Robotics & Design
Multidisciplinary project

A.Y. 2015/2016

POLITECNICO
MILANO 1863

daft drunk

final report

Cipro L. Enerli D. Fanton C.
Principe F. Vago A.

Summary

1 INTRODUCTION 3
2 DESIGN 3
3 ELECTRONICS 5
3.1 Equalizers 5
3.2 LED MATRIXES 7
3.3 RGB RING 9
3.4 THE MAINBOARD 10
3.5 Power Source 13
4 CODE 16
4.1 Leds 16
4.2 Motors 17
5 PROBLEMS 20
5.1 Component Problems 20
5.2 Software Problems 20
6 CONCLUSIONS 22

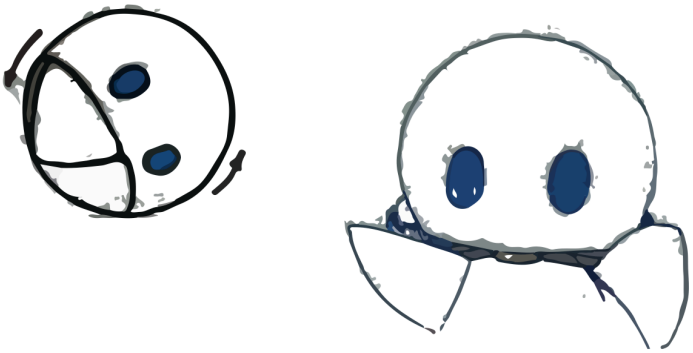
1 INTRODUCTION

Daft Drunk is a dancing robot that can analyze the beat of a song and dance along with real-time processing of frequencies, and on top of this, it also delivers some visual effects.
It has been developed for dancing along electronic music, but it can dance to every music genre as well, always maintaining its sharp and robotic dancing moves.

The main idea behind its development is related to Daft Punk music and imaginary. The French duo revolutionized the electronic music world in the 00's with their productions and outfits, making them look like someone who arrived from a future era.
We wanted to transpose this idea into our robot, making it similar to a "futuristic toy" that dances along a futuristic type of rock, much more electronic than the contemporary one. (in fact the presentation song we chose for our robot is "Robot Rock").

In the following sections we provide a deep explanation about the choices on design, electronics and informatics that we made in order to deliver a final product coherent to what we wanted to achieve.

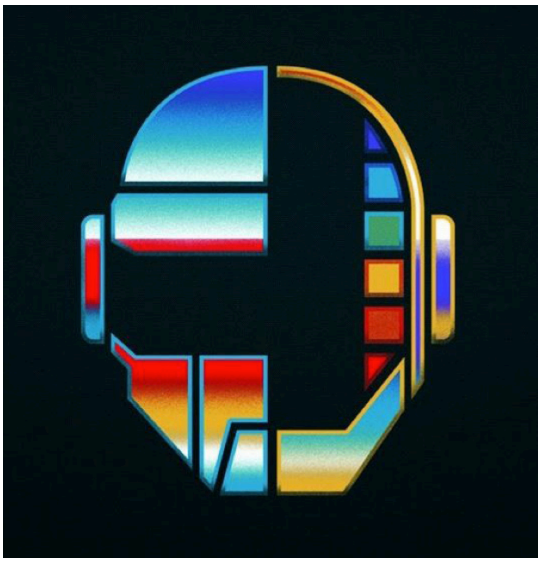
2 DESIGN



Picture 1 – First sketch

Our research was focused on two aspects. On one side, we had settled on the music style which was going to represent our robot, and on the other side, we knew which shape we wanted. The desired musical style is the one which surrounds the music of the group Daft Punk, especially in the 90s. It is a style characterized by LEDs, neon and brilliant colors (golden, stainless, etc.). The shape desired from the beginning is the one of an octopus as in picture 1.

