HELLENIC MEDITERRANEAN UNIVERSITY

SCHOOL OF ENGINEERING AND DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

PROGRAM OF STUDIES: INFORMATICS ENGINEERING

THESIS

TITLE: KLSTR MOBILE CONTROLLER

MARIA PAVLOU (4140)

Advisor: Papadourakis Georgios

ABSTRACT

The purpose of this thesis is the designing and development of an application for the management of lighting systems in large spaces, such as festivals, theaters, theme parks, television studios, etc. Lighting technicians face several problems as in such large events many lights are needed, which they must properly adjust the position, direction, color, intensity, etc. In this thesis, the goal is to create a system that will be integrated into the lighting device and a mobile application.

The system we will create consists of a screen through which one can make appropriate settings and an NFC antenna through which data exchange will be done with the mobile application. Either one can make settings through the screen and then bring the phone to the NFC antenna to update the application, or make changes to the settings through the application and send the data to the NFC antenna.

ΣΥΝΟΨΗ

Σκοπός αυτής της εργασίας είναι ο σχεδιασμός και η ανάπτυξη μίας εφαρμογής για τη διαχείριση των συστημάτων φωτισμού σε μεγάλους χώρους, όπως σε φεστιβάλ, θέατρα, θεματικά πάρκα, τηλεοπτικά στούντιο κ.ά. Οι τεχνικοί φωτισμού αντιμετωπίζουν αρκετά προβλήματα καθώς σε τόσο μεγάλους χώρους χρειάζονται πολλά φώτα, σε καθένα από τα οποίο πρέπει να ρυθμίσουν κατάλληλα τη θέση, την κατεύθυνση, το χρώμα, την ένταση κ.ά. Σε αυτή την εργασία, στόχος είναι η δημιουργία ενός συστήματος που θα ενσωματωθεί στη συσκευή του φωτός και μίας εφαρμογής για το κινητό.

Το σύστημα που θα δημιουργήσουμε αποτελείται από μία οθόνη μέσω της οποίας θα μπορεί κάποιος να κάνει κατάλληλες ρυθμίσεις και μία κεραία NFC μέσω της οποίας θα γίνεται η ανταλλαγή δεδομένων με την εφαρμογή για το κινητό. Είτε μπορεί κάποιος να κάνει ρυθμίσεις μέσω της οθόνης και στη συνέχεια να φέρει σε επαφή το κινητό με την κεραία NFC για να ενημερωθεί η εφαρμογή, είτε να κάνει αλλαγές στις ρυθμίσεις μέσω της εφαρμογής και να αποστέλλει τα δεδομένα στην κεραία NFC.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOLEDGEMENTS

Firstly, I would like to thank my professor and thesis advisor Dr. Georgios Papadourakis for giving me the opportunity to participate in the Blended Academic International Mobility project and for his guidance during it. It was a great experience that provided me with many skills that will be useful in the future.

Also, I would like to thank the students and all the professors of the Blended AIM 2019 for the great experience of working as a team on a big project for a start-up company.

I would also like to thank my colleague and friend with whom I worked on this project, Dosu Jeremiah. Despite the difficulties, we worked together perfectly with full understanding and support.

DEDICATION

This thesis is dedicated to friends and family for their endless love, support, and continuous encouragement throughout my years of study and through the development and writing of this thesis. This would not have been possible without them.

CHAPTER 1: INTRODUCTION

1.1 Summary

This thesis is part of an international Erasmus + program called Blended-AIM (Academic International Mobility), which involves universities and educational institutions from all over Europe. Two groups of ten people are formed and are invited to carry out two different tasks for two newly established companies.
The goal of my team was to design and create an application and a system for managing lighting systems at large events. To achieve this, my team split into a hardware team, a software team, a design team, and a marketing and logistics team.

1.2 Motive

Our application is aimed at lighting technicians, system technicians, and generally people who deal with such lighting systems. Our goal is to solve the problems and limitations that technicians face in many events, festivals, theaters, TV studios, etc., where many lighting systems must be adjusted manually. For example, in a big concert, if one of the lights stops working, someone has to find where it is, go to it and adjust it manually. These are some of the problems that our application tries to solve, as well as reduce the possibility of human error and offer a new idea to the market.

1.3 Blended-AIM (Blended Academic International Mobility)

Blended AIM (Academic International Mobility) is an Erasmus+ funded project made to promote students' employability and support companies hosting internships. Every year 10 educational institutes from European countries like Portugal, Greece, Belgium, United Kingdom, Germany, Iraq, and Austria end up to 2 students each to form a team. The purpose of that is to support the students develop soft skills in an international environment through blended mobility and the teams to develop and present a prototype or a proof of concept for a given project. The students will participate for a semester from abroad, communicate virtually and they will meet for about 2 weeks.

At this moment, a student's professional career depends on mobility and demands certain intercultural skills. The demands on the job market are very high and every student must be able to be competitive. Blended mobility helps the students adapt and learn but it's hardly considered, let alone used, a solution to international mobility's problems. Blended Aim sets the foundation to promote and test blended mobility by providing the resources, training, supporting tools, and information to the students and the companies that host internships. Its purpose is to enable students to work in a real work environment on a real problem. To get in touch with companies, to talk to those who had a unique idea and want to implement it. Participants acquire knowledge about intercultural competence, the relationship between societies, increased awareness of stereotypes and can built curiosity, self-awareness, meaningful relationships with people from different countries.

1.3.1 The history of Blended Mobility

At the beginning of the year, students from ten educational institutions (two from each) gather to undertake a project given by a company. This project is considered as a course and each student receives ECTS after the completion of the project. Students are from different fields of study such as computer science, graphic design, business management so that the project can be completed. The project is usually a product that helps solve some of the company's problems. During the implementation of the work, there are two important meetings. In the first, students meet with each other, meet professors from all institutions and company representatives to discuss how to approach the project and how to work it effectively. In the second, the students present their product to the professors and the representatives of the company and then they are evaluated by them. During the time between the two meetings, the students hold online meetings to present their work and discuss with each other the tasks that are to be done and even the problems they may face in them.

1.3.2 The methodology of Blended Mobility

The preparation of the project starts at the first meeting, where the students get to know each other and the company presents the problem and the challenges. Students have to cooperate, think and discuss the issue and how they could solve it. Every day passes together, the students and the company prepare how they will work, they divide the teams based on each individual's specificity and they define the problem that each group should manage. The group is divided into subgroups according to the subject of study of each member and they undertake to carry out a part of the project. At the end of the week spent analyzing the problem and its solution method, the first presentation of the project they have undertaken and will implement, is made.

Then and until the next meeting, there are weekly meetings of the whole team and the subgroups to present the progress they have made and to discuss which problems have arisen. At the end of the semester, the team meets again and the final product is presented.

Initially Blended Aim invites companies to participate and present to them the advantages they could have by outsourcing their project to a group of international students and their supervisors. The project that will stand out should be mentioned in the development of a product, a service, a proof of concept or a prototype which should be analyzed in terms of marketing design and programming. It is very important that at least one person from the company should be available on a weekly basis to communicate with the students, and to supervise the project. [1]

What Blended Aim does is bring together students from different universities, different specialties, and nationalities to not only find a way to solve a problem but also to collaborate and communicate. This is often accomplished by creating Scrum teams and enforcing a

master scrum to supervise team operations. The teams are organized hierarchically with the project owner who performs the role of administrator and communicates with the representative of the company, the supervisors and the team members working on the part of the project that they have undertaken according to their field of education background. There are also subgroups depending on the subject of study, where each has an Assistant Project Manager to communicate with the project owner.

The acquaintance and communication of the students before the presentation of the project is very important as it helps for the smooth transition of the students to the role they have to serve. A good way to get to know each other is for students to create a website with their personal information to send to the company. Also, before the presentation of the project by the company to the students, some activities are organized such as a game for the culture of each people or the tools that will be used as SCRUM as not everyone is familiar with them.

Blended Aim is a project course unit during which students meet face-to-face in two meetings and work as a team at their home institutions between. It is a distributed course unit running simultaneously at a several distinct institutions for a full semester. It tears down basrriers to international mobility, thus promoting equity and equal opportunities, in a sustainable way since it reduces the number of travels during a mobility period.

CHAPTER 2: FUNDMENTALS

2.1 Analysis and Development Methods

2.1.1 Nexus Scrum

Software development is a complex and difficult task. Even companies face difficulties in the development department and these create problems for them as they progress in their work. The Scrum framework can help, but it is not enough on its own. To go one step further, the Nexus framework is also required. The Nexus Framework is a framework, based on the Scrum and Agile Manifesto principles that help developers minimize group outbursts and integration issues. Developed by Ken Schwaber, co-founder of Scrum and founder of Scrum.org in 2015. Scrum is an agile methodology where products are constructed in a series of constant length repetitions. There are four pillars to this structure: sprint design, stand-ups, sprints, and retrospectives, meaning that you can access previous events. [2]

The Nexus Framework is an extension of the Scrum framework and uses an iterative and step-by-step approach to software scaling and product development. Allows multiple Scrum groups working on a product to be integrated into a single larger group. The Nexus framework consists of 3 to 9 scrum teams working at the same time to successfully develop the product. The main goal is to identify issues between groups and ensure the understanding and use of integration tools. The integration team is a new feature in the Nexus framework and is the team responsible for the successful integration of all tasks created by all Scrum teams into one Nexus. Consists of a product owner, a master Scrum and a few members of each scrum team on a Nexus. Each scrum group must be represented by a team member. Representatives then meet to identify and resolve problems that have arisen and to set goals that they want to achieve in each Sprint. These steps explain the term "Sprint Nexus Design" that is critical to a Nexus framework. In addition, Nexus Daily Scrums helps teams plan the next steps correctly by controlling the completed task. Nexus Daily Scrums is a short meeting between Scrum team members, where not everyone needs to be present, but the representative of each team is enough.

2.1.2 Jira

Jira Software is an application used for issue tracking and project management. Originally designed by Australian software company Atlassian, as a bug and troubleshooter, Jira has evolved into a powerful task management tool for all types of usage, from requirements and management issues to flexible software development. [3]

Jira for teams using agile (flexible) methodologies

Jira provides scrum tables, which are task management nodes where nodes are mapped to custom workflows. Boards provide transparency in teamwork as everyone has access to and visibility into the status of each work item. Time tracking capabilities and real-time performance reports (charts, sprint reports) allow teams to closely monitor productivity throughout the working time range. Teams can start with one type of project or create their custom workflow. Jira issues, also known as tasks, monitor each piece of work that must go through the steps of the workflow to be completed. Customizable permissions allow administrators to specify who can see and perform which actions. With all the project information in place, reports can be created to track progress, productivity and ensure that nothing goes wrong. Scheduling sprints determine what the team should complete in the next sprint from the list of all tasks to be done (backlog). Jira makes this to-do list the center of sprint design, so you can adjust the sprint range, control the speed, and prioritize the tasks.

2.1.3 Slack

Slack is a cloud computing platform founded by Stewart Butterfield and launched as an internal tool for his company, to develop a more inactive online game. It is a messaging application for business, which helps people to work as one team. [4]

Characteristics

Slack offers many IRC (International Relay Chat) activities, which is an application-level protocol that facilitates text-to-speech communication, including regular chat rooms (channels) organized by topic, private groups, and lives messaging. In Slack, you can search for files, chats, and people, and add emojis to your messages, which other users can then click to express their reactions. Also, in its free program, it allows view and searching only the 10,000 most recent messages.

Groups

Slack groups allow members to join the workplace through a specific URL or invitation sent by a group administrator or owner. Although Slack was developed for business and organizational communication, it has been adopted as a community platform, replacing message boards or social media groups.

Conversations

Public channels allow team members to communicate without the use of email or SMS. They are open to everyone in the conversation provided they have been invited to participate first. Private channels allow private chat between smaller groups than the total group. These can be used to divide large groups into their respective projects. Instant messaging, which can include up to 9 people, allows users to send private messages to a specific user rather than to a group of people. Once started, a group of instant messages can be converted to a private channel.

Incorporations

Slack integrates with many third-party services and supports embedded communities, such as Google Drive, Trello, Dropbox, Box, Heroku, IBM Bluemix, Crashlytics, GitHub, Runscope, Zen desk, and Zapier. December 2015, Slack launched the list of software applications, which consists of 150 integrations that users can install. In 2018 Slack announced a partnership with the financial and human resources management company Workday. This integration allows Workday clients to access Workday features directly from Slack.

API

Slack provides an application programming interface (API) for users to create applications and automate processes such as sending notifications under specified conditions and automatically creating internal support. The Slack API has been recognized for its compatibility with many types of applications, frameworks, and services.

