to play a big role in the future of education software [15]. One key ability that makes digital ink beneficial for learning is its potential to support intelligent interaction and visualization features [15].

Technology is being used to facilitate learning experiences inside and outside the classroom [16]. Without depending on direct communication with an instructor, the technology used in this way needs to provide appropriate instructions for a student to operate freely without additional effort. By encouraging students to engage with the content, technology may promote active learning while the system

demonstrates the effects of their changes. To better understand a concept, tutoring experiences guide students through a series of interactive activities. Because the majority of these tools are WIMP-based (Windows, Icons, Menus, Pointers), we end up with low fluidity levels and a slower learning rate, which affects the learning process and might distract the student from his task [17, 18].

Digital Ink allows the interactive tutoring experience without the complications associated with these kinds of tools. Unlike passive learning, where students are merely observers, in active learning students are directly engaged in the process. Because each activity is followed by feedback, students tend to understand and retain better what is being taught [19]. This statement is supported by the Constructivist Theory [20] which says that to get a deep understanding of a topic, it is very important for the learner to be actively engaged in the process. Constructivism could be supported outside the classroom with the growing use of software systems involving intelligent tutoring in education.

Without the active intervention of instructors, learners could still have high levels of interactivity, which aids in the student's independent work [21]. This would be extremely useful in a situation such as the COVID pandemic. Considering most of the classes have been transferred online, digital ink improves remote communication between students and teachers. While bringing the experience closer to a presential class, it brings additional support to the teachers as well.

2.2 Digital Ink in presentations

2.2.1 Microsoft PowerPoint

As previously mentioned, digital presentation software has gained prominence in the pedagogical area over the years. Beyond the many advantages it brings to the instructor, overall research indicates that students prefer PowerPoint type presentations rather than traditional ones [8]. There are many points on whether the use of PowerPoint presentations is beneficial for delivering content: it allows the pre-planning and organization of classroom material; the text is more legible; facilitates the editing and revision of content; material sharing is a lot easier [22–25]. Although it brings major advantages it also has some constraints: lectures tend to be less student-centered; classroom rhythm tends to accelerate; answering students' questions tend to be difficult; and students tend to give less feedback regarding what is being taught [26–28]. Much of these constraints are solved with digital ink integration in presentations, through tablet PCs as input.

Incorporating digital ink requires very few modifications on slide design. Many might think most of the annotations reside in complementary information like highlighting, circling, and underlighting but it is beneficial to leave additional room for writing during class and for including extra explanatory information (Fig. 1). While writing on slides during class-time, students tend to keep up better with the instructor's pace, as the writing shows on the screen in real-time. Students can actively participate in

what the teacher is writing down as notes since everything is being written in real-time. While working through problems, there is not the need to previously prepare a different slide with the solution. Also, leaving additional space can allow the annotation of different approaches to solving a particular problem in order to better understand the solution. As Johnson stated, “Using digital ink in combination with presentation technology is an excellent solution for real-time classroom activities that require input during class, especially when other teaching technologies (chalkboard, whiteboard, overhead projector, or document camera) are unavailable or switching to them is sufficiently inconvenient. Therefore, the digital inking technique described above can be used in any PowerPoint lecture to involve the students in the development of their knowledge.” [29].

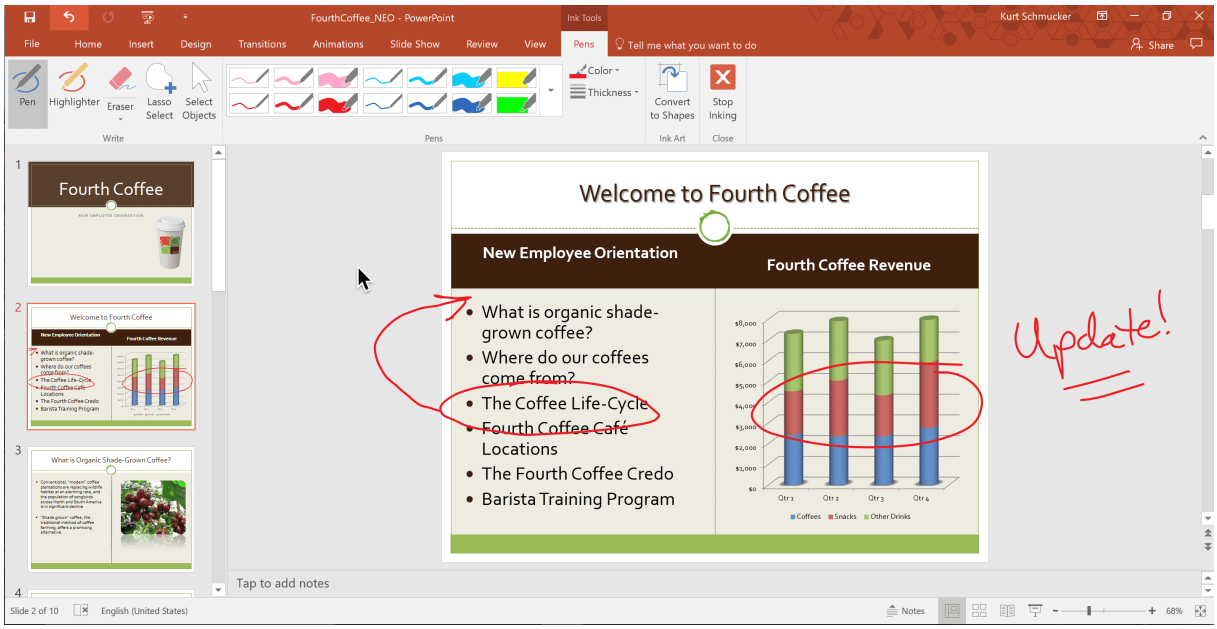

Figure 2.1: Digital Ink tools integration in PowerPoint presentations (Source: Three Tips for Reviewing Documents, https://www.parallels.com/blogs/reviewing-documents/)

2.2.2 Classroom Presenter

Classroom Presenter goes one step further from PowerPoint presentations by supporting in real-time sharing of digital ink slides between the instructor and the student. The framework incorporates the benefits of current computer-based systems with the reliability of the manual systems’ handwriting capability. The system runs over a pen-based device allowing the presenter to handwrite on projected slides. The materials are then broadcasted to other devices for students’ use or are displayed in a public display. Students can also contribute to the public display if the instructor finds it pertinent to use their work as an example, always maintaining anonymity among other colleagues. [30].

This distributed architecture provides a range of benefits to the system: enables flexible distribution of

material for lectures; lowers obstacles to having more students participating in class more comfortably; promotes peer learning by the integration of student work to debates in the classroom; increases the participation and understanding of students through student tasks in class; allows the instructor to be more conscious of whether a student is following up the lecture or not [1].

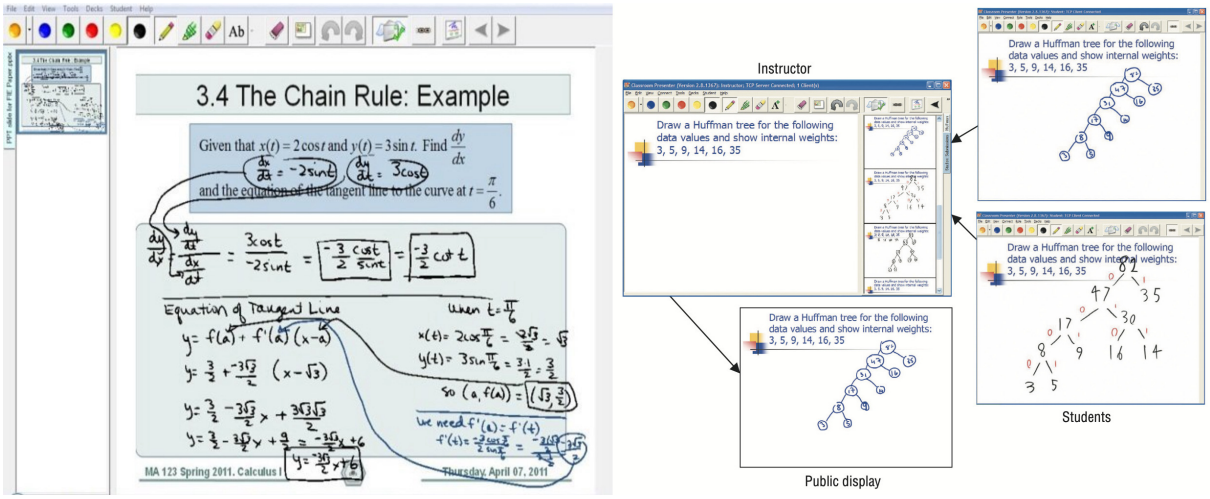

Figure 2.2: On the left: Digital Ink annotations using Classroom Presenter slides; On the right: Different screens of a Classroom Presenter presentation including one instructor view, one public view and two students views (Source: Tablet PC use in freshman mathematics classes promotes STEM retention, [1])

The interface differs depending on whether a student or a instructor is using it. There is also a public display where only slides are shown without any additional features. Classroom Presenter includes mechanisms to allow bidirectional sharing of data between instructor and students devices offering more flexibility in delivering classroom material by improving presentation tools. Dynamicity is provided by allowing the real-time annotation of slides to highlight or to clarify a point. It allows to get feedback on whether a student has understood a certain problem or not and depending on those results the instructor can opt to change the course of his initial approach if some subject is not being clear for someone. Since students' screens are shared anonymously it allows for discussion without embarrassing students who got the wrong result or followed the wrong path. As Anderson et al. stated, "This process causes the instructor to focus on developing activities to promote specific learning goals as opposed to merely covering material" [1]. Displaying student work in the public display is a strong motivator for students to contribute even more as it puts student efforts on the same level as the instructor's content.

This tool was deployed in classrooms in order to get feedback both from the students and the instructors. Classroom Presenter was tested in some computer science courses taught by different instructors. Users manifested enthusiasm about using the application on their courses. Most of the students inquired considered that it increased their attention to the lecture and would encourage its use in other classes. Instructors also thought it was not distracting at all as it improved their student's learning experience. [14].

Although the highlight feature was anticipated to be used by the instructors to draw attention to the

slides, this feature has not received much use. Instead, users used other types of attention marks like underlining and circling around important content. This was caused by the extra effort to switch to highlighting mode due to the many customizable features included. It was also shown that instructors do not tend to waste much time switching pen color. The erasing methods also brought some interesting results. Although erase by stroke is available, page erases were much more frequent. They were used mostly to clear annotation mistakes rather than used with the purpose to erase all the ink content. This happens because page erases only involve one click while stroke erase involves selecting that option, searching for the ink we want to erase, and actually erase it. All these results support the idea that instructors (and users in general) tend to minimize the use of features that involve too much effort for the purpose [14].

With these results, we can identify some good points that will help in the development of the whiteboard application:

- The best app design is the one that provides less cognitive load required to achieve the desired tasks.
- It is important to understand users' needs and desires and which information is important to be included in the system.
- Gestures are a great improvement for task performance and trying to resemble them to commonly used gestures while writing and annotating on regular paper/blackboard would drastically reduce the cognitive load needed.

(Some other interesting considerations would be saving ink's direction while being written or maybe ink changing color to simulate temporal aging of strokes. (This would be a great idea for an editor capable of recognizing the Mandarin language as it matters the pressure, stroke direction, and order of strokes for its handwriting)

2.3 Software for handwritten mathematics

In this section, some tools capable of dealing with handwritten mathematical content will be analysed.

2.3.1 Math Boxes

Math Boxes is a relatively recent user interface based on digital ink designed to ease the process of writing complex mathematical expressions by hand. The system detects relationships including superscripts, subscripts, and fractions by displaying bounding boxes around subexpressions. According to their spatial relationship with nearby words, subexpressions are incorporated in a hierarchy of boxes. By

inserting new words directly into its defined boundaries, math boxes and their relatives are dynamically resized so that the corresponding subexpressions can be easily expanded. Structural recognition feedback is generated by the boxes themselves. By transforming individual characters so that they can be stored in a database, feedback on character recognition can also be provided. Figure 2.3 presents four examples of equations displayed using math boxes.

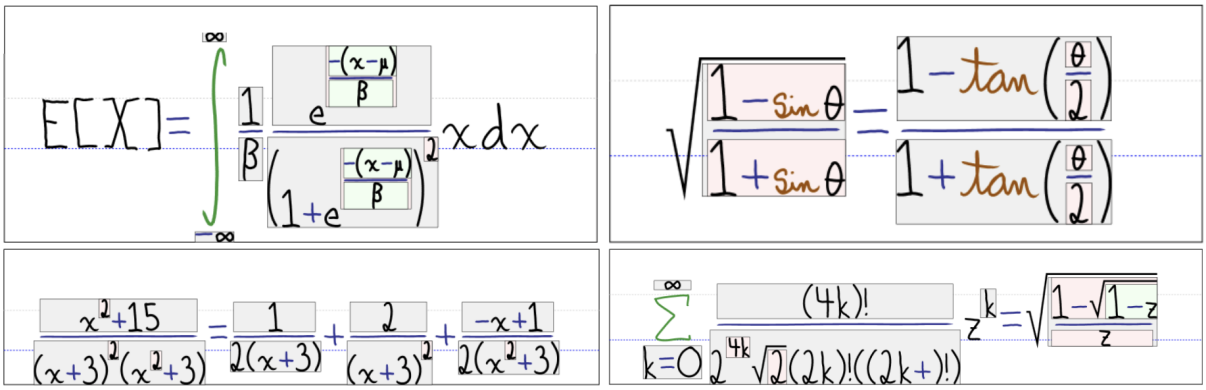

Figure 2.3: Concrete application of Math Boxes in 4 different equations. Every subexpression is contained in a hierarchy of boxes based on their relationship with adjacent subexpressions (Source: [2])

Without technical knowledge of the algorithms and architecture used by the mathematical parser, it can be difficult to identify and correct errors. Another problem happens while writing mathematical expressions. Even if we have the expression available in advance, in most cases we will not allocate enough space in advance to write the whole expression [2].

This user interface (UI) was developed to solve this kind of issues while writing mathematical expressions: lack of space related to writing long/difficult expressions and to make identifying and correcting errors simpler. To detect mathematical syntax, Math Boxes relies on simple structural parsing. Boxes are created once a particular relationship has been determined to exist between two symbols that do not share the same math baseline. These boxes are visible to the user and can easily be extended with a mouseover or by writing inside a box, allowing new symbols to have the necessary space. Based on a study regarding Math Boxes, the following conclusions were taken:

- Problems related to space while writing long expressions were solved. Error correction has been simplified. Many participants considered the UI helpful mainly with expressions containing several levels of subexpressions. It accomplished its goal.
- Users are more likely to use this tool for extensive/difficult equations as accuracy and time of writing improve with expressions' complexity.
- Users' experience and acceptance depends much on a well-designed UI.

2.3.2 MathBrush

People who perform complex mathematical problem-solving exercises often use pen and paper in environments like high-school classrooms, engineering companies, and university research laboratories. These people will turn to computer algebra systems (CAS) to solve computer problems that are either too boring or too difficult to solve by hand if pen and paper fail to help their problem-solving tasks [31]. Any mathematical program is called a computer algebra system if it has the ability to manipulate mathematical expressions in a way close to the conventional manual computations of mathematicians and scientists. However, an inconsistency exists between mathematical work performed on pen and paper and mathematical work performed in a CAS. The need to transcribe two-dimensional mathematical expressions into a one-dimensional sequential form can have a high impact on the interface [32].

Recognition of mathematics can be divided into offline systems that generate equations from scanned images and online systems that allow hand-drawn input and parse math expressions. Many existing offline methods have the storage of physical manuscripts as their objective in the offline domain [33]. Usually, the output of these systems is a typographical representation of math expressions, allowing digital storage of the original paper document and high-quality recreation on a computer screen or printer. Math Brush follows a different path as it focuses on the recognition of online mathematics, where the purpose is to allow a user to insert a math expression into a computer system [3].

The input and manipulation of mathematical expressions are normally the main concerns for pen-based math systems. This means that, for a system to be successful, handwritten content must be easily inserted and the recognition verification should require minimum effort. The use of gestures should also resemble natural gestures that people normally use while working with pen and paper to assure consistency among users [34].

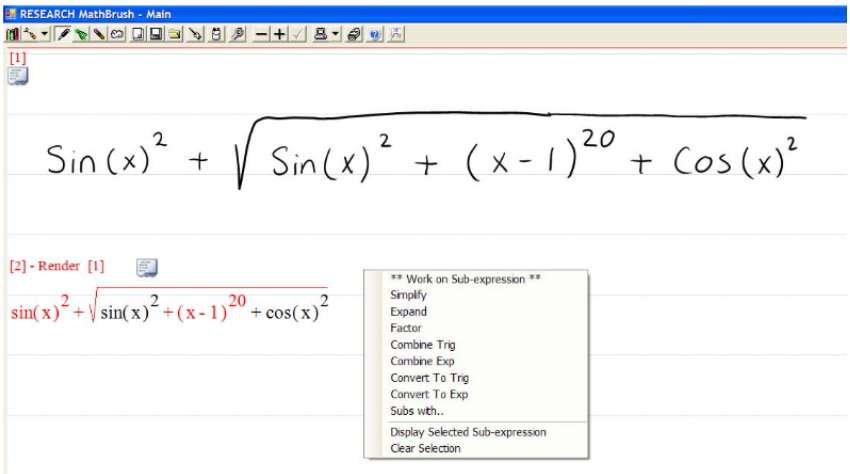

Figure 2.4: MathBrush interface after recognition of an expression (Source: MathBrush: A System for Doing Math on Pen-Based Devices)

MathBrush allows users to draw math input using a pen-input device on a tablet computer, recognizes the math expression, and then supports mathematical transformation and problem-solving (Fig. 2.4) [3]. It allows users to draw handwritten mathematical content and to correct eventual recognition mistakes. It uses recognition algorithms to support the transformation of handwritten content into MathML (Mathematical Markup Language) [35], which helps with the backend communication. It presents trainable recognizers that can be customized to users handwriting (Fig. 2.5) [3].

