*education sciences*

MDPI

*Article*

# Incorporating Arts into Electronics Engineering: A STEAM Approach to an Embedded Systems Programming Course

**Csaba Zoltán Kertész**

Department of Electronics and Computers, Transilvania University of Brasov, Eroilor 29, 500036 Brasov, Romania; csaba.kertesz@unitbv.ro

**Abstract**

The growing demand for electronics engineers is one of the cornerstones of STEM education. Recent trends in education show an extension of the STEM principle into STEAM by mixing Arts with the traditional Science, Technology, Engineering, and Math disciplines. Especially in elementary education, this has beneficial effects by increasing the appeal of STEM disciplines. This STEAM principle is less studied in university settings, but it can be beneficial for engineering students as well. This paper presents a case study of extending an Embedded Systems Programming class to include GUI design elements. Employing graphical user interfaces in embedded devices has been an increasing trend in the last decade, and there is also demand for introducing it into courses concerning embedded systems and microcontrollers. Teaching engineering students about graphic design has two main benefits: it increases the appeal of the course and also leads to better understanding the interaction between the two worlds of Arts and Engineering. The survey results of students after finishing the course show a high satisfaction level.

**Keywords:** STEAM education; Embedded Systems Programming; GUI design

check for updates

Academic Editor: Nicole Pitterson

Received: 15 July 2025
Revised: 31 August 2025
Accepted: 3 September 2025
Published: 10 September 2025

**Citation:** Kertész, C. Z. (2025). Incorporating Arts into Electronics Engineering: A STEAM Approach to an Embedded Systems Programming Course. *Education Sciences*, *15*(9), 1189. https://doi.org/10.3390/educsci15091189

**Correction Statement:** This article has been republished with a minor change. The change does not affect the scientific content of the article and further details are available within the backmatter of the website version of this article.

**Copyright:** © 2025 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

The acronym STEM has long been established to cover education in the fields of Science, Technology, Engineering, and Mathematics, an important focus of training for global competitiveness (Breiner et al., 2012). In recent years a new term, STEAM, has been introduced that incorporates Arts with these fields (Cook, 2016; Land, 2013; Piro, 2010). Many ways to achieve this have been proposed (Henriksen, 2014), like employing artistic design in project-based learning scenarios (Connor et al., 2015).

Studies show that using artistic elements while teaching STEM disciplines can lead to increased creativity among students (Marmon, 2019; Zhbanova, 2019), although it may be unclear how much this is due to the Arts portion and how much it is due to the multidisciplinary nature of STEM disciplines in general, as shown by Aguilera and Ortiz-Revilla (2021). Nonetheless, systematic reviews generally suggest an increase in student creativity and creative thinking (Samaniego et al., 2024), collaborative engagement (Hardiman & JohnBull, 2019), and digital competencies (Deák & Kumar, 2024).

These studies usually focus on primary and sometimes secondary education, also remarking on obvious benefits from the use of art as a didactic supplement, such as science cartoons (Marques et al., 2023), arts-integrated reinforced learning with music or visual arts (Haroutounian, 2019), or even acting and dancing (Gibson, 2020).

At the university level, fewer studies focus on the shift from STEM to STEAM, as the curricula of majors in STEM fields are already well established. Of course, there are fields

that are good candidates for integrating arts, such as Industrial Design (Díaz-Obregón et al., 2019) and Mechanical Engineering (T. Singh, 2021). The curricula of these fields often include dedicated courses for the Digital Arts.

Applying a STEAM principle to Electronics Engineering can also have benefits for students. There are only a handful of studies that refer to STEAM practices in the domain of Electrical or Computer Engineering. For example, Zaher and Hussain (2020) presented a scenario using graphic design to explain accuracy and resolution in analog and digital circuits. While Computer Engineering and sometimes even Electronics Engineering curricula do contain dedicated courses oriented to Digital Arts, like Multimedia, Graphical Programming, etc., integrating arts into domain-specific courses is not usually considered. There are quite a few studies on modern approaches, such as using e-learning technologies and multimedia examples in courses (Wong & Ng, 2016), but there is a relative lack of studies in the literature on the use of a direct STEAM approach, where artistic elements are integrated into engineering curricula.

This paper presents such an approach that integrates notions of visual arts into an Embedded Systems Programming Course in the form of GUI design and programming. These aspects are usually overlooked, as such courses focus on low-level programming, optimization techniques, and hardware–software co-design by integrating microcontrollers into electronic circuits (Barr, 2006; Romanca & Ogruțan, 2011). However, current industry trends increasingly require GUI components in embedded systems. Traditional user interfaces, with buttons, LEDs, and numeric displays, are being replaced more and more by touchscreen interfaces. Once used only in high-end expensive devices, these screens are now often demanded by users accustomed to them on their personal mobile devices. The development of such gadgets is performed by teams of engineers, programmers, and graphical designers. The different backgrounds of team members may lead to communication gaps and conflicting priorities between engineers, who have a good technological understanding of the hardware and the development possibilities, and artists, who concentrate on user experience and subjective feeling of the design without considering any hardware limitations. This can lead to development issues, delays in time-to-market, and compromises that are not always welcomed by the final users. Familiarizing engineering students with artistic concepts can bridge the communication gap between team members and reduce misunderstandings that lead to the issues mentioned above.

The following section explains how artistic concepts in the form of GUI design were integrated into an Embedded Software Programming course at Transilvania University of Brasov. The study material for this course is presented in detail, focusing on the artistic elements introduced by GUI design. The section also describes the tools needed for graphic design and the workflow for turning the design into an engineering solution. There is also an emphasis on the mixing of the two worlds and on how to achieve a better integration between arts and engineering.

