NTNU
Norwegian University of
Science and Technology

Prototyping Material Programming Using Internet of Things

Sondre Johan Widmark

Master of Science in Informatics
Submission date: June 2018
Supervisor: Dag Svanæs, IDI

Norwegian University of Science and Technology
Department of Computer Science

Preface

I wish to especially thank my supervisor Dag Svanæs at the Department of Computer Science (IDI), Norwegian University of Science and Technology (NTNU). I am very grateful for his guidance and engaging enthusiasm for the subject.

I wish to thank Terje Røsand for technical assistance and interesting discussions about technology.

I wish to thank my father for proofreading this thesis.

Sondre Johan Widmark

Trondheim, June 2018

Abstract

This thesis explore material programming as a proposed design practice for computational composites. A computational composite is an envisioned future material as dynamic, interactive and capable of computation. Computational composites as a concept understand the computer as a material in and of itself and predict current technological growth will lead to such materials in the not too distance future. Material programming is the design practice used for exploring the interactive and dynamic properties of such materials. This thesis will create a physical proof of concept for material programming using today's internet of things.

Table of Contents

List of Tables

List of Figures

Abbreviations

API	=	Application Programming Interface
CGI	=	Computer-generated imagery
DHCP	=	Dynamic Host Configuration Protocol
GPIO	=	General-purpose input/output
GUI	=	Graphical user interface
ID	=	Identification
IDE	=	Integrated development environment
IP	=	Internet Protocol
IR	=	Infrared
IoT	=	Internet of Things
LAN	=	Local Area Network
LED	=	Light-emitting diode
MQTT	=	Message Queuing Telemetry Transport
M2M	=	Machine-to-machine
NFC	=	Near-field communication
OS	=	Operating system
REST	=	Representational State Transfer
RFID	=	Radio frequency identification
RGB	=	Red, green, blue (color model for a LED)
SSH	=	Secure Shell (protocol)
TCP	=	Transmission Control Protocol
Wi-Fi	=	Wireless Fidelity

Chapter 1

Introduction

Through recent history, human kind have seen the development and invention of new materials for use in industry and daily life. Everything from steel and concrete, to plastic and semiconductors has significantly raised the quality of materials and the magnitude of products they can produce. Technological innovation in recent years within biotechnology and smart materials has started to influence design, with emerging consumer markets within smart textiles and interactive architecture [26]. With the development of Internet of Things (IoT) and microcontrollers becoming smaller, it is relevant to ask the question of what happens when technological progress reaches a point where technology can be miniaturized to a degree it is possible to embedded it into common materials.

Researchers have introduced the concept of computational composite [27] as a new type of composite material, which regard the computer as a material in itself, and envision a future where common materials incorporate interactive and computational abilities. Such a material has massive potential for use within a number of areas, most notably within various design practices; including interaction design, product design, architecture, prototyping and city planning. In short, computational composites is an envisioned future material which inhabits computational and dynamic ability on a physical level. Imagine a bottle which can physically change color according to the temperature of its contents, or have shape-changing abilities which form to your hand. This bottle is a composite material made up of massive amounts of miniature computational material parts, which can be selected individually and made to dynamically change. The bottle object in question is not the relevant part, but the abilities of the material. Today there exists tiny microcontrollers on a centimeter scale, which if combined together can simulate such a composite material. Computational composites imagine what this would look like when tiny microcontrollers are on a millimeter-scale and can be embedded into other materials, or create common materials inhabiting computational and interactive abilities. Combining large amounts of these millimeter scale computers into one cohesive and composite material, creates a new material made up of many tiny computers. If all these tiny computers exhibit dynamic

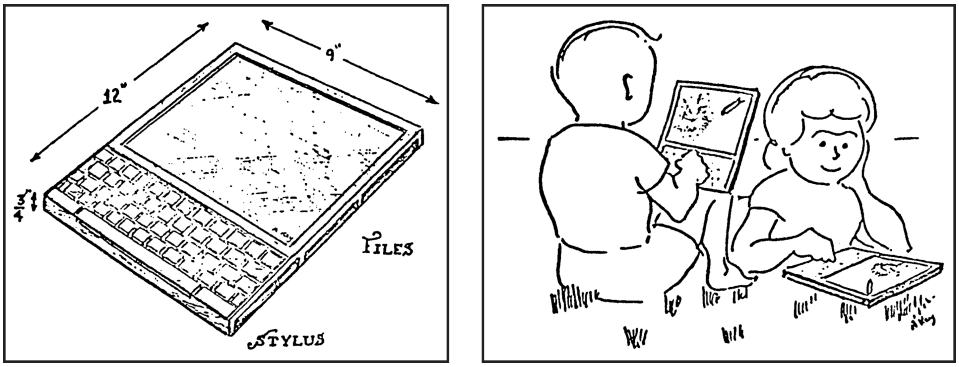


Figure 1.1: Dynabook as envisioned by Alan Kay in his 1972 paper [10]

properties, for instance color or shape changing abilities, the combined composite material has the capability to dynamically change to nearly anything a user desires.

With the introduction of this new material arise the need for a discipline which undertakes a design practice for exploring the dynamic potential of such materials. Material programming would be a way for a designer to interact with a computational composite, exploring the visual and physical changes of a material, and the interactivity of the material. This thesis will expand on the proposed notion of material programming as a future design practice of computational composites and develop a proof of concept for material programming using today's internet of things technology.

1.1 Historical perspective

While a Ph.D. student in 1968, Alan Kay envisioned a concept for a personal computer close to what is today perceived as a laptop or tablet. This concept was later described and developed in his 1972 paper "A personal computer for children of all ages" [10]. In a time where computers were huge boxes almost exclusively created for and used by businesses, Kay foresaw a personal, portable information manipulation device. The concept was titled Dynabook and his original sketches can be seen in figure 1.1. Dynabook was initially introduced in the context of children education, but Kay also recognized adults as potential users. Reading this in the 1970s gave the reader a science fiction impression, and while Kay admits this in his paper, he also argues that the current trends in technology miniaturization and price reduction almost guarantee his ideas have the potential to be realized in the near future.

The perception of the current trends at the time was closely related to a 1965 article by Gordon Moore titled "Cramming more components onto integrated circuits [15]. Moore observed that the number of transistors in integrated circuits doubled every year and projected the development would continue at least for a decade, as seen on the left in figure 1.2. This statement was later revised by Moore in 1975 [14], concluding a growth pro-

jection of doubled number of transistors every two years. Empirical evidence shows this estimation to be surprisingly accurate, as displayed by the actual growth on the right in figure 1.2. Moore also envisioned this increased technological capability leading to home computers, personal portable communications equipment and even automatic controls for automobiles [15], which proves an incredibly precise prediction given that it was made in 1965. This was however not known at the time but did provide Kay with a growth-projection basis for his own bold prediction.

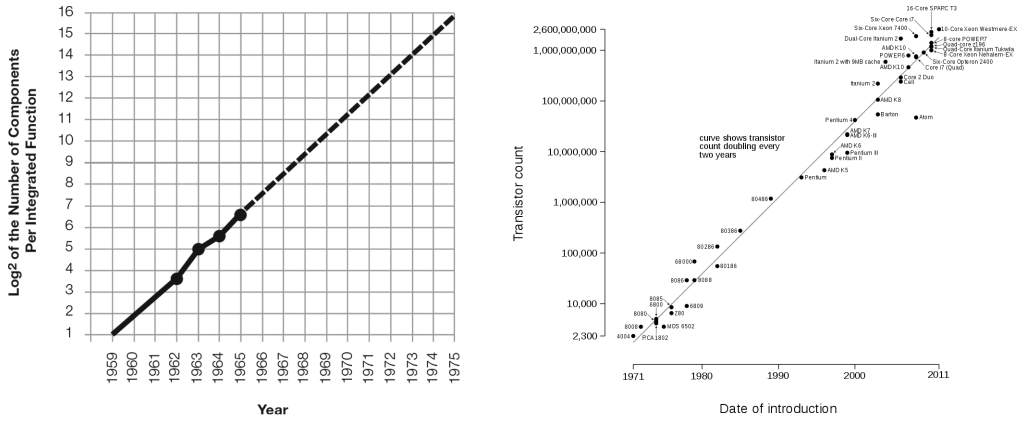


Figure 1.2: Moore's original projection in 1965 [15] on the left and an actual progress graph on the right

Following his Ph.D., Kay went on to join Xerox Palo Alto Research Center, where he continued working on his idea of the personal computer. This resulted in the development of the first version of the Smalltalk programming language in 1971, later described in detail in 1977 [9] [11]. The Smalltalk programming language ran on the Xerox Alto, the world's first computer with an inherently graphical user interface (GUI). Nicknamed 'the interim Dynabook' by Kay, this system introduced GUI with a desktop metaphor, including overlapping windows, a computer mouse, scrollbar and much of the functionality integral to computer systems today. The Xerox Alto can be seen to the left in figure 1.3, with its overlapping windows GUI on the right. The Smalltalk language was inspired by the Simula-67 language developed at the University of Oslo [12] and was one of the first object-oriented languages in the world, and an early influence to programming languages such as Java, Objective-C, Python and Scratch.

Xerox Alto's revolutionary Smalltalk GUI desktop system led to the development of Xerox Star, the first commercial system to integrate a number of technologies standard in computers today. An extraordinary aspect of Smalltalk was that a user could change the system and consequently the visual graphics of the system, all the while that same system was actively running. E.g. a user could make the window scrollbar disappear and have it visually change at the same time the modifications are made. The system and the compiler are both written in the same language, and there is no separation between compile mode and run mode - the system is always in run mode even as the program is being changed.

While developing their next computer, Steve Jobs organized a demonstration of the system for Apple engineers, which largely influenced their first GUI computer, the Apple Lisa.

Although nowhere near the miniaturized size of the envisioned Dynabook, the Xerox Alto with Smalltalk was an integral step in the progress of creating a personal computer with an interactive display. Kay's main motivation was using technology in children's education, and he continued his development of the language and ran experiments where he tried to teach children object-oriented programming [11]. Kay later went on to work at Atari, Walt Disney Imagineering, Apple, and HP. Kay also contributed to the One Laptop per Child project, aimed at providing every child in the world with a laptop.

To summarize Alan Kay's work: he envisioned the modern tablet or laptop in 1968, led the development of the Smalltalk programming language which was used in Xerox Alto, the first personal computer with GUI. The technology for developing a Dynabook is certainly here today, but when asked about whether his Dynabook concept is a reality today, he still doesn't think anyone has gotten it right. He says that 95% of the idea of the Dynabook was a service conception and only 5% was its physical form [5]. He looked upon the personal computer as an educational tool and doesn't think users today are full-fledged users of a system, but rather television watchers of different kind. The idea of the Dynabook was to simulate all existing media in an editable form and extend the notions of reading, writing, sharing and publishing of ideas. According to Kay, current electronic media has actually removed some of the day to day needs for reading and writing, and allowed the civilized world to lapse back into oral societal form. Although Kay doesn't think his Dynabook concept is realized today, it's no question that the current technological advances have enabled its creation, and today's tablets and laptops certainly correspond with the physical form of the envisioned Dynabook.

The purpose of this historical introduction is to illustrate the massive potential of technological development, as well as draw a parallel between Kay's envisioned Dynabook in 1968 and the resulting technological progress, to the proposed notion of material programming today. While the Dynabook concept seemed like mere science fiction at the time, the prediction proves strikingly accurate seen in today's light. This thesis regard material programming in the same sense, and believe ideas from the concept will prove increasingly relevant and accurate as the technological growth continues. Using Moore's law [14] and current technological trends, this inflection point is deemed to occur in a not too distant future. Subsequently, the notion of material programming today is closer to a scientific and futuristic prediction rather than a technically feasible attainment.

1.2 Introduction

The most important aspects of the Xerox Alto in relation to this thesis is the creation of an interim device to simulate envisioned future technology, using the technology available at the time. This is the main purpose of this thesis and prototype, to create an interim proof of concept for material programming using today's technology. With a future-minded

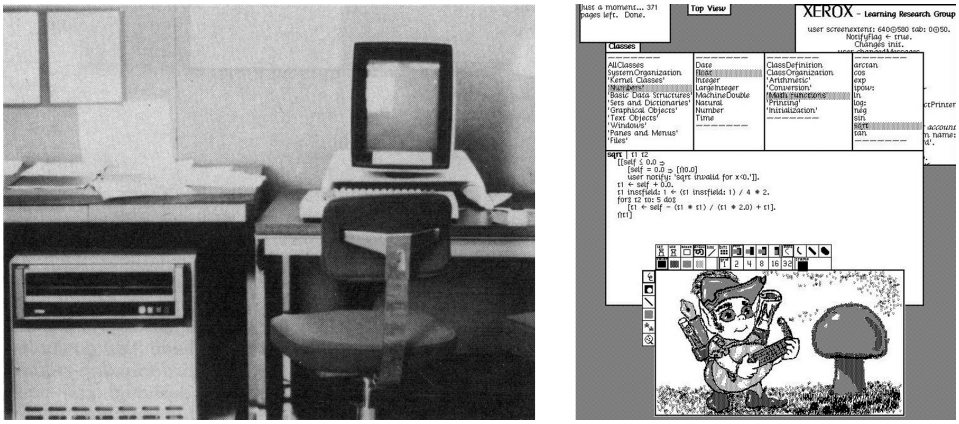


Figure 1.3: 'The interim Dynabook' Xerox Alto [9] on the left and the overlapping windows GUI on the right

concept like material programming, it is not possible to portray it accurately, because the envisioned functionality doesn't exist yet. A natural response to this would be to simulate the technology on a computer, as it might make it easier to display envisioned behavior and functionality.

This has been performed in previous cases this thesis is based on, and such the focus of this research will be to display concepts from material programming with the technology available. There are several challenges related to material programming before it can become a reality, primarily miniaturization of microcontrollers and embedding technology into materials to create computational composite materials. As such, integral to this thesis is the question of how to create a proof of concept for technology and functionality that doesn't exist yet. The prototype will have to use current technology to create an approximation to the future uses and focus on showcasing the concepts from material programming which exist today.

Chapter 2

Background

Today's generation live in increasingly technologically advanced times, and what thought impossible a mere 10 years ago can now be an integral part of our daily lives. There is a growing focus on using technology in all parts of our society, to a point where it is hard to avoid technology on any given day. As our society evolves and old technology become obsolete, our understanding of technology changes over time, along with our usage of the term itself.

Especially in recent years, researchers and enthusiasts alike have focused on combining technology with traditional materials and substances. Although emerging technologies such as 3D printing and Internet of Things (IoT) has enabled innovation on a whole new level, combining and interacting technology with otherwise inanimate objects is not a new concept. In 1884, The New York Times wrote an article about the Electric Girl Lighting Company [1] and their concept of providing women with lights embedded in their dress to function as hostesses or stage performers. This might be the beginning of a field today known as smart textiles.

Some researchers have taken this concept one step further, not only interacting technology with materials, but integrating technology with the material itself. The research field of interaction design has in recent times seen significant interest in how traditional crafting and formgiving practices (e.g. woodworking, pottery, textile production) can influence and inform design choices of interactive technology, an approach with a perspective on the materiality of user experiences and technology as a design material [25]. One such concept is computational composites, a term coined by Anna Vallgårda and Johan Redström in 2007 [27], and is background to much of the research done in this thesis.

2.1 Computational composites

Computational composites were introduced as a new type of computational material, providing the notion of the computer as a material in and of itself. The initial paper aims to describe and understand computers as a material for design, and detail how it can be a part of a formgiving practice. Traditionally, formgiving is the knowledge of a given material combined with the practical skills needed in the process of giving form to the material [28]. Take the formgiving practice of a cabinetmaker as an example. While harboring the knowledge of all different kinds of wood, including its material properties under various conditions, the cabinetmaker also inhabit the practical skills required for giving the wood its intended form, along with knowledge of the correct usage of an assortment of tools. In comparable fashion, imagine the computer as a similar type of material, with all the knowledge and practical skills needed for a formgiving practice. In this day and age, the computer as a material is not as versatile and flexible as traditional materials used in the application of formgiving. While the users may imagine the computer workspace as virtual, the material that is the computer is a physical one. Although not visible to the human eye, the electrical charges and exchange of energy performed at a user's command is a physical reaction, and it is essential to see the computer as a physical artifact to understand its use and potential form as a computational composite [28].

The concept of computational composites has tremendous potential, presuming technological advancements and future availability and versatility of such materials. Given the considerable potential of this concept, albeit latent and merely imagined at this point, it is important for to research and theorize the practices surrounding this concept, so that thinkable use cases, advantages, and pitfalls are researched when the technology catches up.

Although approximations to computational composites exist today, this thesis imagines a future where such technology is readily available and base our research on this premise. Since computational composites are rare in this day and age, this thesis cannot exclusively rely on previous studies to explore the potential of it, including its possible properties and use cases.

2.2 Material programming

Given all this envisioned future potential, it's relevant to delve into the questions of how to use and design this composite material, and further how to explore its dynamic properties and give it functionality. This is the research focus of this paper; once such computational composite materials exist; how does a user design or program it; how does a user interact with it to achieve a given functionality or purpose; what does a design practice for computational composites look like. In contemplating these question, researchers came up with the concept of material programming [26]. With a background in existing knowledge, modern research, and current and envisioned understanding, this thesis will develop a proof of concept for material programming using internet of things (IoT)

technology.

Material programming is foreseen as a way to design and program computational composites. The purpose of material programming is to explore the dynamic potential of computational materials [26]. The dynamic potential of the computational materials is a notion that differs depending on the material, and encompass all the changing properties and functionality of the material. Vallgårda et al. propose two different cases of material programming using physical tools to design them; color-changing material and shape-changing material [26].

2.3 Previous cases of material programming

The introduction paper to material programming [26], Vallgårda, Boer, Tsaknaki and Svanæs present two cases for material programming; a shape-changing material and a color-changing material. To program these materials, the authors present three tools; select tool, color tool and force tool. Before developing this thesis' proof of concept, it is pertinent to study and understand these previous cases and concepts.

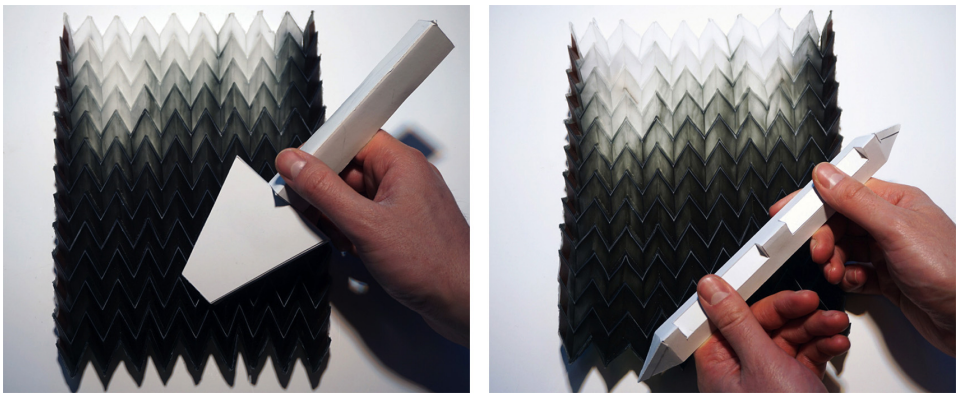


Figure 2.1: Select tool (left) and force tool (right) used on a simulated shape changing material [26]

2.3.1 Shape-changing material

The first case presents a composite material including a CO2 sensor and shape-changing ability. A video demonstration[1] aims to display how a user can program an behavior relation between the local CO_2 levels and a resulting change in shape. In technical terms, this behavior relation is an input-output relationship, with a user exhaling on the material causes change in CO_2 levels which functions as input data, and a subsequent output in the form of shape change on the material. It is worth noting that the material displayed

[1] `https://www.youtube.com/watch?v=0u7DJ-Hxd5s`

in the video does not actually have computational shape-changing ability, these effects are simulated.

The design procedure starts with selecting an area to program using the select tool. Following this, a user can choose another tool to implement change on the selected area. The force tool is then used to change the shape of the material, based on the initially selected area. Meaning the force tool affects the shape of the selected material in the first step. The proposed force tool consists of two sliders, corresponding to the direction of the applied force. After experimenting with the dynamic shape changing capability of the material, the user proceeds to breathe on the material while simultaneously moving his finger along one of the force tool sliders, correlating the exhaling with the power of the force. In a programming by example methodology, a user creates a relation between the breathing and applied force, constructing a transitive input-output behavior relationship. This concept of a mapping relationship between input and output is fundamental to the concept of material programming.

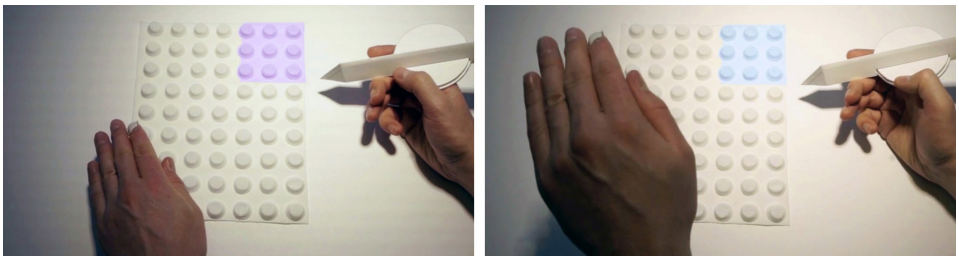


Figure 2.2: Color tool used on a selected part of a color-changing material [26]

2.3.2 Color-changing material

In the second case presented in the paper, the authors simulate a computational composite with the ability to sense proximity and change color. Similar to the previous case, a user begins by selecting an area on the material to be programmed, and then uses a color tool to implement color-change on the selected part of the material. The color tool is directly coupled with the change of color on the selected material. In the example video, this color change is computer generated imagery and used purely for instructional purposes, as it aims to illustrate how a similar functionality would work with an actual color-changing material. The case also incorporates a proximity sensor, which is programmable by a user. The proximity sensor in the material is subsequently programmed to be used as input data to change color on the material. The subject moves their hand up and down vertically above the material while simultaneously moving his finger along slider of the color tool. With programming by example, the proximity sensor is now coupled to the color changes. In this example, the researchers use proximity sensor data as input and color as output. This creates a mapping between the proximity sensor as input and color change on the selected material as output. In different ways, both of these cases explore the concept of a mapping relationship in material programming.

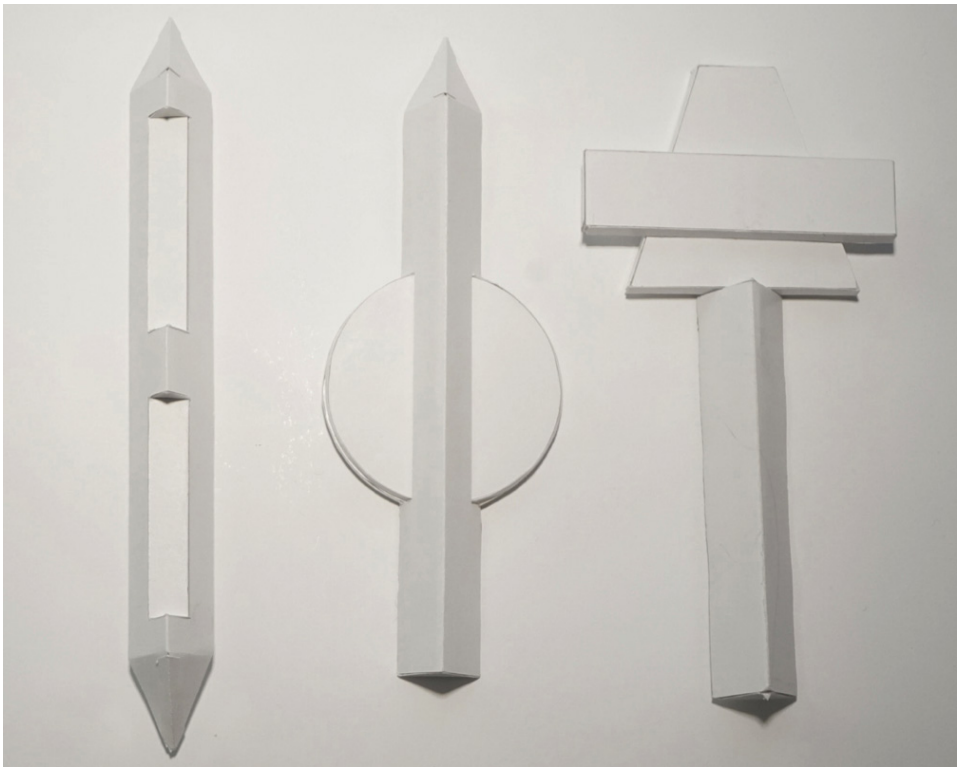


Figure 2.3: Envisioned tools of material programming, from the left; force tool, color tool and select tool [26]

2.3.3 Tool devices

As this concept very future-orientated and still in its early phases of exploration, the prototype to be built in this thesis aims to illustrate the concepts using a minimal case that still illustrates the ideas of material programming. That means it might be advantageous to only include the most essential concept to illustrate the case. In the initial paper introducing material programming there are three tools envisioned for this future design practice; select tool, color tool, and force tool.

2.3.4 Select tool

A select tool is used for selecting which part of a material is going to be programmed. This is arguably one of the most integral concepts to display the intended functionality envisioned in material programming, and can be considered a necessary introductory step for using other tools. For its important role in the design practice, this thesis deems it essential to demonstrate functionality akin to select tool it in the prototype.

2.3.5 Color tool

Color tool is one of the acting tools in material programming, used to implement change on the selected part of the material. Color has many applications scenarios for computational composites, for instance as an inherently physical attribute, part of the aesthetics or as colored light emission. It can be conveniently used in some form for most materials and for this reason this thesis wish to include functionality comparable to color tool in the proof of concept.

2.3.6 Force tool

Force tool is also a type of acting tool, used for exploring the composite material's ability to change shape. This tool is largely material specific, in that there are very few materials in existence today with this ability. At least, very few materials able to change physical shape following a user's command. As such it would be very difficult to demonstrate a force tool in action on a specific type of composite material and even more complicated on a generic material. With this in mind, this thesis have chosen to disregard the force tool for this proof of concept, as this prototype desire to focus on more integral concepts of material programming.

2.4 Related research

MIT Tangible Media Group research a future world where digital and physical artifacts work seamlessly, by giving dynamic physical form to digital information and computation. Hiroshi Ishii et al. published a paper in 2012 titled "Radical Atoms: Beyond Tangible Bits, Toward Transformable Materials" [8]. Looking beyond tangible computational materials, they propose the concept of radical atoms, by assuming future materials inhabiting dynamic form- and appearance-changing abilities which can be altered much like pixels on a screen [8]. This concept share many similarities to material programming and computational composites. To illustrate this concept in the context of digital and physical technology, the authors use an analogy to icebergs floating in water (see figure 2.4). The water represents the digital world, and everything above the water is the physical world. A graphical user interface (GUI) is digital information presented on a screen and such information would be entirely in the digital world, with no properties present in the physical world. It permits visual interaction with information on the screen using some controller devices, such as mouse, touch-screen or keyboard, but however powerful, GUIs are not consistent with our interactions with the rest of the physical world [8].

Tangible interfaces attempt to mitigate this gap, by presenting and manipulating information in a physical embodiment. Typically this entails using technology in conjunction with materials to expose a tangible interface for a user to interact with. This concept is located in the middle of figure 2.4, with the iceberg tip above the water representing the physical attributes available for tangible interaction. This works well in specific circumstances but

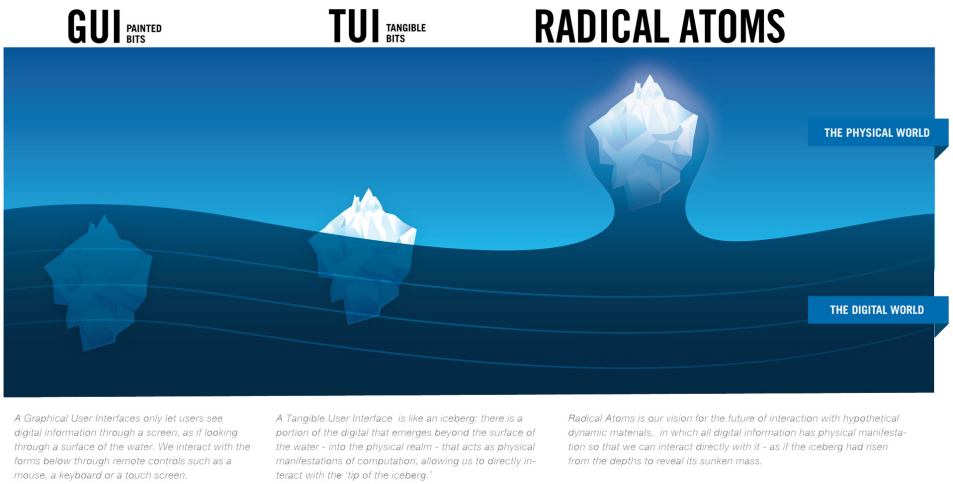


Figure 2.4: Radical Atoms as envisioned by MIT Tangible Media Group [8]

lacks dynamic and physical properties needed to represent changing digital information. In contrast to digital information on a screen, it is significantly harder to change attributes of an object in the physical world, such as form, color, and size. Improving this would require materials with the ability to dynamically change all of its physical attributes, which is not possible with today's technology. Assuming future advancement in technology, Radical Atoms is an envisioned dynamical and physical material that is bidirectionally coupled with an underlying digital model, meaning that dynamic changes in the physical form is reflected in the digital states in real time and vice versa [8]. The physical material uses dynamic affordance as a medium for representing information, while enabling input and output to control the physical material properties and consequently its computational state. This is the functionality is the iceberg to the right in figure 2.4 is trying to convey.

The paper contemplates three core functionalities required of the material for this to be realized; transform, conform and inform. Transform enables the material to change shape in accordance with the computational state. Conform adheres to constraints by the environment and user input. Inform is the dynamic affordances, visual and tangible changes perceived by the user following its transformational capabilities. Rather than focusing on developing a technically feasible solution, this concept functions as a guide for future research exploration. Both in philosophy and envisioned properties, this concept is closely related to that of material programming and computational composites. Although instead of using computer-generated imagery to illustrate the material, this thesis will concentrate on an approximation to such a material today using IoT technology.