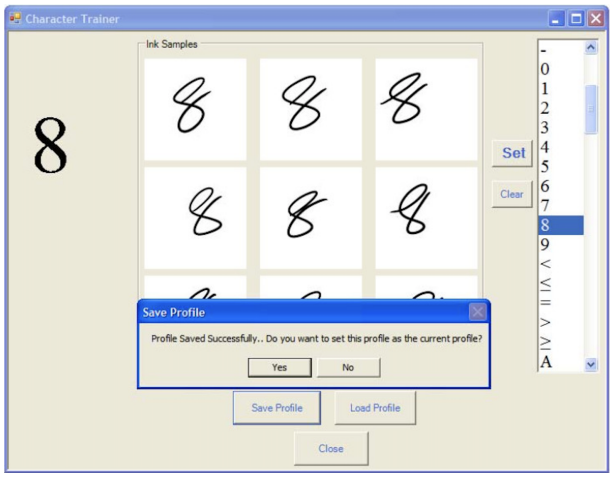

Figure 2.5: MathBrush character recognizer training (Source: [3])

For input, the user can write multiple expressions in the math sheet and has the ability to operate on one or more of such input expressions. The system provides a simple and easy way for users to verify the correctness of their handwritten expressions and, if needed, to correct any errors in recognition. After an ink expression has been entered the user can use context menus to recognize and manipulate the expression. Optionally, it is also possible that such recognition can be performed automatically after a pause or even after each stroke input. Choosing mathematical operations is done by making use of context menus, both with input and output expressions (Fig. 2.6). [34]

While developing an effective system for supporting mathematical tasks, MathBrush is also studying recognition technology and interface design to make the most effective support for their users. It includes several features that support the tuning of the interface to specific users, and the study of math recognition algorithms. Finally, to allow mathematical tasks common to both researchers and students, MathBrush supports the input and output of large expressions, and process logging to capture and archive problem-solving rationale (Fig. 2.7) [31].

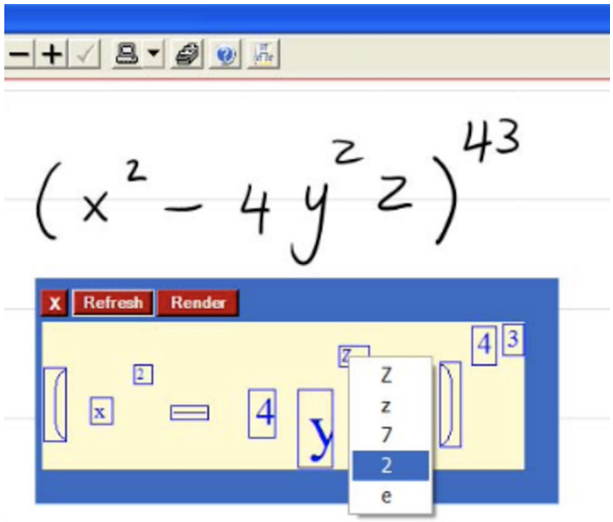

Figure 2.6: MathBrush character recognition (Source: [3])

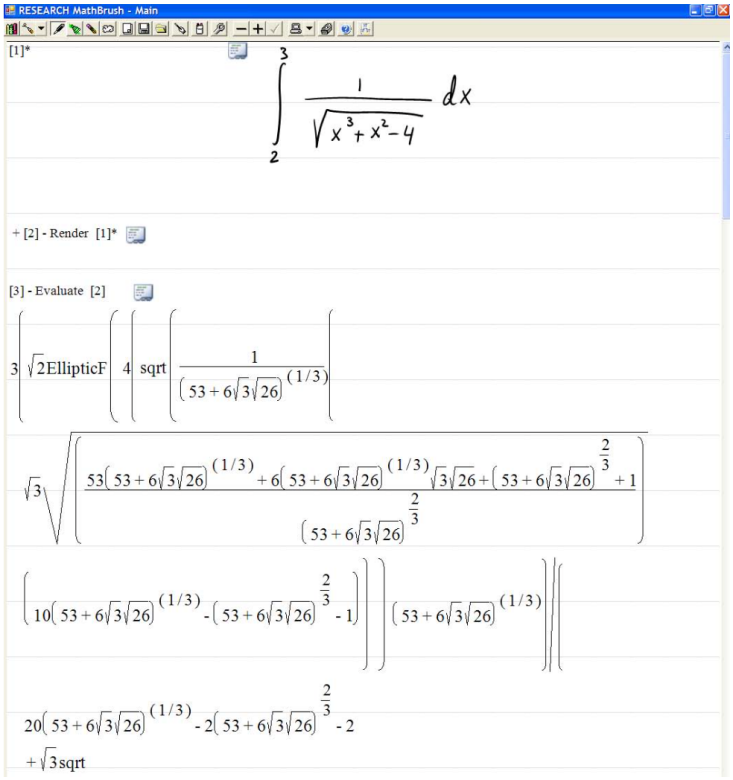

Figure 2.7: MathBrush dealing with long expressions (Source: [3])

2.3.3 MathPath^2

The principle of using computers to render dynamic illustrations of mathematical concepts has a strong background. Systems like Borning's ThingLab [36], Interactive Physics [37], and The Geometer's Sketchpad [38], were used to create dynamic models and illustrations in geometry and physics, although they did not support handwritten mathematics to create these illustrations. Since these systems rely on WIMP tools (Windows, Icons, Menus, Pointers), low fluidity levels and the necessity to switch between modes makes these interfaces difficult to use [18]. While users of these systems can perceive the dynamic behavior of their diagrams, it is difficult for them to gain a concrete understanding of the underlying mathematical phenomena because they are unable to write mathematics [4].

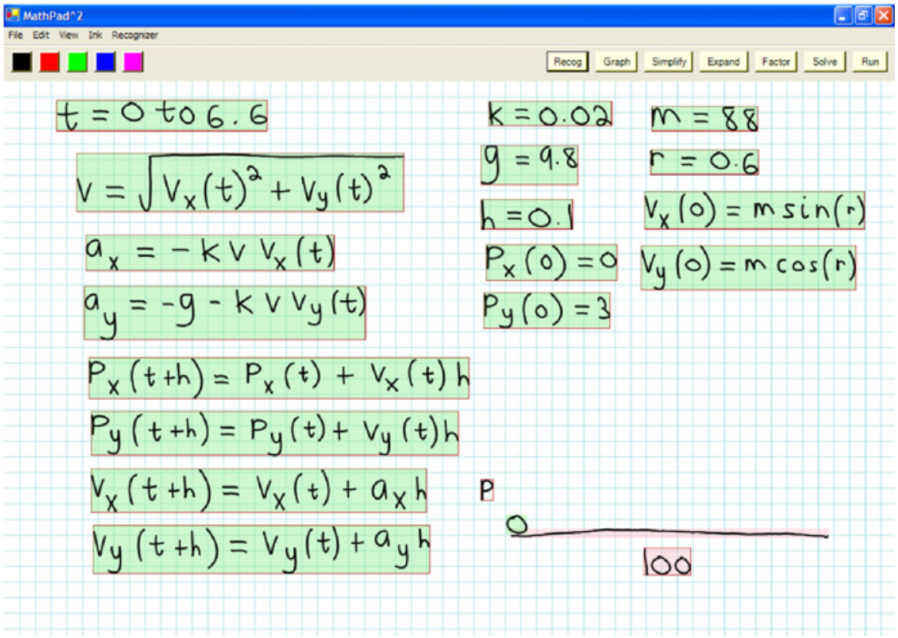

Figure 2.8: A mathematical sketch, created in MathPad^2 (Source: [4])

MathPad is a pen-based application prototype with the purpose of dealing with mathematic and physic concepts through the creation of dynamic illustrations (Fig. 2.8). It allows users to create interactive diagrams by combining handwritten mathematical content with sketches and offers a collection of tools for graphing and analyzing mathematical expressions and solving equations. MathPad relies on mathematical sketching for mathematical problem solving and supports diagrams to improve the process of formulating mathematical content. Because mathematical sketching uses handwritten mathematical expressions, users may use their knowledge of mathematical notation to create mathematical sketches. When users actually write mathematics, they achieve a higher understanding of the concepts represented and learn from their errors [4].

In order to sketch mathematical content, users combine mathematical expressions with drawing

elements by association. It is done implicitly by using diagram labels as input to an inferencing engine or manually using a gestural user interface [5]. For problems involving mathematics and physics, users usually use pen and paper for problem-solving so the idea of MathPad2 is to be as similar and as fluid as this method. Since all interactions are derived from using digital ink, a gestural user interface was developed to fulfill this idea (Fig. 2.9).

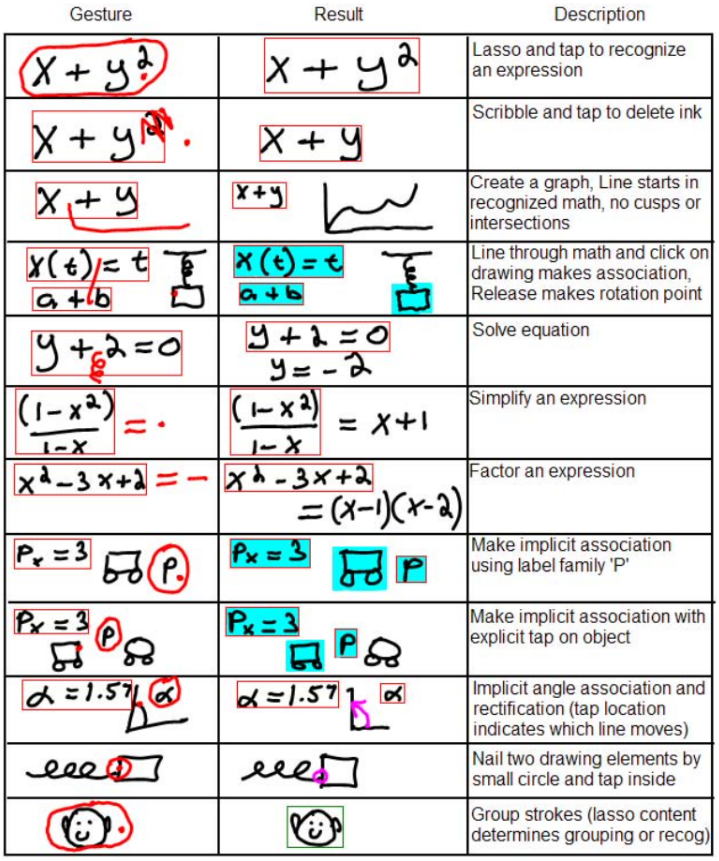

Figure 2.9: Gestures for interacting with MathPad2 (Source: [5])

To prevent gestures from interfering with the entry of drawings and equations, context-sensitivity was used to associate gestures with performed operations. The notion of punctuated gestures, single and multi-stroke compound gestures, and terminal punctuation were used to help to disambiguate gestures from mathematical expressions and drawings [39]. For mathematical expression recognition, a lasso selection over the expression followed by a tap inside must be done. Then, since the UI stores samples from the user, recognized material is presented to the user in his own handwriting. [4].

In math problems, students usually draw diagrams for assisting in the visualization of relationships among variables, constants, and functions. With the appropriate drawing, it makes it easier to solve the math problem. “By animating sketched diagrams from changes in associated mathematical expressions, users can evaluate different formulations with their physical intuitions about motion. They can sense mis-

matches between animated and expected behaviors and can often see that a formulation is incorrect and also make better educated guesses as to why." [5]. In addition to diagram association, MathPad provides users with a toolset on recognized mathematical expressions for graphing and evaluating functions as well as solving equations. Although rudimentary, this set of tools gives some representative early results on adding advanced features to the system. [5].

The results of a usability study show that MathPad is very intuitive. Most gestures are easy to remember and use with minimum training. It has the potential to be an important tool for dealing with mathematical and physical content inside and outside the classroom.

2.3.4 Mathpad Tablet

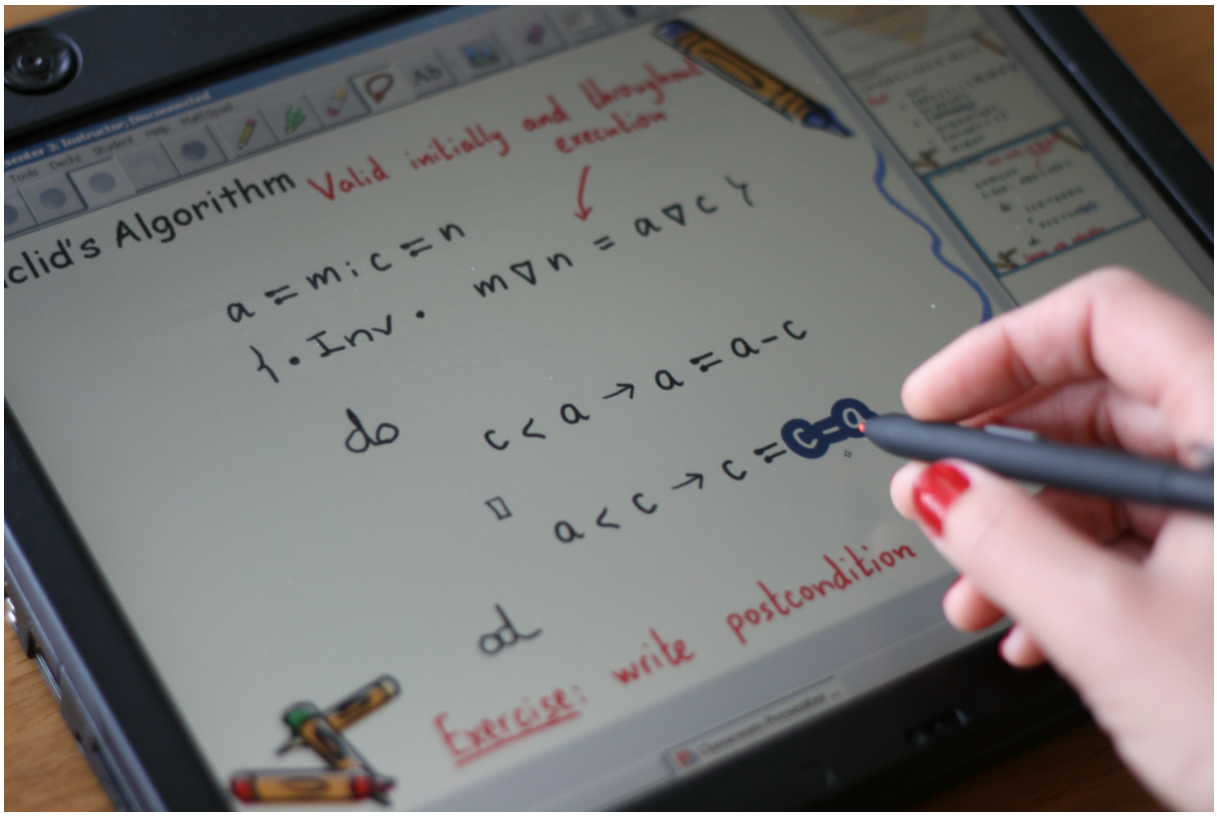

Figure 2.10: MST editor on a Tablet PC manipulating mathematical expressions (Source: [6])

Students and teachers frequently need to display and alter mathematical content as efficiently as possible, however the majority of regularly used technologies, such as computers, have substantial input constraints. Most mathematical symbols are not supported by keyboards, and mouse actions take a long time to complete due to the amount of clicks required. Although blackboards eliminate the limitations imposed by computer inputs, they require the instructor to write everything in class, which leads to mistakes and wasted time. Projected slides are a typical option, meaning that everything has

been planned ahead of time, which may be advantageous in some cases by saving time and avoiding typing errors. Because the input devices used to produce and control slides are quite difficult to modify while giving a lecture, they act more like a guideline. There is frequently a requirement for debate and engagement in presentations including mathematical subjects. [6, 40, 41]

The best approach is to introduce digital ink through pen-based devices to present and display mathematics, although this does not exploit its full potential [6, 11–13]. As Mendes et al. stated, "Tablet PC's computational capabilities can be a valuable help to write calculations and to show the dynamics of the symbols (that the calculational method stresses so much). Even if the presenter does not want to emphasize syntactic manipulations, having, for example, a feature that allows the reliable copy of expressions is a huge improvement." [6, 40, 41].

Mathpad Tablet was created in the context of education and research by merging the benefits of computers with pen-based devices. It includes a structural editor to help with the entry and modification of structured handwritten information, with the goal of improving the display of mathematical content. It also has capabilities that help to make presentations more flexible and interactive [6, 40, 41].

The editor supports in mathematics manipulation and ensures that human errors are less likely to occur to improve user engagement with handwritten calculation proofs. The MST editor allows for runtime definition of operators and redefining of their precedences, as well as user-defined handwritten character representation and problem correction in recognition results (also allows the change of the recognizer). Because the user is likely to add handwritten text and mathematical expressions, everything is kept handwritten to minimize misunderstanding for the user and, in the case of presentations, the students. In some cases, the animation of particular structural alterations may be viewed in real-time to demonstrate how expressions are modified from one step to the next. It helps students comprehend how the manipulation rules operate since the consequence of applying the algebraic rule is not instantaneous.

The definition of a structure for handwritten mathematical material is the core characteristic of MST. It makes operations like selecting and copying mathematic expressions easier by enabling manipulation of expressions. Additionally, motions can be used to initiate various actions. It makes writing mathematics a continuous process by eliminating the need for menu selection. Gestures are adjustable to meet the demands of various users and aim to match the visual image that is commonly associated with its application. [6, 40, 41].

The feedback gathered about the idea and functionality of the MST editor appears to imply that the tools supplied are useable, appropriate for their purpose, and provide a meaningful contribution to mathematical problem teaching and learning.