After describing the tools and methods, the Discussion Section presents the findings of a survey of students at the end of the semester. The proposed course material was run for two different semesters, and at the end of each semester, the students had the option to fill out a questionnaire to offer feedback on the course. This is a usual procedure, also available in any other course, but in this case, it had additional items, where students also had to rate the various modules inside the course. They rated it both for perceived difficulty and final satisfaction levels. Additionally, they had to answer some open questions about what they liked, what they disliked, and what they would propose to enhance the course. Rating the different parts of the course separately allows for better insight into how the new STEAM approach is perceived by the students.

## 2. Materials and Methods

The Embedded Software Programming course at Transilvania University is a final year course for the Bachelor's Degree in Applied Electronics. It aims to teach students to program microcontrollers and design various embedded applications. It is a practical course in which students form teams to develop their own projects during the semester. They learn all aspects of teamwork by assuming the roles for all the aspects of real-life projects concerning embedded system development: project management, architectural design, hardware design, low- and high-level programming tasks, testing, and project documentation.

Students learn to use the most common peripherals inside microcontrollers, such as timers, digital input/output ports, analog-to-digital converters, and UART, SPI, and $I^2C$ communications, as well as interrupt controllers. They learn how to connect different sensors and actuators to these peripherals to interface the microcontroller with the outside world. Also, from a software perspective, they learn to write efficient C code and to optimize it for the microcontroller architecture.

Due to recent trends as described earlier and due to industry prompts, since 2020 a novel element has been added to this course: using a touchscreen interface for the microcontroller and programming GUI elements. Students now use an STM32F746G-Discovery board as a laboratory module to develop and test their assignments, which now include a mandatory GUI component as well.

### *2.1. Tools Used in the Classroom*

The hardware development part of the student project is structured around an STM32F746G-Discovery development kit. This board features a 32-bit ARM microcontroller with an incorporated 2D graphics accelerator, called DMA2D, a $480 \times 272$ pixel touchscreen display, and an Arduino-compatible header that allows attaching various sensor and actuator shields (STMicroelectronics, 2020). With this development kit, students can easily design and simulate different household appliances, automotive devices, or medical devices according to their project assignment. Each student team has a custom extension board connected to the Arduino connector, which contains additional interfaces needed for their assigned project. For example, the extension board for an intelligent greenhouse controller project contains an environment sensor (temperature, luminosity, pressure, and humidity) and several output commutators (which can turn on or off some valves simulating heating and watering controllers).

Students design the GUI to be displayed on the onboard screen and program the microcontroller to interact with the attached sensors. An example student project that implements this intelligent greenhouse controller on this board is presented in Figure 1.

The programming and debugging of the microcontroller are performed using STM32CubeIDE. This tool allows configuring the peripherals of the microcontroller, generating all the glue code for the built-in low-level driver libraries, compiling and linking the code, flashing the microcontroller, and debugging the target directly in-circuit.

The example project from which students start their development already has all necessary components configured for the onboard devices (memory, touchscreen, buttons, and external connectors). Students must add just their specific devices, like $I^2C$ interfaces for digital sensors, ADCs for analog sensors, and GPIO pins for actuators. After configuring all the necessary peripherals, the STM32Cube generates all the necessary C code, so students can dive directly into GUI design.




**Figure 1.** GUI elements on the screen of an STM32F476NG-Discovery board.

GUI programming itself is performed with Embedded Wizard Studio from Tara Systems. This tool allows fast WYSIWYG prototyping of GUI elements and complete asynchronous GUI programming based on a signal-slot mechanism (Banach & Schweyer, 2024). It employs its own programming language, named Chora, which is an object-oriented programming language specially targeted for GUI objects, including native variables for geometry and colors. Figure 2 shows Embedded Wizard Studio in action with the student project presented earlier.