The paper go on to provide an illustrated use case for this concept, a shape-memory clay called Perfect Red. Figure 2.5 display a storyboard of its envisioned functionality. Perfect red represent a future substance inhabiting all the desired functionality related to Radical Atoms; transform, conform and inform. The scenario is even better illustrated in the

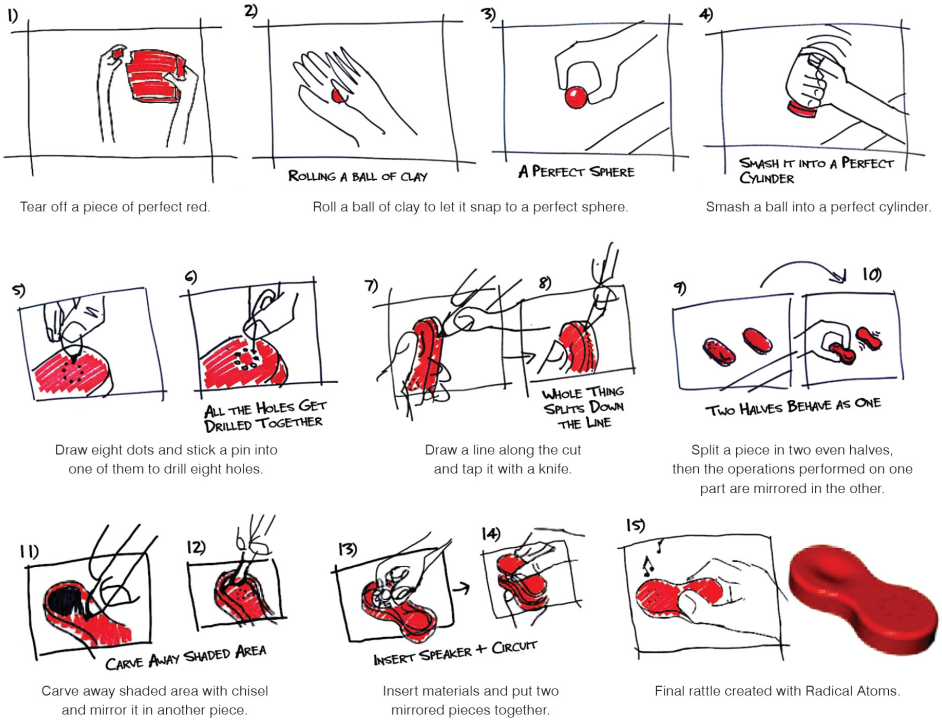


Figure 2.5: Radical Atom concept Perfect Red envisioned functionality [8]

demonstration video[2], but both storyboard and video share the goal of creating a rattle. When the clay-like material is rolled into a ball, it snaps into a perfect sphere. The material is acted upon by external tools, and while the demonstration uses a general instrument such as marker pen, knife, and chisel tools, the tools are not core to the ideas exhibited. This is a distinction the concept of material programming, where the tools used to program the materials are deemed a core functionality and detailed as different purpose-specific tools.

In Perfect Red, the clay-like material can be separated into two equal materials, which both experience the same forces acted upon them. The materials become mirrored, and a change in state on one result in a change on the other. Screenshots of the video demonstration in figure 2.6 display join functionality, where two spheres with holes are combined with a cylinder between them, transforming the material into a solid composite substance. Figure 2.7 show cut example functionality, where a line is drawn with a marker pen and subsequently touched with a knife, leading to the material splitting where the line is.

[2] `https://vimeo.com/61141209`

Figure 2.6: Perfect Red video illustrating join functionality using two spheres with holes joined with a cylinder. The parts ”snap” into one cohesive material as they are pushed together [8]

2.5 Programming paradigms

The goal is of material programming is to conceptualize a design practice material which allows non-technical participants to “program” materials using tools. The programming performed on and between such materials would be contained within the material, with the computational power being a property of the material itself.

Use 3D printing as an example. Today, there is a gap between a designer’s 3D design and the material’s final state. A design is usually performed and refined on a computer, and the ideal goal is to design a finished product before printing the object using a printer. If some part of the material ends up wrong in the final state, there is no way to only change this part of the material. The designer needs to redesign the material and print the entire object again from scratch. Material programming can skip the part of designing on a computer, and give the material itself the possibility to change state. In such an environment it is important to have a practice that enables the designer to stay within the material realm when designing interactive artifacts [26], and not have the need for using a computer or external equipment. Thus the material itself is the only part needed in this design process, as it changes in real-time based on input and interaction of a designer.

Like textual programming has syntax and rules required for it to compile and execute, material programming would need some kind of rules in the same regard. But a major part of the textual complexity is gone, and an end-user could learn how to do something by seeing the cause and effect in real time. No compilation or errors. A user can pick up a tool, try, and see if the action had the desired effect instantaneously. Thus the threshold for “programming” is significantly lowered - to the point where one could almost entirely rely on learning by doing. For this to be a reality, it is pertinent to develop a design practice with a human familiarity of tools, recognition of functionality and affordances, and visual understanding in mind.

Textual programming is by far the most prominent form of development today, but there exist other alternatives out there as well. Material programming is a proposed design

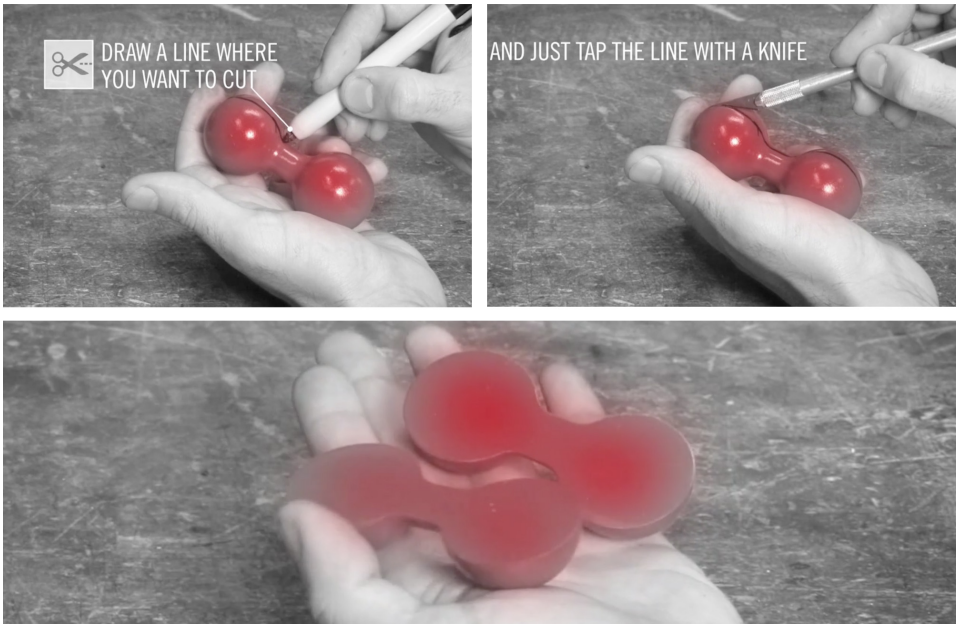


Figure 2.7: Perfect Red video illustrating cut functionality using a marker pen and knife [8]

practice which has its roots from three different programming environments; visual programming and tangible programming. Programming by example can also be utilized but is not relevant for this proof of concept. To gain a better understanding for a future design and programming practice, this thesis will look deeper into these environments and gain understanding for how they are used.

2.5.1 Visual programming

"A picture is worth a thousand words" is arguably one of the most eminent idioms in the English language. It presents the notion that a visual presentation is far more descriptive than words, and a complex idea which would require a paragraph to explain, can possibly be conveyed with a single image. It is thus not hard to imagine why researchers early on began experimenting with visual programming as a way to lower the threshold of programming practice and present problems in a visually recognizable context. Thus, visual programming utilizes people's ability to easily recognize and work with visual patterns, and thereby minimizes the need for learning and working with textual syntaxes [26].

One type of visual programming is called dataflow programming and was pioneered at MIT in the 1960s. William "Bert" Sutherland is one of the first to create such a programming environment [22]. As opposed to a one-dimensional textual programming language, he envisions a two-dimensional environment, where the user could visualize and modify the flow of the program on a screen. He modified his brother's revolutionary interactive

graphical editor Sketchpad [21] to be able to interactively create and modify the flow of a program, as well as creating custom operators. A sketch using this program can be seen on the left in figure 2.8.

Sutherland's program (and many similar to today's date), is based on the concept of lines and boxes, similar to flowchart diagrams. The boxes are treated as operations or entities and connected lines and arrows represent the flow of the program or relationship of entities. In the decade to come after, systems grew considerably in complexity and proponents of visual programming felt that current systems focused too much on the logic of a program, when data manipulation was in fact the intended purpose of the application.

This inspired many researchers to focus more on data-driven programming, most notable outcomes including Prograph [20] and Labview [24]. Prograph used a similar visual dataflow programming paradigm which it later expanded to include elements from object-oriented methodology, creating an "objectflow" paradigm. Development took place mainly throughout the 1980s, with a cross-platform application released in early 1990s. It worked similar to Sutherland's program in the sense of using lines and boxes to direct the flow of the program. Despite a 1989 MacUser Editor's Choice Award for Best Development Tool and a free software version still available, it never reached the widespread use it aspired to.

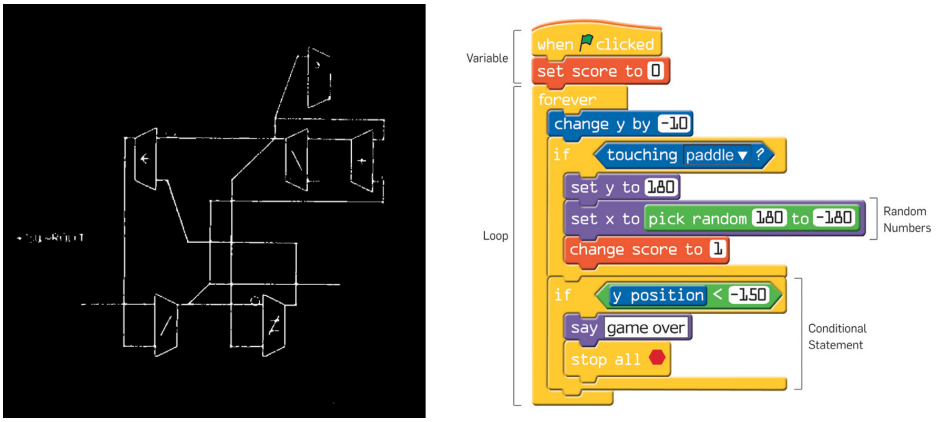


Figure 2.8: Sutherland's program on the left [22] and Scratch functionality blocks on the right [18]

Whereas Prograph's goal was a visual programming language for generic software development, Laboratory Virtual Instrument Engineering Workbench (LabVIEW) took an approach of developing more domain-specific software. The software is commonly used in the areas of data acquisition, instrument control and industrial automation - more specifically designed for engineers and scientists to develop user interfaces for external hardware and perform testing, measurement and control of technical equipment [24]. These are areas which benefit from a visual programming environment as opposed to a textual one, and could be considered a reason why Labview still is in widespread use today. The visual dataflow environment enable its users to interconnect hardware or other sensory input with visual objects on the screen, in form of a gauge, switch or other tool, to monitor and

control equipment in real time.

Max [17] is another visual programming environment, this one focused on multimedia interaction, particularly used by artists to create interactive multimedia for art, music or video performances. Similar to previously mentioned environments, Max lets users build interaction on a visual canvas, and is especially appreciated for its versatile interactivity and real-time sounds. Think of a Max program as a collection of boxes interconnected by lines. Max emphasize the use of text for defining a box's content, as opposed to having different visual appearance of each box related to its respective functionality. It is reasoned that having a large amount of distinct boxes with different design makes it difficult to remember each functionality corresponding to a box, and argues text works better mnemonically. In this respect, Max differs from many other graphical programming environments, although it is worth mentioning the program has made significant changes since the time of creation and relies more heavily on specific functionality boxes now than before.

Scratch is a visual programming environment targeted at novice users, created with educational purposes in mind [18]. It is used at community centers, libraries, museums, elementary schools, high schools and even universities, as an introduction to programming and a way to learn thinking creatively while working systematically. By giving users an opportunity to create interactive stories, animations and games in an environment that enables the user to create whatever they desire, Scratch aspire to make learning programming fun. Scratch programs are created by dragging blocks around on a visual canvas, similar to previously mentioned environments.

With an event-driven programming paradigm, Scratch introduce users to various blocks, which make up the functionality of the program. With blocks such as control and data blocks, the user is familiarized with coding concept such as variables, booleans, if-else, delay and while, often using more comprehensible names. The blocks as put together to form functionality, as seen on the right in figure 2.8. Operator blocks contain mathematical operations in the context of programming, with a full set of math functions similar to a more low-level programming language. Less/greater than, equal, and/or, not, random and modulo, are just some of the functionality available for use. Sensing and motion blocks are blocks that detect things and blocks that move things, giving an equivalent of a program's input/output functionality.

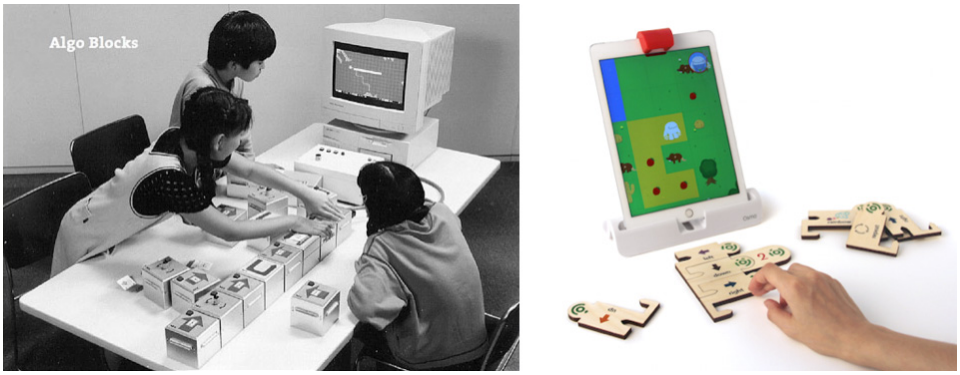
Material programming uses elements and concepts from these programming paradigms when proposing the practice of material programming.

2.5.2 Tangible programming

Tangible programming is a way of interacting with physical objects in a programming environment. The different tangible object can represent elements from programming, such as variables, commands, operations and flow control. Usually the programming is performed by arranging the objects and manipulating their position in a physical space. The thought process behind the programming environment is, "what better way to create something than to build something using your own hands". Using this method, users can

potentially learn about powerful ideas and concepts in a comprehensible context. Tangible programming environments are primarily used as an educational tool to provide children with an introduction to programming [26], using physically and visually recognizable objects such as toys.

One of the earliest examples of a tangible programming environment was developed by Suzuki and Kato, with a concept called AlgoBlocks [23]. AlgoBlocks was a programming language and game using computational building blocks put together, with the goal of connecting the blocks in a way that guided a submarine through an underwater maze displayed on a computer screen. This programming language in action can be seen on the left in figure 2.9. In a similar fashion, the tangible programming environment Strawbies [7] uses wooden tiles to guide a character on a computer screen. This environment can be seen to the right in figure 2.9.



.

Figure 2.9: Tangible programming environments AlgoBlocks [23] to the left and Strawbies [7] to the right

Tangible programming environments use human experiences to create physically recognizable and comprehensible functionality. It relies on familiarity with objects and an instinctive understanding of manipulation of artifacts. It facilitates collaboration through a shared environment, but is not capable of constructing advanced programming and algorithmic expressions.

Chapter 3

Research questions and strategy

The main intention of this thesis is to create a proof of concept for material programming using internet of things (IoT) technology. By building on concepts previously detailed by Vallgårda et al. [26], this thesis will develop a prototype for this design practice.

3.1 Research questions

The research questions are formed to gain knowledge and experience in utilizing IoT technology for realizing material programming. As such, a natural question follows as to how to showcase functionality for a concept which does not yet exist.

3.1.1 Research question 1

Material programming is a somewhat recent concept in a technological perspective and consequently, the research is still in its early phases of exploration. The research carried out so far has been mostly theoretical, aimed at envisioning concepts, and predicting functionality and use cases. Given the futuristic tendencies of the concept, the main intention in current research is to establish a baseline theory and conceptualize what material programming entails. From this perspective, some of the research done on the topic today consist of strategic forecasting of future technological development. This way, when the technology makes progress to a point where material programming is feasible, the theoretical concepts and functionality is ready to be implemented. As such, this thesis is limited by the current technological environment and the examination becomes rather; what can you demonstrate using today's technology and what functionality of an envision concept has to be showcased for it to be considered a prototype of this concept. The first research question is formulated as follows:

What is a minimal case (proof of concept) to illustrate the ideas of material programming?

With this research question, the thesis' case aim to showcase the most minimal case which still illustrates the ideas core to the concept of material programming. Before a physical prototype can be designed and developed, this questions needs to be contemplated. To be able to present a proof of concept for material programming using IoT technology, it is essential to define and formulate what a prototype needs to be display to be considered a physical case of material programming.

3.1.2 Research question 2

There has been performed some theoretical research on the subject of material programming, although limited work has been done on how a full-fledged physical prototype would look in practice. The existing cases illustrating concepts from material programming has been performed primarily with the aid of computer-generated imagery (CGI). In these cases, CGI has been used to alter perceived physical interaction with materials, to make an appearance of inhibiting color- or form changing abilities. The most notable example of this is in the original paper detailing material programming [26], using CGI to make a material exhibit color changing properties.

There also exist low-level prototypes displaying parts of the concepts of material programming. These examples can work well for illustrating a specific functionality but does not show interaction in the context of material programming. As the technological development has yet to reach a level where envisioned material programming is feasible, this thesis aims to develop a high-level proof of concept showcasing the most integral ideas using IoT technology. The second research question is formulated as follows:

How suitable is today's internet of things technology to prototype material programming?

As opposed to the majority of related scientific demonstrations of material programming, this case aspires to develop a physical prototype for a user to interact with. A tangible prototype can naturally only utilize existing technology, and this thesis approach this using technology related to the concept of internet of things (IoT). To answer how suitable IoT is for prototyping material programming, this question has to be seen in the context of research question 1. This way, the suitability of IoT technology can be regarded from the perspective of requirements for a minimal case for material programming.

3.2 Research strategy

A research strategy is an overall approach to answering research questions [16]. In Oates' 2006 book, the author defines six strategies for answering research questions; survey, design and creation, experiment, case study, action research, and ethnography. The most relevant research strategy for this thesis is design and creation, which focuses on developing new IT products, also called *artifacts*. Typically this involves a problem-solving strategy using an iterative approach, and an idea of 'learning by making' to develop the proposed solution. Seen in the light of this thesis, this strategy relates strongly to the approach of prototyping. By using prototyping as the main method of research, this thesis aims to showcase new scientific ideas and concepts enabled by emerging technologies. The prototype may not display all the envisioned functionality of a full-scale realization of material programming, but aspire to make it more comprehensible in a physical and tangible environment. The notion of material programming is an abstract one, with previous research focusing primarily on computer-aided simulated visual demonstration. The abstract nature is in part a result of the current technical un-feasibility of the concept, leading to difficulties in explaining its material and physical properties. This proof of concept desire to conceive the notion in an environment uncoupled from its abstract nature, to reveal and fathom the possible physicality and tangibility of it.

This thesis proposes using IoT technology in this new domain to demonstrate the technical viability of material programming and core concepts from it. The prototype artifact developed is the main focus of this thesis and is itself the main contribution to research knowledge. The proof of concept is in the form of a prototype using IoT technology, which demonstrates ideas and concepts relevant to material programming.

3.3 Prototyping

Prototyping is commonly used in idea conceptualization and concretization, as a method in an early design process to showcase and refine intended functionality and concepts, and subsequently contemplate the execution of these. A prototype is an early stage demonstration of how a full-scale system or concept would work, and typically focuses on some core ideas or principles integral to the concept demonstrated. A prototype does not need to showcase all functionality as intended in an envisioned product, and usually concentrate on some specific functionality. A prototype is merely meant as a visual or physical guide in trying to explain a concept, idea or product. It facilitates an evaluation of a suggested design before producing an end product. Testing in a prototyping process leads to refinement of ideas, which leads to re-designing of new prototypes which are again evaluated. This is an iterative process, which ends with a product or concept ideally rid of many of the initial complications and confusions. The prototyping process culminates in a final conclusion, where the prototype is evaluated according to the initial requirements.

Houde and Hill wrote a paper in 1997 answering the essential question related to prototyping; "What do prototypes prototype" [6]. In their, more technical expression, prototypes

are widely recognized as a core means of exploring and expressing designs for interactive computer artifacts. By focusing on the purpose of the prototype, or rather what the prototype prototypes, this proof of concept aims to make better decisions about how the prototype is built. The prototype must have a clear and obvious purpose and goal, in this case, illuminated and based on the research questions about illustrating the material programming using IoT technology. Before prototyping can begin, it is important to establish what material programming means for this thesis - meaning how and what an IoT prototype would need to display for it to be considered a proof of concept for material programming. Both research questions tackle these considerations head on, with the first question being integral in defining what needs to be demonstrated from material programming for this to be actually be considered material programming. That is, what is a minimal case to illustrate the ideas of material programming. Following this, the second questions contemplate, how suitable is today's IoT technology to prototype material programming. This question is necessarily dependant on the first question, and what a minimal case is defined as in the context of this thesis.

Chapter 4

Technology

This chapter will detail various technologies and concepts relevant to the thesis and the development of a proof of concept. This proof of concept will emphasize the use of open source and open standard technology, which facilitate a transparent process and an easily replicable prototype.

4.1 Internet of Things

The Internet of Things, abbreviated IoT, is an infrastructure for the information society, enabling advanced services by interconnecting virtual and physical things and interchanging information using existing and evolving communication technologies [19]. More plainly, IoT is a network of connected devices embedded with sensors, actuators, electronics, software and connectivity which enables an exchange of data and information. Today's society sees an increasing focus on enabling connectivity and perceptive capabilities in everyday "things", such as fridges, cars, machinery, watches and similar, as well as heart monitor implants, biochip sensors in farm animals and other biomedical applications. IoT aims to bridge the gap between the physical world and its representation within the digital world [29], where the idea is to gather information and the state of the "things" to benefit from this information in a broader context. All devices in a network are given a globally unique identifier, usually in the form of an IP address. IoT has a large focus on machine-to-machine (M2M) applications which communicate without human intervention [2]. In a typical scenario, multiple devices connected to the internet are equipped with sensors to gather data and publish this to a central information hub. This data is subsequently aggregated, analyzed and used to perform changes autonomously based on the sensory information. This proof of concept will use IoT concepts and technology as a basis for a proof of concept. The proof of concept will develop and utilize a number of interconnected and coordinating devices with sensors and actuators to demonstrate material programming.

This chapter will further detail devices, technologies, and protocols relevant for a prototype.

4.2 Hardware: Sensors

This section detail possible sensors for a proof of concept. A sensor is a component used to detect events or perceive a change in a given environment, and subsequently, send this information to another electronic device.

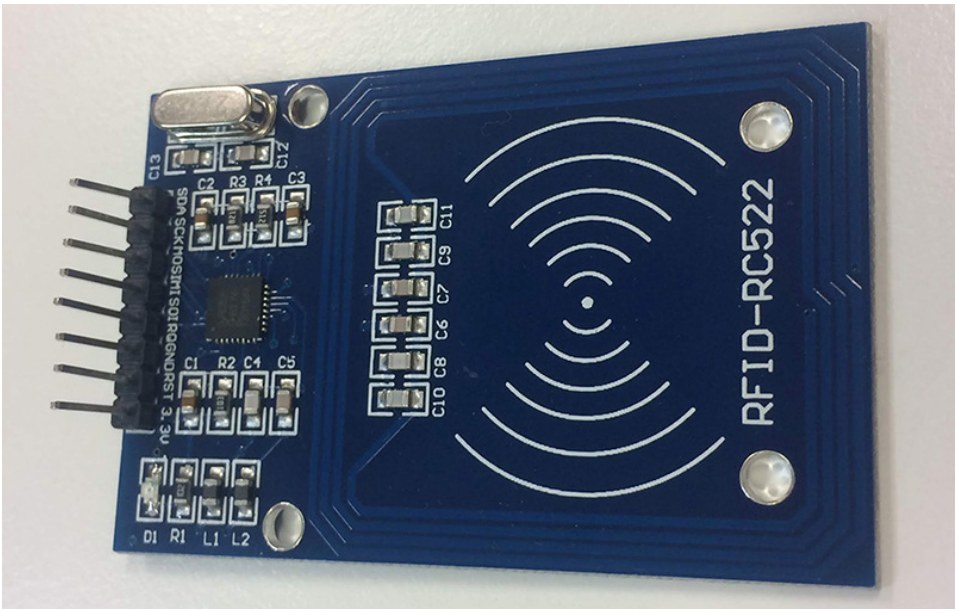


Figure 4.1: RFID reader/writer commonly used in prototyping

4.2.1 Radio-frequency Identification (RFID)

Radio-frequency identifications, abbreviated RFID, is a communication technology used to read and identify tags with electronically stored information. This has many applications in today's society, most notably as contactless payment cards and car toll stations.

RFID systems can be classified in different ways. An active RFID component means it uses electricity and has a power-supply, whereas a passive RFID component uses no electricity. Between two communication RFID components, at least one of them has to be active. The different RFID systems are thus Active Reader Passive Tag (ARPT), Passive Reader Active Tag (PRAT) and Active Reader Active Tag (ARAT). In a prototyping environment, the most common system to use is Active Reader Passive Tag. An active RFID reader can be seen in figure 4.1 and a passive RFID tag can be seen in figure 4.2.

Figure 4.2: One type of passive RFID tag

Technically an RFID reader/writer is a wireless communication protocol, a sensor and an actuator. It can read on and write data to an RFID tag, and uses wireless communication protocols while doing it.

4.2.2 Ultrasonic sensor

Ultrasound is sound waves with a frequency higher than human hearing capacity, usually from 20 kHz to several gigahertz. Ultrasound has a number of potential industrial applications, from ultrasound imaging and sonography, distance measuring and object detection, as well occurring naturally in animals like bats to improve locating and hunting ability. In relevance to the proof of concept, an ultrasonic sensor can be used for distance measuring. Such a sensor can be seen in figure 4.3.

Figure 4.3: A ultrasonic sensor

4.2.3 Button

A button is a simple switch which sends a signal when pressed. This has many potential use cases and scenarios.

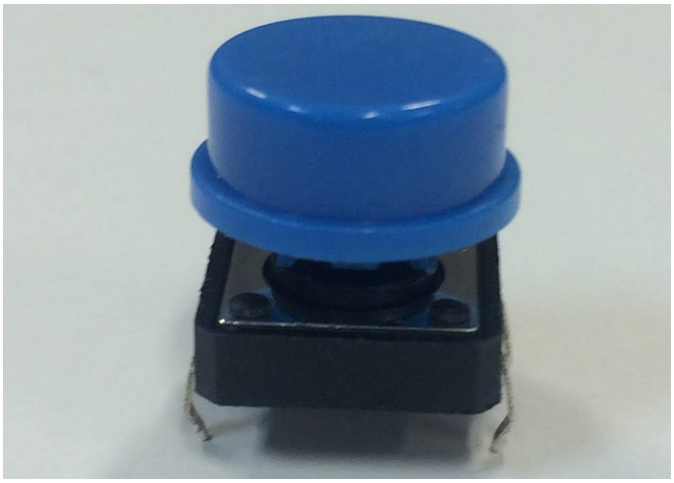


Figure 4.4: A button commonly used in prototyping

4.3 Hardware: Actuators

As opposed to sensors which gathers information, actuators are parts in a system responsible for performing actions. The term is typically used to describe components executing a mechanical movement, such as a motor.

4.3.1 RGB LED

Standard RGB light-emitting diode (LED). RGB is a color model using red, green and blue, blending the colors together to create most colors in the spectrum. This is the most standard color model used on colored lighting and digital media, such as computer screens, phones, and televisions. See figure 4.5 of a common RGB LED.

4.4 Hardware: Devices

For a computational material, a component with a processing unit is needed. A microchip is essentially a small computer with a processing unit, capable of connecting to external input and output signals.

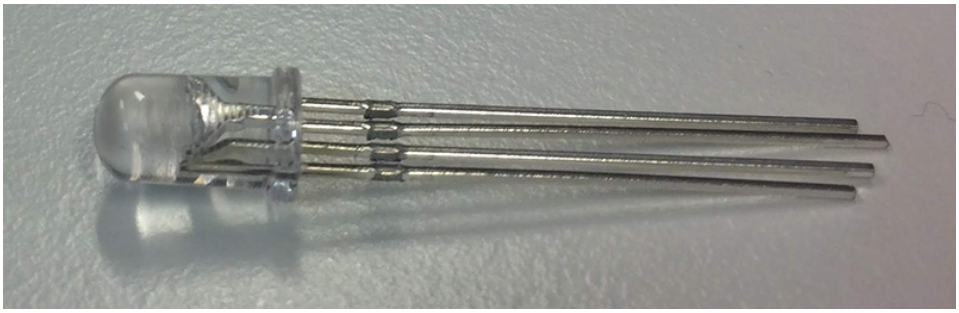


Figure 4.5: An RGB LED commonly used in prototyping

4.4.1 Microcontroller: ESP8266MOD

The ESP8266 is a low-cost microcontroller produced by a number of manufacturers. This specific manufactured model has 16 general-purpose input/output (GPIO) pins, including both 3.3 V and 5 V power-supply pin. See figure 4.6 for one type of modified version of an ESP8266 microchip.

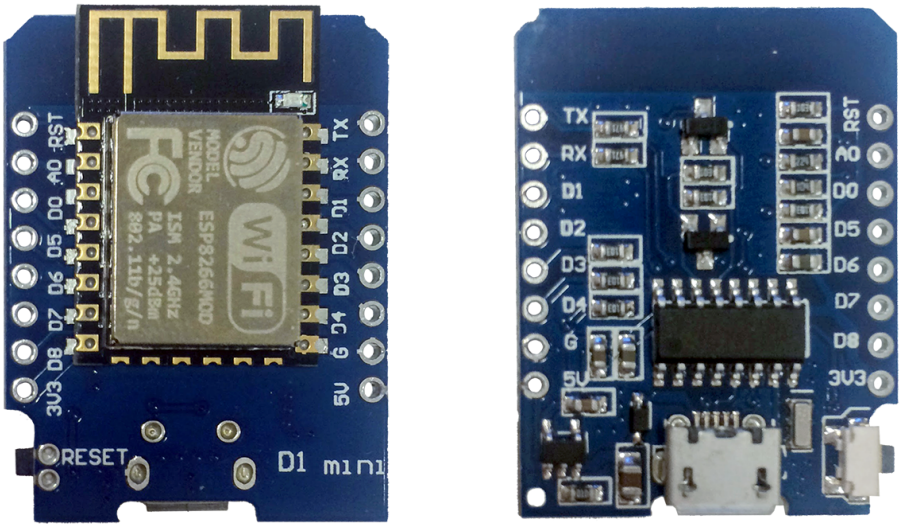


Figure 4.6: A version of an ESP8266MOD microcontroller. Front side on the left and back side on the right

4.4.2 Raspberry Pi 3

Raspberry Pi 3 Model B single-board computer with wireless Local Area Network (LAN) and Bluetooth. The LAN capabilities can prove useful if the prototype requires a local communication network. See figure 4.7 for an image of a Raspberry Pi 3 Model B.