2.4 Handwriting recognition

The idea is to use an existing recognizer, as the implementation of a new one would not contribute to the main idea of the project and would lead to an additional unnecessary workload with poorer results. By using an existing recognizer, we benefit from the specialized work already developed and remain with the possibility of changing the recognizer in the future, as the course of things will lead to a more complete, specialized and with better results recognizer. In this subsection, it will be briefly explain how the handwritten recognition process works and talk about some handwriting math expression recognition software in order to build up knowledge related with this kind of tools and to be able to make the most adequate decisions for the project. Before talking about character recognition itself, it will be giving a brief background about the subject, starting with Optical Character Recognition (OCR) and Intelligent Character Recognition (ICR) technology.

2.4.1 Optical and Intelligent Character Recognition

OCR is a commercial solution for extracting data from printed or written text in a scanned document or image file, then translating the text into a machine-readable format for data processing such as editing or searching 2.11. Its market is rapidly expanding, owing to the rising digitization of company operations to cut labor costs and save valuable man-hours. Although OCR has been deemed a solved problem, one of its important components, Handwritten Text Recognition (HTR), remains a difficult challenge to tackle. Converting or simply recognizing handwritten text to machine-readable form can be tricky due to the wide range of handwriting styles across people and the low quality of handwritten text compared to printed text. Nonetheless, it's a critical issue for a variety of industries, including healthcare, insurance, banking, and, in our case, education and research. Recent Deep Learning advances, such as the introduction of transformer architectures, have accelerated our progress in handwritten text recognition, which is described by the Intelligent Character Recognition (ICR) 2.11. The algorithms used to solve ICR require far more intelligence than those who are used to solve ordinary OCR, and that's why it deserves more attention [1].

A self-learning system known as Neural Network is used in most ICR software to automatically update the recognition database for new handwriting patterns. It expands the capabilities of scanning machines for document processing beyond printed character recognition (a feature of OCR) to include handwritten matter recognition. Because this technique involves recognizing handwriting, accuracy levels may vary. The high accuracy rates are obtained when reading handwriting in standardized formats. Several read engines are frequently utilized within the program to attain these high recognition rates, and each is granted elective voting rights to decide the real reading of characters. Engines built to read

[1] What is Optical Character Recognition?: OCR Technology, https://www.hyland.com/en/resources/terminology/data-capture/what-is-optical-character-recognition-ocr

numbers have higher elective rights in numeric fields, while engines geared to read handwritten letters have higher elective rights in alpha fields. Handwritten data can be automatically fed into a back-office system when utilized in conjunction with a bespoke interface hub, avoiding tiresome manual keying and being more accurate than traditional human data entry. [2].

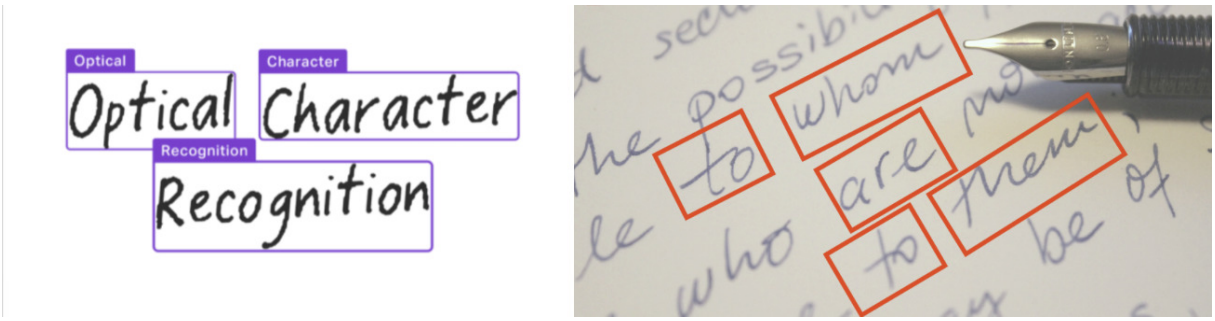

Figure 2.11: Optical Character Recognition vs Intelligent Character Recognition (Source: OCR that thrives in complexity, Digitize Handwriting With Intelligent Character Recognition)

2.4.2 Neural Networks and Convolutional Neural Networks

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. A neural network is a set of algorithms that attempts to recognize underlying relationships in a set of data using a method that mimics how the human brain works. Neural networks, in this context, refer to systems of neurons that can be organic or artificial in nature. Because neural networks can adapt to changing input, they can produce the best possible outcome without requiring the output criteria to be redesigned. The artificial intelligence-based notion of neural networks (ANN) is quickly gaining traction in the creation of trading systems [3].

In the final year of my Master's in the Machine Learning course, we used this kind of technology, more specifically, Convolutional Neural Networks (CNN), to be able to recognize handwritten symbols. Convolutional Neural Networks are a subclass of Neural Networks with at least one convolution layer. They are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. It will not be explain CNN's architecture with any more detail than what we see from the image because it would not have any impact on this project's context, as the goal is to use the technology as a starting point and not to detailly understand it. [4] 2.13.

[2]Handwritten Character Recognition, https://nanonets.com/blog/handwritten-character-recognition/
[3]Neural Network, https://www.investopedia.com/terms/n/neuralnetwork.asp
[4]Convolutional Neural Networks explained, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

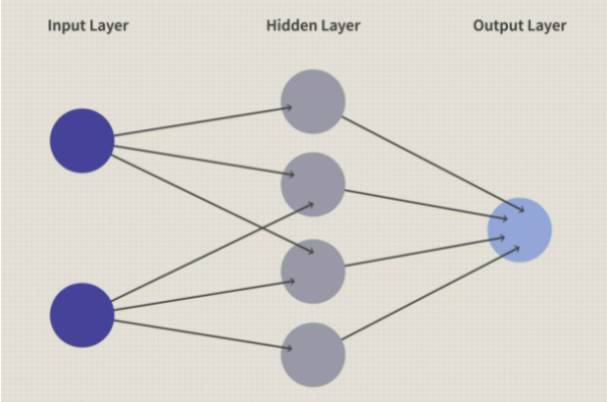

Figure 2.12: A simple Neural Network (Source: What is a Neural Network)

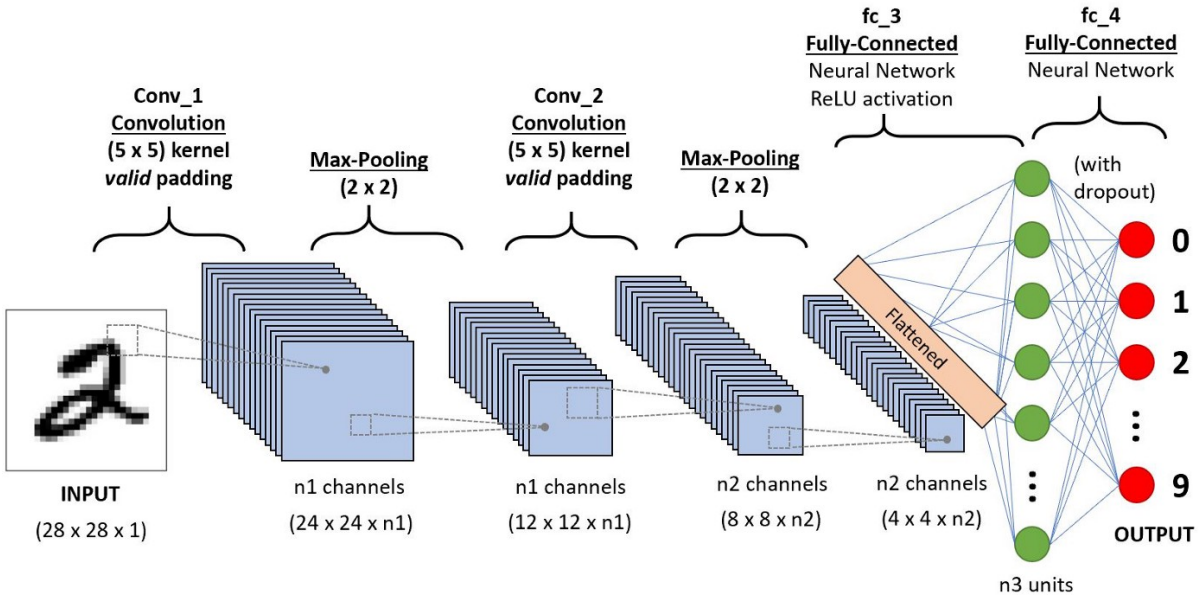

Figure 2.13: A CNN sequence to classify handwritten digits (Source: A CNN sequence to classify handwritten digits)

2.5 Cross-Platform Application Frameworks

We need a larger user base to generate more income. Furthermore, in today's quickly evolving technological world, we have the need for robust cross-platform app frameworks. The reason for this is simple: creating a cross-platform software allows our product to reach a wider audience at a lower cost [5]. Based on the objectives of the future mobile application, we may choose one of two development paths: create two or more native apps, or create a single cross-platform app that works on many devices at once. When we have a large potential, limited time, and a short budget, a cross-platform software is the appropriate choice. Another reason to create a cross-platform mobile app might be if we want a simple app with no complicated animations or functionality [6].

The need for cross-platform app development frameworks has skyrocketed. The main reason for this surge in demand is that cross-platform apps have a far greater reach than native apps. Cross platform applications enable the access to a larger number of individuals. Developers created customized frameworks to make the cross-platform app development task more efficient. Cross-platform app frameworks enable developers to create mobile apps with a single line of code and run them on many devices with few adjustments. There are numerous good cross-platform frameworks for mobile app development available nowadays that allows us to build high-quality apps.

2.5.1 Cross-platform technology options

We can build cross-platform apps using 3 different options:

- Web-based Apps

A web-based application is any software that is accessible via HTTP over a network connection rather than being stored in memory on a device. A web browser is frequently used to execute web-based applications. Web-based apps can also be client-based, in which a tiny portion of the software is downloaded to the user's desktop but processing is done on an external server through the internet [7].

Despite its drawbacks, web-based apps are a popular choice for cross-platform development. Web applications can be used on all mobile and desktop platforms, but they lack native mobile functionalities. For example, iOS system notifications will be missing, apps will be unable to be found in the app store, and performance will be poor. Apart from the aforementioned drawbacks, web technology is ideal for

[5] Best 10 cross-platform app frameworks to consider in 2021, https://www.thirdrocktechkno.com/blog/best-10-cross-platform-app-frameworks-to-consider-in-2021/

[6] Best cross-platform app development frameworks, https://messapps.com/allcategories/development/best-cross-platform-app-development-frameworks/

[7] Web Based Application, https://www.techopedia.com/definition/26002/web-based-application

developing apps that are constantly evolving. PWA (Progressive Web App) capabilities may also be integrated with the recently announced Web APIs, allowing developers to construct web apps that operate similarly to native apps [8].

- Hybrid Apps

Hybrid applications, as the name suggests, combine the advantages of native development with the benefits of online development. Hybrid applications, like native apps, are built using web technologies and operate on the device (HTML5, CSS and JavaScript). Hybrid apps run inside a native container and use the browser engine on the device to render HTML and handle JavaScript locally. A web-to-native abstraction layer gives you access to device features like the accelerometer, camera, and local storage that aren't available in Mobile Web apps [9].

This technology allows developers to use APIs to create online apps with native functionality. As a result, hybrid apps may be published on both the Google Play and Apple App Stores. The two main drawbacks are the lack of performance compared to native apps and the native feature limitations.

- Cross-Platform Native Apps

The process of developing an app that runs across many platforms is referred to as cross-platform development. This is accomplished utilizing tools such as Flutter, React Native, and Xamarin, and the resulting apps may be used on both Android and iOS. While cross-platform development saves time and money, it comes at the expense of quality. It's tough to create an app that works well on several platforms, and the program will require an additional abstraction layer to execute, resulting in inferior performance [10].

The frameworks used to construct these apps create an abstraction layer on top of native systems. It's worth mentioning that cross-platform native programming tools necessitate some amount of native development.

2.5.2 Pros and Cons

Our solution can run on numerous mobile operating systems and is created in a single programming language thanks to cross-platform development. When an app's code is complete, it passes through a bridge that converts it to iOS or Android's native APIs. It helps us to create the system quickly; it saves us money since we only need one codebase to operate the app on many operating systems; it provides for reusable code; and it allows us to reach a huge portion of the mobile app market.

[8] Best cross-platform app development tools, https://blog.back4app.com/best-cross-platform-development-tools/#Best_Mobile_Cross-Platform_Development_Tools

[9] What is a hybrid mobile app, https://www.telerik.com/blogs/what-is-a-hybrid-mobile-app-

[10] Native vs Cross Platform app development, https://www.uptech.team/blog/native-vs-cross-platform-app-development

On the other hand, we have limited access to some native features and can encounter issues with user experience. If we don't want to compromise the user experience and want to take advantage of all of a phone's native features, we may go with native app development [11].

2.5.3 Tools for developing cross-platform apps

2.5.3.A React Native

React Native is a JavaScript framework that allows us to create real-time, natively rendered mobile apps for iOS and Android. It's built on React, Facebook's JavaScript toolkit for creating user interfaces, although it's designed for mobile devices rather than the web. To put it another way, web developers can now create native-looking mobile applications using a JavaScript framework they're already familiar with. Furthermore, because much of the code we create can be shared between platforms, React Native makes it simple to build for both Android and iOS at the same time [12].

React Native allows developers to construct a highly responsive UI design since it focuses on the user interface. React Native has a strong developer community since it is an open-source cross-platform app framework. As a result, developers can receive enough help during difficult moments in their app development project. React Native is a prominent cross-platform app development framework, so developers just have to code once. Apps may operate on a variety of mobile OS systems, including iOS and Android, with just one time development.

2.5.3.B Xamarin

Xamarin is a free, open-source cross-platform software framework, similar to React Native. It was created in 2011 as a stand-alone platform, but Microsoft bought it five years later. Xamarin is not like the majority of cross-platform frameworks. It is based on Mono, an open-source .NET platform implementation. The C# compiler, runtime, and core.NET libraries are all included in this implementation. The project's purpose is to enable C# applications to run on operating systems other than Windows, such as Unix, Mac OS, and others.

Xamarin is simple to use since it just requires developers to be familiar with.NET and C#. Developers may simply utilize diverse third-party codebases using Xamarin. This is due to the fact that it includes C++, Java, and Objective-C libraries directly. When working with Xamarin, developers see fewer runtime problems since it has compile-time verification. App developers and other contributions make up a sizable part of the community.

[11] Best cross platform mobile development tools in 2021, https://litslink.com/blog/best-cross-platform-mobile-development-tools-in-2021

[12] Learning React Native, https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

2.5.3.C Ionic

Ionic framework is perhaps the most impressive AngularJS-based cross-platform app development framework. It is a cross-platform open-source software framework created by Adam Bradley, Max Lynch, and Ben Sperry in 2013. Apache Cordova and AngularJS were the key components of Ionic's main version. Ionic framework mobile applications are hybrid HTML apps since they follow the Apache Cordova concepts. The unique feature of these applications is that they operate in a separate shell on your smartphone. WebView for Android and UIWebView for iOS, for example, are used to execute the apps.

2.5.3.D Flutter

Mobile users expect their apps to have beautiful designs, smooth animations, and great performance. To deliver on this, developers need to create new features fast without compromising on quality or performance. That's why Google built flutter in 2017. Despite being the youngest, Flutter cross-platform app frameworks is making its own distinct image in the present day market. Flutter is a mobile UI framework that provides a fast and expressive way for developers to build native apps for mobile (IOS and Android), for desktop (Mac and PC), and for the Web, using Dart as its main programming language. This is because apps and interfaces made in flutter are built from a single codebase, compiled directly to native ARM code, use the GPU, and can access platform APIs and services. In 2015, Facebook had already launched its cross-platform mobile development framework (for multiple platforms), called React Native. However, since the launch of Flutter, many users prefer to use Google technology. Due to the user experience, development time, and performance, it becomes an alternative to React. Even so, React is a language that has been on the market for a long time and has a large online community. Over the next few years, we'll see fierce competition for popularity between the two frameworks, but it's safe to say that, overall, Google's tool is superior. Flutter is engineered for high developer velocity. Stateful hot reload allows the user to change his code and see it come to life in less than a second without losing the state of the app. Flutter also ships with a rich set of customizable widgets, all build from Google's reactive framework. Flutter moves the widgets, rendering, animation, and gestures into the framework to give the user complete control over every pixel on the screen. Flutter apps follow platform conventions and interface details such as scrolling, navigation, icons, fonts, and more [13] [14].