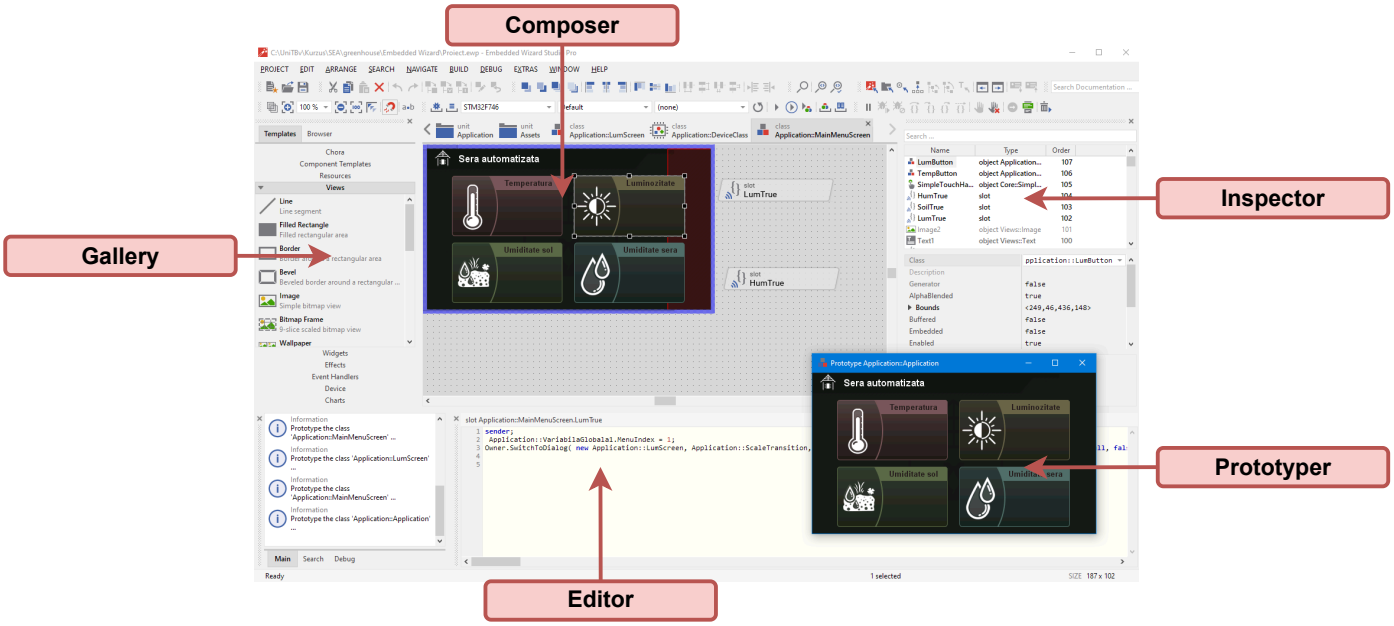



**Figure 2.** Embedded Wizard Studio with a student project.

As highlighted in the figure, the important parts of Embedded Wizard Studio are as follows:

**Composer:** Used to compose graphic scenes from simple graphic objects. An important task for a GUI engineer is to break down a complete graphic design into such objects and recompose it here in this Composer.

**Inspector:** A table containing all the properties of the selected graphic objects. Engineers can interact with these properties to configure the appearance and behavior of the objects.

**Gallery:** A container for a large number of predefined graphic objects that can be inserted into the Composer by a simple drag and drop operation. Engineers can use these objects either directly in the Composer or as starter objects for some other more intricate design elements. They can also define custom objects and add them to the Composer.

**Editor:** Used for writing code needed for interactive or animated objects.

**Prototyper:** A small window with the exact appearance of the embedded screen. It can be used to verify the whole design in real time, including all dynamic behavior, like animations, interaction between objects, and response to user actions.

Embedded Wizard Studio takes care of generating the C code corresponding to the composed scenes, also converting the Chora code into C code. Using the associated Platform Package and Build Environment, the code can be generated for any supported embedded platform, including the STM microcontrollers. For the latter, build systems and integration to STM32CubeIDE are also generated, so the GUI itself can be directly compiled and run on the target without any additional coding necessary.

For creating the graphic elements, students can use Adobe Photoshop CC or GIMP. These are the go-to options for artists and designers to create nice-looking rasterized graphic scenes and objects for the user interface (Hammel, 2012). In addition, vector graphics can be created using Adobe Illustrator or Inkscape. Vector graphics are particularly suitable for designing animated GUI elements that are subject to geometric transformations during animations, resulting in much nicer rendering. Animations can also be designed using Blender or Figma, the latter being specially dedicated to user interface design.

No matter which tools students choose for their artistic endeavors, the resulting graphics have to be broken down into basic graphic objects, which then will be used in the Embedded Wizard Studio composer to recompose the desired scene.

### *2.2. GUI Building Tasks*

Student projects start from a specification established in common agreement with the teacher. Based on these specifications, students have to perform various tasks, both artistic and engineering, to achieve the final product. The main steps are presented as a diagram in Figure 3.

The artistic tasks include drawing the graphic elements and envisioning various animations. The drawings can be in the form of vector art or rasterized graphics. Vector graphics design (with tools like Inkscape or Adobe Illustrator, for example) results in a more profound understanding of the building blocks of a graphic scene, and, due to the mathematical nature of the graphic tools, an easier understanding for engineers. Vector graphics are also lighter on necessary resources memory-wise, but they need much more computing power for rendering. This can be a problem, though, on low-end microcontrollers, so normally the trickier geometries have to be prerendered on a PC and used as raster graphics in the microcontroller.

So, there is also a need to draw raster graphics (with tools such as Adobe Photoshop or GIMP). This is also useful for more creative design, as these tools allow students to experiment with lights, shadows, colors, contrast, and textures, unlocking or even enhancing their artistic abilities to create a more attractive GUI scene.