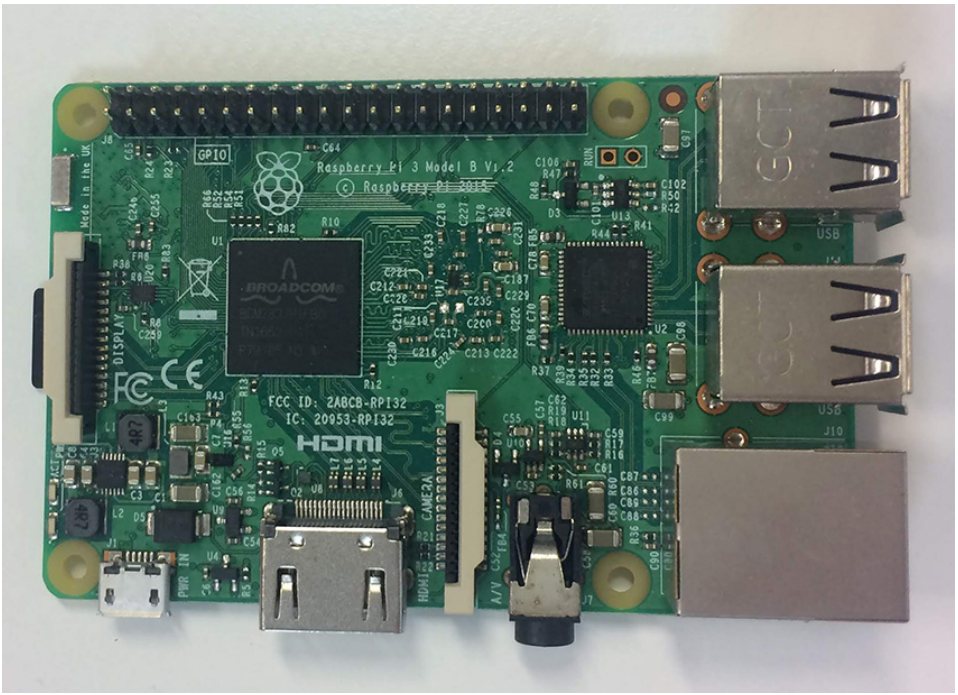


Figure 4.7: Raspberry Pi 3 Model B

4.5 Software

This section specifies software technology used in development of prototypes.

4.5.1 Software platform

To develop a proof concept, a platform for developing prototypes is required. Arduino is an open source hardware and software company delivering an electronic prototyping platform for use with microcontrollers. Arduino sells its own microcontrollers for use, as well as provide an integrated development environment (IDE) for developing software. The Arduino IDE is not limited for use with the Arduino hardware, and a large community exist creating libraries and examples for sensors, actuators, and microcontrollers. It is

arguably the most supported software environment for electronic prototyping and as such became a natural choice to use as a software platform. Early experimentation for this proof of concept also used Arduino hardware, but later progressed to smaller microcontrollers, while still maintaining the use of the Arduino IDE.

4.5.2 Programming language

The Arduino IDE can be used develop in both C and C++ programming language and compiles both of these languages natively. In addition to this, Arduino offers a programming language of its own, which is merely a set of C/C++ functions that can be called from the code. As such, Arduino is not a stand-alone programming language, rather a dialect of C and C++.

4.6 Communication protocols

For a proof of concept to work, it is integral for material parts to be able to communicate reliably and efficiently with each other. Transmission of data and information is a prerequisite for everything else to work as intended, and as such much thought was given to the mode of communication. Infrared was explored as a possible form of wireless communication but was quickly dismissed as too slow and unreliable. Bluetooth was also dismissed for lacking versatility as a result of its need to pair and certify connected devices.

Given a microchip's built-in WiFi module, a natural step in the process was researching methods of transmission over the Local Area Network (LAN). Mozilla's Project Things shows great promise but was deemed too risky considering its early stages of development and testing at the time of writing. Particle.io's device cloud has great infrastructure features and remote device update, but currently requires a purchase of proprietary hardware, which makes it unfavorable for this prototype. After further time examining different methods of communication over the internet, the choice landed on using the MQTT protocol.

Chapter 5

Proof of Concept: Design

This thesis aims to demonstrate material programming by developing a proof of concept using Internet of Things technology. By developing a prototype, this thesis aspires to further increase the understanding of material programming in the context of the research questions. Although it is essential to have an understanding of how to answer the research questions before starting to develop the proof of concept, the prototyping process itself will provide and facilitate knowledge learning along the way, which helps in further understanding and researching the concept. The research questions are inherently and closely related to the development of the prototype, and as such it is important to have an understanding and detailed plan for prototyping material programming in relation to IoT technology.

5.1 Material programming core concepts

A material programming prototype has to simulate a computational composite material consisting of many material parts. Simulating these envisioned microscopic material parts and their functionality is important to illustrate material programming. In short, a composite material enabled for material programming, consist of many computational material parts. The second research question asks how suitable IoT technology is for prototyping material programming, but before such a development can occur, this thesis must define what needs to be showcased for this proof of concept to be considered material programming. This is the task of the first research question, which contemplates what is a minimal case to illustrate material programming. Answering that question, this thesis has to address and list the core concepts needed to display a prototype of material programming. This thesis considers three core concepts integral to showcase material programming:

1. Material parts should have computation and communication capabilities
2. Behavior and interaction of material parts can be programmed using tools
3. A tool should be able to identify and select distinct material parts of a composite material

These are the core concepts this thesis aims to demonstrate using IoT technology. In other words, these are the ideas this thesis deem necessary to showcase in a minimal case for material programming. The core concepts are organized in a manner which describes the three most important aspects of material programming. Number 1 describe the material itself and its properties. Number 2 describe the functionality of the tools involved in the practice. Number 3 describes the connection between material and tool. To successfully illustrate these core concepts, more specific requirements for each concept are needed for a prototype:

1-a. A material part should be able to sense and actuate things

1-b. The material parts should function as one cohesive material with internal communication

1-c. A change in one material part should allow for a change in another

1-d. Material parts should have computation capabilities

1-e. Material parts should themselves store the programmed behavior and interaction

2-a. A tool can program behavior and interaction on material parts

3-a. A tool can identify and select a material part or area (set of materials parts)

This less ambiguous classification of a minimal case makes a realistic evaluation of functionality more attainable. Using this list of requirements, a proof of concept can more easily be assessed from a scientific perspective.

5.2 A minimal case

Using ideas from related cases and the formulation for a minimal case above, the prototype can be designed to accommodate these requirements. The prototype will be considerably rooted in current technological concepts and development from IoT, and aim to illustrate material programming using these.

Core concept number 1 detail that the material part should have a sensing and an actuating component. The sensing ability of a material part should be something with a wide range of data readings, to provide dynamic input and interactivity. An ultrasonic sensor is a distance measuring component, which would provide a user with an interactive way of changing the sensing data of a material part. The sensed data of a material part could in this instance be modified by altering the distance from an ultrasonic sensor to an object. An actuating part should be something easily visible and discernible in various states. An RGB LED has three distinct lights which combined can display a wide range of visible

light. As an actuating component, colored light is easily recognizable and distinguishable in different colors and light strengths. As such, a distance sensor as a sensing component and a colored light as an actuating component both display behavior and interactivity desired of each material part. The core concept also detailed every material part should have computation and communication, and neither of these components enables this. A microcontroller is a small computational device, which can be used to communicate using wireless technology as well as connect to external components. Using these components for each material part would make each part consist of an ultrasonic distance sensing component, an RGB LED actuating component, and a microcontroller handling computation and communication.

A large focus has been put into the mode of communication and how to best use utilize fast and scalable communication between microcontrollers. An integral part of the exploration and research is how to select and identify material devices. Before diving into specifics of the case, it is imperative to gain an understanding of what is built and how behavior and interaction in this proof of concept is achieved. Using the requirements above, a list of scenarios for how a minimal should function has been formulated. The material parts integral to this proof of concept can also be regarded as material devices. This case has the following interaction scenario:

- A user uses a material tool to program behavior on and between pairs of material devices
- The behavior programmed is a mapping relationship between two material devices which specifies an input device, output device, output color and color mode
- The input device publish data from a distance sensor to be used as input data for the output device
- The output devices subsequently uses this distance data, modify it according to given color mode and display it as light output with the selected output color using an RGB LED
- Input-output behavior is merely a mapping relationship which displays color output in real time based on input and given behavior attributes. After a mapping relationship is made, a user can manipulate output by changing the measured distance on the input device
- The affected output will vary dynamically depending on the input and behavior attributes given when the mapping relationship was made
- The output is altered in real time and only when given input, meaning the luminous output will only display or modify once an input device's distance sensor is stimulated

In this scenario, the material parts of the composite material or rather material devices in this prototype, are programmed with behavior between two material devices. This behavior is defined as an input-output relationship, where the behavior of an output device is solely dependant on the interaction stimulated on the input device. A tool is used to select the material devices and change the attributes of a behavior relationship between material

devices. Here is an example of one scenario. A mapping relationship is made with material device 1 as input and material device 2 as output. The behavior of the relationship is determined by the mapping parameters; output color green and normal mode. When a user moves their hand over the input device 1, the device sends input data to the output device, which result in a green light displaying on the output device. If the user moves their hand closer to the input device, the output color will shine even brighter. When a user moves their hand away, the green color turns off and stops displaying light of the output color. This example is made solely to illustrate that when programming these material devices, a user merely programs a behavior relationship between two devices, where one device is declared input and the other output. Also defining this relationship is two behavior attributes; output color and color mode, which alter how input is displayed on an output device.

On a more theoretical level, the concept of material programming is having miniaturized programmable materials making up the fabric of a material itself, meaning a material consists of a number of smaller material units which joined together as one composite material makes up the programmable material. Embedding this as miniature technology on such a small scale is not possible with today's technology, and as such this case aims to show how ideas from material programming can be illustrated and realized with IoT technology. In short, a user program the behavior between distinct material devices (parts), and the combined collection of all material devices (parts) constitutes and simulates one physical programmable material. The programmable part of the collection of materials is the behavior between the material devices, better regarded as a mapping relationship.

5.3 Design practice

The material programming design practice envisions programming behavior on a composite material by selecting areas of the material to perform actions on. For this selection process to work, it is imperative for a select tool to differentiate and identify individual parts of the composite material. By analogy, imagine editing a drawing in an image editing software, where the aim is to change color of a house in a photograph. Before telling the program to change color, it has to be instructed on what parts of the image it should change color on. To a achieve this, there exist a tool to select all the individual pixels the house is comprised of. Similar to selecting an area on a composite material, a select tool has to identify and resolve all the different "pixels" it wants to perform an action on. In this prototyping context, the individual pixels represents the individual devices able to perform actions on. A pixel is a unit in digital imaging which is a physical point on an image and the smallest controllable element of a picture on a screen. As such it is a great analogy for what this case is presenting, but the unit does not translate well to describing points or objects in a physical space. In a computer science and network environment, a node is basic unit used to describe a device or data-point in a network. When describing devices connected to the internet, a node is anything with an IP address. For this reason, node is considered a good way to describe and name material devices in this proof of concept. The material parts will thus be named material nodes for this case. Where one in the

future might be able to select an area within a single composite material on a millimeter scale, in this case, every material device is comparable to a single node for performing actions on. By bringing several devices close together, the intention is to simulate how this could be done with a single composite material consisting of multiple material nodes. For demonstrating this in practice, the proof of concept needs enough devices to show the concept of material programming in effect. This thesis considers 4 material nodes to be sufficient for a prototype.

5.4 Material part prototype design

This chapter has detailed the various needs and requirements connected to building a proof of concept for material programming. The core concepts are integral when designing the material devices and the material tool, and will heavily influence the actual design and implementation. The section detailing a minimal case has already introduced at possible components of a material node (material part). A microcontroller can be used for computation and communication, and is the only thing suitable for this use case in a small size. Connected to the microcontroller is an ultrasonic distance measuring sensor and an RGB LED. Using technology from internet of things and prototyping, the components can be connected together using a breadboard, which is a board for making an experimental model of an electric circuit. A microcontroller requires power supply, which can be connected using an external battery. Using these suggestions for a material node, a design was sketched for how it could look like. This illustration can be seen in figure 5.1. This illustration is for one single material parts, and a composite material would in a proof of concept consist of multiple of such material parts. What is not shown in this proposed design is a way to select and identify the material part

5.5 Composite material as a network of nodes

An introductory overview is better gained from looking at this proof of concept from a network perspective. By drawing a comparison to the material devices as nodes and mapping relationships (behavior) as edges, one can understand this proof of concept as a directed edge labeled graph. At initial start-up, every node in the network is a material device with no connection or edges to other nodes, as shown in figure 5.2. Once a mapping relationship (behavior) is programmed, a directed edge is drawn from input- to output device, labeled with the behavior variables. The maximum number of possible mapping relationships which can be made, is limited by how many times a node can be used as output. There are three possible and distinct outputs for every node; red-, green- and blue color. Only one mapping relationship can be made towards each of these outputs, and the mapping relationship will be overwritten if one output is given a new input.

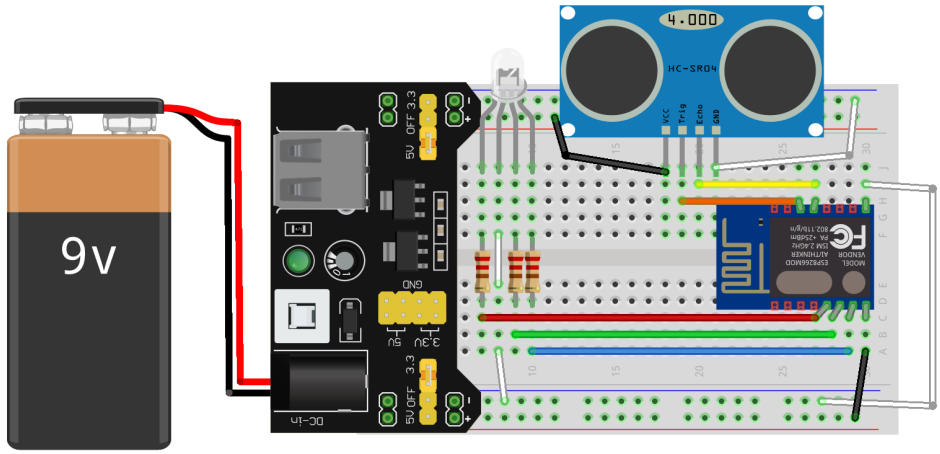


Figure 5.1: Illustration of the material node (without a selection component)

5.5.1 Mapping relationship as edges between nodes

When programming a mapping relationship between two material nodes, the user operates a material tool to declare input and output nodes, including configuring two behavior attributes. From a network perspective, a user merely creates a link between two material nodes, with the link consisting of two attributes. For now, it is not important to know what the attributes signify, only that two attributes are given to each mapping relationship. As mentioned in the previous section, the mapping relationship creates can be compared to an edge labeled graph.

In figure 5.3, three specific mapping relationships have been made. The collection of all the material devices combined simulate one composite material, and the state of the whole composite material can be described as a set of the input-output relationships made. In other words, the state of the whole composite material is a set of all the behavior relationships present between the material parts. There is no stored centralized state of all the material devices, all input-output behavior is stored and declared on the individual material devices in a distributed manner. Though there is no centralized state, reapplying a set of the mapped relationships to entirely new material devices will result in the exact same behavioral state. From a mathematical perspective, each mapped relationship can be defined as $relationship_i$ or shortened r_i:

$$r_i = \langle input_i, output_i, color_i, mode_i \rangle = \langle i_i, o_i, c_i, m_i \rangle$$

The state of the composite material is defined as the set of mapping relationships:

$$R = \Big\{ \langle i_i, o_i, c_i, m_i \rangle, \ldots, \langle i_n, o_n, c_n, m_n \rangle \Big\} = \Big\{ r_i, \ldots, r_n \Big\}$$

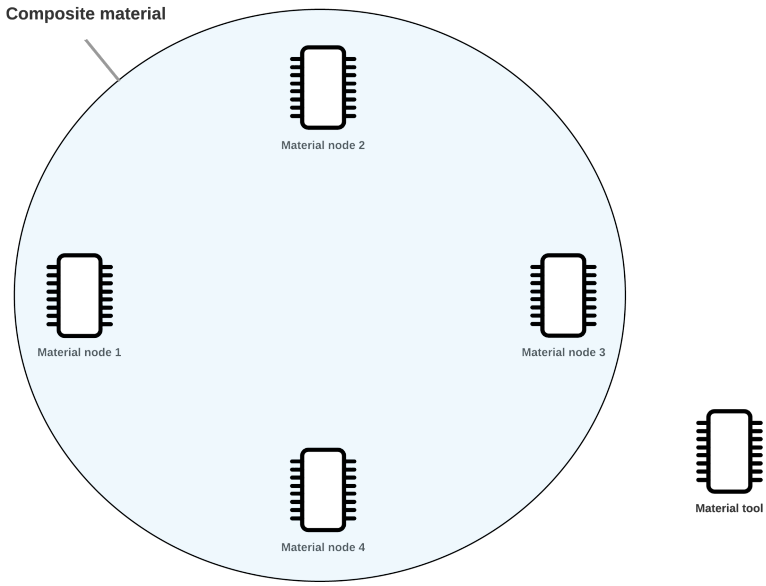


Figure 5.2: The initial network of nodes without any mapping relationships

Following this, the state and mapping relationships in figure 5.3 can be declared as:

$r_1 = \langle 1, 2, green, normal \rangle$
$r_2 = \langle 1, 4, blue, normal \rangle$
$r_3 = \langle 3, 2, red, inverse \rangle$

$R = \Big\{ r1, r2, r3 \Big\}$

$R = \Big\{ \langle 1, 2, green, normal \rangle, \langle 1, 4, blue, normal \rangle, \langle 3, 2, red, inverse \rangle \Big\}$

With r_1 in figure 5.3, a mapping relationship is made with material node 1 and material node 2, with attributes green and normal mode. The direction of the edge extends from the input device to the output device, meaning material node 1 is input and material node 2 is output. The two behavior attributes green and normal mode defines how input is exhibited on the output node. These attributes are only stored on the output node and does not affect how input device publish the distance data needed to create output. A material node can be used as input for multiple output nodes and conversely used as output for multiple input nodes.

In relationship r_2 in figure 5.3, material node 1 is also used as input for material node 4, meaning both material node 2 and material node 4 rely on input from material node 1. This mapping relationship is labeled with the attributes blue and normal mode, indicating

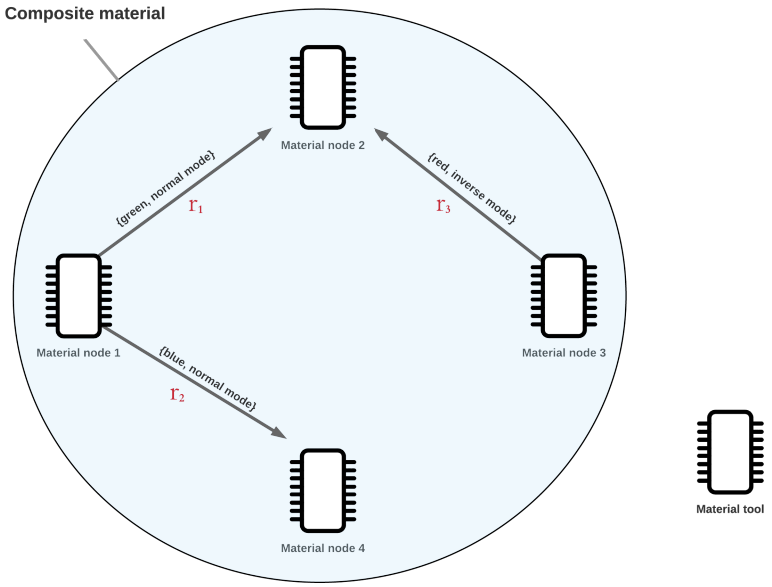


Figure 5.3: Graph of nodes with three mapping relationships: r_1, r_2, and r_3

it displays output in the same mode as material node 2, but with a different output color. In practice, this means material node 2 and 4 changes color brightness in green and blue light at the same time, based on continually changing distance data from material node 1.

Relationship r_3 shown in figure 5.3 is between material node 3 and material node 2, with the behavior attributes red and inverse mode. Given the same input, the inverse mode produces the exact opposite output of normal mode. Meaning, if normal mode produces a color of full brightness, inverse mode produces no light and vice versa. Following this, a published input exactly halfway to the maximum value will produce the same output in both normal and inverse mode. Material node 2 is output with colors green and red with normal mode and inverse mode respectively. Both of these outputs behave independently on the account of disparate input nodes, although the outputs still operate simultaneously with one another. Meaning a user can manipulate the distance on different input devices to blend output colors together, creating an assortment of different color hues.

5.6 Mapping relationship

A central aspect of this proof of concept is exploring the mapping relationship and behavior between material nodes. Every material node has a sensor and an actuator, which in this prototype is in the form of an ultrasonic distance sensor and an RGB LED respectively. These correspond to what the user can use as input and output. This limits the actions of what a mapping relationship can do, but opens up a number of possibilities

in experimenting and playing with different expressions, including how interaction affect multiple material nodes. By using a material tool, a user can create a distinct relationship mapping between an input on one material node and an output of another. An RGB LED has three disparate colors and outputs, which can be controlled independently and simultaneously to reach the desired effect. This means the colors red, green and blue can be used as outputs on one material node while being controlled by input on three different material node.

5.6.1 Input

Each material node has exactly one input in the form of an ultrasonic distance sensor. This device reads distance from the sensor to an object (typically a hand) at a given interval and returns the distance as an integer in centimeter. In simple terms, holding a hand right above the distance sensor reads an input value of 0 cm, and no objects within the maximum measurement range read an input value of 40 cm. The desire is to use a distance range where a user can sit in the same position and still use their hand to encompass the whole distance range. From a sitting position, 40 cm seems like an acceptable range for a user to raise their hand above the table. 40 cm of range gives the material node 40 distinct measuring points or inputs which can be used to control an output. So a distance measured from 0 cm to 40 cm is published as input, which is received on the output node and displayed as output color. One input device can be mapped to any number of output devices, meaning the input-output has a one-to-many relationship, also called a 1-N relationship where N represents any integer. The input isn't required to be mapped to any outputs, but have to possibility to be mapped to many.

5.6.2 Mapping modes

To provide some opportunity for exploration with colors and input-output interaction, each mapping can choose one of two modes of mapping. These modes are called normal mode and inverse mode. The modes do not affect how distance is read and published from input but does affect how input is displayed on the output. The distance is measured from 0 to 40 cm and subsequently published to output devices. The consequence of this input reading is a change in mapped color brightness based on the given input. Color brightness in this context is technically color illumination, but these terms are used interchangeably for this prototype.

Normal mode

The most sensible method of operation would be so that an output color increased brightness in parallel with a hand movement coming closer to the input node. In other words, the color brightness is turned off if nothing is within the 40 cm range, and increases brightness with a hand moving closer, producing full brightness when the hand is right above to the input material node. From a user's perspective, this thesis contemplates this to be the most

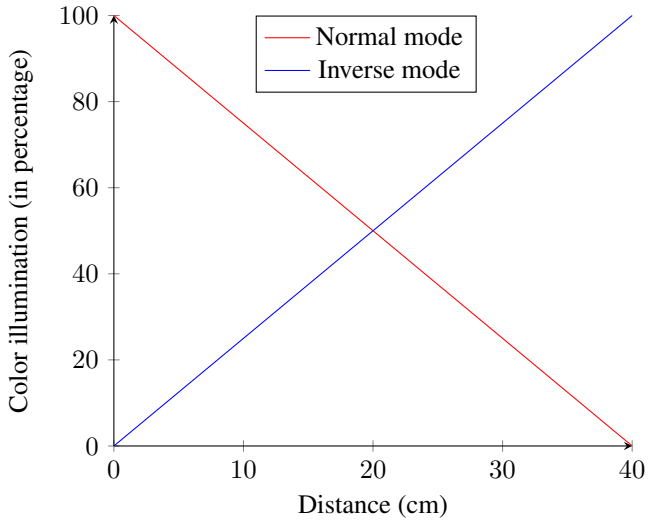


Figure 5.4: How input distance is affected by output mode

logical way to interpret the functionality of how distance input transforms color output. Consequently, this is the normal mode of mapping.

Inverse mode

Contrary to normal mode, the other mapping mode offered is inverse mode. As a differing method, this mode does the exact opposite of the normal mode. When nothing is within the 40 cm distance range, the color is on full brightness. Following this, the color brightness decreases at closer proximity, to turn off completely when the distance is 0. So moving a hand from 50 cm above the input node to right above it, display output as a fully luminous color slowly turning off.

5.6.3 Output

Each material node has three distinct outputs, in the way of colors red, green and blue, writable with color brightness from 0% to 100%. All material nodes are idle until programmed in a mapping relationship. Every mapping relationship has four variables which determine the behavior of the relationship; input device number, output device number, output color and output mode. When a mapping is established, input material node is instructed to start publishing data from the distance sensor and the output material node declares the parameters of the mapping, which includes color and mapping mode. When distance values are published from an input listened to, the output review which color(s)

and mode are mapped to this input and start adjusting the color(s) accordingly. Every output has a one-to-one or 1-1 relationship, meaning they can only be coupled to one distinct input. An output can even be mapped to an input on the same material node but does not have to be mapped to any output.

5.7 Mapping relationship in practice

Now for an illustration of how dynamic input-output interaction would look like in practice, using the mapping relationships in figure 5.3 to demonstrate physical interaction with the material nodes, in the format of a comic strip. To make this easier to understand, each comic panel has a pane explaining the input-output behavior state in the given situation. See figure 5.5 for a short explanation of this pane.

When a material node is the designated output in multiple mapping relationships, several colors will blend together to form new colors. This uses a standard RGB color model to be able to illuminate in most known colors.

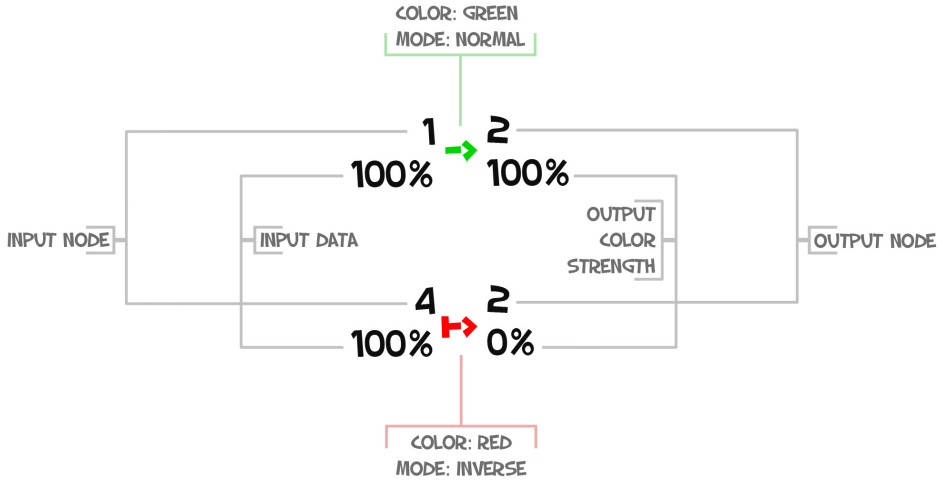


Figure 5.5: Explanation for the pane describing the input-output relationship of each interaction

5.7.1 Interaction scenario 1

Starting of with an easy example illustrating relationship r_1 and r_2 using material node 1, 2 and 4. This interaction is shown in figure 5.6. There are two relationships in this interaction, both with material node 1 as input and normal output mode. This means material node 2 and 4 will respond to input with the same luminous output, but with colors green and blue respectively. As seen in panel 1, no distance input from material node 1, equally results in no output on the output nodes. This is the defining behavior

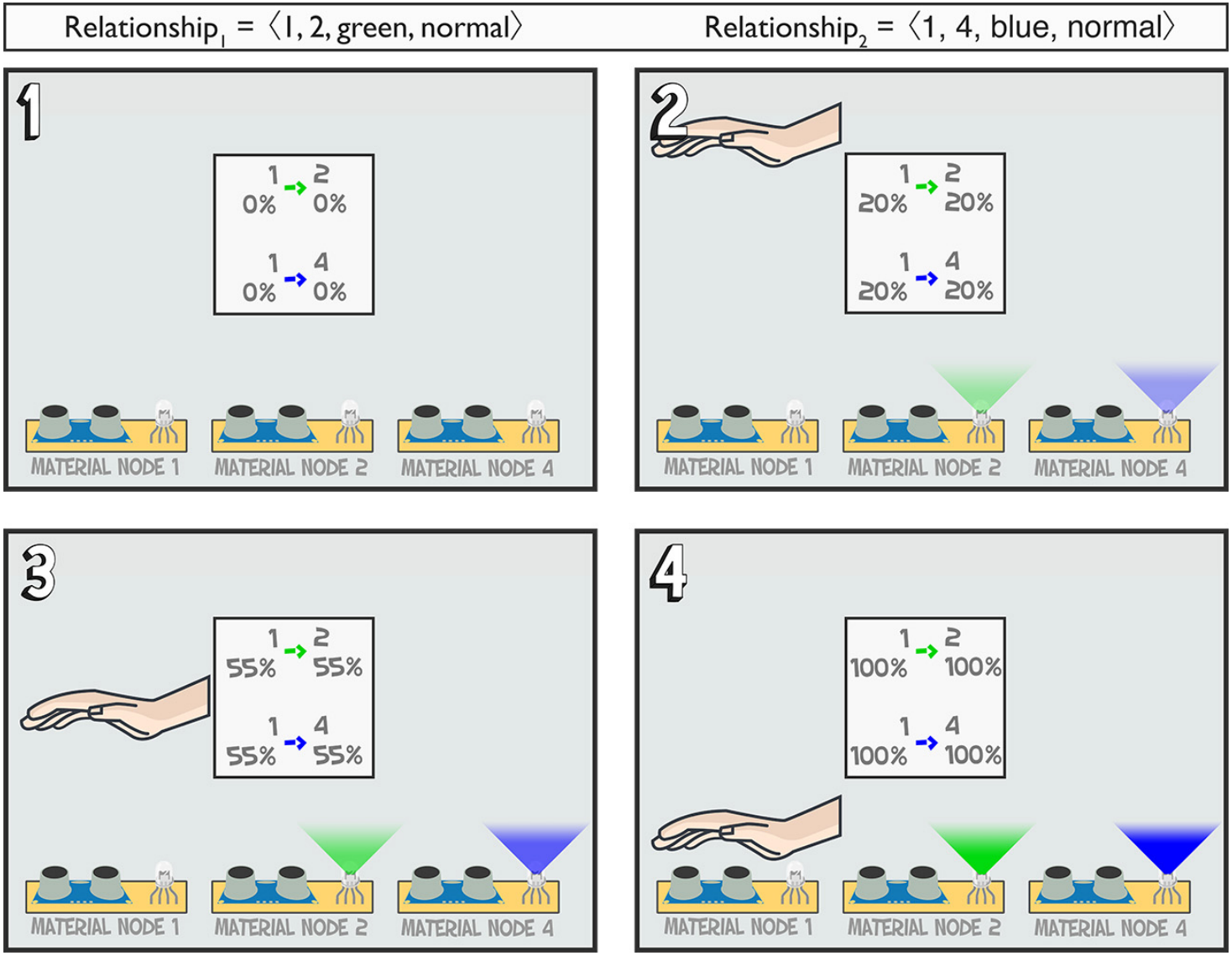


Figure 5.6: Demonstration of physical interaction with mapping relationship r_1 and r_2 in figure 5.3, using material node 1, 2, and 4

for normal mode - increasing output color brightness as the distance decreases, arriving at full illumination as the distance comes close to zero. When the user starts moving their hand within range of the input node's distance sensor, the output nodes respond with a slightly visible color, as shown in panel 2. Following this in panel 3, the user moves their hand closer to the input node, and the ensuing behavior in the output nodes is an increased color brightness. Shown in panel 4, the hand is right on top of the input device, resulting in full color illumination on the output nodes. This relationship interaction is concurrent and simultaneous, meaning a change in input distance result in a real-time immediate output reaction. In this example, the two relationships have different output nodes, but these relationships could have easily been to one output node, with two different output colors increasing in brightness simultaneously on the same node. This was an interaction demonstration using one input node and two output nodes with normal mode. The next interaction scenario will demonstrate how interaction progress when one node is the output in two mapping relationships with different modes.

