RADIOBLOB

Riccardo Alfieri, Riccardo Iaccarino, Manuele Montrasio,
Silvio Sgotto, Francesco Veronesi.

Group Name: *AKGigi* - Group ID: *10*

May 2022

1 Introduction

In the following document we present how we designed and implemented a complete Computer Music System (CMS) with interaction design principles by means of the electronics platform *Arduino* (used as interaction system unit and control device), the *SuperCollider* software (used as computer music unit and provider of OSC messages) and the *Processing* integrated development environment (used as graphical feedback unit).

The CMS we realized is a weather station that processes and transforms climate data into audio signals. In particular, the music that is generated and played in *SuperCollider* is based on the data that are processed by the station and is different for each weather condition. For example, the music that characterises a cloudy and windy day is sadder than that of a warm and sunny day. Similarly, we decided to design in *Processing* the graphical feedback unit, so that it behaves differently depending on the weather conditions.

To summarize, our idea was to create a system that generates ambient generative music, that is influenced by the weather conditions and in some way reflects them.

The report is organised in different sections. The first one covers how we decided to build and set-up the weather station, focusing in particular on the configuration of the *Arduino* module used to store and capture weather data. In the second section we present how the *SuperCollider* project has been structured, covering in detail the various *SynthDef*s and components it consists of and the communication protocols that were used for sharing information. In the third section the *Processing* project will be discussed.

2 Arduino

In this project one of the first goals to reach was to map physical phenomena into numerical parameters. *Arduino* is an open-source platform that allows these kind of manipulations of real quantities, maintaining a very simple interaction both with the IDE and the electronic components. In the following paragraphs different aspects related to this topic will be discussed such as the sensors used, the arrangement of the board (Figure 1) and the station's structure.

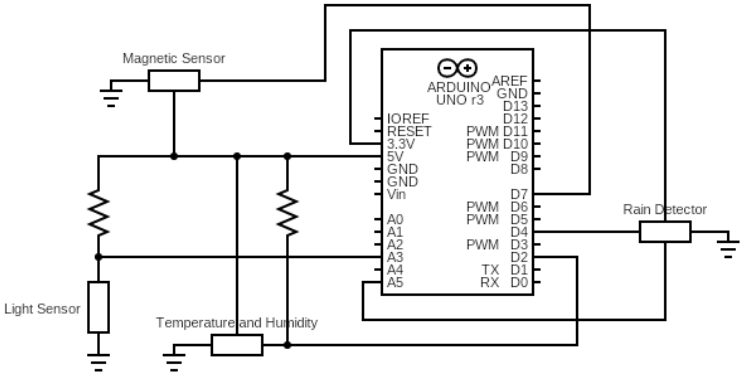

Figure 1: Circuit diagram of the connections between the *Arduino* board and the sensors. For the sake of simplicity, in the scheme we connected a different ground to each one of the components, although in practice they are all wired to the breadboard on a line connected to the GND pin of *Arduino*.

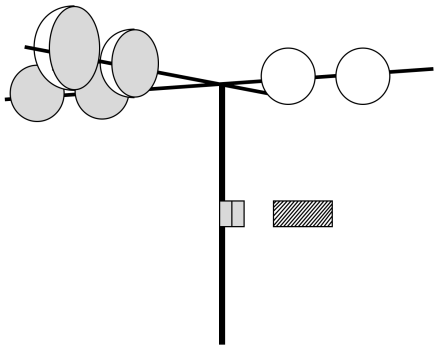

Figure 2: Graphical representation of the anemometer, built with halves of ping-pong balls threaded into two wooden sticks. Two magnets are then bonded onto the base of the structure.

2.1 Components

The idea of a weather station required a lot of different electronic components, in order to acquire each parameter almost in real time, monitoring them consistently. We will deeply discuss about the scheduling of the collection of data at the end of the section; let us now dive into the list of sensors that allowed the realization of this project.

