Tiny Trainable Instruments

by

Aarón Montoya-Moraga

B.S., Pontificia Universidad Católica de Chile (2014)
M.P.S., New York University (2017)

Submitted to the Program in Media Arts and Sciences
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Program in Media Arts and Sciences
August 20th 2021

Certified by .....
Tod Machover
Muriel R. Cooper Professor of Music and Media
Thesis Supervisor

Accepted by .....
Tod Machover
Academic Head, Program in Media Arts and Sciences

Tiny Trainable Instruments

by

Aarón Montoya-Moraga

Submitted to the Program in Media Arts and Sciences
on August 20th 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Can we build flexible and reusable multimedia instruments that are trained instead of programmed? How can we build and publish our own personal databases for artistic purposes? What are the new choreographies and techniques that machine learning running on microcontrollers offer for artists and activists?

Tiny Trainable Instruments is a collection of multimedia devices, running machine learning algorithms on microcontrollers, for artistic purposes. It includes techniques for capturing data, building databases, training machine learning models, and deploying on microcontrollers. The software library created for this project allows for the creation of instruments that react to different inputs, including color, gesture, and speech, to control different multimedia outputs, including sound, light, and movement, using machine learning and embedded sensors.

This thesis emphasizes open source software and artificial intelligence ethics, and includes all the steps on creating these bridges between machine learning and media arts, that are respectful of privacy and consent because of their offline and off-the-grid nature.

Thesis Supervisor: Tod Machover
Title: Muriel R. Cooper Professor of Music and Media

Acknowledgments

thanks to

aaron, agnis, aillaly, alejandra, alexandra, alicia, allison, alonso, anamaría, ana maría, andrea, andreas, andrew, ava, baltazar, beatriz, belén, benjamín, bernardita, braulio, camila, camilo, carla, carmelo, carolina, casey, catalina, charles, christian, claudia, constanza, corbin, cynthia, daniel, daniela, daniella, dano, devora, dorothy, erik, euge, eve, evelyn, felipe, fernanda, francesca, francisco, gabriel, gabriela, gaurav, gene, guillermo, hannah, ignacia, ignacio, isa, jasmine, javiera, jen, jennifer, jesenia, joann, joaquín, john, jorge, josé, juan, julian, juniper, justin, karina, karsten, kat, kathy, katya, kevin, lauren, lee, lily, lindsey, lisa, luis, luisa, luna, madelaine, manaswi, marco, margarita, maría josé, martin, maxwell, mel*, mitchel, moisés, monica, namira, natalia, natalie, nathier, newén, nicolás, nicole, nikhil, nina, nouf, nushin, olivia, pablo, patricio, paula, pedro, peter, priscilla, qianqian, rachel, rafael, raúl, rebecca, rébecca, renata, ricardo, rodrigo, rosalie, rox, roy, roya, russell, ruta, sam, samuel, sankalp, sasha, sayén, sean, sebastián, sejo, sergio, sharon, shawn, shir, sofía, sokio, soledad, sumedha, tigran, tiriree, tk, tod, tom, tyler, verónica, víctor, victoria, viniyata, will, wipawe, yuan, yining, yuli, yusuf, zach.

Contents

Glossary

baud unit of measurement of symbols per second. 80, 82

commit saving changes on a Git repository. 69

fork clone a repository and make your own version. 58, 81

Git a version control system. 35

issue a comment, question, or suggestion on a Git repository. 69

pull request proposing new commits with new features or fixes to a Git repository. 69, 87, 137

scraping computational technique to extract data from human-readable output. 37

synthesynthesis synthesis of synthesizers. 56

Acronyms

AI artificial intelligence. 18, 21, 24, 32, 36, 37, 42, 43, 62, 98, 101

BLE Bluetooth Low Energy. 15, 41, 67, 70, 103, 107

CAMIT Council for the Arts at MIT. 93

CFAA Computer Frad and Abuse Act. 59

COCO Common Objects in Context. 37

COUHES Committee on the Use of Humans as Experimental Subjects. 93

DIN Deutsches Institut für Normung. 79

DIY do it yourself. 18, 55, 67

DMCA Digital Millenium Copyright Act. 59

IDE integrated development environment. 69, 70, 87, 96

IMU inertial measurement unit. 74

ITP Interactive Telecommunications Program. 30–32, 35, 36, 39, 41

k-NN k-nearest neighbors. 36, 43, 72, 97

LED light-emitting diode. 68, 78

MIDI Musical Instrument Digital Interface. 30, 50, 68, 78

MIT Massachusetts Institute of Technology. 93

ML machine learning. 18, 19, 21, 23–25, 32, 34–36, 39, 41–43, 58, 59, 63–65, 67, 68, 72, 74, 76–78, 83, 86, 92, 94, 98–105

NYPL New York Public Library. 38

NYU New York University. 30–32, 35, 36, 39, 41

PCB Printed circuit board. 49, 104

Pd Pure Data. 52–54

PDM pulse-density modulation. 76

PIC Programmable Intelligent Computer. 27

PWM pulse-width modulation. 78, 83

RGB red green blue. 64, 72

USPS United States Postal Service. 96

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

This thesis is the capstone project of my Master's program between the academic years 2019-2021 in the Media Arts and Sciences program at the MIT Media Lab, where I am a Research Assistant in the Opera of the Future and Future Sketches groups.

This thesis is a collection of media arts instruments made with microcontrollers and machine learning (ML), with a strong emphasis on artificial intelligence (AI) ethics and do it yourself (DIY) methods. Its primary audience - beyond academia - is beginners and artists, and it is my hope that this work can inspire a new generation of instrument makers, artists, designers, educators, programmers, policy makers, activists, and enthusiasts.

ML has several barriers of entry, including cost, complexity, and difficulty. Present day industrial ML relies on proprietary software and hardware, as a result of ML models aim for high precision and thus need to be trained for long periods of time, using expensive non-open computational resources, with huge datasets that often are

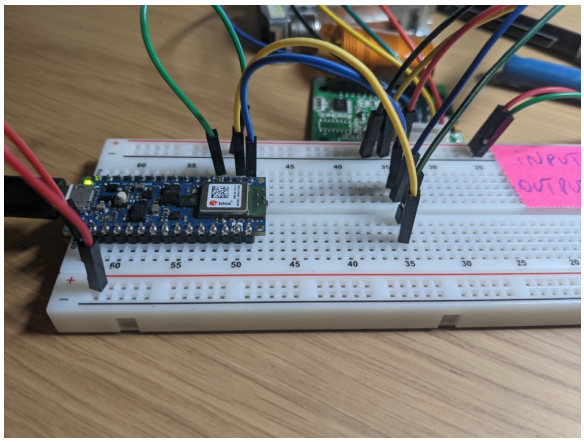


Figure 1-1: Early prototype of Tiny Trainable Instruments
Picture taken by myself

scraped from the internet without the explicit consent of users, and a byproduct of surveillance capitalism.

Figure 1-2: Surveillance camera in a park in Boston MA
Picture taken by myself

The release of the library Arduino TensorFlow Lite Micro in late 2019 [1] i *tiny ML field*, a subset of ML that focuses on hardware and software able to run calculations both on-device and with low power, a stark contrast to many industrial ML applications. The tutorials published by Arduino [2] showed how to build your own database to detect colors and gestures using a microcontroller, and this was an inspiration to include the in this thesis to make an artistic exploration of these emerging techniques.

One aspect that deeply resonated with me was **data agency**, and being in control of your own data, in particular its capture, storage, publication, and use. In this thesis I propose the microcontroller as a way of building our own databases and deploying our models to bypass corporate or government surveillance. During this year and more of pandemic lockdown context, I have found myself several times working alone in my room, capturing data of myself and my living environment, and then building databases for other people to use, in a way that reminds of one of my favorite artists and activists, Ai Weiwei, who in 2012 - while facing government surveillance - decided to livestream from his house [3]. I also see this today as a way of reclaiming data agency.

Figure 1-3: Weiweicam, by Ai Weiwei, 2012
Retrieved from [3]

I am an underrepresented minority in the USA, and it often happens that supposedly automatic neutral technologies fail to detect me. Here is an example of the popular software Zoom, where I spoke out loud: "This is a test to show that Zoom speech to text transcription does not work with my voice because of my accent." Indeed the words "Zoom" and "voice" were transcribed as "soon" and "boys", respectively.

Here is a further experiment with more unusual words, the names of the members of my thesis committee: Tod Machover, Mitchel Resnick, and Zach Lieberman, where the transcription had even more errors.

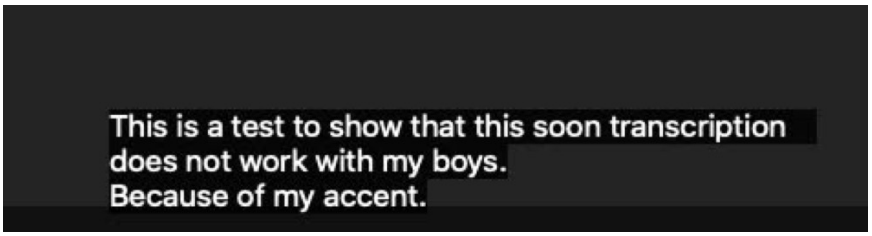


Figure 1-4: Screen capture of speech-to-text on Zoom, introduction
Screen capture by myself

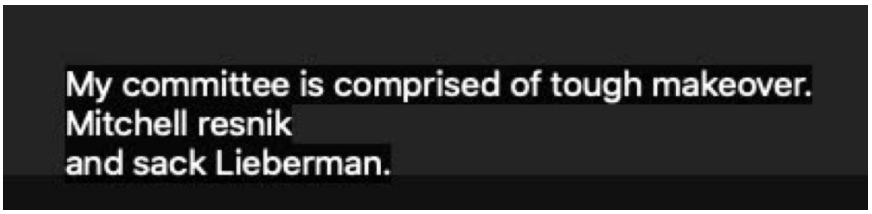


Figure 1-5: Screen capture of speech to text on Zoom, committee
Screen capture by myself

Despite these ML algorithms being promoted by corporations and governments as unbiased and effective, most probably these algorithms have never been exposed to Chilean people with my accent, so they fail in transcribing my message. Because of these errors, I routinely turn off voice assistants and text completion in my devices (if you have interacted with me over text, I typed every single character :)).

These algorithmic decisions can easily have devastating effects in equity and discrimination. A famous example of this was the scrapped internal recruiting tool that Amazon developed with AI, which systematically discriminated against women applicants, thus in fact reproducing and amplifying the existing biases of their own hiring teams[4]. Researcher Timnit Gebru has also published about the dangerous bias and ecological impact of recently released language models [5].

Sadly, we often cannot turn off or opt out of these computational or AI systems. We should in fact be able to, because there is even more at stake than losing our privacy! We could be subjected to harmful **algorithmic bias**, as the Algorithmic Justice League [7] explains on their website:

"In today's world, AI systems are used to decide who gets hired, the

Figure 1-6: Meme about biased data
Retrieved from [6]

> quality of medical treatment we receive, and whether we become a suspect in a police investigation. While these tools show great promise, they can also harm vulnerable and marginalized people, and threaten civil rights. Unchecked, unregulated and, at times, unwanted, AI systems can amplify racism, sexism, ableism, and other forms of discrimination."

I highly recommend watching the Coded Bias documentary [8] currently available on Netflix, to inform oneself about the important and necessary digital advocacy of the Algorithmic Justice League, who have helped me learn the language for navigating these important topics of civil rights.

The final catalyst that led me to this thesis happened a year ago, when I watched a video [9] of a conversation between artists Rafael Lozano-Hemmer and Dorothy Santos. At 36:28 in the video, Rafael says "Face recognition needs to be banned in all applications except art."

This was the perfect spark for starting to work on this project, inspiring me to make more accessible these computational techniques to artists, beginners, enthusiasts, and educators. I think it's crucial for our civil rights and advancing the discourse, that artists experiment with ML in a critical way. Despite the creative and artistic applications that I show during this thesis, and how ML becomes cheaper and more pervasive, it is not the solution to all problems, and might not even be needed in many scenarios.

Figure 1-7: Meme about need of machine learning
Retrieved from [10]

In this 21st century artists have unprecedented access to tools for making more new tools and instruments, and I intend this thesis to be a foundation for a new generation of instrument makers for manipulating audiovisual material, using ML. This approach is really exciting because it allows beginners and artists to train their instruments instead of programming them, by inputting data for tuning instead of having to

write lines of code, and then fixing thresholds for changing their behavior, working off-the-cloud in a decentralized way.

1.2 Objectives and dreams

Infrastructure and cooperation are key to society. I love how I can bike on roads and hike on paths that were built as shared resources for the benefit of everyone. I hope this thesis work can be adopted by fellow artists and educators to create new instruments for arts, and to inspire critical and ethical thinking about AI.

There is a significant educational value in creating your own databases, and in this thesis I propose techniques and software so that people can do this. By building databases with their own data, people will be able to train their own custom ML models that are tailored to themselves, instead of using external databases and inheriting their biases. Also I hope this work will help people appreciate the craft involved in making databases, and raise awareness about the exploitation and problematic lack of consent behind them.

Finally, I want to highlight all the legal documents and regulations that I navigate on a daily basis: as a non-U.S. citizen on a student visa I cannot freelance, as a software programmer I have to comply with various licenses, and as a musician I am still learning about the different ways to publish covers and sampling other artists. Some of the most pervasive documents that we encounter are the terms and conditions of the services we use. As an anecdote, in 2010, GameStation added a soul clause to theirs as an April Fool's prank, making them legal owners of thousands of customers' souls [11]. A study in 2007, it was estimated that people would need around 250 hours every year to actually read the privacy policies shown to them [12]. That is why in this thesis I have tried my best to cite every work that I am either using as a building block, or that served as a direct or indirect inspiration.

I hope this work can be adopted and iterated by others, through building a new generation of private and smart devices, like one of my first goals, a drum machine I can talk to, and that would let me ask for different rhythms mid-performance, or let me use my body gestures to write poems.

1.3 Thesis outline

1. Chapter 1 - Introduction: the context and summary.
2. Chapter 2 - Early experiments: media arts education, microcontrollers, and ML, among others.
3. Chapter 3 - Background and inspiration: research about work by other people which has informed my practice.
4. Chapter 4 - Tiny Trainable Instruments: design strategies for the software and hardware, description of the support team working on this thesis.
5. Chapter 5 - Project evaluation: user feedback, field notes.
6. Chapter 6 - Conclusions and future work: next iterations of the instruments, and their proposed use for educators and artists.

Chapter 2

Early experiments

In this chapter I showcase early experiments in the different fields I worked on for this thesis, with a strong focus on the different aspects of my education and artistic practice. I include many of my breakthroughs and pitfalls that eventually led me to working on this thesis, in an open, detailed, celebratory, and critical narrative.

2.1 Learning microcontrollers

I learned how to program microcontrollers around 2010 in Chile as an undergraduate student of electrical engineering. I was taught how to program PIC microcontrollers with Microsoft's tools, including the operating system Windows and the C# programming language. Around the same time, with my classmate Braulio we made our first project with an Arduino Uno microcontroller. It consisted of a robotic guitar tuner, where the Arduino detected the pitch of a string, analyzed the frequency, and then made a servo motor move the tuning gear of the string, in order to match the desired pitch.

Fast forward to 2013. For my undergraduate thesis I had to complete a capstone

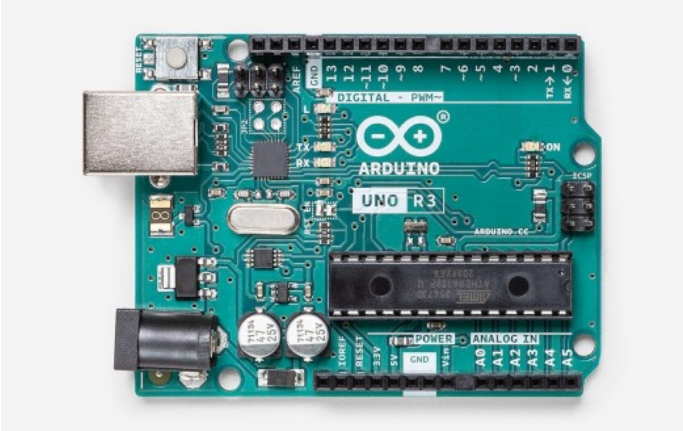


Figure 2-1: Arduino Uno microcontroller
Retrieved from [13]

project and implement many low-level programming techniques, and Arduinos were not allowed because they were considered a shortcut. For this thesis I worked with my classmate Guillermo, and we built a robotic device with a PIC microcontroller programmed with C#. Our code was very specific to that particular chip and project, and hardly reusable or interesting for a wider audience.

Since graduation I haven't programmed with PIC microcontrollers. I realized I wasn't excited about making one-off devices with non-reusable code that I couldn't share, or with having to use proprietary or bulky interfaces for writing or deploying my code. In contrast, Arduinos became a huge part of my practice, because of their low cost, open source nature, ease of programming and uploading the code with a generic USB cable, and because of the enormous and growing available documentation and user contributed libraries, which now also include the TinyTrainable library developed for this thesis project.

The Arduino ecosystem fostered many aspects of my practice, and now looking at it in retrospect, **I realize that what I love is not computers but computation**, making small machines that can crunch numbers for art, and making them openly and in collaboration with friends.

A huge argument that I see people on the internet making against Arduino micro-

controllers, is that they are too expensive (around 30.00 USD) in comparison to the 1/10th of the cost of the bare bones chips and parts you need to build your own microcontroller unit. I am against this argument, since it invisibilizes the effort put in documentation by the Arduino community, and it also assumes that everybody is comfortable soldering and has advanced electronics degrees. Even if you factor in the cost of hours you need to make your own, it doesn't pay for itself. Obviously, I see the educational advantage of building your own things, but it's a slippery slope and it is discriminatory to think that if you build a project with an Arduino, it's not good because it's not from scratch. In fact, I think it's rude to say it's not worth it, as rude as complaining about someone buying bread and making fun of them for not buying the ingredients and cooking it themselves.

Another strong argument against Arduino microcontrollers is about efficiency: that Arduino is too high-level and over-bloated, and that applications and projects could be faster if you programmed in a lower-level language. I also see this argument when comparing different programming languages for graphic arts. Once again, knowing how to program with lower-level programming languages is amazing, and there is value and fun on it. But in my practice, I put a high value on my time and energy, and I'd rather make my code slower or less efficient so I can spend more time making art or resting.

I'd rather that we all have non-painful programming experiences, so that we can spend more time away from screens and making more art!

2.2 Computer music and physical computing

During undergrad studies, I took classes and did research with professors and computer musicians Rodrigo Cádiz and Patricio de la Cuadra. With them I learned the fundamentals of computer music, including languages such as Max and Pure Data,

which I still use to this day. For a class project I created a spoon synthesizer with masking tape, cardboard, and a Makey Makey [14], a device created in 2010 by Eric Rosenbaum and Jay Silver from the MIT Media Lab's Lifelong Kindergarten research group. This was my first hands-on introduction to physical computing, as a way of building my own custom playful interfaces for manipulating sound with computers.

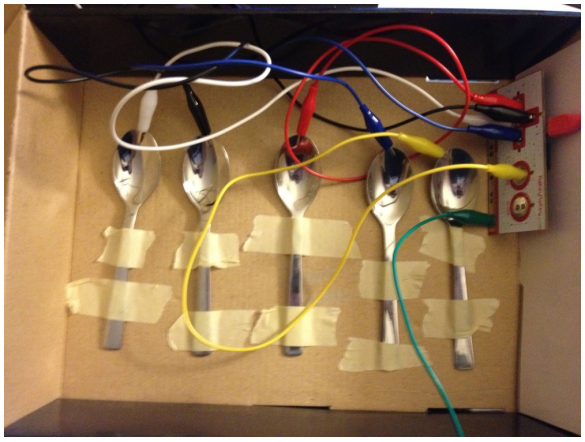


Figure 2-2: Spoons and Makey Makey synthesizer
Picture taken by myself

For this project I applied my practice as a guitar player, where I am constantly mixing and matching different devices on my pedalboard. This inspired me to make this flexible synthesizer with objects lying around me: spoons, masking tape, and cardboard. They also are forgiving, and I was able to change their physical position. The Makey Makey acted as an interface with the computer, where I could assign on the fly different sounds to each spoon.

In retrospect, this project was an amazing experience that followed Mitchel Resnick's 4 P's: it was a Passion Project that I built for Playing with my Peers, which is a constant in my artistic practice.

2.3 Physical computing and more microcontrollers

After graduation in 2013 I freelanced as a software and technology designer and developer for artists. I learned computer protocols and networks, and wrote custom software for live multimedia theater and music shows. Realizing that I wanted a bigger community of people to learn media arts with, I applied to New York University's Interactive Telecommunications Program (NYU ITP), where I joined as a graduate student in 2015. In my first semester, I took the class Introduction to Physical Computing [15], taught by one of Arduino's co-creators, Tom Igoe. Since I was already familiar with electronics and circuits, I focused on learning interface design, human computer interaction, open source hardware and software, and physical computing education.

During my research I was introduced to a wider ecosystem of microcontrollers beyond Arduino, that were made possible because of its open source nature and the community behind it. My favorite example is the Teensy by PJRC [16], which captivated me by two main features: it can send and receive MIDI over USB, and it has a powerful audio library that allows you to play audio samples, create effects, and process real time audio. With Teensy, I was able to make standalone projects in a way that before would have required me to use a full-fledged computer.

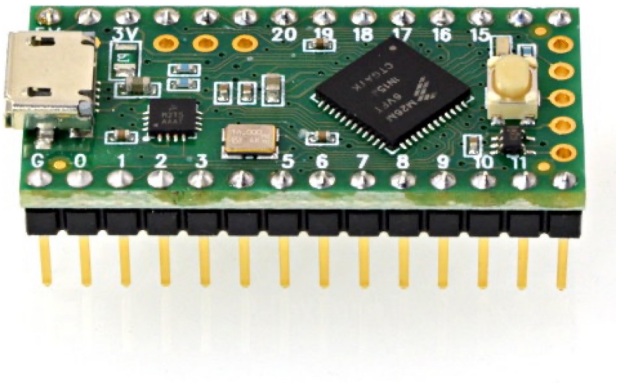


Figure 2-3: PJRC Teensy LC microcontroller with pins
Retrieved from [17]

2.4 Processing, p5.js, Processing Foundation

Processing [18] is an open source software written in Java, and it was started at the MIT Media Lab's Aesthetics and Computations group by Ben Fry and Casey Reas. Over the years I have learned Processing on my own, and through it, computer graphics and interactivity.

