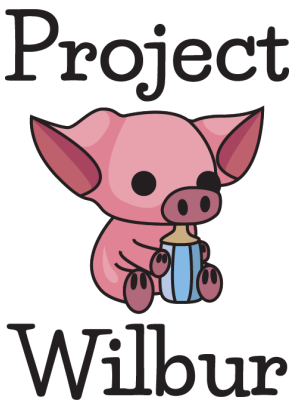

Technological Feasibility Analysis

Authors: Adriel Perez, Alexander ‘Gus’ Siegel,

Nathan Seitz, Taylor Nielsen

Northern Arizona University

CS476: Requirements Engineering

Client: Christopher Mayerl Ph.D., Northern Arizona University, Mayerl Lab

Mentor: Italo Santos

October 16, 2023

Overview

This document is the Technology Feasibility document where the sole objective is to introduce the main challenges of executing our project and how we plan to approach them.

Table of Contents:

Feasibility Introduction

The United States has a history of being touted as the “best of the best” in almost every area of life and business. It is a belief as much as it is a doctrine. However, over the last decade the United States has shown a huge disparity among maternal and infant health when compared to other developed countries. For example, countries in the European Union typically have a preterm birth rate below 9% with one exception in 2015 being Cyprus with a rate of 10.4% [1]. In 2022, there were 3.6 million babies born in the United States and the preterm birth rate was 10.38%, or about 1 in 10 children, which is a drastically large number considering we have some of the best medical professionals and researchers in the world [2].

The Center for Disease Control recommends babies be breastfed for the first 6 months of life [2]. There are many benefits to breast-feeding infants including development of the maternal-infant bond, transference of the mothers gut-microbiome to the infant and better developmental outcomes for growth and speech. Preterm babies have a much higher likelihood of developing infant feeding issues because they are more likely to be initially fed with feeding tubes and are introduced to bottles and breastfeeding later in their development [3]. There is also a percentage of infants born after the natural gestation window that suffer from feeding issues as well. The typical recommendation for these mothers is for them to switch to bottle feeding and try a number of differently structured bottle nipples.

Each company currently making bottle nipples for children with feeding difficulties creates each design by their own standards, typically with little to no research into the issue. There is not a set protocol for developing these solutions to feeding issues. Our client, Dr. Christopher Mayerl is in search of a solution that will assist physicians and nurses working with infants that struggle with feeding by combining the knowledge of human anatomy and physiology with an adjustable tool that will help provide the baby with a true solution that can adapt in a way that supports the infant in getting the best result possible.

Dr. Mayerl’s envisioned solution leaves only a small amount of variables for our team to research and decide on. This includes a thorough investigation of the proposed Arduino as a microcontroller for the nipple, an understanding of the interfaces and complications of the DAC, and a dive into the connections between components. The front end desktop application presents the most open-ended element of this project, as it will be the only part of the project that our team is exclusively responsible for.

Stemming from these known technologies, we can begin exploring foreseen challenges, as well as tactics for overcoming unforeseen challenges. After discussing challenges, we analyze the individual components in more depth. Lastly, we review how each of these components integrate to create our anticipated solution.

Technological Challenges

We have identified some expected challenges and some stretch goals we would like to accomplish by the end of our project that we have outlined below.

Expected Challenges:

- **Front End Development:** We need a simplistic, easy to use user interface that allows the user to control the robotic/pneumatic nipple prototype.
- **Microprocessor and Back End Development:** We need a backend that communicates the changes from the front end to the robotic/pneumatic nipple.
- **Data Acquisition Center/Software Communication:** We need an Arduino that communicates parameter changes with the data acquisition center (DAC).
- **Prototype Simulation:** We need a prototype simulator that will allow us to test all other parts of our software to confirm they work independent of the physical prototype

Stretch Goals:

- We want to create voltage signal encoding to display one graph for the set of parameters that shows different signals based on which parameter changed and when.
- We want to be able to render the graphs in real time for accuracy on the front end.
- Add labels/comments to the graph when a parameter is changed, displayed on the front end side.
- Export CSV files from our Front End.
- Developing a written manual for troubleshooting front end/back end.

Technology Analysis

Challenge: Front End Development

Introduction

Front end user interfaces are a critical component of developing control software because they allow the user to directly control the prototype without physically manipulating the technology themselves. In the context of our project, researchers, such as Dr. Mayerl, need the software to control three parameters of the robotic/pneumatic nipple; hole size, nipple stiffness and milk viscosity. Dr. Mayerl would like to change these parameters by utilizing an application on his PC to manipulate the prototype and coordinate an output to the DAC (DAC) which displays that the change has been triggered.

Without a user interface, Dr. Mayerl and his research team would have to physically manipulate the prototype parameters and trigger the change separately to the DAC. When working with live subjects and expensive technologies, researchers need simplicity. A PC application will allow the primary researcher who is observing changes from the DAC on the PC to concurrently control the robotic/pneumatic nipple instead of needing a separate lab associate to be in charge of controlling the prototype.