Platforms

Slack provides mobile applications for iOS and Android, in addition to Web browsers and macOS, Windows clients (with versions available from the company website and the Windows Store), and Linux. It is also available for the Apple Watch, allowing users to send instant messages, view reports, and make simple replies. It also appeared on the Apple Watch home screen in a 2015 promotional video.

Business model

Slack is a freemium product, a basic product or service is provided free of charge but one is charged for additional features that extend the functionality of the free version of the software. Key paid features include the ability to search over 10,000 archived messages and the addition of unlimited applications and embeds. They claim unlimited user support, but when freeCodeCamp attempted to change its community of more than 8,000 Slack users in 2015, they encountered several technical issues and were informed by Slack support to limit their channels to no more than 1,000 users (ideally about 500). On July 26, 2018, Atlassian announced the closure of its competitors, HipChat and Stride, from February 11, 2019, and the sale of their intellectual property to Slack, with Slack taking over the user base of the services. The companies also announced their commitment to work to integrate Slack with Atlassian services.

CHAPTER 3: WORK PLAN

3.1 State of the art

The purpose of the application and the system that we will create is that a lighting technician can have full control in his hands, for each of the lighting systems in which he is responsible. Through the system consisting of an Arduino, a screen, and an antenna, he can be informed about various features such as position, direction, color, and intensity for one of the lighting systems. Through the screen, he can be informed about them but also to configure them, while through the antenna this data is sent from the system to the mobile application so that it can be checked directly from the device.

3.2 Technologies

The technologies we used as a group of hardware to create the system that will be the lighting device are an Arduino Uno for system programming, an LCD screen for displaying the menu, and an NFC antenna for communicating with the application on the mobile. Below, information on the technologies mentioned are given.

3.2.1 Arduino Uno

The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. [5] The board has 14 digital I/O pins (six capable of PWM output – method of reducing the average power supplied from an electrical signal, cutting off efficiently in separate parts), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment). It can be powered by USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts.

The word "uno" means "one" in Italian and was chosen to mark the initial release of Arduino Software. The Uno board is the first in a series of USB-based Arduino boards. The ATmega328 on the board comes preprogrammed with a bootloader that allows uploading new code to it without the use of an external hardware programmer. It is an easy-to-use device and accessible to everyone as it is quite economical. Whether for beginners or advanced it is a very good application for many different projects such as blink a led or a a security alarm system.

In Figure 1, Arduino Uno is presented.

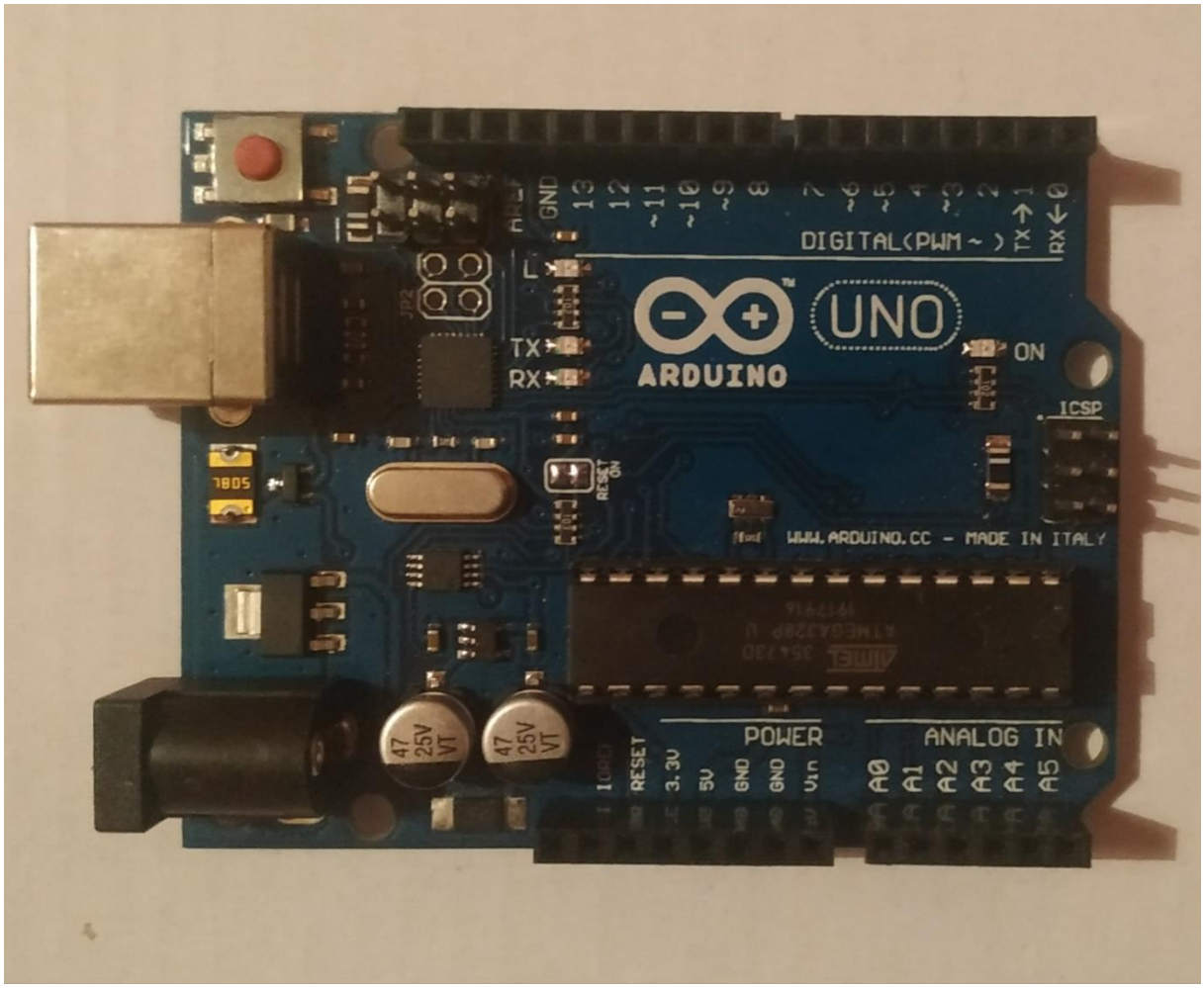

Figure 1. Arduino UNO

Technical specifications

The main technical features of the Arduino as shown in the Table 1, are that it has high efficiency with low power requirements as its operation is at 5V, it has a USB connection to be easily connected to a computer and transfers the code to the controller using Arduino IDE, as well as a barrel plug connector that works with a standard 9V battery. It can also use external power source up to 12V which can be regulated to 5V or 3.3V depending on the project requirements. [6] The 20 pins that it has are useful in many different functions such as UART, I^2C, SPI as well as analog signals. The analog pins measure from 0 to 5V, but they can get programmed to use the high ranger using the function analogReference() and AREF pin.

On the board there is the microcontroller ATMega328P that comes with timers, counters, interrupts, PWM, CPU, I/O pins and based on 16MHz clock which produces more numbers of cycles. There is also a built-in regulation which keeps the voltage low when the device is connected to external device.

The microcontroller on the Arduino has EEPROM memory (Electrically Erasable Programmable Read Only Memory), which is a non-volatile memory and can store byte variables that are kept even when we reset or power off the Arduino. EEPROM is permanent storage similar to a hard drive-in computer, that can be read, erased and rewritten through the EEPROM library. In Arduino Uno and ATMEga328P microchip there is a 1Kbyte EEPROM, however there is a provision of Micro SD card to extend the memory. Lifetime of EEPROM is 100.000 write cycles but unlimited reads.

Table 1. Technical specifications

Microcontroller	Microchip ATmega328P
Operating Voltage	5 Volts
Input Voltage	7 to 20 Volts
Digital I/O Pins	14 (of which 6 can provide PWM output)
PWM Pins	6 (Pin # 3, 5, 6, 9, 10 and 11)
UART	1
I2C	1
SPI	1
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g
ICSP Header	Yes
Power Sources	DC Power Jack & USB Port

General functions of I/O pins

Arduino has the following Input and Output pins, which perform some basic functions such as gets power, to have ground but also to reset when this is necessary.

LED: There is a built-in LED driven by digital pin 13.

VIN: The input voltage to the Arduino board when it is using an external power source. You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

5V: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 20V), the USB connector (5V), or the VIN pin of the board (7-20V).

3V3: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

GND: Ground pins.

IOREF: This pin on the Arduino board provides the voltage reference with which the microcontroller operates.

Reset: It is used to restart the program from the first line of its sketch.

Special functions of I/O pins

Each of the 14 digital pins and 6 analog pins on the Uno can be used as an input or output, under software control.. Each pin can provide or receive 20 mA as the recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50K ohm. The Uno has 6 analog inputs, labeled A0 through A5, each provides 10 bits of resolution (i.e. 1024 different values). By default, they measure from ground to 5 volts, though it is possible to change the upper end of the range using the analogReference() function. [7][8]

In addition, some pins have specialized functions:

Serial / UART: pins 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL serial chip. UART is a computer circuit that mediates the serial communication of computer or computers with devices. [9]

External interrupts: pins 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

PWM (pulse-width modulation): pins 3, 5, 6, 9, 10, and 11. Can provide 8-bit PWM output with the analogWrite() function. PWM is a method of reducing the average power provided by an electrical signal by effectively cutting it into separate parts. The average voltage value supplied to the load is controlled by turning the switch between power supply and load on and off at a fast rate. The more the switch is turned on compared to the off periods, the greater the total power supplied to the load. PWM is particularly suitable for an inertial load operation, such as motors, which are not so easily affected by this discrete switching because their inertia causes them to react slowly. [10]

SPI (Serial Peripheral Interface): pins 10 (SS), 11 (MOSI), 12 (MISO), and 13 (SCK). These pins support SPI communication using the SPI library. SPI is a modern standard of serial communication interface used for short-distance communication, mainly in embedded systems. [11]

TWI (two-wire interface) / I²C: pin SDA (A4) and pin SCL (A5). Support TWI communication using the Wire library [12]. The I²C bus is a serial bus used to connect low-speed peripherals to motherboards, embedded systems, mobile phones, or other electronic devices.

AREF (analog reference): Reference voltage for the analog inputs.

3.2.2 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, as it is a text editor. Ii is used for writing code, compiling it and check if any errors are there when uploading the code to the board. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main(). The user can use the app whenever he wants as it is open source and can also make his own modules and add them. User writes code, compiles it and the IDE generates a Hex file which is sent to the board through the USB cable.

With the rising popularity of Arduino as a software platform, other vendors started to implement custom open source compilers and tools (cores) that can build and upload sketches to other microcontrollers that are not supported by Arduino's official line of microcontrollers. [13]

Arduino IDE is presented in Figure 2, as looks before coding. There are the setup and loop functions.