I was excited to learn more Processing in 2015 in my first semester at NYU ITP's Introduction to Computational Media class, like it had been taught for around a decade. I consider myself lucky, because that year they replaced the curriculum in Processing with a new one based on p5.js, another software library supported by the Processing Foundation.

p5.js was created by Lauren McCarthy as a reinterpretation of Processing, using JavaScript instead of Java, to make art that runs on web browsers and over the internet. With p5.js I learned artistic and creative ways to program web applications, and this experience led me to focus on educational outreach with the Processing Foundation, which I will mention in the next section.

I was not only in awe of p5.js and Processing, but also of the people and the communities behind these software libraries. I took Lauren McCarthy's Performing User class [19] at NYU ITP, and it was my favorite class during my Master's there. She fostered a safe space for us to explore societal and political implications of software, and it led me to add more of my personal convictions, and my body, to my artwork. This class inspired me to make *its-ok-to-die* [20]: a self portrait, with a countdown timer to my projected death time according to the United Nations' estimates. The first version I did was a Processing app running on a Raspberry Pi computer, and I also ported it to p5.js for web access.

Figure 2-4: its-ok-to-die, on a browser with p5.js
Screen capture by myself

2.5 Teaching media arts

While I was a student at NYU ITP, I was the recipient of two summer grants to work on an internationalization project of p5.js. Since my native language is Spanish, and I wanted to share all my passion for web programming with p5.js, I focused on translating the official p5.js website [21] and book [22] to Spanish, and teaching introductory workshops in Spanish in my home country, Chile. My main goal was to let people be able to learn and enjoy p5.js without having to first learn English. Since then, this project has been expanded to other languages, and I am still a contributor to p5.js.

Another reason why I wanted to teach and share p5.js is because still to this day their community statement [23] is one of my favorite documents, fostering an inclusive environment for artists. This document inspired the ml5.js community statement [24], a ML library for arts created at NYU ITP with a strong emphasis on AI ethics.

In my workshops and classes, when I have taught p5.js or ml5.js, I like to take a

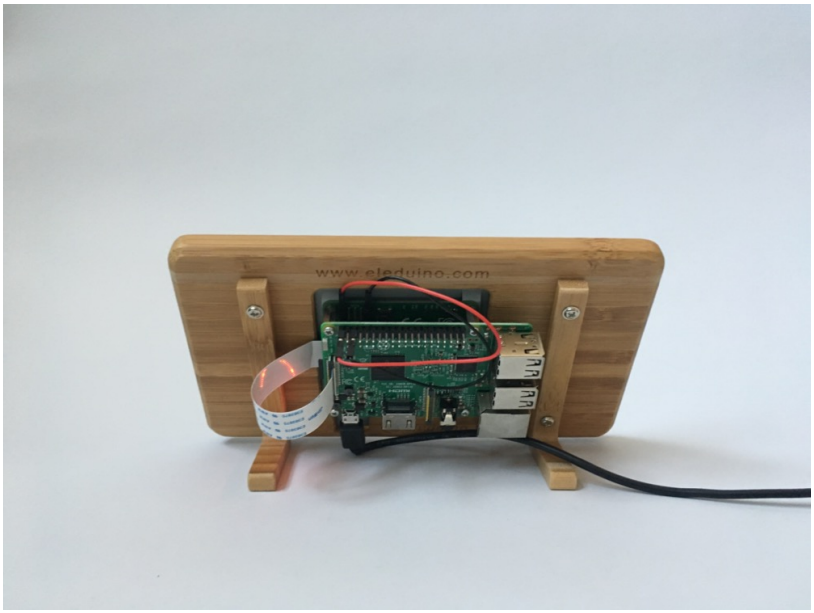


Figure 2-5: its-ok-to-die, on a Raspberry Pi computer
Picture taken by myself

moment with my students to read the community statements, and also to read the source code, and show them the repositories where these libraries are developed. I think it is very educational to take a moment and acknowledge the work made by the community to build these libraries. I also like to point out that these libraries are not made out of scratch (Hennessy Youngman claims that in the arts, we ran out of scratch in the 1960s ([25]). p5.js is a wrapper for drawing and interactivity features of HTML5, and ml5.js is a wrapper of Google's TensorFlow.js library.

Another feature of my classes is that even though I always teach with a particular toolkit, I like to give students an overview of different alternatives that they can use to achieve similar results. I want my students to be able to work in an agnostic way; even if they don't continue using the tools I teach, they learn the fundamentals of programming and arts. Since most of my recent workshops and classes have been short crash courses on complex topics, I make my best efforts to share additional resources with students so that they can continue learning on their own. This is the reason why I enjoy being a contributor and maintainer of documentation of open source projects. It is my way of giving back to the community of volunteers who has

given me access to learning on my own over the years.

I hope that people find the documentation and writing of the TinyTrainable library useful, and that even if they don't end up using it, they learn fundamentals of programming, ML ethics, and how to use the different outputs and protocols I showcase.

2.6 Publishing libraries

After years of working with different software libraries for arts, and publishing my code and teaching with it, I started packaging and writing my own software libraries, with the hope that people can reuse my code and learn from it. My first experiments were in the Python ecosystem, where I have published protestpy [26], a library for creating protest material, and kaputtpy [27], a library for ruining digital devices.

Figure 2-6: protestpy image for protesting against trees
Screen capture by myself

Publishing my libraries so they can be installed with one-click or a command line, and making them depend on other libraries and languages, is for me an exercise in trust, and a way to give back to other artists what I have learned. It's my software version

of building a skatepark or a hiking trail, it's infrastructure so that other people can use it for their own needs, mostly for fun and art :)

Apart from these software libraries, I have made small publishing experiments in other ecosystems, such as Max, Haskell, and Node.js, and the bulk of my work since 2016 is published as open source Git repositories on my GitHub account at `https://github.com/montoyamoraga/`.

Figure 2-7: GitHub contributions
Screen capture by myself

TinyTrainable is my first library for hardware, and this experience has led me to publish libraries and releases for other projects I have been working on during this Master's, which I will continue to build after graduation.

2.7 ML for arts

My first project in ML for arts was in 2016 with my NYU ITP classmate Corbin Ordel, who was a student at Gene Kogan [28]'s ML for Artists class [29]. Together we audited Rebecca Fiebrink[30]'s ML online ML class on the Kadenze platform [31], and learned about her Wekinator [32] platform. As a final project for Gene's class, with Corbin we made the project Piano Die Hard, a digital sculpture consisting of a piano that is watching the trailer for the movie Die Hard, and every time it detects an explosion, it plays music as accompaniment.

The technology stack for this project was very convoluted: a computer looped the trailer of the movie and fed it to an old TV and to an openFrameworks app. This app

Figure 2-8: Piano Die Hard
Retrieved from [33]

outputted statistical descriptors about the color of each picture frame, which were then sent to the Wekinator app, which decided with a k-NN algorithm if the image was either an explosion or not. When it detected an explosion, it sent a control signal to an Arduino microcontroller, which moved a motor connected to a fishing rod and a kitchen utensil, which scratched the strings of the open carcass of an old piano every time there was an explosion.

The training of the model was made beforehand with a selection of 1980's movies, featuring scenes with and without explosions. This project was a direct inspiration for the color input of the TinyTrainable library, also using color data with a k-NN algorithm. It's really fascinating for me to compress all of this complexity of five years ago, needing a computer and different apps, into a single microcontroller. I hope that everybody can enjoy this too!

After graduating from NYU ITP and working for one year as a research resident and graduate teaching assistant, I was exposed to many exciting developments in AI and creativity, including witnessing the creation of the the ml5.js library, and the company RunwayML [34] founded by Cristóbal Valenzuela [35], Alejandro Matamala [36], and Anastasis Germanidis [37]. Through this I became excited about the creative potential of ML, so in 2018 I took a month-long intensive class at the School of

Machines, Making & Make-Believe [38] in Berlin, Germany, facilitated by Gene Kogan and Andreas Refsgaard [39], and organized by Rachel Uwa [40].

During this workshop I got a mix of excitement for the graphic and plastic possibilities that AI allowed me, and frustration while working with databases. The algorithms I was trying to play with needed very specific requirements in terms of their data input, such as specific image resolution, file formats, and file size. So even if I found an existing database, I had to do very tedious labor to adapt them. During this class my work focused on writing scripts to create custom databases with web scraping, which you can find at this repository `https://github.com.montoyamoraga/scrapers`.

I learned web scraping as a student of artist Sam Lavigne. Sam is one of my role models, because he uses computation and comedy to address complex civic topics, including late capitalism, government surveillance and mass incarceration. Not only does he makes amazing artwork, he also creates open source tools and teaches classes so that other artists and activists can learn powerful computational techniques.

Figure 2-9: Sam Lavigne, Training Poses, 2018
Retrieved from [41]

I want to highlight his piece Training Poses (2018) [41], a satirical response to the popular OpenPose [42] library for body detection. Both of these projects use as source material Microsoft's Common Objects in Context (COCO) database, consisting of thousands of images from Flickr, and annotated by workers on Amazon Mechanical Turk. This piece made me aware and worried of the existence of these enormous

databases, and of the potential issues of consent and exploitation. During this research the paper Large Datasets: A Pyrrhic win for computer vision? [43] confirmed my worst suspicions. In this paper, researchers Vinay Uday Prabhu and Abeba Birhane point out that the widely used ImageNet dataset has no consensual images at all!

These concerns have pushed away from creating art with these compromised datasets that don't respect consent. Instead I have opted for making small scale experiments that are educational and that are considered fair use. I have used ml5.js, to make artwork that only runs on the client computer, for privacy. One of such early experiments is Arpillera Mirror (2019) [44], an homage to Chilean multimedia artist Violeta Parra. It consists of a desktop website that applies the style of an arpillera to your computer's camera image, a technique known as style transfer.

Figure 2-10: Arpillera Mirror, 2019
Screen capture by myself

The source images for this project were obtained from the Violeta Parra museum's online collection. My original plan was to use this experiment to pitch two things to the Violeta Parra's museum: that they release their digital collection with permissive licenses, like the Digital Collections from the New York Public Library (NYPL), and to have a style transfer mirror with Violeta Parra's work included in their permanent collection. Since I haven't been able to visit Chile yet, this project is still

unauthorized, and I haven't published the source code.

For people who are interested in the rich field of image generation with ML, I highly recommend the book Making Pictures with Generative Adversarial Networks [45] by Casey Reas, published by Anteism, as of 2021 on its second edition. It's an arts-first book that contextualizes the use of ML algorithms for the creation of images, and uses the metaphor of these algorithms as being similar to the development of the camera. Artists don't necessarily need to understand all the physics or mechanics behind a camera in order to make art with it, but it can definitely help to understand it too. I think that ML is a revolutionary strategy for instrument-making and the arts, while also introducing new civic complexities, and in this thesis I have tried to follow the example of Reas' book, to introduce the technology and contextualize for a new generation of artists and instrument makers.

The last project I want to mention in this section is the class Machine learning for physical computing [46] written and taught in 2020 by software developer Yining Shi [47] at NYU ITP. This class was the first time I saw a project building on top of the recently released library TensorFlow Lite for microcontrollers. This class inspired me to also use this library as a jumpstart for Tiny Trainable Instruments.

Chapter 3

Background and inspiration

3.1 Microcontrollers as alternative to computers

When I joined MIT Media Lab in 2019, I made the decision to focus my efforts on hardware research, so I could make computational art devices, instead of software that needs to run on a laptop computer or a browser. This was fueled by the introduction of restrictive news by Apple, such as advising against the use of apps created by unregistered developers and discontinuing support for 32-bit apps, and by the ever-changing nature of the web, which makes my online artwork break often and need maintenance, and the need for additional corporate infrastructure to keep it online. In contrast I saw hardware as a space where I could deploy my ideas and keep them running without outside intervention.

I started my research by catching up with the newer developments by Teensy. The newer microcontrollers are faster and more powerful, and I used them to design and implement many small projects. I want to mention this personal project, a standalone audiobook I made with rotary encoders and a small screen for navigation, because it led me to include software for support of this same screen in the TinyTrainable

library.

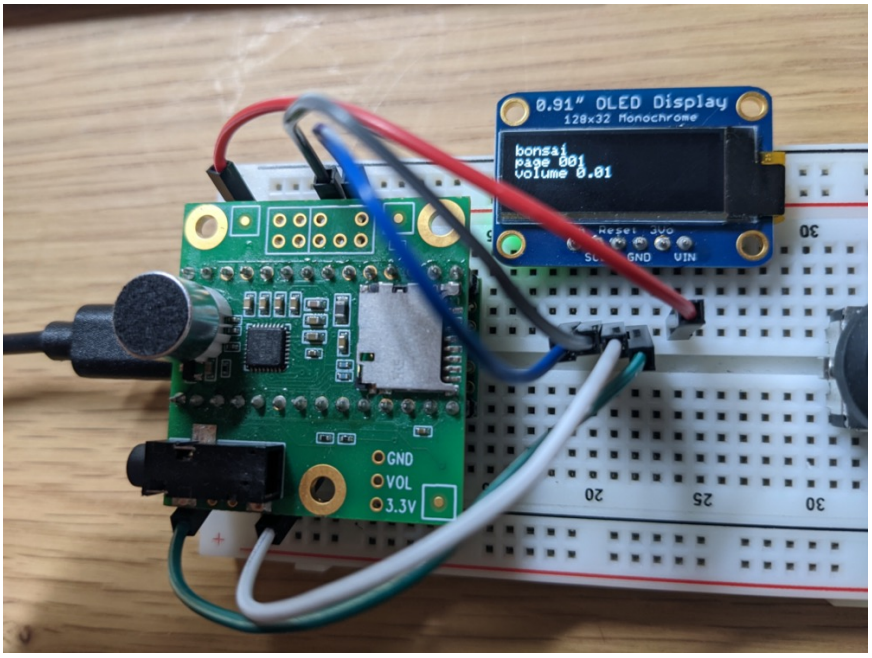


Figure 3-1: Audiobook made with Teensy
Picture taken by myself

In parallel, I researched the evolution of the teaching of physical computing at NYU ITP, and I discovered that they stopped teaching with the now classic Arduino Uno, and have replaced it with the newer series of Arduino Nano microcontrollers, which have a smaller format and and operating voltage of 3.3V instead of 5V.

While browsing this series, I discovered the Arduino Nano 33 BLE Sense that I based my thesis on. It comes with 9 embedded sensors, to measure and detect acceleration, movement, distance, color, plus a microphone. This is an amazing help for beginners, since a huge challenge when you are starting to build your own projects and learning electronics is reading datasheets to understand what sensors you can use with your microcontroller, how to wire them and calibrate them, and then what library supports it. This makes it easier and cheaper to prototype, and this made my work on my thesis easier, since I didn't have to include instructions for wiring sensors or calibrating them, and I could focus on ML and the different outputs for arts.

3.2 Coursework at MIT

As part of the research that directly inspired this thesis, here is the coursework I took at MIT, and my notes about how they inform my theoretical and practical background for Tiny Trainable Instruments.

1. Fall 2019, CMS.901 Current Debates in Media, by Professor Sasha Costanza-Chock
2. Fall 2019, MAS.S65 Recreating The Past, by Professor Zach Lieberman
3. Spring 2020, MAS.826 Sound: Past and Future, by Professor Tod Machover
4. Spring 2020, MAS.712 Learning Creative Learning, by Professor Mitchel Resnick

In the class Current Debates in Media, topics covered included fake news, surveillance, algorithmic bias, data colonialism, climate justice, algorithms of oppression, among others. For my final paper I wrote on the role of the media during the 2019 Chilean protests. This class directly inspired me to look for privacy-first ML, and thinking about AI in a critical way and not as clean safe automation, but as invisibilized labor, exploitation, and algorithmic bias and discrimination in the worst scenarios.

In the class Recreating the Past, I learned about media arts history, and I dived deep into the language C++ which I used for writing the TinyTrainable library, while working with the library openFrameworks, which is one of the most popular open source frameworks and communities for media arts.

In the class Sound: Past and Future, I learned more about the history of different computational advancements for sound, with a strong focus on projects at the MIT Media Lab's own research groups including Opera of the Future, Hyperinstruments, and Music, Mind, and Machine. This class introduced me to many projects and it

helped me decide on making instruments for my thesis, using the latest technologies I could find, in this case, microcontrollers and ML.

In the class Learning Creative Learning, I was introduced to the Lifelong Kindergarten's frameworks and ideas, including the 4 P's (projects, passion, peers, and play), and the design of Scratch as a home with low floor, wide walls, and high ceiling, which I highly recommend to follow up with on Mitchel Resnick's book Lifelong Kindergarten [48]. This class gave me the push to write my software library with a community in mind, starting with the development of it. I made this happen by working with MIT undergrads Peter Tone and Maxwell Wang, who helped me with different aspects of research and development.

3.2.1 TinyML Professional Certificate

Apart from the MIT coursework, between November 2020 and March 2021 I completed the online Tiny Machine Learning Professional Certificate by Harvard University at the platform edx [49]. It is a series of three courses, where they teach responsible AI strategies, theoretical ML, and applied tiny ML with the Arduino Nano 33 BLE Sense microcontroller, wit a focus on industrial applications.

The academic team behind this certificate released the companion Harvard tinyMLx Arduino library [50], which is based on the vanilla Arduino TensorFlowLite for microcontrollers. Through the example of this library, I learned how to build my own library for this thesis.

During this coursework I researched other libraries for machine learning, and discovered the Arduino k-NN library [51]. I included this library as a dependency on my thesis because it allows on-device data capture and model training. This allows for the data never leaving the device, and to enhanced privacy, which is one of the driving concepts of this project.

3.3 Research projects at MIT

Another aspect of my research that I want to highlight is other research projects I worked on while at the MIT Media Lab, because they follow my same interests and passions, and I hope they help inform the process behind my thesis.

3.3.1 SiguesAhi

SiguesAhi [52] (from the Spanish "Are you still there?")is an instrument to detect when oppressive institutions have ceased to exist. It is achieved with microcontrollers with internet connectivity. I built it using a microcontroller from the same series and format but different architecture and capabilities, the Arduino Nano 33 IoT.

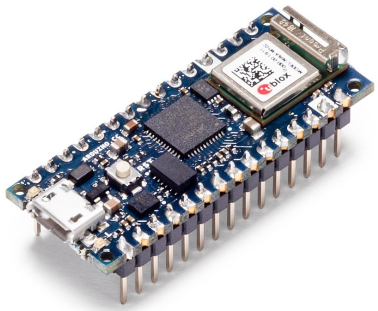


Figure 3-2: Arduino Nano 33 IoT with headers
Retrieved from [53]

SiguesAhi's functionality is accomplished by connecting the microcontroller to the internet, and making it periodically download the first paragraph of the institution's Wikipedia article. Then the microcontroller checks if the first sentence is written in either present or past tense, and determines the existence of the institution. In the examples published in the alpha version of the software library [52], I am monitoring the National Rifle Association, which sadly still exists.

As a feature, I am adding support for Wikipedia articles in different languages, with the hope that people can use this library to make their own instruments to track the

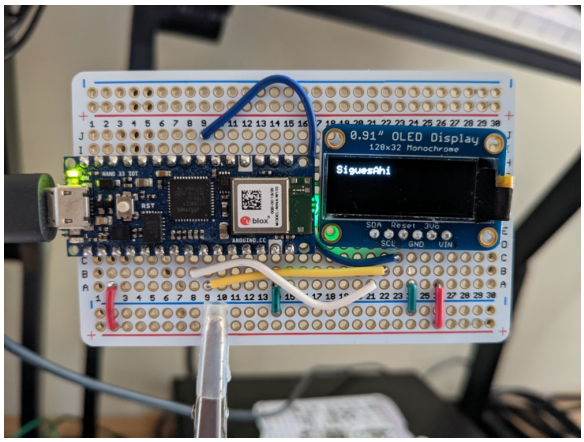


Figure 3-3: SiguesAhi project
Picture by myself

downfall of oppressive institutions wherever they are. I am building one for myself to track the existence of the current Chilean Constitution, written in 1980 by the military dictatorship.

SiguesAhi has been developed in parallel to Tiny Trainable Instruments, and has been possible thanks to the techniques and strategies I have learned about instrument making and publishing software libraries for arts.

3.3.2 Open Drawing Machine

The Open Drawing Machine [54] is a collaborative project with Gaurav Patekar for the research group Future Sketches at the MIT Media Lab. It is a low cost open source machine, consisting of an Arduino microcontroller and hardware for drawing computationally.

Gaurav Patekar designed and 3D-printed the hardware, and I have packaged our code an published a library add-on for openFrameworks. Currently the Open Drawing Machine can receive drawing commands from a computer through a serial port, and the next step of this project is being able to receive commands from other computers through networks and the internet.

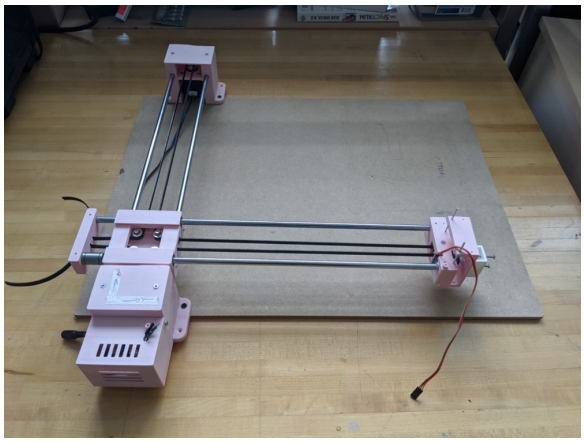


Figure 3-4: Open Drawing Machine project
Picture by myself

3.3.3 Introduction to networks for artists

Introduction to networks for artists [55] is a series of tutorials for beginners, to learn how to set up their own networks and collaborate remotely in peer-to-peer ways for making art, featuring several open source software tools for arts. This project is inspired by the amazing research on remote collaboration, body sensors, peer-to-peer protocols, by movement artist and programmer Lisa Jamhoury [56], including the app Kinectron [57], which I am a contributor to.