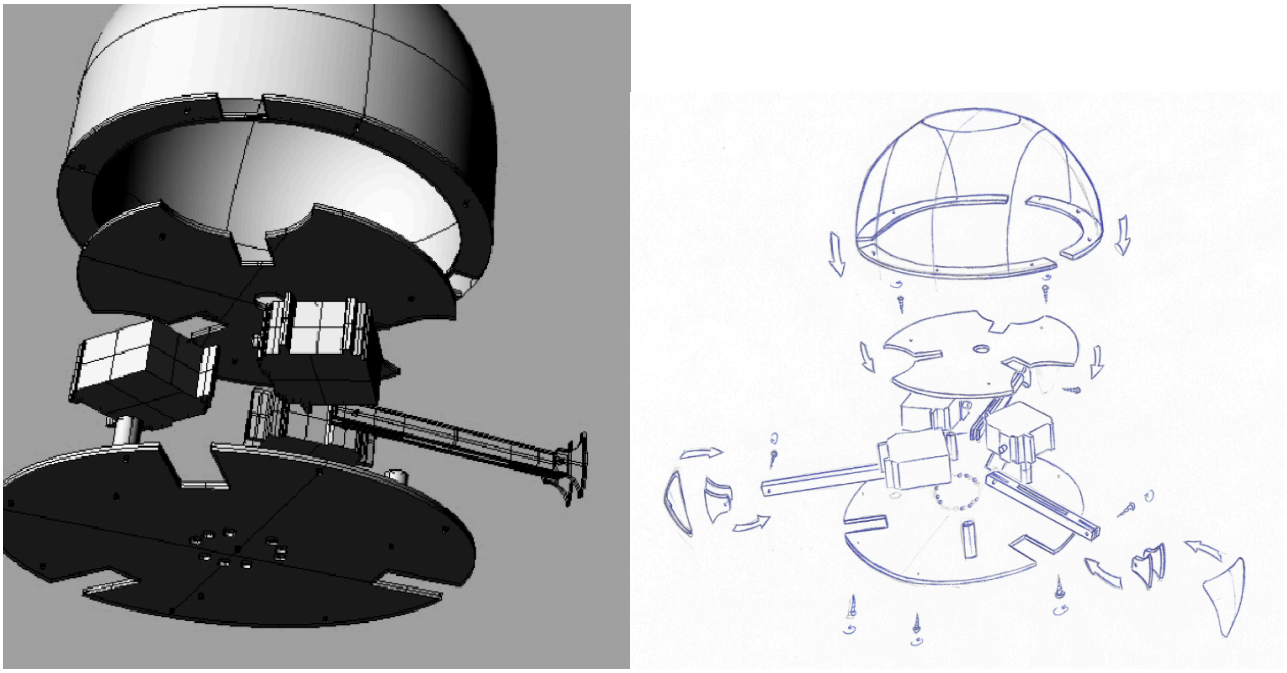
A round head as well as several legs to make it move on the sides, each having movements different from other. After some meetings and prototypes, the shape defined itself more exactly like a body looking like the helmet of one of the DJ of the group, bronzed on sides and black in the center, illuminated with LED in the shrill colors.



Picture 2 - Concept

We decided to reduce the initial 4 legs to 3, in order to simplify the mechanism and especially to allow the robot to move in a more fluid way with respect to 4 points. After several tries, the shape and the organization of the components had been defined.

From then on we worked on the prototype who became the final 3D model (picture 3). Bronzed on sides, metallic black in the center with a window allowing to see its internal leds, it also had LED at ground level. The left internal support is in PMMA cut with laser whereas the external shell and the feet of the legs were thermoformed and styrene painted.



Picture 3 - 3D Model and assembling scheme

3 ELECTRONICS

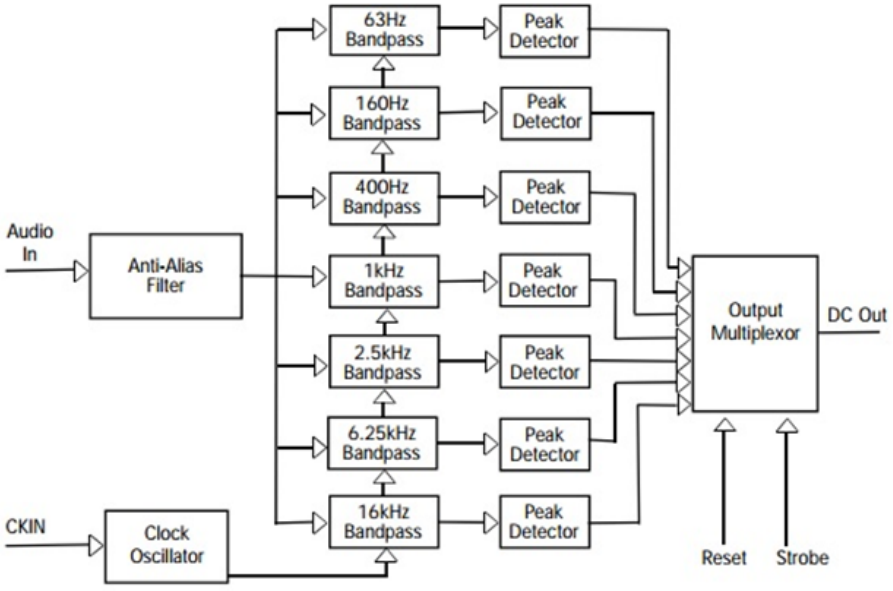
The brain of Daft Drunk is composed by two Arduino Nano.
We have chosen to use Arduino because of its simple programmability and the great availability of ready made libraries, which enable to create complex programs quickly with just few lines of code. The Nano version of Arduino, in particular, let us drastically reduce the space occupied by electronics without reducing performance or pins number.
We used two Arduinos mainly because of the necessity of a high number of pins and to parallelize the management of motors and lights, in order to make the robot more reactive.
In fact we have an Arduino designated to the management of two led matrixes and a circuit to pilot 9 RGB 5mm leds, and another one for the management of 3 servo motors TowerPro MG995, capable of supplying a torque of 13Kg/cm, enough for the robot to stand on its legs.