Figure 5.7: Multiple output colors blending together to form a new color. The boxes and their numbering correlate to the panels and the interaction in figure 5.8

5.7.2 Interaction scenario 2

This example is based on mapping relationship r_1 and r_3 created in figure 5.3, using material node 1, 2 and 3. The previous example displayed interaction when one material node was input in two mapping relationships with normal mode. The goal of this example is to demonstrate interaction when one material node is output to two different input nodes, and how inverse mode affects output. Before beginning, here is a summary of some important information:

- An input node does not store the behavior attributes of the relationship, meaning the input node is not aware which color or mode is used to display its input distance data on the output node
- As a consequence, the input node publish distance measuring data in the same way irrespective of output node's color and mode attributes
- Following received input distance data on an output node, it modifies the data with applicable behavior attributes (output mode and color)
- Multiple output colors on the same node will be displayed simultaneously, resulting in one distinct output color comprised of multiple colors mixed together
  - E.g. Full strength red (100%) and full strength green (100%) on one output illuminate as yellow hue
  - E.g. Full strength red (100%) and half strength green (50%) on one output illuminate an orange hue
- Given the same input, inverse mode output the exact opposite of normal mode. In other words, the sum of output color brightness on normal and inverse mode given the same input is always 100%
- When input measure minimum distance (panel 1, figure 5.8):
  - Normal mode output full strength color (100% illumination)
  - Inverse mode output no color (0% illumination)
- When input measure maximum distance (panel 7, figure 5.8):
  - Normal mode output no color (0% illumination)
  - Inverse mode output full strength color (100% illumination)
- When input measure half of maximum distance (panel 10, figure 5.8):
  - Normal mode output half strength color (50% illumination)
  - Inverse mode output half strength color (50% illumination)

To illustrate the mode attribute's effect on output, look at panel 1 and 7 in figure 5.8. In panel 1, both input nodes measure minimum distances, resulting in a full illuminated green color with normal mode and an absent red color with inverse mode. Conversely, the output in panel 7 is reversed, where both input nodes measure maximum distance, shown

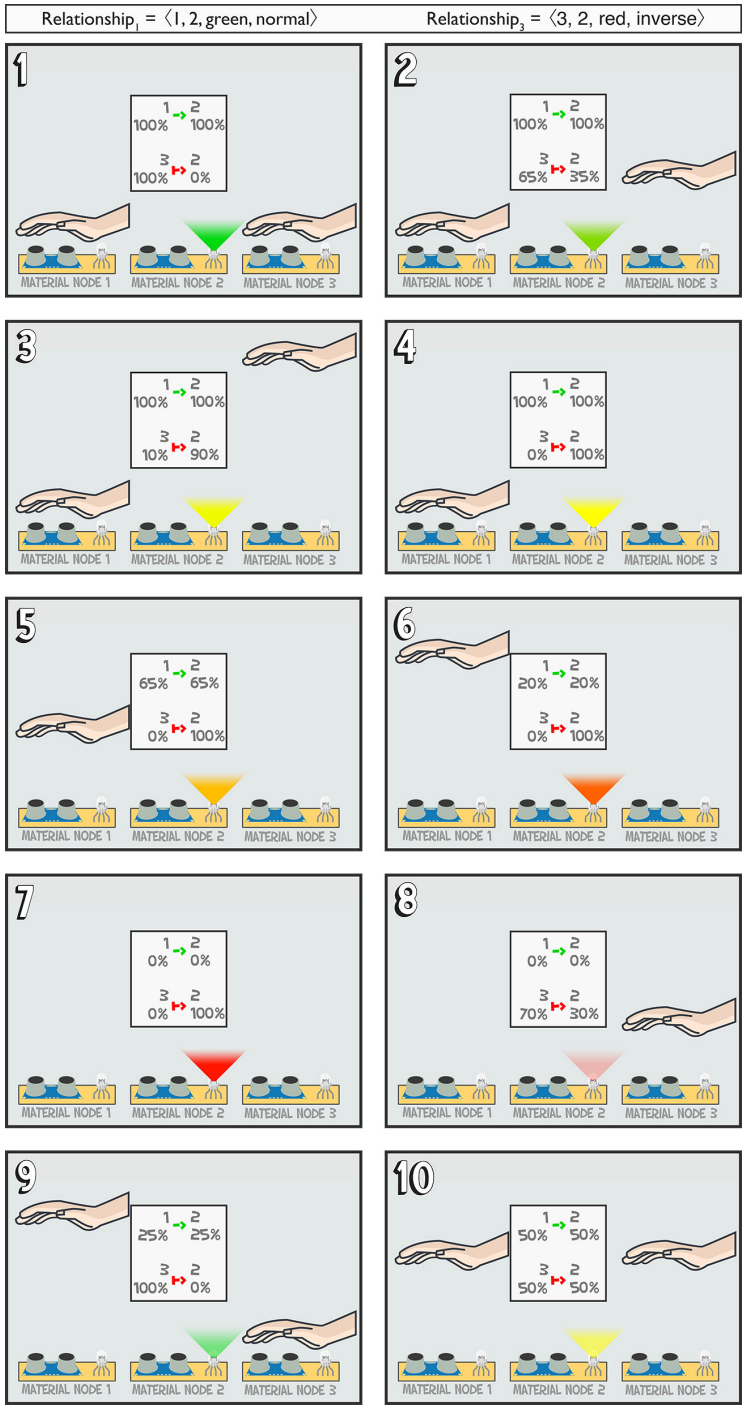


Figure 5.8: Demonstration of interaction with mapping relationship r_1 and r_3 in figure 5.3 using material node 1, 2 and 3

with a full illuminated red color and absent green color. With this functionality, color red will always illuminate to some extent unless input node 3 measure minimum distance, meaning the distance is 0 cm. Each of these panels exhibits interaction which display one or two colors blended together. An illustrative image is shown in figure 5.7 displaying how the blending of the colors affect the output color in this interaction scenario. The image is merely meant to illustrate how blending multiple colors create one new color, and the numbering corresponds to each panel in figure 5.8.

Following panel 1 where green is at full strength and red absent, panel 2 reveal a slightly more yellow-green color, as the red color is more illuminated. By maintaining hand position right above input node 1 which is linked to green output color and slowly moving the other hand upward, the green color will maintain full illumination while the red color gradually gains illumination in parallel with increased input distance on input node 3. Starting movement in panel 2, a slightly more yellow-green color is displayed, becoming more discernible in panel 3, and eventually displaying an obvious yellow color in panel 4 as a result of fully illuminated green and red output colors. To recap, the observed effect over the course of this hand movement in panel 1 to 4, is the green color blending with the increasingly visible red color, coming close to a yellow color as the two colors illuminate on full strength. Mimicking this movement with the other hand, panel 5 to 7 demonstrate the green color's slowly decreasing brightness, as the output color's hue shift from yellow to a solid red. Continuing this experiment, panel 8 shows the total absence of green while color red is only slightly visible. The complete opposite output effect of this is shown in panel 9, where the red color is absent and the green color is slightly visible. When both input nodes measure distance halfway to maximum, the output will be the same strength regardless of mode. This is illustrated in panel 10, where both green and red is halfway illuminated, resulting in a halfway illuminated yellow color.

5.8 Tool device design

This thesis intends to develop a material tool exhibiting functionality similar to that of the select- and color tool in the initial material programming paper [26]. For selection, the performance of various wireless and wired sensors was explored. Most relevant for this functionality are infrared, RFID, photocell, and Wi-Fi. The tool is will be the user's main method of controlling the composite material. As the embedded functionality in the materials might not be inherently obvious, one of the purposes of the tool is to let the user explore the dynamic properties of the material. Ideally, a user can pick up the tool and without prior knowledge or instructions, examine the dynamic attributes of the material and learn how to control its behavior.

5.8.1 Selection

During testing, infrared proved to be a slow and ineffective way of wireless communication, with a high error rate. It provided good functionality for identifying and addressing

specific devices, but the sensor's disadvantages excluded infrared as a technology for selecting material devices. A photocell sensor can be used as a way of selecting a device by moving a hand over the device and measuring the amount of light received in the sensor. It provides very fast data to the device connected, but when tested, the data provided was inconsistent, with different values depending on distinct environmental conditions. The photocell sensor also didn't provide the tool with any identifying information as to which device is selected, and this technology for selecting material devices was deemed to be counter-intuitive and too unreliable. Wi-Fi is integral to the communication between various devices and tools. This thesis does not regard it as a tool for selection, but rather an engine and prerequisite for other technologies and sensors to perform their work. Radio-frequency Identification (RFID) has a large area of application, with modern use cases within contactless payment system and digital ticket identification. With a mechanism analogous to a digital barcode, it uses near-field communication (NFC), where the two electronic devices are brought in close proximity to read or write data. The sensor demonstrated very fast and reliable identification of RFID tags, with the only major drawback being the close proximity it has to be brought together. The technology has the added capability of allowing the reading sensor to read passive RFID tag, and as such only the reading device needs constant electricity, providing the convenience of versatility and portability. Although in some situation this could prove disadvantageous, as the passive component has no way of perceiving or communicate it is being read. To counter this, the material tool can use an RFID sensor to identify material devices and communicating their selection to the material devices using Wi-Fi. That means every material device are given an RFID tag which contains information about which material device it belongs to. When RFID is used to read a material node's RFID tag, the tool will know instantaneously which material node it belongs to. RFID met all the requirements for selection and became this prototype's primary technology for identification.

5.8.2 Color

The initial color tool envisioned [26] is an independent device with a touch slider, used for dragging a finger along to change color. This prototype imagines all the tools as one combined device. The idea is for a user to have all the functionality used for material programming in one device. In testing different technologies relevant to IoT, this thesis has explored possible sensors and actuators for this use, such as a potentiometer, LED, buttons. The idea of a color tool is to real-time adjust the color of the selected material area. A logical actuator for this color change would be an RGB LED. This LED is capable of changing the three colors (red, green, blue) independently of each other, providing the opportunity of three distinct outputs as well as any combination of these. Considering this, there arises a question of how to program this coloring, both from an end-user perspective and from a practical perspective. The previous color-changing case approach challenges this with a touch slider color tool that responds to physical contact. The slider changes color depending on the placement of the finger, correlating the specific placement to a distinct spot in the color range. With this mechanism, it appears as if the touch slider approximately mirrors the visible light color spectrum, ranging from violet to red. This is a logical solution for changing color using only one input value. The downside to this approach is that

there is no way to adjust the light intensity or to have different colors interact and affect each other. With RGB LED actuators, a material device has the opportunity to implement color change as a result of three different inputs. This provides an end user with the ability to alter color hue and intensity based on disparate factors and inputs. As such, this proof of concept wishes to enable three distinct inputs to demonstrate mapping, by using colors changing based on user inputs, as an output in a behavior relationship. Another question arises as to how the end-user manages to program these color changes and map them to inputs. Seeing as how the RBG LED provide three distinct outputs, it is relevant to give the end user an option to select each color for mapping. With a button, a user can cycle the three colors displayed on an RGB LED, and determine mapping relationship output color based on the current illuminated color.

5.8.3 Material tool prototype design

A design for a material part is proposed and using this as a guideline, the material tool should be built to be able to program these. The core concepts detail a requirement for programming the behavior of material nodes, as well as selecting and identifying them. The selecting part was in mentioned in the section above, and RFID was proposed as the most suitable technology in selecting and identifying a material part. Each material part are given an RFID tag which can be identified by the material tool using an RFID reader. With this part covered, the material tool also needs functionality to program behavior of the material nodes. Mapping relationships has been described with two interaction scenarios using an input node and output node, with a behavior relationship given by a color and a color mode. So the material tool needs functionality to choose; input node, output node, output color and output mode. These four attributes are also what defines any mapping relationship. The previous section discusses how to design a material tool able to program color on material parts. Buttons are suggested as a method of changing colors when creating a mapping relationship. Buttons is an easy way to signify functionality is performed when pressing it. With four attributes in every mapping relationship, this thesis proposes using one button as functionality for each of these attributes. So one button to choose input device, one to choose output device, one to choose the color and one to choose the color mode. An RGB LED can be used to inform the user of chosen color and the state of the material tool. Handling all of these components need computation and communication, and for this, the prototype will use a microcontroller. A proposed design illustration is displayed in figure 5.9.

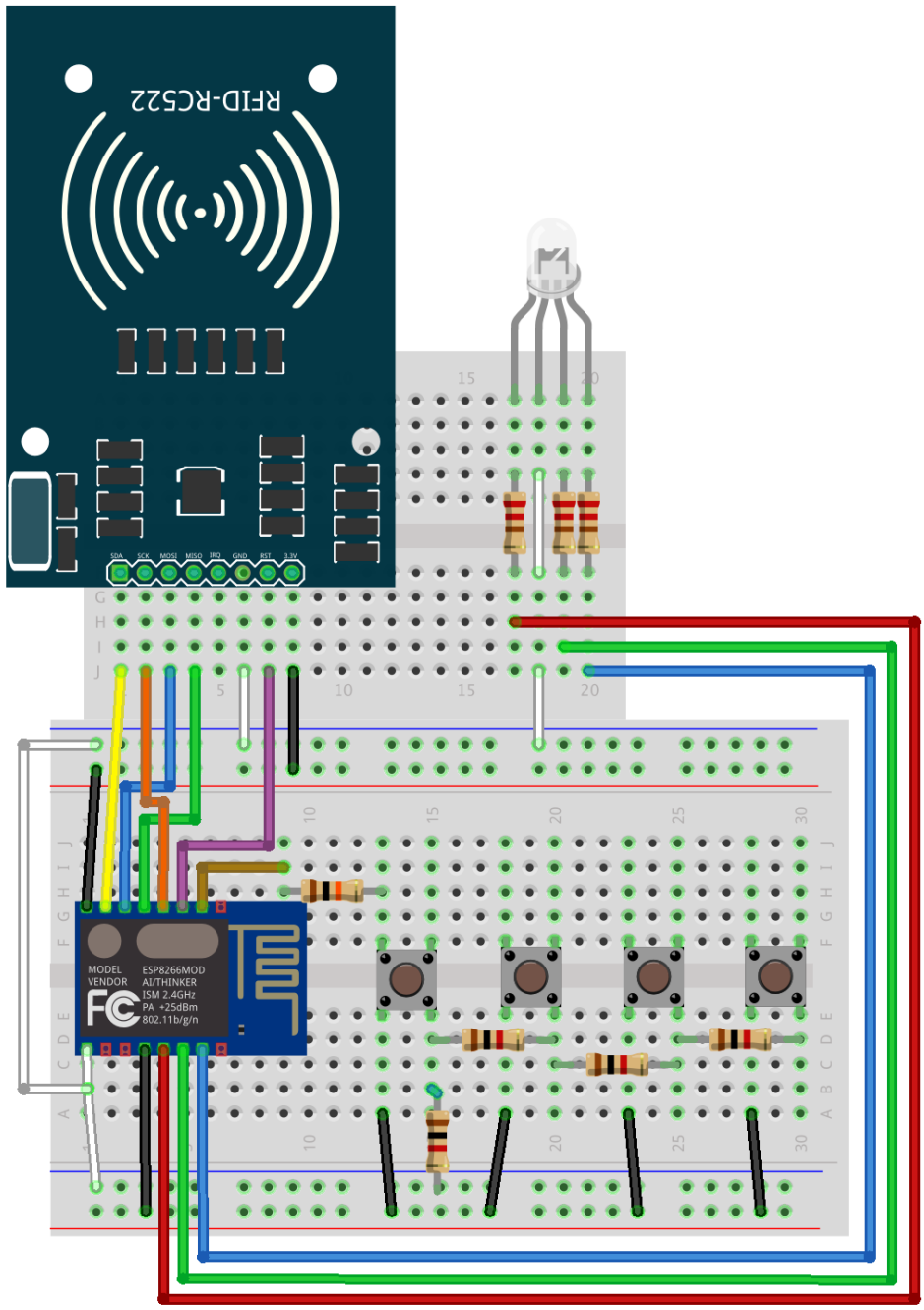


Figure 5.9: Illustration of the material tool

5.9 Communication

Communication might be the most integral part of the whole proof of concept. Almost every aspect of the material nodes rely on efficient and stable communication to perform; input publishing and receiving, creation and removing mapping relationships and interaction between mapped material nodes.

5.9.1 MQTT

MQTT is a protocol built on top of the TCP/IP layer, with machine-to-machine communication and IoT transmission specifically in mind. Core to the protocol's principles is lightweight messaging with a small code footprint, minor power usage, and efficient distribution of information to one or many receivers [13]. It uses a publish-subscribe pattern, meaning messages are not sent to specific receivers, but rather published to a "topic" which any device can subscribe to [3]. This enables many subscribers to receive a message without involving the publisher of the message, resulting in greater network scalability and versatility. Loose coupling between publishers and subscribers make it effortless to create mappings and equivalently easy to discontinue the relationship. The main functionality of the communication protocol in this prototype is to enable the creation and maintenance mapping relationships between material nodes, in addition to publishing input and subscribing to topics based on mappings. The MQTT protocol is similar to a representational state transfer (REST) application programming interfaces (API), in the sense that all needed data is contained in a single transmission and it maintains no centralized state.

The protocol has two defined roles in a network, that of broker and client. A client is any device that subscribes to topics or publishes messages, essentially every material node in this case is its own client. A single client can both receive and publish data, and every client subscribes to input and publishes to output. As the client publishes messages to a topic it does not necessarily have known about the recipients, and as such, it needs an intermediary to forward its messages to the current subscribers. The messaging broker is an intermediary which receives all published messages and forwards them to relevant subscribers of the destination topic. It maintains a list of topics with all devices subscribed to it and routes incoming messages according to the current list. In some ways, the broker acts as a server to the clients, providing requested data at the leisure of the publisher. The broker has a one-to-many relationship and for this proof of concept, only one broker is needed. However, the clients have some requirements for connecting to the broker, primarily a LAN- or internet connection and a hostname or IP address of the broker. More on how MQTT is used in this prototype in chapter 6.

5.9.2 Architecture

Approaching architecture with IoT technology is very interesting, especially when dealing with multiple independently addressable devices.

There exist many providers offering platforms specifically designed for IoT, with specialized systems such as IoT platform as a service (PaaS). These platforms often use a centralized architecture in which a central hub provides back-end services to numerous IoT devices. In an enterprise environment, IoT devices are typically used for data collection from a large number of simple devices with a sensor. The IoT devices measure various conditions and publish the data to the platform, which handles necessary computation and data analysis. The advantage of this is controlling all the connected devices from one central place.

In a distributed and decentralized system, tasks are distributed among the nodes in the network, and the nodes coordinate and communicate their behavior with each other in order to achieve a united goal. This means the IoT devices in the network can interact without a central authority. Typically this means devices can communicate directly with each other, without the interference of another party. This enables autonomous execution of functionality and peer-to-peer communication.

This proof of concept utilizes architecture principles from distributed and decentralized systems, as shown in figure 5.10. The material nodes communicate and coordinate actions with the appropriate nodes in a mapping relationship and distribute the work among each other. An input node is told when to start publishing data and an output node subscribes to this publishing, essentially handing over control to the input node. However, for a node to be able to communicate with other nodes, it requires a connection to these nodes. Following this, the communication is not directly with each other, the material devices utilize an MQTT broker which subsequently routes the message to its recipient. As such, the network and messaging broker simulate direct communication between the nodes. These connections are shown in figure 5.10 as grey bidirectional links, while the mapping relationships between the nodes are shown in green unidirectional links. There are significant uses of distributed principles, wherein that only the material nodes themselves store information about mapping relationships. This information is not stored anywhere else, and as such it is up to the nodes themselves to coordinate input-output and express when a mapping relationship should be removed.

Even though there is no centralized storage or control, a network of material nodes with mapping relationships can easily be replicated to create a new identical network, using the set of mapping relationships. The set of mapping relationships is not stored anywhere, but if a service were to listen to all published messages, it could duplicate this behavior using a digital twin, which is essentially a digital representation of the information. By listening to the MQTT broker, the mapping relationships and interaction can be shown in a digital representation. This opens up a number of opportunities.

For example, one set of material nodes with material tool are located in Norway. A user programs behavior and start interacting with the nodes. Simultaneously, another set of nodes are located in India and are connected to the same MQTT broker. The set of nodes in India would respond to interaction with the nodes in Norway in real-time. Conversely, the user in India could change mapping relationship of the nodes, and the nodes in Norway would be affected by this change. While the coordination and communication are primarily distributed, the architecture and communication of the material nodes enable digital

twins to be monitored and physical twins to be replicated anywhere in the world, with interaction behavior in real-time.

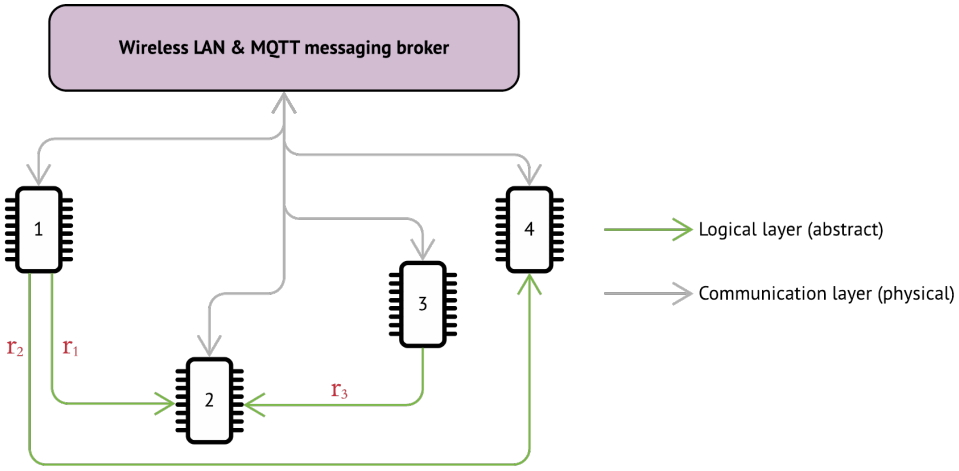


Figure 5.10: Architecture of the network of nodes, using the mapping relationships in figure 5.3

Chapter 6

Proof of concept: Implementation

6.1 Communication

Communication is at the cornerstone of every functionality in this proof of concept. Coordinating creation and removal of mapping relationships, as well as input-output interaction, is an essential part of this case for everything to work. As described in the previous chapter, the chosen protocol of communication is MQTT, which uses a publisher-subscriber pattern. This section will elaborate on the communication process in this proof of concept; how this protocol is implemented and utilized, as well as provide technical insight to how the communication protocol enables coordination in the distributed network of material nodes.

6.1.1 MQTT Topics

Using a publish-subscribe pattern for the communication requires the prototype to organize transmission messages in topics. A topic is essentially a channel, which any MQTT client (material node) can subscribe to. So when a material node publishes a message to a topic, all material nodes subscribed to this specific topic receive the message.

Every material node is an MQTT client and needs to publish messages and subscribe to topics according to real-time interaction in a mapping relationship. This creates a need for being able to address and identify every material node individually in the communication network. In a simple manner, each device is given an integer ID starting from 1 and incremented by one for every device. Topics are organized and named in such a way that every material node recognize the attributes of a mapping relationship; when it is being

used as output, when to start publishing data from the input, when to stop listening to an input, and when to stop publishing input data. Topics are alphanumeric strings and include the material node ID, a number represented as ”ID” in this section when describing any given material node. From powering on, every device subscribes to four topics related to mapping, and uses one other topic for publishing input data. When creating a mapping relationship, a user makes four distinct choices; input node, color, mode, and output node. Each of these variables can be represented and transmitted in condensed form; input and output devices have IDs in the format of one integer digit, inverse mode is either true or false (’t’ or ’f’), and color can be shortened to character ’r’, ’g’, or ’b’. This enables a material device to compress substantial amounts of information concisely in a short message.

An input in a mapped relationship does not publish distance data all the time, as this might cause unnecessary network congestion. Instead, when a mapping relationship is made, the input device is told to start publishing distance and output device is told to start listening. Before explaining the naming and organization of topic and transmission, it is beneficial to understand how the material devices communicate. The material tool handles a majority of the communication, particularly that related to mapping relationships. A generic user operating scenario is as such; a user uses the tool to select (the RFID tag of) an input device, chooses color and mode on the tool, and lastly select (the RFID tag of) the output device with the tool. Immediately following this, the material tool publishes MQTT messages to create the mapping relationship and subsequently relinquish all further input and output communication to the material nodes in the relationship. More explicitly, the material tool instructs the input device to start publishing distance data to its distance topic, and further instruct the output device to start listening to the input device’s distance topic and use this as input with the given color and mode. If removing this mapping relationship is desired, a user chooses the color to stop outputting and select the output device - upon which the tool tells the output to stop outputting the given color. At this point, the output device tells the input device it has one less output to worry about, at which the input stop publishing if it has no other outputs. This might be a lot to follow in pure textual form, but not to worry, this will be explained in a more digestible way using tables and diagrams. This is a list of every topic subscribed to by every material node. ”ID” in every topic name would be replaced by the integer ID of a material device.

- ID-distance
  - Every device publish their distance measure data to this topic
  - Only publishes distance when a mapping is made and the input device is told to start publishing
- ID-output
  - The material node which is made the output of a mapping receives a message here informing it of input device, output color, and output mode
  - For example: input device 3 is mapped to output device 1 with color red and normal mode

  - Output device 1 receives a message on topic ”1-output” with content ”3-rf”
  - This tells the output device to listen to topic ”3-distance” and use this data to produce an output with color red (’r’) and normal mode (inverse mode false is ’f’)

- ID-input
  - The device which is made the input device in a mapping receives information on this topic to start publishing data to topic ”ID-distance”
  - For example: input device 3 is mapped to output device 1 with color red and normal mode
  - Input device 3 receives a message on topic ”3-input” with no content (nil)
  - This tells the input to increment an input counter variable and start publishing data to topic ”3-distance”

- ID-unoutput
  - When removing a mapping relationship, the output device in the mapping receives message from the tool here
  - The output device receives a message instructing which color to stop outputting, and subsequently tells input device it has one less output
  - For example: remove mapping from input device 3 to output device 1 with color red and normal mode
  - Output device 1 receives a message on topic ”1-unoutput” with content ”r”
  - Output device stop listening to input linked to color red, after which it publishes a message to topic ”3-uninput” with no content (nil), instructing the input device to decrement its input counter by one

- ID-uninput
  - When removing a mapping, the output device tells the input device it has stopped subscribing to distance data
  - The input device decrement its input counter by one, and if this was the only input mapping for this device, it stops publishing data to ”ID-distance”
  - For example: input device 3 unlink from output device 1 with color red and normal mode
  - Input device 3 receives message on topic ”3-uninput” with no content (nil)
  - If the input counter is now 0, stop publishing distance data

These are the four topics subscribed to by every material node, which are used for creating and removing mapping relationships. Additionally listed is the ”ID-distance” topic used by every material node to publish distance data when its an input node in a mapping relationship.

6.1.2 MQTT communication example

Table 6.1 shows an example of communication between two material nodes when creating and removing a mapping relationship, with transmissions listed chronologically using a table. This is mostly for introductory illustration, and more specific examples using sequence diagrams will follow in the next section.