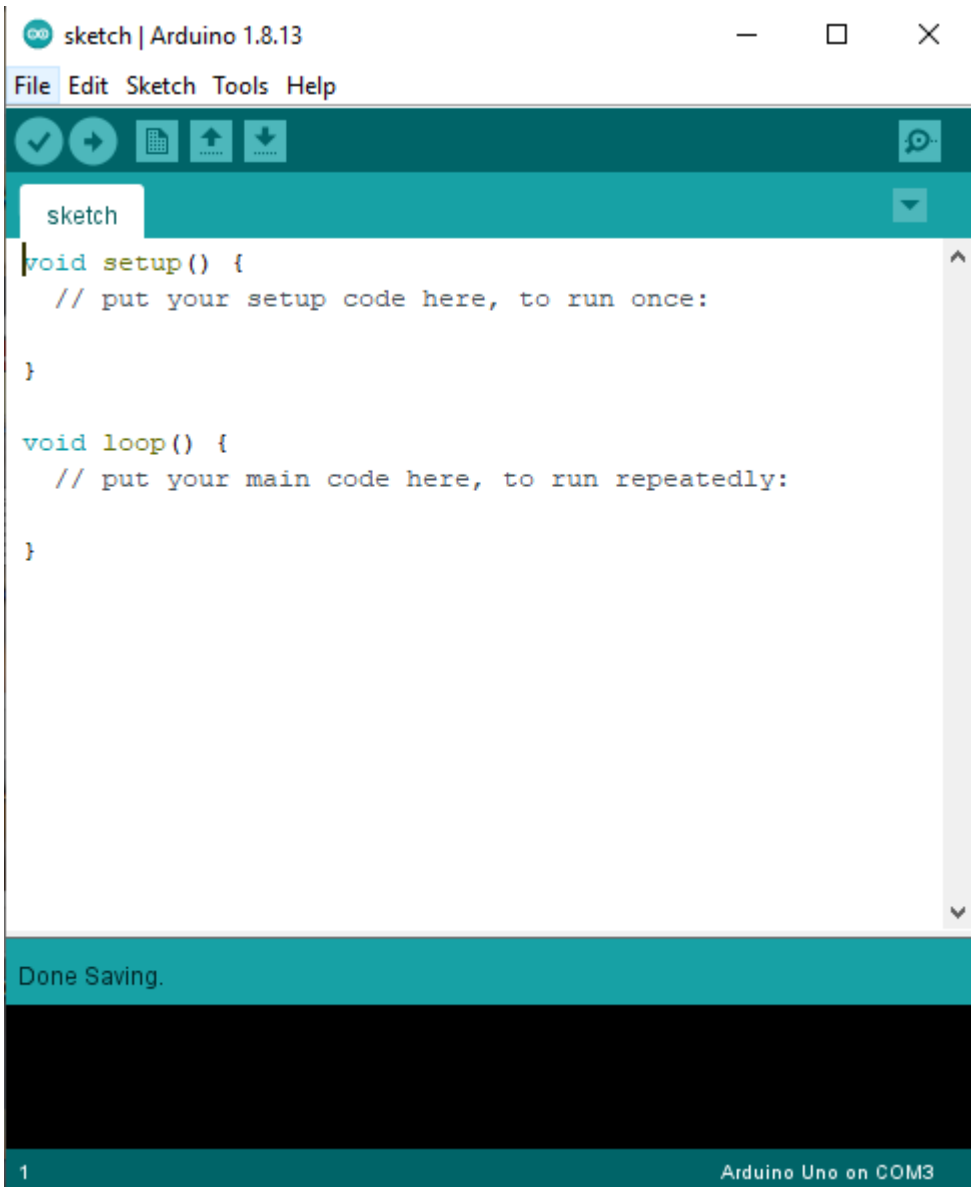

Figure 2. Arduino IDE

3.2.3 Liquid-crystal display (LCD)

The Liquid Crystal Display (LCD) Keypad Shield is a screen 80x58mm that locks directly on the Arduino Uno. Designed for Arduino compatible boards, it provide a user-friendly interface that allows users to have a menu, make choices on it, and more. It works perfectly in 4-bit mode by the Liquid Crystal library found in the Arduino IDE, with which we can control the LCD buttons. [14]

Specification and Interfaces

In the Arduino, the 4, 5, 6, 7, 8, 9 and 10 terminals are used to interface with the LCD screen, as shown in Table 2, and the analog terminals are used to read the buttons. Through its potentiometer, it supports contrast adjustment and on/off function with a backlight. Includes 6 momentary push buttons, up, down, right, left, select and reset and a screen of 2 rows and 16 columns, so it can display 32 characters. By the arrows keys we can create a functional menu control panel and reset button is used for resetting Arduino program. The keypad interface uses only one ADC channel to store digital input/output terminals, so that one pin reads which one of the 5 switches is pressed. The values are read through a 5-phase voltage divider and it has expanded available I/O pins. [15] In Table 2, there are the LCD Interfaces.

Table 2. LCD Interfaces

Pins	Functions
Analog 0 (A0)	Buttons(Select, Up, Right, Down, Left)
Digital 4 (D4)	DB4
Digital 5 (D5)	DB5
Digital 6 (D6)	DB6
Digital 7 (D7)	DB7
Digital 8 (D8)	RS (Choose Data or Signal Display)
Digital 9 (D9)	Enable (read/write data)
Digital 10 (D10)	LCD Backlight Control

DB4, DB5, DB6, and DB7 are used to transfer and receive data between the microcontroller and the monitor. Also, all 5 buttons are connected to the analog A0 terminal to save digital terminals. To read them, the ADC must be used. When a button is pressed, a value is returned to terminal A0 according to the internal resistor circuit, which determines the type of key. That is, if the value that reaches A0 is from 0 to 60, the button that is pressed is the Right, if it is from 61 to 200 it is the Up, if it is from 201 to 400 it is the Left and finally if it is from 601 to 800 is the Select button.

LCD Keypad Shield, which has the 16X2 Screen and 6 buttons in total, is presented in Figure 3.

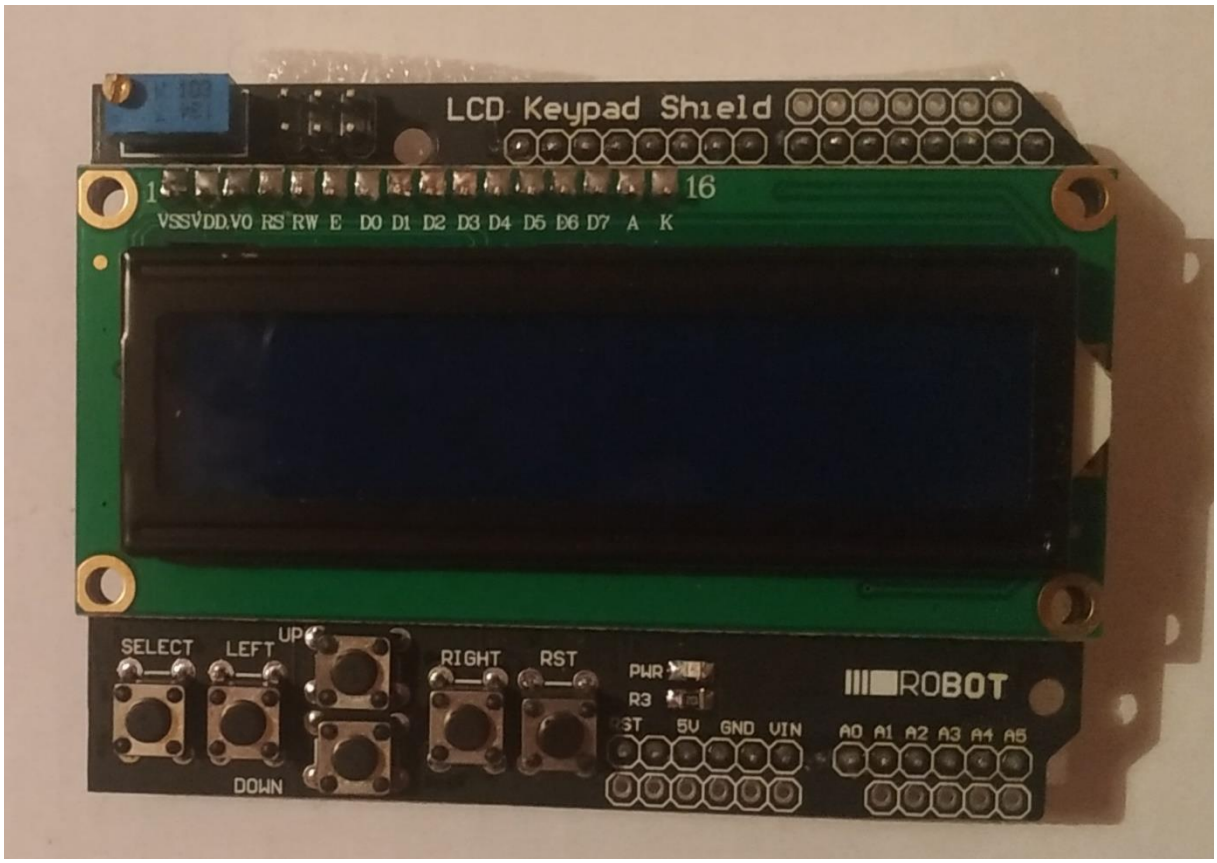

Figure 3. LCD Keypad Shield

3.2.4 LED

LEDs or light-emitting diodes are small electronic components. When the LEDs are energized, they emit light in various colors, such as red, green, or blue. If the current passing through the diode is too high, it may damage the LED. To limit the current flowing through the diode, it is common practice to add a resistor to the circuit, as shown in the Figure 4 below. [16]

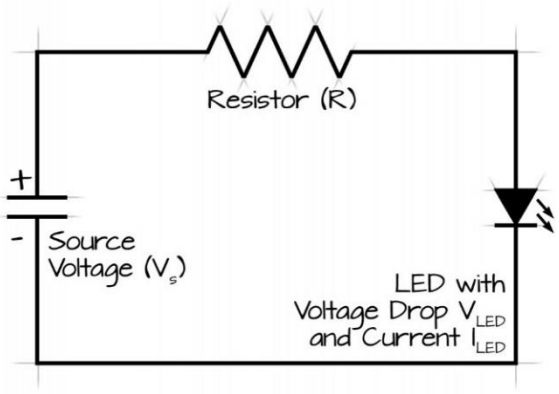

Figure 4. LED Circuit

Characteristics

LEDs are solid-state devices where the light which is generated by them is directional in around 60 degrees, there are in different colors or RGB light mix depends on the wavelength and they have low energy consumption. There are no fragile parts on them, as a conventional light bulb, to be broken so they rarely burn out. Instead, the gradual degradation of its outpout becomes the dominant failure mode of LEDs. The speed of lumen depreciation is closely relevant with the device's junction temperature, which represents the temperature of the point where an individual diode connects to its substrate. Lower junction temperature leads to higher light output and slower lumen depreciation. Therefore, junction temperature is the key parameter for evaluating LED products' life span.

- 1.8-2.4VDC forward drop
- Max current: 20mA
- Suggested using current: 16-18mA
- Luminous Intensity: 20mcd
- Light current <0.1 lm
- Angle of radiation 60 °
- Forward Current 30 mA
- Reverse Voltage 5 V
- Operating Voltage max. 2.50 V
- Normal current 20 mA
- Power Dissipation max. 105 mW

3.2.5 NFC

Near-field communication (NFC) is a set of communication protocols for communication between two electronic devices over a distance of 4 cm or less. By placing two devices close to each other, a virtual reaction is made. NFC devices can act as electronic identity documents and keycards, and are used in contactless payment systems that allows anyone to pay by his phone than the credit card.

Near-field communication (NFC) describes a technology that can be used for the contactless exchange of data over short distances. NFC-enabled portable devices can be provided with application software, for example, to read electronic tags or make payments when connected to an NFC-compliant system. These are standardized to NFC protocols, replacing proprietary technologies used by earlier systems.
Like other "proximity card" technologies, NFC is based on inductive coupling between two so-called antennas present on NFC-enabled devices—for example, a smartphone and a printer—communicating in one or both directions. [17]
The NFC chip works with an antenna with a spiral, which generates an electromagnetic field through which data can be transferred. The device which receives the data, also take instruction on what to do with it. It could take a phone call, open a web page etc.

NFC tags are passive data stores which can be read, and under some circumstances written to, by an NFC device.. Applications include secure personal data storage (e.g. debit or credit card information, loyalty program data, personal identification numbers (PINs), contacts). NFC tags can be custom-encoded by their manufacturers or use the industry specifications.

3.2.6 I²C

I^2C (Inter-Integrated Circuit) is a bus interface connection protocol for serial communication, which invented in 1982. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance. I^2C is appropriate for peripherals where simplicity and low manufacturing cost are more important than speed. I^2C uses only two bidirectional open-collector or open-drain lines. Through the serial data line (SDA) data transfer takes place and serial clock line (SCL) carries the clock signal. Both of them are pulled up with resistors. Typical voltages used are +5 V or +3.3 V, although systems with other voltages are permitted. [18]

With I^2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves. Like UART communication, I^2C only uses two wires to transmit data between devices. It is a bidirectional bus that is used in three data transfer speeds, standard fast and high speed mode.
I^2C is a serial communication protocol, so data is transferred bit by bit along a single wire (the SDA line). Like SPI, I^2C is synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave. The clock signal is always controlled by the master.

With I^2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. [19] The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame.

Addressing

I^2C to let the slave know that data is being sent to it, uses addressing. The address frame is always the first frame after the start bit in a new message. The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address. If the address matches, it sends a low voltage ACK bit back to the master. If the address doesn't match, the slave does nothing and the SDA line remains high.

Read/Write Bit

The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level.

The data frame

After the master detects the ACK bit from the slave, the first data frame is ready to be sent. Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully. The ACK bit must be received by either the master or the slave (depending on who is sending the data) before the next data frame can be sent. After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission.

ADVANTAGES

- Only uses two wires
- Supports multiple masters and multiple slaves
- ACK/NACK bit gives confirmation that each frame is transferred successfully
- Hardware is less complicated than with UARTs
- Well known and widely used protocol

DISADVANTAGES

- Slower data transfer rate than SPI
- The size of the data frame is limited to 8 bits
- More complicated hardware needed to implement than SPI

3.2.7 NTAG I²C Antenna

The NTAG I²C plus combines a passive NFC interface with a contact I²C interface. Designed to be an enabler for NFC in home-automation and consumer applications, this connected NFC tag is a fast, cost effective way to add tap-and-go connectivity to just about any electronic device. These devices maintain full backward compatibility with first-generation NTAG I²C while adding advanced features for password protection, a full memory-access configuration from both interfaces, and an originality signature for protection against cloning.