Desired Characteristics

- **Access:** The prototype our end product will control costs a lot of money to build. Due to the cost of the prototype our client does not have unlimited funds to pay a subscription for private front end development. Ideally, the technology we use will be free to download and use or will have an open source license we can operate under.
- **Ease of Use:** When writing our front end, we want to be able to concurrently see the updated view without having to run the program each time we implement new code or changes by using a built-in designer tool through Visual Studios. There is a feature in .Net Framework that allows the developer to see the changes you make to the front end application in real time. We would like to be able to create and develop in Visual Studios because our front end developer is familiar with that environment. Our front end application needs to be able to communicate with the backend which will be written in C++. Our ideal front end would be written in C++, which would make integration with the back end and simulator seamless.
- **Cross-Platform:** Since we are building the front end on an army of PC computers that utilize Windows,it would be ideal if our framework was able to be used on Linux and MacOS computers without needing to implement additional technology.
- **Community Resources:** The ideal technology would have a highly developed, official documentation that has real complex examples and an active community of online users. An active community of online users looks like recent, up to date, StackOverflow questions, youtube posts from third party users, blog posts, and github repositories that use it.
- **Appearance:** The main reason we are not considering WinForms as a technology is because we want to move away from the 90s user interface look. We want the application to look modern and sleek, with naturally integrated buttons and features that are evolved from the classic square 2D buttons and ‘click here’ icons. We want our application to blend seamlessly with other 21st century applications the user might be using. We would like to create our application in a ‘dark mode’ look. It is not necessary to use a ‘day’ and ‘dark’ mode but that might be a nice feature to include.

Alternatives

Based on our understanding of Arduino technology, we searched for front end development frameworks that could be written in C++. Qt with C++ came up as a C++ GUI framework that is relatively simple to use. Qt was released in 1995 by Haavard Nord and Eirik Chambe-Eng, two Norwegian Institute of Technology Graduates [4] The ‘Q’ in the name ‘Qt’ was chosen arbitrarily because of its stylized look in Emacs Font and the ‘t’ stands for toolkit. Qt is in its sixth iteration today and is used by companies like the European Space Agency [4] Microsoft, Wireshark, Origin and Skype are also rumored to have been built using Qt with C++. Qt is currently owned and maintained by ‘The Qt Company’.

Another front end framework we found whilst searching for C++ GUI frameworks was OpenGL with C++. OpenGL was created in the 1990s by SiliconGraphics also known as (SLI). OpenGL was a primary creator in the 3D technology space early on. In 2006, Khronos Group declared they would reduce the importance of maintaining OpenGL. The last OpenGL update came in 2017. OpenGL is reportedly used in Adobe Photoshop, Adobe Premiere Pro and Minecraft GUIs.

Our front end coder has experience with developing front end applications with Windows Presentation Foundation or WPF with C# and so we wanted to explore whether or not using WPF C# in tandem with C++/CLI would be possible. WPF is similar to WinForms, almost like a 'next generation' framework. It was released in 2006 by Microsoft Windows and utilizes .Net Framework and became open source in 2018.

Analysis

Qt

- **Access 5/5**: At first we were really concerned with researching Qt because it has a steep cost for private Qt Application Development. In figure 1.1 you can see that the cheapest yearly subscription for one developer using Qt is $3,790. However, after some additional research we found that there are a couple options to work with Qt through open source licenses like the LGPL and GPL license, figure 1.2. Being able to use Qt for free is a huge benefit to our project because our client is not comfortable coding, he does not want to pay a steep subscription fee in order to maintain and update the code, which may take a lot of time due to his limited coding experience.
- **Ease of Use 4/5:** Qt's most recent release is version 6.5. It has a built-in designer tool that is compatible with Visual Studios which gives high points to ease of use. Qt is a C++ GUI development framework which means there is no need to create interfaces that communicate between the development framework and our C++ backend code. Qt is written and developed in C++ which means it will be compatible with our backend written in C++. It may require more memory management on the development side which will be a learning curve for the front end developer.
- **Cross-Platform 5/5:** Qt is compatible with Windows, Linux, IOS, Android, MacOS and even microprocessors. Qt even provides a guide for scalability through their website.
- **Community-Resources 5/5:** Qt has its own documentation pages available on its website to help developers better understand the available methods and functions. There are many youtube videos for creating and setting up the Visual Studios environment for Qt. I was also able to find documentation from third party Universities and Companies like Stanford and Microsoft.
- **Appearance 5/5:** Qt implements 3D graphics, animations, and premade components to simplify modern project development and make an arsenal of methods and implementations available to the developer.

OpenGL

- **Access 5/5:** OpenGL is a free downloadable framework for development. It operates under an open source license.
- **Ease of Use 3/5:** OpenGL is written in C++ and can implement libraries GLEW and GLFW in order to create and build front end GUIs. However, OpenGL can be difficult to learn and create with and has not been updated since 2017. OpenGL is still being maintained by its parent company, Khronos Group, but it is no longer being actively developed which may prove problematic down the road.
- **Cross-Platform n4/5:** OpenGL has cross-platform compatibility but because it is not currently being developed it may cease to keep up with newer technologies as they roll out.
- **Community-Resources 3/5:** Both Kronos and OpenGL have some documentation for learning OpenGl. Because of its cross-platform ability Android also has developer examples and documentation.
- **Appearance 4/5:** OpenGL has good graphics and developer tools for being an older technology but because of its age it is not as nice as Qt.