Both the Arduinos can control external peripherals following music's rhythm thanks to a preamplified microphone and an intermediate elaboration circuit based on the IC MSGEQ7.

All the circuitry that animates Daft Drunk has been printed in a way to reduce occupied space and to increase order inside the robot.
In particular, the whole circuit has been divided into the following parts:

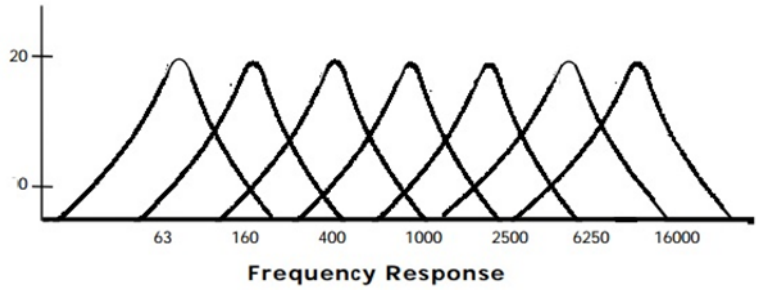
3.1 Equalizers

They are the circuits that let the two Arduinos to recognize sound. We have used two, one for each Arduino, and it's based on the integrated circuit MSGEQ7, whose internal schema is shown below.



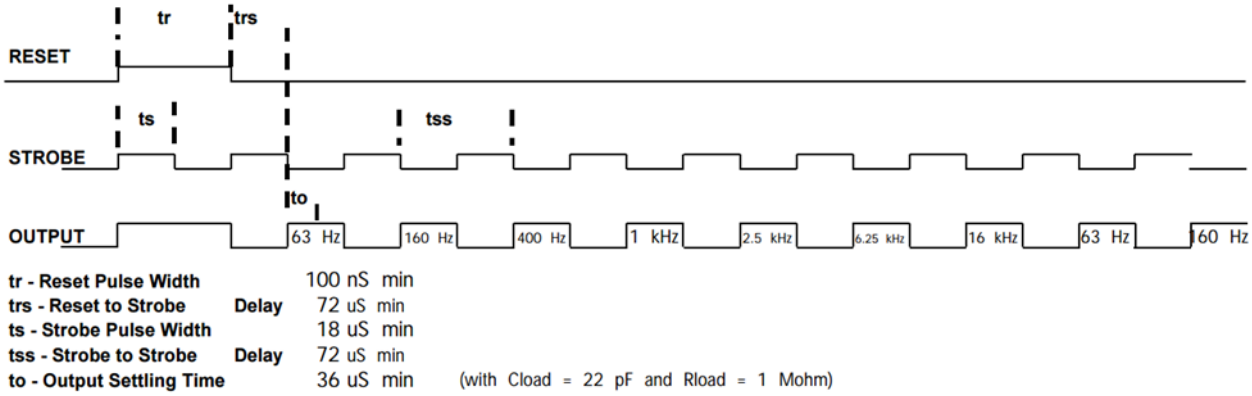
Picture 4. MSGEQ7 internal schema

After an anti-alias filtering, it supplies the input signal to a battery of band-pass filters that cover the full audio band(20Hz-20kHz) and thanks to a multiplexer it allows us to analyze the amplitude of the signals in the 7 output frequency bands, using just two output pins and one analog pin of the Arduino driving it.