Designed to be the perfect enabler for NFC in home-automation and consumer applications, this feature-packed, second-generation connected NFC tag is the fastest, least expensive way to add tap-and-go connectivity to just about any electronic device. NXP NTAG I²C plus is a family of connected NFC tags that combine a passive NFC interface with a contact I²C interface. As the second generation of NXP's industry leading connected-tag technology, these devices maintain full backward compatibility with first-generation NTAG I2C products, while adding new, advanced features for password protection, full memory-access configuration from both interfaces, and an originality signature for protection against cloning. The second-generation technology provides four times higher pass-through performance, along with energy harvesting capabilities, yet NTAG I²C plus devices are optimized for use in entry-level NFC applications and offer the lowest BoM of any NFC solution. I²C and NFC communications are based on simple, standard command sets. All that is required is a simple antenna design, with no or only limited extra components. NTAG I²C plus development board is certified as NFC Forum Type 2 Tag. [20]

The NTAG I²C has 6 pins to communicate with the Arduino, as shown in Figure 5.

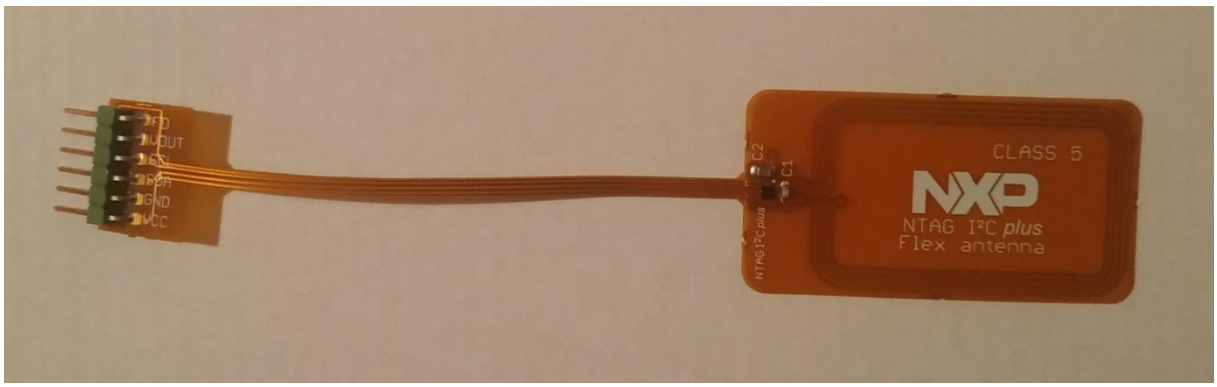

Figure 5. NTAG I²C plus Flex antenna

CHAPTER 4: MAIN PART

4.1 Problem Analysis

KLSTR is a Belgian start-up company founded by Roel Apers, Roel Velkeneers, and Wouter Moors, specializing in the setup and configuration of stage lights for many events. Over the years, they have gained recognition in setting the stages as their purpose is to create a faster and more advanced lighting management system for large and demanding events.

KLSTR's solution aims to create a Fixture, as each light is called, more autonomous and efficient. So it will not take long, as it is now, to configure each one or find and solve a problem. Anyone will be able to configure the Fixtures via his mobile phone or computer and find out where the problem is.

The system called to be implemented by the KLSTR team will have a unique ID to identify each Fixture and an integrated NFC system, through which it will get connected to the mobile of a manager. On the mobile, there will be a suitable application for the management of the systems and their control. Through this application, anyone will be able to control and process the data of each Fixture separately.

4.1.1 Problem Description

During our first meeting in Ghent, the CEO of the company described to us the difficulties that every lighting technician faces during his work. The main problem with this is that the technology that exists at the moment to manage them is very old and limited in capabilities. When building a big light show, technicians want to check if devices work properly before they pull the devices up. And adjust some basic configuration settings. Everything related to the control of these lights is based on a 33-year serial protocol. Thus, the work of the technician becomes time-consuming and difficult. For this reason, the KSTR project was created, to evolve and advance these technologies. The power of KLSTR is based on the integration of a managed Ethernet switch in every device. By combining the network information with the information of the devices, the user has access to all segments of his setup. Our computer-based application displays all this info for the user in an easy-to-use interface while in the background the KLSTR technology handles all complex setup steps to make it plug and play. So we want to create a mobile application, which is the perfect platform to execute these small tests that technicians must do before each show.

So, the Blended AIM team was called to give some ideas on who can we expand the possibilities of KLSTR. After discussing many ideas with the CEO of Constell8, we decided to develop a system on the lightning and a mobile app that will be connected via an antenna for data exchange. Any changes made by the mobile app will be applied directly to the lighting system or any changes made by the system installed in lightning will be transferred to the mobile app. This data exchange will be done through NFC tags, by a pass of the mobile to the NFC tag of the system in the light.

4.1.2 System Requirements

To complete the project, we divided the main part into smaller ones for better management. We were divided into a Software group for the creation of the mobile app and a Hardware group for the creation of the system that will be integrated into the lighting. As a Hardware team, we discussed the system requirements at length, examining the pros and cons of each option. The original idea was to use Raspberry Pi instead of Arduino as it is a ready-made computer, easier to use. But we chose the Arduino because the Raspberry Pi is much more expensive and needs peripherals (keyboard, mouse). Also, for data transfer between the device and the mobile app we had to choose between Wi-Fi, NFC, and Bluetooth. As we saw with the CEO of the company, Wi-Fi would be quite complicated for such a project and Bluetooth has quite a wide range for the project we want, as this would not be safe. In addition to connecting via Bluetooth, we should also create a security protocol. So, we come up with ideas, and to see them flush it out, it's really fun.

4.2 Implementation plan

In Blended AIM 2019, 22 students participated from universities in Germany, Portugal, Greece, Belgium, Scotland, Nigeria, and Iraq. Students were divided into 2 teams and worked on different projects. At first, the 2 teams had to get acquainted with each other and also with the Scrum framework that they would use for the agile software development. Following the Nexus Scrum, they rearranged the initial team into separate smaller ones consisting of the design team, the software team, the hardware team, the business team, and the marketing team. There were even smaller sub-teams in the development team to make the software development faster. The rearrangement was done according to the student's educational field. The first Sprint took place in Ghent, Belgium (IMEC company headquarters). During this week we made the project goals more clear and we discussed the technologies that would be used and on the final product with Contell8's CEO. After that meeting, each student returned to his country and that made the project management a bit harder, mostly on communication. Every 2 weeks, on Monday the students had a general meeting where all the students from all the teams participated and there was an update from every member about their accomplishments and their future goals. In this meeting would be present one or both of the founders of KSTR to discuss any problem that would exist. Also, each group held separate weekly meetings to discuss the work that has been done, the work that is to be done next week, and how they will separate it. As a Hardware team, we met every Friday with Dosu. Those meetings were very important to discuss the problems we had on the development and find solutions.

The whole project was divided into 8 sprints, one every two weeks, until its completion. There was daily communication between the students through the Slack application. Meetings were hosted on Whereby firstly and then at Zoom. That type of communication needed some time to be adopted by all the members but after some sprints, everyone was used to it and the communication was really direct. Although, it was really difficult to find a convenient time to

schedule the meetings to cause all the students had different time zones and daily routines. All the tasks were held on Jira and every scrum team had its tasks. At the end of each Sprint, each team had to assign new tasks for the upcoming Sprint. When a task wasn't completed till the end of a Sprint, it was transferred to the next Sprint and marked as a priority. That meant it has to be done as soon as possible before moving to the new tasks.

Due to the pandemic Covid-19, we did not manage to meet again for the final presentation, but we organized an online presentation where both teams participated, the teachers, the startups, and the IMEC representatives. At the online meeting, each subgroup presented the progress it had made. Introduction to the parts of the application such as the mobile application, the system with the LCD screen, the marketing plan, and the business plan was presented. At the end of the product presentation, the students' work was evaluated by the company representatives and the professors, then questions and open discussion were made. At that meeting, in June, we arranged to travel and meet in Portugal in September but were unable to go again due to the pandemic.

4.3 Implementation

At the beginning of the project, we were divided into sub-development teams that were targeting a specific product. The end-products were a mobile application and a system that are going to communicate. The Hardware's role was to create the system of Arduino LCD Screen an NFC. My role was to deal with the programming part, creating the user interface with the device via the LCD screen and then installing the NFC antenna on the system. Dosu dealt with the organization and simulation of the system as well as the communication with the mobile app. Of course, as we were a group of two, we helped each other a lot and cooperated perfectly. Below we present and explain the code that runs in Arduino for the LCD screen.

4.3.1 Simulation

We could not have a screen with built-in buttons and a potentiometer, that's why we put them separately in the simulation we created for our program. In our real circuit, the buttons potentiometer, and backlight are integrated into our screen.

4.3.1.1 LCD Screen

For the operation of the screen, the wiring shown in Table 3 is created. For the communication of the screen with the Arduino, we need to connect the ground to terminal 2 of the potentiometer and the current pin, VCC to 5V of Arduino. The V0 pins of the two devices are connected as well as the Reset button of the screen with the pin 8 of Arduino. The buttons we will use that are in DB4, DB5, DB6, DB7 of the LCD are also connected to pin 13, pin 4,pin 5, pin 6 and pin 7 respectively in Arduino.

Table 3. Simulation LCD Interfaces

LCD SCREEN	ARDUINO / POTENTIOMETER
GND	Terminal 2 of Potentiometer
VCC	5V of Arduino
V0	Wiper pin of Potentiometer
RS	Pin 8 of Arduino
RW	GND pin of Arduino
E	Pin 13 of Arduino
DB4	Pin 4 of Arduino
DB5	Pin 5 of Arduino
DB6	Pin 6 of Arduino
DB7	Pin 7 of Arduino
LED	Through 220 Ohm Resistor to 5V of Arduino
LED	GND pin of Arduino

4.3.1.2 Buttons

The 4 buttons are used with different resistors to give us a different value, and connect them all to the Arduino A0 pin, because we do not have a screen with built-in buttons, to properly simulate our program.

As shown in the Figure 2, we use the first button as Up with a resistance of 1 kOhm, which gives us a value of 1013. We use the second button as a Right with a resistance of 2 kOhm, which gives us a value of 1003, we use the third button as a Down with a resistance of 3 kOhm, which gives us a value of 993, and the fourth button we use as Left with a resistance of 10 kOhm, which gives us a value of 930. Terminal 1 of the buttons is connected to the 5V of the Arduino, through the resistors, and terminal 2 of the buttons are short-circuited and connected via a 100 kOhm resistor to the Arduino GND.

4.3.1.3 Leds

We use 4 LEDs, one for each of the four menu functions. For "Manual Mode" we have the yellow led, for "Automatic Mode" we have the green led, for "Strobes" we have the red led and for "Dimmer" the blue led. The LED cathodes are connected to the Arduino GND and anodes to the Arduino pins through four 220 kOhm resistors. We connect the yellow led to pin 13, the green led to pin 12, the red led to pin 12 and the blue led to pin 3. There are the 4 buttons and an LCD Screen separately, as we couldn't simulate the Keypad shield we have, in the simulator. Also, the potentiometer here is an external device for the LCD Screen.

In Figure 6, the simulation of the circuit we made is created.

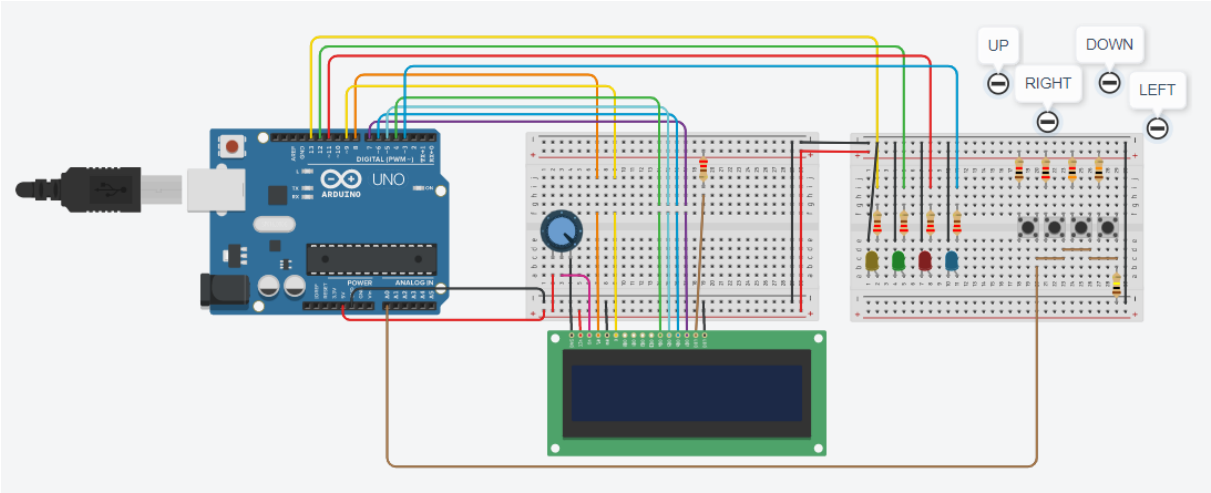

Figure 6. Simulation Circuit

In the two figures below, Figure 7 and Figure 8, the circuit diagram of the above circuit is presented. They represent the connections that are made at the simulator and as we did with our devices.