3.4 Computational media arts instruments

A big concentration of my research at the MIT Media Lab is computational media arts instruments. I define them as devices that convert the data from an input like human gesture, to process and output different media (audio, video, text, light, etc), using digital operations.

Nowadays artists have access to a rich ecosystem of computers, operating systems and software for making art, and it is common for them to rely on their personal computer for making all of their art, on the go, wherever they are. In the 1990s,

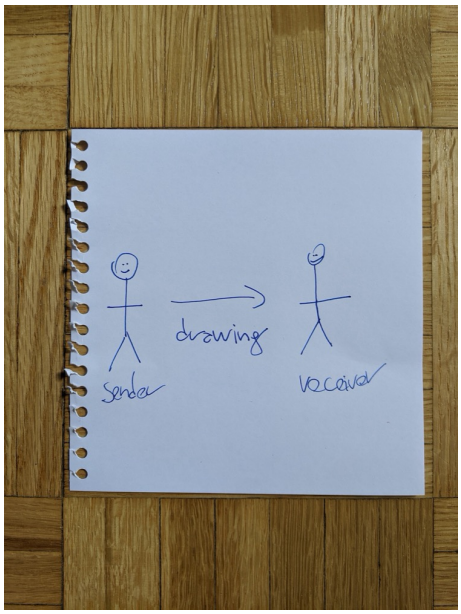


Figure 3-5: Introduction to computer networks for artists project
Picture by myself

musicians stopped having to rely on analog hardware for recording and mixing, and broadly embraced digital technology, with a huge reduction on costs and maintenance.

In this section of my thesis I will share my research on companies making some of my favorite recent computational media arts instruments, with a strong focus on the ones that can be programmed, and that promote open source hardware or software. Many of these instruments often sit at my desk for inspiration, or I spend hours playing with them for my art and learning from them and the communities around them.

The tables 3.1 and 3.2 are respectively summaries of techniques and influences of the instruments that I reference in this section.

3.4.1 Bastl Instruments

Bastl Instruments is a Czech company of multimedia instruments, which has had a huge impact and influence on my research and practice. When I first started researching the Eurorack format some years ago, I visited the shop Control in Brooklyn, NY,

Company	Instrument	Year	Basis	Software
Bastl Instruments	Illuminati	2019	MCU	None
Bastl Instruments	Kastle Drum	2020	MCU	Arduino, C++
Bastl Instruments	Kastle v1.5	2017	MCU	Arduino, C++
Bastl Instruments	OMSynth	2016	IC	None
Bastl Instruments	microGranny 2	2016	MCU	Arduino, C++
Bastl Instruments	Servo	2016	MCU	Arduino, C
Critter & Guitari	Organelle	2016	Linux	Pure Data
Critter & Guitari	EYESY	2020	Linux	Python, Pygame
monome	aleph	2013	Linux	C
monome	norns	2018	Linux	Lua, SuperCollider
Shbobo	Shnth	2013	MCU	Shlisp
Shbobo	Shtar	2017	MCU	Shlisp

Table 3.1: Technical details of media arts instruments

Company	Instrument	Influence
Bastl Instruments	Illuminati	Output with LEDs
Bastl Instruments	Kastle series	Use of breadboard and jumper wires
Bastl Instruments	OMSynth	Distribution as tutorials + parts kit
Bastl Instruments	microGranny 2	Arduino as basis for instrument
Bastl Instruments	Servo	Output with servo motor
Critter & Guitari	Organelle	Scriptable audio computer
Critter & Guitari	ETC, EYESY	Scriptable visuals computer, screen output
monome	aleph, norns	Scriptable audio computer
Shbobo	Shnth, Shtar	Multiple input sensors, scripting

Table 3.2: Influence of media arts instruments

and some modules by Bastl stood out to me, because of their wooden panels and interaction with classic physical computing educational materials, such as motors and solenoids, which was an inspiration for me to include servo motors as an output option in the Tiny Trainable Instruments.

Another inspiration comes from their microGranny 2 granular sampler which is made with an Atmega microcontroller and its firmware is open source and available as a repository on their GitHub account [59], along with many other of their instruments.

Their Kastle synthesizers are also based on microcontrollers, and feature a patchbay for making connections with jumper wires, the same used for prototyping in electronic breadboards. This influenced me to build the Tiny Trainable Instruments with

Figure 3-6: Bastl Instruments Servo module
Retrieved from [58]

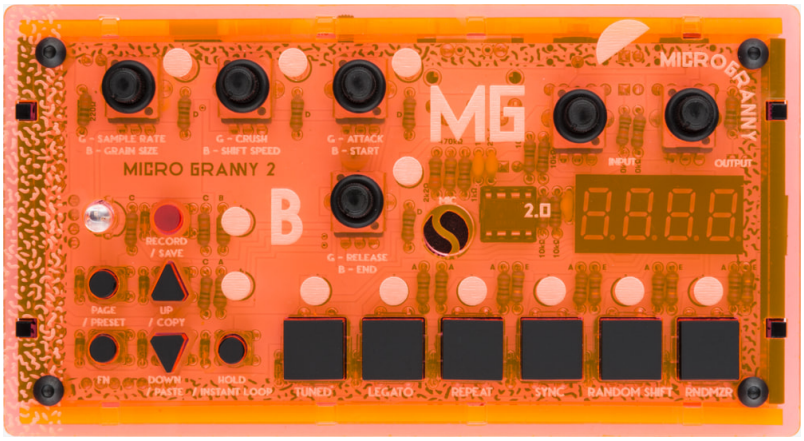


Figure 3-7: Bastl Instruments microGranny 2
Retrieved from [58]

breadboards and jumper wires, instead of custom PCBs.

Also, the Kastle synths are forgiving instruments, their inputs and outputs are robust enough to allow for mistakes in connections, in an electrical and mechanical way, which I think is perfect for safe experimentation. It would be a bummer if the instrument was easy to break, or if it demanded a huge effort in understanding electronics for using it.

As of writing, two different units are in production, both retailing for around 100.00 USD: the Kastle v1.5 melodic / drone synthesizer, and the Kastle Drum, for rhythm. The only difference between these synthesizers is the firmware and the labels on

Figure 3-8: Bastl Instruments Kastle v1.5
Retrieved from [58]

the faceplate. The community is encouraged to write new firmware to modify their behavior.

Figure 3-9: Bastl Instruments Kastle Drum
Retrieved from [58]

Another instrument I want to highlight is the Illuminati, currently discontinued, a device that uses different inputs (audio, control voltage, MIDI messages) to control the light intensity of connected USB lamps, which influenced the conception of Tiny Trainable Instruments as multimedia arts instruments, not only focusing on audio and music, but also printed text, light, and screen output.

The final instrument from this company that I want to highlight is the OMSynth, one of Bastl Instruments' many collaborations with Casper Electronics. This device is an

Figure 3-10: Bastl Instruments Illuminati
Retrieved from [58]

educational and maker circuit development tool for creating synthesizers. It includes basic fundamental blocks, such as battery power, audio input and output, potentiometers for attenuating and boosting signals, and a suite of parts kits for building devices including sequencers, oscillators, and samplers, on the included breadboard. Its release as a kit was also a direct influence in the kits I designed for the workshops I taught with the Tiny Trainable Instruments.

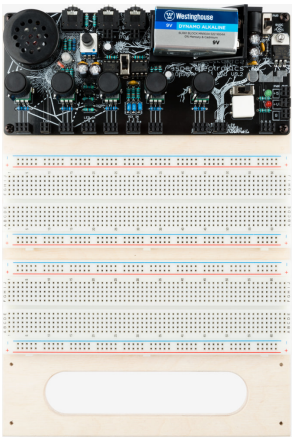


Figure 3-11: Bastl Instruments OMSynth
Retrieved from [58]

Many Bastl standalone instruments are 200.00 USD or less, which is a huge contrast to the 1960s, when a Moog analog system II cost 6,200.00 USD, which was enough to buy a small house [60]. Also, many of their instruments are sold as kits for building and soldering them yourself, for the cheaper cost and the added educational aspect

of having hands-on experience.

3.4.2 Critter & Guitari

Critter & Guitari is an American company based in Brooklyn, NY, which has released several computational microcontroller-based audiovisual instruments, from which my favorite is the currently discontinued Kaleidoloop. It is a sampler with an internal speaker and an included microphone, that allows you to record and playback audio. The instrument features two knobs for controlling the volume and playback rate of audio loops from your recordings. I really admire its portability, in terms of its size, weight, and being battery-powered, encouraging the recording and making of music while walking or in the park. This was a major influence on the design of the Tiny Trainable Instruments; the materials I picked are intended for building standalone instruments that you can power with a USB power bank or a battery, so that you can take them for a walk too.

Figure 3-12: Critter & Guitari Kaleidoloop
Retrieved from [61]

In 2015, Critter & Guitari released the Organelle, a sound computer running the Linux operating system, and the Pure Data (Pd) software for programming patches that reacts to the computer's interface (knobs, encoders, keys), and then generates sound. It is currently on its second iteration, called Organelle M, with added features,

such as an embedded speaker and microphone.

Figure 3-13: Critter & Guitari Organelle M
Retrieved from [62]

Since the sound generation is made with the software Pd, you can achieve the same sonic results on an Organelle or on your personal computer. Despite this, I am convinced that the Organelle is a revolutionary instrument, because it allows artists to use computation in a single-purpose and stable device for their sound, in heavy contrast with the overbloated personal computers we use from activities ranging from watching movies to paying taxes, and which constantly require updates and maintenance. The Organelle also replaces the keyboard and mouse in computers with an interface tailored for contemporary music production, including a musical keyboard for playing notes, knobs and encoders for changing parameters, and a screen for navigating menus.

The final aspects I want to celebrate about the Organelle are its ever growing capabilities, and the community behind it: Critter & Guitari regularly publishes new Pd patches, and the community also organizes in forums and repositories to share their creations. You don't need to be a programmer to enjoy the Organelle, you can download and use the new presets without ever having to open a code editor.

After the release of the Organelle, this company released the ETC visuals computer. It also runs the Linux operating system, and it uses the pygame Python library for generating graphics which can then be projected on a screen. The ETC was discontinued and replace with a compatible and improved instrument, the EYESY, with upgrades such as a new mode for running openFrameworks.

Figure 3-14: Critter & Guitari EYESY
Retrieved from [62]

The current lineup of the Organelle and EYESY computers for arts, make Critter & Guitari a pioneering company in the creation of scriptable instruments that run open source software and foster communities around them. Tiny Trainable Instruments is heavily indebted to the work of this company, trying to promote scripting for microcontrollers for a new generation of artists.

3.4.3 monome

monome [63] is an American company based in upstate New York. Their first instrument was the grid, still in production. In its current iteration, it consists of 128 backlit silicone keys distributed in 8 rows and 16 columns. It has a USB port, through which it can be connected to all sorts of devices, including computers and Eurorack modules.

The monome website features extensive tutorials for learning how to interface the grid with many popular open source hardware and software systems, including Arduino, Processing, Pd, SuperCollider, Python, and Node.js. Over the years I have learned many important lessons about instrument making, software for arts, and live performance with my grid, and I am including what I learned on this thesis project.

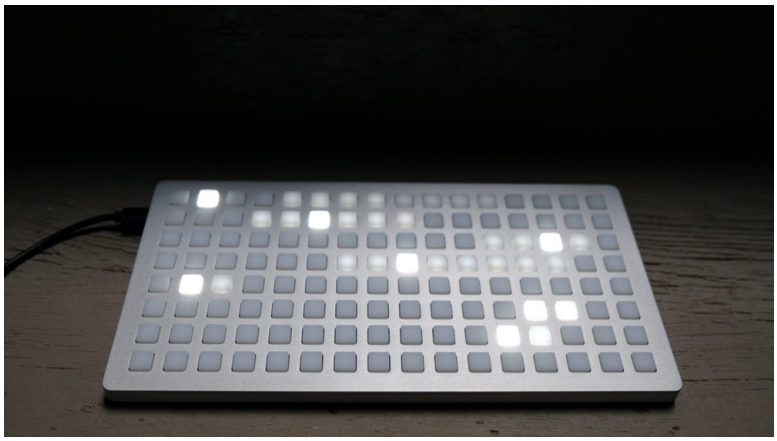


Figure 3-15: monome grid
Retrieved from [63]

Through the grid, I learned about the other instruments by monome, and I want to highlight the open source sound computers they have released over the years, starting with the now discontinued aleph.

Figure 3-16: monome aleph
Retrieved from [63]

The aleph had a focus on community building by releasing the technical documentation and guides for enthusiasts to write and share their own extensions of the software, written in the C programming language. Even though monome only made 100 units of the aleph, it directly influenced their next sound computer, norns, currently available both as an instrument and a DIY shield for the Raspberry Pi computer.

The norns runs the Linux operating system, uses SuperCollider for sound generation,

Figure 3-17: monome norns
Retrieved from [63]

and runs Lua scripts for other tasks, such as reading the data from the buttons and encoders, displaying visuals on the screen, and sharing variables with the sound engine. Users are encouraged to write their own software in these two languages, and there is a strong community, both at monome's forum `https://norns.community/` and at the norns wiki `https://norns.community/`.

3.4.4 Shbobo

In my first semester at MIT I was introduced by classmate Will Freudenheim to the Shbobo Shnth synthesizer by Peter Blasser. The Shnth is a microcontroller-based scriptable instrument, with multiple input sensors, including flex sensors, a microphone, and capacitive skin proximity.

Peter Blasser is an instrument maker, dedicated to synthesynthesis, the synthesis of synthesizers [65]. Peter has created several companies of musical instruments, each one of them with different features and philosophies. The most famous one is Ciat-Lonbarde, which use banana cables to make connections between their jacks, for sharing audio and control signals. Another companies include Tocante, with a line of solar-powered touch-based instruments, and Ieaskul F. Mobenthey, a line of Eurorack modules.

Figure 3-18: Shbobo Shnth
Retrieved from [64]

For this thesis I want to focus on the company Shbobo, a digital collection of instruments based on microcontrollers, which is made up of the already mentioned Shnth, and the Shtar, a 17-fret string instrument.

Figure 3-19: Shbobo Shtar
Retrieved from [64]

Both the Shnth and the Shtar can be programmed using an open source language called Shlisp, available at `https://github.com/pblasser/shbobo`. The Shlisp language allows to use the different sensors on the instruments to control different oscillators, filters, and effects, and interactions between the sound engine and the instrument interface. Just like all of the instruments by Peter Blasser, the Shbobo instruments don't follow classical Western musical conceptions of notes and scales, but rather they follow the approach of pioneers such as Don Buchla, by making their own language for describing the instruments' behavior and sounds.

They promote computer-centric approaches to making sound, such as the use of integers and metaphors of finite state machines, and also allow for different ways of playing and sensing, such as the use of antennas for detecting hand distance, a micro-

phone for detecting speech and whistling, and wooden bars with piezos for detecting pressure.

3.5 Education

This thesis is inspired by the work of the research group Lifelong Kindergarten at the MIT Media Lab, led by professor Mitchel Resnick. In the book with the same title, he builds on Seymour Papert's work, and proposes that educational projects should have "Low floor, wide walls, high ceilings", and that learners thrive when they engage in the 4 Ps: "Projects, passion, peers, and play".

In terms of projects, this thesis includes the release of a software library, so that people can make the software their own, and spin-off their own projects. It is also open source so that people can learn from my mistakes and also fork to adapt to their needs.

In terms of passion, this thesis is a contribution to the ecosystem of instrument makers, so that other people can enjoy my same passion, of making art with computation.

In terms of peers, I have been lucky to have been supported by the MIT UROP office and MIT Media Lab, and had the opportunity to work with MIT undergrad researchers Peter Tone and Maxwell Wang. Also, this project was taught in collaborative workshops where people could discuss their ideas with their peers.

In terms of play, this thesis project is not about correct answers, or even excellent classification with ML, it's all about finding innovative ways to interact with multimedia material, celebrate the small victories and the big glitches, and iterate over and over again.

3.6 Digital rights

So far I have felt comfortable doing scraping in private, for academic or artistic purposes, but not for more public or commercial outlets. For Tiny Trainable Instruments, I was worried about including web scraping techniques, because I didn't want to infringe any copyrights or be involved in litigation.

Because of my concerns, I reached out to the Technology Law Clinic [66] at Boston University's School of Law. My request was for them to assess the potential civil and criminal liabilities if I conducted web scraping on Youtube.com, in order to to train my machine learning algorithms. I also inquired about the legal implications of publishing my databases, my trained models, and related tutoria for encouraging other people to also do web scraping.

The Technology Law Clinic gave me a breakdown of the legal risks associated with web scraping, and they confirmed with me that I could be in violation of three different laws: the Computer Frad and Abuse Act (CFAA), the Digital Millenium Copyright Act (DMCA), and the Copyright Act. This confirmation cemented my decision of using the Arduino microcontroller as a source for my data, so that all my databases would be homemade and fully original. I hope my contributions in how to capture data, parse it, and how to use it to build your own databases is also helpful to others.

I am no expert of digital or human rights at all, but I am convinced that it's a human right to not be surveilled, and I hope this thesis . During my research at the MIT Media Lab, and my class with Sasha Costanza-Chock, I studied or became aware of different efforts in programming for activism, including the Guardian Project by the Electronic Frontier Foundation.

Since ML algorithms need data to be trained on, and corporations and governments are hungry about gathering our data, it is of critical importance for this decade for people to be aware of the pitfalls and dangers of algorithmic bias and surveillance,

and I think the techniques, documentation, and proposals I make for building custom databases for this project, helps in the educational efforts for society's safe practices around their data.

I am constantly in awe about the work by countless activists, including Edward Snowden, Joy Buolamwini, Chelsea Manning, who have spoken truth to power.

3.7 Opera of the Future research projects

During the development of this thesis, I have been fortunate to collaborate on different capacities with other thesis projects by classmates at the Opera of the Future research group, which has directly inspired my work.

3.7.1 Squishies, by Hannah Lienhard

Squishies is Hannah Lienhard's Master's thesis, and consists of novel squishable interfaces for musical expression. We shared discussions about low-level sound design, code reusability, sound art education, and digital instruments. We were part of a Master's thesis working group, facilitated by Roya Moussapour with two other Media Lab classmates, where we workshopped drafts of our thesis. This practice and feedback has been critical in shaping the language and discourse of this thesis document.

3.7.2 Fluid Music, by Charles Holbrow

Fluid Music is Charles Holbrow's PhD thesis. It is a library for library design, documentation for contributors. The design of the interface, documentation, and scope of the thesis were a direct influence on the documentation and API coding style of this project. I was fortunate to collaborate with Charles, by testing the software library,

discussing documentation design, and participating in one of his workshops teaching with Fluid Music, which in turn inspired me to include my own workshop for user feedback and release of this project.

Chapter 4

Tiny Trainable Instruments

This chapter describes the process of conceptualizing and developing this antidisciplinary thesis project, living at the interstitial spaces between the fields of artificial intelligence, electrical engineering, computer science, media arts, music, and pedagogy, among others.

In the next section I explain my personal definitions of the words that make up the title of this project.

4.1 Definition of Tiny Trainable Instruments

4.1.1 Definition of tiny

By *tiny* I mean lightweight handheld instruments that you can fit in a backpack, or take for a walk. This is inspired by my personal practice: for years I have accumulated many small tools for arts that I perform gigs with, or that I carry with me while traveling, mostly small electronic synthesizers, sound mixers, and battery-powered speakers.

Here is an example of a yellow handheld sampler with an included microphone and speaker that I enjoy for field recordings, complete with a green battery pack for portability.

Figure 4-1: Sampler with microphone and portable battery
Picture taken by myself

Tiny is also a nod to the emerging field and community of tiny ML, including - most importantly for me - the tinyMLx professional certificate[49] offered by HarvardX, that I completed while working on this project. I have high hopes that this field empowers people to make technology that is not dependent on the cloud, big data, or centralized servers.

With that, the *tiny* in *Tiny Trainable Instruments* comes from them being handheld devices built on breadboards, from their being able to be powered from a generic USB power bank, and from their using tiny ML processes for interpreting the data from their inputs and reacting via their outputs.

4.1.2 Definition of trainable

By *trainable* I mean a device that can learn by example. This name comes from the ML lexicon, where training is a process whereby an algorithm iteratively finds the numerical values of all parameters (weights, biases) of a ML model.

Training a device is particularly attractive for artists working with sensors. The Arduino microcontroller that this thesis uses has several embedded sensors, which can be used to produce huge streams of data, like this screenshot showing the output of raw data from its RGB color sensor.

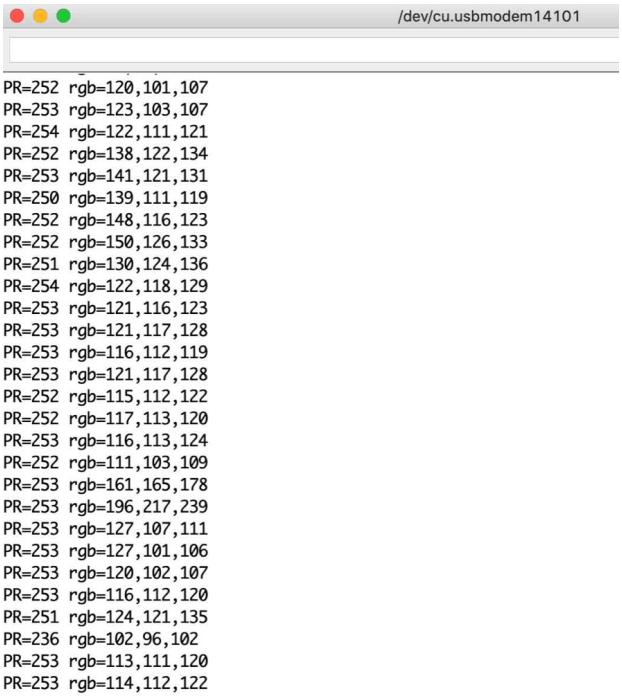


Figure 4-2: Data stream from embedded sensors in an Arduino microcontroller
Screen capture by myself

TOD: what did you do, why did you do it, and how does it work

If we wanted to use this sensor at home to detect a red object, we would start by taking notes of the RGB values of the sensor when the red object is close to it, and would then include this value as a threshold for detection on the software uploaded to the microcontroller. Since the RGB sensor needs consistent lighting to work effectively, we need to be especially careful with the lighting conditions; if they change, we have to look at the stream of data to find the new value for red, and then update the code and reprogram the microcontroller.