WPF

- **Access 5/5:** WPF is a free framework and is incorporated into Visual Studios.
- **Ease of Use 3/5:** It incorporates a built-in designer tool which allows for easy code manipulation with immediate visual results. One downside is that there is virtually no debugging process for binding errors between WPF, XAML and the C# code on the backend. But there are design models such as Model-View-ViewModel which are widely used and make learning and understanding the code much easier because of the organization. WPF and C++/CLI will use XAML and some elements in C# on the front end configuration but will need to use P/Invoke and CLI to make sure the C# code from the front end is correctly defined in order to work with the C++ backend. For simple programs and functions P/Invoke can work to define functions but the more complex the program the less efficient using P/Invoke can be and then the coder needs to look into C++/CLI which will allow the coder to create a management system for C++ code. WPF uses C# which means there is less memory management on the front end side because C# does its own memory management, whereas C++ requires memory management to be declared and handled by the developer.
- **Cross-Platform:** WPF is currently only available for Windows users and is not compatible with other platforms like Linux and MacOS.
- **Community-Resources:** WPF has some documentation through Microsoft but does not have a comprehensive list of examples for each binding and circumstance type. There are some youtube videos and blog posts but many of them are at least a decade old.
- **Appearance:** For a basic application WPF has good visuals but it is primarily a 2D visual development tool.

Chosen Approach

Our team has decided to move forward with Qt because it had the highest average rating with 4.8/5. We believe Qt is the best choice for moving forward because of its community resources, modern appearance and ease of use. We are incredibly excited to begin learning to work with Qt to develop our GUI for the control software of the prototype. Below you will find Table 3.1 which will show the comparison values used to evaluate the technology.

	OpenGL	**Qt**	**WPF**
Access	5/5	5/5	5/5
Ease of Use	3/5	4/5	3/5
Cross-platform	4/5	5/5	1/5
Community Resources	3/5	5/5	3/5
Appearance	4/5	5/5	4/5
Average Score	4/5	4.8/5	3.2/5

Table 3.1: Comparison of three types of frontend frameworks based on ideal criteria

Feasibility Proof

To make sure our front end application works we will utilize the simulator, our fourth challenge, to make sure that the front end controls the right components on the backend. If we are lucky enough to receive a physical prototype that has the capability for all 3 parameter changes next year we will use visual checks to make sure the front end accurately controls the prototype correctly. We will create a set of unit tests as well to make sure our code is as bug free as possible before handing it off to the client.

Challenge: Prototype Microcontroller and Back End Integration

Introduction

Our project needs to have a connection with a microcontroller to work. The microcontroller is what controls the robotic/pneumatic nipple directly by providing power to specific electronic components. While our project is about creating control code for Dr. Mayerl's robotic/pneumatic nipple, we do have a say in what microprocessor the robot is controlled by. Which microprocessor is chosen affects what language we use for the backend, thus increasing the likelihood of overall success for us.

Desired Characteristics

The microcontroller that we end up choosing should have properties that increase the likelihood of success. The microcontrollers that we are choosing between should be: easy to use, compatible with existing hardware, precise, and should let us have a high degree of control from the microcontroller.

- **Ease of use:** Each microcontroller has a programming language that works best with it. Some of these languages are easy to understand and use, while others are arcane and clunky. A high score means that it has a language that is easy to understand, while a low score or 0 would have a language that is near incomprehensible.
- **Compatibility with Existing Hardware:** The microcontroller should be able to be integrated seamlessly and control the existing robotic/pneumatic nipple prototype. A good score means the microcontroller should work perfectly with existing hardware, while a low score would mean the technology is unable to control the robot at all.
- **Precision:** The microcontroller should be able to accurately control the robot, to do the exact same thing every time. A good score means the microcontroller is able to replicate the same state every time, while a low score means the technology chosen would affect the function of the microcontroller and there is no accuracy.
- **Level of Control:** The connection should allow the microcontroller to control exactly how much power is going into each component of the robotic/pneumatic nipple. This is important because we will be needing the exact amount of voltage for some calculations. A good score means the technology should be able to control voltage to each component accurately, while a low score means the microcontroller would have no ability to control the voltage.

Alternatives

In order to explore what the optimal microcontroller is, we are going to test three in order to get an idea. The microcontrollers we are testing will be a PIC, Arduino, and Raspberry Pi. We

chose these three because we either already had them for testing purposes or they were readily available. They are also very commonly used products for this kind of application.

PIC microcontrollers have been around since the mid 70's, and are the gold standard for microcontrollers. The specific version we would be using would be a PIC32MK, which is specialized for use on motors and multi-channel CAN applications. PIC products all use MIPS as their only programming language, which is not great to use, especially when you are not in practice with it.

Arduino is a type of microcontroller that specialties in robotics, but is powerful enough to be a mini computer on its own. Arduino is open source software and hardware, which also would help because it gives us the option to keep the program at a high level, or go extremely low level and do machine programming. Arduino can be programmed with any programming language with compilers that produce binary machine code.

The final technology we are going to explore is Raspberry Pi, which is less of a microcontroller and more of a microcomputer. Originally designed for use in robotics the Raspberry Pi has become more of a complete computer that even has a small GPU on it.

