Designing a Hardware Controller

Used for controlling a Power Grid simulation

Michiel De Rop
Johannes Ketelaars

Delft University of Technology

TU Delft

Designing a Hardware Controller

Used for controlling a Power Grid simulation

by

Michiel De Rop
Johannes Ketelaars

to obtain the degree of Bachelor of Science

Supervised by dr. S.H. Tindemans, Ing. R.N. Koornneef and N.K. (Nanda) Panda

Defended before dr.ir. N.P. van der Meijs, dr. S.H. Tindemans, dr.ing. M. Shahraki Moghaddam and ing. R.N. Koornneef

Bachelor Graduation Thesis
Friday 28th June, 2024

Delft University of Technology

Faculty of Electrical Engineering, Mathematics
and Computer Science

Electrical Engineering Programme

Abstract

As part of making a power grid simulator, a hardware controller is needed. This thesis specifically describes the design of such a controller. The objective is to design it in such a way that it is self-explanatory and interactive. Therefore, the following user controls and feedback are made possible. A small solar panel makes it possible to provide input for a solar generator in the simulation. A slider gives input for a wind farm in the simulation. This is visualized by adding a small physical turbine to the hardware, going faster or slower based on the input. A small knob to go back or forward in the simulation, if pressed, pauses or resumes the time. A RGB LED indicates the situation on the grid. Based on the loading of transmission lines, the color of the LED changes. These user-control actions and feedback from the simulation are processed on an Arduino Mega Rev3 development board. This uses an ATMega2560 microcontroller. To interact with the simulation, bidirectional serial communication is needed between the Arduino and the computer running the simulation with the Godot game engine. These components are implemented on a PCB to finalize the design.

Preface

This report is written as part of the bachelor's graduation project for the bachelor of electrical engineering at TU Delft. The project was initiated to create an interactive power grid visualization and simulation system. Our part in the project was to create a hardware controller. This project has confronted us with the challenges that working as a team brings. It was a great opportunity to gain experience handling projects as a team. The project proved to be difficult to integrate with other teams, this has shown the importance of communication. We experienced a feeling of satisfaction seeing all parts of the project come together and form a whole. New experience was gained due to the need for programming in languages the team members had little to no skills.

First, we want to express our gratitude to our supervisor, Simon Tindemans, for proposing this project. His advice and enthusiasm were of great value in accomplishing the final result. Secondly, we want to thank Remko Koornneef for sharing his expertise when designing the PCB for the project. Lastly, we want to express our gratitude to our BAP group. Thanks to Rik Bieling, Joran Kroese, Björn Buksch, and Bodhi van Dam for their collaboration and support during this project.

As we present this thesis, we are proud of the accomplishments and eager to see the applications of the final result.

Michiel De Rop
Johannes Ketelaars
Delft, June 2024

Contents

1 Introduction

This project is about the hardware components of a power grid visualizer. This visualizer simulates different scenarios of the power grid. This enables users to increase their understanding of the inner workings of the transmission grid network. It depends on how the data is visualized and how the user can play with that data. So the main purpose of the project is to make a complex scientific topic more tangible. The most important thing in science communication is that it informs the audience about the main research results in an understandable format [14]. The project makes use of a simulation of the power grid combined with forecasted data and hardware components.

The combination of hardware and visualization to enhance users understanding of a specific topic is the goal of the project. It has been done in science museums and the combination of physical elements and virtual visualization proved to give a more engaging experience than only a virtual model [17]. This was discussed in a paper where a virtual touchscreen environment with virtual components was compared with a virtual table including physical components [15]. The hardware of this project will enable the project to educate the users [1]. A lot of documentation has been published on how humans interact with computers. This project is about making an interface for users to interact with a simulation. The main benefit that the hardware brings is a sense of touch, which is an important human sense, and touch can communicate a lot of information to a person [16].

The purpose of this project is to create a power grid simulator that enables the user to influence the scenario through a hardware controller with interactive controls and present pre-selected scenarios that pose challenges to the user. The controller should be simple and intuitive to use. A forecasting component should predict the upcoming load of the scenario. The simulator should be kept simple enough for an interested person without expertise in power grid dynamics to understand what is happening. This is done to give the user a sense of how the grid works, what possible problems can arise, and how those can be fixed.

The project is working towards a power grid simulator that can be implemented in a future power grid room. Such a simulator allows people with basic power grid knowledge to simulate different scenarios on a power grid. The project focuses on the design steps towards a functional prototype that are made to make sure future iterations of this prototype can be implemented smoothly.

To create a power grid simulator, a visualization of the power grid has been made using the game engine Godot. To control this simulator, hardware components are needed that can be used to give user input to the simulation. These inputs are used while the simulation is running. To enable communication between Godot and the hardware, an Arduino micro-controller and a C# library are used. The scenario that will be shown in the simulation heavily depends on the real and suitable data that is available to give an accurate representation of a real grid. The dataset must also be extensive enough to train a forecasting algorithm that will encourage the user to interact with the grid based on its predicted

values. The scenario that was chosen is the Dutch aggregated transmission grid of 220KV and 380KV using data from 2018.

After the simulator has been turned on, the Forecasting & Scenario Selection module will find the parameters for the selected scenario and predict the corresponding 24-hour load forecasts for the full duration of the scenario. The parameters and predictions will be sent to the Visualization & Simulation module, where the power flows will be calculated and the grid will thereafter be displayed. The user will then be able to interact with the simulator using the various hardware components, and the Hardware module will transmit the information to the Visualization & Simulation module so that the grid can be re-simulated and re-displayed. Three subgroups have been devised to work on the aforementioned modules. The subgroups are the forecasting subgroup, the visualization subgroup, and the hardware subgroup. The following figure shows how the subgroups are connected.

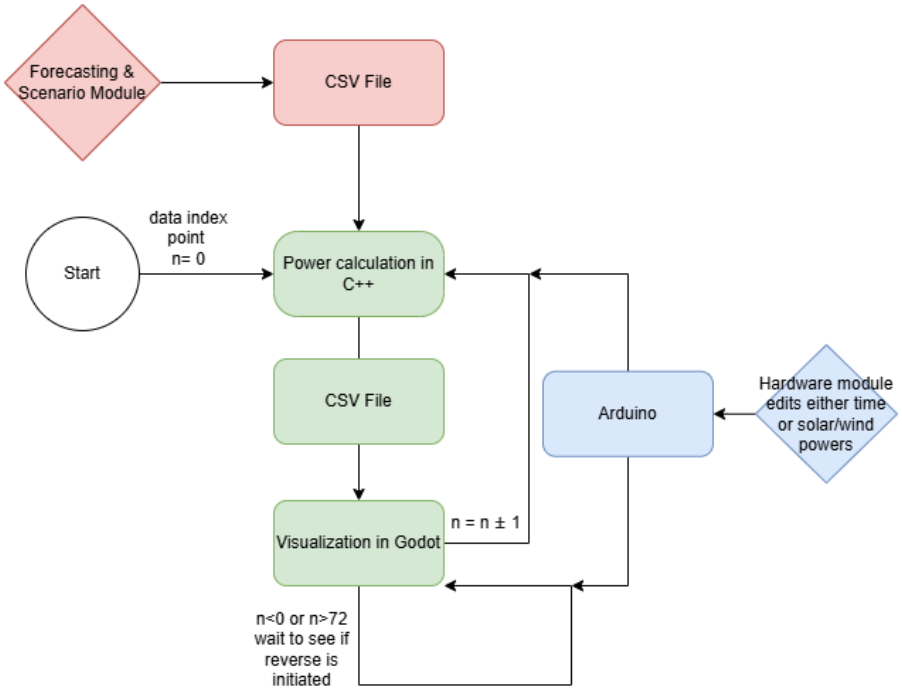


Figure 1.1: Interconnection of the subgroups in the project

This report is about the hardware subgroup. In this report, the first thing that will be discussed is the program of requirements. Then, a section on the prototype design and hardware implementation will follow the discussion of the design of the entire hardware system and its subsystems. The final part is the conclusion and the discussion.

2

Program of requirements

2.1. Problem definition

The hardware subgroup must develop the hardware that the user will use to interact with the power grid visualization. It consists of coding a micro-controller to implement different components that the user can interact with and change the simulation, and it is necessary to integrate the hardware with the game engine used for visualization. It leads to a problem statement:

The simulator needs to be controlled using external components that enable the user to control the visualization using hardware components. These hardware components are connected to the other modules via a computer and enable the user to start working with the simulator.

The requirements were decided based on the future use of the simulator, which would be used in a power grid room. Thus it must serve as an exhibition, allowing people with less understanding of power grids to also be able to interact with it. Therefore the following requirements were come up with in collaboration with the supervisor and other subteams of this project.

2.2. Global requirements

The project has global requirements to make sure all the subgroups work towards the same goal. These requirements are the following:

- The user must be able to interact with the simulated power grid and see how it affects the power grid dynamics.
- The user must be able to see useful information about the grid, such as power generations and loads or line loadings.
- The user must be able to access forecasted values of the grid, which can be useful for taking actions in the grid.
- The user must be able to change the portrayed time in the simulation.
- The modules must be able to send and receive data between one another in an agreed-upon format and form.
- The system must use a working solar and wind generator in real life that links to the scenario.

2.3. Requirements of the hardware

Next to the global requirements, the hardware requirements have been written out.