This is a tedious process, and it inspired me to work on this thesis, so that I could help artists use ML and microcontrollers to capture data, and then train a model,

letting the algorithm figure out the correct answer without one having to write more code!

An educational and creative example I recommend consulting is Teachable Machine by Google [67], a web application that allows you to use your computer's microphone and webcam to generate data, and then to train models that can detect different words, images, or body postures without having to write any line of code.

In particular, I became interested in tiny ML examples that require small databases and that can be trained on-device, for the use of making an instrument whose behavior can be changed mid-performance, like guitar players do with their guitars when they change their tunings between songs. With this thesis, one can program a device that can detect an apple, and then five minutes later, an orange, and then for your last song, a grapefruit. Hat tip to Sonic Youth, a band that used to tour with more than a dozen guitars, each one of them with a different tuning; they would switch instruments between songs.

Figure 4-3: Sonic Youth guitar with custom tunings
Retrieved from [68]

4.1.3 Definition of instruments

By *instruments* I mean devices that transduce energy across different media. Instruments receive an input, process it, and then produce an output. For musical instruments, I like to think that instruments are able to not just produce musical Western scales, but a wide spectrum of sounds encompassing the broad vocabulary of experimental electronic music and electroacoustic music.

On top of that, my personal definition of *instrument* includes having freedom and liberty, not censorship. This is why I consider my bicycle to be an art instrument that transduces my pedaling input into multimedia output: adventure, sweat, and wind in my face. Also, while I am riding it, I am not subjected to the restrictions of any corporate or government playground, there are mostly no rules besides gravity, and there are no backdoors that anyone can use to track me, exploit me, or limit my speed (we have smartphones for corporate and government surveillance, but that's another story).

The liberties that *instruments* make me feel are a huge contrast to the computational tools I am using for crafting this thesis. I am typing on an Apple Macbook, and even though I like its keyboard and portability, I am aware that I face restrictions: Apple wants developers to pay to be certified developers, and if I update my operating system I won't be able to run 32 bits apps anymore. This thesis is hosted on GitHub, which is a service blocked for developers in certain countries due to U.S.A. sanctions [69]. I am from a country recently affected by a dictatorship and human rights violations and some of my relatives have been exiled, so I am particularly sensitive to policies that limit people's freedom of speech and computing.

This discomfort was also a catalyst for the creation of *Tiny Trainable Instruments*, which are based on open source off-the-cloud microcontrollers, and nobody can censor me when I use them for arts, thus being *instruments*, as gratifying as playing with my guitar and bicycle, and I hope you feel that joy too while spending time with this

thesis.

In this section I will explain the process and the results of each component of this thesis:

1. TinyTrainable Arduino software library
2. Code for training ML with DIY databases
3. Educational material and workshop

4.2 TinyTrainable Arduino software library

This project's main contribution is the TinyTrainable Arduino software library, hosted at `https://github.com/montoyamoraga/TinyTrainable`. It allows you to create flexible multimedia instruments, where you can combine any input with any output, and it runs on the microcontroller Arduino Nano 33 BLE Sense [70]. This microcontroller was chosen because it is currently the only Arduino supported by the Arduino TensorFlow Lite library.

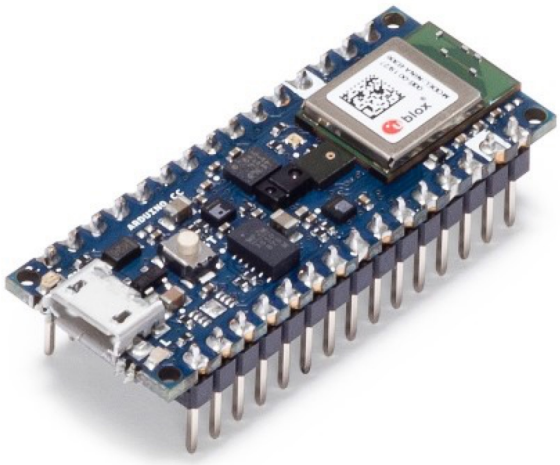


Figure 4-4: Arduino Nano 33 BLE Sense microcontroller with headers
Retrieved from [70]

The TinyTrainable library is flexible, since it allows one to combine any of the three available inputs with any of the seven possible outputs, for a total of twenty-one possibilities summarized on 4.1:

Input \ **Output**	Buzzer	LED	MIDI	Printer	Screen	Serial	Servo
Color							
Gesture							
Speech							

Table 4.1: Matrix of inputs and outputs

For the inputs I used the embedded sensors of the Arduino microcontroller with their corresponding open source libraries, and processed the input data with two recently released open source ML libraries, summarized on 4.2:

Input	**Sensor library**	**ML library**
Color	Arduino_APDS9960	Arduino_KNN
Gesture	Arduino_LSM9DS1	Arduino_TensorFlowLite
Speech	PDM	Arduino_TensorFlowLite

Table 4.2: Software dependencies for inputs

The outputs are actuators that rely on extra hardware, and in some cases, external libraries, which are summarized in 4.3.

Output	**Actuator library**
Buzzer	-
LED	-
MIDI	-
Printer	Adafruit Thermal Printer Library
Screen	Adafruit_SSD1306
Serial	-
Servo	Servo

Table 4.3: Software dependencies for outputs

The behavior of the TinyTrainable library is already designed, and is explained in this chapter. The software implementation is in flux, and I do my best to publish new releases with notes on how the library has changed over time, including bug fixes, clearer language for the examples, and different ways of interfacing with the methods and variables of the TinyTrainable library.

This library is a repository hosted on GitHub, to foster collaboration via issues and pull requests. Each commit is documented commits, to show people the whole process and effort involved in writing this thesis project.

4.2.1 Repository structure

The TinyTrainable library is a repository hosted at `https://github.com/montoyamoraga/TinyTrainable`, where one can review all the history and commits, to see how the whole library or each file has evolved over time.

The structure of the folders follows two simultaneous specifications: it includes the necessary file and folder names for being packaged and indexed as an Arduino Library, and on top of that it complies with GitHub guidelines for licensing libraries, and for automatic workflows for code testing.

The source code of the library is written in C++ and is located in the src/ folder. The examples live in the examples/ folder, and are written in the Arduino language. The examples are divided into four folders: one for each input, and one extra for checking the wiring of each output. In the assets/ folder, I included some C++ files with trained models for the examples.

4.2.2 Installation

The library can be downloaded from the Releases section of the repository at `https://github.com/montoyamoraga/TinyTrainable/releases`, where you also have access to the complete history of releases over time. To install it, you need to uncompress the .zip into a folder, and then make it discoverable by the Arduino integrated development environment (IDE).

Since that method is not automatic and could be prone to errors, I made the effort

to publish the TinyTrainable library, by complying with the latest Arduino Library Manager specifications, detailed on their repository `https://github.com/arduino/library-registry/`. With that, from the Arduino IDE you can open their Library Manager and do a 1-click installation of the library, or as an alternative use arduino-cli, the Arduino command line tool on your terminal.

4.2.3 Hardware basics

The TinyTrainable library at its current iteration has only one strict hardware requirement: it only runs on the Arduino Nano 33 BLE Sense, a microcontroller released in 2019. For powering it you need a generic micro USB cable, which can also be used to upload code to it from a computer running the Arduino IDE.

Figure 4-5: Micro USB cable
Retrieved from [71]

To build a Tiny Trainable Instrument, you also need a breadboard, where you can place the Arduino microcontroller, and some jumper wires to connect it to one of the many possible output devices that I will describe soon.

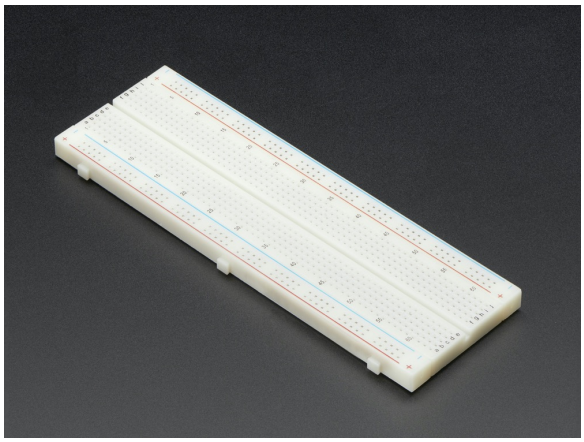


Figure 4-6: Breadboard
Retrieved from [72]

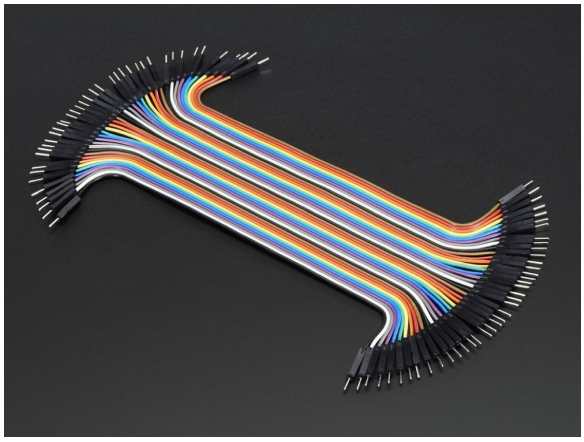


Figure 4-7: Jumper wires
Retrieved from [73]

4.2.4 Inputs

As I mentioned before, for the inputs no extra wiring is needed, since we are using the embedded sensors of the Arduino microcontroller to detect the three possible inputs: color, gesture, and speech. These inputs were designed to foster different behaviors with the library, which are detailed in the following sections.

I decided that each input will be able to detect three different categories, because I thought that more categories would make the instruments more cumbersome to train, needing larger databases, and longer training times. Hopefully in the future I can adapt the TinyTrainable library to make this a variable number of categories it can

classify, but for now this is hardcoded in the library. Nonetheless, if you want to use the library for your own project with a different number of categories, since the library is open source and documented, it should be easy to adapt it to your needs.

4.2.5 Color input

The color input allows artists to build Tiny Trainable Instruments that can detect three different colors of objects near the microcontroller. I recommend that this is the first input that people try, since it is the easiest and fastest of the other inputs to learn and use. It is based on the examples included with the Arduino_KNN library [51].

When detecting color, the library acts as a wrapper of the Arduino_APDS9960 library [74], needed to access the data of the embedded distance and color sensor of the microcontroller. Both of these sensors are used together, so that the instrument can detect nearby objects, and when they are close to the microcontroller, it reads a one pixel RGB value, detecting how red, how green, and how blue any object is.

This RGB data is stored in the microcontroller, to build a database that then is used to train a ML algorithm, in particular a k-nearest neighbors (k-NN), implemented with the Arduino_KNN library. After the algorithm is trained, the instrument is able to detect between the three different colors.

The main difference between this input and the other ones, is that both the data capture and the model training happen on the device, and it is not persistent: every time the Arduino microcontroller is turned on it needs to be trained again. This behavior has the upside of making the data input and the trained model never leave yor microcontroller, for enhanced privacy. The downside is that if you cannot preserve the data or the model, which are erased when the microcontroller is disconnected from power.

4.2.6 Gesture input

The gesture input allows artists to build Tiny Trainable instruments that can detect three different motion gestures, by moving the microcontroller in space. It is based on the Magic Wand example of the Arduino TensorFlow Lite library [75].

Figure 4-8: Magic wand example
Screen capture by myself

This Magic Wand example is able to detect gestures, and print the result over serial port. It is made up of several files, and to make modifications you have to replace code in various of these files. The TinyTrainable library acts as a wrapper for this complexity, only exposing to the user some variables and methods for setting up the instrument.

The equivalent example on the TinyTrainable library that has the same functionality as Magic Wand is called gestures_serial. Currently it is only 36 lines of code, most of them comments or space, plus a file with the trained model. The TinyTrainable library also expands the original functionality of outputting messages over serial port, to include the other six outputs supported.

When detecting gestures, the TinyTrainable library acts as a wrapper of the Arduino_LSM9DS1 library [76] , needed to access the data of the embedded inertial measurement unit (IMU). The library includes an Arduino sketch to capture the data of the three gestures we want to recognize, which then needs to be saved on a computer. These files with the database are imported in a Jupyter notebook to train the ML model with Google TensorFlow. The trained model is then used with the TinyTrainable library and uploaded to the Arduino microcontroller to detect the three different gestures.

Even though for both the color input and the gesture input we use the Arduino to capture the data, for the gesture input we do need a computer to process the data; the drawback is that we still need a computer for training the model. A positive aspect is that the training is persistent: the Arduino microcontroller only needs to be trained once for the same database, and remembers the model every time it is turned on, unlike the color input which forgets the trained model.

A contribution of this thesis is that it expands existing examples that rely on the cloud, in particular the freemium service Google Colaboratory [77] (or Colab for short). Training on the cloud has some advantages; for example, in this case the Google Colab resources are faster than training on my own and most computers, but I think this comes at the expense of losing privacy.

These gesture database we are creating is not compromising information, or makes you identifiable for governments or corporations, like your fingerprints or your DNA. Because of this low-stakes, I don't discourage people from using Google Colab with this thesis, but I do want to stress that we can circumvent these corporate systems. For this reason, a contribution in my thesis is that I am including code and strategies for people to be able to train the models on their own computers, in a more private way.

4.2.7 Speech input

The speech input allows artists to build Tiny Trainable instruments that can detect three different sounds or utterances, by emitting sound next to the microcontroller. It is based on the Micro Speech example of the Arduino TensorFlow Lite library.

Figure 4-9: Magic wand example
Screen capture by myself

This Micro Speech example example is able to detect speech utterances, and print the result over serial port. It is made up of several files, and to make modifications you have to replace code in various of these files. The TinyTrainable library acts as a wrapper for this complexity, only exposing to the user some variables and methods for setting up the instrument.

The equivalent example on the TinyTrainable library that has the same functionality as Micro Speech is called speech_serial. Currently it is only 34 lines of code, most of them comments or space, plus a file with the trained model. The TinyTrainable library also expands the original functionality of outputting messages over serial port, to include the other six outputs supported.

When detecting speech, the library acts as a wrapper of the PDM library [78], used to access the pulse-density modulation (PDM) data of the embedded microphone. This input is the only one where we are not using the microcontroller to capture the data: the documentation includes instructions for building your own database with the microphone in your computer, in order to capture the data of the three different words we want to recognize.

In a similar fashion as the gesture input, the database files are then imported in a Jupyter notebook to train the ML model with Google TensorFlow. The trained model is then used with the TinyTrainable library and uploaded to the Arduino microcontroller to detect the three different words.

The speech input is the most complex and time-consuming one to implement, and we need a computer for both capturing the data and training the model. A positive aspect is that just like the gesture input, the training is persistent, the Arduino microcontroller only needs to be trained once for the same database, and remembers the model every time it is turned on, unlike the color input which forgets the trained model.

Just like the gesture input, the speech input can be trained either on the cloud using Google Colab, or can be trained on your machine. Because of its complexity, I do advise that the time difference is significant: on my Macbook the model took between one and two days to train, in comparison with the gesture model which takes around two hours in my machine.

The speech database we built is the most compromising one, since it is identifiable, and it could even be used for cloning our voice with deepfake techniques. So this is the one I encourage people to be more mindful about when publishing their custom databases.

4.2.8 Outputs

The seven possible outputs of the TinyTrainable library were chosen to showcase many artistic mediums, with the hope that this library can be incorporated into the practice of artists working with light, visuals, sound, sculpture, dance, and poetry, among others.

The TinyTrainable library includes four examples for each one of the seven outputs: one for checking that the wiring is correct on the breadboard, and combinations with each one of the three inputs. Each example is an homage to the classic *hello world* examples of computer science education; that is, they are introductory building blocks, intended to teach artists the basics of each output so then they can continue on their own.

All the materials for the outputs are illustrated by the corresponding image of the product page on the website Adafruit, which I recommend for beginners and artists interested in building their own Tiny Trainable Instruments.

4.2.9 Buzzer output

The buzzer is a device that can vibrate and that emits sound. The TinyTrainable library allows you to output a different sound for each of the three different input classifications, by controlling two fundamentals of the sound: frequency and duration.

I included support for the buzzer because of its low cost and low fidelity sound. I think these specifications make a great contrast with the high complexity of the technology involved in doing tiny ML. I think of Tiny Trainable Instruments with a buzzer output, to be cutting-edge technology driving a cheap buzzing sound. This is an inviting and playful combination that I also want to convey in the other outputs of this project.

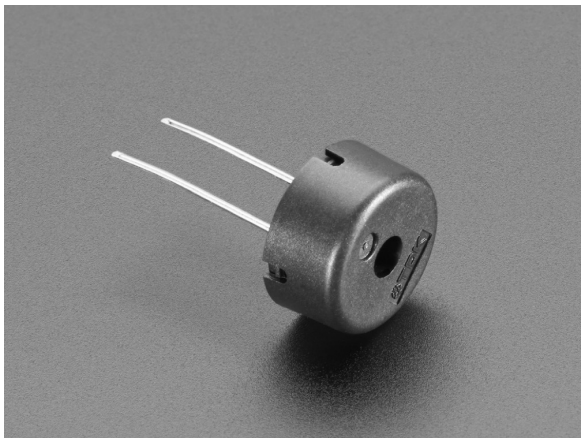


Figure 4-10: Buzzer
Retrieved from [79]

This output requires no additional libraries, the Arduino microcontroller by itself is able to generate the pulse-width modulation (PWM) output required to drive the buzzer.

4.2.10 LED output

The light-emitting diode (LED) is a cheap and fundamental element for making art with lights. The TinyTrainable library allows you to toggle on and off three different LEDs, depending on which of the three possible inputs was detected by the ML algorithm.

This output requires no additional libraries, the Arduino microcontroller by itself is able to provide enough power to light up most hobby LEDs.

4.2.11 MIDI output

Musical Instrument Digital Interface (MIDI) is a protocol invented in the 1980s to allow musical instruments to be interconnected and share information, including musical notes, control parameters, and timing. Traditionally, MIDI instruments have a

Figure 4-11: LED
Retrieved from [80]

5 pin DIN connector, (from the German Deutsches Institut für Normung), like the one pictured here for use on a breadboard.

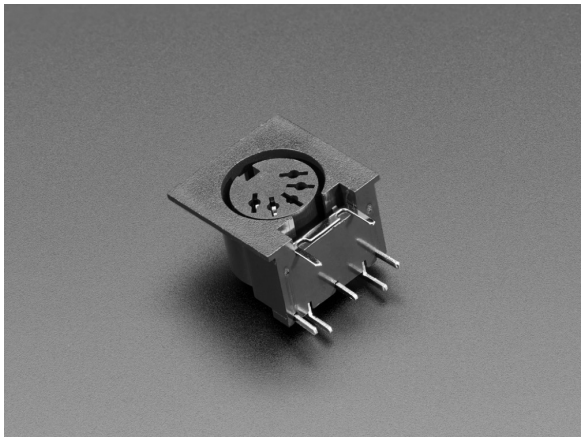


Figure 4-12: MIDI DIN connector
Retrieved from [81]

MIDI can be used for transmitting all sorts of data, and thesis examples how to use MIDI to control different aspects of MIDI-enabled instruments, including synthesizers and drum machines. As a bonus, I wrote examples to interface with three synthesizers from the volca series by Korg: volca bass (analog bass machine), volca beats (drum machine), and volca keys (polyphonic synthesizer). I picked these synthesizers to have both melodic and percussive musical elements, because of their huge popularity, their relatively low price of 150.00 USD, their extensive MIDI capabilities, and because I am a huge fan of the instrument designs by Tatsuya Takahashi [82].

This output requires no additional libraries, since the Arduino microcontroller by itself is able to communicate via serial protocol, and the MIDI protocol is a subset of serial, at a particular baud rate.

4.2.12 Printer output

I used to associate printing with expensive toners and ink cartridges, and mostly frustration. This radically changed when I took a physical computing class, where I learned about thermal printers and how I could use Arduino microcontrollers to drive them! I find these thermal printers amazing because they are mostly a one-time investment and they don't need anything besides paper and power, so I have used them for interactive art installations since then. That is why I decided to include support for them on this project.

The thermal printer kit I recommend on the bill of materials is around 60.00 USD, and includes the power supply and paper roll. I picked this one because it is sold and supported by Adafruit, and I am a huge fan of their software libraries and tutorials.

Figure 4-13: Thermal printer kit
Retrieved from [83]

With this output, you can create a Tiny Trainable Instrument that can print different texts and in various fonts and formatting, depending on the classification of the input. I hope this inspires a new generation of computer poets and interactive literature!

This output requires 1 additional library, the Adafruit Thermal Printer library, which is open source and includes several examples for expanding the functionality of it. If you want to use an alternative hardware or software for thermal printing, I recommend you modify the TinyTrainable library, and include any other additional software you want.

4.2.13 Screen output

In contrast to the static nature of the output of the thermal printer, which prints a fixed message on a piece of paper, I included support for using a screen for dynamic messages on a surface.

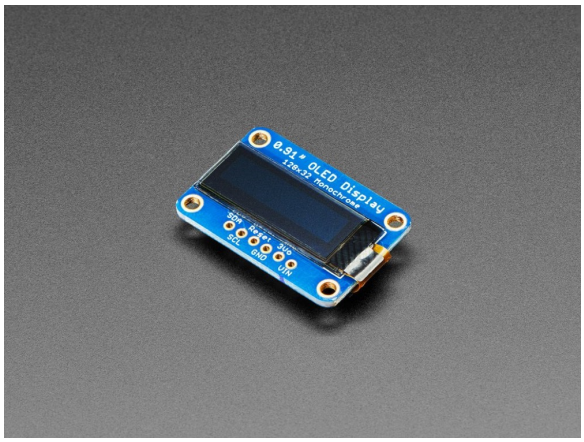


Figure 4-14: Screen
Retrieved from [84]

This output requires 1 additional library, the Adafruit_SSD1306 library (and its own dependencies), for the particular screen I recommend on the bill of materials. The same vendor Adafruit offers many alternatives in different sizes, technologies, and color support. I encourage people to use different screens from different vendors, and fork and adapt the TinyTrainable library to accomplish it.