Analysis

PIC

- **Ease of Use 1/5:** The PIC microcontroller uses MIPS, an assembly language. This is fine for its usual use, which is a component of a larger board, but for our purposes it would not work. Assembly languages are not ideal because they come with a learning curve and there are better options that exist and currently use high level languages.
- **Compatibility with Hardware 3/5:** PIC does not typically interface with the motors that are used on the robotic/pneumatic nipple, even the version that is designed to interface with motors.
- **Precision 5/5:** PIC has shown that it will not lose accuracy after long periods of use.
- **Level of Control 5/5:** PIC is great at sending specific amounts of data, or electricity, to a target which gives it a high level of control.

Arduino

- **Ease of Use 5/5:** Arduino is an interesting choice because it is both open hardware and software. This unique distinction makes it a good choice because we will have access to whatever part of the hardware that we need. Arduino can use any language that compiles to machine code (C, C++, C#) .
- **Compatibility with Hardware 4/5:** The current prototype of the robotic/pneumatic nipple uses Arduino to control it, so the compatibility with hardware should be a 5/5, but given the current state of the robotic/pneumatic nipple, it is only 4/5.
- **Precision 4/5:** The ME team is using Arduino compatible motors to control the function of the robotic/pneumatic prototype so using an Arduino microcontroller allows us to take

advantage of the motors already in use and implement the precision processes already in place.

- **Level of Control 4/5:** While an Arduino does not have as much low level control as PID or Raspberry Pi, it does have pin level voltage control..

Raspberry Pi

- **Ease of Use 5/5:** You can program a Raspberry Pi using any language that compiles to binary which means it would be relatively easy to work with.
- **Compatibility with Hardware 3/5:** There is one single component of the existing robotic nipple that has a hard time interfacing with a Raspberry Pi, so it scored a 3/5 on Compatibility with hardware.
- **Precision 4/5:** Raspberry pi can be precise with motors that are designed to be used by a Raspberry pi. However, the ME team uses motors compatible with Arduino so they would not be entirely compatible with the prototype.
- **Level of Control 4/5:** While it is possible to control a raspberry pi to the pin level, there are blocks in place to stop less experienced programmers from messing with them. It is possible to remove those blocks, but it is not recommended and not needed for our project.

Chosen Approach

We chose the Arduino microprocessor because it has the highest combined score of 4.25/5, see Table 3.2 below. This is good news for us because the Mechanical Engineering team we are working with has already developed the prototype.

	PIC	**Arduino**	**Raspberry Pi**
Ease of Use	1/5	5/5	5/5
Compatibility with Hardware	3/5	4/5	3/5
Precision	5/5	4/5	4/5
Level of Control	5/5	4/5	4/5
Average Score	3.5/5	4.25/5	4/5

Table 3.2: Comparison of three types of microcontrollers based on ideal criteria

Feasibility Proof

To make sure that the microcontroller is working correctly, we will be testing the robot independently and making sure that it hits the metrics that our client wants. After the robot is calibrated, we will be able to test and use it. The simulator will also help the developer code for the Arduino because it will make sure that we are giving it consistent inputs and receiving consistent outputs.

Challenge: Data Acquisition Center/Software Communication

Introduction

At the end of this capstone, Dr. Mayerl wants his current DAC which is the PL3516 PowerLab 16/35 by AD Instruments, to gather information from either the final prototype provided by the ME team, or a simulation prototype in case the Mechanical Engineer team's prototype is unfinished or unusable. There are many ways to communicate with the DAC, some better than others, whether that is through hardwiring or wireless. All options have their pros and cons when it comes to performance and maintainability. The features we would like to care for are the best available practices, how safe they are to implement and maintain, how compatible the approach is with different softwares, how much expandability can one provide for the ports available in an acquisition center, and finally, with how much accuracy can the changes to a specific feature in the robotic nipple be displayed in a way that Dr. Mayerl can visualize and utilize the data for his research.

Transferring data change outputs to the DAC is a necessary feature to implement in our product. Dr. Mayerl has stated that although the data he can produce with the DAC is accurate and always up to date, he needs a way to determine when the parameters for hole size, hole pressure, or milk viscosity change, using the DAC.

Being able to visualize positive changes in the subjects' ability to feed is vital to identifying the root cause of the subjects' feeding problems thus allowing researchers to study how to improve feeding problems through the use of the robotic/pneumatic nipple prototype. The team hopes to start off with piglets to then potentially move on to human babies, which is the best possible outcome for this prototype.