2.3.1. Mandatory requirements

- The hardware modules must be safe to interact with for the user. The user must not experience health implications by using the hardware.
- All the hardware components must have a function that can be used to control the power grid simulator or change variables in the power grid simulator.
- The hardware builds must be re-executable. A future team must have the knowledge to build the hardware from scratch.
- The hardware must consist of a component that lets the user change the simulation time. The user must be able to control the direction of time and stop/start the simulation.
- The hardware must incorporate solar into the design. To enable the user to change the amount of solar received in a given node inside the power grid simulator.
- The hardware must incorporate wind into the design. To enable the user to change the amount of wind received in a given node inside the power grid simulator.
- The hardware must be able to receive information from the computer that is running the simulation. This must be done without the simulation stopping or experiencing delay. To enable the user to have feedback on the situation on the power grid.
- The hardware must be able to send information to the computer that is running the simulation. This must be done without the simulation stopping or experiencing delay. This is done to let the user control the simulation with the hardware.
- The hardware must have a safe and reliable power supply. To supply the components with power.
- The hardware modules need to be self-explanatory, when a user walks in and sees the final design it needs to be clear what the components do either by their design or signs drawn upon them.
- The hardware should have components indicating the situation on the simulated grid. To give the user feedback on whether the power grid is in a stable state or not.

2.3.2. Trade-off requirements

- The hardware shall be as simple to use as possible. To make sure the user knows how to use the components that control the simulation
- The hardware should have a good aesthetic casing.
- The hardware should minimize the complexity of controlling all the hardware components by using a micro controller.
- The Hardware component used for controlling time should be a motorized wheel.

3

Design process

3.1. Preliminaries

For this project, numerous different programs and coding libraries were used to create a working simulator. The most important libraries and programs that were used in the hardware part of the project will be listed below.

3.1.1. Godot game engine

The Godot game engine is used for the visualization of the power grid. Godot is an open-source game engine in which both 2D and 3D projects can be made. The game engine was chosen because of its support for the coding languages C++ and C#. Godot makes use of GDSCript, an internal programming language. Godot is based upon nodes, allowing for scenes and games to be built up consisting of different smaller parts working together. The fact that Godot is open source and has support for C++ made it usable for this project.

In Godot scenes can be created to make simulations or games. These scenes are built up from different nodes that can be programmed to have a certain function. These nodes can, for example, control time, certain lines, or the whole background map. The nodes that were used for this program were built up In two different ways, certain nodes were constructed using the Godot C++ extension library, in which the programming language C++ can be used. Other nodes were programmed by connecting a script to the node and programming in C#. These two methods worked together, which was a major advantage of using the Godot game engine. For the hardware part, a lot of the code was programmed using C# scripts. The visualization primarily used the C++ Godot extension. An example of the scene that was used for this project is seen in the following image.

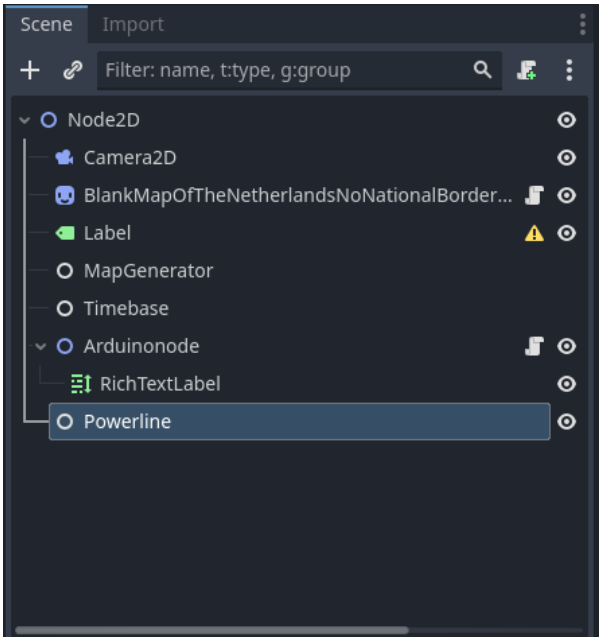


Figure 3.1: A scene in Godot consisting of several nodes

3.1.2. Arduino IDE

The Arduino IDE is a platform used to control the Arduino micro controller. The platform can be used to write code to the Arduino and test if the system works. In the application, a lot of libraries are available to use inside this program. This platform supports the coding language C++, which enables the Arduino micro controller to execute the required functions and enable the hardware to communicate with the computer. [11]

3.1.3. System.IO.Ports

This is an aNET library that is used in the Godot game engine to get the inputs via the COM ports of the personal computer from the Arduino. The library is made by Microsoft and is useful for implementing hardware in a project like this.\

3.1.4. KiCad

The KiCad software was used to develop the PCB chips for the circuits that were designed for all the hardware modules. Kicad is a program where you can make circuits, transform these circuits into PCB chips, and then output a file to print this PCB chip. The Kicad software has a lot of components included. This made finding the footprints of the components that are used for the project easier. The Kicad platform was used to make a logic design of how the final hardware setup would be and which microcontroller pins should be connected.

3.2. The hardware components

The program of requirements gives a good starting point for the global design of the hardware system. The modules needed to implement a design that validated all these requirements. The hardware components must be recognizable to the user to make sure they know how to use the hardware [2]. This is why it was decided to go for things like buttons and knobs to control the simulation instead of things a much smaller part of the predicted users are familiar with, such as game controllers. In hardware design, it is important to keep in mind how users will interact with the components. McNamara and Kirakowski (2006) propose a method of evaluating human-computer interfaces that consists of evaluating three elements: functionality, usability, and user experience. [19]

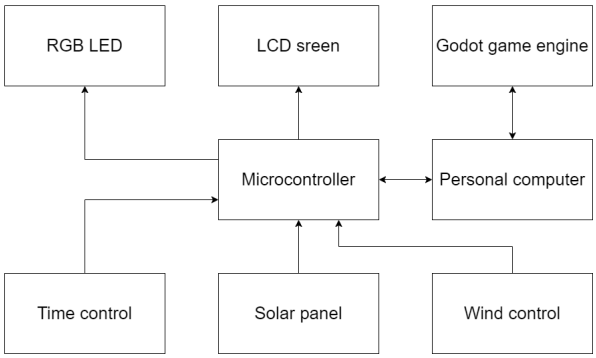


Figure 3.2: The hardware setup

The design must include components that let the user control the simulation time, the wind generation, and the solar generation. The complexity of implementing solar and wind in the design was to keep it as simple as possible. [10] So components that output signals to control these internal variables were designed. Next to this, a component was needed that showcased to the user what the current situation in the simulation was. This was done to make sure the hardware could receive data from inside the simulator and to make sure the user had a feedback system. This gave the following hardware setup: 3.2.

As can be seen in the figure, the final design consists of a micro controller connected to four different modules, the four different modules are:

- The component that controls the simulation time
- The component that shows the situation on the Grid
- The component that controls the wind generation
- The component that controls the solar generation

The microcontroller was needed to enable communication between the computer and the components. The microcontroller had to read user inputs for wind generation, solar generation, and the time controller. The second function of the microcontroller was to control the RGB LED and the LCD screen to give the user feedback on what was happening in the simulation. Lastly, a method of combining all these modules into a final design was developed, which consisted of combining all these different components on a single PCB. In the next part, the design of every part of the hardware will be elaborated upon.

3.2.1. Micro controller

To control all the individual hardware components, a device was needed that could connect to all these individual components and on which code could be uploaded, to make sure the hardware could connect to the computer. After multiple options were considered, it was chosen to make use of an Arduino Mega Rev3 development board. This board makes use of the Atmega2560 AVR micro controller.

One of the main strengths of using an Arduino development board is the library support for interfacing with hardware sensors. An Arduino board does not make use of a full operating system, unlike a Raspberry Pi. Making use of a full operating system would be overkill for the requirements of this project since mainly simple read and response actions are needed. This allows the Arduino to be more predictable in real time, reacting very fast to inputs from hardware components and actuators. Arduino boards provide many digital I/O pins and analog input pins. Especially these analog pins are missing on a Raspberry Pi, making hardware integration with an Arduino more easy. Last but not least, the Raspberry Pi's GPIO pins are not 5V compatible, requiring 3.3V. For example, the DC motor driver

for the wind controller needs 5V inputs. This also makes integration more difficult [21].

The Arduino Mega development board was then chosen over other Arduino development boards. This is due to the high number of input and output pins needed for this project. The Arduino Mega provides 54 digital I/O pins and 16 analog inputs, whereas the Arduino Uno only provides 14 digital I/O pins and 6 analog inputs. Many digital I/O pins are needed to connect all hardware components; for example, the LCD screen alone would already need 6 pins. This makes the Arduino Mega better suited for more complex, multi-component projects like this. Also, much room is left for expanding the hardware controller in the future.

3.2.2. Time controller

To control the simulation time, a hardware component was needed that could pause and play time and that could change the direction of time. At the start of the project, it was discussed to develop a time-wheel connected to a brushless DC motor via a shaft. Connecting a rotary encoder to this setup allowed for control of this setup. This idea was scaled down because the component that controlled the time did not have to be that complex. This would only decrease the usability of the design, so it would not help a user control time but make it unnecessarily difficult to control time. [19] It was decided to use a rotary encoder in combination with a button.

A rotary encoder works in the following manner: it has two output pins with signals A and B and a common pin. The output pins make contact with the common pin, which generates two square waves. Due to the placement of these output pins, the two square waves are 90 degrees out of phase. The rotary encoder measures the rotation direction by measuring the state of output B at the falling edge of output pin A. If the state of B is low, the rotation direction is anti-clockwise; if the state of A is high, the rotation direction is clockwise [4].

The rotary encoder that was chosen for this was the Bourns PEC11R-4225F-S0024 [23]. This is an incremental rotary encoder that has an internal button. This made the component really useful for the project because it could emit all the signals needed to control the simulation time. This component was included in combination with the filter circuit described inside the provided data sheet to make sure the output signals worked adequately. The filter circuit used can be seen in the following figure.