4.2.14 Serial output

This output uses the same USB cable we mentioned earlier, for delivering the power and uploading code to the Arduino microcontroller. We can use the same cable to send messages over the serial protocol, and receive them on the Arduino IDE, or in any other software that reads from the serial port.

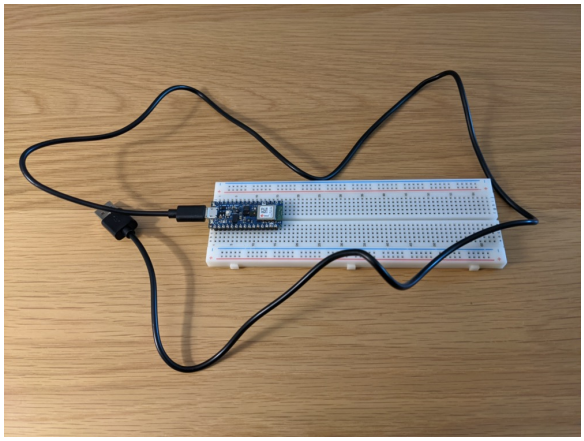


Figure 4-15: Tiny Trainable Instrument with serial output
Picture taken by myself

This output is useful for proofs of concept, for quick testing of the input classification before wiring a more complex output, and also for doing low-level communication with other serial software on your computer.

This output requires no additional libraries, since the Arduino microcontroller by itself is able to communicate via serial protocol, and the MIDI protocol is a subset of serial, at a particular baud rate.

4.2.15 Servo output

This final output is one of the cheapest ones, along with the buzzer, and that is why both were included in the thesis workshop. This is another example taken from the tradition of learning how to use servo motors in physical computing classes when

learning how to program Arduino microcontrollers. That is why I included support for them in the library.

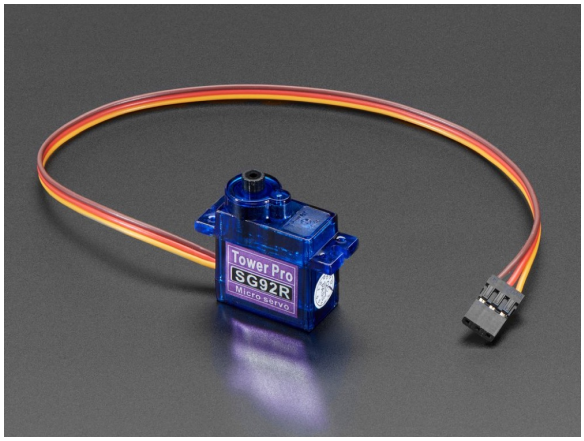


Figure 4-16: Micro servo motor
Retrieved from [85]

I see the servo motor as a very versatile output, which can be thought of as a generator of movement. So our Tiny Trainable Instrument can dance in space, or point at different places. For this library, I programmed support for moving the servo at different tempos, even with some random noise for hiccupy movement, so that the servo motor can be thought of as a robotic arm for drumming or tapping on different surfaces for music. In this image, you can see an example of this with some tape and wire for extending the arm of the servo motor.

This output requires no additional libraries. The Arduino microcontroller by itself is able to generate the pulse-width modulation (PWM) output required to drive the servo.

4.3 Code to build databases and train ML models

One of the biggest challenges I have encountered when working with ML is to find a suitable database, that I am comfortable to work with, and that I know how to use. Often, the solution to my artistic project is building my own databases, which I

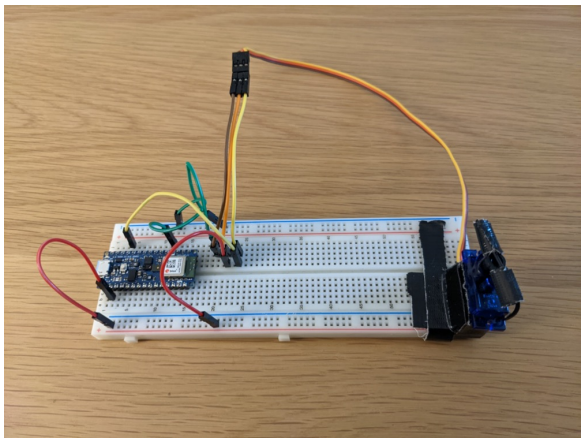


Figure 4-17: Tiny Trainable Instrument with servo output
Picture taken by myself

have done in the past by scraping the web, or doing very tedious manual work. I do these experiments on my computer and do not publish them because I don't want to infringe on anyone's copyright.

This project does not include techniques for web scraping, since I don't want anyone to run into legal trouble, and also because I enjoyed using the Arduino microcontroller and my computer in the privacy of my room, to build the necessary databases for gesture and speech recognition, and I want to share that with artists.

In the repository that holds this thesis document, available at `https://github.com/montoyamoraga/tiny-trainable-instruments`, I included a folder called *notebooks/*, with Jupyter notebooks written in Python 3 and using the Google TensorFlow library for training.

4.4 Educational material and workshop

In this thesis my aim is to acknowledge all the steps, dependencies, and labor needed to create the project. It has been built in small increments, which have been saved and published in the repositories that make up this project, because there is a great

educational and scholarly value in documenting and reading the history of a project. This is also a a way to give back to the community of artists who generously sharing not only the output of their work, but also their process and their tools.

This educational aspect is present throughout the included documentation: the source code is commented, there are README files for helping navigate each repository and their folders, the TinyTrainable library includes examples with each section commented and explained, and the library complies with the Doxygen standard for generating automatic documentation.

I also designed, wrote, and taught a remote workshop for beginners with the TinyTrainable library, which is explained in detail in the next chapter.

4.5 Design principles

Here is a summary of the main design principles I followed when making decisions and building this thesis project.

4.5.1 Affordable

The software library is virtually free, and it can be downloaded from the internet at no monetary cost; only bandwidth and time and hard drive costs.

The materials picked were the cheapest ones I could think of, and they are often used in similar educational contexts because they can be reusable and are forgiving to mistakes. The microcontroller is a one-time investment, and the outputs range from 2.00 USD to 60.00 USD.

The materials for the workshop for building Tiny Trainable Instruments with buzzer,

serial, and servo outputs had a cost of only 60.00 USD, including shipping.

4.5.2 Open

All examples included with this library were written with the aim of showing the fundamentals of how to build the instruments and different ML-enabled manipulation of multimedia material, so that people could build on top of it and make it their own, by changing the values of variables and adding more functionalities.

4.5.3 Remixable

All the process and the code is open, and I wish that the TinyTrainable library can be used as a dependency on other people's projects. I also hope that people modify this project and make it their own, building their own spin-offs.

4.5.4 Private

A huge emphasis I wanted to stress is to make people feel safe while handling the data. Maybe nowadays we are used to being surrounded by surveillance cameras when we are in public, or even at our homes with devices that listen to us and are connected to the cloud. I hope that this thesis can provide a computational experience that feels safe, respects your privacy, and makes you critical and aware of the issues of modern industrial machine learning applications.

4.6 Development

This thesis has been developed in collaboration with MIT undergrad researchers Peter Tone and Maxwell Wang, and funded by the MIT UROP office and the MIT Media Lab. Peter Tone helped with research in data structures, library writing, and programming different sections of the library, from the most experimental features to the most polished ones, and also helped with the design of the user-facing methods and variables. Maxwell Wang helped with proofreading the code, making sure the language of the examples and documentation were appropriate for beginners, and helped with the planning and conceptualization of the workshop.

This collaboration was remote, and made possible over issues and pull requests in repositories, and notes in shared Google Drive documents.

The most difficult bottlenecks that I faced while programming this library were solved by office hours with Chilean artist and programmer Roy Macdonald [86].

4.7 Code details

This thesis is distributed as a repository, hosted on the GitHub platform, and available at https://github.com/montoyamoraga/tiny-trainable-instruments.

The auxiliary files, such as the LaTeX project for this document, and the auxiliary Jupyter notebooks, and documentation and tutorials are included in this repository.

The main software component of this project is the TinyTrainable library, available at https://github.com/montoyamoraga/TinyTrainable and also through the Arduino IDE.

The code included in this library is distributed through the folders:

1. examples/
2. src/

4.7.1 src/

This folder contains the C++ files for the TinyTrainable library. It includes the base files TinyTrainable.h and TinyTrainable.cpp , with all the basic functionality of the library. Additional files live on specific subfolders, and are explained below.

Code for inputs

This folder contains the C++ files for the output base class, and the files for each of the 3 inputs that inherit from the base class. Each input file imports the necessary library dependencies and acts as a wrapper for them.

Code for outputs

This folder contains the C++ files for the output base class, and the files for each of the 7 outputs that inherit from the base class. Each output file imports the necessary library dependencies and acts as a wrapper for them.

Code for speech recognition

This folder tensorflow_speech/ contains auxiliary files for the speech input, copied from the examples from the Arduino TensorFlow Lite library. Unless otherwise noted, these files are included without modifications and distributed through the Apache License included on each file's headers.

4.7.2 examples/

This folder contains the code examples for the TinyTrainable library. They are organized in four subfolders, one for each input (color, gesture, speech), and one for checking the wiring of each output. Each subfolder has one example for each of the seven possible outputs. There is an additional example for creating the gesture database, for a total of twenty-nine cod examples.

Examples for checking connections

The code examples on the check/ folder are written as an introduction for the different outputs, including comments for the functionalities written in the TinyTrainable library, and notes for wiring each output correctly.

Examples for each

The code examples on the color/, gesture/, and speech/ folders are written to use each input, and use them to output multimedia art through the different supported outputs.

4.8 Auxiliary tools

This is a summary of some auxiliary tools I used for making this project.

4.8.1 clang-format

Tool for automation of formatting to source code. More information at `https://clang.llvm.org/docs/.ClangFormat.html`.

4.8.2 Doxygen

Tool for generating documentation from the source code. More information at `https://www.doxygen.nl/`.

4.8.3 GitHub Actions

Every time we push code to the TinyTrainable repositories, a GitHub action creates a virtual machine, and runs a script to generate the Doxygen documentation and push it to the gh-pages branch, hosted at `https://montoyamoraga.github.io/TinyTrainable`.

4.8.4 Jupyter

Jupyter is a free, open-source browser application that allows users to easily read and write code in a clean, accessible environment. Code is segmented into cells, which users can run individually by clicking into and selecting the triangle "play" button at the top. Subsequent code runs based on operations done in previous cells. Basically, Jupyter notebooks allow programmers to create clean, step-by-step interactive walkthroughs for their code. More information at `https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/index.html`.

4.8.5 Markdown

Markdown is a lightweight markup language with simple, intuitive syntax. Aside from a few key differences, it is largely the same as plaintext. The documentation of this project is written using Markdown, including this document! More info at `https://guides.github.com/features/mastering-markdown/`

Chapter 5

Workshop and user testing

In the past I have taught several classes and crash courses of software of arts, and I used this experience to create a workshop for teaching Tiny Trainable Instruments. Teaching with this thesis project was conceived as a way of user testing and releasing to a wider non-academic audience. In this chapter I explain the workshops's design process, the feedback received from the students, and the challenges faced.

5.1 Workshop design

The Tiny Trainable Instruments workshop primary audience is artists and educators with no previous technical knowledge of programming, ML, or microcontrollers. It is a hands-on class, where students build their own Tiny Trainable Instruments, create their own databases, and train their own ML models.

The workshop consists of two sessions of two hours each, taught in two consecutive days for a total of four hours. The first session is focused on installation of the software, connecting the hardware components, and instruments using the color input. The second session is about capturing data and training models, for gesture and

speech detection. During both sessions we use different outputs, including serial for text, servo for movement, and buzzer for sound.

This workshop was taught three times, the first two in English for people based in the U.S.A., and the third one in Spanish for people based in Chile. Each workshop had between six and seven students, for a total of twenty students. The three workshops were taught in the span of ten days, and each iteration informed the next one. Between sessions I tweaked the documentation, the class script, and the different code examples to make them easier to understand.

The workshop is designed to be taught virtually or presentially. The students require a laptop computer, internet connection, and a bill of materials for building the Tiny Trainable Instruments. To eliminate cost barriers, I applied and was awarded a generous grant of 2,000.00 USD from the Council for the Arts at MIT. This funding was used to pay for all the materials and shipping for the 20 participants, who signed up, attended for free, and kept the materials to keep on building Tiny Trainable Instruments on their own.

The workshop also complied with the requirements and was approved by the MIT Committee on the Use of Humans as Experimental Subjects (COUHES). This workshop posed a minimal risk to the participants, since effortes were made so that the students wouldn't share any data with me, and so that they were always in control of their privacy. The materials for building the Tiny Trainable Instruments are safe, low voltage, and used in similar educational settings, so all requirements were met for participants' safety. Additionally, the teaching was conducted over videoconferencing software, to ensure all health protocols were respected.

5.2 Workshop promotion

For promoting the workshop I collaborated with designer Renata Gaui [87], who designed the bilingual workshop flyers in English and Spanish. I posted these flyers on my Instagram and Twitter accounts, and they were shared by artists, designers, educators, and activists from different communities in U.S.A. and Chile.

Figure 5-1: Workshop flyer cover, in English
Flyer by Renata Gaui

The flyers highlighted the multimedia and political orientation of Tiny Trainable Instruments, explaining that in the workshop people would learn how to use ML with different inputs, including color, gesture and speech, to control different artistic outputs, including text over serial communication, buzzer sounds, and motor movement. The flyer also included a picture of a Tiny Trainable Instruments prototype, made with the Arduino microcontroller on a breadboard, jumper wires, a servo motor, and masking tape, to convey an artisanal and welcoming environment for beginners.

More than ninety people signed up for the twenty available spots. I consider this open call to be succesful, since it reached people who I didn't know at all who were entushiastic about this project. I picked the twenty students, so that I could have a

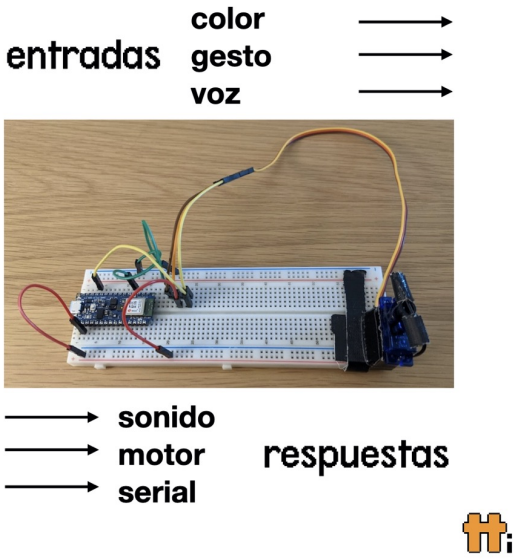


Figure 5-2: Workshop flyer multimedia inputs and outputs, in Spanish
Flyer by Renata Gaui

diverse crowd, including artists, designers, musicians, programmers, educators, and activists. I included some acquaintances and former students, with the hope to get deeper feedback from them during and after the workshop. It also helped me to have more confidence while teaching on this new virtual format, and after a one year hiatus of organizing workshops.

5.3 Workshop logistics

The three workshops were taught respectively on June 22-23, June 25-26, and June 29-30 2021. Before each one, the participants were sent a basic kit for building Tiny Trainable Instruments, including these six materials: Arduino microcontroller, breadboard, jumper wires, USB cable, buzzer, servo motor.

The first four materials are needed for any Tiny Trainable Instruments, and the last two were picked for outputting sound and movement, to appeal to a wide audience of artists. They were also selected over because of their easy wiring and cheap cost,

compared to the other outputs supported by the software library.

The Arduino microcontroller can be acquired from the Arduino online store, or from other electronics distributors. The rest of the materials are available on Adafruit, picked because of its focus on being an electronics store for artists and beginners. For the thirteen students in the U.S.A. I acquired the materials from sellers Arduino and Adafruit, and then shipped them to their addressses their kits via United States Postal Service (USPS).

Figure 5-3: Workshop packages for the students in U.S.A.
Picture by myself

For the seven students in Chile, I acquired the materials from the same vendor, and shipped them to my mother's home in Chile. She also sorted them in kits and then distributed them to the participants, via pickup or mail.

In parallel to the shipping of the materials, the participants were sent instructions via email with links to the project's documentation, in particular to install the Arduino IDE, the TinyTrainable library, and its dependencies. If students had problems while installing, I helped them over email or videoconferencing. Over the course of teaching the workshops, I updated the installation instructions to make them clearer and updated.

Figure 5-4: Workshop packages for the students in Chile
Picture by Bernardita Moraga

5.4 Workshop curriculum

In the first session, the students make sure their installation of the software is correct. After that, they unbox their materials, and place their Arduino microcontroller on their breadboard. Then they connect it with their USB cable to their computer and upload their first code example, to check that both hardware and software is functioning correctly.

When this is done with success, the students learn about the embedded RGB and proximity sensors used on the color input. Students learn how to capture their own color database, and train their k-NN algorithm. As an output, first they learn how to read the color classification on their serial port, and then they wire the buzzer to output sounds with different frequencies and durations. The first session concludes with students modifying the code examples, and are encouraged to try to wire and output movement with their servo motor.

The second session focuses on the gesture and speech detection. The students learn how to capture gesture data with their Arduino microcontroller, and then organize it in a formatted database. Then they learn how to use Google Colab for training on the cloud, and discuss the privacy implications of uploading their data to an external

server. Since the data they are handling is not compromising, and for time reasons, we use Google Colab, and they learn the alternatives to train these models on their own machine.

The students learn all the steps involved in creating their database, formatting it, training the model, and adapting it for use with their microcontroller. The final part of this session is an overview of the speech recognition of Tiny Trainable Instruments, and they learn strategies for creating their own speech databases, and resources to gather training from external resources.

5.5 Workshop feedback

After the completion of the workshops, I sent the students a Google form to ask for anonymous feedback. The students gave very encouraging reviews that makes me consider this workshop a success, and I hope to teach it again, and expanding it to a semester-long class for college education.

The students were asked to describe the workshop with one word. They responded: interesting, challenging, fun, innovative, inspiring, didactical, exciting, fun, new, fast-paced, and experimental.

They said that they learned programming fundamentals, the new artistic possibilities that tiny ML offers, ethical AI, and algorithmic bias. All of them also reported that they wanted to keep on learning on their own programming, multimedia art, and ML.

Their favorite aspects of the workshop are the thoughtfulness and usability of the library, its practical and hands-on approach, its way of making ML more understandable and inviting, the thorough step-by-step explanations, and the joy of learning together with other artists.

The main complaints were that they wished the workshop was longer and in person, and that they wished we could have covered more advanced topics, including how to package the Tiny Trainable Instruments in a more robust platform, instead of a breadboard. Their suggested additions include video tutorials, more code examples and more sessions for the workshop.

5.6 Workshop challenges

As an instructor the main challenge for me was the rhythm of the workshop, mainly because the code examples around 10 minutes to compile. I choreographed the workshop so that students opened a new example, started the compilation, and while it happened we read and studied the code. With this technique, right after the compilation and upload to the microcontroller, we were ready to start using the code example.

My other challenge was to keep the language of the workshops inviting instead of overwhelming. My strategies to foster a positive environment, included sharing with the students all the issues I had while developing the library, so they could appreciate the labor involved in writing this library and educational materials. I also gave them context about how recently published were the ML libraries we used, so that they could appreciate the novelty and timing of this project.

A wide array of challenges from the students were reported on the feedback form. Some of them had problems with their hardware, including internet issues or their computers being too slow to download and compile the code examples. Others wished the workshop was presential, so that we could collaborate in person. Some students also thought that the first session was really easy to understand, and with a good pacing, and that the second session was too advanced for them, and that they would rather dedicate this time to learn with more depth the topics presented in the first

session.

5.7 Final thoughts

My favorite moment during the final workshop was when the students in Chile exchanged their contact information to stay in touch, share their art, and form a study group to keep on learning and collaborating. It happened unprompted, and I hope that they can incorporate microcontrollers and ML on their artistic practice.

During the workshop the students were shown how to modify the code examples. We explored different strategies for incorporating Tiny Trainable Instruments to music pieces with the buzzer and servo motor outputs, butbBecause of the time constraints we were not able to elaborate as further as they wished. I hope that the students are encouraged to continue learning on their own, and I hope I can continue developing open source hardware, software, and educational material to teach these topics.

Chapter 6

Conclusions and future work

In this thesis I have presented all the stages of the design and development of new standalone multimedia instruments using machine learning and microcontrollers, emphasizing AI ethics. The project includes software examples, hardware suggestions, educational material, and strategies for ethical off-cloud machine learning and creation of custom artisanal databases.

This thesis is also the basis for further research, including the creation of subsequent multimedia instruments and software libraries, the writing of new courses and educational units at the intersection of arts, physical computing, interaction design, and computational ethics.

6.1 Contributions

Concrete contributions:

1. Publishing TinyTrainable, a software library for creating instruments with ML and microcontrollers for multimedia art.

2. Design, writing, and teaching a 4-hour workshop for beginners, enthusiasts, and artists to teach with the TinyTrainable software library.

3. Publishing code and tutorials for creating custom databases for gesture and speech recognition, and for deleting metadata for privacy concerns.

4. Publishing custom-trained ML models for gesture and speech recognition.

5. Publishing code and tutorials for training ML algorithms on the cloud and on personal computers for privacy and agency.

6. Publishing other related software libraries, such as MaquinitasParams for communication with other instruments, and MaquinitasRitmos for rhythmic data.