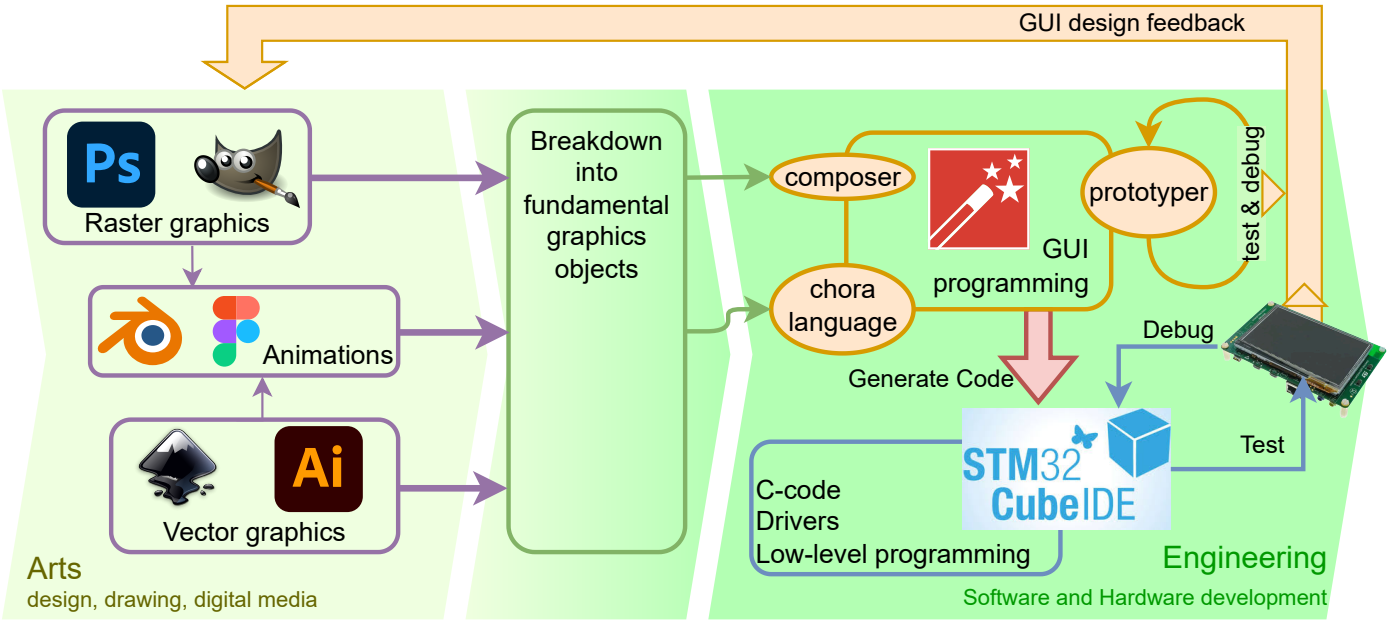



**Figure 3.** Embedded GUI programming course flow with the main tasks involved in the student projects.

The engineering tasks include high-level GUI programming, low-level microcontroller programming, testing, and debugging.

Initially, graphics are imported into the Composer of Embedded Wizard and animations and event handling are implemented in Chora language. These are then tested using the built-in prototyper. The prototyper can replay the entire GUI, including high-level events, on the PC, offering a first impression of the GUI. Development and prototyping form a feedback loop: graphics and animation issues can be detected during debugging, and, if needed, the whole process starting from the graphic design step can be reiterated.

Subsequently, Embedded Wizard generates the build environment and C code. These are brought into STM32CubeIDE for another round of testing and debugging, this time directly on the target. At this step, mostly low-level issues are detected, but some graphical issues, especially color-related issues coming from the different types of display, can also be identified and returned to the graphic design stage for adaptations.

Most importantly, there is a design phase that bridges the artistic and engineering tasks. Graphics must be adapted to the requirements of the GUI programming environment and to the constraints of the limited resources of the microcontroller. Graphic scenes on the screen must be constructed from some basic graphic elements available in the Embedded Wizard composer, like lines, rectangles, shadows, raster graphics, or vector graphics, which can be combined using simple alpha blending.

The sophisticated blending mechanisms in Embedded Wizard would allow the use of any type of graphics art created with the previously mentioned tools; however, the limited resources of the microcontroller are not always enough for complicated graphics. So, all the design elements must be broken down into basic elements, which reduce the memory and CPU performance costs on the microcontroller.

For example, complete raster graphics may need more memory than available on the microcontroller, or complicated vector graphics may need more processing power to achieve real-time response than the CPU allows. To reduce the cost, raster graphics can be broken into pieces in such a way that some pieces (like lines or rectangles) are rendered directly by the CPU and not stored in memory. Likewise, complicated vector graphics can be partially rendered and used as raster graphic pieces, so displaying the whole screen can still be done in real time.

Animations can also be oriented toward the possibilities of the microcontroller. For example, complex geometric transformations can be reduced into simple moving and resizing effects, which, combined with additional semitransparent blending of components, create the visual impression of the original effect.

Students learn by simple examples of all available graphic operations, which can be run at reasonable cost on the microcontroller. From these, they can then decide how to break down the graphic drawing coming from Photoshop into smaller pieces and how to reconstruct them in the composer.

### 2.3. Advantages of Embedded GUI Design Course

Teaching engineers about GUI design is beneficial from two perspectives. First, there is the usual transversal perspective: the additional artistic components broaden the learning experience, increase motivation levels for the students, and provide additional competencies.

Students familiarize themselves with basic concepts of drawing and design, such as color models, human perception, directing eye movement with lighting effects, placement of important elements on the screen, symmetry, perspective, rule of thirds, golden ratio, etc. Using a mathematical explanation for most of these concepts resonates quite well with the engineering background of the students.

The second perspective is an engineering one. Traditionally, GUI design is performed by specialized designers trained in all aspects of drawing techniques, human perspective, and psychology. However, they are used to high-end drawing tools, creating their art on personal computers, tablets, and mobile phones, using high-end computing resources. Embedded devices, on the other hand, are characterized by a lack of resources, reduced processing power, and strict power usage requirements. Engineers familiar with the internals of the microcontroller as well as with some graphic design knowledge can make the transition from high-end resources to low-level design much more convenient.

Microcontrollers—even those that have a dedicated display interface inside—usually are capable of very basic graphical operations only. These include DMA transfer from a pixel framebuffer to the display itself and some raw operations for filling the pixel framebuffer. Among these, we can enumerate blitting (copying rectangular areas of various sizes), clipping (restricting the copy operation to a given target area), and alpha blending (mixing the color information from the source area and the destination area based on the value of a transparency channel).

Normally, more advanced graphical operations, which are handled by a dedicated GPU on higher-end systems, have no acceleration support on smaller microcontrollers, so they have to be executed directly in the CPU, requiring a large amount of computation. This can result in lower framerates, higher power consumption, and a possible negative impact on the other low-level operations for which the microcontroller is used in the first place.