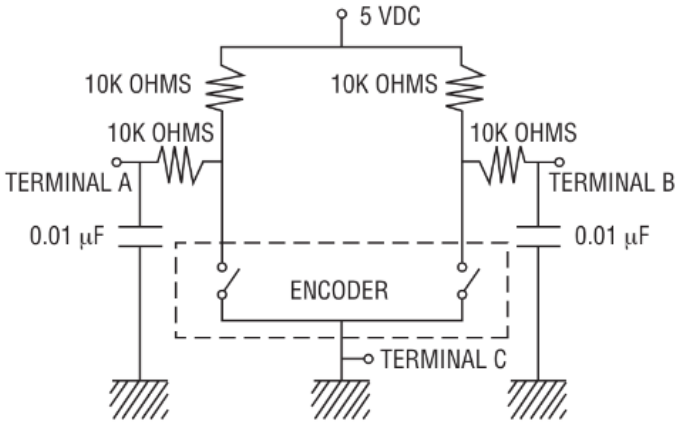


Figure 3.3: Filter circuit of the rotary encoder

This filter reduces high-frequency noise. The implementation of the rotary encoder into the prototype and the testing of the controller can be found in chapter 4.

3.2.3. Solar generation

As stated in the requirements ??, the hardware must incorporate solar energy in the design, and this design of the solar should be self-explanatory. Therefore, it was chosen to make use of a small solar panel to provide an input for a Solar generator in the power grid simulation. This is done by measuring the power generated by the solar panel and scaling it to a value between 0 and 700 MW. It is then transmitted to the simulator. This value was chosen to be realistic in combination with the forecasting model of the simulation, which is based on a dataset used for an open data-based model of the Dutch high-voltage power system [25].

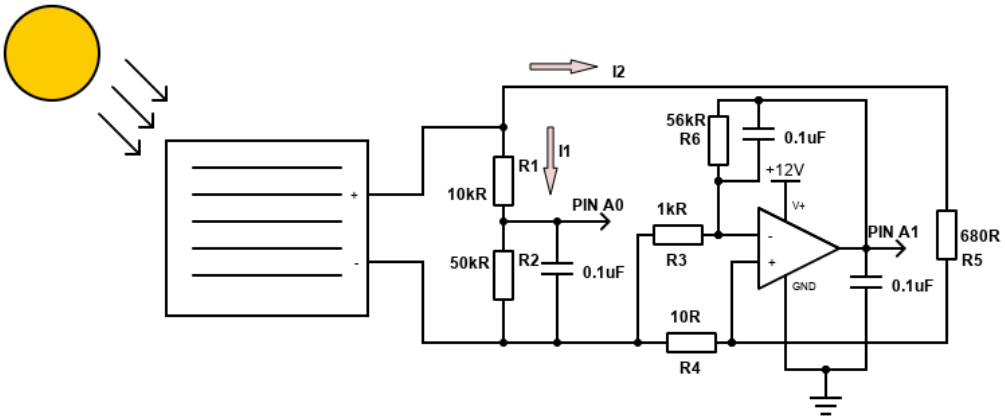
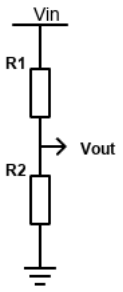


Figure 3.4: Power measurement circuit of the solar panel

To measure the power of a solar panel by using a microcontroller, a power measurement circuit has to be set up. The circuit design can be seen in figure 3.4. The solar panel used is capable of generating 300 mW with a maximum voltage of 6 volts. Therefore, the maximum current it is able to generate is 50 mA. Considering the Arduino analog pins need a voltage between 0 and 5 volts, a voltage divider is needed. For resolution purposes, it is best to use as much of the range as possible. Therefore, the voltage divider should scale 6 volts down to 5 volts.



$$\frac{V_{out}}{V_{in}} = \frac{R_2}{R_1 + R_2} \tag{3.1}$$

Figure 3.5: Voltage divider

Considering equation 3.1, the resistor values were chosen to be $R_1 = 10k\Omega$ and $R_2 = 50k\Omega$. These resistor values were chosen considerably higher than the load resistor. This is done to make most of the current go through the load and make the power consumption of the voltage divider as low as possible. Then the voltage is measured with the analog A0 pin of the Arduino. To prevent high-frequency noise and smooth transient changes, a capacitor is placed in parallel with R_2. The value of this capacitor is based on the cutoff frequency for noise and the resistance.

To measure the current, a 10 ohm resistor is used in series with the load resistor. This resistor causes a small voltage drop. A non-inverting amplifier increases this voltage drop by 57 times with a gain of 0. Based on the voltage drop, the current can be calculated using Ohm's law. Since the resistor is in series with the load, the current will be the same. Since the solar panel maximally generates a voltage of 6 volts, the maximum expected current through the 10 Ohm resistor can be calculated. The values in the following equations are based on figure 3.4.

$$R_{eq} = (\frac{1}{R_1 + R_2} + \frac{1}{R_4 + R_5})^{-1} = 682.15\Omega \tag{3.2}$$

$$I_{total} = \frac{V_{solar}}{R_{eq}} = 8.8mA \tag{3.3}$$

$$I_1 = \frac{V_solar}{R_1 + R_2} = 0.1mA \tag{3.4}$$

$$I_2 = \frac{V_{solar}}{R_4 + R_5} = 8.7mA \tag{3.5}$$

First, the total current through the system is calculated; this is done by calculating the equivalent resistance of the circuit 3.2. Then the total current through the circuit, inclusive the voltage divider for voltage measurement, can be calculated 3.3. After calculating both current going through the voltage divider 3.4 and through the load and 10 Ohm resistor 3.5, it can be found that only 1 percent of current, and thus power, is lost on measuring the voltage. This is the reason why the voltage divider resistors were chosen to have such high values, to minimize their power consumption. For this application, the choice was made not to use a shunt resistor for the current measurement. The reason for this is the extremely small voltage drop the shunt resistor would have, since these resistors are in the range of mili Ohms. These resistors are made to handle high amounts of power, which is, in our case, not applicable. As said before, the maximum amount of power a solar panel can generate is 0.3 watts. A regular 10 Ohm resistor, which creates a greater voltage drop, is more easily suitable for that. In this way, a lower gain is needed for the amplifier, and power loss still remains very low.

3.2.4. Wind generation

To enable the user to control a wind generator in the simulation, an interactive, clear control method had to be designed. The first idea was to use the same type of power measurement circuit as described before in the solar part. Then a small DC motor with a small fan would act as a generator. The user would then be able to blow air into the fan to generate electricity that the circuit could measure. Depending on how hard a user blows into the fan, the Arduino sends a power value to the computer running the simulation. However, during testing, it was found that by blowing into the fan, only a very minimal generation occurred. It would also not be very comfortable for the user to constantly blow into a fan to perform a control action in the simulation. Therefore, a different approach was chosen.

A more user-friendly way to control the wind generation in the simulator would be to control it with a potentiometer slider. This would give an input between 0 and 900 MW; this is based on the model the forecasting is using [25]. Then the inputted power is visualized using a DC motor with a fan spinning faster or slower based on the user input. The motor is controlled using a driver, which is in turn controlled by a PWM signal from the Arduino. The dc motor has an external voltage supply since it would draw too much power from the Arduino development board. The full schematic can be seen in the figure above 3.6. More information on this can be found in the section about the power supply 3.2.7.

The L298N driver is designed to control up to two DC motors or one stepper motor. For this application, only one DC motor is used. One of the advantages of using this driver is the separation of the source and DC-motor from the micro controller. Electromotive Force (EMF) voltage spikes will not be able to do damage to the Arduino. The driver can also control the motor in a two-directional way. In this application, only one direction is used. The driver contains an H bridge using NPN transistors [9]. The

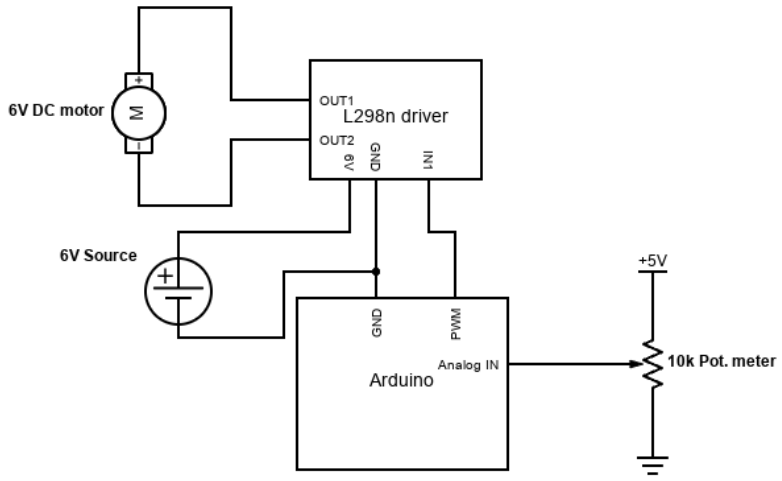
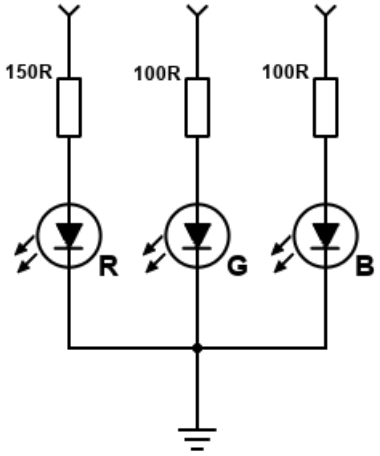


Figure 3.6: Control circuit of wind generation

driver board has a voltage divider that gives a steady voltage of 5V.