Desired Characteristics

- **Best Practices:** The definition for good practices is to be able to deliver a product using the most effective and efficient methods. Our team is committed to implement the best practices available which differ on any approach we decide is best to implement. This is an important feature to take into consideration because not implementing good practices at the outset of the project will make modifications in the long run more costly and challenging.
- **Safety:** Safety is an important feature to keep strongly in mind since we will be dealing with physical components that can potentially pose a safety hazard for our client, the DAC, and/or human babies/piglets. Our team is committed to test and continuously monitor the development of our physical device and its components to ensure that the final product is comprised of high standards.
- **Software Compatibility:** With today's technology, it is key to make sure that a product or service is available in as many pieces of software as possible. The main benefit of this is to increase accommodation for future clients and whatever software they depend on. In this case, our team wants to make sure that our prototype can be connected to the Mechanical Engineer team's prototype, and if possible, to a prototype that could be in the works in the case where the Mechanical Engineer team's prototype is not fully functional.
- **Expandability:** Being able to build upon an existing product is vital to allow it to be competitive and scalable. Because there are many ways to go about communicating with the DAC, expandability can differ. Our team wants to make sure that the approach we decide to take allows for the most efficient and effective expansion for our prototype, whether that is with adding new features for the robotic/pneumatic nipple, or having different ways of recording and storing data.
- **Accuracy:** It is indispensable for data utilized in research to be precise as this allows results to be trustworthy, reproducible, and less prone to errors. Our team is aiming to collect, update, and display data to the client with the utmost precision possible and in ways that are best-suited for the type of data managed. We recognize how important Dr. Mayerl's research is so we want to make sure that the data cannot be interfered with when being collected or updated to ensure that only unaltered data is displayed.

Alternatives

After conducting some research and interviewing Dr. Mayerl, we were able to identify the exact model of the DAC, which in return allowed us to be able to find an owner's guide to aid us in the production of our project. At a later date, the team decided to discuss and investigate what the most favorable methods for communication with the DAC would be. As the meeting progressed, our team narrowed down our approaches to three viable choices to choose from when it comes to communicating with Dr. Mayerl's DAC.

The first option would be to physically connect a computer into the DAC using one of the available ports on the Powerlab's back panel. Having a physical connection between the DAC and a computer allows for low latency and a secure data transfer. This approach is what the client is currently using to display data acquired from the DAC. A secondary approach would be to splice wires from the prototype motor provided by the ME team which can allow for wire length extensions and compatibility with other kinds of wires. As stated before this approach was presented to us by the ME team. Finally, a third approach would be to create a wired connection between an Arduino and the DAC.

An Arduino is a small microcontroller board, also called a platform, that allows for electrical components to be integrated, as well as allow code to control the components or how the board works. One of our team members (Alexander 'Gus' Siegel) proposed we take this approach as it is very simple to modify and customize an Arduino to establish a connection between it and the DAC.

Analysis

Spliced Wires From Prototype Motor

- **Best Practices 2/5:** As an industry standard typically, we do not see products built using wire splicing because of its power inefficiency.
- **Safety 1/5:** Splicing wires from a DC motor is the approach that the ME team took when building the prototype. However, there are some significant safety concerns for those in contact with the prototype and the motor itself. Splicing wires can be done safely in the correct environment and requires significant knowledge.
- **Software Compatibility 1/5:** Although wires can be used for software connection, it is not a reliable option for software compatibility
- **Expandability 2/5:** Splicing a wire into multiple wires is possible up to point. While it is possible to easily expand wire length through splicing, ensuring that the extensions are properly insulated is not an easy task, which reduces safety.
- **Accuracy 1/5:** Splicing wires can result in data loss and reduced voltage transmission which leads t o inconsistent results and more difficult troubleshooting.

Wired Connection From Computer

- **Best Practices 5/5:** Using a wired connection between a computer and a DAC allows us to establish a connection. There are many libraries and existing software that can help with the scalability of our final product.
- **Safety 4/5:** The only concern with safety is that data can be interrupted by interrupting the wired connection between the computer and the DAC which in return can corrupt data or damage the DAC.
- **Software Compatibility 5/5:** A computer can install software to make a program compatible with anything.
- **Expandability 4/5:** One small issue with expandability is that some needed features

could require our team to create a program or library from scratch and there are limited ports on the DAC. If the connection to the DAC is wired from the computer that takes an additional port on the DAC because the prototype will be wired to the DAC separately.

- **Accuracy 3/5:** Receiving accurate results with a computer connected to a DAC is simple and allows for the data to be stored and used for different purposes. However, making sure that the timing between when parameter changes take effect on the prototype and the DAC receives information that the parameter has changed will be difficult because the backend will need to confirm the change before sending the trigger to the DAC and this can affect research data.

Wired Connection From Arduino

- **Best Practices 5/5:** Good practices can be followed when using an Arduino board because code and certain electrical components can be combined to provide an efficient and real time solution to many problems, in this case having our data collected, updated, and displayed.
- **Safety 3/5:** Arduino can be considered safe to use but as more external components are added, safety can slightly decrease. Another small problem is that the physical Arduino can become damaged through use and manipulation which can decrease the quality of input or output.
- **Software Compatibility 5/5:** Arduinos can be compatible with any kind of software or computer since all that is needed for an Arduino to establish a connection is a simple USB port which is very common for all computers to have.
- **Expandability 5/5:** Because many types of electrical components can be added to a board, this allows for our team to expand upon one of the many paths necessary to tackle this imperative feature.
- **Accuracy 5/5:** Arduinos can be programmed to receive, mutate, and send data in real time which can provide the highest degree of data accuracy if done correctly. As the Arduino is connected to the DAC, no time will be wasted in transmitting information to a secondary source before going to the DAC.