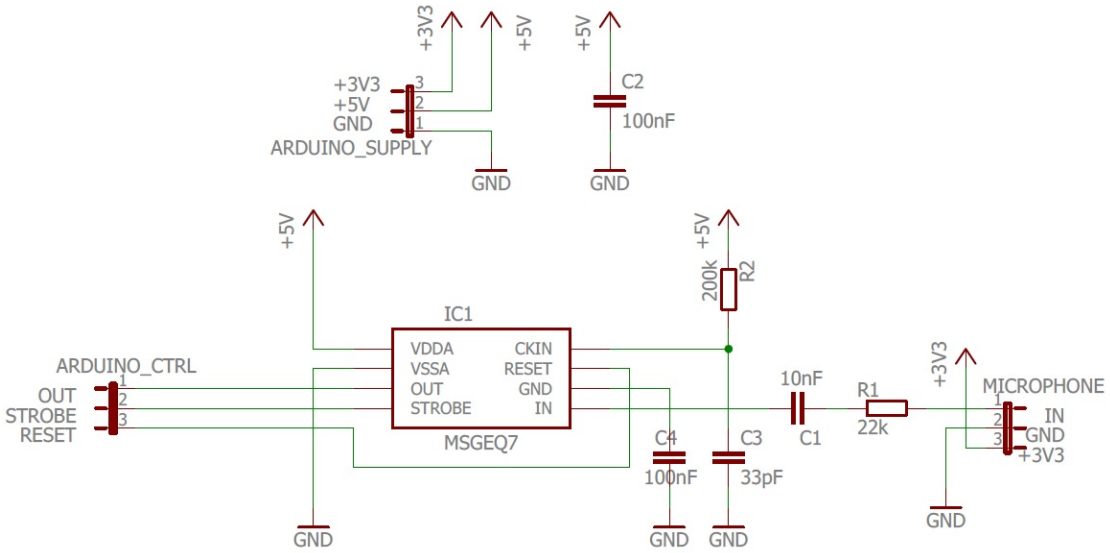
Picture 5. Frequency response of MSGEQ7

The DC peak output for measurement is selected using the reset and strobe pins. Reset high resets the multiplexer.
Reset low enables the strobe pin. After the first strobe leading edge, 63Hz output is on OUT. Each additional strobe leading edge advances the multiplexer one channel (63Hz, 160Hz, 400Hz, 1kHz, 2.5kHz, 6.25kHz, 16kHz etc.) and this will repeat indefinitely. The multiplexer read rate is also the output decay time control. The strobe timing is shown below:

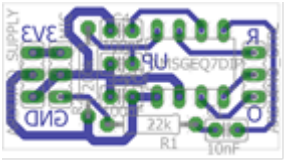


Picture 6. MSGEQ7 strobe timing

We used two different circuits for each Arduino because using a single one would have required a perfect synchronization and exchange of information between the two microcontrollers. The circuit scheme of the integrated circuit has been suggested by the producer and is the following:



The master used for the printing of the circuit is this one:



The dimensions and disposal of the connectors were chosen in order to connect efficiently these modules on the mainboard.