Abstract contributions:

1. Demonstrating how a broader range of people can use ML to support their creative expression.

2. Developing strategies for artistic reappropation of the inherent imperfections and shortcoming of ML.

3. Sharing the step by step process of conceptualizing and implementing a software library for artists.

6.2 Lessons learned

1. Writing software for artists is hard.

2. Writing software libraries for other artists is even harder.

3. Collaborating with other people is essential to write usable code and clear educational material.

4. Documenting all design decisions and steps is key to explaining why and how your project works.

5. Navigating laws, licenses, and copyright is really hard.

6.3 Future work

6.3.1 Hardware for new instruments

This thesis relies on an Arduino microcontroller because of their open source nature, commercial availability, software and community support, and detailed documentation.

In particular I picked the Arduino Nano 33 BLE Sense, because of two main reasons at the time this project started in late 2020: it is currently the only Arduino supported by Google's TensorFlow Lite for microcontrollers library and the HarvardX certificate on tiny ML. Also, because of its convenience of having embedded sensors, which makes it simpler and cheaper to acquire data for live interaction and for building custom databases, eliminating barriers to instrument makers and prototypers.

Microcontrollers come and go. Most probably this Arduino will be discontinued, but the strategies and software presented in this thesis can be adapted to other microcontrollers and software architectures. I am particularly looking forward to this thesis project being ported to other Arduinos, PJRC Teensy, and Adafruit Circuit Playground. They feature different software stacks, such as Python instead of C++, and through that cater to a wider community of people building multimedia instruments and artwork.

In terms of the outputs of the Tiny Trainable Instruments, I focused on creating many parallel multimedia approaches, including making sounds with piezo buzzers

and MIDI, manipulating light with LEDs, creating movement and rhythm with servo motors, and printing text with thermal printers and screens. This is to appeal to a large audience of artists and learners, interested in different mediums. I hope this thesis project inspires the addition of further outputs, and that people can contribute back to the library to share these new capabilities with everyone.

The Tiny Trainable Instruments are built with prototyping electronic breadboards, to make explicit their open-endedness, and to promote experimentation and lower barriers. A further iteration of this project that I wish to make is creating custom PCBs with fixed wiring, and also enclosures and packaging.

6.3.2 Software for new instruments

This thesis has been published as an open source software library for Arduino. It promotes modularity and adaptability: a Tiny Trainable Instrument can be any combination of the multiple inputs and outputs. The file structure of the source code and the software dependencies of this library were also written with flexibility in mind, to encourage the remix and adaptation of this library to further projects.

A challenging aspect of this project is the breadth of the disciplines combined, and its novel application of ML in microcontrollers. There is a community of makers creating open source standalone multimedia instruments, but the required skills are still hard to acquire. I hope this project helps fostering a new generation of instrument makers, who can learn from this project, like I have learned from my favorite makers.

Additionally, the principles of this project, including being as cheap as possible, and as open as possible, are designed to encourage experimentation and hacking, but also can pose additional challenges. I hope this project encourages people to learn how to make instruments, and also engages in discourse about the creation of new curricula for the next generation of instrument makers and artists.

Another challenging aspect of writing software for multimedia instruments is its licensing, both choosing a license and also respecting and understanding the license of other code and resources we are using. The dependencies of this software are mostly other libraries by Arduino, Google, Adafruit, and with different licenses including public domain, MIT, and Apache. I hope this document helps to navigate these legal complexities and that this project helps artists and enthusiasts to navigate this landscape and overcome these barriers.

6.3.3 Educational impact

This project was built to inspire and celebrate a new generation of coursework, workshops, and books, in the disciplines of ethical ML and microcontroller-based instruments.

I hope that this thesis project is adopted by educators, to introduce students to ML, physical computing, media arts, and computational ethics. It would be amazing if aspects of Tiny Trainable Instruments could be incorporated into new and existing music, arts, sculpture, and computer science curricula, to create a new wave of instrument makers and media artists.

Appendix A

Context

Since joining MIT in summer 2019, I didn't leave the U.S.A., and it's the longest stretch I have had of not visiting my home country Chile. This thesis in particular was written between November 2020 and August 2021, mostly in Boston, MA, U.S.A., while on a F-1 visa.

A.1 Language

My native language is Spanish, and this thesis was written in English, using the metric system, and the Gregorian year-month-day format.

I tried to avoid violent language, including some widespread conventions of computer science which I hope become obsolete, such as executing a file, or screenshot, instead of running a file or screen capture.

A.2 Software

This thesis document was written using LaTeX and the Microsoft Visual Studio Code editor, and then exported to the PDF format.

The TinyTrainable library was written in C++, and packaged as an Arduino library, relying on open source library dependencies by Adafruit, Arduino, and Google.

The auxiliary code is a mix of Python scripts, Jupyter Python notebooks, and shell scripts.

The documentation was written using Markdown.

The workshops were taught using the videoconferencing software Zoom, and organized via Google Forms.

A.3 Hardware

This project was written on a 2017 Macbook Air 13-inch, running the macOS 10.15.7 Catalina operating system.

The software library and the software examples were written to be deployed on the Arduino Nano 33BLESense microcontroller.

A.4 Collaborators

Priscilla Capistrano is the senior administrative assistant in the Opera of the Future research group and made sure that everything worked.

Peter Tone is a MIT undergraduate student, who was a researcher, designer, programmer, and tester of the TinyTrainable Arduino library, as part of the MIT UROP program.

Maxwell Wang is a MIT undergraduate student who was a documentation writer, and hardware and software tester, as part of the MIT UROP program.

Roy Macdonald solved my most difficult programming questions, and helped to implement the solutions.

The Council for the Arts at MIT provided the generous funding of the workshop materials.

Renata Gaui designed and created the flyers for the workshops.

Bernardita Moraga packaged and distributed the materials for the workshop in Chile.

Appendix B

Scripts

For this thesis I developed some scripts which are included in this repository on the scripts/ folder.

I included comments and variables with the hope of making them readable and useful for other people, and also published them on a standalone repository with a MIT License at `https://github.com/montoyamoraga/scripts`.

B.1 Formatting code with clang-format

clang-format is a command line tool for formatting code. This script was written to auto format the code from the Arduino/C++ library TinyTrainable.

```
echo "formatting with clang"
find "$PWD/../TinyTrainable/src" "$PWD/../TinyTrainable/examples" -iname
    "*.cpp" -o -iname "*.h" -o -iname "*.ino" | while read f
do
  clang-format -i "$f"
  echo "formatted $f"
```

```
done
```

B.2 Converting formats with ffmpeg

ffmpeg is a command line tool for converting audiovisual files between formats. This script was written to convert audio files from .mp3 to .ogg format for training a database for speech recognition.

```
# clear command line
"clear"

# directory name
DIR_MEDIA="media"

# extension of original files
EXT_ORIGINAL="mp3"

# extension of desired files
EXT_DESIRED="ogg"

# announce start running script
echo "start running " $PWD/$0

# check if files/ exists
if [ -d "$DIR_MEDIA" ];

# if files/ exists then
then

  echo "success, $DIR_MEDIA/ exists"
```

```
# check if there are .mp3 files in files/
if [ -f $DIR_MEDIA/*.$EXT_ORIGINAL ];

  # if there are files with $EXTENSION in directory
  then

    echo "success, there are matching $EXT_ORIGINAL files"

    # iterate over every matching file in directory
    for i in $DIR_MEDIA/*.$EXT_ORIGINAL;

      # pipe the filename into cut
      # -d is delimiter of '.'
      # -f is the field number, indexed in 1
      # it retrieves the filename without the extension
      do name=`echo "$i" | cut -d'.' -f1`

      echo convert "$i" $EXT_ORIGINAL to $EXT_DESIRED

      # ffmpeg conversion
      ffmpeg -i "$i" "${name}.$EXT_DESIRED"

      echo converted "$i" to $EXT_DESIRED

      # delete original file
      rm "$i"

      echo deleted "$i"

    # finish iteration
    done
```

```
    # if there are no matching files in directory
    else
      echo "fail, no $EXT_ORIGINAL files in $DIR_MEDIA/"

  # end of if statement for matching files
  fi

# if directory does not exist
else
  echo "fail, $DIR_MEDIA/ does not exist"

# end of if statement for existence of directory
fi

# announce finished running script
echo "finished running " $PWD/$0
```

B.3 Deleting metadata with exiftool

exiftool is a command line tool for reading and writing metadata from files. This script was written to delete metadata from images, like GPS coordinates added by modern smartphones, only keeping the actual image.

```
# clear command line
"clear"

# directory name
# DIR_MEDIA="media"
DIR_MEDIA="$PWD/../thesis/images"
```

```
# extension of files
EXT_ORIGINAL="jpg"

# announce start running script
echo "start running " $PWD/$0

echo "looking for files with extension $EXT_ORIGINAL in $DIR_MEDIA/"

# check if directory exists
if [ -d "$DIR_MEDIA/" ];

# if directory exists then
then

  # announce directory exists
  echo "success, $DIR_MEDIA/ exists"

  find "$DIR_MEDIA" -iname "*.$EXT_ORIGINAL" | while read f
    do
      exiftool -all= -overwrite_original "$f"
      echo "formatted $f"
    done

else
  # announce directory does not exist
  echo "fail, $DIR_MEDIA/ does not exist"

# end of if statement for existence of directory
fi

# announce finished running script
```

```
echo "finished running " $PWD/$0
```

B.4 Converting formats with pandoc

pandoc is a command line tool for converting between formats. This script was written to convert from .tex files to .docx files, so that each chapter of this thesis document could be uploaded to Google Docs for feedback from the committee.

```
echo "pandoc latex to docx"

cd "$PWD/../thesis"

# iterate through all .tex files in thesis/
find "$PWD" -iname "*.tex" | while read f
do
  # retrieve basename
  base=$(basename "$f" .tex)
  # delete original docx file
  rm -f "$PWD/docx/$base.docx"
  # create new docx file with pandoc
  pandoc -s -o "$PWD/docx/$base.docx" "$f"
done

# do all the files manually, and with bibliography
pandoc -o "$PWD/docx/aaron-thesis.docx" "$PWD/cover.tex" "$PWD/chap1.tex"
    "$PWD/chap2.tex" "$PWD/chap3.tex" "$PWD/chap4.tex" "$PWD/chap5.tex"
    "$PWD/chap6.tex" "$PWD/appa.tex" "$PWD/appb.tex" "$PWD/appc.tex"
    "$PWD/appd.tex" "$PWD/appe.tex" --bibliography "$PWD/main.bib"
```

Appendix C

Documentation

For this thesis I wrote the following documentation, included at the docs/ folder of the repository, and also included here.

Bill of materials

Notes about the microcontroller

This project is based on the microcontroller `Arduino Nano 33 BLE Sense`. Please don't confuse it with the similarly named `Arduino Nano 33 BLE`! It is recommended to get the one ***with headers***, so you can immediately start using it on a breadboard without needing to solder the headers on it.

Minimum materials

This is the list of minimum required materials for Tiny Trainable Instruments

Item	Quantity	Cost (USD)	Retailer	Comment
Arduino Nano 33 BLE Sense with headers	1	33.40	Arduino	Microcontroller
Breadboard	1	5.95	Adafruit	Prototyping
Jumper wires	1	3.95	Adafruit	Connections
Micro USB cable	1	2.95	Adafruit	Power

Sound and movement outputs

These are the recommended materials for starters, and are taught in the workshop.

Item	Quantity	Cost (USD)	Retailer	Comment
Piezo buzzer	1	1.50	Adafruit	Output sound
Micro servo	1	5.95	Adafruit	Output movement

Additional outputs

These additional outputs include more expensive or more complex materials, and they are recommended for more advanced users. They are not covered on the beginner workshop, but are supported by the Tiny Trainable Instruments project and software library.

Item	Quantity	Cost (USD)	Retailer	Comment
LED	1	6.95	Adafruit	Output light
MIDI DIN	1	1.75	Adafruit	Output MIDI data
Thermal printer	1	61.95	Adafruit	Output printed text
128x32 OLED screen	1	12.50	Adafruit	Output screen

Installation

This guide includes information as of June 2021, and we will explicitly include the software versions we are using.

We advise to install the same versions we are using, and if there is any issue with the library or related software, please us know via email or an issue the repository.

For additional documentation, please visit the official Arduino docs website at docs.arduino.cc, and in particular the documentation of the Arduino Nano 33 BLE Sense microcontroller at docs.arduino.cc/hardware/nano-33-ble-sense.

Arduino IDE

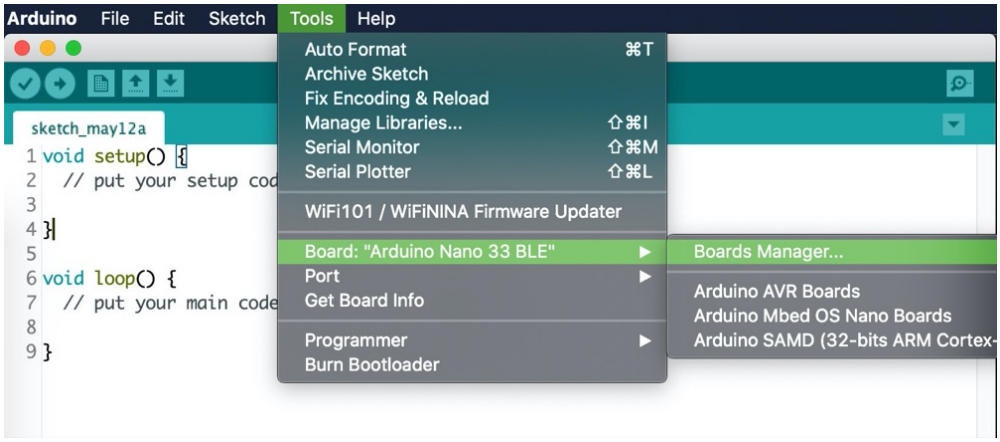
Download and install the Arduino IDE, available at https://www.arduino.cc/en/software. Select the stable release corresponding to your computer's operating system.

As of June 2021, we are using Arduino IDE 1.8.15.

Arduino Mbed OS Nano boards

After installing the Arduino IDE, we need to install the core and necessary libraries for the Arduino Nano 33 BLE Sense microcontroller. Open the Arduino IDE and navigate on the menu to the `Boards Manager`:

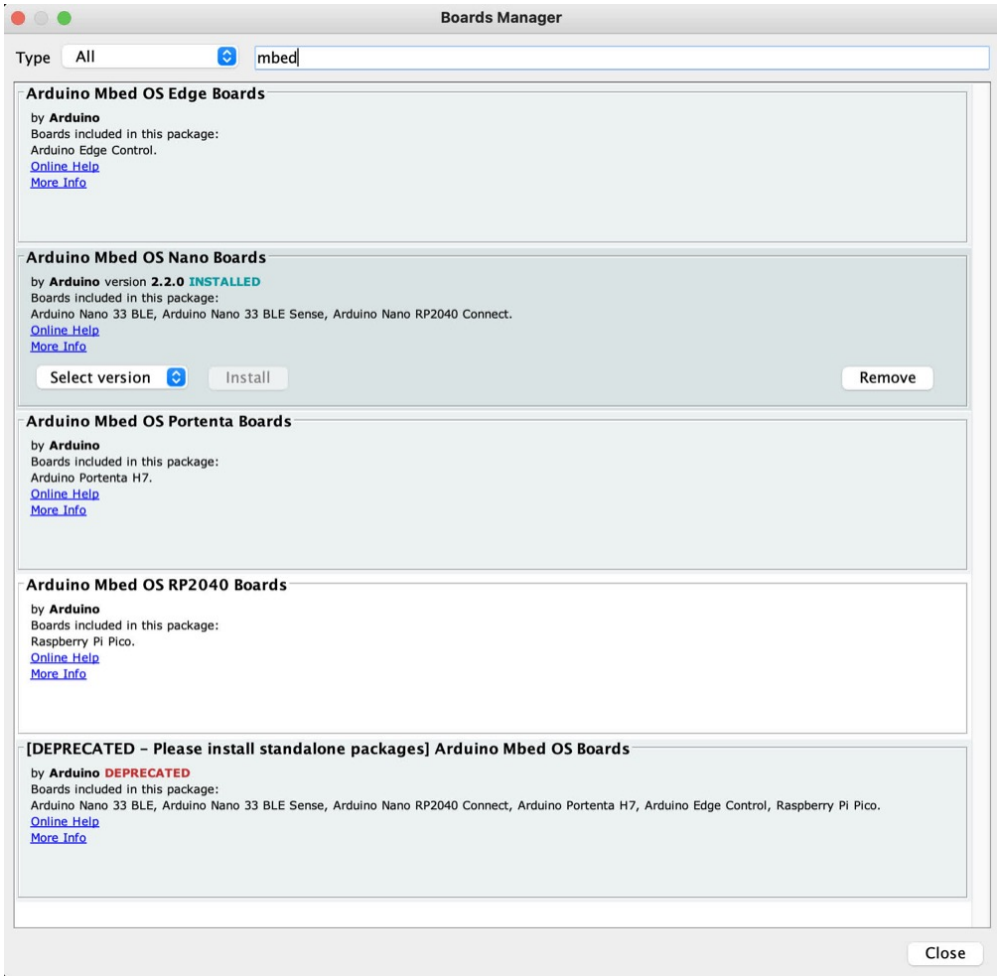
```
Tools > Board: "<board_name>" > Boards Manager...
```



Use the search bar to find the option `Arduino Mbed OS Nano Boards` and install it, this might take a while.

Please note that if you look for "Mbed", several different options will appear, be careful with the similar named one called `Arduino Mbed OS Boards` which is deprecated and we should not install.

As of June 2021, we are using version 2.2.0.



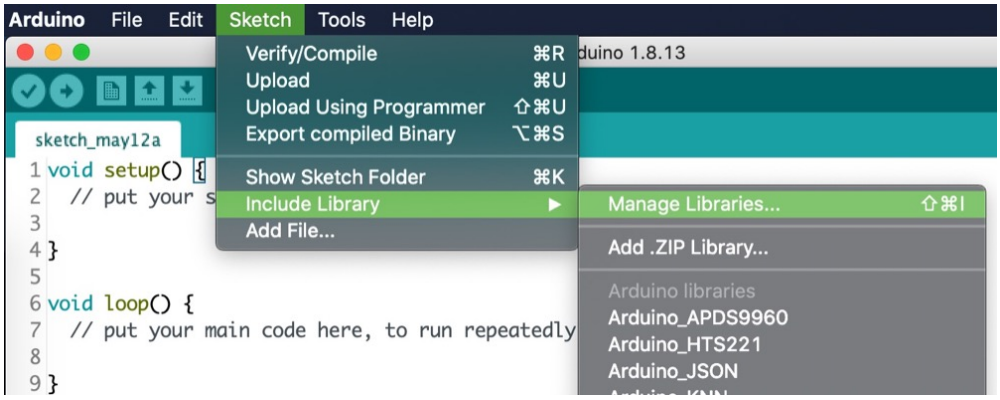
After the installation is complete, we can select the board we are going to work with (Arduino Nano 33 BLE), from the `Tools` menu:

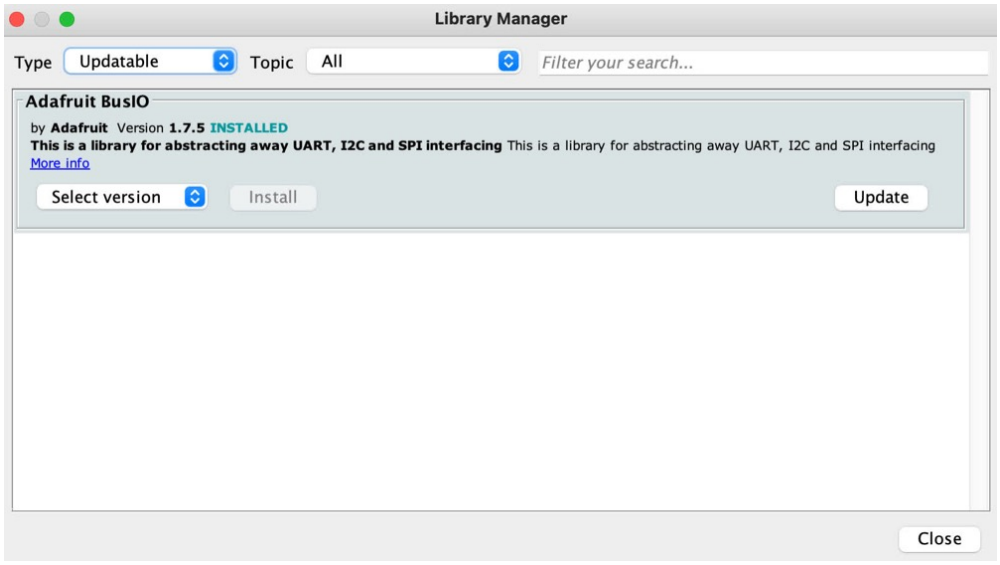
```
Tools > Board: "<board_name>" > Arduino Mbed OS Nano Boards > Arduino Nano 33 BLE
```

Please note that this option is valid for both Arduino Nano 33 BLE, and for the board we are using, the Arduino Nano 33 BLE Sense.

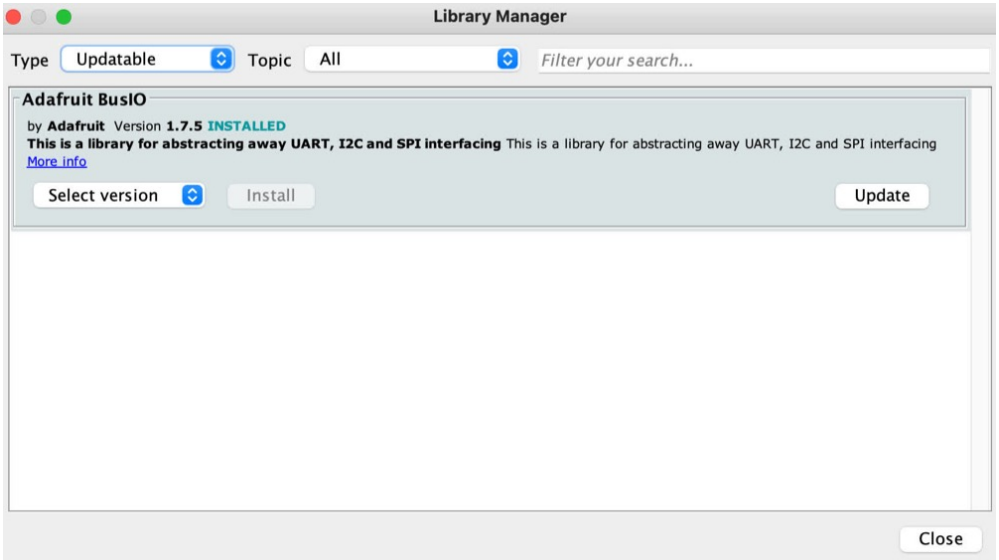
Arduino libraries

Before installing the TinyTrainable library for this project, please first update all your installed libraries. On the Arduino IDE, navigate on the menu to `Tools > Manage Libraries... >`, and then on the `Type` dropdown menu select `Updatable`.