Chosen Approach

In order to decide which approach to take when addressing the challenge of communication with the DAC, our team had to carefully analyze and consider what the consequences would be for every individual approach. As stated before, the wire splicing method is just not efficient enough; it would take many more precautions to ensure any safety at all as well as ensuring that the data received and displayed is accurate. The only main benefits for splicing wires is that the range of the wires can be increased very easily and that the wires can be easily customized to the needs of the project.

On the flip side, having a connection between a computer and the DAC has its own share of pros and cons. Starting with the pros, having a wired connection allows for fast and secure

data transfers. The only noticeable drawback about using a computer is that it is not always efficient to carry around a computer to connect with the DAC.

Finally, establishing a connection using an Arduino is the most optimal approach to take for our project with an average score of 4.6/5. Creating a connection between an Arduino and the DAC would allow for a cost-effective solution to the presented challenge, as well as permitting real time data acquisition which is a key characteristic for our project. It is worth mentioning that this approach also provides a portable device. The only drawback about using an Arduino is the extra work included such as soldering and coding in a language the Arduino can use and understand. Table 3.3 below shows the 3 technologies compared against each other.

	Wired Connection from Computer to DAC	**Spliced Wires direct from Prototype motor to DAC**	**Wired Connection from Arduino to DAC**
Best Practices	5/5	2/5	5/5
Safety	4/5	1/5	3/5
Software Compatibility	5/5	1/5	5/5
Expandability	4/5	2/5	5/5
Accuracy	3/5	1/5	5/5
Average Score	4.4/5	1.4/5	4.6/5

Table 3.3: Comparison of three types of connections based on ideal criteria

Feasibility Proof

To make sure that our Arduino prototype runs smoothly, our team will create a simulation of what the collected data will look like when displayed, including when changes are made. The simulation will consist of the basic requirements for the project itself, which are to display a graph or visual cue when a change in parameter occurs; hole size, hole stiffness, milk viscosity. If the simulation succeeds, the team will proceed to conduct testing with the actual DAC to verify that the Arduino is establishing a connection.

Challenge: Prototype Simulation

Introduction

Since the project interfaces with hardware components whose completion schedule may not be aligned with our own we deemed it necessary to develop a software component that simulates the inputs and outputs of an expected nipple prototype. The simulator would be a separate, yet closely related software component that can be toggled to interface with the control software. While an ideal project will have no need for this component, we foresee this challenge as being a necessary one, to guarantee we can prove our project meets requirements, with or without a working hardware component.

Desired Characteristics

- **Creation Feasibility:** The creation of the simulator must be within the realm of possibility for us to completely build, test, and implement throughout the course of the project. It should not be so easy as to be trivial, but it also cannot become complex enough that it overshadows the challenge of the primary project. A higher score in this category communicates how balanced the proposed solution is.
- **Testability:** We must be able to easily test the inputs and outputs of the simulator, to ensure it matches the expected behavior of the physical prototype. This will need to be something we can quickly check and measure, as testing the simulator may be a significant element of the development process. A higher score in this criteria corresponds to a solution that can be repeatedly, quickly, and easily tested, run or reset**.**
- **Scope:** We want the simulator to simulate as closely as possible an actual prototype. This effectively means we want to get as close as reasonably possible to having our own robotic nipple, without actually having one. A higher score in this criteria shows how closely the given solution approximates an actual robotic nipple. This criteria is weighted more heavily than others, since a simulator that simulates poorly will not be very helpful for proving that our software meets its requirements.
- **Accuracy:** Potentially most importantly, the simulator must be accurate, and reflect the actual voltage values and other input/outputs related to the prototype. This overlaps slightly with scope as well as testability, but is important enough to warrant its own characteristic. A higher score in this section indicates how accurate the simulator's output values are, and how precisely it acts when given certain inputs.

Alternatives

We envision three potential types of simulator to solve this challenge. The first type of simulator we could develop would be a purely frontend design. The second would be a simulator that runs on the frontend and the backend, including the microcontroller if applicable. The third

potential simulator would involve building an additional component with hardware connections to the project, that simulates the connections we expect to have with the nipple prototype.

The first type of simulator would exist as an additional extension of the frontend. To apply it, there would be some additional argument applied at the startup of the program. While active, it would act as the project's backend, effectively faking all the data received and output. Instead of signals being sent between the computer and the microcontroller, the signals output from the frontend would be received by the simulator instead, and any output that the frontend expects would be similarly sent back from the simulator to the frontend.

Our second potential approach would function on the client computer as well as the microcontroller, if applicable. Enabling it would be a similar situation as the frontend simulator, with a flag set at program startup. While active, it would receive signals sent from the microcontroller, rather than signals being sent to the prototype. To complete its simulation, it would also send back the expected signals to the microcontroller software, which in turn would be sent back to the frontend. Since this type of simulator may not have a working microcontroller to work on, it would also have a configuration to execute only on the project backend. Though this would be additional work to create this simulator, it is a necessary step to ensure our ability to guarantee functionality of our control software.