2.5.3.E Flutter whiteboard applications

Because Dart is not a native mobile language, it is hard to find such a specific type of content as programmers usually use Flutter to build more simple UIs. In terms of the whiteboard applications there is not much to show and comment on. All these applications share more or less the same features and

[13] Flutter. A revolução mobile da gigante Google, https://pplware.sapo.pt/internet/flutter-a-revolucao-mobile-da-gigante-google/
[14] Productively build apps, https://flutter.dev/?gclsrc=ds&gclsrc=ds

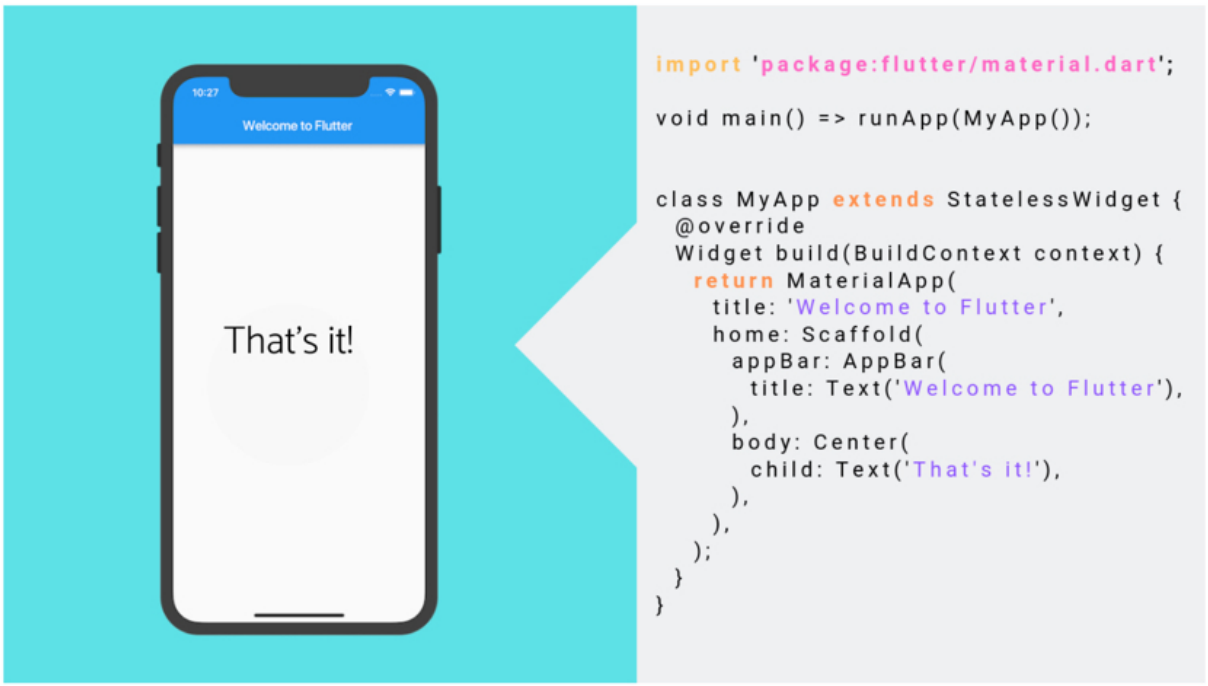

Figure 2.14: Basic Flutter interface coded in Dart (Source: Flutter. A revolução mobile da gigante Google)

overall organization. What is common to all is a pen tool, an eraser tool, and a tool that allows changing color. As we can see from the picture below, they all have really poor designs and very minimalist interfaces.

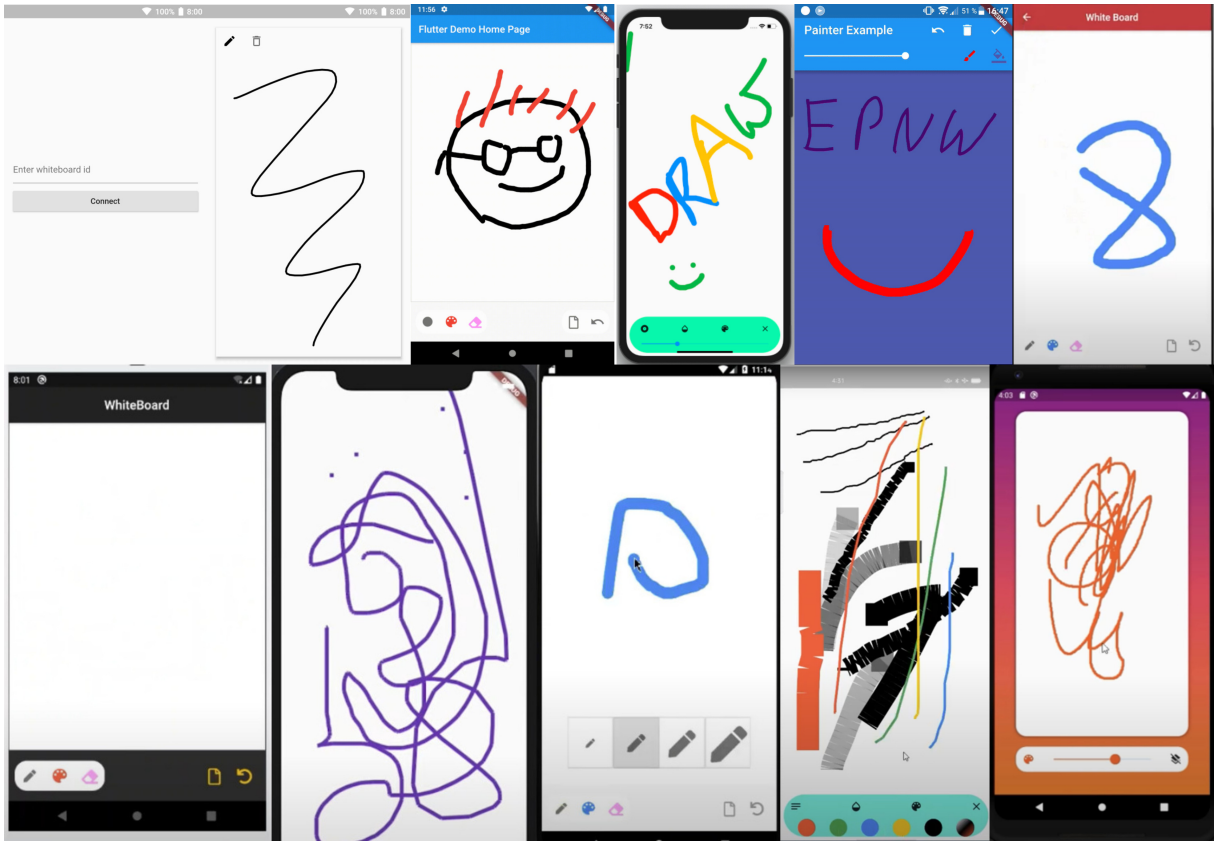

Figure 2.15: Whiteboard applications build with Flutter.

3

Development of the Whiteboard Application

Contents

3.1 Starting point choice

The idea was to find a functional whiteboard application with some key features already implemented and use it as starting point, as the goal of the project was not to develop a whiteboard application but to build upon it to allow structure editing of mathematical content.

Regarding the choosing of the starting point, it was very challenging to find an application with these characteristics with an open-source code. We ended up with two possible solutions: the XBoard [1] and the Xournal++ Mobile [2].

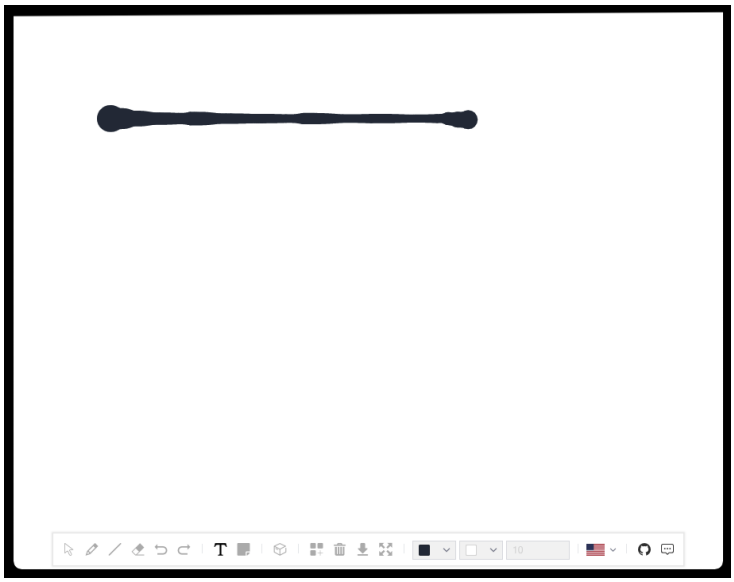

Figure 3.1: Screenshot of the XBoard whiteboard online application

Xboard, which is implemented in JavaScript, is an online whiteboard with a very minimalist yet simple interface with some tools on the bottom bar. Stood out a pen-like tool with the allowance to change color and width, an eraser, a tool that allowed to make straight lines, and a tool for text boxes. Although it filled the requirements regarding the initial features, it had lots of functionality problems, starting with the basic stroke, as we could not draw a straight line because the stroke would suffer a lot of width variations through the path, as seen in figure X. The background color feature was not the changing the color of the background but instead, the page would get filled with a solid color that would be noticed when using the eraser, as it would leave a white path underneath the area we were trying to erase, with no possibility to repair the erased area. Strokes themselves had other problems related to responsiveness. In the end, considering XBoard's code last update was on August 16 of 2019, was not worth using as it was barely usable and would take a lot of effort to fix the mentioned problems.

[1] Github repositary for XBoard, https://github.com/OXOYO/XBoard
[2] Xournal++ Mobile, https://github.com/sr-lab/xournalpp_mobile

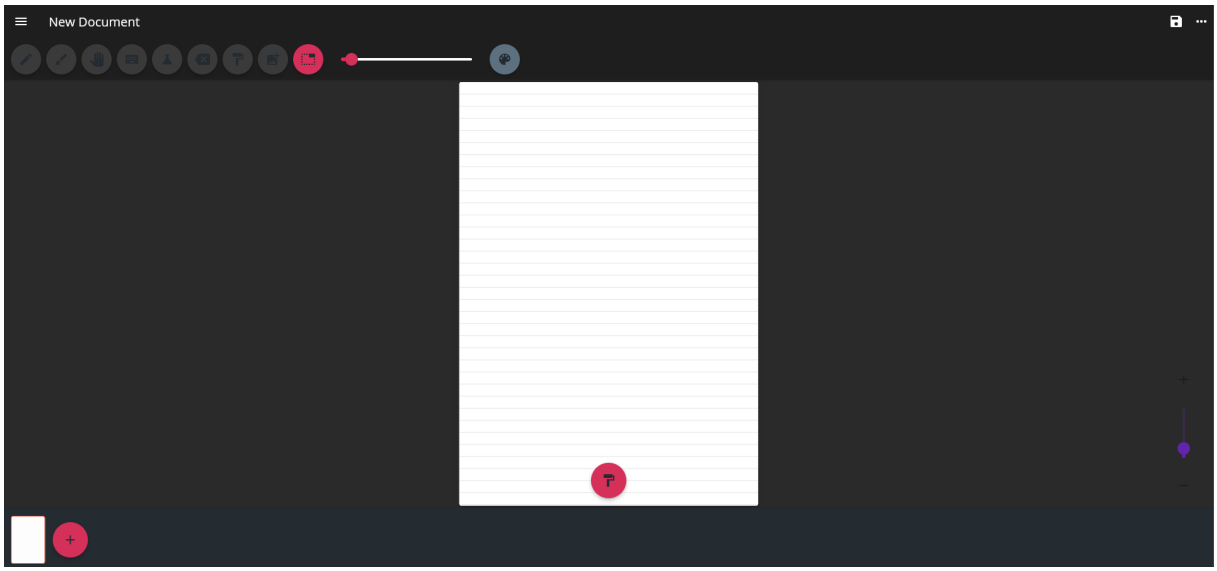

Figure 3.2: Screenshot of the Xournal++ Mobile whiteboard desktop application

Xournal++ Mobile [3], coded mainly in Dart, is a port of Xournal++ files and features to various platforms by using the Flutter toolkit. As previously mentioned, this app already won extra points for using Flutter, as it allows the user to have a single codebase and to deploy the app on a lot of different platforms. The overall design of the app is very clean, as a well-organized toolbar with a pen-like tool and an eraser already working, allows to change the page background pattern (none, solid color, lined, ruled, graph), supports multiple pages, and is possible to save the current state of the document. In the pictures bellow we can see the drawer (where we return to the home screen, open files and create new ones) and the bottom navigation bar (where we change the background aspect and create new pages) 3.3. Although the implemented tools were not completely reliable, through a usability test it seemed the right application for the starting point, as it did not share the same usability issues with XBoard and was constantly being updated (until that date). Xournal++ Mobile code's last update was on May 30, 2021.

It seems like none of these applications will have any further updates as the main ideas were already implemented and some serious drawbacks must have been shown, as they were left with a lot of "not implemented" features. Because we never went forward with the XBoard, we can only suppose that's what happened. Regarding Xournal, conclusions were already taken (which will be disclosed later on) as the last semester was spent understanding the app, the language, and its limitations.

[3]Github repositary for Xournal++ Mobile, https://github.com/xournalpp/xournalpp_mobile

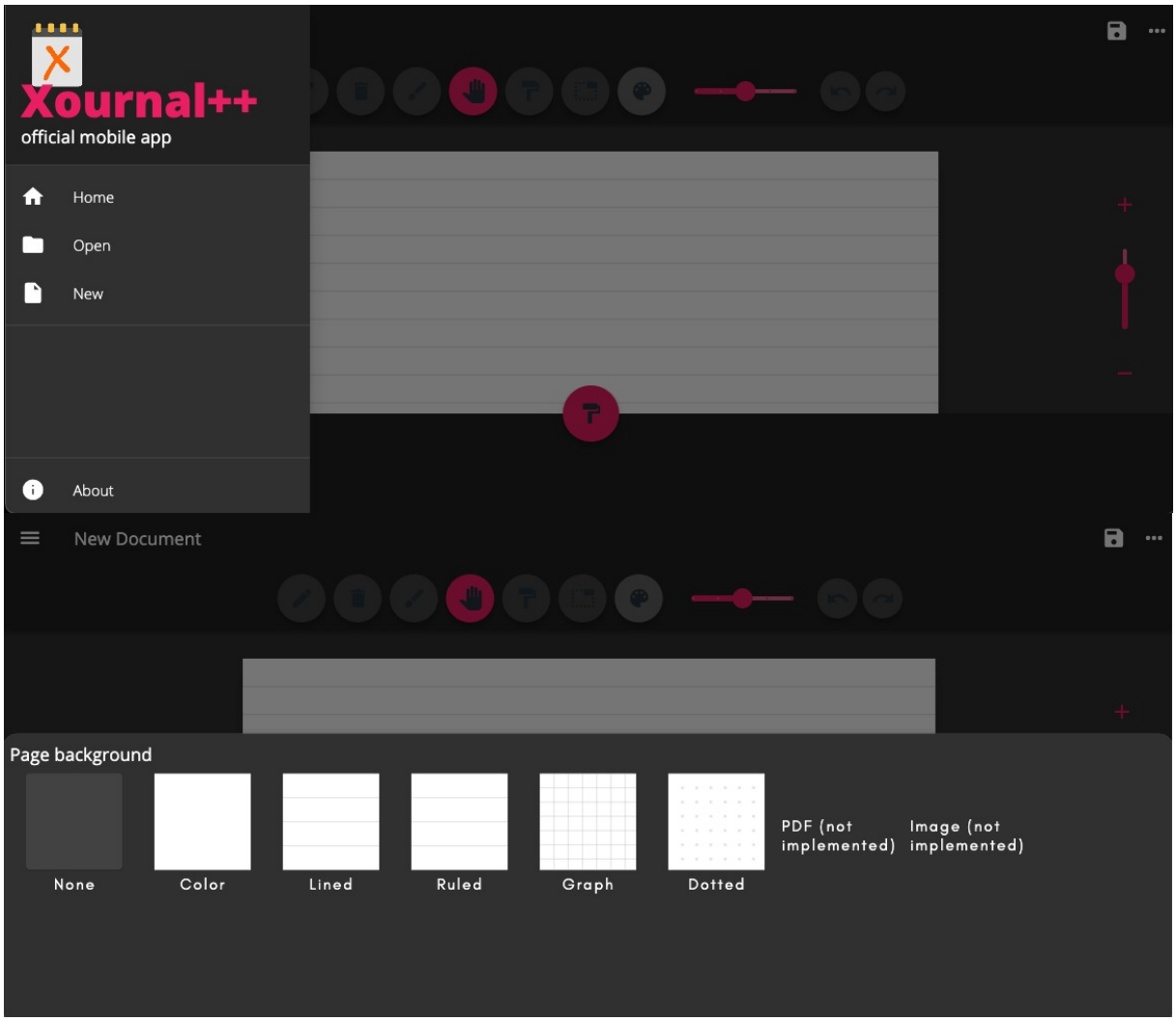

Figure 3.3: Drawer with Home, Open and New; BottomNavigationBar to change background

3.2 Flutter Widgets explained

In this sub-section, it will be mentioned some widgets that were used in the development of the project. This concepts are important to understand because they will be later used to explain all the decisions made during the development of the project, and, more concretely, where it stopped.

3.2.1 CustomPaint

Most widgets in the framework handle basic features and functionality, and even if they don't, we'll hardly need to design it ourselves because modifications like rounding corners, altering opacity and replacing colors are all simple to achieve. When it comes to more creative design, such as an arrow drawn from one location to another, a sophisticated loading animation, or a ticket receipt that resembles a ticket paper, we may want a solution to render shapes that ordinary widgets cannot give [4] [5].

Most if not all UI frameworks provide a Canvas API — a way for drawing custom graphics that does not involve existing UI components. A Canvas is pretty much what is sounds like: we can draw lines, shapes, curves, etc. using the Paint (color, stroke, ...) we prefer. This allows us to render custom components which can be exactly the shape and size we want them to be. In Flutter the CustomPaint widget provides a Canvas for us to use. We use the CustomPainter class to actually draw our graphics on the screen. The three main things to take a look at are: CustomPaint, CustomPainter, and the Canvas class.

CustomPaint is a widget that generates a canvas on which we can draw during the paint phase. When CustomPaint is requested to paint, it first asks its painter to paint on the current canvas. After it paints its child, the widget asks its foregroundPainter to paint.

The CustomPaint widget supplies the Canvas while the CustomPainter is where we write logic for painting. The painter and foregroundPainter are both CustomPainters which define what is drawn on the canvas. However, the instructions from the painter are drawn first. After that, the child is rendered over the background. And finally, the instructions from the foregroundPainter are drawn over the child.

```
1 CustomPaint(
2     child: childWidget(),
3     foregroundPainter: ForegroundPainter(),
4     painter: Painter(),
5 )
```

[4] A deep dive into CustomPaint in Flutter, https://medium.com/flutter-community/a-deep-dive-into-custompaint-in-flutter-47ab44e3f216

[5] Drawing shapes in flutter with CustomPaint and ShapeMaker, https://blog.logrocket.com/drawing-shapes-in-flutter-with-custompaint-and-shape-maker/

Now that we have a CustomPaint widget which will provide the Canvas, we need to actually paint something. To do this, we subclass the CustomPainter class to create our own.

```
class Painter extends CustomPainter {

  @override
  void paint(Canvas canvas, Size size) {}

  @override
  bool shouldRepaint(CustomPainter oldDelegate) {}

}
```

The paint() function supplies the actual Canvas instance which we can use to invoke the drawing functions. It also gives us the size of the canvas which is either the child size or supplied through the size parameter.