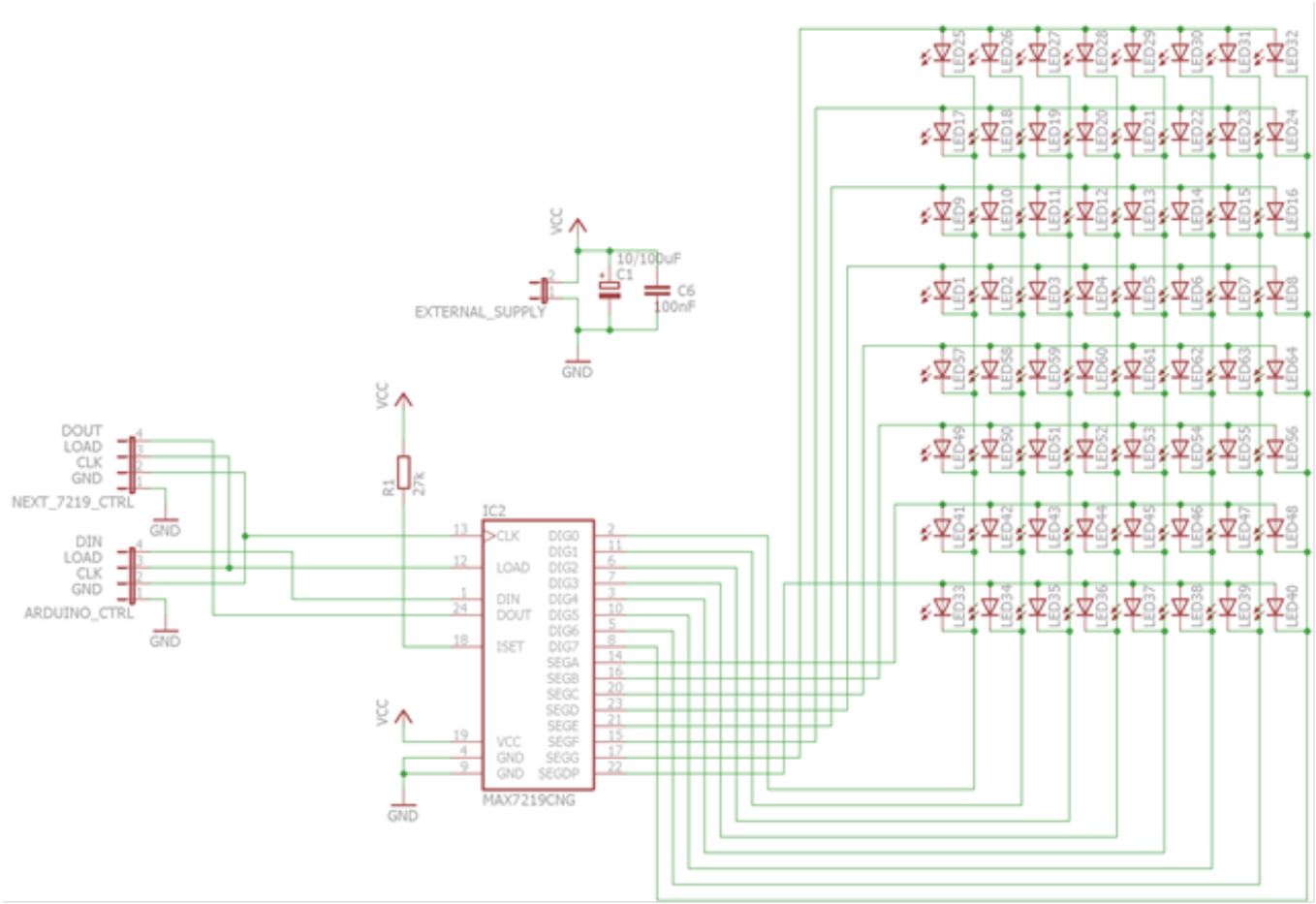
3.2 LED MATRIXES

In order to create light effects typical of the style by which Daft Drunk is inspired and in order to make it able to show emotions, we realized two matrixes composed by 8x8 3mm high luminosity leds. We chose to use two matrixes instead of a single one of 16x8 to adapt the flat surface of the matrixes to the spheric surface of the helmet which they were attached to.

Thanks to the integrated MAX7219 it was possible to manage the matrixes in a way that each led could be addressed by its own in a serial way, so that we could save some outputs on the Arduinos.

Another important aspect in the usage of this integrated is that the absorbed power is always limited because the leds won't ever be turned on simultaneously all together (at maximum can be on8 leds per matrix). The impression that the leds are all turned on is given by the high refresh rate.

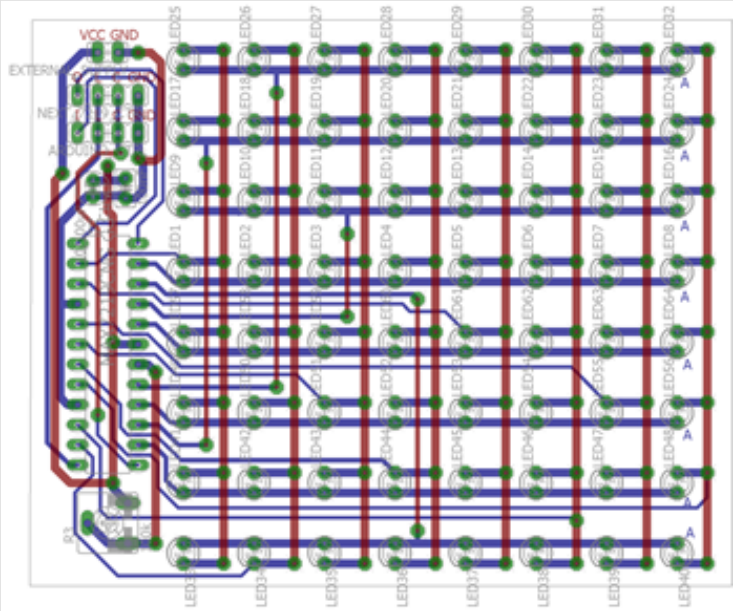
The circuit scheme is the following:



The Din and Dout pin of the MAX7219 allow us to connect the two matrixes in waterfall, simply connecting the "NEXT_7219_CTRL" connector of the first matrix to the "ARDUINO_CTRL" of the second one.

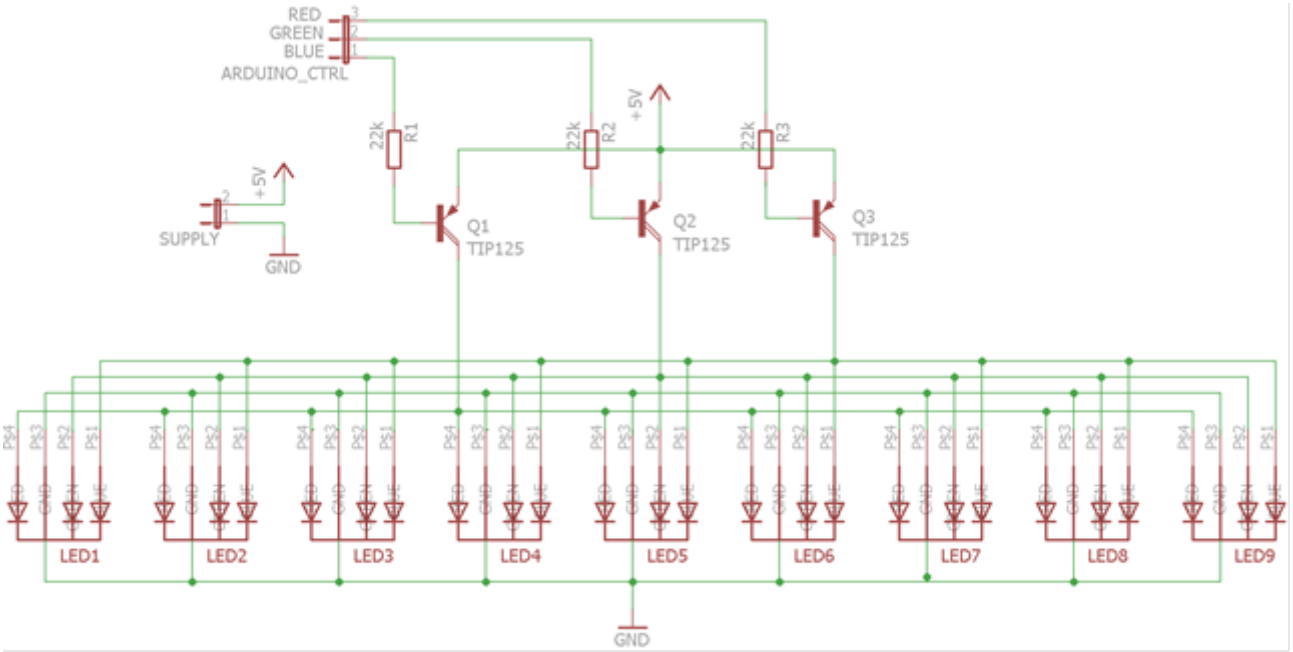
The two matrixes will be managed simultaneously using only 3 digital pins of the Arduino that are gonna be connected to DIN, LOAD and CLK of the ARDUINO_CTRL connector of the first matrix
Resistence R1 allow us to set the maximum current and voltage that can pass through the leds. The leds we used support a maximum voltage of 3,3V and a maximum current of 20mA. These measures allowed us to choose a proper resistence of 27k thanks to the technical table in the datasheet of the producer.
The capacitors connected in parallel eliminate the disturb on alimentation.