Topic	Message	Subscriber	Publisher	Reaction/Effect
Tool creates a mapping from input node 1 to output node 3 with color red and normal mode				
3-input	nil	Node 3	Tool	Node 3: Increment input counter by one Start publishing to topic [3-distance]
1-output	3-rf	Node 1	Tool	Node 1: Subscribe to topic [3-distance] Link input to color red with normal mode
Mapping is created and transmissions are now based on interaction with the input node				
3-distance	5	Node 1	Node 3	Node 1: Illuminate color red 5%
3-distance	36	Node 1	Node 3	Node 1: Illuminate color red 36%
3-distance	68	Node 1	Node 3	Node 1: Illuminate color red 68%
3-distance	83	Node 1	Node 3	Node 1: Illuminate color red 83%
Tool removes mapping relationship with color red on output node 1				
1-unoutput	r	Node 1	Tool	Node 1: Unsubscribe to topic [3-distance] Unlink red, publish to topic [3-uninput]
3-uninput	nil	Node 3	Node 1	Node 3: Decrement input counter by one Stop publishing to topic [3-distance]

Table 6.1: Example communication when creating and removing a mapping relationship

6.1.3 MQTT broker

Using the Mosquitto MQTT library, any operating system or platform can run an MQTT broker and make the broker available through its public IP address. The MQTT clients (material nodes) requires an IP address of the MQTT broker coded into the device. The broker's IP address is contingent on the platform or server running Mosquitto. If the broker is running from a personal computer, this IP address is subject to possible change every time the computer reconnects to a network (wireless or otherwise). In this situation, every device would have to be reprogrammed continuously as the IP address change on the MQTT broker. But before the clients can start to think about connecting to the broker, they need to establish an internet connection. Without this, there is no communication link to connect through. Every microcontroller on a material device is fitted with a WiFi module, but here arise the same predicament as with MQTT clients connecting to an MQTT broker; a demand for ever-changing variables hardcoded into each device, namely the Wi-Fi name and password. Another delicate consideration is the latency from when a message is published to the broker, to when the message arrives at a receiving device. The intention of input interaction changing output is to simulate real-time physical change based on a number of dynamic factors combined, and updated multiple times per second. The goal was

never to make a proof of concept ready for consumer-market, but it was a desire to make all parts of this case ready for use out of the box. Meaning if this proof of concept were to be given to a test user, the devices were ready and did not require to be programmed again. An alternative solution to this issue of an MQTT broker's changing IP address, would be hosting a broker on a server with a static IP address. This solves the issue of a fixed IP address, but does not solve the obstacle of changing wireless network parameters or consider the latency effect of using a server somewhere on the internet. Another idea emerged, which included constructing a designated Local Area Network (LAN) and MQTT broker, with the sole purpose of handling communication between material devices.

Raspberry Pi

A Raspberry Pi 3 Model B was configured to work as an access point and wireless LAN, essentially providing material devices a Wi-Fi router with a static IP address to use as an MQTT broker. To connect material devices dynamically as they use the network, the Raspberry Pi was also configured as a DHCP server. Dynamic Host Configuration Protocol (DHCP) is a protocol which enables a server to automatically provide and assign IP addresses to connected devices. This made connecting to a persistent wireless network simple with fixed router name and password. If the Raspberry Pi is connected to an existing network via Ethernet cable, it also enables all general internet access for all connected devices, including World Wide Web and remote access control through SSH. This makes it easy to observe and examine all MQTT communication remotely. Although this is great, a connection to an existing network is in no way necessary for everything else to work. Even without internet access, all devices connected to the Raspberry Pi still have full communication capabilities by using the wireless LAN.

In short, material devices connect to a Raspberry Pi used as a router and wireless LAN, which run an MQTT broker on a static IP address. The configured Raspberry Pi is not an integral aspect of material programming or this proof of concept, but it does facilitate the communication needed to demonstrate the case. With the configured router complete, setting up a functioning communication network becomes a matter of plug and play, going a long way to streamline further development work.

6.2 Material node internal state

As mentioned in section 5.9.2 describing the architecture, the distributed system architecture makes it so that there is no central state, and each node store the mapping relationship data appropriate for the current situation. To accomplish coordination across material nodes without a central point of control, it is imperative that each node maintains an up-to-date state. This means each material node is required to store the data needed for its own mapping relationships.

The pattern of communication on each device's subscribed MQTT topics, use a uniform structure across all material nodes. This also applies when maintaining an internal state.

Using these principles, every node must store data and mapping relationships in the same way, or from a programming perspective; in equivalent variables. These are the variables relevant to mapping relationships stored in the state of each material node:

- Device ID: Unique ID type
  - A single integer identifying the material node
  - Identical to the data of an RFID tag laying next to the material node
- Input counter: Integer
  - Integer counter of the number of current relationships where this node is an input
  - Conditions the node to publish distance data when the variable is greater than 0
- Red/green/blue input node ID: Unique ID type (x 3)
  - Three separate integer variables
  - Store the material node ID of the input node for each output color
  - Declared 0 unless in an active mapping relationship
- Red/green/blue output mode: Boolean (x 3)
  - Three separate boolean variables
  - Normal mode = false and inverse mode = true
  - Store the output mode linked to each output color
  - Declared 'false' unless in an active mapping relationship with inverse mode

6.3 Communication scenarios

With information about the material node's internal state and communication structure in mind, this section will look more closely at how communication function in specific scenarios. Used to demonstrate this are four actors; material tool, material node 1, material node 2 and MQTT broker. MQTT topics are denoted with brackets followed by the message, using the pattern [topic]: message. Communication is displayed in a sequence diagram, with MQTT transmissions in black and relevant internal functionality in grey.

6.3.1 Powering on material nodes

After powering on a material node, the device will connect to the LAN and MQTT broker on the Raspberry Pi. As shown in figure 6.1, following connection with the MQTT broker,

a material node will subscribe to the node's four topics used to manage mapping relationships. Depending on the material node integer identification, these four topic are ID-input, ID-output, ID-uninput and ID-unoutput.

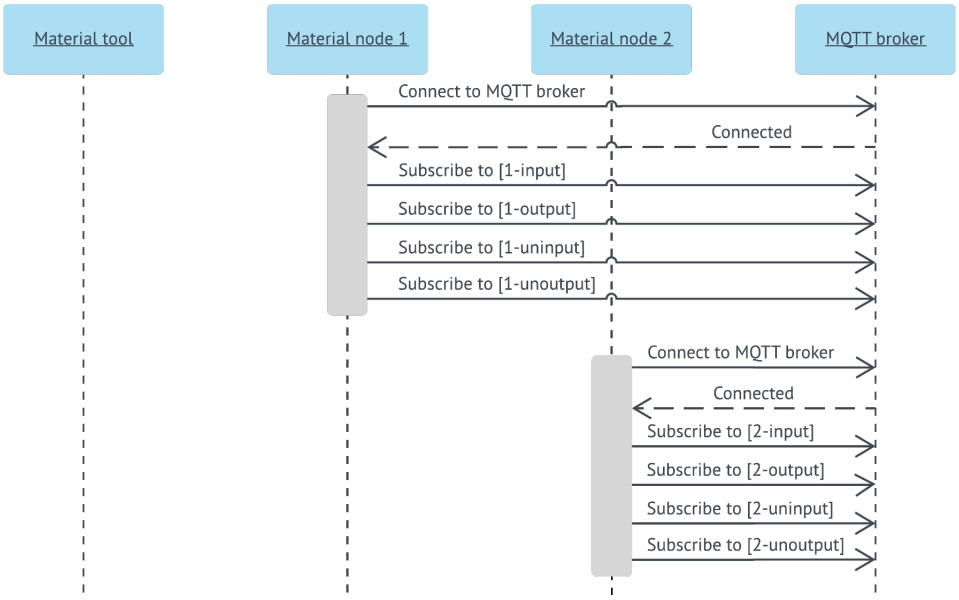


Figure 6.1: MQTT communication after powering on material node 1 and 2

6.3.2 Create mapping relationship

In this scenario, a user operates the material tool to create a mapping relationship with material node 1 as input and material node 2 as output. The behavior attributes given in this mapping are color green and inverse mode. In a mathematical formulation, this relationship is defined as $r = \langle 1, 2, green, inverse \rangle$

This scenario is shown in figure 6.2. The tool starts by telling material node 1 it is now an input, which prompts it to start publishing distance data to the node's designated input topic, [1-distance]. Following this, the material tool messages the output node about its role in the mapping. The output node receives message "1-gt" on topic [2-output], and initiates a procedure for mapping the device as an output node. The number leading up to the hyphen indicates the ID of the material node used as input. In this case, the ID of the input node is "1", leading to the output node subscribing to the topic [1-distance]. The two characters after the hyphen are the relationship attributes, and affect how input data will be altered before displayed as output. The first character is one of three alternatives; 'r' for red, 'g' for green or 'b' for blue. The second character has two alternatives, working as a boolean for the output mode; 't' if inverse mode is true or 'f' if inverse mode is false (implying normal mode is true). In other words, when material node 2 receives message

[2-output]: 1-gt, this is interpreted as ”Use material node 1 as input, subscribe to input node’s distance topic [1-distance] and display this data as output using color green and inverse mode”. Now the output node has established a link to the input node and will use its distance data as input to display colored light as output until told otherwise. After a mapping relationship is created, all communication going forward will be exclusively made from input node to output node.

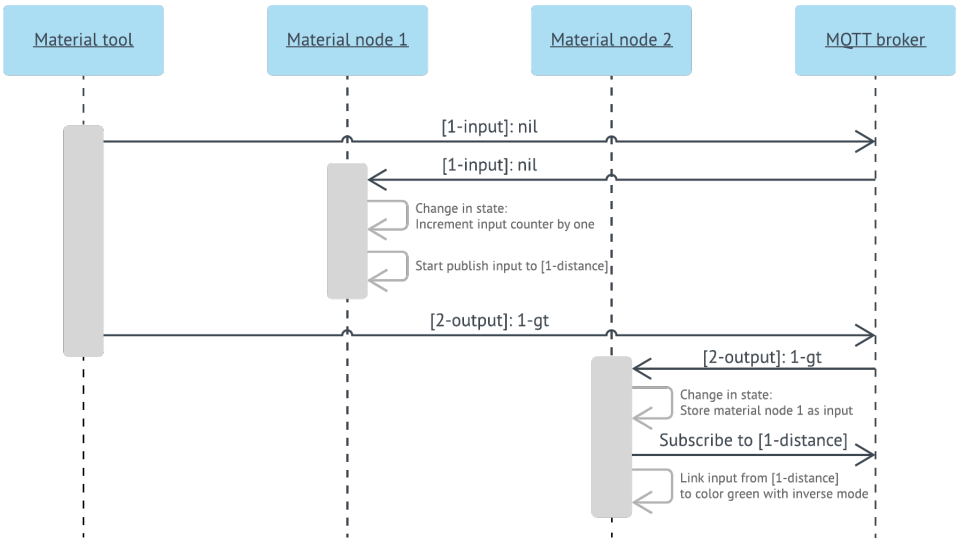


Figure 6.2: MQTT communication when creating mapping from input node 1 to output node 2, with color green and inverse mode

6.3.3 Communication in a mapped relationship

At this point, the mapping relationship is created and communication continues in an input-output pattern, detailed in figure 6.3. The input node uses its ultrasonic sensor to constantly measure the distance and publish the distance data to topic [1-distance] whenever the distance changes. The distance measurement interval in this proof of concept is set to every 200 milliseconds; meaning the distance is read and published on the input node, and received and displayed on the output node 5 times per second. For one output node with 3 distinct inputs, this amounts to 15 distinct values received and changed per second. Though this time interval is configurable, this time interval was deemed to more than sufficiently convey the concurrency of interaction in mapping relationships. Whenever a distance measurement value is the same as the previous value, the input publishing is disregarded, as the output node already display the correct output for the given input. As an additional publish make no change to the output state, this mechanism is performed to avoid unnecessary network constraint.

After distance is measured and published on the input node, it is received on the output node, which immediately examines which color(s) is linked to this input. Verifying the

input value is part of a mapped relationship, the output node applies applicable mode changes and writes the value to appropriate color(s). Following the creation of a mapping relationship in figure 6.2, the output mode is configured as inverse mode. This means the output node inverse all received distance values, as seen in the sequence diagram in figure 6.3.

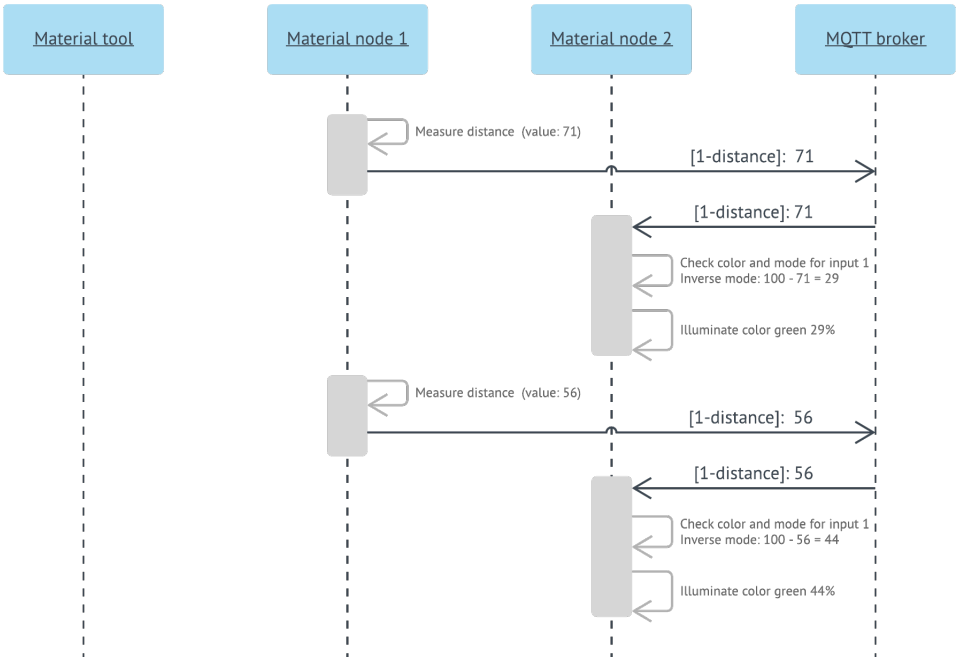


Figure 6.3: MQTT communication between two nodes following the mapping in figure 6.2

6.3.4 Unlink a mapped relationship

A user may wish to remove a mapped relationship between and as such this functionality has been made available to the user, as shown in figure 6.4. The broker maintains no state of all relationship, as all of this information is distributed on the material nodes. Hence the material tool has no knowledge of which material nodes are mapped together and accordingly cannot command both input and output to remove a mapped relationship. This is resolved by instructing the output node to remove the relationship of a given color, and have that output node further message the input node that it has one less output. This causes the input node to decrement its input counter, which causes it to stop publishing distance data if the input counter is zero.

The mapping relationship removal process is initiated by the material tool, who publishes a message on the output node's unoutput topic [2-unoutput], with a message containing the character of the color to be removed from a mapping. The output node proceeds to remove the mapping made to this output color and unsubscribe from the appropriate input distance

topic. Still aware of the input node's ID, the output node publishes to the input node's uninput topic [1-uninput], with a message with no content (nil). Receiving a message on the uninput topic, the input node decrements its input counter, and discovering the input counter is now zero, stop publishing distance data to [1-distance]. With this complete, both the input and output node have removed all knowledge of this previously mapped relationship.

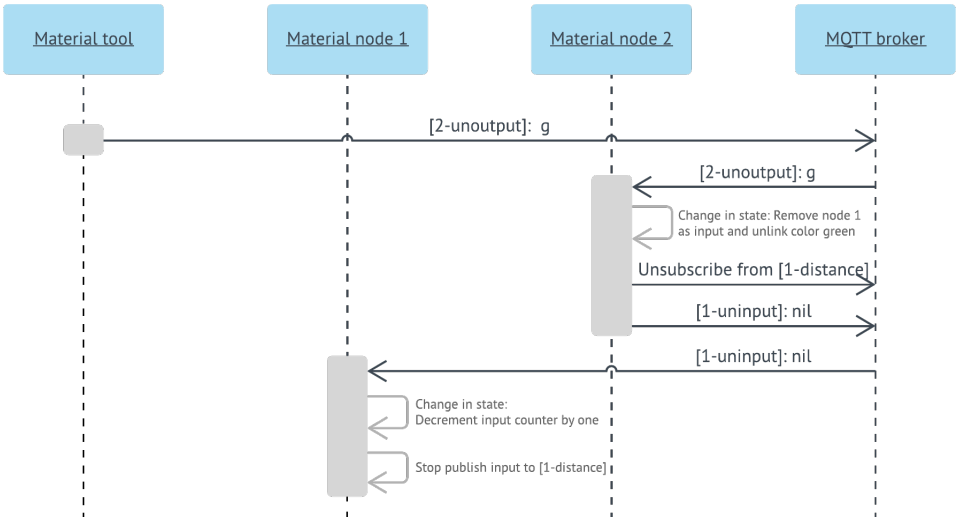


Figure 6.4: MQTT communication when removing the mapped relationship created in figure 6.2

6.4 Prototype: Hardware

Now onto the actual build of the prototype. Based on the material programming core concepts and minimal case, this proof of concept have a list of requirements for the prototype. This was used to create a proposed design for the material node and material tool in the previous chapter. Based on this prospective design, a list of hardware was drafted and schematics for the prototype was created.

6.4.1 Material node: Sensors and actuators

This prototype will use four material nodes used for the proof of concept, each material node with a similar set of sensors and actuators. Following is a full list of technology for each material device:

- ESP8266MOD
  - Small and cheap microcontroller - modified version of ESP8266
  - Built-in Wi-Fi module
  - 16 general-purpose input/output pins
  - Provide $3.3\,\mathrm{V}$ and $5\,\mathrm{V}$ power-supply pins
- RGB LED
  - Common cathode LED with colors red, green and blue
  - Require three $220\,\Omega$ resistors
- Ultrasonic distance measuring sensor
  - Common HC-SR04 ultrasonic sensor
  - Uses pulsating ultrasonic sound to measure distance
  - Require $5\,\mathrm{V}$ power-supply pin
- Passive RFID tag
  - Common MiFare Classic RFID tag laying next to the material device
  - Coded with the ID value used for identifying the material devices it belongs to
  - Integer ID coded in hexadecimal on the first 8 bytes of sector 1 block 7
- Battery power supply
  - Breadboard power supply Power MB V2 made by YwRobot
  - Provide $3.3\,\mathrm{V}$ and $5\,\mathrm{V}$ power-supply pins
  - Powered by common alkaline $9\,\mathrm{V}$ battery

6.4.2 Material node: Layout and implementation

With a decision on actuators and sensors based on the proposed design in figure 5.1 in the previous chapter, component layout and schematic was created for the prototype. A simplified view of hardware with pin connections to the microcontroller can be seen in table 6.2. Using these pin connections, complete schematics detailing all hardware and their connections can be seen in figure 6.5.

6.4.3 Material tool: Sensors and actuators

Only one material tool is needed to program the material nodes. Previous discussions surrounding core concepts, requirements and needed functionality of selection and color change, a list of compatible hardware equipment was formed:

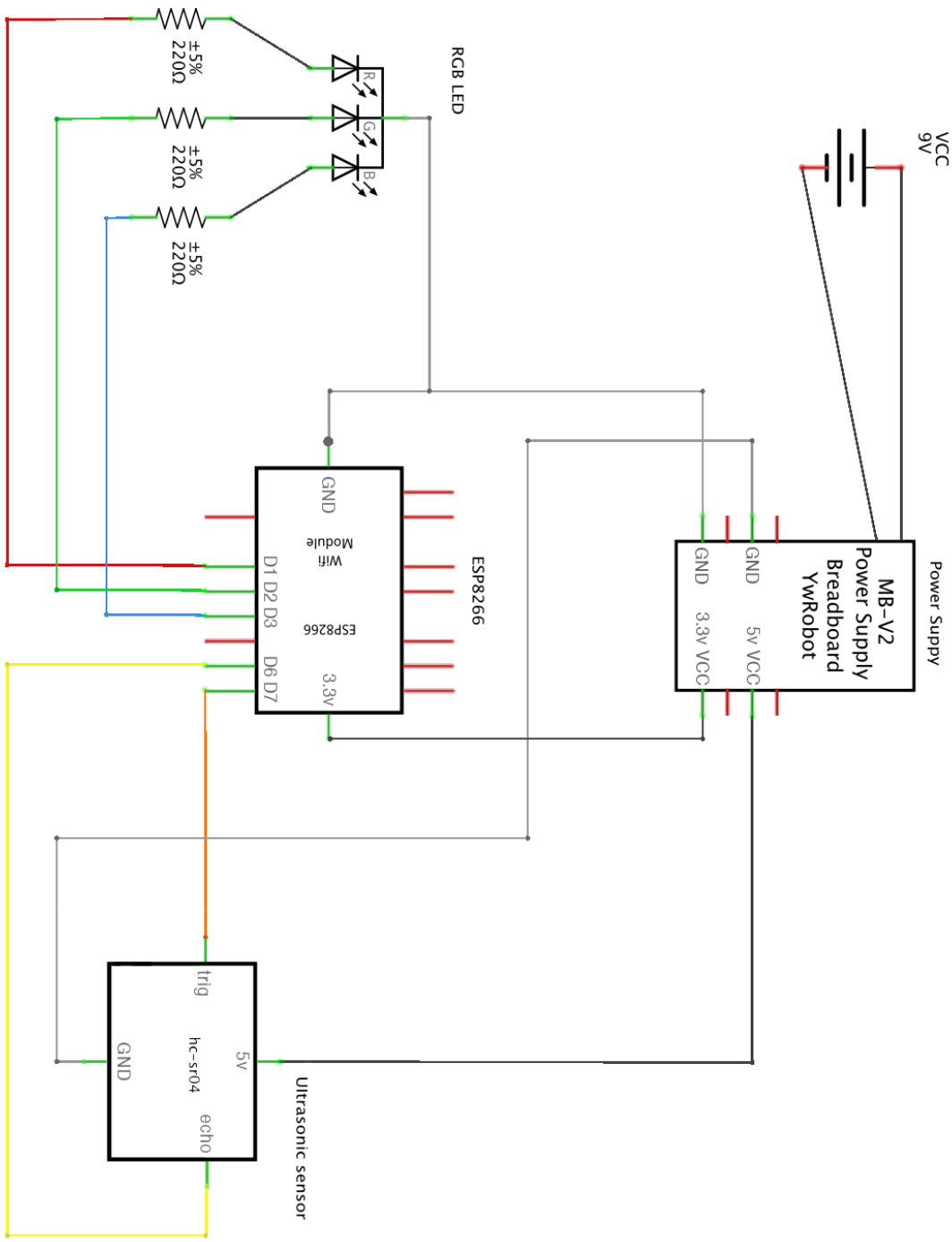


Figure 6.5: Schematic for the material node based on the design in figure 5.1

Microcontroller pin	Pin mode	Resistance	RGB LED	Ultrasonic sensor
5 V (power-supply pin)				VCC
GND (ground reference)			Cathode	GND
D1	Output	220 Ω	Red	
D2	Output	220 Ω	Green	
D3	Output	220 Ω	Blue	
D6	Output			Trigger
D7	Input			Echo

Table 6.2: Material node: Components connected to the microcontroller and their pin connections

- ESP8266MOD
  - Small and cheap microcontroller - modified version of ESP8266
  - Built-in Wi-Fi module
  - 16 general-purpose input/output pins
  - Provide 3.3 V and 5 V power-supply pins
- RGB LED
  - Common cathode LED with colors red, green and blue
  - Require three 220 Ω resistors
- RFID reader/writer
  - Common model RFID-RC522 sensor/actuator
  - Can read and write most RFID tags
  - Use Serial Peripheral Interface (SPI) to transfer data
  - Reads the ID of a passive MyFare Classic RFID tags, laying next the a material node
  - Also used to write the material node's ID to the RFID tags
- Button (4 units)
  - Common prototyping button
  - Is always in one of two states: pushed or not pushed
- Power supply
  - The material tool's breadboard doesn't have room for a breadboard battery power supply
  - Uses a portable USB power supply (power bank)

6.4.4 Material tool: Layout and implementation

The previous chapter detailed core concepts and requirements of a material tool in material programming. From this, a proposed design was created for the material tool, seen in figure 5.9. This lead to a list of hardware for the material tool in the previous chapter.

As seen in the list, the material tool uses 4 buttons, each to control the functionality of input, output, color, and mode. A small obstacle arises when there are an insufficient number of pins on the microcontroller for each of the four buttons to be connected to each its own digital pin. Instead, the four buttons are connected in a row using voltage division and read using an analog signal. Using this method of voltage division, it is possible to multiplex multiple buttons to one signal, by identifying different ranges of voltage. Each of the four buttons is serial connected with a $1000\,\Omega$ resistor, and the analog signal with one $10\,000\,\Omega$ ohm resistor. This technical solution removes the obstacle of not enough digital pins on the microcontroller.

The material node is powered by a breadboard power supply using external battery. Unfortunately, the material tool does not have room for another component on its breadboard. In an effort to adopt a simple solution, the material tool is powered using a micro USB cable connected to a portable power-supply (powerbank).

Following a decision on for the material tool, pin connections on the microcontroller were associated with each of the hardware's connectors, as seen in table 6.3. Using these designated connections, and the design in figure 5.9 in the the previous chapter, schematic for material tool was created. This schematic can be seen in figure 6.6.

Pin connection	**Resistance**	**RFID-RC522**	**RGB LED**	**Buttons**
GND (ground reference)		GND	Cathode	GND
3.3 V (power-supply pin)		VCC		
5 V (power-supply pin)	4 x $1000\,\Omega$			VCC
A0	$10\,000\,\Omega$			Analog read
D0	$220\,\Omega$		R (Red)	
D1	$220\,\Omega$		G (Green)	
D2	$220\,\Omega$		B (Blue)	
D3		RST (Reset)		
D5		SPI SCK		
D6		SPI MISO		
D7		SPI MOSI		
D8		SPI SDA/SS		

Table 6.3: Material tool: Components connected to the microcontroller and their pin connections

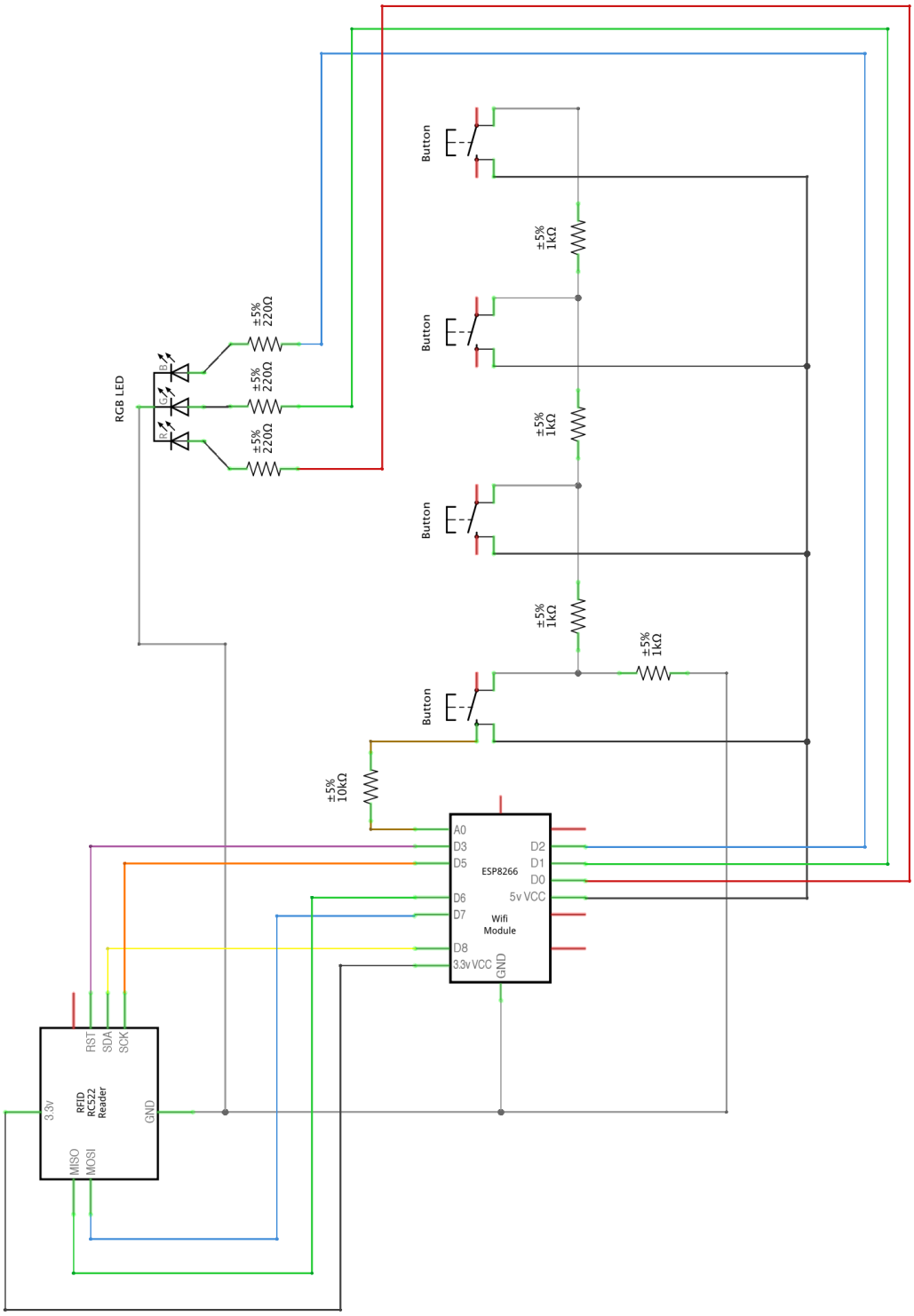


Figure 6.6: Material tool schematic based on the illustration in figure 5.9 and pin connections in table 6.3

6.5 Operating procedure

Now that the physical aspects of the prototype is detailed, it is time to explain how to actually use and interact with the proof of concept. This section will focus on how a user can operate the tool to perform material programming on the nodes.

6.5.1 Material tool buttons

As previously mentioned, the behavior of every mapping relationship is decided by four factors; input node, output node, color and mode. Consequently, the material tool has four separate buttons, each corresponding to functionality for the four variables. In an effort to make this choice a logical and sequential one, the idea is for the user to start on the left with the first button, and go one right for every choice, to end up on the last button to the right.

When creating a mapping relationship, the first action performed is to select an input node, and as such the first button on the left is the input button. The most rational subsequent action was debated much, whether to select an output node or choose relationship behavior first. Much contemplation resulted in the choice of letting a user configure the behavior variables first. After an input node is selected, the next choice is to choose which color to use, which is the second button from the left. Following a pick of color, an output mode can be chosen using the third button from the left. Lastly, an output node is selected using the button furthest to the right.