The Paint class stores attributes related to painting or drawing objects on the screen. Attributes like color, stroke width and style are all specified in here.

Finally, some common used methods in painting are drawRect, to create a virtual rectangle, drawPoints, to create single points in the screen, drawCircle, to create a circle, and so on. In our case, the most used method is called drawPath. drawPath is used for creating custom shapes by using lines and quadratic beziers 3.4 but we use it to create strokes [6].

Figure 3.4: most common use of drawPath in CustomPaint

[6] Flutter Custompaint tutorial. Draw a custom shape path in Flutter, https://medium.com/devmins/flutter-custom-paint-tutorial-draw-a-custom-shape-path-in-flutter-afbbf0202941

3.2.2 ClipRect

The ClipRect widget is used to clip its child using a rectangle. It is associated with the Clippers family. The main use of clippers is to clip out any portion of the widget as required. By default, ClipRect prevents its child from painting outside its bounds, but the size and location of the ClipRect can be customized using a custom clipper. ClipRect is commonly used with CustomPaint, Align and Center, which commonly paint outside their bounds. In the code bellow its presented a simple ClipRect wrapping an Align widget to show just the top half of an image.

```
1 ClipRect(
2   child: Align(
3     alignment: Alignment.topCenter,
4     heightFactor: 0.5,
5     child: Image.network(userAvatarUrl),
6   ),
7  )
```

3.2.3 ListView

ListView is the most commonly used scrolling widget. It is used to group several items in an array and display them in a scrollable list. It displays its children one after another in the scroll direction. The list can be scrolled vertically, horizontally, or displayed in a grid. This is how our toolbar is organized.

3.2.4 FloatingActionButton (FAB)

A floating action button is a circular icon button that hovers over content to promote a primary action in the application. Floating action buttons are most commonly used in the Scaffold. All the buttons on the toolbar are displayed using FABs.

3.2.5 Stack

A stack is a widget that positions its children relative to the edges of its box. This class is useful if we want to overlap several children in a simple way. For example, having some text and an image overlaid with a gradient and a button attached to the bottom. This not only allows brilliant custom designs but also some really cool animations. This widget is used to display the canvas and everything that is drawn in it.

3.2.6 Listener

A Listener is a widget that calls callbacks in response to common pointer events. It listens to events that can construct gestures, such as when the pointer is pressed, moved, then released or canceled. It does not listen to events that are exclusive to mouse, such as when the mouse enters, exits or hovers a region without pressing any buttons. For these events, we use MouseRegion. This is the widget used in the class PointerListener (explained in the end of the development) responsive for dealing with all the pointer events happening on the canvas.

3.3 Initial changes to the interface

It was decided that the original application layout and overall organization (with minor adjustments) should be maintained, as it was already well designed and well structured. In this section, it will be mentioned all the changes made to the original interface and explained the reason behind them. All the changes and new features implemented were taking in consideration that the application is designed mainly for pen-based devices, although it can be used on computers and smartphones as well. The image below 3.5 shows the final application interface. The application screen was zoomed so it would be more visible in the document. The application has the same aspect as the original, although it seems different.

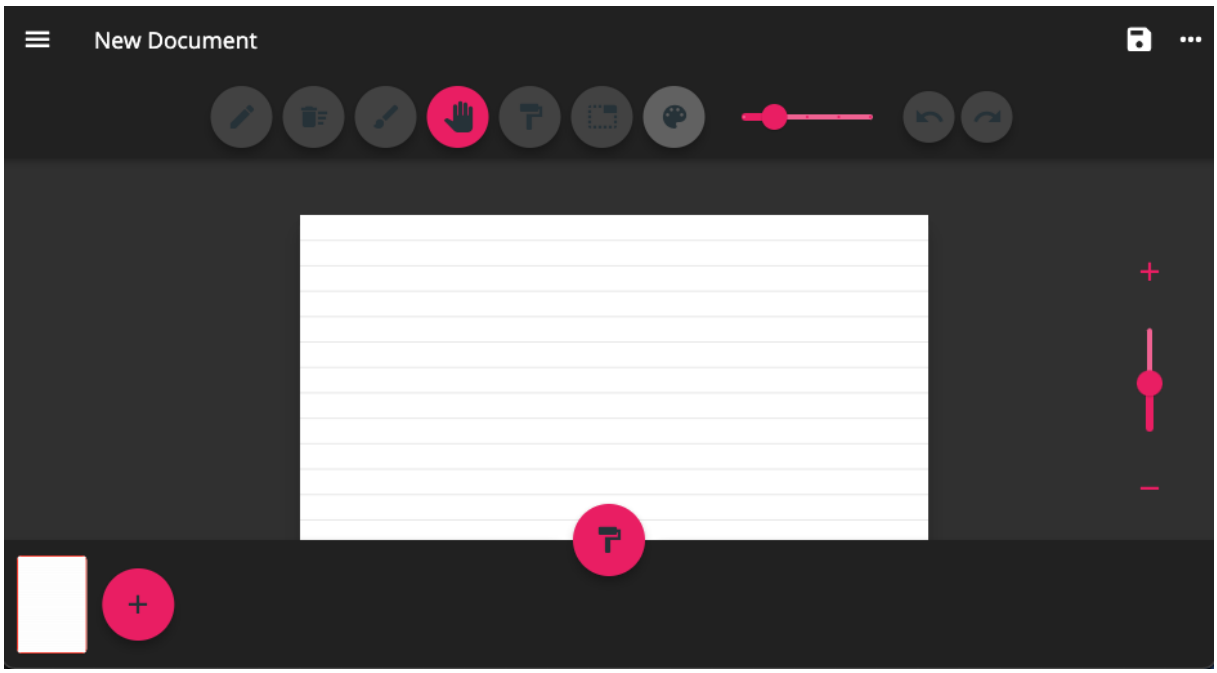

Figure 3.5: Final interface of the application

Staring with the toolbar. The toolbar was aligned at the left end of the bar, which seemed off from the

rest of the interface. A center alignment for design and functional purposes was implemented because it is easier for the user to change between tools, as they are much closer to the canvas and consequently pressed much faster. There were a lot of icons with not implemented features (Text, LaTeX, Whiteout eraser, Image, and Select) which were removed for being a waste of space and to avoid misguiding the user to click on icons without functionality. A much more clean, precise and easy to use toolbar was achieved.

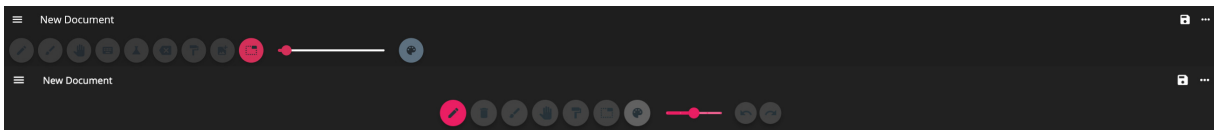

Figure 3.6: On top: old tool bar; on bottom: new tool bar

Regarding the stroke width bar, it makes no sense for the bar to have values from 0.1 to 30 because, in this range of values, we find overly thin and overly thick strokes that are just unpractical to use. Also, having a starting value of 2.5 on this scale, free width values through the bar, and no indication of the current stroke width value, makes it challenging or nearly impossible for the user to find the same width used before, leading to a huge width variety and inconsistency that turns the canvas exhausting and puzzling to read. The original width bar was replaced for a quantitative stroke width range (from 1 to 5, with 3 as a starting value) and a tooltip was added alongside a dynamic text box indicating the current width size. In this scenario, although the user has a much more limited range of choice, all the values available are plausible to use, the user has real-time knowledge regarding the exact stroke width he is choosing and is much easier to replicate a previously used stroke width.

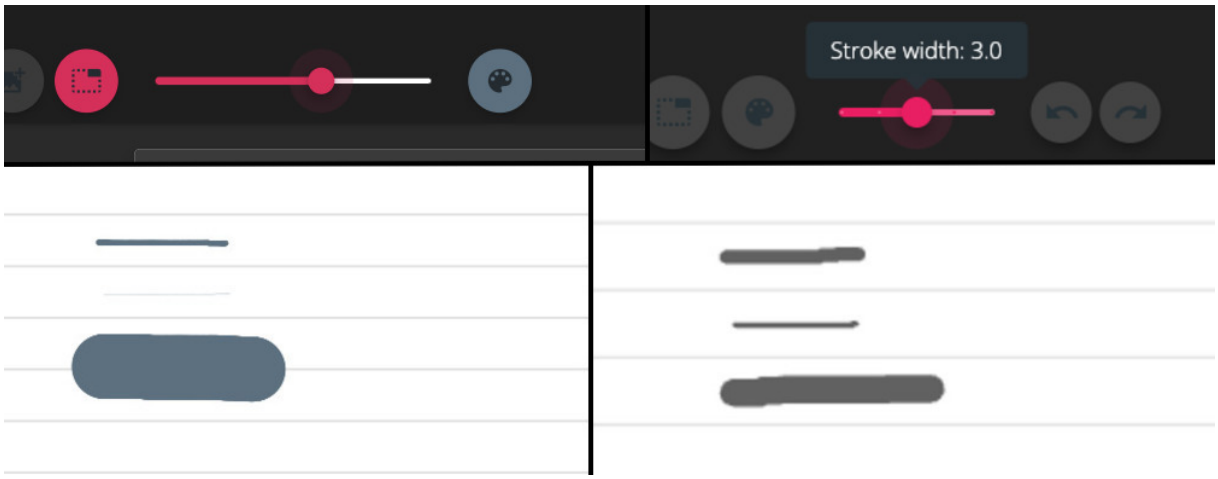

Figure 3.7: on the left: old stroke width bar and old stroke width; on the right: new stroke width bar and new stroke width

Zoom In/Out bar was standing out (in a negative way) from the rest of the application, as it was barely noticeable. The button and background colors of the bar were changed so it could match with the

interface design. Color to the ”+” and ”-” buttons was added to make some contrast with the background and a tooltip for both the plus and the minus buttons was added too. Tooltips can be seen on hover, in case the input device is a mouse, and on long press, in case the input device is a stylus pen or any other similar device. With these changes the zoom bar is way more visible and included in the overall app design.

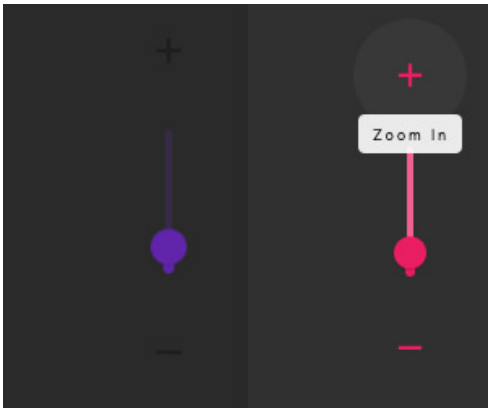

Figure 3.8: on the left: old Zoom In/Out bar; on the right: new Zoom In/Out bar

The way the stroke was being treated outside the canvas area was fixed. Initially, we could draw freely on the application and the stroke part that was drawn outside the canvas area would get cut off after the user finished the action . It was giving a false notion of canvas area as this update was not made in real-time. This problem was solved by wrapping the Stack (in which the CustomPaint responsible for the drawing was inserted in) with a ClipRect, which prevented the user from drawing outside the canvas. With this fix, writing outside the canvas area is no longer possible. For example, if the user initiates a stroke in the canvas area but tried to extend it to the outside, the stroke will simply stop showing on that part: If the user comes back to the canvas, the stroke will continue and will be dealt as the same stroke.

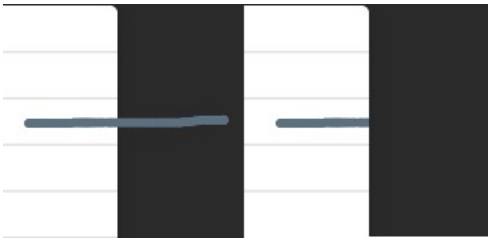

Figure 3.9: on the left: stroke before finishing the action ; on the right: stroke after update

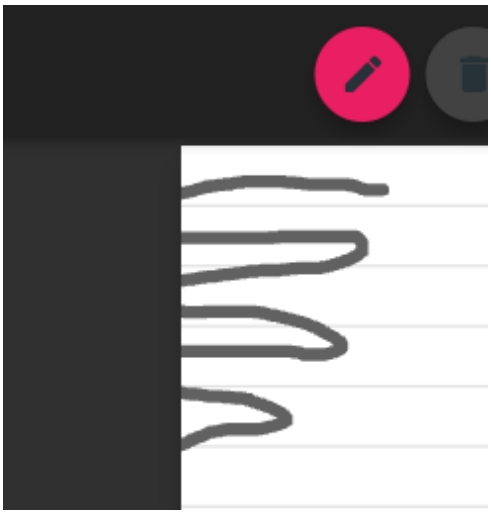

Figure 3.10: stroke constantly being drawn inside and outisde canvas area

3.4 New features implementation

In terms of what was already implemented in the project, there were some drawbacks regarding usability. In this section, those usability flaws will be mentioned and they were solve. It will also be explaining the new features that were implemented and an explanation for decisions made will be given.

3.4.1 Eraser

The eraser tool had some malfunctions associated. Although in the majority of times it worked properly, there were times where the stroke we were trying to erase would not get precisely cut apart (or would not get erased at all). The reason for what was happening could not be found, as it was not distinguishable if it was a consequence of a width variation malfunction or simply just a wrong implementation.

Because this eraser could not completely be relied on , another type of stroke deletion was implemented: the erase-by-stroke tool. The original eraser tool was kept as well as they both have different uses (but not as a default tool). With this implementation, the app ended up with two functional eraser tools. For more general uses, the default tool allows us to delete whole strokes/set of strokes. If we only want to partially delete a stroke, we use the second eraser. In the sequence below 3.11, we can see how both tools work and how we can change between them.

As soon as we hover (or long-press) on the eraser button, the tooltip ”Eraser by Stroke” immediately shows so that the user knows the tool he is using (and more specifically which kind of eraser). The eraser deletes the whole stroke (or multiple ones) as soon as the mouse pointer (or pen) collides with the content coordinates. Plus sign (+) was deleted on step 3. If we want to change between eraser tools,

we just need to click again on the eraser icon. As we can see in step 4, the icon and the tooltip changed to the corresponding eraser tool. Finally, in step 5, we can see the letter ”a” being partially deleted. The stroke is now handled as two different strokes and the rest of the content stored in the array is shifted to the right.

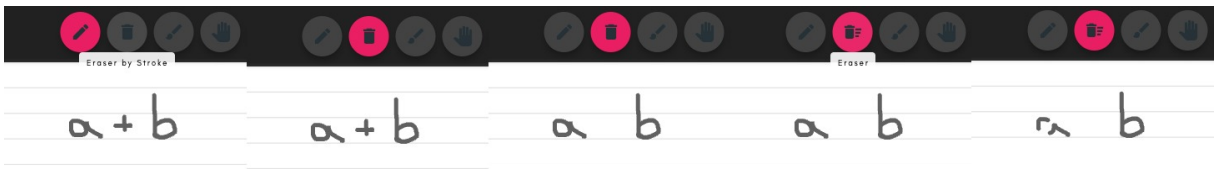

Figure 3.11: Example of how both delete methods work and how we can switch between them. Expression is written and Eraser button is hovered/long pressed. Eraser button is pressed; User clicks on the ”+” sign; Eraser button is pressed; User hovers on the canvas

3.4.2 Highlighter

After we used the highlighter, the new highlight stroke would have no opacity whatsoever, acting like a normal pen stroke with 5 times its width, which is the predefined width value for highlighter. It was discovered that the stroke would acquire highlighter properties (0.5 opacity) as we were deleting it. Anyhow, this tool wasn't usable, so it was re-implemented as we can see in the figure bellow. The highlighter can be useful when the user wants to emphasize some of his written content as the new stroke is see through. (Highlighter stroke is dealt as a normal stroke when performing the actions described above and bellow).

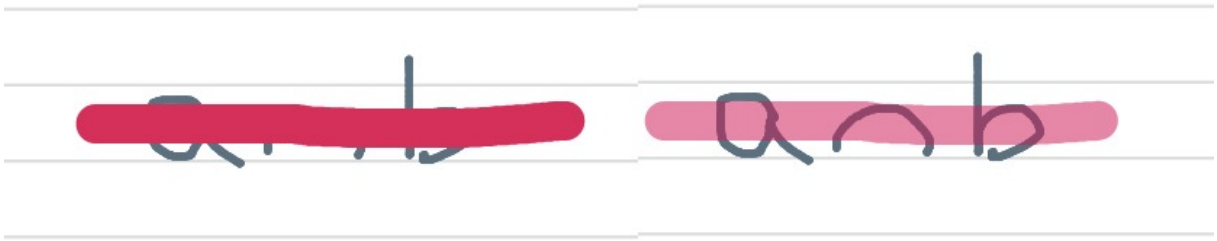

Figure 3.12: On the left: previously implemented highlighter; on the right: new implementation

3.4.3 Whiteout eraser

The whiteout eraser functionality was implemented, which is kind of self-explanatory. Similar to the previews tools, has a tooltip associated with the button to help the user always be sure of what kind of tool he is using. After the user selects the ”Whiteout eraser” button, as soon as a click on the canvas area happens, all the content is immediately erased. For preventing miss clicks, this functionality was purposefully not executed on button press but on canvas press, with immediate visual effect. If by any

means the user still miss-clicks and erases the whole page, it is possible to revert the action by clicking again in the Whiteout eraser Icon. All the content that has been deleted is stored until the user uses another tool. For example, if the user whiteouts everything and then selects the pen tool and writes new content, it is no longer possible for the old content to be recovered. In the original Xournal++ Mobile, the user could also delete everything that was written on the canvas by creating a new file. The problem with creating a new file is that it also restarts the color being using, the stroke width, the canvas position on the screen, the zoom, the page background color, and, in the case multiple pages are being used, all the remaining content.