The third potential simulator approach would involve building an additional hardware component that acts as a robotic nipple prototype, albeit without any actuators or moving pieces. Instead of being software that overrides or intercepts data within the control software itself, this simulator would have the same expected physical connections as we anticipate the nipple prototypes to have. It would, for all intents and purposes, be exactly what the control software expects to see, save for actually having any moving parts. Rather, it would emulate the hardware of the prototype, sending and receiving signals through the same means that a physical prototype would use.

Analysis

Frontend Simulator

- **Creation Feasibility 5/5:** The frontend simulator is easily within the realm of possibility to create it as an additional piece of the project, as its execution solely on the frontend effectively removes complications in communicating with our backend or the DAC.
- **Testability 5/5:** From a testing perspective, the frontend simulator would require very little setup. Testing this kind of simulator would add a trivial amount of time and difficulty as easy as setting up the rest of the software.
- **Scope 1/5:** The scope of this simulator will simulate the outputs we want to deal with, its non-communication with integral parts of the project mean this type of simulator is a very poor approximation of an actual prototype.

- **Accuracy 2/5:** This simulator can get exactly the outputs we expect, but they will not be grounded in any way, and will not have any impact on the project outside of displaying the results to frontend.

Backend + Frontend Simulator

- **Creation Feasibility 3/5:** For creation feasibility, this simulator involves a more complex approach, but still sits within the realm of possibility, since it is purely a software solution. While it will certainly be more difficult to implement than the frontend-only simulator, it is certainly still a solution that is within the realm of possibility for a CS team.
- **Testability 3/5:** In addition to this simulator being more challenging to create, this approach will be more difficult to test. Having the simulator function with or without a microcontroller adds a bit of challenge in testing, but overall it will not be incredibly difficult.
- **Scope 4/5:** This approach is as close as we can get with pure software. Without adding any additional hardware, this simulator will execute and act as a robotic nipple up to the point of actually making hardware movements.
- **Accuracy 4/5:** Accuracy will involve more components than frontend, as a software only solution, we can not perfectly mimic what an actual prototype's input and output values will be. It will, however, be using inputs and outputs in the expected points of the project, and actually sending data to and from a microcontroller if applicable.

Hardware Simulator

- **Creation Feasibility:** Building this simulator simply is not realistic.. This approach would involve researching, purchasing, coding, and integrating a nipple prototype with all the expected hardware connections, just without any actual servos or actuators. As a Computer Science team, it is outside the realm of our abilities and project to build something of this caliber.
- **Testability:** As with feasibility, testing a simulator with physical hardware would be exceptionally more complex than the two software-only solutions.It would often require being in the actual Mayerl lab to test this simulator, potentially with Dr. Mayerl present, since we would need to actively have access to his computer as well as the DAC.
- **Scope:** For the purposes of a simulator, it perfectly captures what we want to simulate. Without needing to implement the complexities of servo or actuator movement, we can implement exactly what we want to expect, sending it across the expected communication channels, involving all of the elements of the project.
- **Accuracy:** An approach such as this one would include all of the software that would be necessary for the prior two solutions, as well as incorporating actual hardware, it is as accurate as we can get.

Chosen Approach

Based on these factors, we plan to move forward with a simulator that functions on the frontend and backend, and interfaces with any present microcontrollers. This design has the highest average score of 3.5/5. Since it sits comfortably within the range of acceptable creation feasibility, we know we will be able to overcome this challenge. Additionally, its expected testability score of 3 will allow us to ensure we can test the software whenever we need to. Its scope and accuracy are not as perfect as they could be, but given the exceptionally low creation feasibility of the hardware simulator, this option remains as the next best option. Table 3.4 below, shows the comparison of all three technologies.

	Front End Simulator	**Front End + Back End Simulator**	**Hardware Simulator**
Creation Feasibility	5/5	3/5	0/5
Testability	5/5	3/5	2/5
Scope	1/5	4/5	5/5
Accuracy	2/5	4/5	5/5
Average Score	3.25/5	3.5/5	3/5

Table 3.4: Comparison of three types of simulators based on ideal criteria

Feasibility Proof

To ensure our simulator will work, we will develop it incrementally and with versatility in mind. Its creation will be closely tied to, and influenced by the requirements we designate at the end of this semester. To ensure the simulator will be able to mimic as many different robotic nipple prototypes as possible, we will construct it to work with as many or as few components as we expect to have. This could mean the simulator is truly only a frontend simulator, or it could mean that it exists up to and including the microcontroller outputs. This will be an integral part of the project, since our requirements proof hinges on it, and other solution components will similarly depend on it.