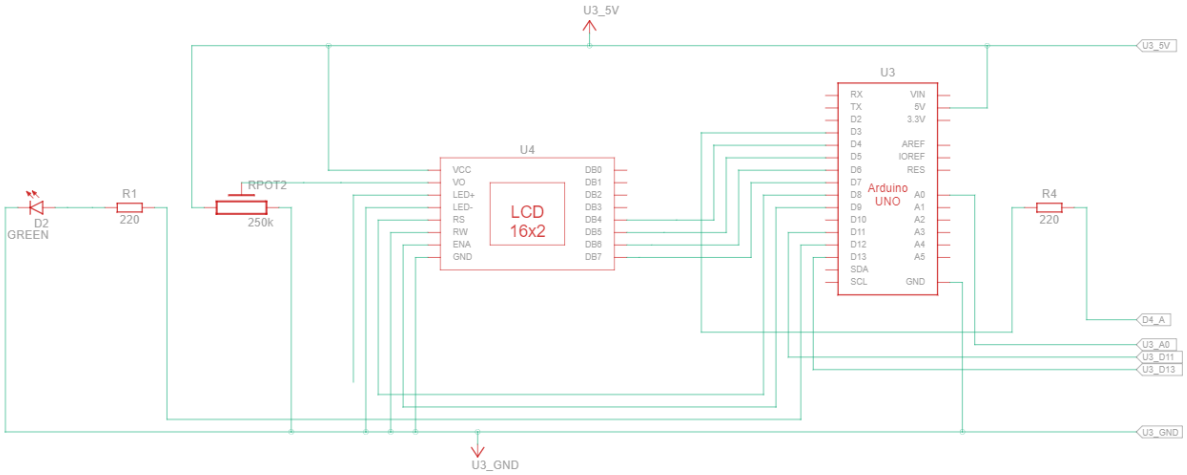

Figure 7. Simulation Circuit Diagram 1

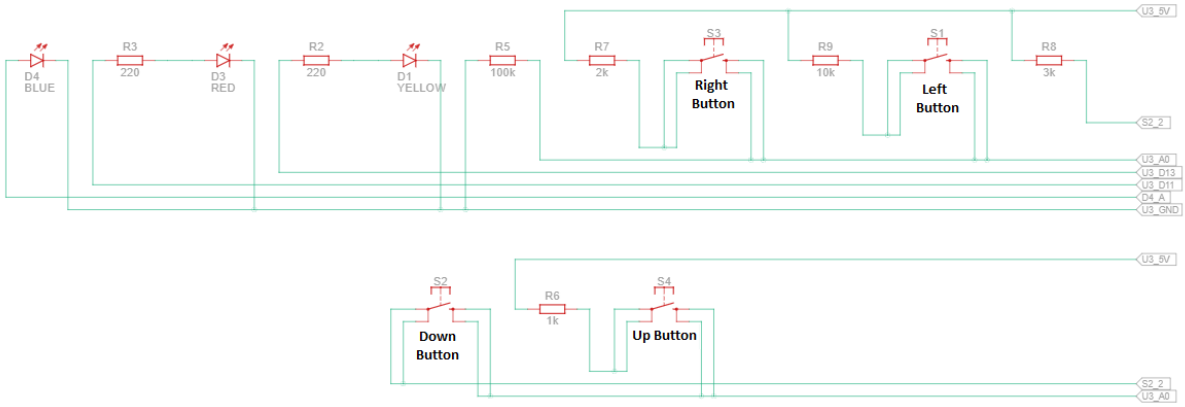

Figure 8. Simulation Circuit Diagram 2

4.3.2 Coding Explanation

4.3.2.1 Menu Options

The menu options that is created of Manual Mode, Automatic Mode, Strobes and Dimmer.

- Manual Mode - options for manual adjustment,

- Automatic Mode – we read data from the mobile app via NFC,

- Strobes for choosing whether or not to turn on the light, and

- Dimmer, which is a dimmer to adjust the light intensity.

As shown in Figure 9, the menuItems is created as an array of Strings with the 4 menu parameters that mentioned above.

```
String menuItems[] = {"Manual_Mode", "Automatic_Mode", "Strobes", "Dimmer"};
```

Figure 9. Menu Items

4.3.2.2 Declarations

Firstly, there is the definition of the variables that are going to be used later on the program. There are the navigation button variables, which we need for the buttons, and the Menu control variables which are the menuPage to keep in which page we are, the maxMenuPages to calculate how many pages we need and the cursorPosition, which we need to know in which menu item we are. In Figure 10, there are the variables that are created as integer values.

```
// Navigation button variables
int readKey;
int savedDistance = 0;

// Menu control variables
int menuPage = 0;
int maxMenuPages = round(((sizeof(menuItems) / sizeof(String)) / 2) + .5);
int cursorPosition = 0;
```

Figure 10. Declarations

4.3.2.3 Arrows

We create three arrows that we will use in the menu display for its smooth and accurate operation. The first arrow we produce is the Up arrow to move to the above option, then the Down arrow to move to the next, and last an arrow to the right to indicate what element we are in. In Figure 11, the variable downArrow is created as an array of 8 bytes, which create the down Arrow as shown in the comments.

```
byte downArrow[8] = {
  0b00100, //   *
  0b00100, //   *
  0b00100, //   *
  0b00100, //   *
  0b00100, //   *
  0b10101, // * * *
  0b01110, //  ***
  0b00100  //   *
};
```

Figure 11. Down Arrow

In Figure 12, the Up Arrow is created as an array viarable upArray of 8 bytes and in Figure 13 the array menuCursor, consists of 8 byte, represents the cursor will be used in the menu to show in which menu item we are stopped.

```
byte upArrow[8] = {
  0b00100, //   *
  0b01110, //  ***
  0b10101, // * * *
  0b00100, //   *
  0b00100, //   *
  0b00100, //   *
  0b00100, //   *
  0b00100  //   *
};
```

Figure 12. Up Arrow

```
byte menuCursor[8] = {
  B01000, //  *
  B00100, //   *
  B00010, //    *
  B00001, //     *
  B00010, //    *
  B00100, //   *
  B01000, //  *
  B00000  //
};
```

Figure 13. Menu Cursor

4.3.2.4 Libraries

In Figure 14, there is the declaration of the libraries we need. The Wire library is used for the I^2C interface function, the Liquid Crystal for the LCD screen function, and we also state the terminals that we will use for the function of the screen and the buttons. The function we use is:

LiquidCrystal (rs, enable, d4, d5, d6, d7). [21]

Rs indicates the number of the Arduino pin that is connected to the RS pin on the LCD.

Enable specify the number of the Arduino pin that is connected to the enable pin on the LCD screen.

The d4, d5, d6, d7 are the numbers of the Arduino pins that are connected to the corresponding data pins on the LCD.

Analog pin 0 of the Arduino is used to read the pushbuttons.

```
#include <Wire.h>
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

Figure 14. Libraries

4.3.2.5 Setup

We initialize the serial screen with the command Serial.begin(speed) [22] which sets the data rate in bits per second (baud) for serial data transmission.

We call lcd.begin (cols, rows) [23] which initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display. This command needs to be called before any other LCD library commands.

After that, we clear the LCD screen [24] and positions the cursor in the upper-left corner, and print text to the LCD [25]. "CONSTELL8" is printed in the first line and "KLSTR" in the second, as shown in the Figure 15. To position the LCD cursor, the lcd.setCursor() is used, that set the location at which subsequent text written to the LCD will be displayed. Delay() [26] function pauses the program for the amount of time (in milliseconds) specified as parameter.

Figure 15. Constell8

We create the three character-arrows using the function lcd.createChar (num, data) [27], where LCD is a variable of type LiquidCrystal, num is which character to create (0 to 7) and data is the character's pixel data [28]. By this function, we create a custom character for use on the LCD. Up to eight characters of 5x8 pixels are supported (numbered 0 to 7). The appearance of each custom character is specified by an array of eight bytes, one for each row. The five least significant bits of each byte determine the pixels in that row.

In Figure 16, the code that created the image above is displayed as well as the initializations of the LEDs. The specified pins (13, 12, 11, 3) are configured to behave as an output, calling the function pinMode(pin, mode).

pin: the Arduino pin to set the mode of.

mode: INPUT, OUTPUT, or INPUT_PULLUP.

```
void setup() {

  Serial.begin(9600);

  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(4, 0);
  lcd.print("CONSTELL8");
  lcd.setCursor(6, 1);
  lcd.print("KLSTR");
  delay(1500);

  lcd.createChar(0, menuCursor);
  lcd.createChar(1, upArrow);
  lcd.createChar(2, downArrow);

  pinMode(13,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(3,OUTPUT);

}
```

Figure 16. Setup

4.3.2.6 Loop

The main program consisting of the loop function contains 3 functions, as shown in Figure 17, that are created after. The first function creates the menu that appears on the screen every time that page is changed, the second function displays the appropriate cursors in the correct positions depending on the page we are on and the last function is the main function of the program.

```
void loop() {
  mainMenuDraw();
  drawCursor();
  operateMainMenu();
}
```

Figure 17. Loop

4.3.3 Functions

4.3.3.1 Function 1 – Draw Menu

This function will generate the 2 menu items that can fit on the screen. They will change as you scroll through your menu. Up and down arrows will indicate your current menu position.

We create a void function, as shown in Figure 18, since it does not return anything and display on the serial monitor the page we are on. We clear the screen, put the cursor starting in column 1 on line 0, and display item 0 (menuPage) from the menu list (menuItems). Next, we move the cursor to start column 1 on line 1 and display item 1 (menuPage+1) from the menu list (menuItems).

```
void mainMenuDraw() {
  Serial.print(menuPage);
  lcd.clear();
  lcd.setCursor(1, 0);
  lcd.print(menuItems[menuPage]);
  lcd.setCursor(1, 1);
  lcd.print(menuItems[menuPage + 1]);
```

Figure 18. Main Menu Draw

Subsequently, we create an if statement to show the right arrow either up or down, depending on which page we are on.