3.2.5. RGB LED

To enable the user to get feedback from the grid, the RGB LED was implemented. An RGB LED is a light source that can have three different LEDs. A red, green and blue one. These can be used to implement almost all colors. The RGB LEDs have three internal LEDs that share either a common anode or common cathode. The RGB LED that is used in this project makes use of a common cathode. This means all LEDs have a common ground. To control the colors, a PWM signal is connected to the pins of the internal LEDs.



$$R_{red} = \frac{V_{source} - V_{red}}{I_f} = \frac{5V - 2V}{20 * 10^{-3}A} = 150\Omega$$

$$R_{green} = R_{blue} = \frac{V_{source} - V_{green/blue}}{I_f} = \frac{5V - 3.2V}{20 * 10^{-3}A} = 90\Omega$$

Figure 3.7: Connection circuit of RGB LED

The LED pins are connected through a resistor to the Arduino, giving the PWM signal. This resistor is calculated based on the data sheet of the RGB LED [5]. The forward voltage of the R LED is typically 2 volts; for the G and B LEDs, this is typically 3.2 volts. The forward current of all LEDs is 20mA. After calculating the voltage difference with the supply voltage, Ohm's law can be applied to calculate

the needed resistance.

The function of the LED is to communicate data from the simulation to the user. This is done to give the user an indication of the situation on the grid. The grid has transmission lines; in the simulation, these lines get a certain color based on their loading. A green, orange, or red light is possible. The green light needs only the G part of the LED to be activated, the orange light needs a combination of R and G to be active and the red light only needs the R part to be active.

3.2.6. LCD screen

The Hardware wind and solar components were connected to an LCD screen to see what the Arduino would output and to give the user some feedback on what the current value of wind and solar generation would be. The LCD screen was implemented for user feedback but also for testing purposes. Due to the LCD, variables could be plotted on it without the need for a computer.

3.2.7. Power supply

The power supply has an incredible importance in a system. The design of it is often left until the end of a project since bench supplies do the job perfectly for testing purposes. [3]. For this project, the choice was made to use a 12V power supply. This would be used directly to supply the LM358 of the solar power measurement circuit 3.2.3. This provides a large overhead voltage for the 4.95V the operational amplifier should output maximally. These 12V power supplies are widely available on the market to buy for low prices.

The power supply needs to convert 230V AC to 12V DC. In this way, the hardware controller can just be plugged into a plug for it to work. The choice was made to buy a pre-built power supply for this. Due to the low safety risks. Working with 230V brings serious risks. When choosing a pre-built power supply, many options are available. There are mainly two types of power supplies: linear power supplies and switching power supplies. A linear power supply uses a transformer to step down the voltage and then a rectifier circuit to make it a DC voltage. These types of supplies have very low electrical noise and are simple, low-cost, and robust. However, it is not very efficient; a lot of energy is wasted and converted to heat. But these types of power supplies are not very flexible, usually requiring a fixed input voltage [3].

Another option is using a switching power supply; these are usually more energy-efficient and flexible. However, they require much more complex circuitry.

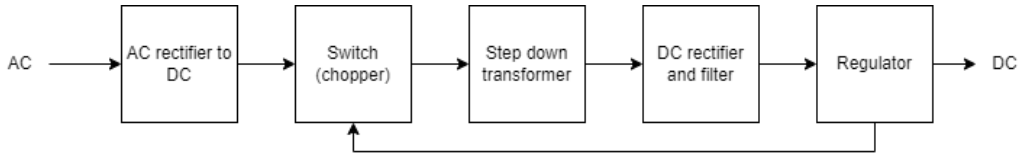


Figure 3.8: Functional diagram of a switching power supply

In figure 3.8, the basic principles of a switching power supply are visualized. First, AC is directly converted to DC without first stepping down. The result is high-voltage DC. Then, by frequency switching, a high-frequency square wave is made. The amplitude of this square wave is then stepped down by a transformer. After the transformer, it is rectified to DC again and goes through the regulator, resulting in a lower DC voltage. The regulator gives feedback to the switch on which frequency to switch to get the desired voltage.

In our design, a voltage of 6 volts is required for the DC motor when controlling the wind. This could be done by using a voltage divider, but that would be extremely inefficient. When used for high voltage reductions and high power applications, they dissipate a lot of power as heat. A buck converter steps

down the voltage by increasing the current. This results in a high efficiency since the inputted power is almost the same as the outputted power. Therefore, a buck converter would be used to step down 12V DC to 6V DC.

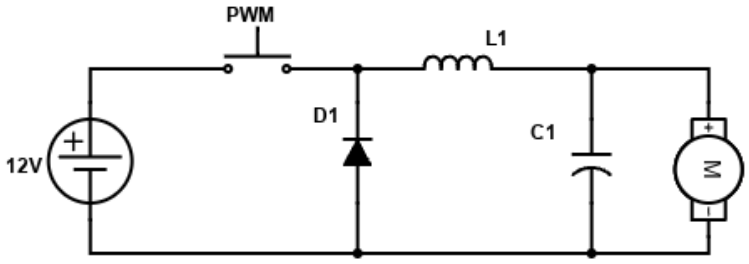


Figure 3.9: Schematic of a buck converter

In figure 3.9, a basic schematic of a buck converter is shown. When the switch is closed, the inductor L1 is directly connected to the source; in this case, that would be 12 volts. The current through the inductor will start to increase, and energy will be stored in the inductor. The capacitor will charge during this period. When the switch opens, the diode becomes forward-biased, and the inductor will start to discharge its stored energy. If the voltage across the capacitor, induced by the collapse of the magnetic field of the inductor, is higher than the voltage across the capacitor, it will still charge the capacitor. Otherwise, both the inductor and capacitor will discharge through the motor.

$$V_o \approx V_{in} \cdot DC \tag{3.6}$$

Formula 3.6 describes the role of the PWM driving the switch in the buck converter formula. Usually, this switch is a MOSFET. The output voltage has approximately a linear relationship with the Duty Cycle (DC). For this project, the DC would be set 0.5 to reduce 12V to 6V.

3.3. Communication with Godot

To make the hardware usable inside the Godot game engine and to enable the user to interact with the simulator, a method that enabled communication between the micro-controller and The Godot game engine was needed. Three different methods were considered to make the communication happen. In the following paper, the serial communication between Arduino and Godot was established. [22]

The first method consisted of downloading an open source library[24] that could be included in the Godot project to add communication methods to the already existing nodes in Godot. The benefit was that this method was used successfully by the visualization team to create a working project that could be programmed using C++ code and C++ libraries. This was implemented using Scons, a software construction tool that builds the project using Python scripts, and the Godot C++ repository. To implement this method, the main files from the libraries needed to be cloned inside the project files of the Godot project and then compiled using SCons.

The second method was using a C# library called System.IO.Ports and using the Script function of Godot to create different signals which could then be emitted to other nodes in the Godot project. The benefit of this method was that it used a coding library that was already used in several projects.

The third method was to make the Arduino micro-controller write keyboard inputs towards the computer, enabling Godot to execute different commands depending on the keyboard inputs the Arduino would give the PC. A disadvantage of this method is that the Arduino would take over the keyboard so no inputs could be given to the PC.

It was decided to go with the first method of downloading a library that could incorporate serial communication between the Arduino and the Godot engine. This open-source serial communication library had a short installation manual on GitHub. Due to the outdated library, the installation kept failing. So this method proved to be unusable. There were several other open-source GitHub libraries, but none were well-documented. So it was decided to go for the second method.

This was an implementation of a serial connection using the System.IO.Ports library inside a C# script. These were implemented, and a C# code was developed that changed a rich text file inside Godot. The signals were created in the code and outputted using the “emit.signal” function of the library [8]. After this, it was necessary to specify inside the functions of the visualization team how to call the correct signal that changed the values, for example, solar and wind generation or that controlled the timebase.

To enable the hardware modules to handle signals sent from the simulation, the microcontroller and the code controlling it had to be able to receive data. The serial write methods of the Serial.IO.Port C# library were used to write the data to the Arduino. The Arduino has a library contained in its software package to receive serial communication. [12]

3.4. PCB Design

The last step of the hardware design was developing a working prototype. This prototype had to be a combination of all the different components and include methods of explaining to the user what is happening or how to change certain simulation parameters.

For the prototype, a PCB chip was needed, on which all the connections between modules and the microcontroller were implemented. The PCB consists of an Arduino shield on which the Arduino can be mounted and it was designed to contain all the different hardware components needed in the final design.

The software used in the design of the PCB is KiCAD. This platform was chosen because of its accessibility and the availability of footprints for this certain program. [7] The KICAD software provided a good interface and a wide array of hotkeys and functions. [13] In this program, the first step is designing a schematic using components. The next step after the connection between the different components is set up is assigning footprints, either from the provided libraries or from manufacturers or suppliers of electronic components.

The next step in the PCB design is implementing the actual design for the complete hardware module. This is done by placing all the footprints in the correct manner in a PCB editor. In this editor, the connection which will be printed on the actual PCB can be drawn. For the design, it was important to pack everything as close as possible; this was done to keep the final hardware module as small as possible.

4

Prototype implementation & Results

4.1. Hardware components

For the prototype implementation of the hardware, all the different modules were tested individually. The complete setup was tested as the final step. The hardware circuits were tested using a breadboard. The hardware components were controlled using code that can be found in appendix ??.