The whiteout eraser tool allows us to keep all the previously mentioned settings and start on an empty canvas, with the possibility to revert the action (which is not possible if a new file is created). By being in the center of the screen in the toolbar, also facilitates performing this action. This tool can be handy when the user wants to delete everything that is written on the canvas and wants to keep the previously set up settings.

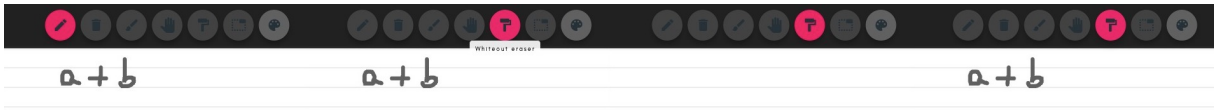

Figure 3.13: Sequence of the whiteout tool being used: Expression is written; Whiteout Eraser button is pressed; Canvas is pressed; Whiteout Eraser button is pressed again.

3.4.4 Undo and Redo

The Undo and the Redo buttons were implemented and, just like the whiteout, are self-explanatory. These two buttons (alongside the Color button) are the only buttons that do not permanently change color when pressed, as the change of color only occurs when a tool is selected. The Color button changes color after a new color is selected from the color pallet and only changes again when a different color is selected. When Undo/Redo buttons are pressed, they suffer from a fade in/fade out kind of animation, changing from their color to a more light one and changing back to the original color in a small matter of time, so the user knows the button as been used.

The Undo allows us to revert the last added stroke all the way back to an empty canvas and the Redo allows us to advance all the way forward to our starting point (before Redo is used). As soon as a new stroke is introduced to the canvas, all the previously deleted strokes are permanently deleted and are no longer available for recovery through the Redo. These tools work together with both eraser tools, sharing a similar behavior. If a stroke has been deleted by Eraser-by-Stroke, it is recoverable through the Redo button. If a stroke has been partially erased or cut apart leaving us with two strokes (through the other eraser), is possible to reintegrate the deleted part back into the original stroke. It was decided not

to include what had been deleted by the Whiteout Eraser for recovery because it was already included in the feature to undo that action in the Whiteout Eraser itself.

This feature can be useful if the user mistakenly erased some content he wants to recover or wants to remove what has just been drawn, without having to select it. The figure bellow shows 3 different scenarios of this functionality 3.14.

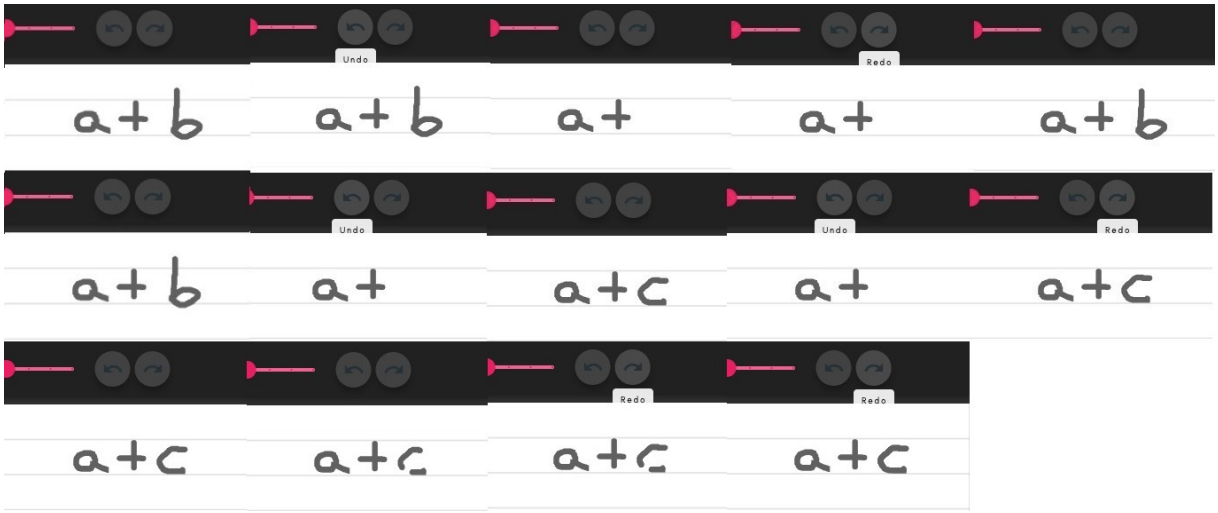

Figure 3.14: Sequence of the Undo/Redo tools being used.
First scenario: Expression is written; Undo button is hovered/long pressed; Undo button is pressed; Redo button is hovered/long pressed; Redo button is pressed.
Second scenario: Expression is written; Undo button is pressed; New stroke is drawn; Undo button is pressed; Redo button is pressed.
Third scenario: Expression is written; Eraser is used; Redo button is pressed; Redo button is pressed.

3.4.5 Gestures

Some features that are not as evident as the previously mentioned were also implemented, as they are not triggered by buttons but by gestures instead. For example, if the user is using the pen tool and wants to change to eraser mode, it is possible to switch between these two without actually having to press the eraser button. By simply double tapping the canvas on pen mode the eraser tool is immediately triggered. The inverse is also possible. If the user has the eraser mode activated and wants to switch back to drawing mode, a simple double tap on the canvas will do it. If any other tool is selected and a double tap action on the canvas is executed, the pen tool was chosen to be the default switch back.

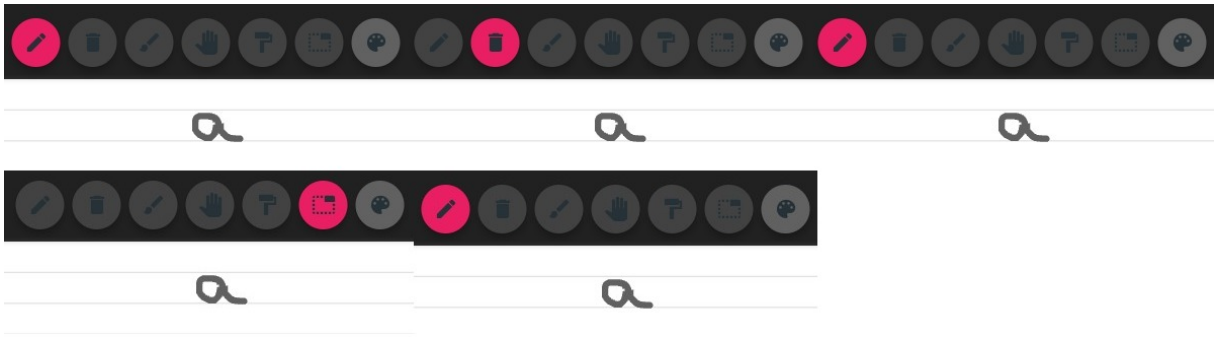

Figure 3.15: Sequence of actions triggered by double tap on canvas: Expression is written; User double taps on the canvas; User double taps on the canvas again; User switches to select mode; User double taps on the canvas

Similar to what happens on double-tap actions, long-press also has functionality. If the user is using the pen and wants to switch to highlighter mode, a long press on the canvas area will change between the tools and the inverse is also possible. In the case another tool is selected, the default switchback tool is also the pen. By adding these gestures to switch between tools, it takes from the user the additional effort to press on buttons, making the overall experience a bit more interesting and fluid.

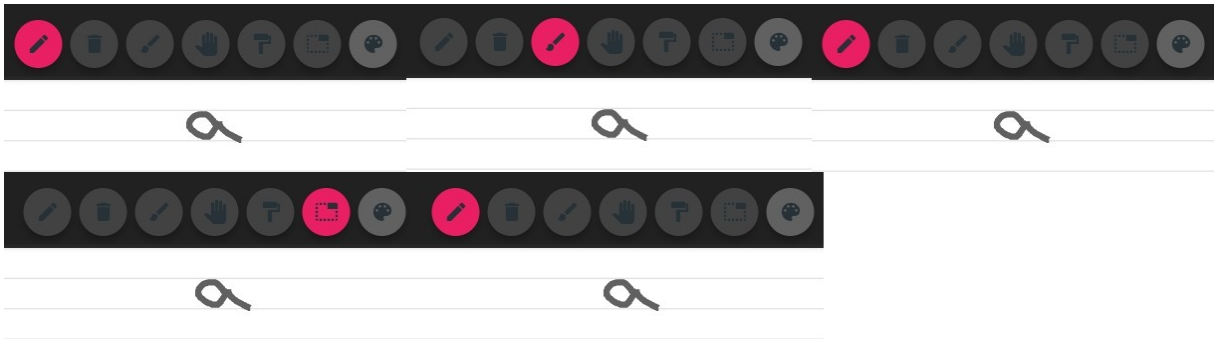

Figure 3.16: Sequence of actions triggered by long press on canvas: Expression is written; User long presses on the canvas; User long presses on the canvas again; User switches to select mode; User long presses on the canvas

3.4.6 Select

Initially, there were three ideas of how to implement the select function. The first idea consisted of a selection made by an adjustable rectangle or a free form where the user would press his pen/mouse against the canvas and make a form around what he wanted to be selected. The second idea was to simply press the content and it would become selected. The third idea, which was the one implemented, consists of a mix of these two. The content is selected by simply dragging on the canvas over what we want to select. The selection is made by a pen-like tool with the same width as the normal stroke, which can be changed in the toolbar. Initially, this stroke was resembling a highlighter (with low opacity levels) but it was decided to keep it fully invisible, as it seemed not to make a difference in functional matters and looked more aesthetic.

The select functionality is divided in 3 main steps:

3.4.6.A Detect selection

The first thing to do is to detect what is being selected. We need to check for collisions between the coordinates that are constantly being pressed against the collection of strokes that are already drawn and stored on the canvas.

After this step, the user has to have the knowledge of what is selected and what is not, so it is necessary to somehow be able to distinguish the selected strokes.

Initially, all the selected strokes' colors were changed to blue, but this solution would be problematic and confusing if the user opted for using another stroke color other than the default grey. As multiple strokes with different colors would be added to the canvas, it would not be possible to distinguish the original color until the content was deselected. The solution for this problem was to change the strokes color opacity so the original colors could be maintained while selecting and give the user a more clear perception of his content.

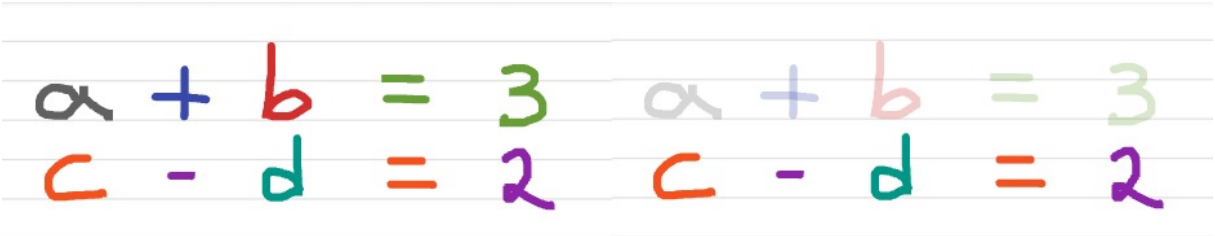

Figure 3.17: On the left: drawn expressions on the canvas; on the right: first expression is selected

Because the original strokes were already drawn and stored, their appearance could not be changed or modified in any sense. The solution was to access the array where strokes were being stored, erase those strokes and replace them for new strokes with the same characteristics but with lower opacity

values. This was only possible because previously drawn strokes were being used and only changing color values and not stroke points. If we wanted to change the stroke's appearance they would have to be redrawn.

3.4.6.B Isolate the selected content

Now that the selected strokes are already known, it was decided to make a rectangle around what is selected to better isolate the content from the rest. This rectangle is updated in real-time (redrawn with bigger dimensions), as the user is selecting new content. Five key points were added to make the presentation more aesthetic (top left, top right, bottom left, bottom right and middle).

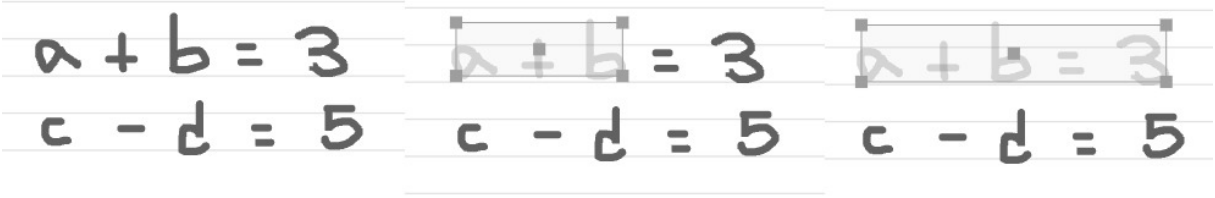

Figure 3.18: Sequence of an expression being selected

3.4.6.C Drag and drop content

The solution for dragging the selected content was to include a semi-transparent container over the rectangle area so it would be possible to wrap it around a Draggable widget and be able to drag and drop it. In this case, as the strokes are already drawn in the canvas, it is possible for deleting and replacing the strokes for new ones with new offsets.

To be able to select some content, the user presses the Select button and starts to press on the screen, just like he was drawing. The difference is that no stroke is shown and wherever the user presses the canvas, if it encounters a stroke, it will be selected. If the user made a bad selection or wants to select new content, he just needs to press the canvas outside the selected area and a new selecting will begin. It is also possible to restart the selection by pressing the selection button. If the content is selected and the user wants to delete it without having to manually delete every stroke, it is possible to do that by just simply dragging the content to the outside of the canvas area. If the user presses any functionality button it deselects the content. If the user no longer wants to use the select and wants to switch back to writing, the gestures on double tap and long press for returning to the pen tool also work.

The select functionality is described below in a sequence of prints. Firstly, the user presses the select

button. As he no longer wants the last expression, he selects and drags it to the outside of the canvas so he could easily delete it. The expression is immediately deleted as the user lets go of the screen. The user makes another selection and wants to keep a part of it. He drags the expression to the edge of the canvas as he knows everything that is outside the canvas is deleted. With the new selected expression, the user correctly places it where he wants and finally presses outside the selected area to deselect.

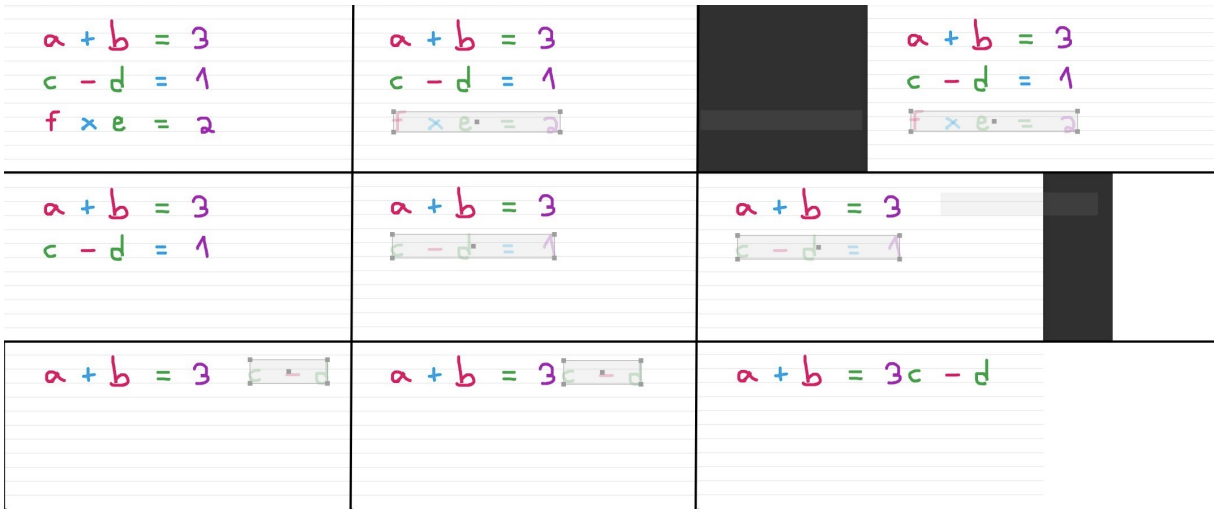

Figure 3.19: Sequence of the select tool being used: User presses the select button; selects expression; drags the expression to outside the canvas; makes new selection; drags the expression to the edge of the canvas; moves the selected content; clicks outisde the selected area

3.5 Problems while developing the application

3.5.1 Explanation

In this part it will be explained the most relevant problems that appeared while developing the app to be able to justify the decisions made and so it can help other programmers to develop over similar Flutter APIs.

While changing the toolbar and its tools, the idea for switching between the erasers was to implement an expandable FloatingActionButton as the one in the figure 3.20.

The button would expand through a smooth animation to reveal the additional buttons. The problem with this idea was not the implementation itself but the integration with the ListView widget. We use ListView to display the toolbar as it is one of the few widgets that allows to have a scrollable list and does not overflow when we shrink the app. If we tried to make the animation inside this widget it would simply get overflowed (although we are not moving in the Y axis) or get real messy in terms of the animation itself.