The printed circuit was realized double face because of the high number of wires. The master is the following:



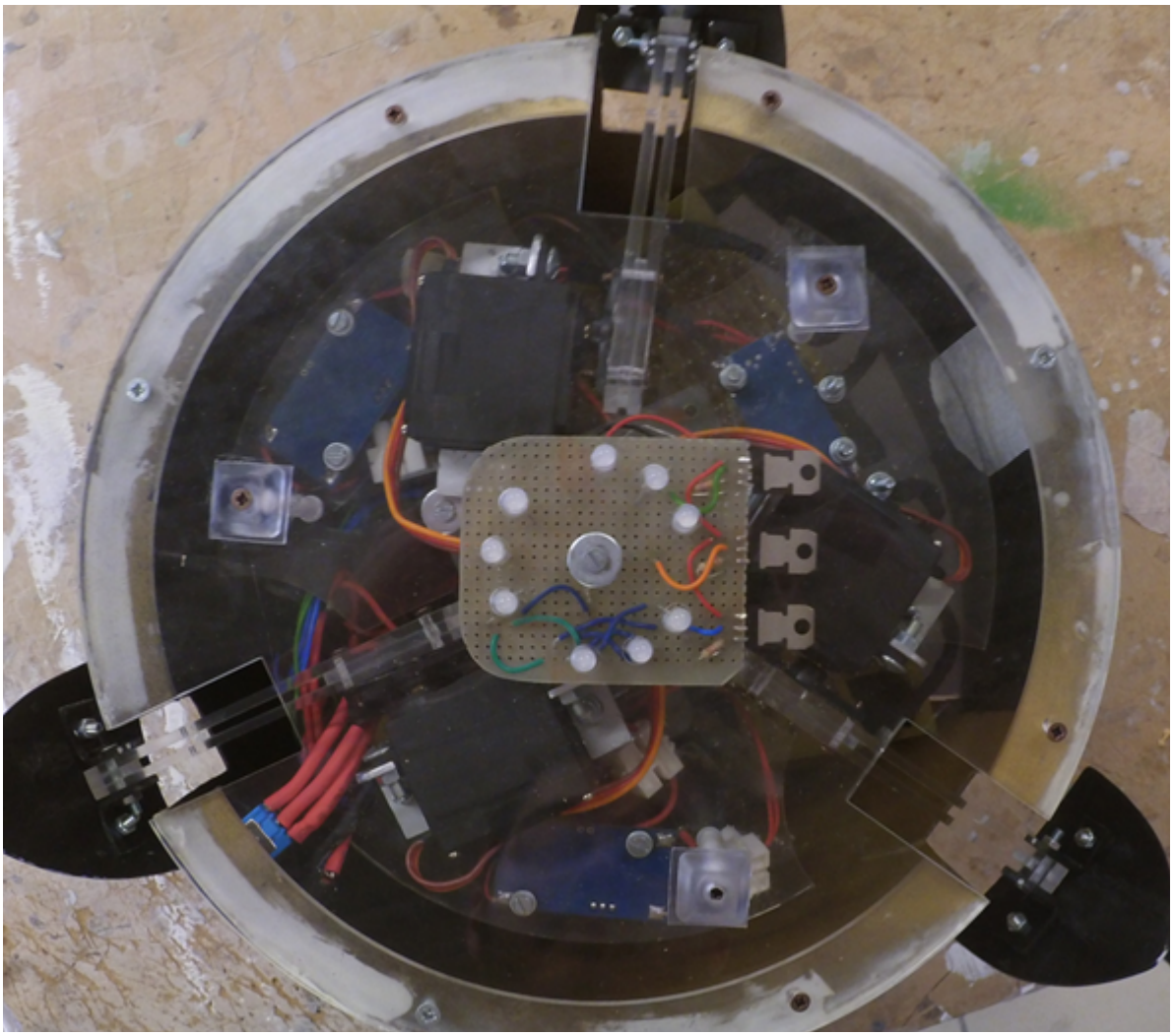
3.3 RGB RING

Another visual effect that we wanted to create consists in a halo of changing color on the bottom surface. To do so we have used 9 RGB 5mm leds with common cathode arranged in a ring of 5cm of diameter, controlled by Arduino through TIP125 transistors. The schema is the following:



Given that Arduino is capable of supplying a maximum current of 40mA at each digital output and 9 leds connected in parallel can arrive at a demand of 180mA (as each led can ask up to 20mA), we opted to use transistors. Transistors, in fact, are widely used as switches controlled in tension, so that the Leds can drain current directly from the power supply and Arduino's outputs are not overwhelmed by a too high demand of current.