Technology Integration

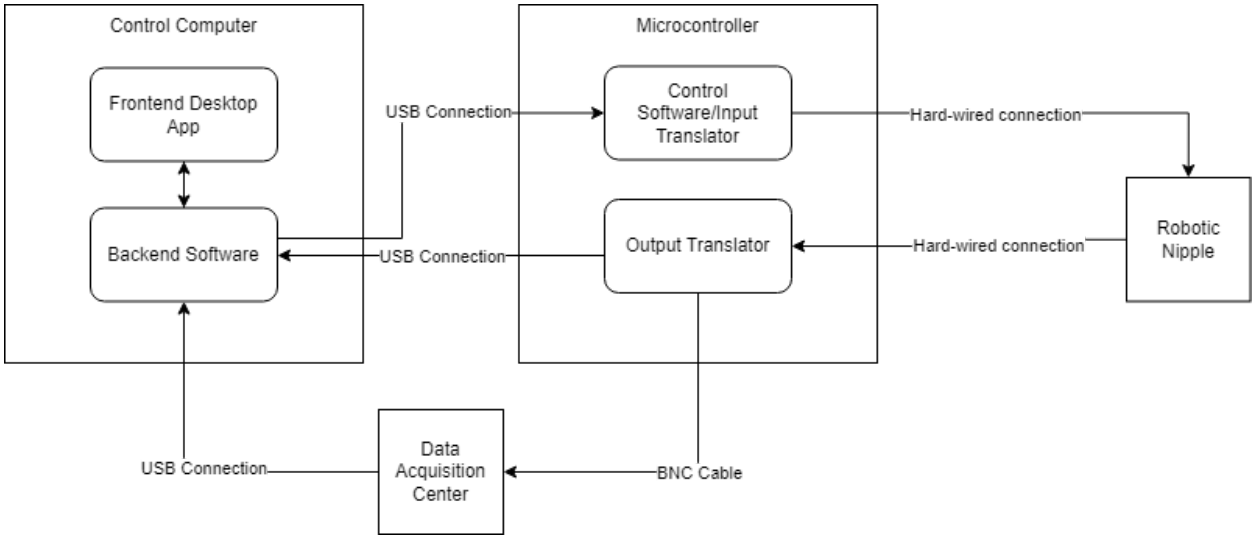

Diagram 4.1: This diagram demonstrates the critical components and connections of our overall project design.

Diagram 4.1 illustrates hardware, software, and connections we expect the project to include. The four major components, represented by squares, include the control computer, microcontroller, robotic nipple, and the DAC. The control computer will be any Windows computer that hosts our frontend and backend software. The microcontroller is the Arduino we expect to use, and will host the software responsible for the interface between the robotic nipple and the DAC. The robotic nipple itself describes the component that is actually being controlled. This may take the form of the ME team's delivered solution, our simulator, or another solution. The DAC groups the data, and we must interface with it for information syncing purposes.

The rounded rectangles represent software our team will develop. This includes the frontend desktop app, backend software, control software, and output software. The frontend represents what will allow for control of the nipple; it will be the human interface element. The backend bridges the gap between the frontend and the microcontroller. The control software receives input from the backend, and passes the relevant information onto the robotic nipple. Lastly, the output software receives data from the robotic nipple, and uses that information to provide the relevant inputs back into the DAC.

Also represented in this diagram are the electrical connections between discrete components, but they are listed more for completeness than anything. The control computer and the microcontroller communicate via a USB cable. This will allow data to be sent to and from the software we host on the microcontroller.

The connections between the microcontroller and the robotic nipple are marginally more discrete, by merit of being hard wired. Lastly, the connection between the microcontroller and the DAC will be achieved through a BNC cable, and it will purely be output of voltages from the

microcontroller. While a USB connection exists between the DAC and the control computer, our software will not interact with it unless we pursue stretch goals that require us to.

Conclusion

Our team wants to help Dr. Mayerl with his research regarding feeding problems in infants by using a nipple prototype whose features (hole size, stiffness, fluid viscosity) can be changed. This brings upfront a few challenges that we have to address throughout the development of our prototype.

The first challenge presented was the front end development and how it would be incorporated into the final product. After some research, QT was chosen as our front end framework. The main reason for this is because there are many community resources available for us to use as well as the added benefit of the framework being convenient to use.

On the other hand, the second challenge presented was the selection of a piece of hardware that the team will be using to develop on. Because the ME team is already developing their prototype using Arduino, our team decided to also choose Arduino as our main development platform since it would seem appropriate to be able to expand upon what the ME has already accomplished.

Conversely, the third challenge presented was establishing a connection between the DAC and the hardware chosen from the second challenge. Our team decided to go with the Arduino connection method since it would be the most expandable and cost-effective option to follow through with due to the fragile equipment we will be dealing with.

Finally, the fourth challenge presented was creating a simulation of what the data should be like when modified and displayed. Ultimately, our team determined that creating a simulator that functions on the frontend and backend would be the best option out of the three since it would be the most feasible approach to create a fully functional simulation.

Our team plans to go through the development of the approaches stated above before interacting with Dr. Mayerl's DAC. We want to ensure that everything works as intended before trying to use any equipment that is important to Dr. Mayerl's research and projects.

References

[1] M. Delnord, B. Blondel and Z. J, “What Contributes To Disparities In The Preterm Birth Rate In European Countries?,” 27 April 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4352070/#:~:text=In%20countries%20with%20comparable%20levels,among%20live%20births%20in%20Europe.

[2] B. E. Hamilton, J. A. Martin and M. J. J. K. Osterman, “Births: Provisional Data for 2022,” June 2023. [Online]. Available: https://www.cdc.gov/nchs/data/vsrr/vsrr028.pdf.

[3] Giulia Vizzari et al. 2023. Feeding difficulties in late preterm infants and their impact on maternal mental health and the mother–infant relationship: A literature review. *Nutrients* 15, 9 (May 2023), 2180. DOI:http://dx.doi.org/10.3390/nu15092180