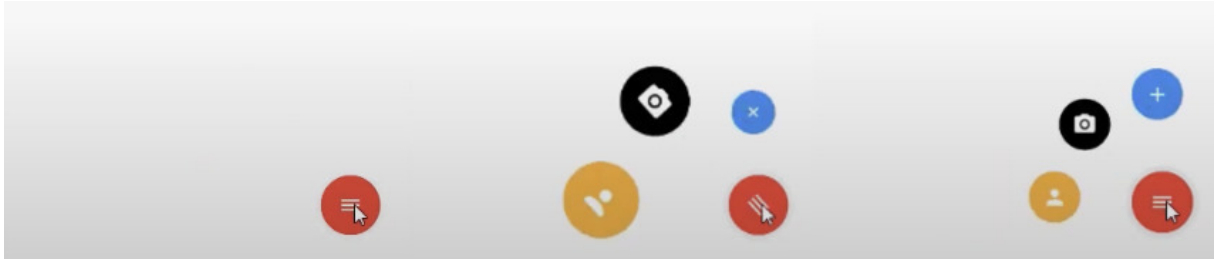

Figure 3.20: Expandable FloatingActionButton.

It was also tried to wrap this FAB into a Transform widget to manually change its coordinates but the onPressed method would be stuck in the original location of the button. It was tried to make a non-clicable container where the buttons would be expandable, but it would lead to two empty button slots because when we create those buttons, although there are hidden at first, flutter previously defines their location after expansion. It was tried to change the type of button but it would be very noticable that the button was different as they have different on click behaviors. Finally, the last approach before giving up on the expandable Fab was to make the button outside the ListView itself, but it would get us new problems with the ListView interaction 3.21.

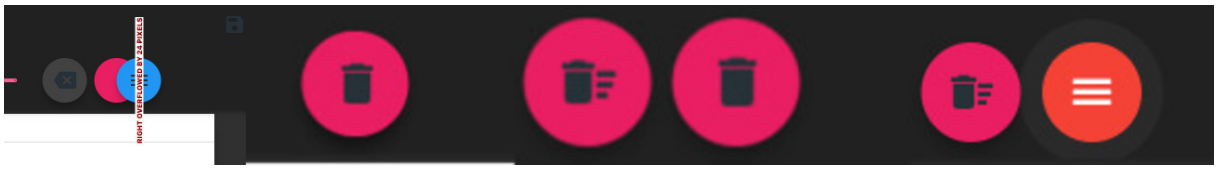

Figure 3.21: Overflow of the FAB inside the ListView; expandable Fab animation with strange behavior; expandable animation with another type of button.

Regarding the development of the first selection method proposed, more concretely, the rectangle selection. The idea was to make an expandable rectangle controlled by the user to be able to select content. In Flutter, in order to draw a rectangle or a custom shape, we have to define its parameters in the custompaint that is drawing the shape. Because we wanted to build the rectangle in real time and would not have access to its coordinates until the user finished the action, we had to discard this idea. Regarding the lasso selection, it would be doable for creating a pen-like tool to circle what we want to be selected. The problem relies on the fact that the content would only be selected after we finished the form, so it would not be seen in real-time. It would also be challenging to calculate the area inside this free form.

With respect to the second selection method proposed, the user only had to touch the characters he wanted to be selected. This is not possible in Flutter, as the CustomPaint refuses to respond to any kind of gestures. It was even tried to wrap the strokes with another gesture detection widgets (like

GestureDetector, InkWell and MouseClick) and to use external packages to deal with this situation but none of them fixed the problem. Also, this kind of selection would be more viable if the strokes were already being recognized and treated as a character, as it would be frustrating for the user to click on a stroke and the whole character not be selected.

In the Detect Selection step, the original idea was to give the selected content some kind of glow to distinguish what was selected from what was not, but this was not possible to implement in Flutter. It was then thought of decomposing the stroke in RGB components and max out one of those (for example, by setting the blue value to 255), but this solution was also not possible because it would simply change to a different color for every stroke and could not be recognized as selected.

In the isolation of the selected content part of the selection, the original idea with the five key points was different from just adding an aesthetic look. These 4 corners points were originally added to shrink and expand the selected content and the middle points to move that same content. As previously mentioned, it was not possible to find a way to detect gestures hover CustomPaint so this idea was discarded. Those points were left for the referred reason.

The dragging part was tricky. Because CustomPaint did not allow for gesture detection, another solution had to be found. An idea was to use a FloatingActionButton on the center of the rectangle so we could wrap it around a Draggable widget and move the selected content. The problem here is that, with any kind of button, Flutter assumes that any given coordinates are global. Because what is inside of the Stack where the canvas is inserted everything is dealt with local coordinates, it is not possible for this button to be centered. The button would always get deviated from the center of the rectangle 3.22.

Manually adjustment of the button position was approached but because its coordinates are defined globally, when we changed the app's zoom, it would be wrongly placed again. It was tried to convert these global coordinates to local but still had no luck. By making a transparent container in front of the rectangle and wrapping it around a GestureDetector, it was possible to convert the global coordinates into local ones through a RenderBox and make the drag and drop possible.

This solution for the dragging still suffers from some kind of global/local problems as we can see in the figure below 3.23 (in this case, both containers are created with the same dimensions but the one being dragged changes size while dragging). It was also tried for the strokes to move along the container while being dragged but the Draggable widget does not support on dragging coordinates. It is only possible to access on drag end coordinates, and then make the update.

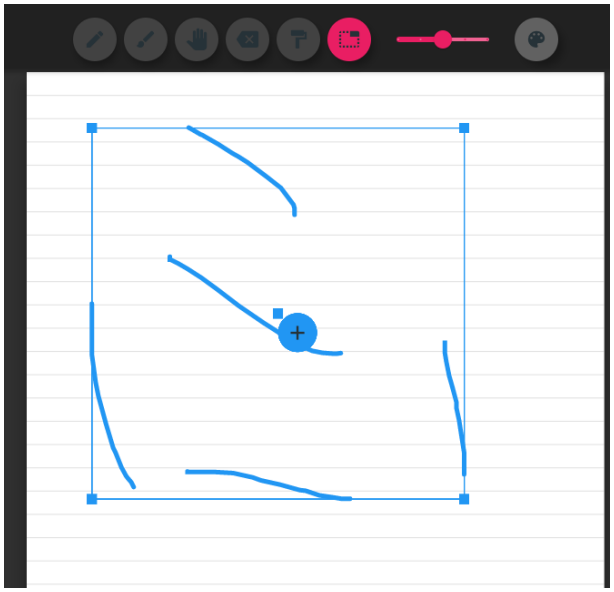

Figure 3.22: Select using a FloatingActionButton

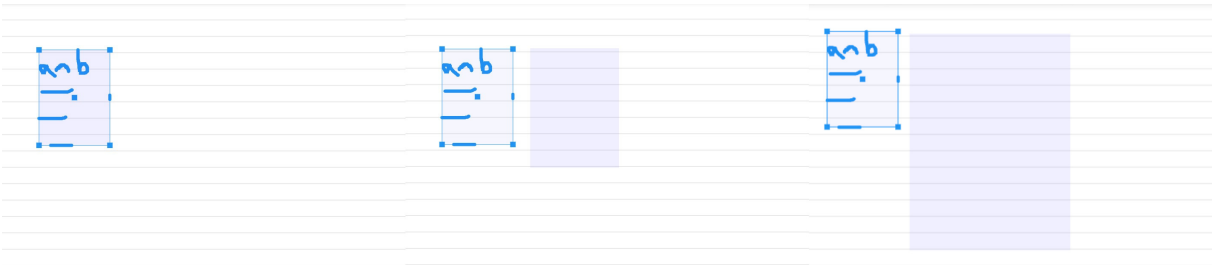

Figure 3.23: Sequence of an expression being dragged: Content is selected; Content is dragged in full screen mode; Content is dragged in a small application window

3.5.2 End of the development

After the selection was implemented, the idea was to allow for copy pasting the selected content. By simply double pressing what was selected, the expression would immediately be copied to a new location below. That old expression would be deselected and the new expression selected in its place, so the user could now move the copied content. Here is the first prototype of this functionality. 3.24.

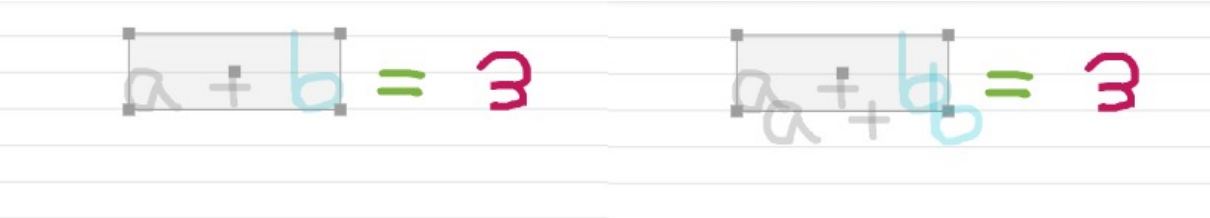

Figure 3.24: Selected expression; User double clicks on the selected expression

In order to be able to understand why this functionality and the development of the structure editing part of the application did not went further, its necessary to understand a bit more about Flutter and the overall project structure. That is why, in this part, it will be a bit more technical.

Similarly to the example provided in 2.5.3.D, the app is organized inside a Scaffold, which allows for the interface to have a bunch of prebuilt widgets like the app bar, the bottom app bar and the drawer. In the pseudo-code bellow, the project structure was simplified just so we can have some notions for what is about to be explain.

```
1 CanvasPage{
2     Scaffold(
3         drawer
4         body: PointerListener
5         appBar
6         bottomNavigationBar
7     )
8 }
```

In the PointerListener class, the body of our app, is where we handle all that happens on the canvas. The Canvas page is where we define what to do with what just happened.

For example, for a stroke to be saved: as soon as the user presses the canvas, PointerListener has a method that immediately starts to track every point the user has pressed and when the user releases the pointer, it saves all those points in a XppStroke structure. It sends that structure to CanvasPage to be stored in a collection of strokes and, at the same time, draws those strokes in the canvas through a

CustomPaint widget.

Initially, while trying to do the copy/paste functionality, the idea was to go to the structure where strokes where being saved, duplicate the selected strokes and change their offset as described in 3.24. Because the strokes have to be drawn before being able to be displayed on the canvas, we ended up with what is shown in the picture below 3.25.

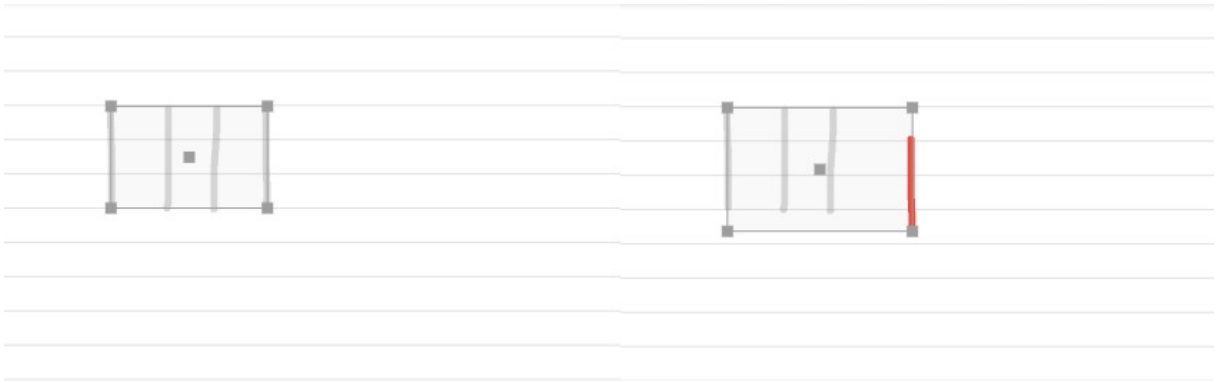

Figure 3.25: Problem with duplicating the strokes without drawing them

Although the strokes were being added to the collection of strokes, they were not being added to the canvas, so the old strokes would be replaced by the new ones at the canvas. In the image 3.25 we can see the last stroke being redrawn at the new location (in red). To solve this problem, a stroke had to somehow be drawn and stored with its own structure. The selected strokes's attributes (tool, points and color) were used to paint the new strokes, and that is what is shown in figure 3.24. The problem at this step is how the code is arranged inside the PointerListener. Because what is been drawn at the canvas is inside a Stack of widgets (so we can overlap drawings), this stack only accepts widgets inside, so we cannot call functions or any other methods. As the directly drawn strokes at the canvas were being saved before they were painted (through onPointerMove method used by PointetListener as explained before), they just had to be paited inside this stack at this step. Our new replicated strokes, on the other hand, can not use this method because they are really never manually drawn, they are built through code. This being said, we could not tell the interface to build a XppStroke structure with these newly drawn strokes neither to store them in the collection of strokes. The development of the project was stopped here because, without being able to duplicate content and classify it as an unique stroke/character, the recognition and the structure manipulation would not be possible.

Is important to mention that most available packages online are not available at the moment because of a new flutter update. The only thing we can do is wait for packages developers to fix the problem. It won't stop us from building or running our code until the packages author update to v2 Android embed-

ding on Flutter. The system evaluation was a bit harmed by this update because it was not possible to run the code on the iPad version. A downgrade to a previous version of Flutter and an upgrade of the disk space was tried manually but with no successful.

```
Running "flutter pub get" in xournalpp_mobile-main...            2,421ms
The plugins `disk_space, flutter_absolute_path` use a deprecated version of the
Android embedding.
To avoid unexpected runtime failures, or future build failures, try to see if
these plugins support the Android V2 embedding. Otherwise, consider removing
them since a future release of Flutter will remove these deprecated APIs.
If you are plugin author, take a look at the docs for migrating the plugin to
the V2 embedding: https://flutter.dev/go/android-plugin-migration.
```

Figure 3.26: Console error

4

Evaluation

Contents

The purpose of evaluating the system is to ascertain whether the implemented features are usable and if the application fulfills its overall goal, which is to support the presentation and manipulation of mathematical content. It will also reveal eventual bugs that the application might show. So feedback could be acquired about whether the project is usable or not, 10 users interacted directly with the system through 2 phases: In the first phase, through a scripted series of actions that the user had to complete; in the second phase, the idea was that the user could move freely over the interface. It was offered to the testers a one-page sheet explaining the interface and the action to perform 4.1.

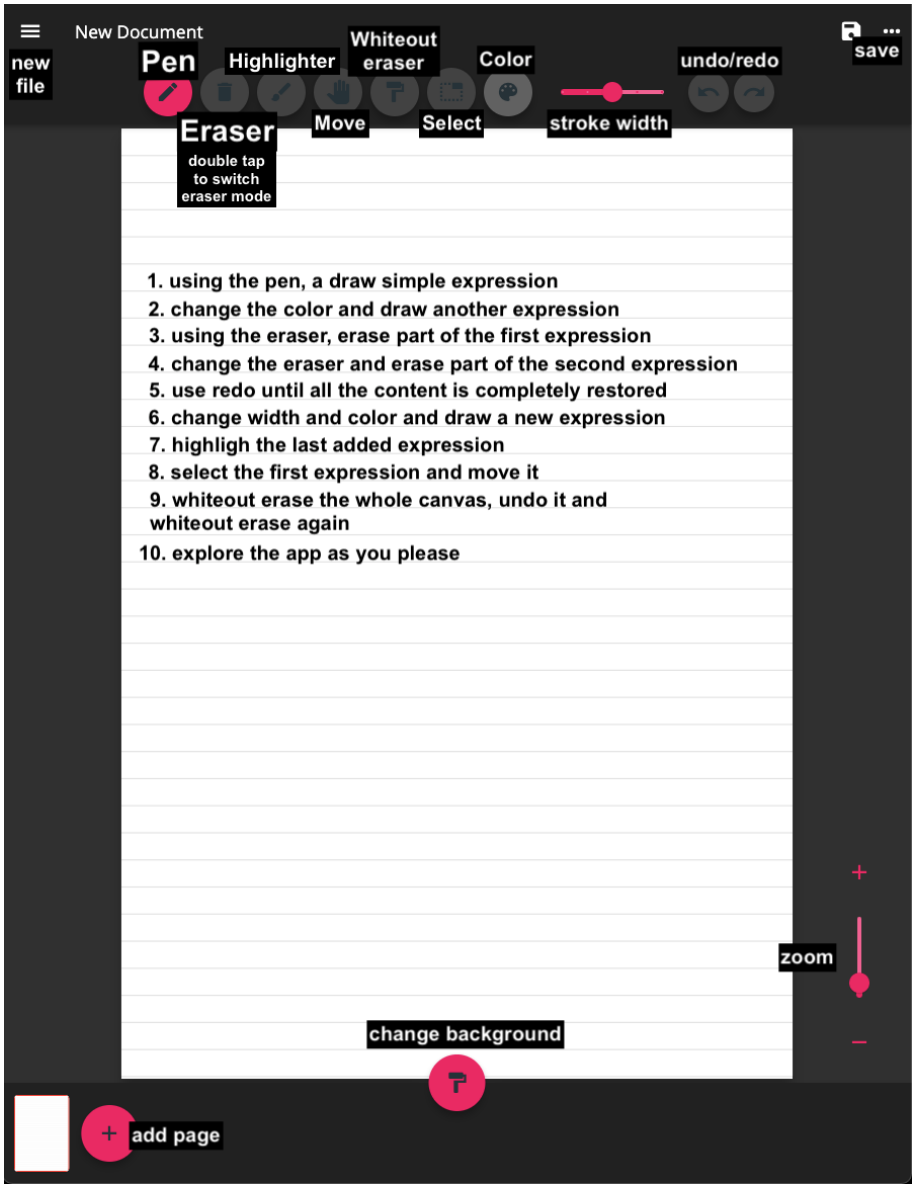

Figure 4.1: Script for the evaluation process

It was decided not to do an extensive questionnaire with questions like ”totally agree” or ”totally disagree” after the usability test and, instead, asking users to have a small conversation in the end where they would give some overall feedback of the experience and some suggestions of how the interface could be improved. It was asked the users to think out loud during the experience so it would be possible to understand what was going through their minds and to better analyze the experience later. Through observation of the users' actions and thoughts, and by communication, it was collected a detailed evaluation of the system. This way, the user was not overloaded with the experience and it was generated very interesting data regarding the usability of the system.

After the approval for starting the user test, the sheet of paper mentioned in 4.1 was handed. After the users read the script, it was explained to them how the interface worked and how each feature should be used. The select and the gestures were left unexplained. Testing the interface was then started and the following analysis and results were collected.