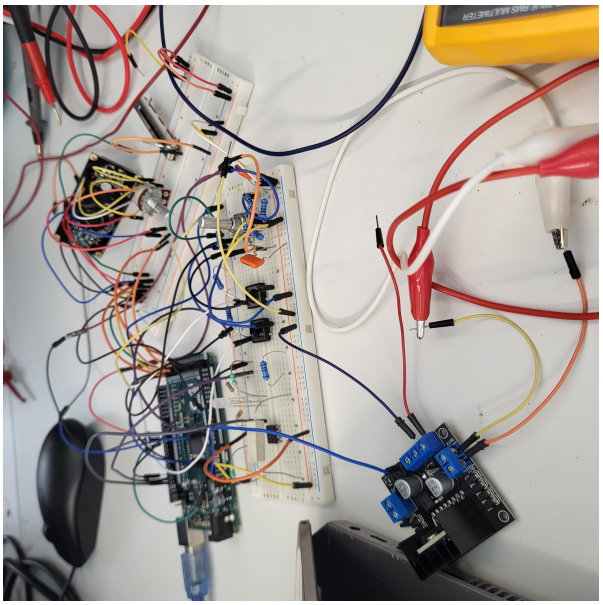


Figure 4.1: Test Setup

4.1.1. Time controller

To test the time controller, the rotary encoder was connected to the Arduino using the filter circuit as described in figure ??. This was done by using two capacitors and 4 resistors. To connect the switch of the rotary encoder, a pull-up resistor was used. The Arduino pins that were used for this connection can be found in table 4.1.

Table 4.1: Arduino connection of the rotary encoder

Arduino pin	Connection
Digital 2	Terminal A
Digital 3	Terminal B
Digital 38	Switch

To use the rotary encoder to control the time, a communication protocol was developed with the Godot engine. The rotary encoder had to output -1 or 1, depending on the rotation direction. Next to this, a 2 or 3 was sent if the simulation had to stop or start. In the section about the Godot communication, this will be discussed further.

To make sure the rotary encoder outputted the required signals, a function was developed and included in the code that was uploaded to the Arduino. An open-source library was used in this code called Rotaryencoder [18].

To test the design, the serial monitor of the Arduino was used. In this tool of the Arduino IDE, you can see what gets written to the computer's serial port. To test the rotary encoder, it was checked to see if the outputs provided were the correct responses. After this was found to be correct, the Rotary encoder was connected to the Godot game engine to see if it outputted the correct values, and the correct implementation was executed. The Godot C++ code that was changed to implement the rotary encoder was the timebase code found in Appendix A.2. In this code, which was made by the visualization sub-group, two functions were added to implement the change in time. A connection was set up between the timebase file and the file where the signal was defined. This is the Arduino file found in the appendix A.1 that is connected to the Arduino Node in Godot. A function was added that changed the flow of time depending on which signal was outputted to the Godot engine.

This setup was tested rigorously to see if the flow of time changed. This was done by using a text file to see what signals were sent to Godot. To see if the direction of time changed, print statements were added to the code.

4.1.2. Solar generation

To implement the design for the solar component, the components were ordered, and the designed circuits were connected to the solar panel and to a power supply. To measure the output, the voltage divider and the operational amplifier were connected to the Arduino. The pins that were used on the Arduino can be found in the following table ??.

Table 4.2: Arduino connection of the solar circuit

Arduino pin	connection
Digital 13	voltage divider
Digital 12	output operational amplifier

There were some difficulties because of the LM358 operational amplifier that was used. One side of this amplifier was faulty so the circuit did not seem to work. But these were fixed in time and the circuit proved to be working. in the following figure, the testing setup is provided.

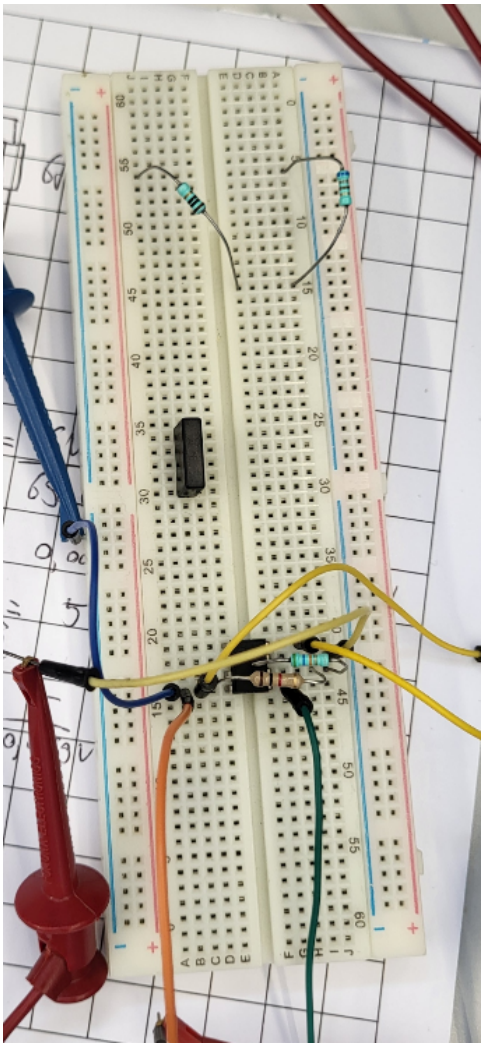


Figure 4.2: Solar testing circuit

While testing the solar circuit connected to the solar panel, the measurements provided proved to be quite noisy. The value of the solar generation variable kept varying. To compensate for this, a filter was implemented in the software. The algorithm that was chosen was the exponential filter algorithm [20]. This method was chosen over several other software filtering methods, like averaging or taking the running average. These other methods cause a delay in measuring since many measurements have to be taken to then take the average. These types of filtering take up some memory, whereas the exponential filtering algorithm only remembers its last measured value.

$$Y_n = w.X_n + (1 - w) \cdot Y_{n-1}; \tag{4.1}$$

This equation4.1 shows how the measurements are filtered. Y_n is the filtered measurement, and X_n is the actual measurement. The parameter w is a weighting parameter, having a value between 0 and 1. When the weighting parameter is high, it doesn't smooth the measurements much, but it is very responsive. When it is low, it does smooth a lot, but it is very slow in responding to changes in measurements. Another advantage is that this filter can be adjusted by only changing the weighting parameter. The code implementing this filter can be found in the appendixA.3. To regulate the amount of power the solar panel is producing, a light bulb would be present.

4.1.3. Wind generation

The Wind implementation was tested in three different steps. Firstly, the sliding potentiometer was connected to the Arduino, and a code was written, which can be found in Appendix A.3, that gave the wind value to the Arduino and the computer. The sliding potentiometer was connected to the Arduino in the following way:.

Table 4.3: Arduino connection of the sliding potentiometer

Arduino pin	connection
analog 7	pin 1
5 volts	pin 2
Ground	pin 3

The potentiometer has been proven to be a little bit noisy during testing. When not moved, the analog reading sometimes slightly changes. To get rid of this, a simple if statement was used in the code to check if the reading has changed by more than a certain value before implementing any changes.

4.1.4. RGB LED

The RGB LED was implemented in combination with three resistors to make sure the correct colours were emitted. The RGB led was connected to the Arduino in the following way:. [6]

Table 4.4: Arduino connection of the RGB LED

Arduino pin	connection
Digital 13	Blue
Digital 12	Green
Digital 11	Red

4.1.5. Power supply

For the power supply, as described before 3.2.7, a prebuilt switching power supply is used. For converting 12V to 6V for the DC motor, a prebuilt buck converter module is used, a TC-9927140. The cost of buying the individual components would have been higher than buying a mass-produced, prebuilt unit. The reliability and performance of this unit have been tested.

4.2. Communication with Godot

To implement the communication method, different coding files were used, which can be found in Appendix ??. Some additions belong to these files written by the simulation team to enable the simulator and the Godot game engine to act upon signals sent by the Arduino through the serial communication port. The same holds for the signals sent by the files inside of the game engine towards the Arduino.

To be able to use signals in different codes, it was necessary to set up a connection between these nodes. This was done by using the relative path between these nodes and using callables to set up the functions for which the signals could be used.

4.2.1. Sending data to Godot

The signals that were sent to Godot consisted of Strings. All the signals and their functions in the simulator are listed in the table below.

Table 4.5: Signals emitted to the Godot engine

String	Function
-1	go backward in simulation time
1	go forward in simulation time
2	play
3	pause
Sxxx	change solar to xxx
Wxxx	change wind to xxx

The code of the visualization team handled the signals inside the Godot engine. There are four signals that control the simulation time. The go forward and go backward signals are 1 and -1. They get emitted if the rotary encoder is turned. A 1 gets emitted if the encoder is turned clockwise, and a -1 is emitted if the encoder is turned counterclockwise. The pause or play signal, so a 2 for play and a 3 for pause, gets emitted if the user presses the rotary encoder and it stops or starts the simulation time.

The signal controlling the wind generation is used in the visualization and is transmitted whenever the user changes the wind input with the sliding potentiometer. The signal outputted is used in the simulation as a value for the power generated in a wind farm used as a generating node.

The signal used to change the solar generation is gathered by the solar panel module of the hardware and is emitted to the simulation whenever the user presses a button. The value is used inside the simulation for the power generated in a solar park.

4.2.2. Receiving data from Godot

The signals that were sent to the Arduino can be seen in the following table.

Table 4.6: signals received from the Godot engine

Signal	Function
1	Turn RGB LED green
2	Turn RGB LED yellow
3	Turn RGB LED red

All of these values were sent depending on the number of overloaded or almost overloaded lines inside the simulation. The signals control the color of an RGB LED controlled by the microcontroller. The signal

emitted to the Arduino was a three if a line exceeded its maximum rated capacity in the simulation. If the Arduino receives this signal, the LED turns red. If there are more than five lines higher than 95 percent of their rated capacity, the code writes a 2 to the Arduino, and the RGB LED turns yellow. If both of these conditions are not met, a 1 gets emitted and the RGB LED turns green.

4.3. PCB design

To implement the final schematic to make a PCB, all the different circuits were combined into a big schematic that showed the total hardware modules. The center of this hardware module is the Arduino. All the components were connected to the Arduino in such a manner that it would fit on as small a PCB as possible.