See figure 6.7 for an image of the material tool with a name of each functionality on the buttons. Corresponding to this figure, here is a list of the functionality of each button in the correct order:

1. Input
2. Color
3. Mode
4. Output

6.5.2 Material tool functionality

Now that the functionality of each button is explained, it is time to describe how to use them. The RGB LED is used in conjunction with the buttons to instruct the user of received commands, successful reads and completed commands. For the material tool, the RGB LED is the only actuator visible to a user, and thus it becomes a natural way of providing a user with information. The RGB LED provide feedback to a user about where in process they are and is the main method of inspecting which color and mode is chosen. In large parts of the process, the RGB LED will be illuminated in different colors to indicate the state and the progress. In addition to this, every time a user press a button, the RGB

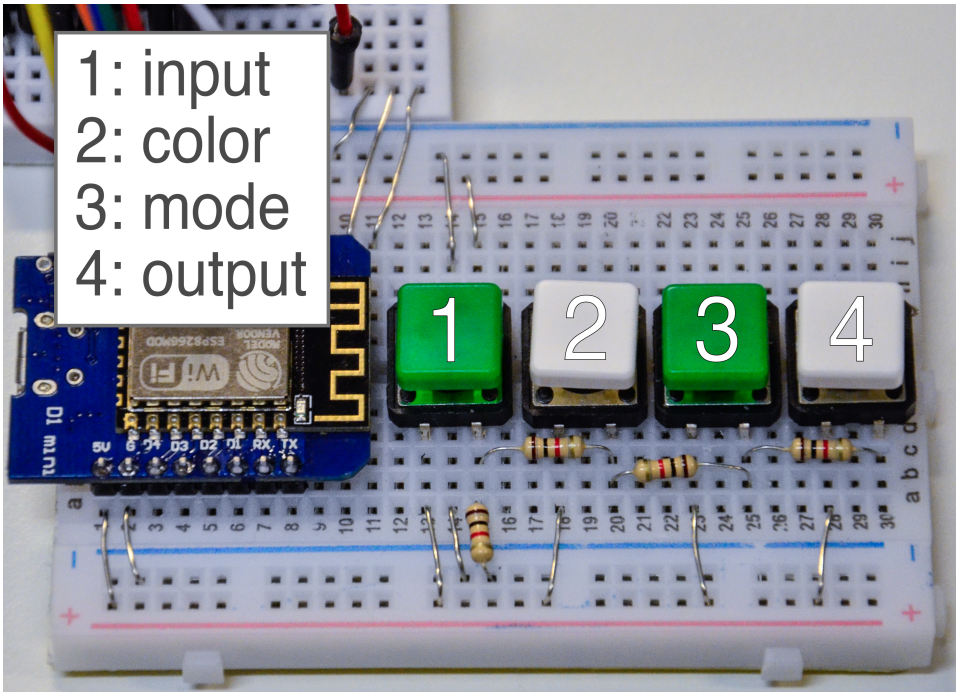


Figure 6.7: Functionality of material tool buttons

LED will carry out a brief light feedback to indicate the button press has been perceived. Because the RGB LED is illuminating and in various states most of the time, the light performs a ”reverse flash”, ceasing all color illumination for a very brief time period, just long enough for a user to notice. This functions as luminous feedback and executes at the same time the user feels tactile feedback from pressing the button. In many ways, the RGB LED functions as a command line interface for the material tool, providing a user with constant information about ongoing operations and chosen attributes. Any illegal or incorrect sequence of button presses will result in three rapid red flashes, after which the tool returns to its previous state. As the material tool is powered on, it will briefly flash a green light during setup to indicate it has connected to the LAN and MQTT broker, followed by a solid white light indicating the tool is ready for programming. A state diagram detailing a user's interaction with the material tool can be seen in figure 6.8.

To create a mapping, a user begins by pressing the input button. The RGB LED will flash briefly to indicate it is ready to read input device. The user proceeds to move the tool over to the RFID tag which is in the proximity of the material node chosen as input. Once an RFID tag is identified, a green light will flash to indicate a successful read, after which a solid green color is displayed. If a material node was selected by mistake, the user can press the input button again and read a new RFID tag. Now the input device is declared and a user can proceed to make configurations to the relationship behavior.

With the color and mode button, a user can alter the behavior attributes. There are three colors available to choose and by pressing the color button, the material tool cycles through them. The RGB LED will always illuminate in the currently chosen color, meaning three presses at the color button will change the color from green to blue with the first press, blue to red in the second press, and from red back to green with the third press. After a color is chosen, the user can proceed to configure output mode. There are two modes available; normal mode and inverse mode. Normal mode is chosen as default and is indicated by a solid and continuous light in the chosen color. A press on the mode button will put the mode to inverse, indicated by blinking light, alternating between the chosen color and no light. A press on the mode button again will subsequently put the mode back to normal mode.

With the behavior configured, a user can go ahead with selecting an input device. A press on the output button results in a short flash indicating the tool is ready to read a material device. Proceed to read the output node's RFID tag. Immediately following this, the material tool publishes the mapped relationship attributes to both input and output nodes, and flash twice white to announce the command is successfully sent. The relationship is now mapped on both material nodes, and the material tool goes idle and waits for further commands.

6.5.3 Mapping scenario

The previous section explained generic material tool functionality. This section will explain user interaction with the material tool and specific material nodes when trying to create a relationship mapping. The goal in this scenario is to create a relationship mapping from material node 1 as input to material node 2 as output, with behavior attributes blue and inverse mode. This scenario is also demonstrated using the format of a comic strip and can be seen in figure 6.9.

To begin with, place material node 1 and 2, as well as the material tool within reach (panel 1). The material tool is ready to program when it is flashing a solid white light. Press the input button and a short blink indicate the tool has registered your command and is ready to read the input node (panel 2). Move the tool over the RFID tag of material node 1 (panel 3). The tool will blink twice green to indicate the RFID tag has been read, and the input node selected. Now the material tool is ready to change behavior attributes (panel 4). The tool will default to green color and normal mode, so in case this is the desired attributes, a user can skip behavior configuration and continue with selecting an output node. This scenario specifies a blue color, so perform a press on the color button, and the RGB LED will change color to blue (panel 5). The goal behavior is using inverse mode, and the solid non-blinking light indicates the system is set to normal mode. By pressing the mode button, the LED starts blinking continuously to signal it is in inverse mode (panel 6). Now that the behavior attributes are configured, press the output button to select an output node (panel 7). Move the material tool above the RFID tag of material node 2 (panel 8). The LED will blink white three times to indicate the output node is read and the relationship mapping is published to the nodes. Now the mapping relationship is created and a user can start interacting with the material nodes (panel 9 and 10).

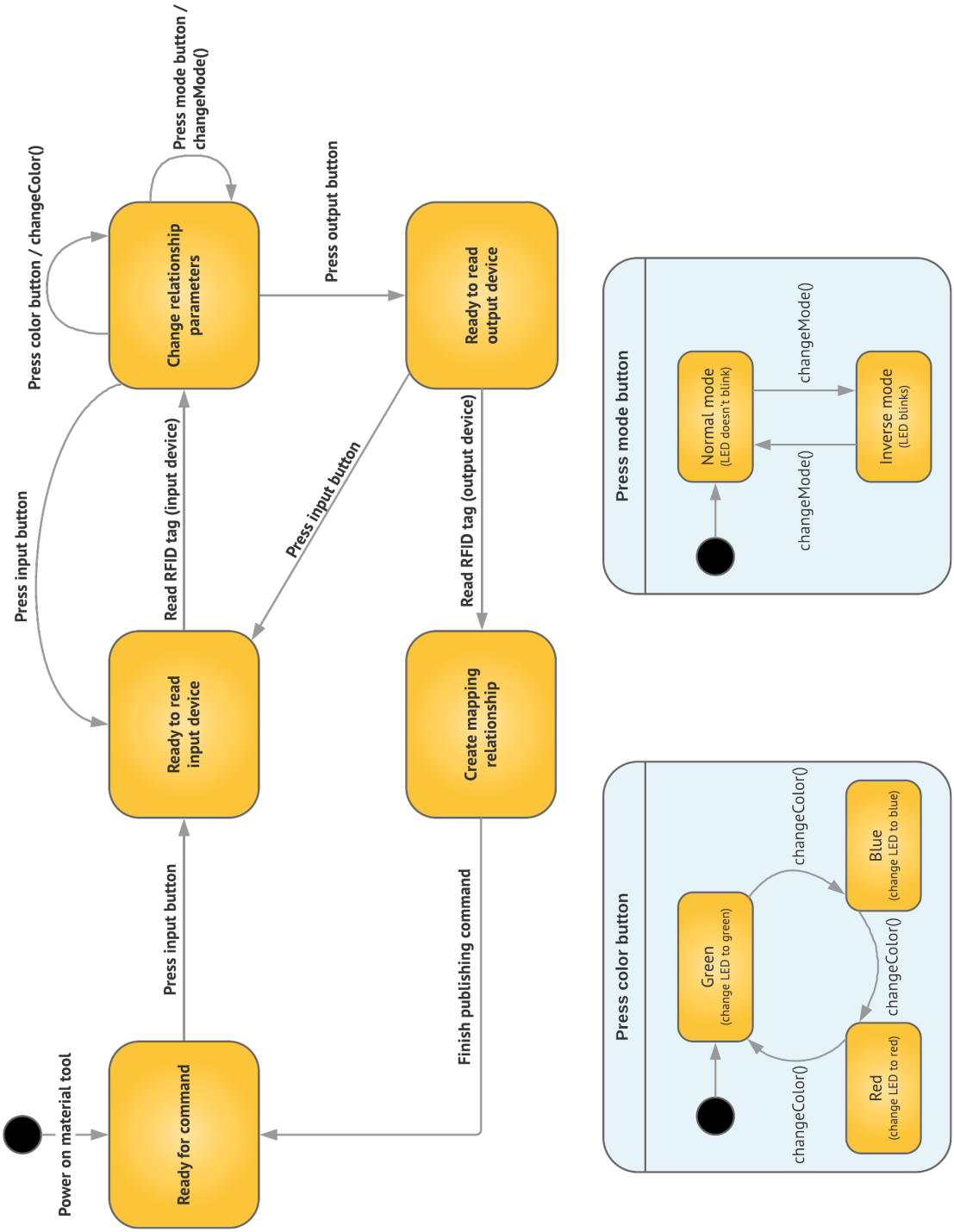


Figure 6.8: Material tool state diagram

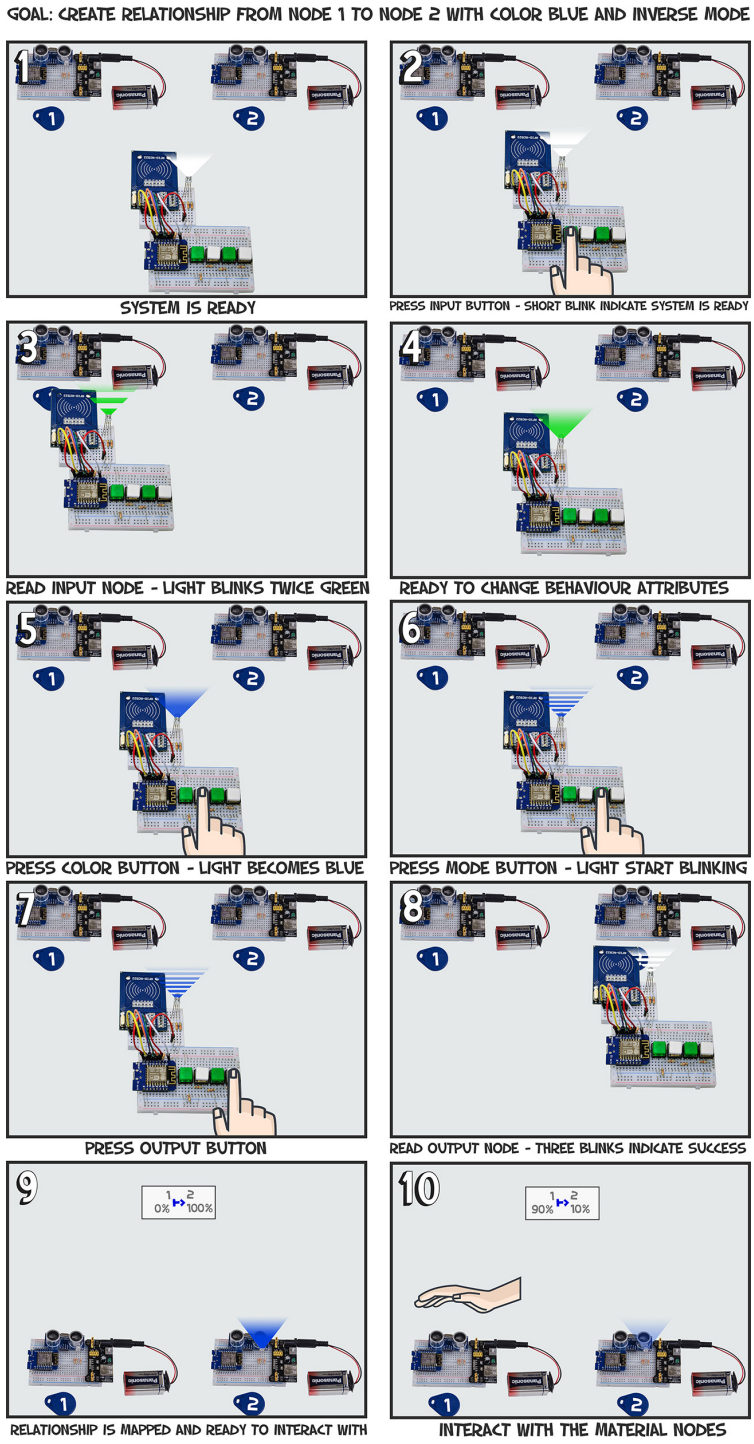


Figure 6.9: User scenario for creating a relationship mapping

6.6 Proof of concept: Software

6.6.1 Code

Developing this proof of concept has required enormous amounts of learning, testing, trying and failing. Many different microcontrollers, sensors, and actuators were tested and developed to intermediary prototypes before the proof of concept ended up looking like it does. All this trying and testing during the process made the software for the final proof of concept more stable and of higher quality. The code used for the material nodes can be found in the Appendix A on page 97 and the code for the material tool is found in Appendix B on page 110.

6.6.2 Libraries

In the development of the prototype, several Arduino libraries are utilized in the code to be able to take appropriate use of sensors, actuators, and communication. Most notable is the Arduino Core library which enables using the Arduino IDE and Arduino functionality on an ESP8266 microchip. See table 6.4 for a full list of all libraries used.

Used for	Library	Creator	Version	Available
RFID	MFRC522	Miguel Balboa	v1.4.0	github.com /miguelbalboa /rfid
ESP8266	Arduino Core	ESP8266 Community Forum	v2.4.1	github.com /esp8266 /Arduino
Wi-Fi	ESP8266WiFi (part of Arduino Core)	ESP8266 Community Forum	v2.4.1	github.com /esp8266 /Arduino
MQTT	PubSubClient	Nick O'Leary	v2.6	github.com /knolleary /pubsubclient
Serial Peripheral Interface (SPI) communication	SPI (included in Arduino IDE)	Arduino	latest	Arduino IDE
Ultrasonic distance measuring	NewPing	Tim Eckel	v1.9.0	bitbucket.org /teckel12 /arduino-new-ping

Table 6.4: Libraries used

6.7 Proof of concept: Hardware

This sections includes pictures of the finished proof of concept. See the material node in figures 6.10 and 6.11. See complete images of the material tool in figures 6.12 and 6.13. All the material nodes and material tool in the same environment can be seen in figure 6.14.

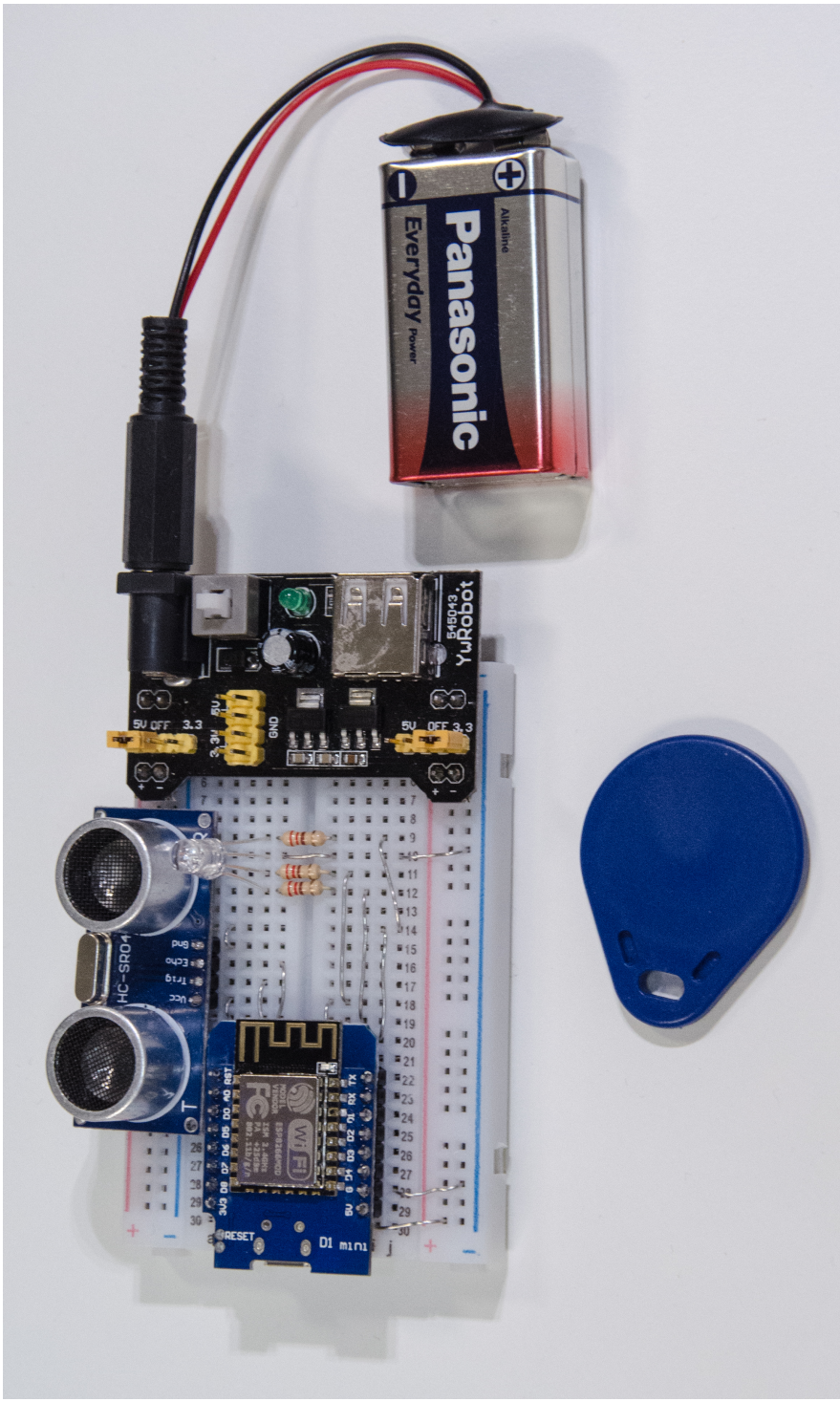


Figure 6.10: Material node (including RFID tag used for selection)

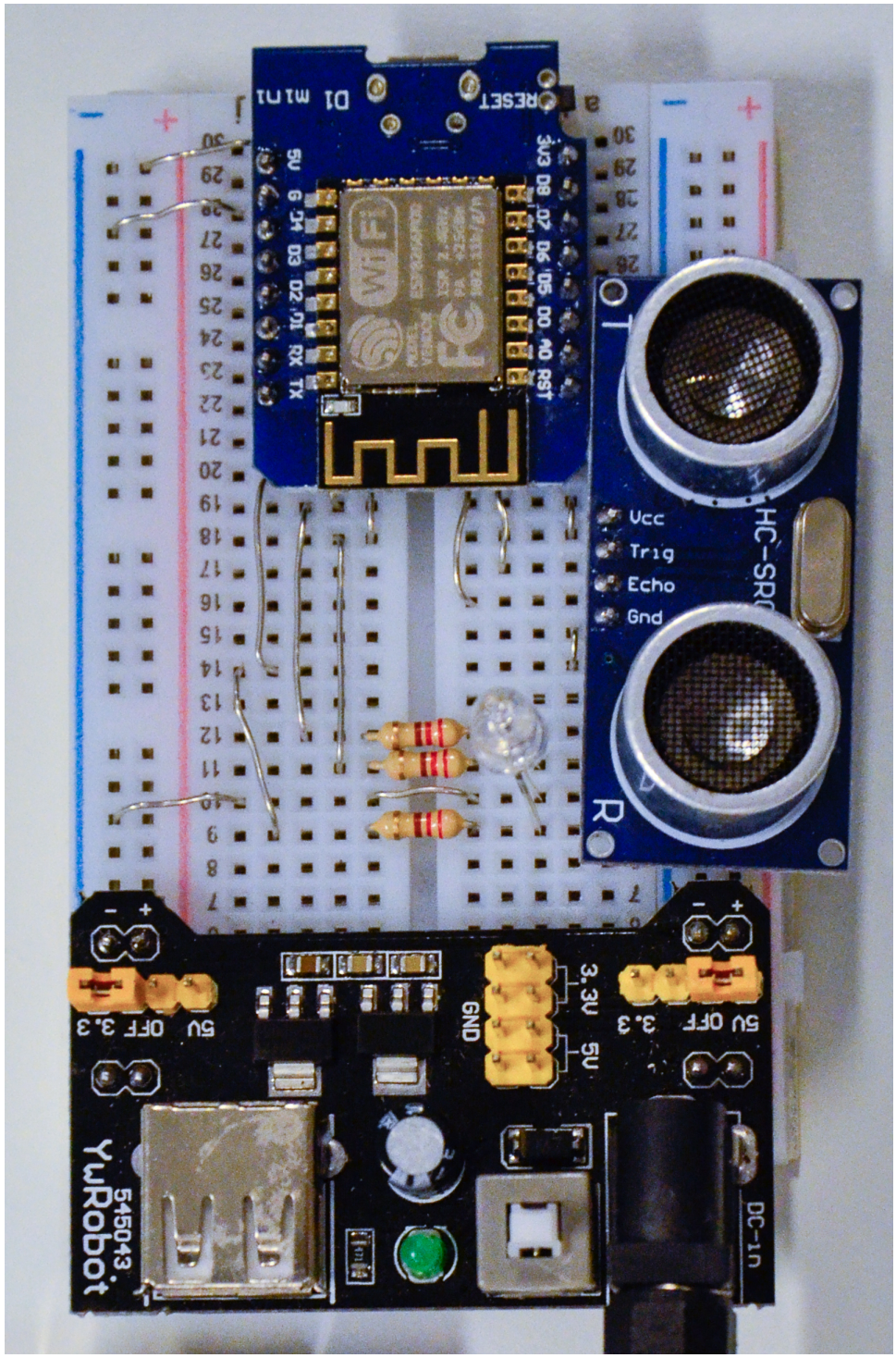


Figure 6.11: Material node viewed from the top (without the RFID tag)

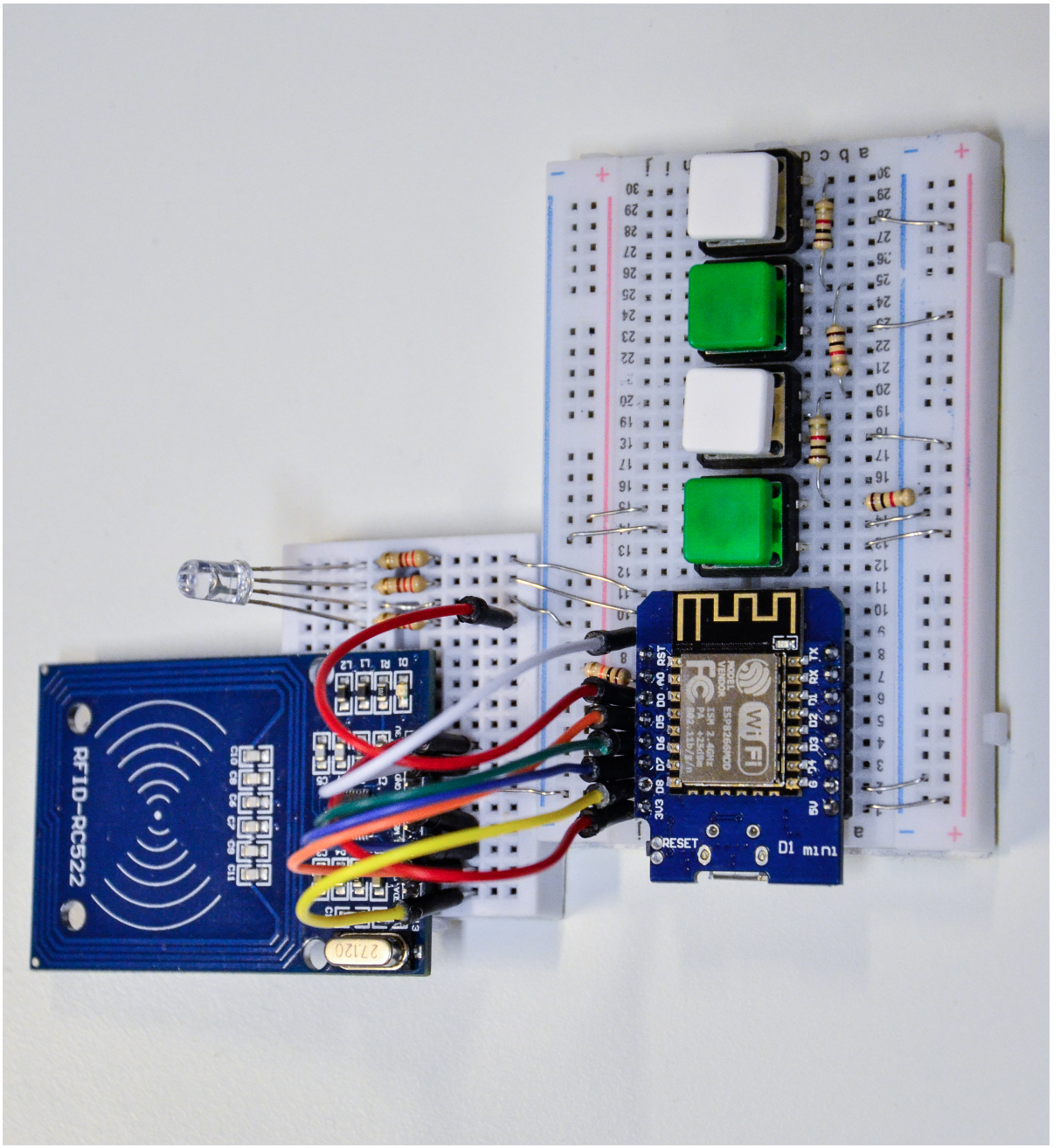


Figure 6.12: Material tool

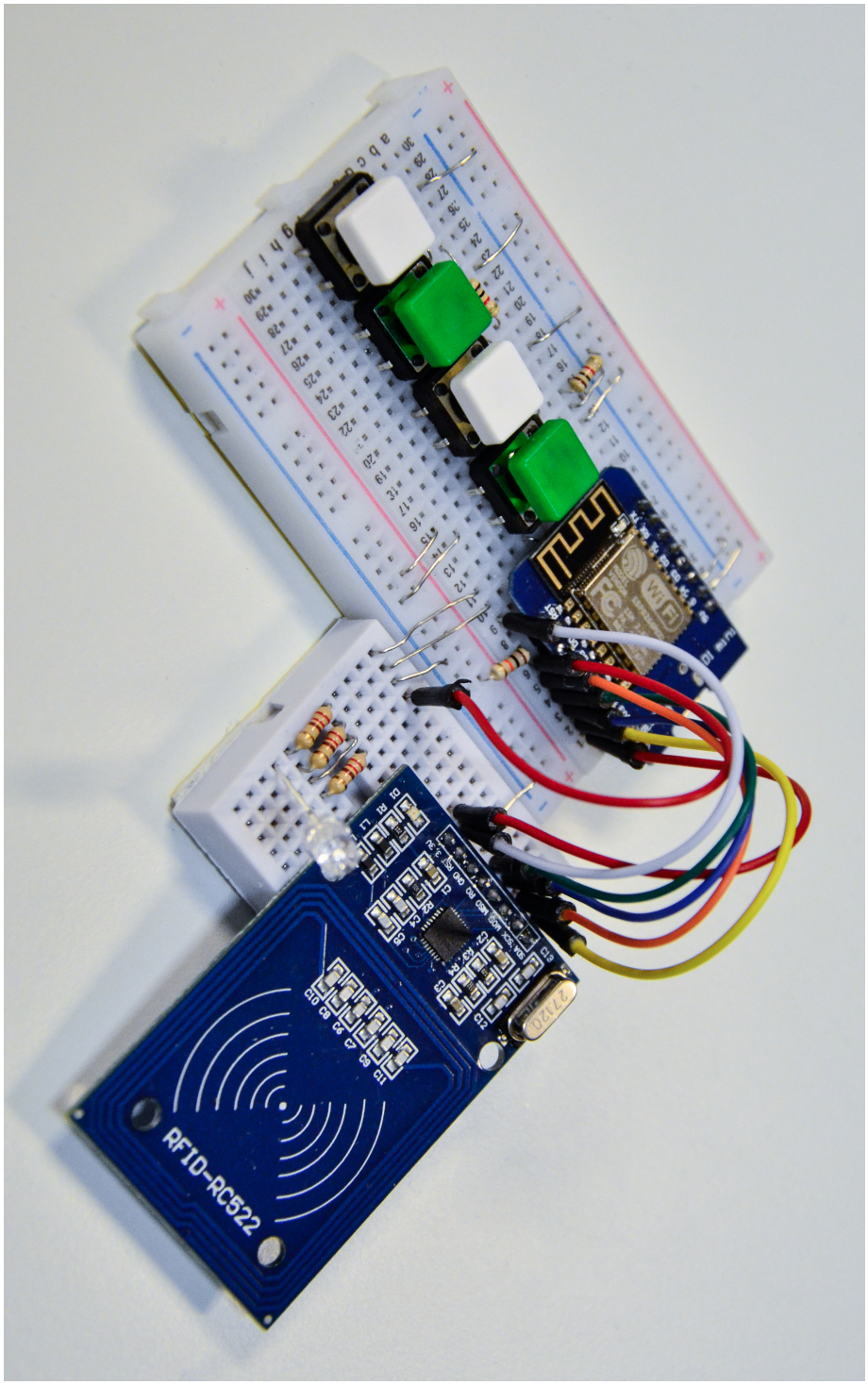


Figure 6.13: Material tool viewed from the side

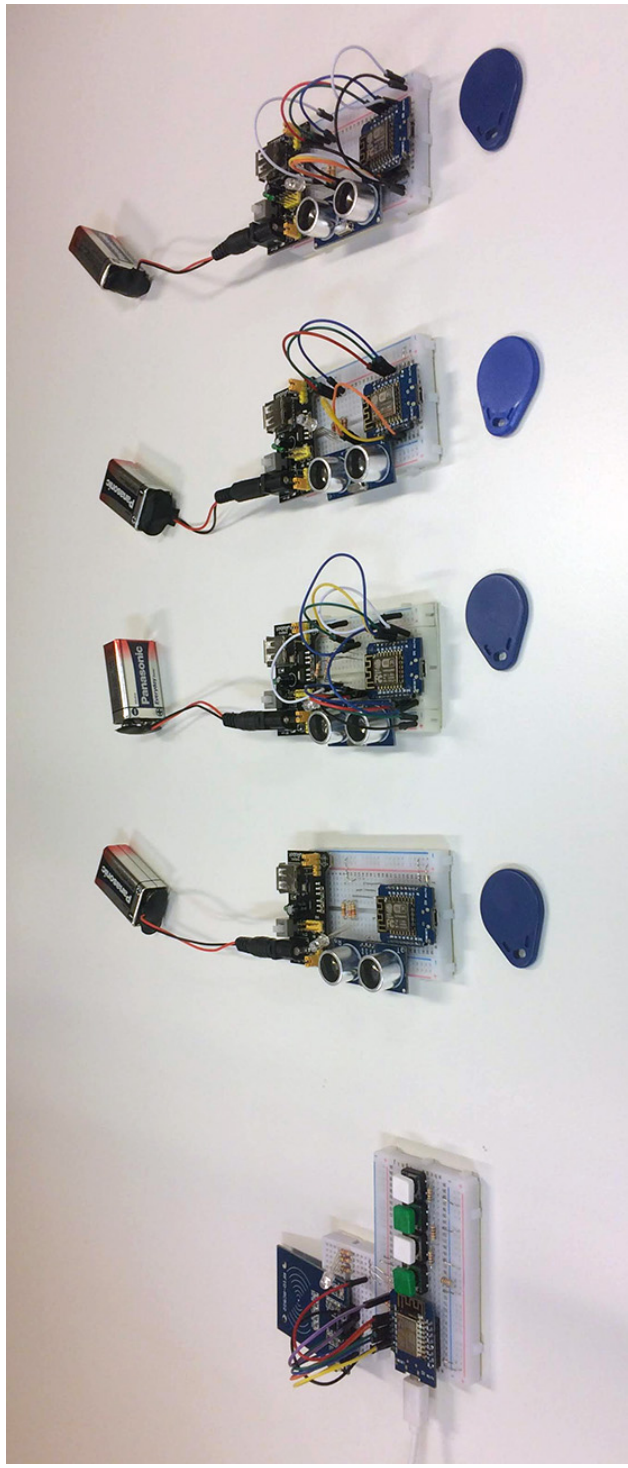


Figure 6.14: All four material nodes and the material tool used in the same environment

Chapter 7

Discussion and evaluation

This chapter will discuss the research questions and evaluate the proof of concept.