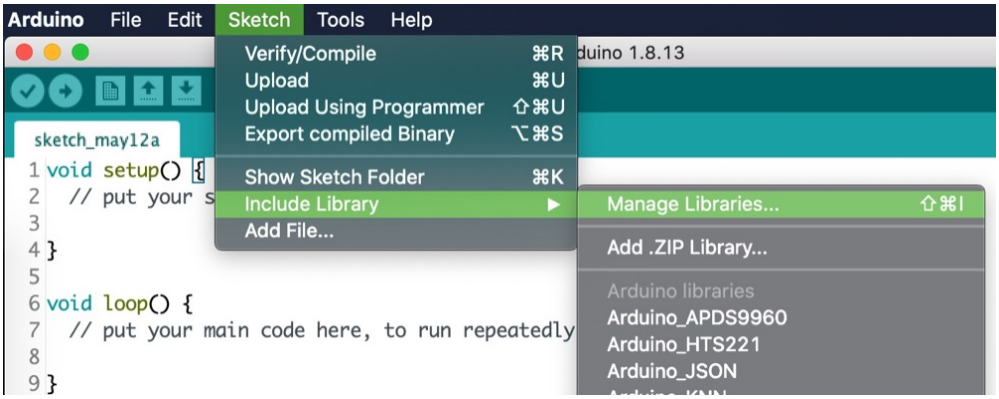


To update each outdated library to their latest version, hover on top of each library, and click on the button `Update`. For this example we are showing the updating of the library Adafruit BusIO, which is installed on my computer, but most probably is not on yours, and you don't need it for this project either.

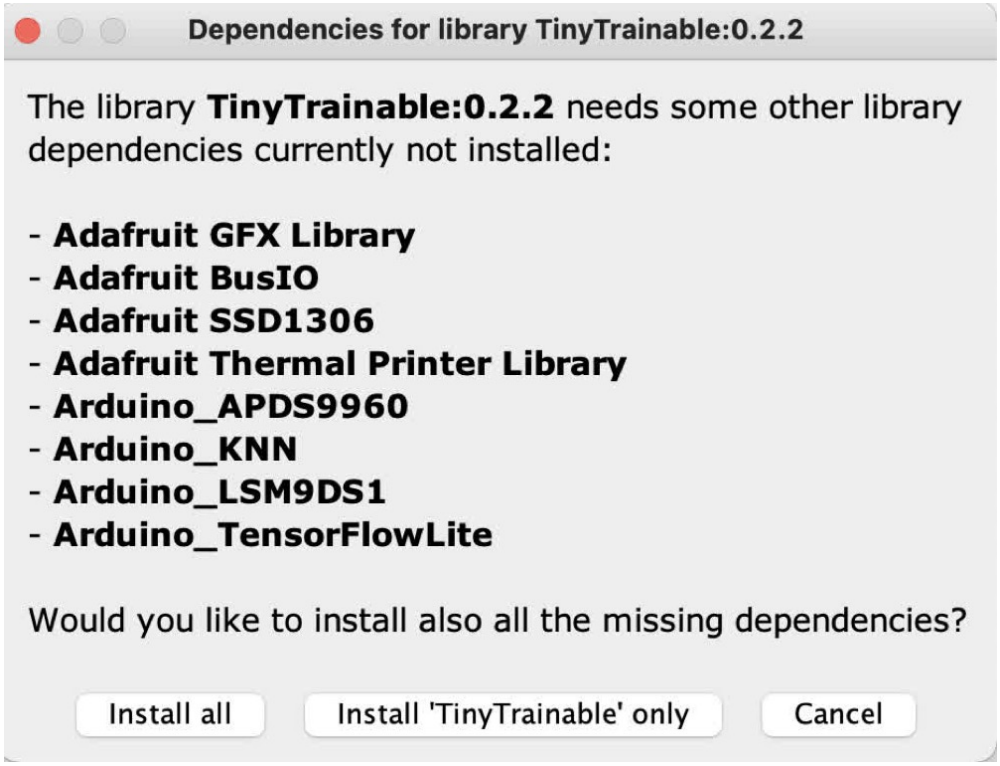


Please repeat this process until there are no updatable libraries left.

Next we will install all the libraries needed for this project. On the Arduino IDE, navigate on the menu to `Tools > Manage Libraries... >`



Go to the search bar of the Libraries Manager and type `TinyTrainable`. This installation will give you the option to also install its dependencies, select `Install all` to download them.



As of June 2021, the latest version 0.2.2 of the TinyTrainable library has these dependencies:

Libraries for using the embedded sensors of our microcontroller:

- `Arduino_APDS9960`: color, proximity
- `Arduino_LSM9DS1` acceleration, magnetic field, gyroscope orientation

Libraries for machine learning:

- `Arduino_KNN`: k-nearest neighbor algorithm.
- `Arduino_TensorFlowLite`: microcontroller version of the TensorFlow machine learning library. Please download the latest non-precompiled version.

Libraries for multimedia output:

- `Adafruit GFX Library`: for output with screen.
- `Adafruit SSD1306`: for output with screen.
- `Adafruit Thermal Printer Library`: for output with thermal printer.
- `Servo`: for output with servo motors.

Python for machine learning

For input-color, you only need Arduino libraries.

For input-gesture and speech, you either need to install specific Python libraries on your computer, or use the free Google Colab service, because we will create databases and train algorithms on a computer.

For beginners, we suggest using Google Colab, because it will be an easier installation, and the algorithms will run faster.

If you decide to run the algorithms on your machine, you will need Python, TensorFlow and Jupyter.

These are the versions we will be using, as of June 2021:

- Python 3.8.6
- TensorFlow 2.3.2
- Jupyter Lab 3.0.5

Your computer might have Python already installed, but it might be one that is not compatible with the TensorFlow version we are using, so we suggest using a Python version manager, like the tool pyenv https://github.com/pyenv/pyenv.

After installing pyenv, open the terminal and go to this repository. If you dont know how to download a repository to your machine, follow this tutorial about cloning repositories from GitHub.

```
cd tiny-trainable-instruments/
```

Check that pyenv is able to read the .python-version file

```
pyenv versions
```

You should see a list, with the version we are using and an asterisk, to highlight that this is the Python version we will use. If there is no asterisk and it says that the required version of Python is not installed, use the command:

```
pyenv install <python version number>
```

If you are using an old version of pyenv, there's a chance that the install won't work; copy the entire command pyenv gives you (including the &&'s) and enter it into the terminal. Then once pyenv is updated, try the above command again.

Now that you have the correct version of Python, create a virtual environment (which we will name env) using the Python package venv. Most dependency problems can be solved by using a virtualenv; we can’t support issues not

using a virtualenv due to the huge variety of system configurations. On your terminal type:

```
python -m venv env
```

Activate the virtual environment with this command, which you will use every time you want to enter the venv:

```
source env/bin/activate
```

Now your terminal should have every new line starting with (env). Your command prompt should look something like this:

../docs/images/1-arduino-boards-manager

The pip of your Python virtual environment might need updating; you can update to the latest version with the command

```
pip install --upgrade pip
```

Then use pip to install the Jupyter packages, along with their dependencies:

```
pip install -r requirements.txt
```

Now you can run the Jupyter Lab tool with `jupyter-lab`. This will open a tab on your browser to navigate through the files in your computer and allow you run code and read the documentation.

The code for input-gesture and input-speech is written using Jupyter notebooks, which have the extension .ipynb, and are located on the folder `instruments/`. The documentation is written in several Markdown files with extension .md. These files are on the folder `docs/`, which includes an index on README.md.

If you double click on a Markdown file, it will open an Editor window with the Markdown code. To view the rendered text you can right click and select "Open with Markdown Preview". If you have internet connection, it might be more convenient to access the online documentation on the online repository.

To close the Jupyter notebook server, press `ctrl+c` in the terminal (even on OSX; it's not `cmd`) and confirm with `y`.

To exit the virtual environment once you're done, use the command `deactivate`. Note that the command `jupyter-lab` will not work until you reactive the virtual environment.

Wiring

Conventions

Wires:

- Red = 3.3 V power from Arduino
- Green = Ground from Arduino

Breadboard

Breadboards are built so that within each of the rows, the 5 tie points in the columns labelled `a–e` are electrically connected inside the board and act as a single electrical node, and same with `f–j`.

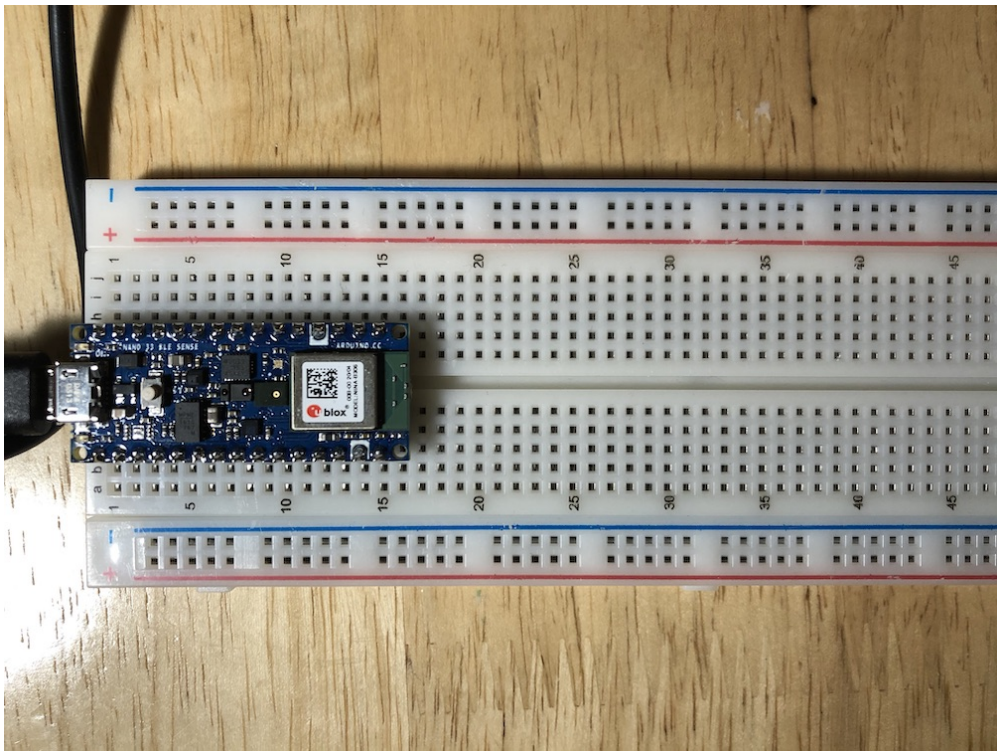
In addition, there are 2 columns to each side of the breadboard, where each column is one electrical node. Conventionally, we connect the positive voltage to the column labelled `+`, and the ground to the column labelled `–`.

A full breadboard guide is available at https://learn.adafruit.com/breadboards-for-beginners/breadboards.

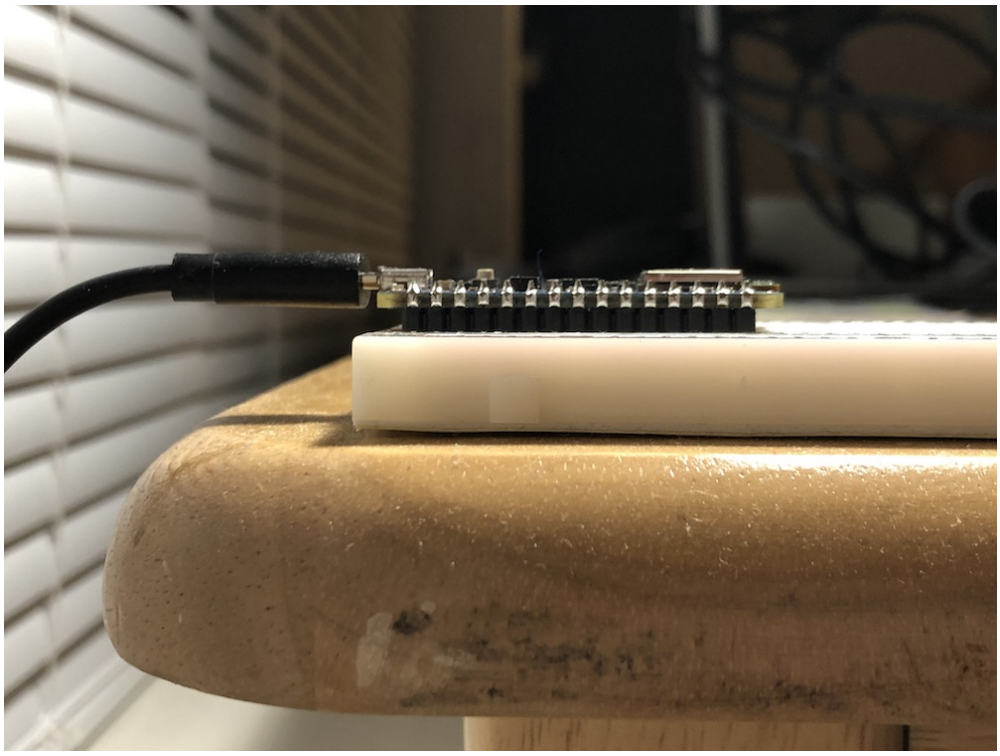
Arduino microcontroller

The Arduino Nano BLE 33 Sense we are using has 30 pins in total, 15 on each side. The official pinout is available at https://content.arduino.cc/assets/Pinout-NANOsense_latest.pdf.

We recommend placing the microcontroller at the top of the breadboard (C1 to C15 and G1 to G15) with the USB Micro port facing up.



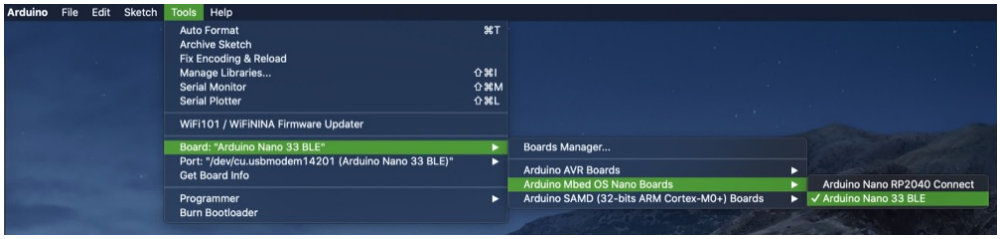
Note that the microcontroller should be flush with the breadboard; none of the headers should be visible.



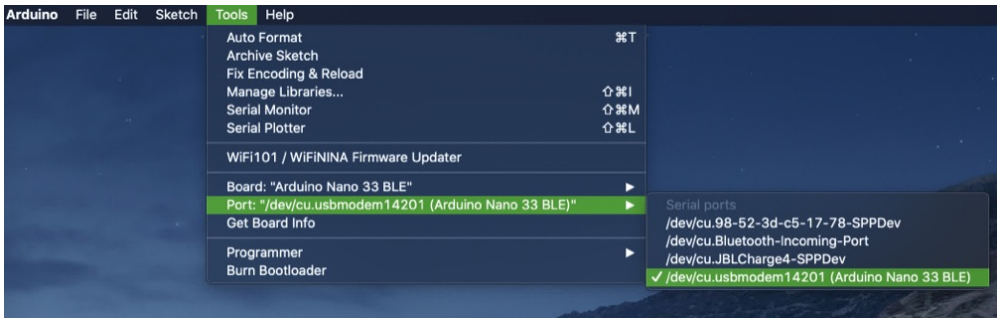
Your first example

Connect your Arduino microcontroller to your computer with the USB cable and open the Arduino IDE software.

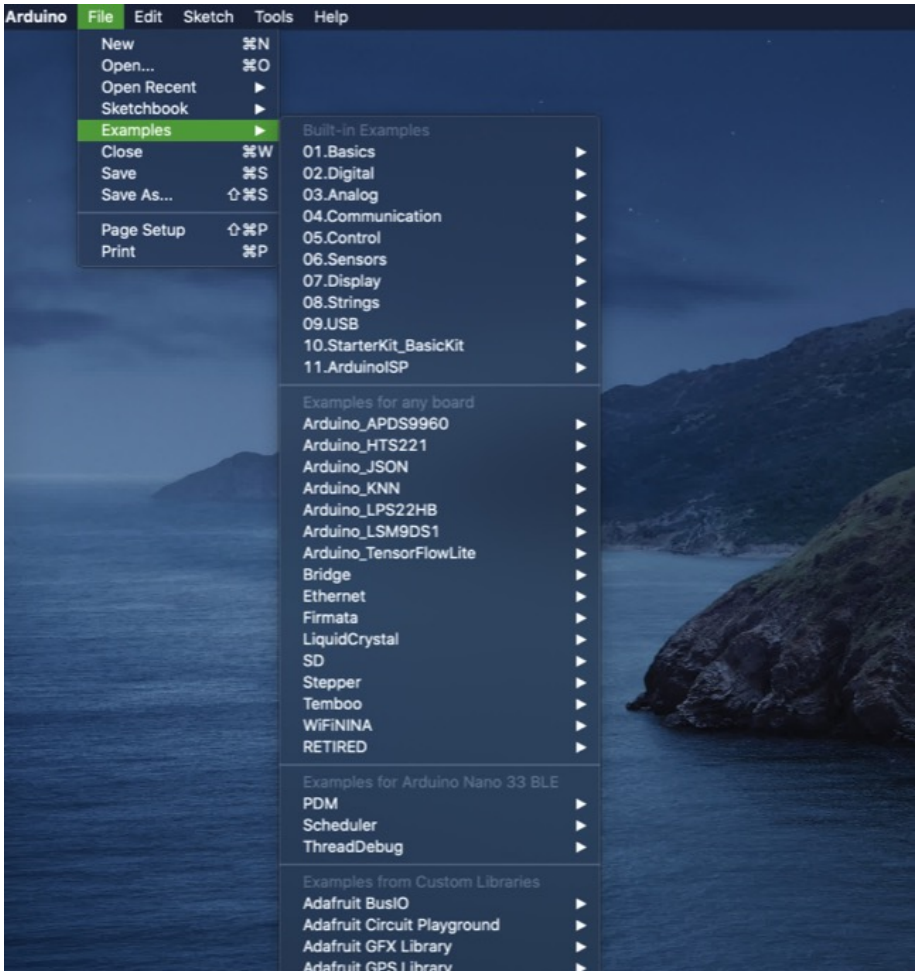
On the board, select the `Arduino Nano 33 BLE`.

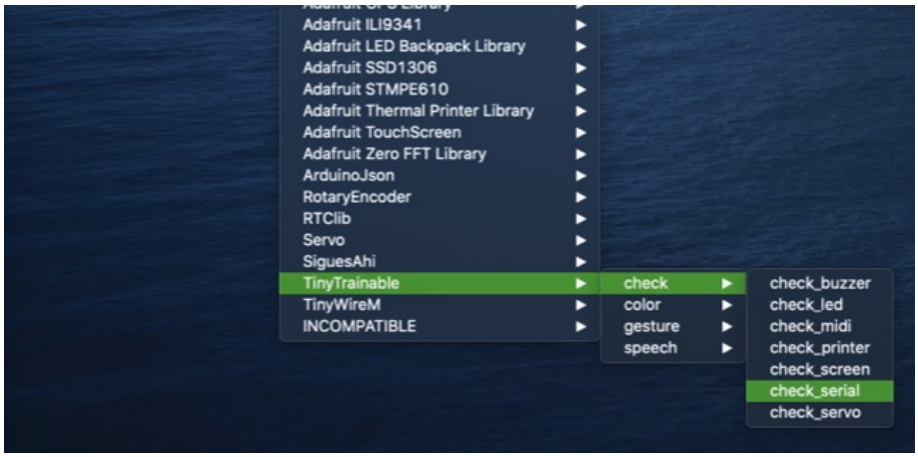


Then make sure yor port points to your Arduino, the number is irrelevant, and the actual text changes between computers.



Now let's open the example `check_serial`, included with our TinyTrainable library.