- **Temperature and Humidity Sensor**: For this combined parameters we chose to rely on the "DHT11" by *AZDelivery*. This sensor acquires both temperature (in Celsius degrees) and humidity (in percentage) exploiting a proprietary protocol and a dedicated library that allows the access to these quantities through a digital input of the board.
- **Rain Detector**: Again by *AZDelivery*, we have used a plug and play rain detector, indeed the only wiring required are those of tension and of the two inputs, no further components are necessary. Moreover we want to specify that we decided to exclude the digital input, since the analog one is already a good indicator of the rain's behaviour, giving also an accurate feedback of the magnitude of this phenomenon.
- **Light Sensor**: Also known as photoresistor, this is probably the most intuitive component we used, specifically to acknowledge the presence of clouds in the sky. As for the previous sensor, an analog input has been used to read the values of light detected in real time.
- **Magnetic Sensor**: The last atmospheric variable we wanted to take into account is the wind, therefore we built from scratch a windmill, with a magnet glued to its base, as shown in Figure 2. The magnetic sensor has the role of counting how many rotations our rudimentary anemometer does every 2 seconds, finally communicating with the board through a digital input.

2.2 Board and station configuration

Despite the number of sensors and wires used, the configuration of the board is actually quite simple. Indeed every single component can be treated as an isolated circuit with respect to the others, just with some little precautions. As previously mentioned, two sensors (light and temperature-humidity) can be used in a plug and play fashion, hence we can just connect them to the 5V line, the GND (ground) and to their respective input pins. For the other two components we need to add a resistor in series to the concerned pin of the sensor, which will behave as a variable resistor. In this case the input pin has to be connected in between this two elements, in order to sense the voltage divider output that will vary with the physical quantity detected. One last comment can be made about the circuit: both for the need to add resistors and for a clearer management of the wires, we used a breadboard on which we attached the different circuits.