7.1 Research question 1

What is a minimal case (proof of concept) to illustrate the ideas of material programming?

The research questions and the background for their formulation are detailed in chapter 3. The second research question is closely related to the first, and to evaluate the proof of concept, this thesis needs to address the definition of a minimal case of material programming. In the early parts of chapter 5 this thesis contemplated the requirements for a minimal case of material programming. Three high level, core concepts were formulated:

1. Material parts should have computation and communication capabilities
2. behavior and interaction of material parts can be programmed using tools
3. A tool should be able to identify and select distinct material parts of a composite material

These concepts give a proof of concept some baseline for required functionality but is intentionally high-level and abstract. A material tool is used to select material nodes and program behavior between them. A communication protocol enables the material parts to communicate and coordinate behavior with each other. Relationship mappings are stored distributes on the actual nodes, and no central state is stored anywhere. Following these core concepts defined in chapter 5, more specific requirements for a minimal case prototype was devised for each of the core concepts. In the following list, this thesis will try to answer how each of the requirements for a minimal has been addressed.

1-a. A material part should be able to sense and actuate things

- Each material part are given one sensor to sense things and one actuator to actuate things
- The sensing component could be an ultrasonic sensor used for distance measuring
- The acting component could be an RGB LED used to change colors

1-b. The material parts should function as one cohesive material with internal communication

- Multiple material parts simulate one cohesive composite material, as a network of material parts with a ”physical” communication layer and an abstract logical layer
- A communication layer detail the direct communication connection between material parts
- A logical layer detail material parts’ behavior and relationship, and can be represented as a directed edge labeled graph

1-c. A change in one material part should allow for a change in another

- A mapping relationship enables a direct interaction relationship between material parts
- In a mapping relationship, a sensing change in an material part can result in an immediate acting change in another material part
- This interaction relationship can also be described as an input-output relationship
- Interaction between input and output can happen near instantaneous in what can be perceived as real-time behavioral interaction

1-d. Material parts should have computation capabilities

- Material parts can be a microcontroller with an embedded processing unit capable of computation
- Material parts can use sensing data from one material part to compute an appropriate actuating response in another material part

1-e. Material parts should themselves store the programmed behavior and interaction

- A distributed architecture in a network of material parts can facilitate internal storage of the state of a material part
- In a distributed network topology, each material part store its connections and relationships to all other material parts

2-a. A tool can program behavior and interaction on material parts

- A material tool can be used to change behavior and interaction between material parts
- A material tool can change the behavior of material parts with different colors and modes
- A material tool can create multiple coexisting and interacting behaviors on material parts
- A material tool can remove behavior and interaction on material parts

3-a. A tool can identify and select a material part or area (set of materials parts)

- A material tool can be used to select material parts
- A material tool can be used to select a sensing material part and an acting material part
- A material part can incorporate an identity component which can be read by a material tool
- A material tool can use RFID technology to identify and select material parts

With this list, this thesis aims to address each and every one of the specific requirements of the research question.

7.2 Research question 2

How suitable is today's internet of things technology to prototype material programming?

This research question has to be seen in relation to the first research question. Core to this research question is the use of technology related to the concept of internet of things (IoT). IoT is centered around the usage of sensors and actuators, connected to microcontrollers and microprocessors. These devices are connected to the internet, and typically publish data to a central server for analysis, processing, and further actions. This proof of concept takes usage of the notion of IoT using microcontroller, sensors, actuators and small devices connected to the internet, but does not implement some of the typical IoT architecture.

A commonplace way of using IoT devices is interconnecting all servers, storage functionality, data collection, and analysis in a cloud service. Cloud computing is a term used for gathering all relevant services under one shared pool of resources, which can be ubiquitously accessed from all internet connected devices. Cloud computing is a typical infrastructure in IoT technology, but not a defining part of the concept. The intention of this prototype is using a distributed architecture without any central knowledge or storage service, and this goal was achieved by storing internal state of interaction and mapping relationship on the material nodes themselves.

IoT encompasses a wide range of technologies and ideas, no one more correct than another. One goal when using IoT to develop this prototype, was to utilize the open source and open standard nature of IoT. There exist many proprietary standards and technologies which could have been used; Nordic Semiconductor, Texas Instruments, and Particle.io to mention a few. Instead, the use of open standards facilitates easy replication of research and further exploration and development beyond the scope of this thesis. All of the components and technologies used in the making are available for any user for use or purchase. Every microcontroller, sensor, actuator or component used in this proof of concept is off-the-shelf units without any modification or configuration before built into the prototype. Every component can be bought individually online for no more than a few US dollars. The computational component of each material device, the ESP8266MOD microcontroller, is for instance bought from China for $3-4. The Arduino IDE software platform is arguably the popular platform for developing prototypes in the world.

Communication is integral to every functionality in this proof of concept, as it should when the baseline for material programming is interaction and material parts connected with each other. The communication structure was built in a way as to separate each functionality in different channels (MQTT topic). This enables any device to take a look at one message in any given channel and know instantaneously what the message entails. This approach makes each communication transmission independent and open for examination as an isolated incident. This communication structure has another purpose: scalability. In this thesis, four material devices were used, each with a possibility of three distinct outputs. Each of these outputs can change color illumination according to input data 5 times a second. Using this communication architecture, this proof of concept facilitate enormous scalability. This prototype used 4 material devices as a way to demonstrate the feasibility of certain ideas and concepts of material programming, but there is no reason why this cannot be scaled up to simulate a composite material consisting of many more material parts. It does not have to be limited to material programming either, as the ideas of this prototype can be used in a number of use cases related to IoT.

Following a list of requirements for a minimal case for material programming, this proof of concept has addressed each topic to some extent. Core to this prototype is interaction with sensing component on one material part resulting in a behavior on an acting component on another part. This thesis considers this functionality to be displayed. To the extent that is possible using today's technology, this thesis deems IoT to be a viable method of illustrating the ideas of material programming.

7.3 Reflection

As the envisioned technology in future material programming undoubtedly doesn't exist yet, this proof of concept is an interesting solution to prototyping material programming using IoT technology today.

7.3.1 Proof of concept

Given the detailed requirements of the minimal case, a distributed network architecture enabled the proof of concept to disregard the use of a central computing or storage unit. A fascinating consequence of this is that the proof of concept only has only one programming lifecycle phase. In other words, the prototype developed in this thesis is always in run mode. This is similar to the functionality displayed by Alan Kay's Smalltalk programming language mentioned in the historical introduction [4]. In this prototype, programming using the tool and interacting with the material nodes does not interrupt the execution or workings of the network of materials. The material nodes function in a asynchronous and distributed manner, performing input-output interaction and listening to new commands in real time, unconcerned of simultaneous creation or removal of mapping relationships.

An important aspect of this prototype is the communication structure, facilitating every interaction and behavior in the proof of concept. As much as the communication structure was created for the specific purpose of this prototype, it has considerable potential as a general guide for future prototypes for material programming and IoT otherwise. One aspect of this proof of concept should as such be considered a framework for future prototypes. While technologies, libraries, and frameworks continuously change and develop over time, ideas and concepts are brought with. These ideas are usually not brought with "as is" with its explicit specifications, but provide a baseline for future idea improvements.

7.3.2 Reflection on related research

The most relevant existing research is that of Vallgårda, Boer, Tsaknaki and Svanæs' initial paper on material programming [26]. This paper is the main background and catalyst for this thesis and the development of the proof of concept. However, this thesis differentiates itself from the initial paper by focusing heavily on a physical prototype using existing technology. Although there surely exist some form of a prototype in the world using a distance sensor and an RGB LED connected to a microcontroller, this thesis puts the prototype in a setting and environment, with a very specific purpose.

The "programming" of material devices and the communication network facilitating it, makes this proof of concept very different from similar IoT products. The prototype showcased in this thesis has many similarities with the case of the computer-aided illustration of a color-changing material described in section 2.3.2. In this case, the sensing part of the material is a proximity sensor and the acting part is colored light. This bears much resemblance to this thesis' prototype, using a distance sensor as a sensing component and an RGB LED as an acting component. In the case of the color-changing material, the actuating part is a single acting color, which cycles through all visible colors using one input. In this regard, this thesis' prototype differs considerably, where the acting output color is a result of three distinct colors varying in light intensity. This dynamic interaction using mapping relationship is a large part of what makes this proof of concept distinctive in its own way.

Another relevant research project is Radical Atoms [8] detailed in section 2.4. This project has a large focus on the concept development of future materials, with the properties of the materials themselves being the main concentration. This notion of future materials as transformable and capable of computation share many similarities with computational composite [27] detailed in section 2.1, which is the foundation for material programming. Radical Atoms focus for the most part on the transformable properties of materials in a purely physical manner. Meaning materials as tangible and physically changeable by touch, snapping to geometric objects when shaped by a user's hands. While this is also a potential property of an envisioned computational composite material, the larger focus lies in the diversity of applications for a computational composite material, with transformation just being one of many potential properties.

The paper proposes three requirements for Radical Atoms as a material; *transform* its shape to reflect computational state and user input, *conform* to constraints imposed by the environment and user input, and *inform* users of its transformable capabilities [8]. 'Transform' here describes more a physical dynamic property, but can translate to the this prototype's ability to change its output color as a result of (possibly multiple) user inputs, and thus expose its internal state. 'Conform' fits well with the prototype's mapping relationship, where each mapping is created using specific behavior attributes it is not allowed to go beyond. 'Inform' refers to the dynamic affordances of the material inherently signaling to a user what it can do. Although not equally relevant, the envisioned goal of material programming is to create tools and materials which exhibit this inform functionality in explaining its functionality solely by looking at its construction. Although not exactly the same, parallels can be drawn between Radical Atoms authors' three requirements, and this thesis' three core concepts. Radical Atoms focus more on the material itself, in similarity to computational composites. Where Radical Atoms uses generic materials with no computational itself, material programming is more focused on the design practice and design tools involved in exploring the dynamic properties of such an envisioned future material.

7.3.3 Reflection on research strategy

This thesis uses the research strategy describe by Oates [16] as design and creation. This focus on the creation of new IT products, or artifact as described in the paper. In many ways, this strategy worked well for the thesis and the development of the proof of concept. With an initial perception in mind of how a final prototype would look like, the author of this thesis started experimenting with simple and well-known prototyping components, and progressed from this point. Although the initial perception was thought clear to begin with, the envisioned proof of concept changed greatly during the development process, progressing from basic sensor readings to a diverse collection of external components and smaller microcontrollers. This thesis has definitely embraced the problem-solving and iterative approach of this research strategy. To begin with, the author was unaware of the concept of computational composites and material programming. In a 'learning by making' fashion, a significant amount of knowledge has been accumulated in the course of the making of this thesis. From a development and prototyping perspective, as well as

from a vocational and subject perspective.

With the proof of concept being the main contribution to research knowledge, this thesis has demonstrated the technical feasibility of material programming using internet of things technology. Although this demonstration is naturally limited by the current state of technological progress, core concepts and ideas from material programming have been demonstrated from a physical and tangible perspective.

Granted the development has helped further increase the research knowledge of material programming in a physical format, the author wishes he had more time to explore additional ways of researching the final proof of concept. The author would have wanted to host workshops or experiments with test users, and explore how users unfamiliar with the concept would use the prototype. Also testing various functionality and understand how a user's comprehension of material programming and mapping relationships would look like in detail. The primary focus of this thesis was to try to make one "full-fledged" proof of concept for material programming, and given the open source nature of the prototype, much further work can be performed beyond the scope of this thesis.

Chapter 8

Conclusion

8.1 Further study

As mentioned at the end of the previous chapter, it would be very interesting to perform user experiments using this proof of concept. Also, try to teach users how to ”program” programmable materials in this concept and try to understand which parts of the prototype is comprehensible and which part can be improved further.

Using Moore’s law [14] to understand the trends and state of current technological growth, significant additional functionality can be expected of computational composites in the future. With the miniaturization of microcontrollers and integrated circuits follows a number of technological advancements beneficial in improving a material programming prototype using the available technology. The embedding of technology into a material itself is a central idea.

Further work can expand on the capabilities of programmable behavior and interaction, with the usage of new sensors and actuators possibly changing a user’s experience altogether. An expansion could explore different functionality a material tool, in addition to experimenting with the interface of the tool itself.

This proof of concept uses 4 material devices to showcase the functionality. It would be interesting to see how this scale up using the same hardware and software, using for example 30 or 50 material devices. In such a scenario, selecting one and one material part might be strenuous, and the material tool could be refined to select an area of material parts. This could be performed with a modification of the existing code and would result in an very interesting prototype.

8.2 Conclusion

This thesis has explored prototyping material programming using internet of things technology. The work is based on the somewhat abstract concept of computational composites and material programming, with the existing demonstrations being primarily computed-aided simulations. The goal of this thesis has been to develop a physical proof of concept for material programming using internet of things technology, with an additional intention of researching the viability of the concept using today's technology. The thesis has contemplated how to best showcase ideas and concepts from an envisioned future design practice, including which ideas are possible to display with the available technology.

The research questions were constructed around these topics, asking ”what is a minimal case (proof of concept) to illustrate the ideas of material programming?” and ”how suitable is today's internet of things technology to prototype material programming?”. Given the futuristic ideas of material programming including materials with properties not in existence, it is relevant to ponder how a prototype can showcase such ideas. The ideas from material programming were condensed into three core concepts, relatively uncoupled from the technological implementation. Using these, more specific requirements were able to be formulated, and further interpreted into abstract and physical properties of a prototype. The proof of concept has explored and experimented with the physicality of interactive dynamic materials, with user-programmed behavior.

With technology from internet of things core to the development, material devices were produced with sensing, actuating and computational components. This proof of concept has shown it is possible to develop a prototype displaying concepts of material programming, with a focus on using open standards and readily available off-the-shelf components. The communication architecture and structure has been a core idea in the development, and facilitates substantial scalability in terms of the size of the network. In many ways, this proof of concepts also work as a guide or framework for building similar and future prototypes. Interestingly, the design practice presented demonstrated has only one programming or design lifecycle phase, that of run mode. In other words, programming of the programmable material can be performed simultaneously with interaction with material nodes in predefined mapping relationship. The prototype proves an interesting take on material programming using today's technology. Using ideas and concept from the proof of concept provide future researchers with knowledge about the process of realizing material programming.

As mentioned in the historical introduction, Alan Kay envisioned Dynabook in the late 1960s and early 1970s long before the technology existed or such ideas were commonplace. He went on to participate in building the world's first graphical user interface personal computer, which he nicknamed ”the interim Dynabook” [10]. The interim Dynabook was nowhere near his vision, but it was a crucial milestone in his vision and a contributing factor to why personal computers look the way they do today. In a similar manner like Kay, the author of this thesis consider this proof of concept an interim prototype for material programming.

Bibliography

[1] Electric girls. The New York Times. April 26th 1884, Retrieved from https://timesmachine.nytimes.com/timesmachine/1884/04/26/106175469.pdf.

[2] Eric Brown. Who needs the internet of things? *Linux.com*, 23, 2016.

[3] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. Introducing the qest broker: Scaling the iot by bridging mqtt and rest. In *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, pages 36–41, Sept 2012.

[4] Adele Goldberg and Alan Kay. *Smalltalk-72: Instruction Manual*. Xerox Corporation, 1976.

[5] David Greelish. An interview with computing pioneer alan kay. *Time*, 2, April 2013.

[6] Stephanie Houde and Charles Hill. What do prototypes prototype? In *Handbook of Human-Computer Interaction (Second Edition)*, pages 367–381. Elsevier, 1997.

[7] Felix Hu, Ariel Zekelman, Michael Horn, and Frances Judd. Strawbies: Explorations in tangible programming. In *Proceedings of the 14th International Conference on Interaction Design and Children*, IDC '15, pages 410–413, New York, NY, USA, 2015. ACM.

[8] Hiroshi Ishii, Dávid Lakatos, Leonardo Bonanni, and Jean-Baptiste Labrune. Radical atoms: Beyond tangible bits, toward transformable materials. *interactions*, 19(1):38–51, January 2012.

[9] A. Kay and A. Goldberg. Personal dynamic media. *Computer*, 10(3):31–41, March 1977.

[10] Alan C Kay. A personal computer for children of all ages. In *Proceedings of the ACM annual conference-Volume 1*, page 1. ACM, 1972.

[11] Alan C Kay. The early history of smalltalk. In *History of programming languages—II*, pages 511–598. ACM, 1996.

[12] Verna Knapp. The smalltalk simulation environment. In *Proceedings of the 18th Conference on Winter Simulation*, WSC '86, pages 125–128, New York, NY, USA, 1986. ACM.

[13] Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Nagesh Subrahmanyam, Rong Xiang, Gerald Kallas, Neeraj Krishna, Stefan Fassmann, Martin Keen, et al. *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*. IBM Redbooks, 2012.

[14] G. E. Moore. Progress in digital integrated electronics [technical literaiture, copyright 1975 ieee. reprinted, with permission. technical digest. international electron devices meeting, ieee, 1975, pp. 11-13.]. *IEEE Solid-State Circuits Society Newsletter*, 11(3):36–37, Sept 2006.

[15] Gordon E Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8):4, 1965.

[16] Briony J Oates. *Researching information systems and computing*. Sage, 2006.

[17] Miller Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002.

[18] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Commun. ACM*, 52(11):60–67, November 2009.

[19] Y Series. Global information infrastructure, internet protocol aspects and next-generation networks. *ITU-T Recommendation Y*, June 2012.

[20] Scott B Steinman and Kevin G Carver. *Visual programming with Prograph CPX*. Manning Publications Co., 1995.

[21] Ivan E Sutherland. Sketchpad a man-machine graphical communication system. *Transactions of the Society for Computer Simulation*, 2(5):R–3, 1964.

[22] William Robert Sutherland. *The on-line graphical specification of computer procedures*. PhD thesis, Massachusetts Institute of Technology, 1966.

[23] Hideyuki Suzuki and Hiroshi Kato. Interaction-level support for collaborative learning: Algoblock—an open programming language. In *The First International Conference on Computer Support for Collaborative Learning*, CSCL '95, pages 349–355, Hillsdale, NJ, USA, 1995. L. Erlbaum Associates Inc.

[24] Jeffrey Travis and Jim Kring. *LabVIEW for everyone: graphical programming made easy and fun*. Prentice-Hall, 2007.

[25] Vasiliki Tsaknaki, Ylva Fernaeus, Emma Rapp, and Jordi Solsona Belenguer. Articulating challenges of hybrid crafting for the case of interactive silversmith practice. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, DIS '17, pages 1187–1200, New York, NY, USA, 2017. ACM.

[26] Anna Vallgårda, Laurens Boer, Vasiliki Tsaknaki, and Dag Svanæs. Material programming: A design practice for computational composites. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*, page 46. ACM, 2016.

[27] Anna Vallgårda and Johan Redström. Computational composites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 513–522, New York, NY, USA, 2007. ACM.

[28] Anna Vallgårda and Tomas Sokoler. A material strategy: Exploring material properties of computers. 2010.

[29] Daqiang Zhang, Huansheng Ning, Kevin S. Xu, Feiyu Lin, and Laurence Tianruo Yang. Internet of things. *J. UCS*, 18:1069–1071, 2012.

Appendices

Appendix A

Both of the code appendices are also found in this github repo: `https://github.com/sondrew/MaterialProgramming/`