```
  if (menuPage == 0) {
    lcd.setCursor(15, 1);
    lcd.write(byte(2));
  } else if (menuPage > 0 and menuPage < maxMenuPages) {
    lcd.setCursor(15, 1);
    lcd.write(byte(2));
    lcd.setCursor(15, 0);
    lcd.write(byte(1));
  } else if (menuPage == maxMenuPages) {
    lcd.setCursor(15, 0);
    lcd.write(byte(1));
    lcd.setCursor(0, 1);
    lcd.print("               ");
  }
}
```

Figure 19. Menu Pages

Based on the code presented in Figure 19, if we are on page 0, we place the cursor in column 15 on line 1 and display the down arrow we created as byte[2]. We use the lcd.write() function to write a character to the LCD [29]. In this page, there is only the down arrow, as shown in Figure 20 and Figure 21.

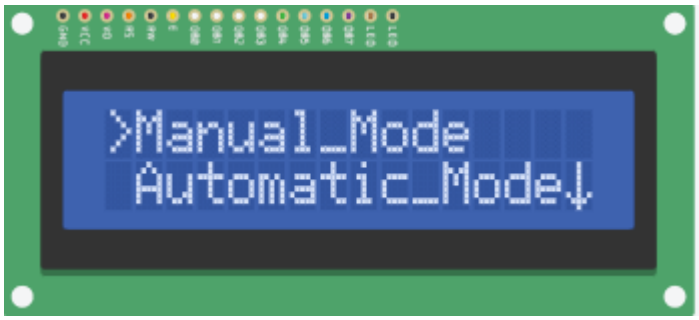

Figure 20. Page 0 – Only Down Arrow

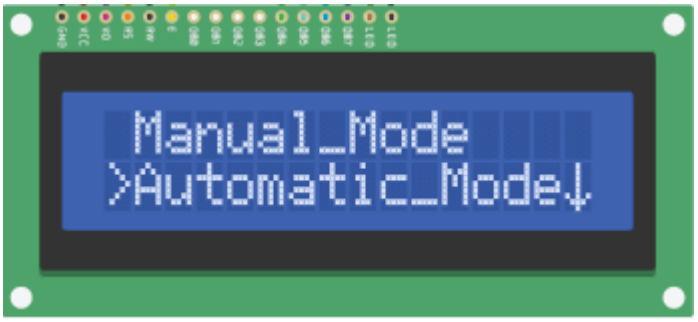

Figure 21. Page 0- Only Down Arrow

Otherwise, if we are neither on page 0 nor on the last one, place the cursor in column 15 on line 1, where we display the down arrow and place the cursor in column 15 on line 0, where we display the up arrow we made as a byte[1]. In these pages there are both up and down arrows, as shown in Figure 22 and Figure 23.

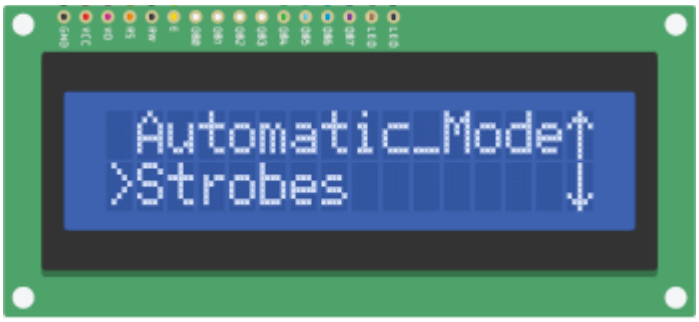

Figure 22. Page 1 – Both Arrows

Figure 23. Page 2 – Both Arrows

Finally, if we are on the last page (maxMenuPages), we put the cursor in column 15 on line 0 and display the up arrow. Then we place the cursor in column 15 on line 0 and display an empty string of 16 characters, so it will be an empty line without arrows. In these page there is only the up arrow, as shown in Figure 24.

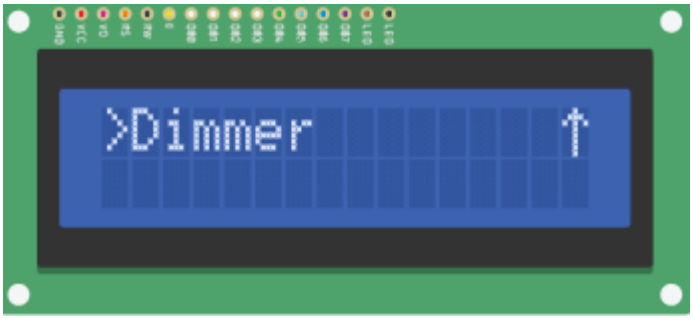

Figure 24. Page 3 – Only Up Arrow

4.3.3.2 Function 2 – Draw Cursor

When called, this function will erase the current cursor and redraw it based on the cursorPosition and menuPage variables.

We create a void function since it does not return anything and firstly create a for loop that runs two times, one for each line, where we put the cursor in column 0 on line x, and where the cursor was, puts the space.

The menu is set up to be progressive (menuPage 0 = Item 1 & Item 2, menuPage 1 = Item 2 & Item 3, menuPage 2 = Item 3 & Item 4), so in order to determine where the cursor should be you need to see if you are at an odd or even menu page and an odd or even cursor position.

If we are on a menu page that is exactly divided by 2 (even), and the cursor position is exactly divided by 2 (even) that means the cursor should be on the first line (line 0). We put the cursor in column 0 on line 0 and display the menu cursor which is declared as byte[0]. If the menu page is even and the cursor position is odd that means the cursor should be on the second line (line 1). We put the cursor in column 0 on line 1 and display the menu cursor which is declared as byte[0]. In Figure 25, the code of these functions is represented.

```
void drawCursor() {
  for (int x = 0; x < 2; x++) {
    lcd.setCursor(0, x);
    lcd.print(" ");
  }
  if (menuPage % 2 == 0) {
    if (cursorPosition % 2 == 0) {
      lcd.setCursor(0, 0);
      lcd.write(byte(0));
    }
    if (cursorPosition % 2 != 0) {
      lcd.setCursor(0, 1);
      lcd.write(byte(0));
    }
  }
```

Figure 25. Draw Cursor – Even Page

As shown in Figure 26, when we are on page 0 and the cursor is in position 0, both menu page and cursor position are even numbers, so the cursor shows the first item of the screen.

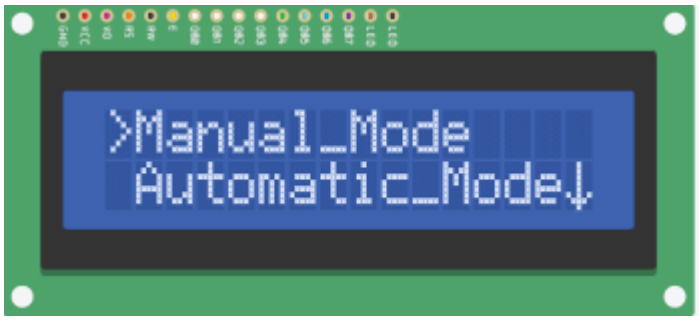

Figure 26. Even page – Even cursor

As shown in Figure 27, when we are on page 0 and the cursor is in position 1, menu page is even number and cursor position is odd number, so the cursor shows the second item of the screen.

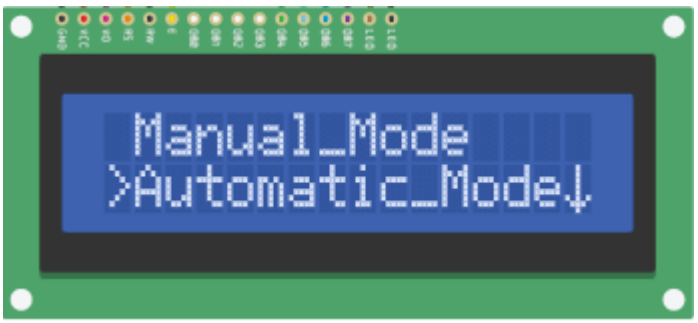

Figure 27. Even page – Odd cursor

As shown in Figure 28, when we are on page 2 and the cursor is in position 3, menu page is even number and cursor position is odd number, so the cursor shows the second item of the screen.

Figure 28. Even page – Odd cursor

If we are on a page that is not exactly divisible by 2 (odd), and the cursor position is exactly divided by 2 (even) that means the cursor should be on the second line (line 1), as shown in the code is represented in Figure 29. We put the cursor in column 0 on line 1 and display the menu cursor which is declared as byte[0]. If the menu page is odd and the cursor position is odd that means the cursor should be on the first (line 0). We put the cursor in column 0 on line 0 and display the menu cursor which is declared as byte[0].

```
if (menuPage % 2 != 0) {
  if (cursorPosition % 2 == 0) {
    lcd.setCursor(0, 1);
    lcd.write(byte(0));
  }
  if (cursorPosition % 2 != 0) {
    lcd.setCursor(0, 0);
    lcd.write(byte(0));
  }
}
}
```

Figure 29. Draw Cursor – Odd Page

When we are on page 1 and the cursor is in position 2, menu page is odd number and cursor position is even number, so the cursor shows the second item of the screen, as shown in Figure 30.

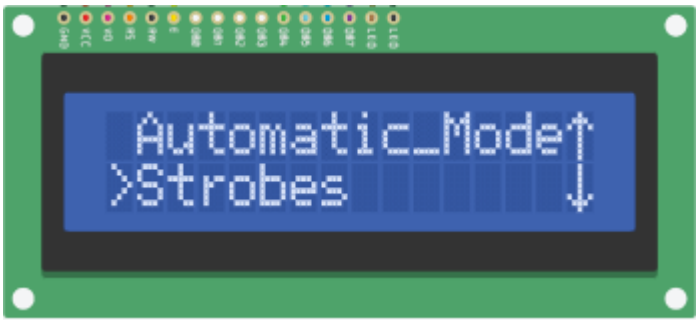

Figure 30. Odd page – Even cursor

As shown in Figure 31, when we are on page 3 and the cursor is in position 3, both menu page and cursor position are odd numbers, so the cursor shows the second item of the screen:

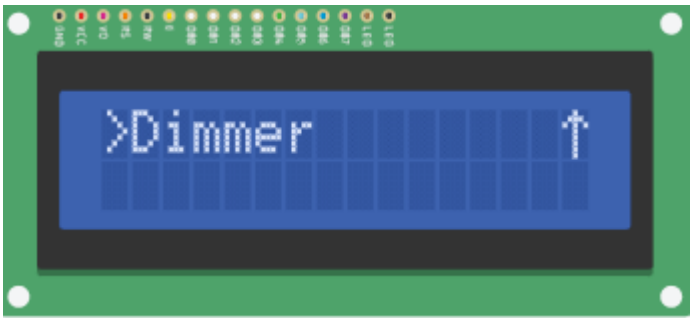

Figure 31. Odd page – Odd cursor

4.3.3.3 Function 3 – Main Menu

In Figure 32, there is the function for the main operation of the program. The function operate MainMenu is created as a void function as we don't need to use any variable from this.

Firstly, we create a variable to keep when a button was pressed. When activeButton is 0 no button has been pressed while if it becomes 1 it means that it has been pressed.

We create a while loop where as long as we have not pressed a button, we read in the variable readKey lest a value come with the analogRead function in pin A0, if a button was pressed. If the value of readKey is less than 790 it makes a very short delay and re-reads pin A0.. Immediately after, the button variable calls the evaluateButton function with the readKey variable value, where it returns a number depending on which button was pressed.

```
void operateMainMenu() {

  int activeButton = 0;

  while (activeButton == 0) {
    int button;
    readKey = analogRead(0);
    if (readKey < 790) {
      delay(100);
      readKey = analogRead(0);
    }
    button = evaluateButton(readKey);
```

Figure 32. Operate Main Menu

The button variable is 0 if no button is pressed otherwise it takes the values 1, 2, 3 and 4.

We create a switch with the variable button argument, where depending on its value the appropriate menu with the corresponding arrows is created on the screen.

Case 0

When button returns as 0 there is no action taken. We call break to exit the switch.

Case 1 – Right Button

This case will execute if the "forward" button is pressed (Right button).

We equalize the button variable with 0 so that it does not re-enter the switch and create a new switch depending on the position of the cursor where the case that is selected here is dependent on which menu page you are on and where the cursor is.

- If the cursor is in position 0, function 5 (Manual Mode) is called and we exit this switch.
- If the cursor is in position 1, function 6 (Automatic Mode) is called and we exit this switch.
- If the cursor is in position 2, function 7 (Strobes) is called and we exit this switch.
- If the cursor is in position 3, function 8 (Dimmer) is called and we exit this switch.

We make the variable active Button equal to 1, to keep that a button is pressed. Finally, we make the variable activeButton equal to 1, to keep that a button is pressed, call function 1 to create the menu that corresponds to this page, we call function 2 for proper display of the cursor that shows our data and we exit the switch.

In Figure 33, there is the Switch where the variable button is evaluated.

```
switch (button) {
  case 0:
    break;
  case 1:
    button = 0;
    switch (cursorPosition) {
      case 0:
        menuItem1();
        break;
      case 1:
        menuItem2();
        break;
      case 2:
        menuItem3();
        break;
      case 3:
        menuItem4();
        break;

    }
    activeButton = 1;
    mainMenuDraw();
    drawCursor();
    break;
```

Figure 33. Switch Menu Items – Case 0 & Case 1 – Right Button

Case 2 – Up Button

This case ,which is shown in Figure 34, will execute if the "previous" button is pressed (Up).

First, we make the button equal to 0 so that it does not hold a button value.

If we are on page 0, we make the position of the cursor as it was minus 1, that is, it goes to the previous line, and we call the constrain function for its value. This constrains a number to be within a range. The cursorPosition will remain as long as it is between 0 and the maximum number it can get, or it will become 0 if the cursorPosition is less than 0 or it will become the maximum if the cursorPosition is greater than this.

If we are on an even number page and the cursor position is an even number, we make the page we are on as it was minus 1, that is, we go to the previous one. We call the constrain function for the menuPage variable where if the page we are on is set to 0 and the maximum number of pages remains the same if it is less than 0 it will become 0 and if it is greater than the maximum it will become the maximum.

We place the cursor as it was minus 1 and call the constrain function for it, where the cursorPosition will remain as long as it is between 0 and the maximum number it can get, or it will become 0 if the cursorPosition is less than 0 or will be maximized if the cursorPosition is greater than this.

Finally, we call function 1 to create the menu that corresponds to this page, we call function 2 for proper display of the cursor that shows our data, we make the variable activeButton equal to 1, to keep that a button is pressed and exit the switch.