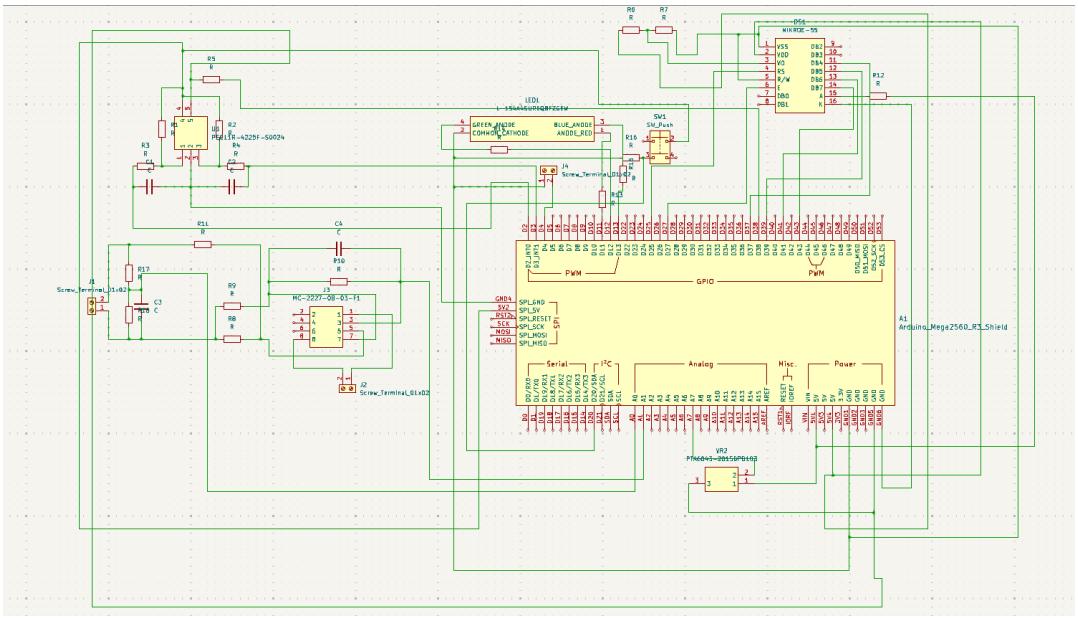


Figure 4.3: The schematic of the total design made in the program Kicad

The schematic was made using as many of the footprints as the actual components to make sure they would all fit exactly as expected. The connections were made in the smartest way possible, so all the components would fit around or on top of the Arduino. The schematic was made using a shield for the Arduino Mega. On this shield, pins can be soldered, and this shield can be placed on top of the Arduino, making the Arduino removable. This is handy if problems arise with this component.

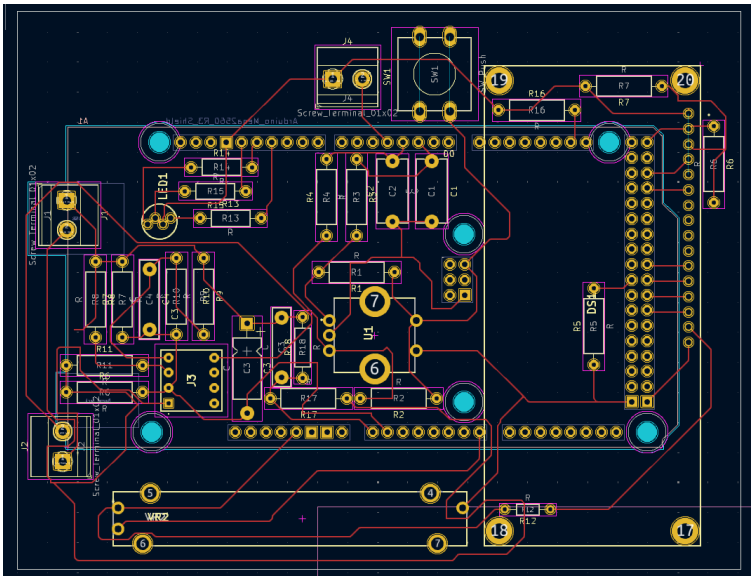


Figure 4.4: The final design to implement the PCB

The PCB design that can be seen in figure 4.4 is made using the Kicad software, and the schematic can be found in figure 4.3. This was done by placing all the components as closely together as possible to make the PCB cheaper. Some components were placed on top of each other. This is because the Arduino is placed on the bottom of the shield, so components can be soldered on the top part. The LCD screen is also placed over the shield and a couple of resistors. This is because it needs to be visible to the user when it is using the hardware module. The button and rotary encoder are also placed on top, so they can be accessed and controlled if the PCB is connected to the Arduino. The RGB LED was connected in such a manner that it could be seen from the user's perspective. The connections between components are made using 0.5 mm wide connections. These are all placed in a way that minimizes the risk of disconnection. The things that are not on the Arduino but will be connected using screw terminals are the following.

- The power supply
- The buck converter
- The L298N motor driver
- The dc motor
- The solar panel

All of these components need to be fitted inside the casing as well. The footprints for a couple of components were missing to test the PCB design, it was printed out and the necessary changes were made to a couple of footprints of certain components to make sure they could be fitted on the final PCB design.

5

Conclusion

To conclude this report, which described the building of a hardware controller for a power grid visualization, the hardware had to be implemented in the visualization and had to have specific functions. The final product that was designed validated the requirements that were stated in the program of requirements. The hardware module contains components that control solar generation, simulation time, and wind generation. The hardware components that were designed are safe to use, have a functional use in the power grid simulator, can be controlled using a micro-controller, and can change parameters in the Godot simulator.

This thesis demonstrates a harmonic interaction between a simulation based on software and a hardware controller based on hardware and software. Also, it highlights the practical challenges faced and explains the accompanying innovative solutions.

6

Discussion and recommendation

6.1. Discussion

For the hardware, the problem definition was the most difficult part of the project. It was hard to define the boundaries for what was achievable within the time frame of the project. Different designs with different complexities were considered. For example, in the first iteration of the time controller, a rotating shaft was controlled by a rotary encoder, a microcontroller, and a motor. This was considered to be too difficult and time-consuming. The hardware consisted of many different parts, so all the different modules needed to be simple and easy to implement.

The amount of software that needed to be implemented, especially the Godot code, was underestimated. To enable the hardware and Godot to communicate, a lot of different methods were tried and it took almost two weeks to fully implement. This time could have been a lot shorter if the methods used were documented better. So for future projects, it is important to write down all the installation steps of different programs. A lot of time was spent retracing the steps of other sub-teams or retracing the steps of other teammates. This could be improved in documentation by writing installation manuals for every software program of the project.

The last thing that was a major issue in this project was the testing. Due to the number of different modules that needed to be implemented, the testing setups were chaotic. The decision to use breadboards for testing was useful for testing every component individually, but for integrating the different modules, it was time-consuming to check every connection and make sure nothing was connected in the wrong way. This could be improved by testing components individually and, if they work as expected, soldering them together and using them as modules to test the final prototype. To improve testing in the future, a PCB has been designed to implement the full prototype, but due to a lack of time, this has not been tested yet.

6.2. Recommendations

For this project, recommendations for future groups would be adding different components to make the hardware module have more functions. This would enable the user to make more choices and take intelligent actions to solve problems on the power grid simulator. Right now, the user has three variables that can be controlled. This can be increased with other actions, such as building or breaking lines or tuning some lines on or off. This is pretty complex due to the dependency on the work of the other subgroups.

Another recommendation would be to improve the current components. For example, the time control could benefit from a speed-up or speed-down mode. The current feedback about the system from the

user is minimal; adding more warning components could help.

The final recommendation would be to test the system with users. Currently, not much testing has been done with inexperienced users; the control of mechanisms has been kept simple but user testing has not been achieved yet.

References

[1] Sue Allen and Joshua P Gutwill. “Creating a program to deepen family inquiry at interactive science exhibits”. In: Curator: The Museum Journal 52.3 (2009), pp. 289–306.

[2] Alethea Blackler, Vesna Popovic, and Douglas Mahar. “Intuitive Interaction Applied to Interface Design”. In: New Design Paradigms: Proceedings of International Design Congress (IDC) 2005. Ed. by M. C. Ho. International Design Congress, CD Rom, 2005, pp. 1–10.

[3] Marty Brown. Power Supply Cookbook. 2nd ed. Boston: Newnes, 2001.

[4] Neil Cameron. Electronics Projects with the ESP8266 and ESP 32. Apress, 2020, pp. 559–584. url: https://link.springer.com/chapter/10.1007/978-1-4842-6336-5_19#citeas.

[5] CHINA YOUNG SUN LED TECHNOLOGY CO., LTD. YSL-R596CR3G4B5C-C10 RED/GREEN/BLUE Triple Color LED. Online. Accessed: insert-access-date-here. 2023. url: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf.

[6] Matt Clary. “Interfacing to an LCD screen using an Arduino”. In: College of Engineering (2015).

[7] Peyton Reade Crosby. “Analysis of the Design and Manufacure of a Printed Circuit Board for a High Altitude Balloon Payload”. In: (2023).

[8] Godot Engine Documentation. Signals in Godot. [Online; accessed 14-June-2024]. 2024. url: https://docs.godotengine.org/en/stable/getting_started/step_by_step/signals.html.

[9] SparkFun Electronics. L298 H-Bridge. Online. Accessed: insert-access-date-here. n.d. url: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf.

[10] Jill Engel-Cox and Kerrin Jeromin. “Effective Communication of Energy Science and Technology”. In: Climate and Energy 40.8 (2024), pp. 1–8.

[11] Mohamed Fezari and Ali Al Dahoud. “Integrated development environment “IDE” for Arduino”. In: WSN applications 11 (2018), pp. 1–12.