The code specifically for the material node is found here: `https://github.com/sondrew/MaterialProgramming/blob/master/material_node/mp_material.ino`

```
/*
    MATERIAL NODE
    Part 1 of 2. Code for a master thesis proof of concept:
    'Prototyping Material Programming Using Internet of
       Things'
*/

// LIBRARIES
// Simplified distance measuring
#include <NewPing.h>
// Handle Wi-Fi connection for ESP8266
#include <ESP8266WiFi.h>
// MQTT client - utilizes Wi-Fi library
#include <PubSubClient.h>

// RGB LED pins
#define RED   D1
#define GREEN D2
#define BLUE  D3

//  Ultrasonic sensor pins
#define TRIGGER D6
#define ECHO    D7
// Ultrasonic sensor variables
#define MAX_DISTANCE 40
```

```
25 NewPing sonar(TRIGGER, ECHO, MAX_DISTANCE);
26
27 // VARIABLES FOR CONTROLLING INPUT PUBLISHING
28 // Number of devices using this device as input - publishes
       data if > 0
29 int inputCounter = 0;
30 int newTime = millis();
31 int oldTime = millis();
32 // Distance measuring time interval (millisec)
33 const int INTERVAL = 200;
34
35 //  VARIABLES FOR INPUT READING AND PUBLISHING
36 // Store distance as centimeters int in range 0-40 (
      MAX_DISTANCE)
37 int distance;
38 // Maps distance from 0-40 to 0-255 (8 bits)
39 int convertedDistance;
40 // Don't publish distance 0 multiple times if not changed
41 boolean publishZero = false;
42
43 // ID of this device and the ID on the RFID tag. Contains
      an integer
44 // but uses char for easy comparison and compatibility with
       topic names
45 // IMPORTANT: Increment this variable for each new material
       node
46 const char deviceId = '1';
47
48 // OUTPUT COLOR VARIABLES - TWO FOR EACH RGB COLOR
49 // redInput store deviceId of input node and redInverse
      store output mode
50 int redInput = 0;
51 boolean redInverse = false;
52 int greenInput = 0;
53 boolean greenInverse = false;
54 int blueInput = 0;
55 boolean blueInverse = false;
56
57 // MQTT/WIFI VARIABLES
58 const char* ssid = "WiFiName";
59 const char* password = "WiFiPassword";
60 // IP/host running your MQTT broker
61 const char* mqttServer = "127.0.0.1";
62 // Create WiFi client instance
63 WiFiClient espClient;
```

```
64 // Create MQTT client instance with WiFi client
65 PubSubClient client(espClient);
66 // Initialize variable for last received data
67 long lastMsg = 0;
68 // Initialize variable for publishing data
69 char msg[50];
70
71 // SUBSCRIBED TOPIC
72 const String topicInput = (String) deviceId + "-input";
73 const String topicOutput = (String) deviceId + "-output";
74 const String topicUnInput = (String) deviceId + "-uninput";
75 const String topicUnOutput = (String) deviceId + "-unoutput
       ";
76 // Name for topic to listen/publish to for input data
77 const String listeningTopicName = "distance";
78 const String publishTopicName = (String) deviceId + "-" +
       listeningTopicName;
79
80
81 void setup() {
82
83   Serial.begin(115200);
84   // Set pin modes for ultrasonic sensor and RGB LED
85   pinMode(TRIGGER, OUTPUT);
86   pinMode(ECHO, INPUT);
87   pinMode(RED, OUTPUT);
88   pinMode(GREEN, OUTPUT);
89   pinMode(BLUE, OUTPUT);
90
91   // Run helper function to setup Wi-Fi
92   wifiSetup();
93   // Set server with given variable and default MQTT port
94   client.setServer(mqttServer, 1883);
95   // Callback invoked when message on subscribed topic is
         received
96   client.setCallback(callback);
97
98   // Flash green to indicate system is initializd
99   analogWrite(GREEN, 250);
100  delay(1000);
101  turnOffLed();
102
103  Serial.println("Material␣" + (String) deviceId + "␣setup␣
         complete");
104 }
```

```
105
106 void loop() {
107   newTime = millis();
108
109   // Check if MQTT client is connected, (re)connect if not
110   if (! client.connected()) {
111     reconnect();
112   }
113   // Allow the client to process incoming messages and
        maintain its connection to the server
114   client.loop();
115
116   // Return if time interval is not reached
117   if (newTime < INTERVAL + oldTime) return;
118
119   // Publish distance if node is used as input
120   if (inputCounter > 0) {
121     distance = sonar.ping_cm(); // Distance is in range 0
          -> 40 (MAX_DISTANCE)
122     // Output can be written to with 8 bits
123     // Convert distance to a range of 0-255
124     convertedDistance = distance * 6.25;
125     //Serial.println("cm: " + (String) distance);
126     if (distance != 0 ) {
127       Serial.println("distance: " + (String) distance + "\t
            converted: " + (String) convertedDistance);
128       publishMessage(convertedDistance, publishTopicName);
129       publishZero = true;
130       oldTime = millis();
131     } else if (publishZero) {
132       Serial.println("Publish zero");
133       publishMessage(0, publishTopicName);
134       publishZero = false;
135       oldTime = millis();
136     }
137   }
138 }
139
140
141 void turnOffLed() {
142   analogWrite(RED, 0);
143   analogWrite(GREEN, 0);
144   analogWrite(BLUE, 0);
145 }
146
```

```
147 // ESP8266 WiFi setup
148 void wifiSetup() {
149   delay(10);
150   Serial.println();
151   Serial.print("Connecting␣to␣");
152   Serial.println(ssid);
153   // Start connecting to WiFi
154   WiFi.begin(ssid, password);
155 
156   // Print loading text while connecting
157   while (WiFi.status() != WL_CONNECTED) {
158     delay(500);
159     Serial.print(".");
160   }
161 
162   Serial.println();
163   Serial.println("WiFi␣connected");
164   Serial.print("Local␣IP␣address:␣");
165   Serial.println(WiFi.localIP());
166 }
167 
168 // MQTT function invoked when message is received
169 void callback(char* topic, byte* payload, unsigned int
      length) {
170   Serial.print("Message␣arrived␣[");
171   Serial.print(topic);
172   Serial.print("]:␣");
173   String message = "";
174   for (int i = 0; i < length; i++) {
175     Serial.print((char)payload[i]);
176     message += (char) payload[i];
177   }
178   Serial.println();
179   String splitTopicName = splitString(topic, '-', 1);
180   // Check and compare which topic a message is received on
181   // E.g. topic == "1-input"
182   int compareInputTopic = strcmp(topic, topicInput.c_str())
        ;
183   // E.g. topic == "1-output"
184   int compareOutputTopic = strcmp(topic, topicOutput.c_str
        ());
185   // E.g. topic == "1-uninput"
186   int compareUnInputTopic = strcmp(topic, topicUnInput.
        c_str());
187   // E.g. topic == "1-unoutput"
```

```
188   int compareUnOutputTopic = strcmp(topic, topicUnOutput.
          c_str());
189   // E.g. topic == "3-distance"
190   int compareListeningTopic = strcmp(listeningTopicName.
          c_str(), splitTopicName.c_str());
191
192
193   if (compareListeningTopic == 0) {
194     // Receiving data from input node
195     outputControl(topic, message);
196   } else if (compareOutputTopic == 0) {
197     // Use this device as output
198     outputSubscription(message);
199   } else if (compareInputTopic == 0) {
200     // Use this device as input
201     inputSubscription(message);
202   } else if (compareUnOutputTopic == 0) {
203     // Stop this device from being used as output
204     unOutputSubscription(message);
205   } else if (compareUnInputTopic == 0) {
206     // Stop this device from being used as input
207     unInputSubscription(message);
208   }
209 }
210
211 // Reconnect to MQTT broker if connection is lost
212 void reconnect() {
213   // Loop until reconnected
214   Serial.println("Device␣ID␣as␣string:␣" + (String)
          deviceId);
215   Serial.print("Standalone␣char:␣");
216   Serial.println(deviceId);
217   while (! client.connected()) {
218     Serial.println("Attempting␣MQTT␣connection...");
219     // Attempt to connect
220     String clientName = "Material␣" + (String) deviceId;
221     Serial.println("Client␣name:␣" + clientName);
222     if (client.connect(clientName.c_str())) {
223       Serial.println("Connected␣to␣MQTT␣Broker␣at␣" + (
              String) mqttServer);
224
225       // Subscribe to MQTT topics used for coordinating
              mapping relationships
226       client.subscribe(topicInput.c_str());
227       client.subscribe(topicOutput.c_str());
```

```
228       client.subscribe(topicUnInput.c_str());
229       client.subscribe(topicUnOutput.c_str());
230
231       // If connection is lost, reconnect to topics used as
              input
232       resubscribe();
233
234       // If you wish, publish a test announcement when
              connected
235       // client.publish("test", "hello broker");
236     } else {
237       Serial.print("failed, rc=");
238       Serial.print(client.state());
239       Serial.println(" try again in 5 seconds");
240       // Wait 5 seconds before retrying
241       delay(5000);
242     }
243   }
244 }
245
246 // Handle all data from an input node in a mapped
      relationship
247 // E.g. topic = "3-distance" and message = "142"
248 void outputControl(String topic, String message) {
249   int inputDevice = charToInt(topic[0]);
250   int data = message.toInt();
251
252   // Change multiple colors if input node is used for
        several outputs
253   if (inputDevice == redInput) changeColor('r', data);
254   if (inputDevice == greenInput) changeColor('g', data);
255   if (inputDevice == blueInput) changeColor('b', data);
256 }
257
258 // Use this device as an input node in a mapped
      relationship
259 // If inputCounter is > 0, publish input data to listening
      topic
260 void inputSubscription(String message) {
261   inputCounter++;
262   // String outputDevice = (String) message[0];
263   // Serial.println("This device (" + (String) deviceId +
        ") is now an input for device " + outputDevice);
264 }
265
```

```
266
267 // Use this device as an output node in a mapped
      relationship
268 // Starts listening to the input node's distance topic and
      uses this
269 // data to alter the output (color and mode) given the
      behaviour attributes
270 void outputSubscription(String message){
271   int inputDevice = charToInt(message[0]); // deviceID of
        input node
272   char color = message[2]; // I.e. 'r', 'g' or 'b'
273   char inverse = message[3]; // I.e. 't' (inverse mode) or
        'f' (normal mode)
274
275   //Serial.println("inputDevice: '" + (String) inputDevice
        + "'\ncolor: '" + color + "'\ninverse: '" + inverse +
        "'");
276
277   // No need to set inverse to false, as it's instatiated
        to false and reset if unsubscribed
278   if (color == 'r') {
279     redInput = inputDevice;
280     if (inverse == 't') {
281       redInverse = true;
282       changeColor('r', 255);
283     }
284   } else if (color == 'g') {
285     greenInput = inputDevice;
286     if (inverse == 't') greenInverse = true;
287   } else if (color == 'b') {
288     blueInput = inputDevice;
289     if (inverse == 't') blueInverse = true;
290   } else {
291     Serial.println("Error in outputSubscription: Color not
        valid [" + message + "] ");
292   }
293
294   // Subscribe to input node's distance topic
295   String topic = (String) inputDevice + "-" +
        listeningTopicName;
296   client.subscribe(topic.c_str());
297   Serial.println("This device ("+  (String) deviceId + ")
        is now linked to device " + (String) inputDevice + "
        with color " + color);
298   // printInputState();
```

```
299 }
300
301
302 // Remove mapping using this device as input
303 void unInputSubscription(String message) {
304   if (inputCounter > 0) inputCounter--;
305 }
306
307
308 // Remove mapping using this device as output
309 void unOutputSubscription(String message) {
310     char color = message[0];
311     Serial.println("Unsubscribing␣input.␣Color␣'" + (String
          ) color + "'.␣Printing␣state");
312     printInputState();
313     int inputDevice = 0;
314
315     if (color == 'r' ) {
316       Serial.println("Remove␣mapping␣to␣color␣red");
317       inputDevice = redInput;
318       redInput = 0;
319       redInverse = false;
320       changeColor('r', 0);
321     } else if (color == 'g') {
322       Serial.println("Remove␣mapping␣to␣color␣green");
323       inputDevice = greenInput;
324       greenInput = 0;
325       greenInverse = false;
326       changeColor('g', 0);
327     } else if (color == 'b') {
328       Serial.println("Remove␣mapping␣to␣color␣blue");
329       inputDevice = blueInput;
330       blueInput = 0;
331       blueInverse = false;
332       changeColor('b', 0);
333     }
334
335     // Print state after mapping relationship removal
336     // printInputState();
337
338     // Stop subscribing to input node's distance topic
339     String unsubscribeTopic = (String) inputDevice + "-" +
          listeningTopicName;
340     client.unsubscribe(unsubscribeTopic.c_str());
```

```
341     //Serial.println("Unsubscribed topic [" +
            unsubscribeTopic + "]");
342
343     // Tell input device to remove this mapping
344     String unInputTopic = (String) inputDevice + "-uninput"
            ;
345     publishMessage(charToInt(deviceId), unInputTopic);
346     // Serial.println("Sent unInput signal to topic [" +
            unInputTopic + "]: '" + (String) deviceId + "'");
347     // turnOffLed();
348 }
349
350 // Helper function for debug
351 void printInputState() {
352   Serial.println("COLOR \t\t DEVICE \t INVERSE");
353   Serial.println("Red \t\t " + (String) redInput + " \t\t "
          + (String) redInverse);
354   Serial.println("Green \t\t " + (String) greenInput + " \t
          \t " + (String) greenInverse);
355   Serial.println("Blue \t\t " + (String) blueInput + " \t\t
          " + (String) blueInverse);
356 }
357
358
359 // If MQTT connection breaks, this will make sure your
360 // client resubscribes to all devices used as input
361 void resubscribe() {
362   turnOffLed(); // Make sure a color doesn't get stuck
          displaying a color
363   String topic;
364
365   if (redInput != 0) {
366     topic = (String) redInput + "-" + listeningTopicName;
367     Serial.println("Resubscribing red LED to topic " +
            topic);
368     client.subscribe(topic.c_str());
369   }
370   if (greenInput != 0) {
371     topic = (String) greenInput + "-" + listeningTopicName;
372     Serial.println("Resubscribing green LED to topic " +
            topic);
373     client.subscribe(topic.c_str());
374   }
375   if (blueInput != 0) {
376     topic = (String) blueInput + "-" + listeningTopicName;
```

```
377     Serial.println("Resubscribing␣blue␣LED␣to␣topic␣" +
            topic);
378     client.subscribe(topic.c_str());
379   }
380 }
381
382
383 void changeColor(char color, int ledValue) {
384   // color == 'r' || 'g' || 'b'
385   // 0 <= ledValue <= 255
386
387   // Have a threshold for low values, as the sensor is bad
          at reading short distances
388   int inverseMinValue = 26;
389   int ledStrength;
390
391   if (color  == 'r') {
392     if  (!redInverse && ledValue == 0) ledStrength = 0;
393     else if (!redInverse && ledValue != 0) ledStrength =
            255 - ledValue;
394     else if (redInverse && ledValue == 0) ledStrength =
            255;
395     else if (redInverse && ledValue < inverseMinValue)
            ledStrength = 0;
396     else if (redInverse && ledValue != 0 ) ledStrength =
            ledValue;
397     Serial.println("Set␣color␣red␣to␣value␣" + (String)
            ledStrength);
398     analogWrite(RED, ledStrength);
399   } else if (color == 'g') {
400     if (!greenInverse && ledValue == 0) ledStrength = 0;
401     else if (!greenInverse && ledValue != 0) ledStrength =
            255 - ledValue;
402     else if (greenInverse && ledValue == 0) ledStrength =
            255;
403     else if (greenInverse && ledValue < inverseMinValue)
            ledStrength = 0;
404     else if (greenInverse && ledValue != 0) ledStrength =
            ledValue;
405     Serial.println("Set␣color␣green␣to␣value␣" + (String)
            ledStrength);
406     analogWrite(GREEN, ledStrength);
407   } else if (color == 'b') {
408     if  (!blueInverse && ledValue == 0) ledStrength = 0;
```

```
409     else if (!blueInverse && ledValue != 0) ledStrength =
            255 - ledValue;
410     else if (blueInverse && ledValue == 0) ledStrength =
            255;
411     else if (blueInverse && ledValue < inverseMinValue)
            ledStrength = 0;
412     else if (blueInverse && ledValue != 0) ledStrength =
            ledValue;
413     Serial.println("Set␣color␣blue␣to␣value␣" + (String)
            ledStrength);
414     analogWrite(BLUE, ledStrength);
415   }
416 }
417
418
419 // Helper function of returning a substring before or after
        a (char) seperator
420 String splitString(String data, char separator, int index)
      {
421     int found = 0;
422     int strIndex[] = { 0, -1 };
423     int maxIndex = data.length() - 1;
424
425     for (int i = 0; i <= maxIndex && found <= index; i++) {
426         if (data.charAt(i) == separator || i == maxIndex) {
427             found++;
428             strIndex[0] = strIndex[1] + 1;
429             strIndex[1] = (i == maxIndex) ? i+1 : i;
430         }
431     }
432     return found > index ? data.substring(strIndex[0],
            strIndex[1]) : "";
433 }
434
435 void publishMessage(String data, String topic) {
436   /* We start by writing our data as formatted output to
        sized buffer (using snprintf)
437    * In this case we use a String, and we must format it as
          such (%s)
438    * There are different formatting options based on the
        data type:
439    *
440    * %lu unsigned long
441    * %ld signed long
442    * %d integer
```

```
443     * %f float
444     * %s string
445     * %c char
446     */
447   const char* top = topic.c_str();
448   snprintf (msg, 75, "%s", data.c_str()); // Store our data
          'tagId' in the sized buffer 'msg'
449   Serial.print("Publish␣message:␣");
450   Serial.println(msg);
451   client.publish(top, msg);
452 }
453
454 void publishMessage(int data, String topic) {
455   /* We start by writing our data as formatted output to
          sized buffer (using snprintf)
456     * In this case we use an integer, and we must format it
           as such (%d)
457     * There are different formatting options based on the
           data type:
458     *
459     * %lu unsigned long
460     * %ld signed long
461     * %d integer
462     * %f float
463     * %s string
464     * %c char
465     */
466   const char* top = topic.c_str();
467   snprintf (msg, 75, "%d", data); // Store our data 'tagId'
          in the sized buffer 'msg'
468   Serial.print("Publish␣message:␣");
469   Serial.println(msg);
470   client.publish(top, msg);
471 }
```

Appendix B

The code specifically for the material tool is found here: `https://github.com/sondrew/MaterialProgramming/blob/master/material_tool/mp_tool.ino`

```
1  /*
2      MATERIAL TOOL
3      Part 2 of 2. Code for a master thesis proof of concept:
4      'Prototyping Material Programming Using Internet of
           Things'
5  */
6
7  // LIBRARIES
8  // Used for SPI communication for RFID (included in the
       Arduino IDE)
9  #include <SPI.h>
10 // RFID library
11 #include <MFRC522.h>
12 // Handle Wi-Fi connection for ESP8266
13 #include <ESP8266WiFi.h>
14 // MQTT client - utilizes Wi-Fi library
15 #include <PubSubClient.h>
16
17 /* BUTTONS
18 Multiple buttons with voltage division use one analog pin
19 Analog button value is read in 10 bits (range 0-1024)
20 The different buttons will have analog values within this
       range
21 Listed from left to right (closer to further away from
       input):
22
23 BUTTON    ANALOG VALUE IN RANGE
24 Input     900-1024
25 Color     600-750
26 Mode      400-500
27 Output    300-390
28 */
29 #define BUTTONS   A0
30 // RGB LED pins
31 #define LED_RED   D0
32 #define LED_GREEN D2
33 #define LED_BLUE  D1
34 // RFID pins
```

```
35 #define RST_PIN   D3
36 #define SS_PIN    D8
37
38 // VARIABLES FOR FUNCTIONALITY FLOW CONTROL
39 // Variables for reading and comparing new and old analog
       button values
40 int newAnalogValue;
41 int oldAnalogValue;
42 // newTime and oldTime control flow based on an interval
       timing system
43 int newTime;
44 int oldTime;
45 const int INTERVAL = 200;
46 boolean inputPressed = false;
47 boolean outputPressed = false;
48 // Illumination feedback - LED will stop flashing for
49 // x ms to emulate tactile feedback upon button press
50 const int tactileFlash = 50;
51
52 // RFID VARIABLES
53 // Create MFRC522 instance (RFID reader)
54 MFRC522 rfid(SS_PIN, RST_PIN);
55 // Initialize RFID authentication key variable
56 MFRC522::MIFARE_Key key;
57 // Location of sector of material node ID
58 byte sector         = 1;
59 // Location of block containing the material node ID (first
       8 bytes)
60 byte blockAddr      = 4;
61 // Location of sector keys
62 byte trailerBlock   = 7;
63
64 // MQTT/WIFI VARIABLES
65 const char* ssid = "WiFiName";
66 const char* password = "WiFiPassword";
67 // IP/host running your MQTT broker
68 const char* mqttServer = "127.0.0.1";
69 // Create WiFi client instance
70 WiFiClient espClient;
71 // Create MQTT client instance with WiFi client
72 PubSubClient client(espClient);
73 // Initialize variable for last received data
74 long lastMsg = 0;
75 // Initialize variable for publishing data
76 char msg[50];
```

```
77
78 // MAPPING RELATIONSHIP VARIABLES
79 int inputDevice = 0;
80 int outputDevice = 0;
81 char outputColor = 'g';
82 boolean blinkHigh = true;
83 boolean invertedDistance = false;
84
85
86 void setup() {
87   Serial.begin(115200);
88   // All buttons as one analog input
89   pinMode(BUTTONS, INPUT);
90
91   oldTime = millis();
92
93   // Initialize SPI bus to communicate with RFID reader
94   SPI.begin();
95   // Initialize MFRC522 (RFID reader)
96   rfid.PCD_Init();
97   delay(200);
98
99   // Run helper function to setup WiFi
100  wifiSetup();
101  // Set server with given variable and default MQTT port
102  client.setServer(mqttServer, 1883);
103  // Callback invoked when message on subscribed topic is
       received
104  client.setCallback(callback);
105
106
107  // Initialize RFID hex key to FFFFFFFFFFFF (factory
       default)
108  for (byte i = 0; i < 6; i++) {
109    key.keyByte[i] = 0xFF;
110  }
111
112  // RGB LED pin modes
113  pinMode(LED_RED, OUTPUT);
114  pinMode(LED_GREEN, OUTPUT);
115  pinMode(LED_BLUE, OUTPUT);
116  // Set idle/awaiting command state to color white
117  setColor(200, 200, 200);
118  delay(200);
119  Serial.println("\nMaterial␣programming␣tool␣ready");
```

```
120 }
121
122
123 void loop() {
124   // Check if MQTT client is connected, (re)connect if not
125   if (! client.connected()) {
126     reconnect();
127   }
128
129   // Allow the client to process incoming messages and
        maintain its connection to the server
130   client.loop();
131
132   newTime = millis();
133
134   newAnalogValue = analogRead(BUTTONS);
135   // Read analog button value every loop - disregard values
        under 100
136   if (newAnalogValue > 100) readButtons(newAnalogValue);
137
138   // Run in every loop to blink LED if inverse mode is
        active
139   // Use a timed interval flow
140   blinkColor();
141
142   // Look for new RFID tags
143   if ( ! rfid.PICC_IsNewCardPresent()) {
144     delay(50);
145     return;
146   }
147   // Select the RFID tag
148   if ( ! rfid.PICC_ReadCardSerial()) {
149     delay(50);
150     return;
151   }
152
153   // Initialize RFID reading variables
154   MFRC522::StatusCode status;
155   byte buffer[18];
156   byte size = sizeof(buffer);
157
158   // Authenticate RFID tag using known MiFare Classic
        default keys:
159   // github.com/miguelbalboa/rfid/blob/master/examples/
        rfid_default_keys/rfid_default_keys.ino
```

```
160   status = (MFRC522::StatusCode) rfid.PCD_Authenticate(
          MFRC522::PICC_CMD_MF_AUTH_KEY_A, trailerBlock, &key,
          &(rfid.uid));
161   if (status != MFRC522::STATUS_OK) {
162     Serial.print(F("PCD_Authenticate() failed: "));
163     Serial.println(rfid.GetStatusCodeName(status));
164     return;
165   }
166
167   // Print the unique id of the RFID tag (not the same as
          material node ID)
168   //Serial.print(F("Card UID:"));
169   //dumpByteArray(rfid.uid.uidByte, rfid.uid.size);
170
171   // Print sector 1 of RFID tag
172   //Serial.println(F("\nCurrent data in sector:"));
173   //rfid.PICC_DumpMifareClassicSectorToSerial(&(rfid.uid),
          &key, sector);
174
175   // Read first 8 bytes of block 4 on sector 1 containing
          the material node ID
176   status = (MFRC522::StatusCode) rfid.MIFARE_Read(blockAddr
          , buffer, &size);
177     if (status != MFRC522::STATUS_OK) {
178       Serial.print(F("MIFARE_Read() failed: "));
179       Serial.println(rfid.GetStatusCodeName(status));
180     }
181
182   // Print all 16 bytes of block 4 sector 1
183   //Serial.print(F("Data in block ")); Serial.print(
          blockAddr); Serial.println(F(":"));
184
185   // Store RFID data containing material node ID as String
186   String identification = returnByteString(buffer, 8);
187   // Cast material node ID to int - removes redundant hex
          values
188   int deviceNumber = identification.toInt();
189   //Serial.println("deviceNumber: " + (String) deviceNumber
          );
190
191   // The read material device is an input node
192   if (inputPressed && !outputPressed) {
193     // Input device already set - cannot read again
194     if (inputDevice != 0) {
195       //Serial.println("Input device has already been set (
```

```
            device " + (String) inputDevice + ")");
196       failureRead();
197     } else { // Use ID of read RFID tag as input node
198       inputDevice = deviceNumber;
199       //Serial.println("Input set to device " + (String)
              inputDevice);
200       successfulRead();
201     }
202   } else if (inputPressed && outputPressed) {
203     // Input node is already set
204     // Use ID of read RFID tag as output node
205     if (inputDevice != 0) {
206       outputDevice = deviceNumber;
207       // Publish mapping relationship over MQTT
208       //printVariables();
209       // Publish MQTT message to input and output node
210       publishCommand();
211       // Light feedback to indicate successfull command
212       successfulCommand();
213       //Serial.println("Resetting all settings");
214       resetVariables();
215     } else {
216       // Both input and output pressed without any tags
              read
217       // This means the read tag is told to remove mapping
              with chosen color
218       outputDevice = deviceNumber;
219       //Serial.println("UnOutput device " + (String)
              outputDevice + " with color " + (String)
              outputColor);
220       publishUnOutput();
221       successfulUnOutput();
222       //Serial.println("Resetting all settings");
223       resetVariables();
224     }
225   } else {
226     Serial.println("RFID␣tag␣was␣scanned␣(device:␣" + (
          String) deviceNumber + ")␣but␣no␣command␣was␣run␣");
227     failureRead();
228     setColor(200, 0, 200);
229   }
230
231   // Halt PICC
232   rfid.PICC_HaltA();
233   // Stop encryption on PCD
```

```
234   rfid.PCD_StopCrypto1();
235 }
236
237 // Dump RFID hexadecimal data to serial
238 void dumpByteArray(byte *buffer, byte bufferSize) {
239   for (byte i = 0; i < bufferSize; i++) {
240     Serial.print(buffer[i] < 0x10 ? " 0" : " ");
241     Serial.print(buffer[i], HEX);
242   }
243 }
244
245 // Return RFID hexadecimal data as a String
246 String returnByteString(byte *buffer, byte bufferSize) {
247   String hexString = "";
248   for (byte i = 0; i < bufferSize; i++) {
249     hexString += (String) buffer[i];
250   } return hexString;
251 }
252
253 // Reset all mapping relationship variables after command
254 void resetVariables(){
255   outputColor = 'g';
256   inputPressed = false;
257   outputPressed = false;
258   inputDevice = 0;
259   outputDevice = 0;
260   invertedDistance = false;
261 }
262
263 // Helper function usde for debug
264 void printVariables() {
265   Serial.println();
266   Serial.println("Input device: " + (String) inputDevice);
267   Serial.println("Output device: " + (String) outputDevice)
        ;
268   Serial.println("Color: " + (String) outputColor);
269   Serial.println("Inverted distance: " + (String)
        invertedDistance);
270   Serial.println();
271 }
272
273 // Helper function for reading analog button voltage values
274 boolean inInterval(int compare, int low, int high) {
275   return !(compare < low) && !(high < compare);
276 }
```

```
277
278 // Blink RGB led if inverse mode is active
279 void blinkColor() {
280   if (!invertedDistance) return;
281   if (newTime < oldTime + INTERVAL) return;
282
283   if (blinkHigh) changeColor(outputColor, 250);
284   else changeColor(outputColor, 0);
285
286   // Alternate on/off blinking
287   blinkHigh = !blinkHigh;
288   oldTime = millis();
289 }
290
291 // Read analog button values and direct flow
292 void readButtons(int analogValue) {
293   if (inInterval(newAnalogValue, 300, 390)) {
294     // Output button pressed
295     if (inputPressed && inputDevice != 0) { // Ready to
          read output
296       outputPressed = true;
297       setColor(0, 0, 0);
298       delay(tactileFlash);
299       changeColor(outputColor, 250);
300     } else if (inputPressed && inputDevice == 0) { // Ready
          to read unOutput
301       outputPressed = true;
302       setColor(0, 0, 0);
303       delay(tactileFlash);
304       setColor(0, 0, 250);
305       outputColor = 'b';
306     } else { // input button is not pressed or input device
          is not set
307       failureRead();
308       resetVariables();
309       setColor(200, 200, 200); // idle light
310     }
311   } else if (inInterval(newAnalogValue, 400, 500)) {
312     // MODE BUTTON PRESSED
313     if (inputDevice != 0) {
314       invertedDistance = !invertedDistance;
315       Serial.println("invertedDistance = " + (String)
            invertedDistance);
316       // If user turns off inverted distance while LED is
            LOW (while blinking)
```

```
317       // it can be stuck there. This is to make sure it's
              on HIGH
318       changeColor(outputColor, 250);
319     } else if (inputPressed && outputPressed) {
320       failureRead();
321       changeColor(outputColor, 250);
322     } else {
323       failureRead();
324       setColor(200, 200, 200); // idle light
325     }
326
327   } else if (inInterval(newAnalogValue, 600, 750)) {
328     // COLLOR BUTTON PRESSED
329     if (inputDevice != 0 || (inputPressed && outputPressed)
            ) {
330       alternateColors();
331     } else {
332       failureRead();
333       setColor(200, 200, 200); // idle light
334     }
335
336   } else if (inInterval(newAnalogValue, 900, 1025)) {
337     // INPUT BUTTON PRESSED
338     if (inputPressed) {
339       Serial.println("Input node was already set. Reseting
              variables");
340       resetVariables();
341     }
342     setColor(0, 0, 0);
343     delay(tactileFlash);
344     setColor(200, 200, 200); // idle light
345     inputPressed = true;
346   }
347   delay(400);
348 }
349
350
351 // Alternates between displaying color red, green and blue
352 void alternateColors() {
353   if (outputColor == 'r') {
354     outputColor = 'g';
355     setColor(0, 250, 0);
356   } else if (outputColor == 'g') {
357     outputColor = 'b';
358     setColor(0, 0, 250);
```

```
359   } else if (outputColor == 'b') {
360     outputColor = 'r';
361     setColor(250, 0, 0);
362   }
363   Serial.println("Color is now set to " + (String)
          outputColor);
364 }
365
366
367 // Helper function turning off all light
368 void turnOffLed() {
369   analogWrite(LED_RED, 0);
370   analogWrite(LED_GREEN, 0);
371   analogWrite(LED_BLUE, 0);
372 }
373
374
375 // Fade in and out twice with red
376 void failureRead() {
377   turnOffLed();
378   for (int num = 0; num < 3; num++) {
379     analogWrite(LED_RED, 250);
380     delay(100);
381     analogWrite(LED_RED, 0);
382     delay(100);
383   } changeColor(outputColor, 250);
384 }
385
386
387 // Fade in and out twice with green
388 void successfulUnOutput() {
389   for (int num = 0; num < 2; num++) {
390     for (int i = 0; i < 255; i += 2) {
391       setColor(0, 0, i);
392       delay(3);
393     } for (int n = 255; n > 0; n -= 2) {
394       setColor(0, 0, n);
395       delay(3);
396     }
397   } setColor(200, 0, 200);
398 }
399
400
401 // Fade in and out twice with green
402 void successfulRead() {
```

```
403   for (int num = 0; num < 1; num++) {
404     for (int i = 0; i < 255; i += 2) {
405       setColor(0, i, 0);
406       delay(3);
407     } for (int n = 255; n > 0; n -= 2) {
408       setColor(0, n, 0);
409       delay(3);
410     }
411   } for (int i = 0; i < 255; i += 4) {
412       setColor(0, i, 0);
413       delay(3);
414   }
415 }
416
417
418 // Fade in and out trice with all colors
419 void successfulCommand() {
420   for (int num = 0; num < 3; num++) {
421     for (int i = 0; i < 255; i += 2) {
422       setColor(i, i, i);
423       delay(3);
424     } for (int n = 255; n > 0; n -= 2) {
425       setColor(n, n, n);
426       delay(3);
427     }
428   } setColor(200, 0, 200);
429 }
430
431
432 // Helper function writing all colors simultaneously
433 void setColor(int colors[]){
434   analogWrite(LED_RED, colors[0]);
435   analogWrite(LED_GREEN, colors[1]);
436   analogWrite(LED_BLUE, colors[2]);
437 }
438
439
440 // Helper function writing all colors simultaneously
441 void setColor(int red, int green, int blue) {
442   analogWrite(LED_RED, red);
443   analogWrite(LED_GREEN, green);
444   analogWrite(LED_BLUE, blue);
445 }
446
447
```

```
448 // Helper function writing one color with given strength
449 void changeColor(char color, int ledValue) {
450   // color == 'r' || 'g' || 'b'
451   // 0 <= ledValue <= 255
452
453   if (color  == 'r') {        // RED
454     analogWrite(LED_RED, ledValue);
455   } else if (color == 'g') {  // GREEN
456     analogWrite(LED_GREEN, ledValue);
457   } else if (color == 'b') {  // BLUE
458     analogWrite(LED_BLUE, ledValue);
459   }
460 }
461
462
463 // Helper function configuring the Wi-Fi
464 void wifiSetup() {
465   delay(10);
466   Serial.println();
467   Serial.print("Connecting␣to␣");
468   Serial.println(ssid);
469   // Start connecting to WiFi
470   WiFi.begin(ssid, password);
471
472   // Print loading text while connecting
473   boolean waitingWifi = true;
474   while (WiFi.status() != WL_CONNECTED) {
475     delay(500);
476     Serial.print(".");
477     if (waitingWifi) {
478       setColor(100, 100, 0);
479       waitingWifi = !waitingWifi;
480     } else {
481       turnOffLed();
482     }
483   }
484   turnOffLed();
485
486   Serial.println();
487   Serial.println("WiFi␣connected");
488   Serial.print("Local␣IP␣address:␣");
489   Serial.println(WiFi.localIP());
490 }
491
492
```

```
493 void callback(char* topic, byte* payload, unsigned int
      length) {
494   Serial.print("Message arrived [");
495   Serial.print(topic);
496   Serial.print("]: ");
497   String message = "";
498   for (int i = 0; i < length; i++) {
499     Serial.print((char)payload[i]);
500     message += (char) payload[i];
501   }
502   Serial.println();
503
504   // Material programming tool should not subscribe
505   // to any topics or receive any messages
506
507 }
508
509 // Reconnect to MQTT broker if connection is lost
510 void reconnect() {
511   // Loop until reconnected
512   while (! client.connected()) {
513     Serial.println("Attempting MQTT connection...");
514     // Attempt to connect
515     String clientName = "Tool";
516     //Serial.println("Client name: " + clientName);
517     if (client.connect(clientName.c_str())) { //clientName.
          c_str())) {
518       Serial.println("Connected to MQTT Broker at " + (
            String) mqttServer);
519
520       // Material programming tool should not subscribe
521       // to any topics or receive any messages
522
523     } else {
524       Serial.print("failed, rc=");
525       Serial.print(client.state());
526       Serial.println(" try again in 5 seconds");
527       // Wait 5 seconds before retrying
528       delay(5000);
529     }
530   }
531 }
532
533
534 // Remove mapping relationship for given output node and
```

```
    color
void publishUnOutput() {
  String unOutputMessage = (String) outputColor;
  String unOutputTopic = (String) outputDevice + "-unoutput
      ";
  Serial.println("UnOutput␣[" + unOutputTopic + "]:␣" +
      unOutputMessage);
  publishMessage(unOutputMessage, unOutputTopic);
}


// Publish a mapping relationship command to input and
    output
void publishCommand() {
  char invert;
  if (invertedDistance) invert = 't';
  else invert = 'f';
  String outputMessage = (String) inputDevice + "-" + (
      String) outputColor + invert;
  String outputTopic = (String) outputDevice + "-output";
  String inputMessage = (String) outputDevice;
  String inputTopic = (String) inputDevice + "-input";
  Serial.println("Output␣message␣[" + outputTopic + "]:␣" +
       outputMessage);
  Serial.println("Input␣message␣[" + inputTopic + "]:␣" +
      inputMessage);
  publishMessage(outputMessage, outputTopic);
  publishMessage(inputMessage, inputTopic);
}


// Publish MQTT message
void publishMessage(String message, String topic) {
  /* We start by writing our data as formatted output to
      sized buffer (using snprintf)
   * In this case we use an integer, and we must format it
       as such (%d)
   * There are different formatting options based on the
       data type:
   *
   * %lu unsigned long
   * %ld signed long
   * %d integer
   * %f float
   * %s string
```

```
570     * %c char
571     */
572   snprintf (msg, 75, "%s", message.c_str()); // Store our
        data 'tagId' in the sized buffer 'msg'
573   Serial.print("Publish message: ");
574   Serial.println(msg);
575   client.publish(topic.c_str(), msg);
576 }
```