To design a GUI for such a microcontroller that avoids such inconveniences, the designer must be aware of the microcontroller features, the possible acceleration techniques, and the mathematics behind the executed graphical operation to properly estimate the computation power required.

Engineering students can learn how to achieve the desired look-and-feel using only lines, rectangular areas, linear gradients, and color casts of various rasterized graphical objects. This makes it possible to interact better with designers to come up with designs that also run efficiently on the microcontroller.

### 2.4. Student Feedback

To evaluate student perception about the new modules and to improve the course, student feedback is collected at the end of each semester. From the many instruments

available to obtain this feedback (Richardson, 2005), the simplest form of a questionnaire was used in which students had to rate different components of the course on a five-point scale similar to (Kanwar & Sanjeeva, 2022) and (Adnan et al., 2016). They had to rate their level of satisfaction, with one point indicating very low satisfaction and five points indicating very high satisfaction, as well as to rate the perceived difficulty of the given component, with one point indicating very difficult and five points indicating very easy. The rating questions included every component of the course separately, including familiar ones—e.g., microcontroller architecture, programming Arduino-like microcontrollers, low-level and high-level programming of STM32 microcontrollers, and using each of the peripherals in the microcontroller—as well as newly introduced components—GUI design, Embedded Wizard programming, Chora language, asynchronous GUI programming, low-level integration, testing, and debugging the GUI application. Overall satisfaction levels were also separately asked for the course (theoretical elements) and for the laboratory (practical elements).

Besides the questions to rate different components, students also had to answer 3 open-ended questions:

- What did you like about the course?
- What did you dislike about the course?
- What enhancements would you propose for the course?

The questionnaire was administered on the Moodle e-learning platform immediately after the final exam. The Moodle standard questionnaire module was used, with response mode set to anonymous. The platform only checks the student identities to make sure that each student can answer only once, but the answers are not linked to the student accounts, and only the total scores are displayed.

The completion of the questionnaire was performed on a voluntary basis, students being informed on the questionnaire page about the purpose of the survey and that no personal data would be recorded before starting to complete the forms. Completing all questions was also optional, so there was some fluctuation in total responses for each question. Approximately 25–30% of the students responded each semester.

## 3. Results and Discussion

At the end of the semester, students were asked to provide their insights on adding the GUI development modules to this course. They had to complete a survey with numeric ratings and open-ended responses about their perception of the difficulty and the added value of GUI design classes compared to more traditional embedded programming assignments.

First, they had to rate on a scale of 1–5 (from hard to easy) their perceived difficulty of the various elements of the class:

- The microcontroller programming tasks, like working with peripherals, configuring the MCU, polling and interrupt handling, and interaction with various sensors.
- The GUI design task, including the drawing of UI elements, conceptualizing the dynamic behavior of the components, animations, using external tools like Photoshop or Figma, and then bringing the art into EmbeddedWizard.
- The asynchronous programming needed by the GUI components, handling events, implementing the signals and slots, and avoiding race conditions.

Figure 4 presents the distribution of the student ratings about the perceived difficulty.

By performing a visual inspection of the distribution plots, we can observe a somewhat normal bell-shaped distribution, noting that the MCU tasks and the programming tasks are leaning towards the easy side, while the GUI design has a more balanced perceived difficulty. This is most likely due to the students' familiarity with computer architectures,

digital electronics, and programming in general, skills already acquired in previous years. The artistic touch in the GUI design tasks and the use of tools that are not similar to any previously studied tool cause the students to perceive their assignments as more difficult compared to the other tasks.

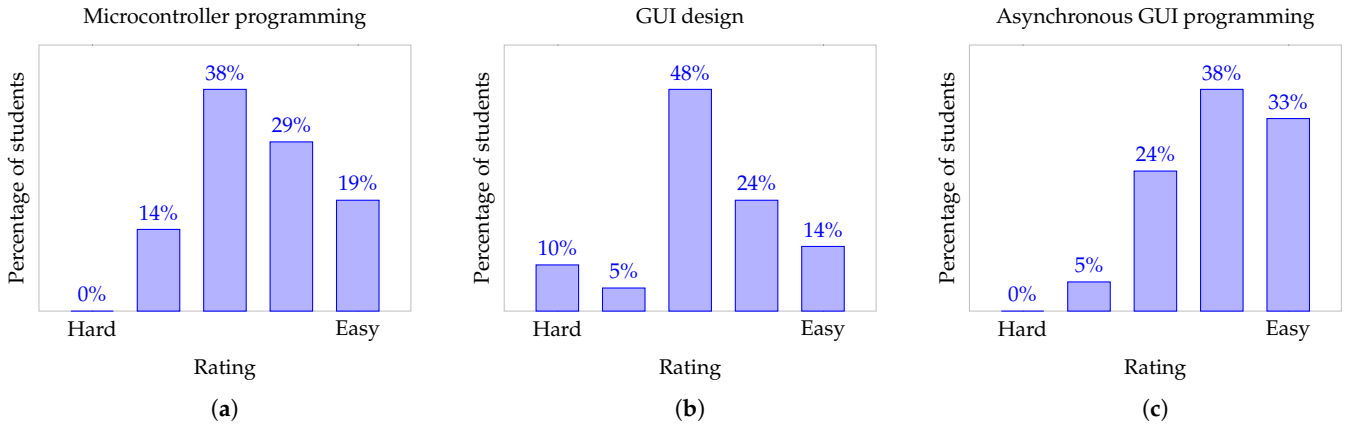



**Figure 4.** Perceived difficulty of various course elements: (**a**) microcontroller programming and configuring peripherals; (**b**) GUI design, drawing UI elements, conceiving animations; (**c**) asynchronous style programming for GUI components and events.