[12] Ahmad Adamu Galadima. “Arduino as a learning tool”. In: 2014 11th International Conference on Electronics, Computer and Computation (ICECCO). IEEE. 2014, pp. 1–4.

[13] David L Jones. “PCB design tutorial”. In: June 29th (2004), pp. 3–25.

[14] Mihaela Sabina Jucan and Cornel Nicolae Jucan. “The power of science communication”. In: Procedia-Social and Behavioral Sciences 149 (2014), pp. 461–466.

[15] Joyce Ma et al. “Using a tangible versus a multi-touch graphical user interface to support data exploration at a museum exhibit”. In: Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction. 2015, pp. 33–40.

[16] I Scott MacKenzie. “Human-computer interaction: An empirical research perspective”. In: (2024), p. 41.

[17] Paul Marshall. “Do tangible interfaces enhance learning?” In: Proceedings of the 1st international conference on Tangible and embedded interaction. 2007, pp. 163–170.

[18] Mathertel. RotaryEncoder Arduino Library. Accessed: 2024-06-10. 2022. url: https://github.com/mathertel/RotaryEncoder.

[19] Niamh McNamara and Jurek Kirakowski. “Functionality, usability, and user experience: Three areas of concern”. In: interactions 13.6 (2006), pp. 26–28.

[20] MegunoLink. 3 Methods to Filter Noisy Arduino Measurements. Accessed: 14th of June 2024. 2023. url: https://www.megunolink.com/articles/coding/3-methods-filter-noisy-arduino-measurements/.

[21] Nur Qamarina Mohd Noor et al. “Arduino vs Raspberry Pi vs Micro Bit: Platforms for Fast IoT Systems Prototyping”. In: Open International Journal of Informatics 6.1 (2018), pp. 1–12.

[22] Francesca Santucci et al. “An immersive open source environment using godot”. In: Computational Science and Its Applications–ICCSA 2020: 20th International Conference, Cagliari, Italy, July 1–4, 2020, Proceedings, Part VII 20. Springer. 2020, pp. 784–798.

[23] NXP Semiconductors. PCF856l. Datasheet. https://www.farnell.com/datasheets/3717670.pdf. 2021.

[24] Joel Setterberg. GDSerCommPlugin. https://github.com/NangiDev/GDSerCommPlugin. Version 1.0. Accessed: 2024-06-09. 2019.

[25] Wouter Zomerdijk et al. “Open Data Based Model of the Dutch High-Voltage Power System”. In: Proceedings of the IEEE Innovative Smart Grid Technologies Conference Europe. Accessed: June 10, 2024. IEEE, 2022, pp. 1–5. doi: 10.1109/ISGT-Europe46778.2022.9960703.

A

Code

A.1. Code used for the communication with Godot

```
using Godot;
using System;
using System.IO.Ports;

public partial class Arduino : Node2D
{
        [Signal]
        public delegate void SMREventHandler(string message);
        [Signal]
        public delegate void WindEventHandler(float message2);
        [Signal]
        public delegate void SolarEventHandler(float message3);


        SerialPort serialPort;
        RichTextLabel text;

        private int rl = 0;
        private int ol = 0;
        private int redlines5 = 0;
        private int oldvalue = 0;
        private int oldvalue2 = 0;
        private float a = 0;
        private float b = 0;


        private void PrintNodePaths(Node node, string prefix = "")
        {
                GD.Print(prefix + node.Name);
                foreach (Node child in node.GetChildren())
                {
                        //PrintNodePaths(child, prefix + node.Name + "/");
                }
        }
        private void RLR(int rls)
        {
        rl = rls;
        GD.Print("Red Lines Received:", rls);
        }

        private void OLR(int ols)
        {
                ol = ols;
                GD.Print("Overloaded Lines Received:", ols);
        }
        private void Callredlines()
        {
```

```
48                  Node parentNode = GetParent();
49                  var child = GetParent().GetChild(3).GetPath();
50                  GD.Print(child);
51                  Node mapNode = GetNode<Node>("/root/Node2D/MapGenerator");
52                  Callable C2 = new Callable(this, "RLR");
53                  mapNode.Connect("RLS2", C2);
54                  Callable C3 = new Callable(this, nameof(OLR));
55                  mapNode.Connect("OLS2", C3);
56                  //powertest2.Connect("RLS", C2);
57                  //power.Connect("Ardwrite", C2);
58                  GD.Print("connection established");
59          }
60          //Called when the node enters the scene tree for the first time.
61          public override void _Ready()
62          {
63                  Node parentNode = GetParent();
64                  PrintNodePaths(GetTree().Root);
65                  text = GetNode<RichTextLabel>("RichTextLabel");
66                  Callredlines();
67                  serialPort = new SerialPort();
68                  serialPort.PortName = "COM4";
69                  serialPort.BaudRate = 9600;
70                  serialPort.Open();
71
72          }
73
74          public void Ardwrite(int overloadedlinesA, int redlinesA)
75          {
76                  int RGBlight = 0;
77
78                  if (overloadedlinesA >= 1)
79                  {
80                          RGBlight = 3;
81                          serialPort.Write("3");
82                  }
83                  else if(redlinesA >= 3 )
84                  {
85                          RGBlight = 2;
86                          serialPort.Write("2");
87                  }
88                  else
89                  {
90                          RGBlight = 1;
91                          serialPort.Write("1");
92                  }
93                  GD.Print("Light value send to arduino =", RGBlight);
94          }
95          // Called every frame. 'delta' is the elapsed time since the previous frame.
96          public override void _Process(double delta)
97          {
98                  //GD.Print("Red Lines Received:", rl);
99                  if(ol != oldvalue || rl != oldvalue2)
100                 {
101                         Ardwrite(ol, rl);
102                 }
103                 oldvalue = ol;
104                 oldvalue2 = rl;
105
106                 if(!serialPort.IsOpen) return;
107
108                 if(ol != oldvalue || rl != oldvalue2)
109                 {
110                         Ardwrite(ol, rl);
111                 }
112                 oldvalue = ol;
113                 oldvalue2 = rl;
114                 string serialMessage = serialPort.ReadExisting();
115
116                 if (!String.IsNullOrWhiteSpace(serialMessage))
117                 {
118                         if (serialMessage[0]== 'W')
```

```
119                         {
120                                 string modifiedMessage = serialMessage.Substring(1);
121                                 if (float.TryParse(modifiedMessage, out float windPowerValue)
                                        )
122                                 {
123                                         //power.Call("set_wind", 2.0f);
124                                         text.Text = modifiedMessage;
125                                         EmitSignal(nameof(Wind),windPowerValue);
126                                 }
127                         }
128                         else if (serialMessage[0] == 'S')
129                         {
130                                 string modifiedMessage2 = serialMessage.Substring(1);
131                                 if (float.TryParse(modifiedMessage2, out float
                                        solarPowerValue))
132                                 {
133
134
135                                         //power.Call("set_solar", 2.0f);
136                                         GD.Print(solarPowerValue);
137                                         text.Text = modifiedMessage2;
138                                         EmitSignal(nameof(Solar), solarPowerValue);
139                                 }
140                         }
141                         else
142                         {
143                                 text.Text = serialMessage;
144                                 EmitSignal(nameof(SMR), serialMessage); //
145                          }
146                 }
147         }
148
149
150 }
```

A.2. Timebase code

```
1  #include "timebase.h"
2 #include <godot_cpp/core/class_db.hpp>
3 #include <godot_cpp/variant/utility_functions.hpp>
4
5
6 using namespace godot;
7
8 void Timebase::_bind_methods() { //Create the sized property (as size is already a thing) to
      depict the size of the whole grid
9     ClassDB::bind_method(D_METHOD("set_timer_amount", "new_amount"), &Timebase::
          set_timer_amount);
10    ClassDB::bind_method(D_METHOD("get_timer_amount"), &Timebase::get_timer_amount);
11    ClassDB::bind_method(D_METHOD("_on_timebase_timeout"), &Timebase::_on_timebase_timeout);
12    ClassDB::bind_method(D_METHOD("_on_direction_signal"), &Timebase::_on_direction_signal);
13    ClassDB::add_property("Timebase",PropertyInfo(Variant::FLOAT,"amount"), "set_timer_amount
          ", "get_timer_amount");
14    ADD_SIGNAL(MethodInfo("time_index_signal", PropertyInfo(Variant::INT, "time_index")));
15 }
16
17 Timebase::Timebase(){
18     amount = 5;
19     flowdirection = 1;
20     timeindex = 0;
21 }
22
23 Timebase::~Timebase(){
24 }
25
26 void Timebase::_ready(){
27     timebase->set_wait_time(amount);
28     timebase->set_autostart(true);
29     timebase->set_one_shot(false);
30     timebase->set_timer_process_callback(TIMER_PROCESS_IDLE);
```

```
31      add_child(timebase);
32      timebase->connect("timeout", Callable(this, "_on_timebase_timeout"));
33
34      Node* parentnode = get_parent();
35      Node2D* arduinonode = parentnode->get_node<Node2D>("Arduinonode");
36      UtilityFunctions::print(arduinonode);
37      if (arduinonode){
38          arduinonode->connect("SMR", Callable(this, "_on_direction_signal"));
39      }
40      else{
41          UtilityFunctions::print("Arduino connection node not found");
42      }
43
44  }
45
46  void Timebase::_on_timebase_timeout(){
47      UtilityFunctions::print("index: ", timeindex, " amt: ", amount, " dir: ", flowdirection);
48      emit_signal("time_index_signal", timeindex);
49      if ((timeindex + flowdirection) < 0){
50          timeindex = 0;
51      }
52      else{
53          timeindex += flowdirection;
54      }
55  }
56
57  void Timebase::_on_direction_signal(String message){
58      int selector = message.to_int();
59      switch(selector){
60                  case -1:
61                      flowdirection = -1;
62                      break;
63
64                  case 1:
65                      flowdirection = 1;
66                      break;
67
68                  case 2:
69                      if (!timebase->is_paused()){
70                          timebase->set_paused(true);
71                      }
72                      break;
73
74                  case 3:
75                      if (timebase->is_paused()){
76                          timebase->set_paused(false);
77                      }
78                      break;
79
80                  default:
81                      break;
82                  }
83      return;
84  }
85
86  void Timebase::set_timer_amount(const float new_amount){
87      amount = new_amount;
88      timebase->set_wait_time(amount);
89      return;
90  }
91
92  int Timebase::get_timer_amount(){
93      return amount;
94  }
95
96  void Timebase::_input(InputEventKey *event){
97      if (timebase){
98          Ref<InputEventKey> key_btn = event;
99          if (key_btn->get_keycode() && key_btn->is_pressed()){
100             int keyselector = key_btn->get_keycode();
101             UtilityFunctions::print(keyselector);
```

```
            switch(keyselector){
                case 65:
                    if(amount > 0){
                        old_amount = amount;
                        amount = amount - 0.25;
                        if ((old_amount - 0.25) <= 0.2){
                            UtilityFunctions::print("Can't go slower!");
                        }
                        else{
                            timebase->set_wait_time(amount);
                        }
                    }
                    break;

                case 68:
                    amount = amount + 0.25;
                    timebase->set_wait_time(amount);
                    break;

                default:
                    break;
                }
            }
        }
        return;
    }
```