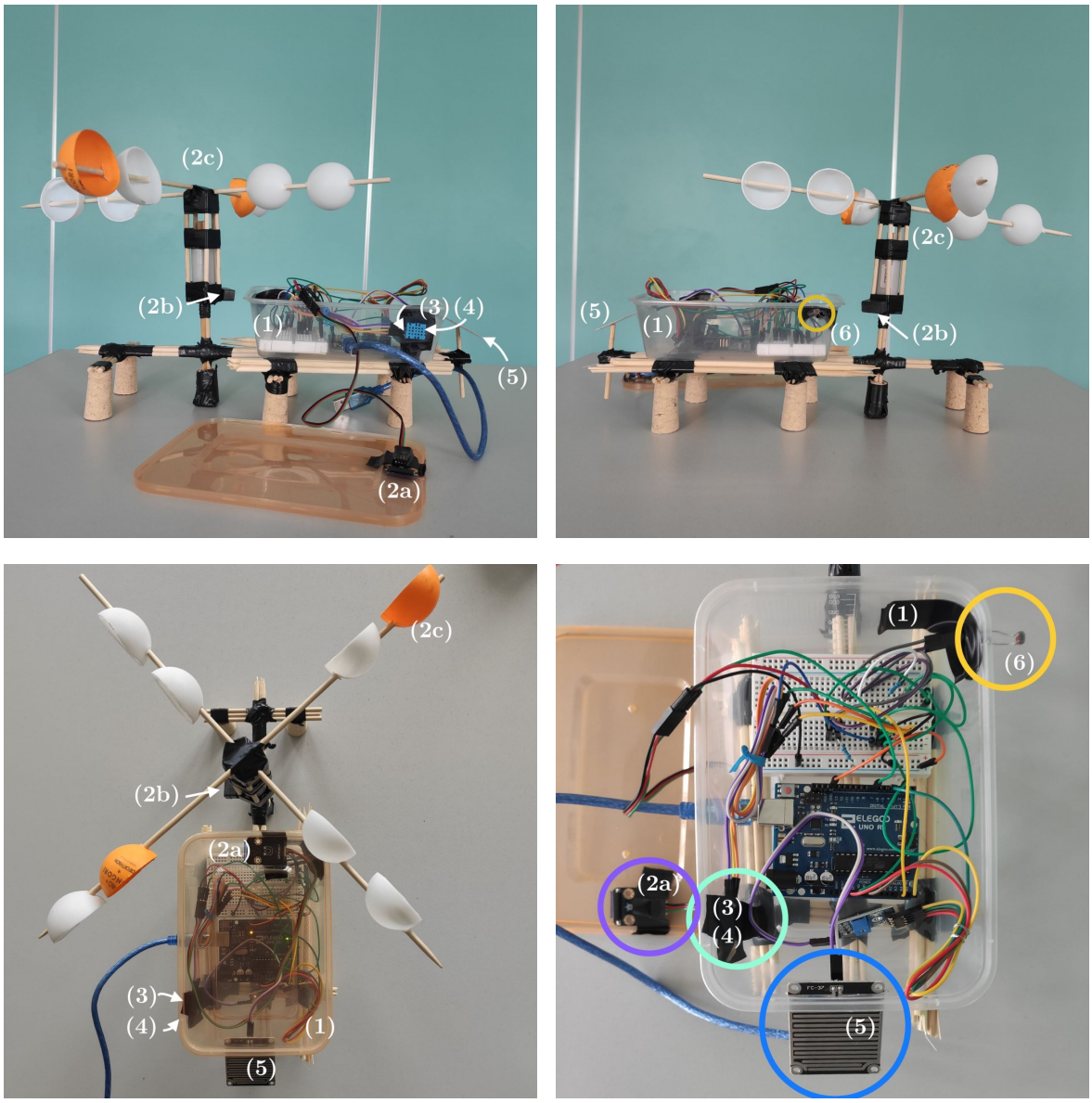

Figure 3: The weather station.

In Figure 3 we present how the weather station was constructed and assembled. The case (1), chosen in plastic to guarantee a waterproof housing, contains all the electronic equipment: the

magnetic sensor (2a), the temperature and humidity sensors (3, 4), the rain detector (5), the light sensor (6), the *Arduino* board and the rest of the circuit, including the USB cable.
In the structure the most elaborate set of instrument is the one that we use to compute the wind speed. As shown in Figure 2, the set consists of the windmill (2c), built with ping-pong balls, wooden sticks and two coupled ABEC 5 bearings to be specifically light and with low friction, the magnet attached to it (2b) and the corresponding sensor (2a).

3 SuperCollider

SuperCollider was used for the creation of a musical loop that is able to change according to the values that receives from the sensors of *Arduino*. The main problem was to create a loop that sounds pleasant even when some of its parameters change. In this section we are going to discuss how we achieved that, choosing properly the characteristic of the loop and the parameters we decided to change. The section will end with the explanation of the link between *SuperCollider* and *Arduino*.

3.1 Loop implementation

The main tool that SuperCollider offers to create loops is the `Routine`. We achieved to create a whole song by compiling simultaneously a series of `Routine`, one for each instrument. We did not use a single `Routine` for all of them because we wanted every instrument loop duration to be manipulated separately. This expedient, along with other ones, had the effect to create a loop that was different every time. The other useful *SuperCollider* tool we used is the `Pbind` class, that allows to easily control a series of events, such as a series of notes. Regarding the sounds we used some `SynthDef`s and samples we found online, that we later modified to achieve our goals. Now we are going to discuss one by one the three main section of the loop: the *melodic*, the *harmonic* and the *rhythmic* part.

3.1.1 Melodic section

The melodic section is composed by different instruments. Each one of them is played in a different `Routine` with different casual duration, implemented in such a way that the rhythmic structure rules are followed. The main problem was that completely random melodies are usually not musically enjoyable, so we decided to write them ourselves. So every instrument has its own short melodies from which the `Routine` randomly choose one every time. The scale is managed by the class `Scale`, that is used also for the harmonic section.

3.1.2 Harmonic section

The harmonic section is divided into two `Routine`, one for the bass and one for the chords. The first one simply plays a note every one or two measures, randomly chosen between the first, fourth and fifth grade of the scale. The chords are built by a random group of notes of the scale played together and held for one or two measures.

3.1.3 Rhythmic section

The rhythmic section is composed by a kick, a "clap" snare, a softer snare for ghost notes and a hi-hat. The four sounds are played in the same `Routine`, that always last one measure. Just like for the melodic section, we created a series of rhythmic parts from which the `Routine` randomly

chooses every iteration. The choice in this case is not completely random, but it is developed in such a way that when the current scale is "happier" the chosen part is denser and vice versa for a "sadder" scale. This mechanism will be clearer in the next section, where we will discuss the way in which the weather parameters get from *Arduino* changes the music ones.