In addition to these initial perceptions, the students were also surveyed on how well they perceived the addition of the GUI design to the study material and their overall satisfaction. Again, they had to rate on a scale of 1–5 the following:

- How much they agreed with the addition of artistic notions into the course material (theoretical part).
- How much they agreed with the addition of drawing tools, animation design, and GUI programming into the laboratory assignments (practical part).
- Their overall satisfaction with including embedded GUI design in the class.

The first two items were rated from entirely disagree to entirely agree, and the last item from very low to very high satisfaction. The resulting distributions are presented in Figure 5.

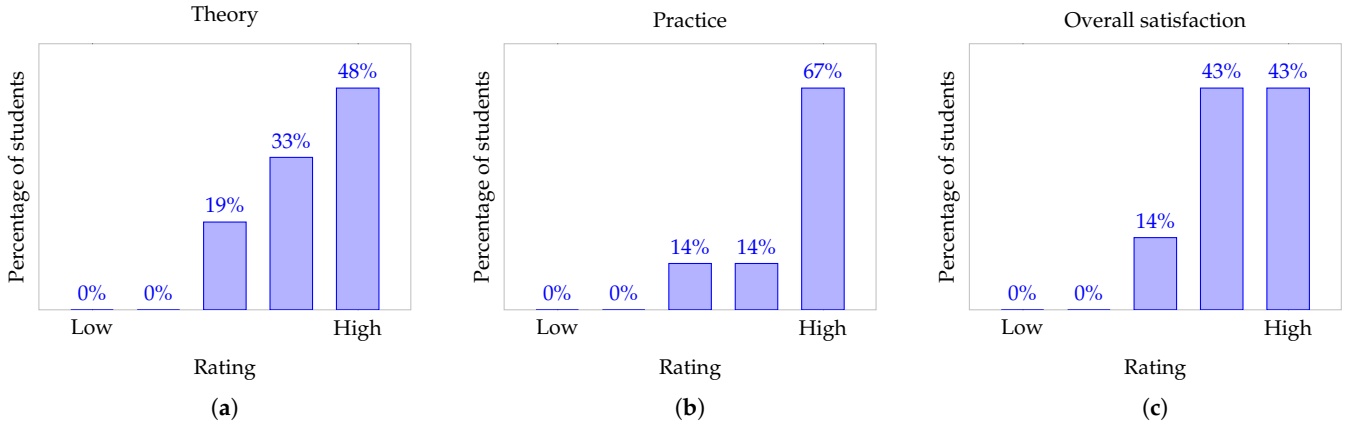



**Figure 5.** Student ratings about the additional GUI design elements: (**a**) appreciation of the GUI elements in the course; (**b**) appreciation of the GUI elements in the laboratory assignment; (**c**) overall satisfaction level for the GUI design classes.

According to these results, while the introduction of GUI design and programming was perceived as an additional difficulty, most students welcomed them and were satisfied with the outcome by the end of the semester. The practical parts were somewhat better regarded

than the theory, but none of the students expressed disagreement or dissatisfaction with the additional course matter, even if they reported higher difficulty and longer accommodation to the new concepts.

In addition to rating the various elements of the course, students also provided open feedback on their opinions. Many of the students just expressed their preference for this new matter, for example:

- "*I liked programming in Embedded Wizard on the GUI side.*"
- "*I really liked GUI programming, it is very intuitive.*"
- "*Very interesting, it is something new to me and is very easy to use.*"
- "*It is more interesting and easier, but more complex*"

Usually, the interest was seen for the more practical and interactive approach of GUI programming, as a few other students noted explicitly:

- "*For me it really looked interesting to work in this program, because I learned something different and new. I liked that we did something practical.*"
- "*It was a very dynamic and interactive course. Within the team, we made a completely functional program from 0, which is really motivating and interesting to understand how an application works and how it is created.*"
- "*Overall, the project based on Embdedded Wizard was very pleasing and interactive.*"

Also, the end result of their project with their own designed GUI was well regarded:

- "*I liked the end result, I learned how to make a graphical interface.*"
- "*It was a project on which I was able to work with pleasure, succeeding to view it intuitively in real time.*"

Working in Embedded Wizard itself had also been considered an incentive:

- "*I liked to work in Embedded Wizard, I learned a lot of new things and teamwork was a really good idea.*"
- "*I understood the functioning of Embedded Wizard, I liked much to program in this environment.*"
- "*I like using Embedded Wizard and I would like to work on GUI in the future.*"
- "*I liked this programming language, but it is very complex.*"

One recurring point from students, who detailed more why they like GUI design in embedded programming, was to recommend more such elements, either by having some dedicated classes for it, having more laboratory sessions, or having access to it earlier.

- "*I really liked to work on, simulate, and test applications with Embedded Wizard. I have discovered a new and interesting programming language, Chora. I have learned what the processes are through which a product arrives to the user. Starting from specifications, design, programming, and up to testing and implementation. It is a shame that we didn't have more laboratory hours.*"
- "*I was interested in GUI programming, this course offered us the opportunity to learn about this branch of programming so widespread in our days. In my opinion, this matter should already be taught by the second or the third year.*"
- "*This matter should be taught in the third year and split over two semesters, one full semester for GUI design and another for practical implementation on the board.*"

Here we have to note that this course is in the fourth year, and it was the first time they had an encounter with graphics (besides the first-year technical drawing course, which cannot really be considered as artistic drawing).