4.1 Phase 1: Users perform a script of actions

At steps 1, 2, and 3, there is not much to say because no problems were found. It was observed that most users were still getting used to the interface so they took a bit to select the tool asked to select. This changed after the users finished the scripted part of the user testing and explored the app by themselves. Some users asked if they could use the eraser at step 1 because they wanted to rewrite the expression. As the important here is to evaluate the performance and usability of each tool, it was asked to follow the script because the aspect of the content was not important.

At step 4, some users had problems with the originally implemented eraser, because it was not deleting the way they wanted (this problem was already addressed in Section 3).

At step 5, some users questioned why the redo was being used for adding back the deleted strokes and suggested it should be the undo to revert the last actions. It was noticed, in this step, that there were some problems restoring previously partially deleted strokes using the original eraser tool. This might have happened because the removal of stroke points was not being done instantaneously but instead through a series of small steps. In some cases, users had to redo around 10 times to restore a small portion of the removed part of the stroke.

At step 6, although all the testers manage to change the width and the color of the stroke with no

problem whatsoever, some of them forgot they had the eraser tool selected and had to change it back to the pen tool. Step 7 was also done with ease by all the users as the tool was kind of self-explanatory.

At step 8, it was wanted to see how the user would try to select content. The majority of the users were using lasso and rectangle selection and the rest was using a selection where they just had to click on the strokes they wanted to be selected. Users that tried to use the lasso selection couldn't select anything because the pointer didn't hover any content. Users that tried to press directly on the strokes they wanted to be selected found it strange why the content was not being selected but the animation was showing. People that were used to using rectangle selection immediately discovered how to select, as the initial part in this kind of selection is similar. At this step, it was explained to everyone how to correctly use this functionality.

Some users commented on the fact that the content was not moving along the container while being moved. Some people tried to make a new selection while one was still on the screen and got confused why they could not select more content. Users commented that sometimes they were not able to perceive where the box was while being dragged.

At step 9, after the whiteout was executed, most users tried to use the undo button instead of using the whiteout button to undo the whiteout eraser. Maybe this was due to the name ”undo” that was used to describe this step. After it was explained again how to revert this action properly they suggested one more time that undo should undo the last action and redo remake the last action.

4.2 Phase 2: Users move freely over the interface

And now, at step 10, it was told to the users to freely explore the app as they pleased and in the end to give small feedback regarding the overall usability of the app. Before letting them explore the interface, it was disclosed how the ”on double-tap” and ”on long-press” would allow them for changing between tools and that they could erase selected content by simply throwing it to the outside of the canvas.

Because it was just explained how the gestures worked, all the users decided to test out this new feature. At this step, only one user remembered to change back to the pen tool to perform gesture events. Because the rest of the buttons were linked to these gestures, this was not a problem. All the users found the ”on double-tap” and ”on long-press” gestures very helpful, as they would not need to manually press buttons to change between tools. At this step, all the users were able to use the select functionality for themselves and to test how content could be deleted while selected.

Some people also noticed it was not possible to draw single dots on the application. This functionality had to be removed so the gestures to change between tools could be implemented. Some suggested that the stroke width and color should not be the same to pen, highlighter, and erased. Independent stroke width and color for each one of them should have been configured.

They mentioned again the fact that strokes were not moving alongside the container while being dragged. This problem was not able to be solved as the widget used for dragging content around only allowed to access the pointer's offset after the drag was ended. The interface would also bug sometimes while trying to select the content that was previously partially deleted and added again (using the original eraser tool).

After some usages, users found interesting the way the select functionality was implemented, as they could have a more precise notion of what was being selected and make more customized and precise decisions, and rapidly adapted.

4.3 Usability testing conclusions

In the end, all the users complemented the overall experience with the interface. They also mentioned some usability issues (most of which had already been noted while they were testing the application) and made really strong suggestions on how the interface could be improved.

The fact that gestures were introduced to trigger action was highly praised. Not only for being able to change between tools faster but also the way the selected content could be deleted. Users said they would like to see this implemented in a more advanced and robust interface.

Users suggested:

- PDF and images should be possible to incorporate
- it should be able to shrink/expand selected content by pressing the edges
- be able to zoom in/out using a pinch gesture with 2 fingers
- select mode being automatically turned on when long pressing strokes
- being able to print screen by swiping up with 3 fingers
- some additional selection modes should be incorporated
- undo/redo buttons should be used to undo/redo actions and not be related with the strokes directly
- each stroke should have its own color and width
- detect different levels of pressure while drawing

- add text boxes
- allow for copy pasting selected expressions

The conclusion with this evaluation is that, although with some flaws, the implemented features are usable and the application fulfills its overall goal. This whiteboard application revealed itself suitable for inserting and manipulating mathematical content and to be a good starting point for being able to go through with the development of the first structure editor interface supported in multiple devices.

5

Conclusion and Future Work

Contents

Pen-based devices have the potential to play an important role in the future of the classroom. With digital ink as the main input, users can take advantage of the technology without having to sacrifice the pen and paper style of writing. Using tablet PCs only for writing does not exploit its full potential. It can help to solve problems related to mathematical content and lets students express themselves in innumerous ways, offering instructors insight into the student thought process and generating interesting artifacts for discussion. Furthermore, digital ink also facilitates active learning, allowing students to work freely while being directly engaged in the learning process. Tablet PC's computational capabilities can be a valuable help to manipulate and present mathematical content. Even if the user does not need syntactic manipulation of expressions, having features to reliably copy content can make a huge difference.

The idea was always to be focused on pen-based devices and on how digital ink could be a valuable asset to the educational area. Because I was extremely interested in learning about mobile development, when professor João Ferreira introduced me to the Xournal++ Mobile for the first time, I did not have much to think about before deciding to use it as starting point. Besides a well-designed interface, a pen and an eraser already implemented, it was built using Flutter. Flutter lets us program in a single code base (in dart) and auto-generate applications for iOS, Android, Windows, Mac, Linux, and even for the web. This is truly interesting to think from a programmer's point of view because not even a single line of code has to be changed for the interface to run on different devices and operating systems. Although Flutter itself has only 4 years of existence, I decided to learn a new language and embrace the opportunity I was being given.

Multiple different widgets were used on the project, but the most relevant to the theme was the CustomPaint. Because everything that is drawn on the canvas is produced by this widget, it was something to look into and be comfortable with. As mentioned before, this widget is used mainly for drawing custom graphics that does not involve UI components to be able to produce innovative and create designs. We go a little further with CustomPaint. The method that is used for creating custom shapes was used to build paths between the points the user presses on the canvas and consequently create what we call an "XppStroke" structure.

As Flutter is relatively new, is deductible that there is still not much information online, at least compared to Python or Java. Because, in our case, the subject is even more specific, the amount of information that can be gathered and the help we can get is even more reduced. This was the main drawback while developing the project. We could not manage to find much help from online searching as there was not enough development in this subject. Making a whiteboard application on Flutter is way more specific than just building a simple user interface with buttons and menus as a common developer uses

it for.

When we look at the flutter whiteboard interfaces mentioned earlier, we can see there is not much to compare to and to be based on. These interfaces have a pen, an eraser, and that is it. If we compare those interfaces to the Xournal++ Mobile that we started with, we can see how superior this application is at every level. Although the tools for on canvas actions are the same (a pen, an eraser, and a color change), this interface is much more appealing design and structure-wise and has already tools implemented for changing the background, for adding new pages, and for saving the document. It looked like a promising application. Unfortunately, they stopped updating it as soon as the development started, so they must have come to the conclusion it was no longer worth it to keep updating.

While making changes to the application and adding new features, I realized I was not able to copy and paste previously drawn strokes because of the way the code was organized. This was already explained in . This being said, it was not possible to go further with the structure editing part of this thesis original idea because it was simply not possible to manipulate and store strokes created via code, as they had to be physically drawn to be saved and handled.

It is also important to mention that, after Flutter updated to the 2.5 version, we no longer were able to use external packages as they were considered deprecated and were going to be removed in future releases. The authors have to migrate these plugins to the V2 embedding before we can use the packages again.

5.1 Achievements

By using Xournal++ Mobile as a starting point, it was developed the most advanced whiteboard application built with Flutter until today (to my knowledge). In addition to the design and functional changes made to the interface itself, it was created a new eraser, a highlighter, a whiteout eraser, undo and redo buttons, a select functionality, and added some action triggered with the usage of gestures. We can say this thesis was a pioneer when it comes to whiteboard applications built-in Flutter and that the contributions are remarkable since it is very little (to none) exploration in the field (once more, to my knowledge). The work that has been done here brings new contributions to the state of the art regarding whiteboard applications built-in Flutter and gives us an insight into what kind of obstacles we might face when trying to explore this toolkit and more particularly, the CustomPaint widget.

5.2 Future Work

Firstly, it would be reasonable to make changes to the interface based on the user testing results to improve the overall usability of the system. The integration of pdf and image files, text boxes, and making the interface rely way more on gestures like the ones described at the end of the evaluation would be the next step. If somehow it would be possible to overcome what ended the development of the application in the first place, the development of the project would go to a whole new level with being able to manipulate strokes.

The starting point would be to implement the copy-pasting of selected expressions. With the integration of a handwritten mathematics recognizer, the points that are being pressed on the screen could automatically be subjected to the recognizer while being drawn and as soon as the user finishes the action, recognized characters would be stored instead of the strokes. Another possible approach could be to use the recognizer on selected content and then proceed to store strokes as characters in another structure. Because every individual has his handwriting, it would be interesting to have a model for handwritten characters which was updated as the user draws new content. Just this feature alone would be a lot to think about mainly because recognition systems normally output multiple results with different percentages of being correct and we have to be able to select or to give the user the decision on which recognition was correct. It is also important to consider the fact the system might output an incorrect answer and the user would have to be able to change that. We also would have to think of a way for the user to take advantage of the recognition without overloading the system.

Finally, a good contribution to this thesis would be to have an article published about the usage of CustomPaint for building whiteboard applications using Flutter.

Bibliography

[1] R. Anderson, R. Anderson, P. Davis, N. Linnell, C. Prince, V. Razmov, and F. Videon, "Classroom presenter: Enhancing interactive education with digital ink," *Computer*, vol. 40, no. 9, pp. 56–61, 2007.

[2] E. M. Taranta and J. J. LaViola Jr, "Math boxes: A pen-based user interface for writing difficult mathematical expressions," in *Proceedings of the 20th International Conference on Intelligent User Interfaces*, 2015, pp. 87–96.

[3] G. Labahn, E. Lank, S. MacLean, M. Marzouk, and D. Tausky, "Mathbrush: A system for doing math on pen-based devices," in *2008 The Eighth IAPR International Workshop on Document Analysis Systems*. IEEE, 2008, pp. 599–606.

[4] J. J. LaViola Jr, "An initial evaluation of mathpad2: A tool for creating dynamic mathematical illustrations," *Computers & Graphics*, vol. 31, no. 4, pp. 540–553, 2007.

[5] J. J. LaViola Jr and R. C. Zeleznik, "Mathpad2: a system for the creation and exploration of mathematical sketches," in *ACM SIGGRAPH 2006 Courses*, 2006, pp. 33–es.

[6] A. Mendes, R. Backhouse, and J. F. Ferreira, "Structure editing of handwritten mathematics: Improving the computer support for the calculational method," in *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, 2014, pp. 139–148.

[7] F. Konukman, E. Rabinowitz, M. W. Kernodle, and R. N. McKethan, "The effective use of powerpoint to facilitate active learning," *Journal of Physical Education, Recreation & Dance*, vol. 81, no. 5, pp. 12–16, 2010.

[8] R. A. Bartsch and K. M. Cobern, "Effectiveness of powerpoint presentations in lectures," *Computers & education*, vol. 41, no. 1, pp. 77–86, 2003.

[9] L. Anthony, J. Yang, and K. R. Koedinger, "Evaluation of multimodal input for entering mathematical equations on the computer," in *CHI'05 Extended Abstracts on Human Factors in Computing Systems*, 2005, pp. 1184–1187.

[10] T. D. Trang, “Using ppt in the esl classroom: Benefits and drawbacks from high school students’ perspectives,” *Transforming English Language Education in the Era of Globalization*, pp. 301–309, 2015.

[11] J. Cromack, “Technology and learning-centered education: Research-based support for how the tablet pc embodies the seven principles of good practice in undergraduate education,” in *2008 38th Annual Frontiers in Education Conference*. IEEE, 2008, pp. T2A–1.

[12] T. J. Fitzgerald, “The tablet pc takes its place in the classroom,” *The New York Times*, vol. 9, 2004.

[13] B. Simon, R. Anderson, C. Hoyer, and J. Su, “Preliminary experiences with a tablet pc based system to support active learning in computer science courses,” in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, 2004, pp. 213–217.

[14] R. J. Anderson, C. Hoyer, S. A. Wolfman, and R. Anderson, “A study of digital ink in lecture presentation,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 567–574.

[15] M. A. Imtiaz, R. Blagojevic, A. Luxton-Reilly, and B. Plimmer, “A survey of intelligent digital ink tools use in stem education,” in *Proceedings of the Australasian Computer Science Week Multiconference*, 2017, pp. 1–8.

[16] L. Anthony, J. Yang, and K. R. Koedinger, “Evaluation of multimodal input for entering mathematical equations on the computer,” in *CHI’05 Extended Abstracts on Human Factors in Computing Systems*, 2005, pp. 1184–1187.

[17] S. Cheema and J. J. LaViola, “Applying mathematical sketching to sketch-based physics tutoring software,” in *International Symposium on Smart Graphics*. Springer, 2010, pp. 13–24.

[18] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos, *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

[19] C. C. Bonwell and J. A. Eison, *Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports.* ERIC, 1991.

[20] J. Mills, A. Bonner, and K. Francis, “The development of constructivist grounded theory,” *International journal of qualitative methods*, vol. 5, no. 1, pp. 25–35, 2006.

[21] M. Ben-Ari, “Constructivism in computer science education,” *Acm sigcse bulletin*, vol. 30, no. 1, pp. 257–261, 1998.

[22] A. E. Johnson, “Digital ink: in-class annotation of powerpoint lectures,” *Journal of Chemical Education*, vol. 85, no. 5, p. 655, 2008.

[23] J. E. Susskind, "Powerpoint's power in the classroom: Enhancing students' self-efficacy and attitudes," *Computers & education*, vol. 45, no. 2, pp. 203–215, 2005.

[24] A. Szabo and N. Hastings, "Using it in the undergraduate classroom: should we replace the blackboard with powerpoint?" *Computers & education*, vol. 35, no. 3, pp. 175–187, 2000.

[25] J. M. Apperson, E. L. Laws, and J. A. Scepansky, "The impact of presentation graphics on students' experience in the classroom," *Computers & Education*, vol. 47, no. 1, pp. 116–126, 2006.

[26] R. M. Felder and R. Brent, "Random thoughts: Death by powerpoint," *Chemical Engineering Education*, vol. 39, no. 1, pp. 28–29, 2005.

[27] D. Hlynka and R. Mason, "'powerpoint'in the classroom: What is the point?" *Educational Technology*, vol. 38, no. 5, pp. 45–48, 1998.

[28] D. Hlynka, "Postmodernism in educational technology: update: 1996-present," *Handbook of Research for Educational Communications and Technology*, vol. 2, pp. 243–246, 2004.

[29] A. E. Johnson, "Digital ink: in-class annotation of powerpoint lectures," *Journal of Chemical Education*, vol. 85, no. 5, p. 655, 2008.

[30] R. Anderson, R. Anderson, B. Simon, S. A. Wolfman, T. VanDeGrift, and K. Yasuhara, "Experiences with a tablet pc based lecture presentation system in computer science courses," in *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, 2004, pp. 56–60.

[31] G. Labahn, E. Lank, M. Marzouk, A. Bunt, S. MacLean, and D. Tausky, "Mathbrush: A case study for pen-based interactive mathematics," in *Proceedings of SBIM, Eurographics Workshop on Sketch-Based Interfaces and Modeling. Geneva, Switzerland: Eurographics Association*, 2008.

[32] K. Ruthven, "Instrumenting mathematical activity: Reflections on key studies of the educational use of computer algebra systems," *International Journal of Computers for Mathematical Learning*, vol. 7, no. 3, pp. 275–291, 2002.

[33] S. Toyota, S. Uchida, and M. Suzuki, "Structural analysis of mathematical formulae with verification based on formula description grammar," in *International Workshop on Document Analysis Systems*. Springer, 2006, pp. 153–163.

[34] G. Labahn, S. MacLean, M. Marzouk, I. Rutherford, and D. Tausky, "A preliminary report on the mathbrush pen-math system," in *Maple 2006 Conference*, 2006, pp. 162–178.

[35] D. Carlisle, P. Ion, R. Miner, N. Poppelier *et al.*, "Mathematical markup language (mathml) version 2.0," *W3C recommendation*, vol. 21, 2001.

[36] A. Borning, “The programming language aspects of thinglab, a constraint-oriented simulation laboratory,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 3, no. 4, pp. 353–387, 1981.

[37] W.-M. Roth, “Affordances of computers in teacher-student interactions: The case of interactive physics™,” *Journal of research in science teaching*, vol. 32, no. 4, pp. 329–347, 1995.

[38] E. McClintock, Z. Jiang, and R. July, “Students’ development of three-dimensional visualization in the geometer’s sketchpad environment.” 2002.

[39] J. J. Laviola Jr, “Mathematical sketching: a new approach to creating and exploring dynamic illustrations,” 2005.

[40] A. Mendes and J. F. Ferreira, “Towards verified handwritten calculational proofs,” in *Interactive Theorem Proving*, J. Avigad and A. Mahboubi, Eds. Cham: Springer International Publishing, 2018, pp. 432–440.

[41] A. Mendes, “Structured editing of handwritten mathematics,” Ph.D. dissertation, University of Nottingham, 2012.