3.2 Changing parameters

The main problem was to find a series of music parameters that are able to change the music in a perceivable way, without affecting his enjoyability. However, once we found the right ones, we linked each one of them to a weather parameter. The way in which the meteorological condition affects the music is the following: for each weather parameter we chose an "optimal" value (for example the absence of rain, so that the value we get is 0) and then we calculate the difference between that value and the detected one. Larger the difference, the most affected is the music. Hereafter we are going to explain one by one how this happens.

- **Wind → BPM**: the wind velocity, got by the anemometer we built, increases the BPM proportionally, from 60 (with no wind) to 100 (with maximum wind). The change of the BPM is implemented by modifying the `TempoClock` of *SuperCollider*, that controls all the time dependent events of the code.
- **Temperature → LPF/HPF**: the user has the possibility, to choose the temperature he prefers. If the measured temperature is higher than that, a low pass filter is applied to all the sounds, if it is lower, a high pass filter is applied instead. The cutoff frequency of the filter is proportional to the distance between the chosen temperature and the measured one.
- **Humidity → Reverb**: if the detected value is higher than 45%, the dry parameter, associated to the reverb applied to the melodic and rhythmic sounds, raises proportionally to it.
- **Brightness → Instruments**: as the environmental brightness decreases, the melodic instruments are turned off one by one. For minimum brightness the volume of the rhythmic instruments is halved, so that only them and the harmonic section remains. During the night the brightness is no more considered and the rhythmic section is turned off too.
- **Mood → Scale**: this is the most interesting part regarding the interaction between *SuperCollider* and *Arduino*. `Mood` is a variable that is calculated as a weighted average of all the normalized deviation of output of the *Arduino* sensors from the best weather ones. It is a measure of the quality of the weather and goes from 0 ("optimal" weather) to 1 ("worst" weather). The weights have been decided considering how much each value affects the quality of the weather, for instance the greater weight has been assigned to the rain sensor output. The value of `mood` corresponds to a change of the scale through the class `Scale`. In particular we decided to use the modal scales, ordered from lydian, which is brightest one, for an almost "optimal" weather to locrian, the darkest one, for the opposite case. The interesting thing is that the contour of the melodies remains the same, but the notes vary with the changing modes, giving to the loop a different mood. Also the harmonic section is affected by the change of the scale, just like the melodic one.

3.3 Connection with Arduino

All of the above would not have been possible without the implementation of the communication between the two software. For this purpose we exploited the possibility of *Supercollider* and *Arduino* to write and read on serial ports, but of course this could not work without further

changes, due to the different protocols used by the two programming languages. Since *Arduino* treats its serial output as a set of Ascii characters, the main idea behind the messages that will be sent is to separate different parameters with different letters; finally *Arduino* will use the Serial class to write on a selected port. On the other side *Supercollider* can read from the serial port just by assigning it to a variable, which will collect everything that passes by. Briefly, we can summarize the other passages of this process saying that from this sequence of characters we need to identify the different values (made possible by the addition of letters) to separate them into our desired parameters as a final step.

4 Processing

In this section we will talk about the development of our graphical feedback unit in the *Processing* framework. In particular, we will discuss about the main idea behind the visualization of the many variables included in our system, its implementation and finally the communication between *Processing* and *SuperCollider*. As with any weather website, we decided to display every kind of event detected by our *Arduino* station, but of course we did not limit ourselves to a simple "widget" look with just basic icons.

4.1 Project structure

In order to get a clearer code structure which implements an objects oriented language, we exploited an organization in classes. The latter have been designed to represent environmental elements, SVG shapes and interactive objects, hence handling them all separately. Since our target was to implement a dynamical graphical feedback unit dependent on weather parameters, we chose to work mainly in the `"draw()"` section recalling the methods designed in our classes.

In the project we also used many external libraries, mandatory to implement the following features. The communication between *Processing* and *SuperCollider* (that will be further discussed in depth) has been made possible thanks to `netP5` and `oscP5`, whereas the interactive objects (a volume knob and a slider) rely on the `controlP5` library.