A couple of students expressed the initial difficulties they found when adopting the new methods, but this almost always resorted to the GUI programming tools themselves:

- "*A little hard to adapt to Embedded Wizard, but this is normal when the work environment changes. After a little time spent with it, the process becomes easier.*"
- "*It is not an easy tool to use, it is very different from what we have learned so far, but it is very interesting.*"
- "*I consider that this matter helped us a lot, especially for the future, because we could learn how to make the connection between hardware and software. From my point of view, the disadvantage of using programs like Embedded Wizard is that tutorials available on the Internet about this sort of stuff are rather scarce. Even so, we had luck that at the course these things were presented to our understanding.*"

As noted by the last student, much of the content from the edge between artistic tools and programming tools, and most notably the Embedded Wizard IDE itself, has a lower presence in mainstream tutorials for engineers, so this also created additional perceived difficulty for some students, who were accustomed to learning from YouTube tutorials.

Some other students did well despite these difficulties:

- "*It seemed an interesting idea for me to use this environment because I could do something different. Even if it is the first time I used it, I think that I have managed very well.*"

These observations also correlate well with the difficulty and satisfaction ratings analyzed earlier. Students found the design and artistic elements more difficult initially compared to the more familiar programming concepts, but after getting accustomed to them, they really started to like them, as indicated by the higher satisfaction levels.

Finally, we also have some opinions comparing GUI programming (with everything involved) to another (introductory) embedded programming course, where students were programming Arduino boards:

- "*It is an interesting environment for application development, attractive, but GUI programming is more cumbersome than Arduino programming.*"
- "*It is OK, there is support on the Internet. Personally, I like more GUI programming than Arduino.*"

So, first it is noted that it is a more difficult task to perform a complete GUI development compared to the more entry-level Arduino, but another student still expressed higher interest in this type of programming. Someone else also noted that accommodation to this method can vary from student to student. At least some of them still found it easier to design a GUI:

- "*It is very intuitive, I liked to work in this environment. I felt just like when I'm programming in Python: anything I wanted, I managed easily and efficiently. Maybe for an Arduino programmer it wouldn't be so easy to accommodate, but for me it was.*"

Although there is a clear lack of similar studies in the context of Embedded Systems, some existing studies regarding the STEAM approach in university settings point to similar results: students perceive STEAM techniques to have a positive impact on their studies and wish to add them to more courses (M. Singh et al., 2024).

Overall, we can conclude from these observations that the students really liked the addition of GUI design, drawing, and creating designs that they could also try out on a real hardware implementation. The programming for animations and user interactions was also regarded well compared to other low-level microcontroller programming activities. In the end, the students do recommend adding such elements to other engineering courses as well.

## 4. Conclusions

Integrating arts education into engineering courses has two-fold benefits for students. First, it increases student motivation, gives them a wider perspective, and boosts creativity. The survey results indicated that students were thrilled by the opportunity to also learn some artistic elements and many reported increased satisfaction. Second, it offers training

for aspects that are in increasing demand from industry, and better interaction between engineers and designers. Using digital drawing tools and familiarizing oneself with basic concepts in Arts allows a more profound understanding of design solutions, and experience with tools linking the graphics with the low-level programming on resource-limited equipment allows better feedback from the engineers to the designers for finding better compromises between user experience and technical limits.

Based on the feedback from the students, the STEAM approach presented in this paper proves to be moderately challenging, as it brings both student and teacher out of their comfort zone, but by the end of the course, the overall satisfaction with these new artistic elements is quite high. Artistic elements in the form of GUI design were well received by the students, and they even expressed their interest for such elements to be added to other courses as well.

However, the feedback survey used in this paper is quite limited as it just offers some general perceptions of the students, and it is based on voluntary participation and a relatively small sample size. Future research should involve larger cohorts and examine how prior artistic experience influences the outcomes, as well as how STEAM integration correlates with academic performance.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** The study was conducted in accordance with the Declaration of Helsinki, and approved by the Ethics Committee of Transilvania University (10430, 31 July 2025).

**Informed Consent Statement:** Informed consent was obtained from all subjects involved in the study.

**Data Availability Statement:** The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| 2D | Two-Dimensional |
| ADC | Analog-to-Digital Converter |
| DMA | Direct Memory Access |
| GPIO | General Purpose Input–Output |
| GUI | Graphical User Interface |
| $I^2C$ | Inter-Integrated Circuit |
| IDE | Integrated Development Environment |
| LED | Light-Emitting Diode |
| MCU | MicroController Unit |
| SPI | Serial Peripheral Interface |
| STEAM | Science, Technology, Engineering, Arts, Maths |
| STEM | Science, Technology, Engineering, Maths |
| UART | Universal Asynchronous Receiver Transmitter |
| UI | User Interface |
| UX | User eXperience |
| WYSIWYG | What You See Is What You Get |

## References

Adnan, A.-R., Mohamed, A.-F., Tarek, A., Mun, S., & Hosny, H. (2016). Measuring student satisfaction with performance enhancement activities: Evidence from business education. *International Journal of Information and Education Technology*, *6*(10), 741–753. [CrossRef]

Aguilera, D., & Ortiz-Revilla, J. (2021). STEM vs. STEAM Education and student creativity: A systematic literature review. *Education Sciences*, *11*(7), 331. [CrossRef]

Banach, P., & Schweyer, M. (2024). *Embedded wizard manual.* Available online: https://doc.embedded-wizard.de/ (accessed on 10 December 2024).