A.3. Code for the Arduino Mega

```
/*
Code For BAP project Power Grid Visualization
Written by Michiel De Rop and Johannes Ketelaars
*/



#include <LiquidCrystal.h>  // libraries
#include <RotaryEncoder.h>

//pin configuration
#define PIN_RA 2
#define PIN_RB 3
#define LED_R 11
#define LED_G 12
#define LED_B 13
#define windControl A2
#define motorControl 4
#define startPin 23
#define selectSolar 24
#define selectWind 26
#define voltageSolar A0
#define currentSolar A1
#define playbutton 35

// Constants for LCD
const int rs = 52, en = 50, d4 = 48, d5 = 46, d6 = 44, d7 = 42; //pin connection LCD
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Initialization of global variables
int PowerWind = 0;
int PowerSolar = 0;
int OldPowerSolar = 0;
int displayW;
int displayS;
int readingW;

int buttonsolar = LOW;
int stateW = HIGH;
int previousS = LOW;
int previousW = LOW;
```

```
42 int previousRW = HIGH;
43 int previousRS = HIGH;
44 int windRead = 0;
45 int oldwindRead = 0;
46
47 unsigned long timeS = 0;  // Time of last solar button toggle
48 unsigned long timeW = 0;  // Time of last wind button toggle
49 const unsigned long debounce = 200UL;  // Debounce time in milliseconds
50
51 bool papla = false;
52 float weighting = 0.5;  //weighting parameter exponential filtering
53 float oldSmoothVoltage = 0;
54 float oldSmoothCurrent = 0;
55 int lastPos = 0;
56
57 RotaryEncoder encoder(PIN_RA, PIN_RB);
58
59 void checkPosition()
60 {
61   encoder.tick(); // just call tick() to check the state.
62 }
63
64 void setup() {
65   lcd.begin(16, 2);  // Initialize the LCD
66   pinMode(startPin, INPUT);       // Set the start pin as input
67   Serial.begin(9600);              //startup serial communication
68   pinMode(motorControl, OUTPUT);     //set pint for wind motor as output
69
70   pinMode(LED_R, OUTPUT);          //set LED pins as output
71   pinMode(LED_G, OUTPUT);
72   pinMode(LED_B, OUTPUT);
73   analogWrite(LED_R, 0);           //start RGB LED in green
74   analogWrite(LED_G, 255);
75   analogWrite(LED_B, 0);
76
77   attachInterrupt(digitalPinToInterrupt(PIN_RA), checkPosition, CHANGE);    //use interrupts
       for the rotary encoder
78   attachInterrupt(digitalPinToInterrupt(PIN_RB), checkPosition, CHANGE);
79   attachInterrupt(digitalPinToInterrupt(playbutton), pauseplay, CHANGE);
80
81   oldwindRead = analogRead(windControl) + 10;
82   Serial.println("S0");                                  //initialize solar and wind value to
       computer with 0
83   Serial.println("W0");
84 }
85
86 void loop() {
87   buttonsolar = SolarButton();            //check if solar is activated
88   if(buttonsolar == HIGH){
89     PowerSolar = readSolar();             //if solar activated, read its power
90     if(OldPowerSolar != PowerSolar){
91       Serial.print("S");                  //send solar value to computer if value is updated
92       Serial.println(PowerSolar);
93     }
94     OldPowerSolar = PowerSolar;
95   }
96   lcd.clear();                            //clear LCD screen
97   SolarLCD(buttonsolar, PowerSolar);     //send solar information to LCD
98   encoder.tick();
99   pauseplay();                           //function for pause and play button on rotary
       encoder
100  // Check the position
101  int newPos = encoder.getPosition();
102
103  if (Serial.available() > 0){          //check if any communication has been received
104    RGBLed();
105  }
106
107  if (newPos > lastPos) {                  //send time controls to computer from rotary
       encoder
108    Serial.println("1");  // Turned right
```

```
109   } else if (newPos < lastPos) {
110     Serial.println("-1"); // Turned left
111   }
112   lastPos = newPos;
113
114   windRead = analogRead(windControl);       //read potentiometer for wind control
115   windInput(windRead);                      //give wind value to motor, LCD and computer
116   delay(100);
117 }
118
119 void RGBLed(){
120   int GridState = Serial.read();
121   if(GridState == '1'){
122       analogWrite(LED_R, 0);     // color green
123       analogWrite(LED_G, 255);
124       analogWrite(LED_B, 0);
125   }
126   else if(GridState == '2'){
127     analogWrite(LED_R, 255);     // color orange
128     analogWrite(LED_G, 165);
129     analogWrite(LED_B, 0);
130   }
131   else if(GridState =='3'){      //color red
132     analogWrite(LED_R, 255);
133     analogWrite(LED_G, 0);
134     analogWrite(LED_B, 0);
135   }
136 }
137
138 int readSolar() {
139   float power;
140   int voltageValue = analogRead(voltageSolar);                                    //read
        analog inputs
141   int currentValue = analogRead(currentSolar);
142   float smoothVoltage = weighting*voltageValue + (1-weighting)*oldSmoothVoltage;  //
        implementing exponential filter
143   float smoothCurrent = weighting*currentValue + (1-weighting)*oldSmoothCurrent;
144   oldSmoothVoltage = smoothVoltage;
145   oldSmoothCurrent = smoothCurrent;
146   float realVoltage = (voltageValue * (6/ 1023.0));                               // convert
        analog value to actual voltage, take voltage divider into account
147   float realCurrent = (oldSmoothCurrent / 1023.0) * 0.00869;                      // convert
        analog value to current
148   float measuredpower = realVoltage*realCurrent;                                  //
        calculate power
149   if(measuredpower >= 0.041712){                                                  // give max
         power value above 80percent of max power solar
150     power = 700;
151   }
152   else
153     power = (measuredpower / 0.041712) * 700;                                     //scale to
          0 to 700 range
154   return round(power/10)*10;                                                      //round
        power to nearest multiple of 10
155 }
156
157
158 int SolarButton() {
159   // Code for managing solar button press
160   int stateS;
161   int readingS = digitalRead(selectSolar);
162
163   // if the input just went from LOW and HIGH and we've waited long enough
164   // to ignore any noise on the circuit, toggle the output pin and remember
165   // the time
166   if (readingS == HIGH && previousS == LOW && millis() - timeS > debounce)    //function to
        change state with one button press
167   {
168     if (stateS == HIGH){
169       stateS = LOW;
170       Serial.println("S0");
```

```
171     }
172     else{
173       stateS = HIGH;
174     }
175     timeS = millis();
176   }
177   previousS = readingS;
178   return stateS;
179 }
180 void SolarLCD(int stateS, int displayS){
181   if(stateS == HIGH){                          //print values on LCD
182     lcd.setCursor(0,0);
183     lcd.print("Solar=");
184     lcd.print(displayS);
185     lcd.print("MW ON");
186   }
187   else{
188     lcd.setCursor(0,0);
189     lcd.print("Solar=0MW OFF");
190   }
191 }
192
193 void windInput(int windval){
194   int scaledWind = map(windval, 0, 1023, 0, 900);       //map value of wind power to
        simulator value
195   int motorInput = map(windval, 0, 1023, 0, 255);       //map value of wind power to an input
         value for the motor
196 if(abs(windval - oldwindRead)>5){                       //filtering for small noise
197   if(windval < 50){
198     analogWrite(motorControl, 0);                       //control motor and LCD
199     lcd.setCursor(0,1);
200     lcd.print("Wind=0MW OFF");
201     Serial.println("W0");
202   }
203   else if(windval >= 50){
204     analogWrite(motorControl, motorInput);
205     lcd.setCursor(0,1);
206     lcd.print("Wind=");
207     lcd.print(scaledWind);
208     lcd.print("MW ON");
209     Serial.print("W");
210     Serial.println(scaledWind);
211   }
212 }
213 oldwindRead = windval;
214 }
215
216 void pauseplay(){
217   if(digitalRead(playbutton) == HIGH){
218     if(papla == false){
219       papla = true;
220       Serial.println(3);
221     }
222     else{
223       papla = false;
224       Serial.println(2);
225     }
226     delay(300);
227   }
228 }
```