4.2 Graphical visualization of weather parameters

The interface presented to the user is filled with many different elements capable of returning a real time feedback. We exploited various methods and kinds of files, depending on which technique better suited the element to display. We will now showcase every piece of our GUI, explaining how they have been merged in the project.

- **Day and night**: Two radial gradients that act as background themes (.png), switching at a given time with a smooth transition.
- **Stars**: Twinkling celestial objects that help recreating the night-time atmosphere. Randomly positioned during the setup phase, these little circular ellipses can shine thanks to the implementation of simple `line` elements that appear occasionally.
- **Wind**: Implemented as the oscillating movement of the trees (.svg) that we achieved thanks to what is probably the graphical escamotage we are most proud of. The `PShape` class allowed us to pick every vertex from each figure, which we then made oscillate with a greater magnitude for those ones towards the tip.

- **Clouds**: Moving elements in the sky (.png) that reflect the amount of light sensed. The main feature of these components is their coherence, since they are constantly looping around the window with a Pac-man like behaviour, but they will not come back if the atmospheric situation varies. The only limitation is the impossibility to catch any quantity that could give us a feedback on the presence of clouds by night.
- **Rain**: Drops falling accordingly to the weather condition. For this element we relied simply on the tools offered by *Processing*, creating oblong ellipses the move their position over time.
- **Informations**: A volume knob and a temperature slider. The first one is pretty much straight forward, while the second one refers to the possibility of the user to set their ideal temperature, as already mentioned in Chapter 3. This is due to the "perfect weather" variable `mood` for which we wanted to include the user's opinion (also to deal with the changing of the seasons). To collect this informations we used the `controlP5` class.

Figure 4: The Processing graphical feedback unit for four different climate conditions. Top-left: high daylight levels and no clouds in the sky. Top-right: the clouds and the rain make the the day sky darken. Bottom-left: A calm night. Bottom-right: a night with rain.

4.3 Communication with SuperCollider

Unlike the communication with *Arduino*, in this case both *SuperCollider* and *Processing* need to exchange messages between them. Despite this, the implementation has been way more trivial, since the `oscP5` and `netP5` libraries ensured an immediate transfer of messages through the loop-back port of our host. From *SuperCollider* we send the weather parameters that will be then

collected and converted into graphical language in *Processing*. In the opposite direction the informations regarding the master volume and the preferred temperature will travel to finally modify the respective parameters in the music played.

5 Conclusion

The aim of the project was to implement a CMS in which *Arduino*, *Supercollider* and *Processing* are exploited. In general we could say that we achieved this goal, but during the process there has been some issues that are worth discussing. We will illustrate the positive and negative aspects of our project in this section.
Starting with the pros, the connection between all the different types of software and the hardware has been completed successfully. As a result the weather changes are converted into musical variations (almost) in real time.
Moreover we are pleased to have built a compact structure for our project. In particular the design and building of the anemometer has gone through some technical difficulties, specifically the friction between the windmill and the main structure.
Regarding the musical part, we are satisfied with the fact that the mood of the music is connected quite well to the weather, playing a generally considered "sadder" scale when the weather is "bad" (rainy, cold, cloudy etc.) and vice versa. Also the GUI is responsive as we intended, representing graphically the weather both with the changes in his appearance and providing real time values for atmospheric parameters.
Moving to the negative aspects of our project, the most evident problem is the fragility of the structure. It is mainly made by wood sticks and it would be easily damaged by the elements.
In addition, the anemometer is rudimentary and the values we get from it aren't accurate enough to calculate the speed of the wind (we just have a qualitative information about the intensity of the wind).
Regarding the software part, music could be perceived as repetitive as it is composed of a finite number of musical and rhythmic patterns that follow one another randomly.
The limits of this project are naturally converted to the future developments. A metal and waterproof case would be fundamental to use this device outside, and the "DIY" anemometer should be replaced with a pre built one to have better accuracy and achieve the possibility to calculate the speed of wind in real time. Finally, to eliminate the repetitiveness of the music, the best solution would be to implement a program which continuously generates new melodies and rhythmic patterns. This could be achieved, for example, by means of machine-learning methods.