Click on the arrow to the right for uploading the code, which will be shown on the bottom of the Arduino IDE, with the message `Compiling sketch`.

```
29  myTiny.setStateLEDBuiltIn(true);
30  delay(pauseTime);
31  myTiny.setStateLEDBuiltIn(false);
32  delay(pauseTime);
33
34  // cycle through the 6 colors of the RGB LED
35  myTiny.setStateLEDRGB(true, red);
36  delay(pauseTime);
```

```
Compiling sketch...
ResolveLibrary(SPI.h)
  -> candidates: [SPI]
/Users/montoyamoraga/Library/Arduino15/packages/arduino/tools/arm-none-eabi-gcc/7-2017q4/bin/arm-n
Alternatives for Adafruit_GFX.h: [Adafruit_GFX_Library@1.10.10]
ResolveLibrary(Adafruit_GFX.h)
  -> candidates: [Adafruit_GFX_Library@1.10.10]
/Users/montoyamoraga/Library/Arduino15/packages/arduino/tools/arm-none-eabi-gcc/7-2017q4/bin/arm-n
Alternatives for Adafruit_SSD1306.h: [Adafruit_SSD1306@2.4.5]
ResolveLibrary(Adafruit_SSD1306.h)
```

The compilation might take several minutes, and after it is done, the message will change to `Uploading...`

```
36  delay(pauseTime);
```

```
Uploading...
writeBuffer(scr_addr=0x34, dst_addr=0x1c000, size=0x1000)
[=======                  ] 25% (29/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x1d000, size=0x1000)
[=======                  ] 26% (30/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x1e000, size=0x1000)
[=======                  ] 27% (31/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x1f000, size=0x1000)
[=======                  ] 28% (32/114 pages)write(addr=0x34,size=0x1000)
```

This process is shorter, and after it you will see the message `Done uploading`.

```
35  myTiny.setStateLEDRGB(true, red);
36  delay(pauseTime);
```

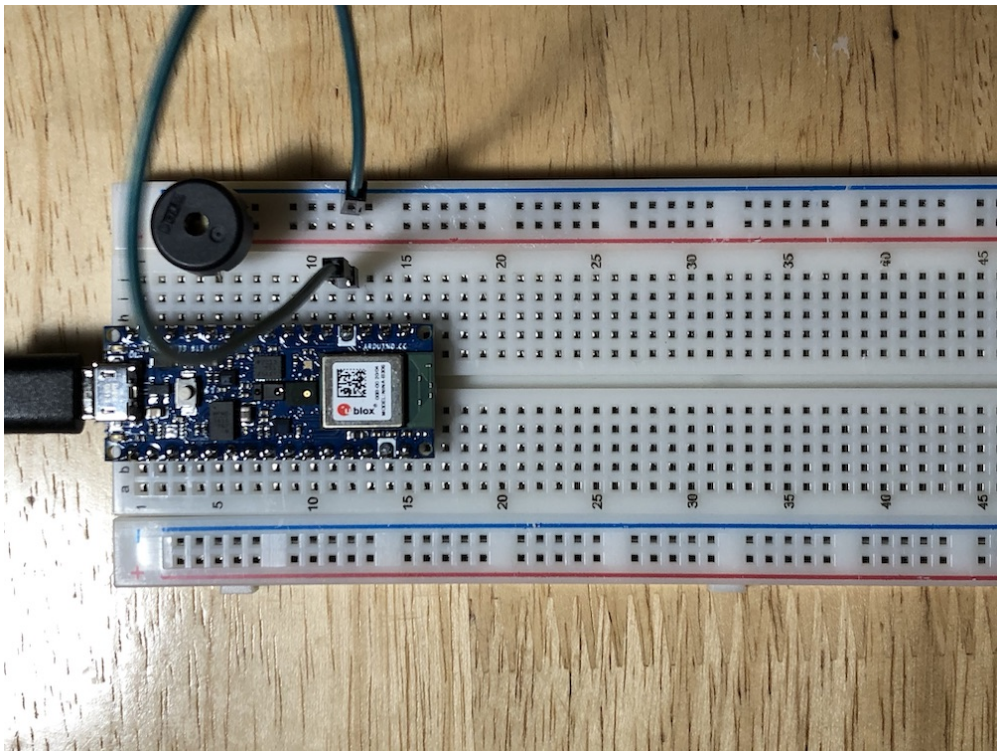
```
Done uploading.
[=========================  ] 94% (108/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x6c000, size=0x1000)
[=========================  ] 95% (109/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x6d000, size=0x1000)
[=========================  ] 96% (110/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x6e000, size=0x1000)
[========================== ] 97% (111/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x6f000, size=0x1000)
[========================== ] 98% (112/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x70000, size=0x1000)
[========================== ] 99% (113/114 pages)write(addr=0x34,size=0x1000)
writeBuffer(scr_addr=0x34, dst_addr=0x71000, size=0x1000)
[==========================] 100% (114/114 pages)
Done in 18.200 seconds
reset()
```

On the upper right corner of the window, click on the magnifying glass icon for opening the `Serial monitor`. Make sure the settings on the bottom match the ones on your computer, and that's it!

You uploaded your first example to your Arduino, which is now busy sending the messages you seen on the screen, and also showing all the different lights it has :)

Ground

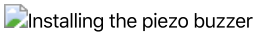
Notice that the 14th pin on the left side and the 12th pin on the right side are labelled with white paint; this marks ground, also identified on the pinout. Take a wire (preferably green by convention for ground) and connect it from I12 to anywhere on the top righthand negative rail (the upper 25 pins), like this:



Outputs

Buzzer

Next, connect one of the legs of the piezo buzzer to the node labelled D8 on the pinout (which should be row 5 on the breadboard). Connect the other leg to the ground rail. Your wiring should look like this:


Installing the piezo buzzer

Now you're good to go! Upload `check_buzzer` to the microcontroller, open the serial monitor (top right button in the Arduino IDE), and follow the instructions from there!

LED

MIDI

MIDI Din jack

5 pins, only 3 are used.

Printer

We are using a thermal printer from Adafruit.

https://www.adafruit.com/product/2753

It has 5 cables:

VH - red - connect to the power supply 5V - 9V DTR - yellow - connect to GND on the Arduino TX - green - data out of the printer RX - blue - data in to the printer GND - black - connect to GND on the Arduino

We use a power supply, whose ground is connected to the one on the Arduino.

The power supply is 9V, center positive. Here is one available:

https://www.adafruit.com/product/276

Serial

Use a micro USB cable to connect to a computer.

Servo

The servo we are using has three cables:

- Yellow: signal
- Orange: power
- Brown: ground

Contributing

Issues

If you find an error or have a comment, please start a discussion by submitting an issue on our repositories!

- https://github.com/montoyamoraga/tiny-trainable-instruments/issues
- https://github.com/montoyamoraga/TinyTrainable/issues

Pull requests

Here is a step by step guide to make pull requests to this repository.

- Create a free GitHub account
- Fork the repository
- Clone your new repository to your computer

```
git clone https://github.com/your_username/tiny-trainable-instruments.git
```

Optionally, you can also clone the submodules of this repository, with the command

```
git submodule update --init --recursive
```

- Change directory (cd) into the project folder

```
cd tiny-trainable-instruments
```

- Make your changes
- Stage and make a commit to your repository on your computer

```
git add .
```

```
git commit -m "your comment"
```

- Push your commit to your personal fork on GitHub

```
git push
```

- Open your repository online
- Open your pull request and wait for comments or approval

Contributing documentation

For more information about how to contribute documentation to an open source artistic project, we recommend looking at the documentation by the p5.js project, available at https://github.com/processing/p5.js/blob/main/contributor_docs/contributing_documentation.md

Adding submodules

If you think there are more repositories we should include as submodules for archival purposes, use the following command, replacing GITPATH with the location of the repository you want to include, and FOLDERPATH with the destination.

```
git submodule add GITPATH FOLDERPATH
```

Helper scripts

The helper scripts are located on the assets/ folder. To run them, cd to assets/ and then use the following commands

Compiling code

This script uses arduino-cli for checking the compilation of all examples of TinyTrainable.

```
sh compile-code.sh
```

Delete metadata

This script uses exiftool to delete the metadata of all pictures in docs/

```
sh delete-metadata.sh
```

Format code

This script uses clang-format to organize all the code in TinyTrainable

```
sh format-code.sh
```

Markdown to PDF

This script uses pandoc to convert the documentation from Markdown to PDF format.

```
sh markdown-to-pdf.sh
```

Appendix D

Open source contributions

During this thesis I contributed the following pull requests to open source projects that are either direct dependencies or inspirations.

Author	Repository	Contribution
Adafruit	[88, Adafruit_SSD1306]	[89, Format binary numbers]
tinyMLx	[50, TinyMLx Arduino Library]	[90, Update architecture name]
Yining Shi	[46, ML for Physical Computing]	[91, Fixed some typos]

Table D.1: Pull requests to open source projects

Appendix E

Rules of thumb

During this thesis I have tried to follow these rules of thumb:

- Openly share small steps
- Learn by failing often
- Contribute back
- Use tools you like
- Cite other people
- Sleep as much as possible

Bibliography

[1] Dominic Pajak Sandeep Mistry. How-to get started with machine learning on arduino. https://blog.tensorflow.org/2019/11/how-to-get-started-with-machine.html, 2019. [Online; accessed 12-August-2021].

[2] Sandeep Mistry Dominic Pajak. Fruit identification using arduino and tensorflow. https://blog.arduino.cc/2019/11/07/fruit-identification-using-arduino-and-tensorflow/, 2019. [Online; accessed 12-August-2021].

[3] Anthony Wing Kosner Forbes. Beyond girls around me: Artist ai weiwei turns the creepy surveillance on himself. https://www.forbes.com/sites/anthonykosner/2012/04/04/beyond-girls-around-me-artist-ai-weiwei-turns-the-creepy-surveillance-on-himself/, 2012. [Online; accessed 02-August-2021].

[4] Jeffrey Dastin Reuters. Amazon scraps secret ai recruiting tool that showed bias against women. https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G, 2018. [Online; accessed 02-August-2021].

[5] Tom Simonite Wired. What really happened when google ousted timnit gebru. https://www.wired.com/story/google-timnit-gebru-ai-what-really-happened/, 2020. [Online; accessed 12-August-2021].

[6] Janelle Shane. bias laundering edition. https://twitter.com/JanelleCShane/status/1405598023619649537, 2021. [Online; accessed 02-August-2021].

[7] Algorithmic Justice League. Algorithmic justice league - unmasking ai harms and biases. https://www.ajl.org/, 2021. [Online; accessed 02-August-2021].

[8] Shalini Kantayya. Coded bias. https://www.codedbias.com/, 2020. [Online; accessed 12-August-2021].

[9] Art in America. Technology and public art with rafael lozano-hemmer. https://www.youtube.com/watch?v=QgVdEmqmuEE, 2020. [Online; accessed 02-August-2021].

[10] Paige Bailey #BlackLivesMatter. untitled. https://twitter.com/DynamicWebPaige/status/1407179134736896010, 2021. [Online; accessed 02-August-2021].

[11] Catharine Smith HuffPost. 7,500 online shoppers accidentally sold their souls to gamestation. https://www.huffpost.com/entry/gamestation-grabs-souls-o_n_541549, 2010. [Online; accessed 02-August-2021].

[12] Lorrie Faith Cranor Aleecia M. McDonald. The cost of reading privacy policies. *I/S: A Journal of Law and Policy for the Information Society*, 4(3):543+, 2008.

[13] Arduino. Arduino uno rev3 | arduino official store. https://store.arduino.cc/usa/arduino-uno-rev3, 2021. [Online; accessed 31-July-2021].

[14] Makey Makey. Makey makey - joylabz official makey makey store. https://makeymakey.com/, 2021. [Online; accessed 13-August-2021].

[15] NYU ITP. Itp physical computing. https://itp.nyu.edu/physcomp/, 2021. [Online; accessed 12-August-2021].

[16] PJRC. Pjrc: Electronic projects. https://www.pjrc.com/, 2021. [Online; accessed 12-August-2021].

[17] PJRC. Teensy-lc usb development board. https://www.pjrc.com/store/teensylc_pins.html, 2021. [Online; accessed 31-July-2021].

[18] Processing. Welcome to processing! https://processing.org/, 2021. [Online; accessed 12-August-2021].

[19] Lauren McCarthy. Performing user. https://itp.nyu.edu/classes/performinguser/, 2016. [Online; accessed 05-August-2021].

[20] Aarón Montoya-Moraga. its-ok-to-die. https://github.com/montoyamoraga/its-ok/tree/main/its-ok-to-die, 2017. [Online; accessed 12-August-2021].

[21] p5.js. home | p5.js. https://p5js.org/es/, 2021. [Online; accessed 14-August-2021].

[22] Processing Foundation Press. Introducción a p5.js. https://processingfoundation.press/product/introduccion-a-p5-js/, 2018. [Online; accessed 14-August-2021].

[23] p5.js. Community statement. https://p5js.org/community/, 2021. [Online; accessed 06-August-2021].

[24] ml5.js. Community statement. https://ml5js.org/about/, 2021. [Online; accessed 06-August-2021].

[25] Jayson Musson Hennessy Youngman. Art thoughtz: How to make an art. https://www.youtube.com/watch?v=vVFasyCvEOg, 2011. [Online; accessed 06-August-2021].

[26] montoyamoraga. protestpy. https://pypi.org/project/protest, 2017. [Online; accessed 06-August-2021].

[27] montoyamoraga. kaputtpy. https://pypi.org/project/kaputt/, 2017. [Online; accessed 06-August-2021].

[28] Gene Kogan. Gene kogan. https://genekogan.com/, 2021. [Online; accessed 14-August-2021].

[29] Gene Kogan. Machine learning for artists at itp-nyu spring 2016. https://ml4a.github.io/classes/itp-S16/, 2016. [Online; accessed 14-August-2021].

[30] Rebecca Fiebrink. Dr. rebecca fiebrink - goldsmiths, university of london. https://www.doc.gold.ac.uk/~mas01rf/homepage/, 2021. [Online; accessed 14-August-2021].

[31] Rebecca Fiebrink Kadenze, Goldsmiths University of London. Machine learning for musicians and artists. https://www.kadenze.com/courses/machine-learning-for-musicians-and-artists-v/info, 2021. [Online; accessed 14-August-2021].

[32] Rebecca Fiebrink. Wekinator | software for real-time, interactive machine learning. http://www.wekinator.org/, 2021. [Online; accessed 14-August-2021].

[33] alt AI. Exploring the intersection of artificial intellienge and art. https://alt-ai.net/, 2016. [Online; accessed 31-July-2021].

[34] RunwayML. Runway | create impossible video. https://runwayml.com/, 2021. [Online; accessed 14-August-2021].

[35] Cristóbal Valenzuela. Cristóbal valenzuela. https://cvalenzuelab.com/, 2021. [Online; accessed 14-August-2021].

[36] Alejandro Matamala. Alejandro matamala ortiz. https://www.matamala.info/, 2021. [Online; accessed 14-August-2021].

[37] Anastasis Germanidis. Anastasis germanidis. https://agermanidis.com/, 2021. [Online; accessed 14-August-2021].

[38] Making & Make-Believe School of Machines. School of machines, making & make-believe. https://schoolofma.org/, 2021. [Online; accessed 14-August-2021].

[39] Andreas Reefsgaard. Home | andreas reefsgaard. https://andreasrefsgaard.dk/, 2021. [Online; accessed 14-August-2021].

[40] Rachel Uwa. Rachel uwa. https://racheluwa.medium.com/, 2021. [Online; accessed 14-August-2021].

[41] Sam Lavigne. Training poses. https://lav.io/projects/training-poses/, 2018. [Online; accessed 26-July-2021].

[42] CMU-Perceptual-Computing-Lab. openpose. https://github.com/CMU-Perceptual-Computing-Lab/openpose, 2021. [Online; accessed 14-August-2021].

[43] Abeba Birhane Vinay Uday Prabhu. Large image datasets: A pyrrhic win for computer vision? *CoRR*, abs/2006.16923, 2020.

[44] Aarón Montoya-Moraga. Arpillera mirror (2019). https://montoyamoraga.github.io/arpillera-mirror, 2019. [Online; accessed 14-August-2021].

[45] Casey Reas. *Making Pictures With Adversarial Networks*. Anteism, first paperback edition, 2019.

[46] Yining Shi. Introduction to machine learning for physical computing. https://github.com/yining1023/Machine-Learning-for-Physical-Computing, 2021. [Online; accessed 31-July-2021].

[47] Yining Shi. Yining shi. https://1023.io/, 2021. [Online; accessed 14-August-2021].

[48] Mitchel Resnick. *Lifelong Kindergarten*. The MIT Press, first paperback edition, 2018.

[49] HarvardX edx. Tiny machine learning (tinyml) professional certificate. https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning, 2020. [Online; accessed 03-August-2021].

[50] tinyMLx. Harvard_tinymlx arduino library. https://github.com/tinyMLx/arduino-library/, 2021. [Online; accessed 31-July-2021].

[51] arduino. Arduino_knn library for arduino. https://github.com/arduino-libraries/Arduino{_}KNN, 2020. [Online; accessed 16-August-2021].

[52] montoyamoraga. Arduino library to make instruments that check if institutions still exist. https://github.com/montoyamoraga/SiguesAhi, 2021. [Online; accessed 29-July-2021].

[53] Arduino. Arduino nano 33 iot with headers | arduino official store. https://store.arduino.cc/usa/nano-33-iot-with-headers, 2021. [Online; accessed 01-August-2021].

[54] Aarón Montoya-Moraga Gaurav Patekar. Open drawing machine. https://github.com/montoyamoraga/open-drawing-machine, 2021. [Online; accessed 01-August-2021].

[55] Aarón Montoya-Moraga. Introduction to computer networks for artists. https://github.com/montoyamoraga/intro-to-computer-networks-for-artists, 2020. [Online; accessed 01-August-2021].

[56] Lisa Jamhoury. Lisa jamhoury - new media artist. https://lisajamhoury.com/, 2021. [Online; accessed 13-August-2021].

[57] Shawn van Every Kinectron, Lisa Jamhoury. Kinectron. https://github.com/kinectron/kinectron, 2021. [Online; accessed 13-August-2021].

[58] Bastl Instruments. Official website. `https://www.bastl-instruments.com/`, 2021. [Online; accessed 24-July-2021].

[59] Bastl Instruments. Github account. `https://github.com/bastl-instruments`, 2021. [Online; accessed 13-August-2021].

[60] Trevor Pinch and Frank Trocco. *Analog days: the Invention and impact Of The Moog synthesizer*, chapter 3: Shaping the Synthesizer, page 68. Harvard University Press, first harvard university press paperback edition edition, 2004.

[61] Critter & Guitari. Kaleidoloop. `https://web.archive.org/web/20150206042159/http://www.critterandguitari.com/collections/instruments/products/kaleidoloop`, 2015. [Online; accessed 24-July-2021].

[62] Critter & Guitari. Official website. `https://www.critterandguitari.com/`, 2021. [Online; accessed 24-July-2021].

[63] Monome. Official website. `https://www.monome.org/`, 2021. [Online; accessed 24-July-2021].

[64] Shbobo. Official website, 2021. [Online; accessed 24-July-2021].

[65] Peter Blasser. Stores at the mall. Master's thesis, Wesleyan University, 2015.

[66] Boston University School of Law. Technology law clinic. `https://sites.bu.edu/techlaw/`, 2021. [Online; accessed 14-August-2021].

[67] Google. Teachable machine. `https://teachablemachine.withgoogle.com/`, 2020. [Online; accessed 03-August-2021].

[68] Sonic Youth. Illustrated equipment guide. `http://www.sonicyouth.com/mustang/eq/gear.html`, 2021. [Online; accessed 03-August-2021].

[69] GitHub. Github and trade controls. `https://docs.github.com/en/github/site-policy/github-and-trade-controls`, 2021. [Online; accessed 04-August-2021].

[70] Arduino. Arduino nano 33 ble sense with headers | arduino official store. `https://store.arduino.cc/usa/nano-33-ble-sense-with-headers`, 2021. [Online; accessed 28-July-2021].

[71] Adafruit. Usb cable - usb a to micro-b - 3 foot long: Id 592. `https://www.adafruit.com/product/592`, 2021. [Online; accessed 28-July-2021].

[72] Adafruit. Full sized breadboard: Id 239. https://www.adafruit.com/product/239, 2021. [Online; accessed 28-July-2021].

[73] Adafruit. Premium male/male jumper wires - 40 x 6" (150mm): Id 758. https://www.adafruit.com/product/758, 2021. [Online; accessed 28-July-2021].

[74] arduino. Arduino_apds9960 library for arduino. https://github.com/arduino-libraries/Arduino{_}APDS9960, 2020. [Online; accessed 16-August-2021].

[75] tensorflow. Magic wand example. https://github.com/tensorflow/tflite-micro/tree/main/tensorflow/lite/micro/examples/magic{_}wand, 2021. [Online; accessed 16-August-2021].

[76] arduino. Arduino_lsm9ds1 library for arduino. https://github.com/arduino-libraries/Arduino{_}LSM9DS1, 2020. [Online; accessed 16-August-2021].

[77] Google. Welcome to colaboratory. https://colab.research.google.com, 2021. [Online; accessed 12-August-2021].

[78] arduino. Arduinocore-mbed, pdm library. https://github.com/arduino/ArduinoCore-mbed/tree/master/libraries/PDM, 2021. [Online; accessed 16-August-2021].

[79] Adafruit. Piezo buzzer [ps1240]: Id 160. https://www.adafruit.com/product/160, 2021. [Online; accessed 30-July-2021].

[80] Adafruit. Super bright white 5mm led (25 pack): Id 754. https://www.adafruit.com/product/754, 2021. [Online; accessed 30-July-2021].

[81] Adafruit. Breadboard-friendly midi jack (5-pin din): Id 1134. https://www.adafruit.com/product/1134, 2021. [Online; accessed 30-July-2021].

[82] Tatsuya Takahashi. Tatsuya takahashi | instrument designer. https://www.tatsuyatakahashi.com/, 2021. [Online; accessed 12-August-2021].

[83] Adafruit. Mini thermal receipt printer starter pack: Id 600. https://www.adafruit.com/product/600, 2021. [Online; accessed 30-July-2021].

[84] Adafruit. Monochrome 0.91" 128x32 i2c oled display - stemma qt / qwiic: Id 4440. https://www.adafruit.com/product/4440, 2021. [Online; accessed 30-July-2021].

[85] Adafruit. Micro servo [ps1240]: Id 169. https://www.adafruit.com/product/169, 2021. [Online; accessed 30-July-2021].

[86] Roy Macdonald. roymacdonald. https://roymacdonald.github.io/, 2021. [Online; accessed 12-August-2021].

[87] Renata Gaui. Renata gaui - home. https://renatagaui.com//, 2021. [Online; accessed 15-August-2021].

[88] Adafruit. Arduino library for ssd1306 monochrome 128x64 and 128x32 oleds. https://github.com/adafruit/Adafruit{_}SSD1306, 2021. [Online; accessed 31-July-2021].

[89] montoyamoraga. change format of binary numbers. https://github.com/adafruit/Adafruit{_}SSD1306/pull/209, 2021. [Online; accessed 31-July-2021].

[90] montoyamoraga. update name of architecture. https://github.com/tinyMLx/arduino-library/pull/8, 2021. [Online; accessed 31-July-2021].

[91] montoyamoraga. Update readme.md. https://github.com/yining1023/Machine-Learning-for-Physical-Computing/pull/1, 2021. [Online; accessed 31-July-2021].