Barr, M. (2006). *Programming embedded systems with C and GNU development tools: Thinking inside the box* (2nd ed.). O'Reilly Media.

Breiner, J. M., Harkness, S. S., Johnson, C. C., & Koehler, C. M. (2012). What is STEM? A discussion about conceptions of STEM in education and partnerships. *School Science and Mathematics*, *112*(1), 3–11. [CrossRef]

Connor, A. M., Karmokar, S., & Whittington, C. (2015). From STEM to STEAM: Strategies for enhancing engineering & technology education. *International Journal of Engineering Pedagogy (iJEP)*, *5*(2), 37–47. [CrossRef]

Cook, P. R. (2016). Adding art to STEM. *Communications of the ACM*, *59*(10), 8–9. [CrossRef]

Deák, C., & Kumar, B. (2024). A Systematic review of STEAM education's role in nurturing digital competencies for sustainable innovations. *Education Sciences*, *14*(3), 226. [CrossRef]

Díaz-Obregón, R., Nuere, S., D'Amato, R., & Islán, M. (2019). Strengthening the interaction of art and science through rubric-based evaluation models: The final degree project of the dual degree of engineering in industrial design and mechanical engineering. In J. García-Prada, & C. Castejón (Eds.), *New trends in educational activity in the field of mechanism and machine theory* (pp. 141–148). Springer International Publishing.

Gibson, R. (2020). STEM into STEAM. In *Transforming the curriculum through the arts* (pp. 199–220). Springer International Publishing. [CrossRef]

Hammel, M. J. (2012). *The artist's guide to gimp: Creative techniques for photographers, artists, and designers (covers gimp 2.8)* (2nd ed.). No Starch Press.

Hardiman, M. M., & JohnBull, R. M. (2019). From STEM to STEAM: How can educators meet the challenge? In A. J. Stewart, M. P. Mueller, & D. J. Tippins (Eds.), *Converting stem into steam programs: Methods and examples from and for education* (pp. 1–10). Springer International Publishing. [CrossRef]

Haroutounian, J. (2019). Artistic ways of knowing: Thinking like an artist in the STEAM classroom. In A. J. Stewart, M. P. Mueller, & D. J. Tippins (Eds.), *Converting stem into steam programs: Methods and examples from and for education* (pp. 169–183). Springer International Publishing. [CrossRef]

Henriksen, D. (2014). Full STEAM ahead: Creativity in excellent STEM teaching practices. *The STEAM+ Journal*, *1*(2), 15. [CrossRef]

Kanwar, A., & Sanjeeva, M. (2022). Student satisfaction survey: A key for quality improvement in the higher education institution. *Journal of Innovation and Entrepreneurship*, *11*(1), 27. [CrossRef]

Land, M. H. (2013). Full STEAM ahead: The benefits of integrating the arts into STEM. *Procedia Computer Science*, *20*, 547–552. [CrossRef]

Marmon, M. (2019). The emergence of the creativity in STEM: Fostering an alternative approach for science, technology, engineering, and mathematics instruction through the use of the arts. In M. S. Khine, & S. Areepattamannil (Eds.), *Steam education: Theory and practice* (pp. 101–115). Springer International Publishing. [CrossRef]

Marques, D., Neto, T. B., Guerra, C., Viseu, F., Aires, A. P., Mota, M., & Ravara, A. (2023). A STEAM experience in the mathematics classroom: The role of a science cartoon. *Education Sciences*, *13*(4), 392. [CrossRef]

Piro, J. (2010). Going from STEM to STEAM. *Education Week*, *29*(24), 28–29.

Richardson, J. T. E. (2005). Instruments for obtaining student feedback: A review of the literature. *Assessment & Evaluation in Higher Education*, *30*(4), 387–415. [CrossRef]

Romanca, M., & Ogruțan, P. (2011). *Sisteme cu calculator încorporat: Aplicații cu microcontrollere*. Editura Universității Transilvania.

Samaniego, M., Usca, N., Salguero, J., & Quevedo, W. (2024). Creative thinking in art and design education: A systematic review. *Education Sciences*, *14*(2), 192. [CrossRef]

Singh, M., Azad, I., Qayyoom, M. A., & Khan, T. (2024). A study on perceptions and practices of STEAM-based education with university students. *Social Sciences & Humanities Open*, *10*, 101162. [CrossRef]

Singh, T. (2021, March 3–6). *Importance of digital arts in interdisciplinary context*. 2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering (pp. 43–48), Cha-am, Thailand. [CrossRef]

STMicroelectronics. (2020). *Discovery kit for STM32F7 series with STM32F746NG MCU* [Computer software manual]. Available online: https://www.st.com/resource/en/user_manual/um1907-discovery-kit-for-stm32f7-series-with-stm32f746ng-mcu-stmicroelectronics.pdf (accessed on 17 January 2025).

Wong, W. K., & Ng, P. K. (2016). An empirical study on e-learning versus traditional learning among electronics engineering students. *American Journal of Applied Sciences*, *13*, 836–844. [CrossRef]

Zaher, A. A., & Hussain, G. A. (2020, April 27–30). *STEAM-based active learning approach to selected topics in electrical/computer engineering*. 2020 IEEE Global Engineering Education Conference (EDUCON) (pp. 1752–1757), Porto, Portugal. [CrossRef]

Zhbanova, K. (2019). Developing creativity through STEM subjects integrated with the arts. *Journal of STEM Arts, Crafts, and Constructions*, *4*(1), 1.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.