```
case 2:
  button = 0;
  if (menuPage == 0) {
    cursorPosition = cursorPosition - 1;
    cursorPosition = constrain(cursorPosition, 0, ((sizeof(menuItems) / sizeof(String)) - 1));
  }
  if (menuPage % 2 == 0 and cursorPosition % 2 == 0) {
    menuPage = menuPage - 1;
    menuPage = constrain(menuPage, 0, maxMenuPages);
  }
  if (menuPage % 2 != 0 and cursorPosition % 2 != 0) {
    menuPage = menuPage - 1;
    menuPage = constrain(menuPage, 0, maxMenuPages);
  }

  cursorPosition = cursorPosition - 1;
  cursorPosition = constrain(cursorPosition, 0, ((sizeof(menuItems) / sizeof(String)) - 1));

  mainMenuDraw();
  drawCursor();
  activeButton = 1;
  break;
```

Figure 34. Case 2 – Up Button

Case 3 – Down Button

The case in Figure 35 will execute if the "next" button is pressed (Down).

First, we make the button equal to 0 so that it does not hold a button value.

If we are on an even number page and the cursor position is an odd number, we make the page we are on as it was plus 1, that is, we go to the next one. We call the constrain function for the menuPage variable where if the page we are on is set to 0 and the maximum number of pages remains the same if it is less than 0 it will become 0 and if it is greater than the maximum it will become the maximum.

If we are on an odd number page and the cursor position is an even number, we make the page we are on as it was plus 1, that is, we go to the next one. We call the constrain function for the menuPage variable where if the page we are on is set to 0 and the maximum number of pages remains the same if it is less than 0 it will become 0 and if it is greater than the maximum it will become the maximum.

We place the cursor as it was plus 1 and call the constrain function for it, where the cursorPosition will remain as long as it is between 0 and the maximum number it can get, or it will become 0 if the cursorPosition is less than 0 or will be maximized if the cursorPosition is greater than this.

Finally, we call function 1 to create the menu that corresponds to this page, we call function 2 for proper display of the cursor that shows our data, we make the variable activeButton equal to 1, to keep that a button is pressed and exit the switch.

```
    case 3:
        button = 0;
        if (menuPage % 2 == 0 and cursorPosition % 2 != 0) {
          menuPage = menuPage + 1;
          menuPage = constrain(menuPage, 0, maxMenuPages);
        }

        if (menuPage % 2 != 0 and cursorPosition % 2 == 0) {
          menuPage = menuPage + 1;
          menuPage = constrain(menuPage, 0, maxMenuPages);
        }

        cursorPosition = cursorPosition + 1;
        cursorPosition = constrain(cursorPosition, 0, ((sizeof(menuItems) / sizeof(String)) - 1));

        mainMenuDraw();
        drawCursor();
        activeButton = 1;
        break;
    }
  }
}
```

Figure 35. Case 3 – Down Button

4.3.3.4 Function 4 - Buttons

The function in Figure 36 is called whenever a button press is evaluated. The LCD shield works by observing a voltage drop across the buttons all hooked up to A0.

We create the int function evaluateButton as it returns an integer value, and takes as an argument integer x. In it, we define the result variable with a value of 0 initially, and depending on the value of x the result takes and returns a value.

If x is less than 50 the result becomes 1 and means that the right button is pressed, if it is greater than or equal to 50 and less than 195 the result becomes 2 and means that the up button is pressed, if it is greater than or equal to 195 and less than 380 the result becomes 3 and means that the down button is pressed, and finally if it is greater than or equal to 380 and less than 1790 the result becomes 4 and means that the left button is pressed. Finally, the result returns with the value it finally got depending on the value of x.

```
int evaluateButton(int x) {
  int result = 0;
  if (x < 50) {
    result = 1; // right
  } else if (x < 195) {
    result = 2; // up
  } else if (x < 380) {
    result = 3; // down
  } else if (x < 790) {
    result = 4; // left
  }

  return result;
}
```

Figure 36. Evaluate Button

4.3.3.5 Function 5 – Manual Mode

This function executes when you select the "Manual Mode" item from the main menu. "Manual Mode". This is the first item of the menu list that's why we call it menuItem1(), which is a void function as it doesn't return anything.

We set the activeButton to 0 so that it no longer holds that a button is pressed, clear the screen, we place the cursor in column 1 on line 0 and display "Enter Fixture". After a short delay of 500 milliseconds, we move the cursor to column 0 on line 1 and display "50" as the Fixture ID, we turn on the yellow LED, using the digitalWrite function where we send a HIGH signal to pin 13 of the Arduino. The pin has been configured as an OUTPUT with pinMode (), so its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW. The two arguments we use are the Arduino pin number and HIGH to light the led.

In Figure 37, the first menu item is represented.

```
void menuItem1() {
  int activeButton = 0;

  lcd.clear();
  lcd.setCursor(1, 0);
  lcd.print("Enter Fixture");
  delay(500);
  lcd.setCursor(1, 1);
  lcd.print("Fixture ID:");
  lcd.print("50");

  digitalWrite(13,HIGH);
```

Figure 37. Manual Mode

We create a while loop, in Figure 38, where as long as we have not pressed a button, we read the Arduino pin A0 in the readKey variable. If the value of readKey is less than 790 it makes a very short delay of 100 ms and re-reads pin A0. Immediately after, the button variable calls the evaluateButton function with the readKey variable value, where it returns a number depending on which button was pressed.
We call the switch with the button argument, where it has only one case. This case will execute if the "back" button is pressed (Left button), so the buttons' value is equal to 4. If the buttons' value is 4, we make the button equal to 0 so that it does not hold the value of the button, we make the activeButton equal to 1 so that it holds that a button was pressed, we erase the yellow led with the digitalWrite function where now we call it with pin 13 of Arduino and LOW to turn off the led and exit the switch.

```
while (activeButton == 0) {
  int button;
  readKey = analogRead(0);
  if (readKey < 790) {
    delay(100);
    readKey = analogRead(0);
  }
  button = evaluateButton(readKey);
  switch (button) {
    case 4:
      button = 0;
      activeButton = 1;
      digitalWrite(13,LOW);
      break;
  }
 }
}
```

Figure 38. Yellow Led

4.3.3.6 Function 6 – Automatic Mode

This function executes when you select the "Automatic Mode" item from main menu. This is the second item of the menu list that's why we call it menuItem2(), which is a void function as it doesn't return anything.

We set the activeButton to 0 so that it no longer holds that a button is pressed, clear the screen, we place the cursor in column 1 on line 0, display "Automatic Mode" and after that we place the cursor in column 1 on line 1 and display "Initializing". We turn on the green LED, using the digitalWrite function where we send a HIGH signal to pin 12 of the Arduino. The pin has been configured as an OUTPUT with pinMode(), so its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW. The two arguments we use are the Arduino pin number and HIGH to light the led.

In Figure 39, the second menu item is created.

```
void menuItem2() {
  int activeButton = 0;

  lcd.clear();
  lcd.setCursor(1, 0);
  lcd.print("Automatic Mode");
  lcd.setCursor(1, 1);
  lcd.print("Initializing...");

  digitalWrite(12,HIGH);
```

Figure 39. Automatic Mode

We create a while loop in Figure 40, where as long as we have not pressed a button, we read the Arduino pin A0 in the readKey variable. If the value of readKey is less than 790 it makes a very short delay of 100 ms and re-reads pin A0. Immediately after, the button variable calls the evaluateButton function with the readKey variable value, where it returns a number depending on which button was pressed.
We call the switch with the button argument, where it has only one case. This case will execute if the "back" button is pressed (Left button), so the buttons' value is equal to 4. If the buttons' value is 4, we make the button equal to 0 so that it does not hold the value of the button, we make the activeButton equal to 1 so that it holds that a button was pressed, we erase the green led with the digitalWrite function where now we call it with pin 12 of Arduino and LOW to turn off the led and exit the switch.

```
  while (activeButton == 0) {
    int button;
    readKey = analogRead(0);
    if (readKey < 790) {
      delay(100);
      readKey = analogRead(0);
    }
    button = evaluateButton(readKey);
    switch (button) {
      case 4:
        button = 0;
        activeButton = 1;
        digitalWrite(12,LOW);
        break;
    }
  }
}
```

Figure 40. Green Led

4.3.3.7 Function 7 - Strobes
This function executes when we select the "Strobes" item from main menu. This is the third item of the menu list that's why we call it menuItem3(), which is a void function as it doesn't return anything.

We set the activeButton to 0 so that it no longer holds that a button is pressed, clear the screen, we place the cursor in column 1 on line 0 and display "Strobe Mode" and after that we place the cursor in column 0 on line 1 and display "UpRightDownLeft" as the choices to move the light. We turn on the red LED, using the digitalWrite function where we send a HIGH signal to pin 11 of the Arduino. The pin has been configured as an OUTPUT with pinMode(), so its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW. The two arguments we use are the Arduino pin number and HIGH to light the led.

In Figure 41, the third menu item is created.

```
void menuItem3() {
  int activeButton = 0;

  lcd.clear();
  lcd.setCursor(1, 0);
  lcd.print("Strobe Mode");
  lcd.setCursor(0, 1);
  lcd.print("UpRightDownLeft");

  digitalWrite(11,HIGH);
```

Figure 41. Strobe Mode

We create a while loop in Figure 42, where as long as we have not pressed a button, we read the Arduino pin A0 in the readKey variable. If the value of readKey is less than 790 it makes a very short delay of 100 ms and re-reads pin A0. Immediately after, the button variable calls the evaluateButton function with the readKey variable value, where it returns a number depending on which button was pressed.
We call the switch with the button argument, where it has only one case. This case will execute if the "back" button is pressed (Left button), so the buttons' value is equal to 4. If the buttons' value is 4, we make the button equal to 0 so that it does not hold the value of the button, we make the activeButton equal to 1 so that it holds that a button was pressed, we erase the red led with the digitalWrite function where now we call it with pin 11 of Arduino and LOW to turn off the led and exit the switch.

```
  while (activeButton == 0) {
    int button;
    readKey = analogRead(0);
    if (readKey < 790) {
      delay(100);
      readKey = analogRead(0);
    }
    button = evaluateButton(readKey);
    switch (button) {
      case 4:
        button = 0;
        activeButton = 1;
        digitalWrite(11,LOW);
        break;
    }
  }
}
```

Figure 42. Red Led

4.3.3.8 Function 8 - Dimmer

This unction executes when you select the "Dimmer" item from main menu. This is the fourth item of the menu list that's why we call it menuItem4(), which is a void function as it doesn't return anything.

We set the activeButton to 0 so that it no longer holds that a button is pressed, clear the screen, we place the cursor in column 1 on line 0 and display "Dimmer Mode" and after that we place the cursor in column 0 on line 1 and display "Incr + / Decr -" as the choices to change the brightness. We turn on the blue LED, using the digitalWrite function where we send a HIGH signal to pin 3 of the Arduino. The pin has been configured as an OUTPUT with pinMode(), so its voltage will be set to the corresponding value: 5V for HIGH, 0V (ground) for LOW. The two arguments we use are the Arduino pin number and HIGH to light the led.

In Figure 43, the menu item "Dimmer" is created.

```
void menuItem4() {
  int activeButton = 0;

  lcd.clear();
  lcd.setCursor(2, 0);
  lcd.print("Dimmer Mode");
  lcd.setCursor(0, 1);
  lcd.print("Incr + / Decr -");

  digitalWrite(3,HIGH);
```

Figure 43. Dimmer Mode

We create a while loop in Figure 44, where as long as we have not pressed a button, we read the Arduino pin A0 in the readKey variable. If the value of readKey is less than 790 it makes a very short delay of 100 ms and re-reads pin A0. Immediately after, the button variable calls the evaluateButton function with the readKey variable value, where it returns a number depending on which button was pressed.
We call the switch with the button argument, where it has only one case. This case will execute if the "back" button is pressed (Left button), so the buttons' value is equal to 4. If the buttons' value is 4, we make the button equal to 0 so that it does not hold the value of the button, we make the activeButton equal to 1 so that it holds that a button was pressed, we erase the blue led with the digitalWrite function where now we call it with pin 3 of Arduino and LOW to turn off the led and exit the switch.

```
while (activeButton == 0) {
  int button;
  readKey = analogRead(0);
  if (readKey < 790) {
    delay(100);
    readKey = analogRead(0);
  }
  button = evaluateButton(readKey);
  switch (button) {
    case 4:
      button = 0;
      activeButton = 1;
      digitalWrite(3,LOW);
      break;
  }
 }
}
```

Figure 44. Blue Led

4.3.4 NTAG I²C Antenna

To use the NTAG I²C Antenna we need to use I²C wiring. The pins we need are the VCC, GND, SDA and SCL, which are connected as shown in Table 4.

Table 4. NTAG I²C Antenna

NTAG I²C ANTENNA	ARDUINO
SDA	A4
SCL	A5
VCC	5V
GND	GND

In the Arduino IDE we install the Arduino Ntag Master library, which contains the programs we need to see if the antenna works, to write and read from it.

From the menu, select File, Examples, Arduino-ntag-master where we find the functions we will use.

First, we select ntagTest, in which we see the introduction of the libraries it needs to run. A very important library is <Wire.h>, which allows you to communicate with I2C devices, often also called "2 wire" or "TWI" (Two Wire Interface). In the setup after we initialize the serial screen and display a message, if ntag is not found we display a suitable message. We continue with the dialing of 5 functions for its basic function, with which we display the serial number

of the tag in hexadecimal, we write and read in the EEPROM memory and it reads data from the registers. So, we see that our antenna is working and we can use it.

In Figure 45, we represent the output of the program we used.

```
start

Serial number of the tag is: 4 BD 91 4A 94 51 80
Writing block 1
Writing block 2

Reading memory block 1
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
Reading memory block 2
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
Reading bytes 10 to 20: partly block 1, partly block 2
Writing byte 15 to 20: partly block 1, partly block 2
Write success

Reading memory block 1
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 70
Reading memory block 2
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
1
0
1
1
0
```

Figure 45. NTAG Test

In the WriteTag function, we use NdefMessage to declare the message we will create and by calling the message.addUriRecord() we write the message we want. We also display a message about the successful or unsuccessful completion of the action. In the ReadTag function using the NfcTag function, we declare the tag with which by calling tag.print() we display the message that contains our tag in hexadecimal.

There are also the CleanTag, which resets a tag back to factory-like state, and EraseTag, which erases a NFC tag by writing an empty NDEF message.

For our program, we need to read and write through the antenna, as we want to communicate with any mobile phone that contains our mobile app and get data from it. We also want the mobile app to read data through the antenna as manual changes may have been made to our system.

4.3.5 Circuit Diagram

The wiring we use with our LCD Keypad Shield, Arduino and Leds is shown in Table 5.

The Digital Arduino pins we use are 3, 11, 12, 13 to light up the Leds and 4, 5, 6, 7, 8, 9 to communicate with the LCD Screen.

The Analog Arduino pins we use are A0, A4, A5 to communicate with the NTAG I^2C Antenna.

They are all connected to VCC and GND of Arduino.

Table 5. Circuit Wiring

ARDUINO PINS	LCD KEYPAD SHIELD / LEDS / I^2C
3	Blue Led
4	DB4
5	DB5
6	DB6
7	DB7
8	RS
9	E
11	Red Led
12	Green Led
13	Yellow Led
A0	A0
A4	SDA
A5	SCL
VCC	5V of LCD
	Cathodes of Leds
	VCC of NTAG
GND	GND of LDC
	Anodes of LEDS through Resistors
	GND of NTAG

In the Figure 46 below, the layout of the circuit we created is presented. Because there was no LCD Keypad Shield to use, schematically added the 4 buttons and pin A0 to the screen. NTAG I^{2}C Antenna is also drawn to show the configuration.

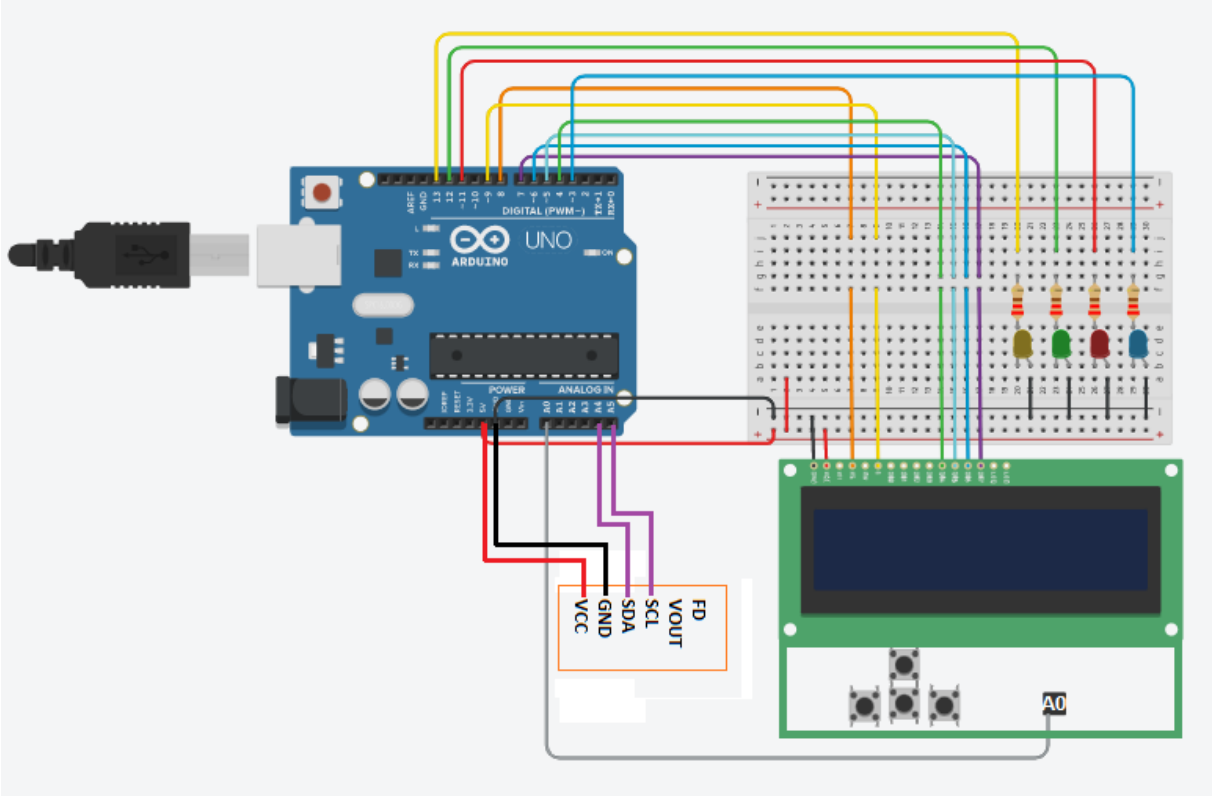

Figure 46. Circuit Diagram

CHAPTER 5: RESULTS

5.1 Conclusion

This thesis was held through the project of Blended Aim. Through this project, there is a collaboration of universities from all over Europe and beyond, for the execution of two projects for startups companies. Students from different universities and specialties are divided into two groups, one for each project, and collaborate most of the time remotely to complete them.

The project of the group I participated in, was created by a Belgian start-up company, named Constell8 which specializes in managing lighting systems for large events such as concerts. The result of this project is a product named KLSTR which consists of a mobile app and a system that is integrated into every lighting.

KLSTR was inspired and created by the need for development in the field of lighting, as the technologies used to date are based on a 33-year protocol. By using the system we created, the management of such systems becomes faster and more efficient. The lighting technician is in charge of managing each of the lights and no longer needs to adjust each one closely. All it has to do is make the settings he needs through the mobile app and just run his phone over the NFC tag that is in the light to write the changes on it. By this project, the light technician has the opportunity to make settings either manually on the device or via his mobile phone. We also ensure the security of the system as to pass the changes from the application to the lighting we have to bring the mobile phone to the system at a short distance, this is the reason why we chose NFC technology for its implementation.

It was an amazing and unique experience the opportunity given to all of us, to work with students from different universities with different qualities, different nationalities, and located in different parts of the world. We needed to be organizational, patient, and receptive to everyone's views and ideas to work effectively, as we did. There were several difficulties due to the distance, the time difference, the difference in the way everyone works and way of thinking but we respect each other. We were divided into smaller groups and undertook separate tasks depending on the field of studies of each. Dosu and I are the Hardware team, where I dealt with the system in the light. Despite the difficulties, we had very good cooperation and we always supported each other.

Blended Aim Mobility allowed us to work professionally for the first time as students and gain new knowledge, experiences, and opportunities. We had the opportunity to meet and discuss different technologies, to learn about them, and to work professionally. Technologies like Arduino and NFC are highly developed nowadays as they can be used in many different ways in different projects. We needed to deal with such technologies as we will be prepared either for school lessons or future work on topics such as the Internet of Things.

5.2 Future work and extensions

We have created the on-screen menu, but it is not functional yet. We can scroll to it but it does not do any other function.

We need to integrate the antenna function in our basic code so that through Auto Mode we can read the data from the mobile app via NFC tag, and pass it from the NFC antenna to the light, through the Arduino.

We need to make the functions for Manual Mode so that through the buttons we can change the ID of Fixture, through Strobes the movement and through the Dimmer the brightness. To do this, we will need to send the values we want to the appropriate memory point they already have and control with their existing system.

Finally, we have to test with the mobile app that the data is exchanged correctly and finally do the packaging of the system.

A future optimization is to replace NFC with Bluetooth technology. To do this, however, we must ensure the security of this system so that someone unauthorized cannot tamper with the system, as Bluetooth has a wider range than NFC.

REFERENCES

[1] Nuno Escuderio, Paula Escuderio, Ricardo Almeida, Ana Barata, Tatjana Welzer, Giorgos Papadourakis, Blended Academic International Mobility: tearing down barrirers to mobility in a suistanable way

[2] The Definitive Guiede to Scaling Scrum with Nexus, January 2021

<https://www.scrum.org/resources/online-nexus-guide>

[3] Atlassian, What is Jira for

<https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for#Jira-for-requirements-&-test-case-management>

[4] Wikipedia, Slack (software)

<https://en.wikipedia.org/wiki/Slack_(software)>

[5] Wikipedia, Arduino Uno

<https://en.wikipedia.org/wiki/Arduino_Uno>

[6] Mobasshir Mahbub, ReserchGate, Automated Control Signal Reception Acknowledgement System Using Nrf24101p Wireless Transeiver Module and Arduino

<https://www.researchgate.net/publication/332277135_Automated_Control_Signal_Reception_Acknowledgement_System_Using_Nrf24l01p_Wireless_Transceiver_Module_and_Arduino>

[7] Arduino Uno R3 Datasheet

<https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>

[8] Atmel, ATMega 328P, 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash Datasheet

<http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>

[9] Wikipedia, UART

<https://el.wikipedia.org/wiki/UART>

[10] Wikipedia, Pulse-Width Modulation

<https://en.wikipedia.org/wiki/Pulse-width_modulation>

[11] Wikipedia, Serial Peripheral Interface

<https://en.wikipedia.org/wiki/Serial_Peripheral_Interface>

[12] Arduino, Wire Library

<https://www.arduino.cc/en/reference/wire>

[13] Wikipedia, Arduino IDE

<https://en.wikipedia.org/wiki/Arduino_IDE>

[14] ElectroPeak, Arduino Project Hub, Using 1602 LCD Keypad Shield Arduino

<https://create.arduino.cc/projecthub/electropeak/using-1602-lcd-keypad-shield-w-arduino-w-examples-e02d95>

[15] DF Robot, DFR0009 LCD KeyPad Shield For Arduino

<https://wiki.dfrobot.com/LCD_KeyPad_Shield_For_Arduino_SKU__DFR0009>

[16] Grobotronics, LED Diffused 5mm

<https://grobotronics.com/led-diffused-5mm-elrd.html>

[17] Wikipedia, Near-Field Communication

< https://en.wikipedia.org/wiki/Near-field_communication>

[18] Wikipedia, I^2C (Inter-Integrated Circuit)

<https://en.wikipedia.org/wiki/I%C2%B2C>

[19] Scott Campbell, DIY Electronics, Circuit Basics, Basics of the I^2C Communication Protocol

<https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>

[20] Product datasheet Company Public, NTAG I^2C plus: NFC Forum T2T with I^2C interface, password protection and energy harvesting

<https://www.nxp.com/docs/en/data-sheet/NT3H2111_2211.pdf>

[21] Arduino, Liquid Crystal Constructor

<https://www.arduino.cc/en/Reference/LiquidCrystalConstructor>

[22] Arduino, Serial.begin()

<https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/>

[23] Arduino, Liquid Crystal Begin

<https://www.arduino.cc/en/Reference/LiquidCrystalBegin>

[24] Arduino, Liquid Crystal Clear

<https://www.arduino.cc/en/Reference/LiquidCrystalClear>

[25] Arduino, Liquid Crystal Print

<https://www.arduino.cc/reference/en/libraries/liquidcrystal/print/>

[26] Arduino, delay()

<https://www.arduino.cc/reference/en/language/functions/time/delay/>

[27] Arduino, Liquid Crystal Create Char
<https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar>

[28] Arduino, Liquid Crystal Displays (LCD) with Arduino
<https://docs.arduino.cc/learn/electronics/lcd-displays>

[29] Arduino, Liquid Crystal Write
<https://www.arduino.cc/reference/en/libraries/liquidcrystal/write/>