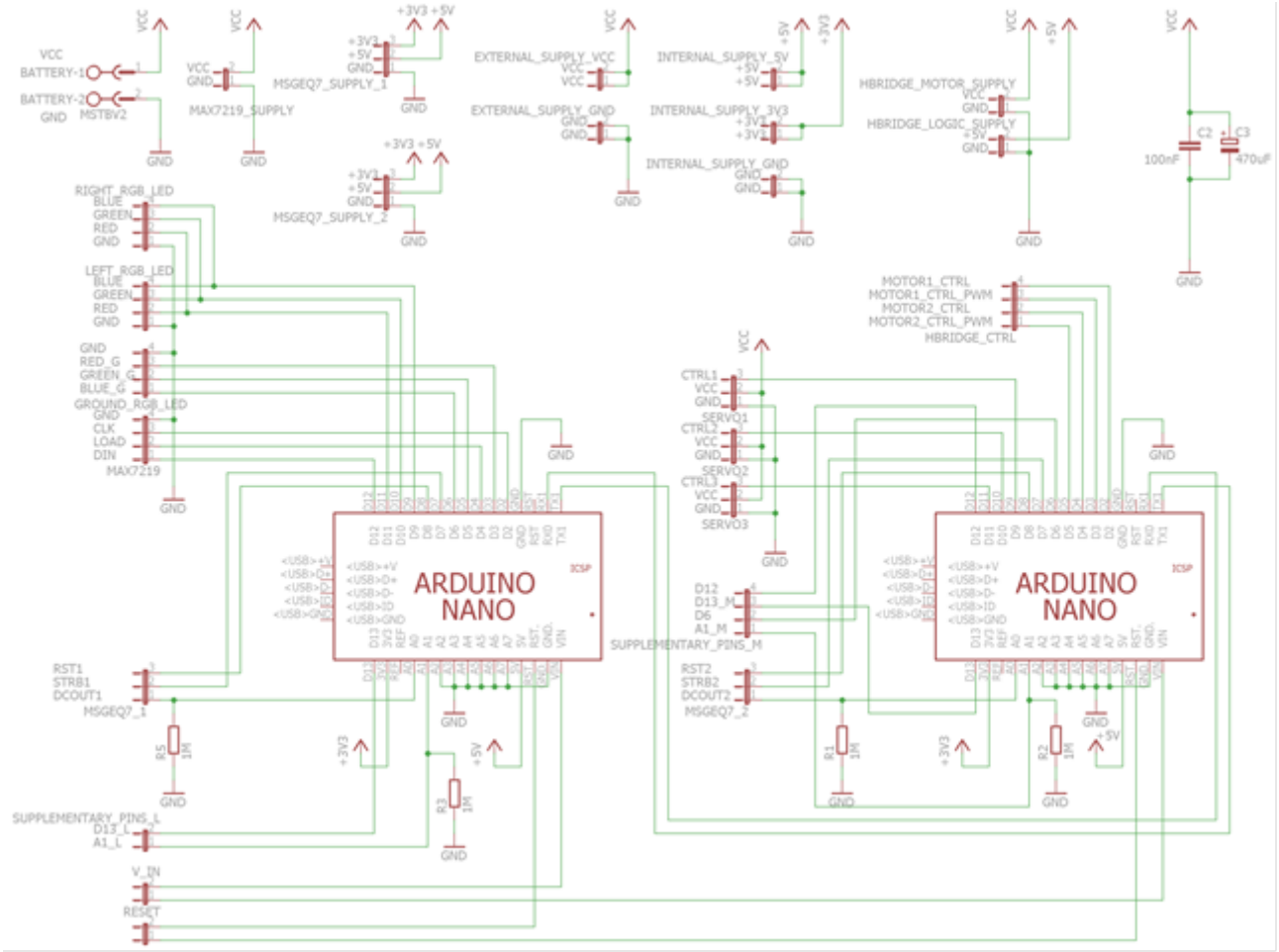
The choice of using the TIP125, that is a PNP transistor by Darlington, is because of practical/economic matters. The optimal solution should have been using channel P MOSFET with a minimum Vsg lower than 5V, because this would have granted us to pilot leds without draining any current from Arduino pins, to consume less power and, above all, to dissipate less heat. Unfortunately, because of the low availability and the higher cost of this component, we decided to use the TIP125.
The 22K resistor is essential for the correct polarization of the transistor and to guarantee a total current in the leds below 180mA.
In this case, given the simplicity of connections, the circuit has been realized on a stripboard.



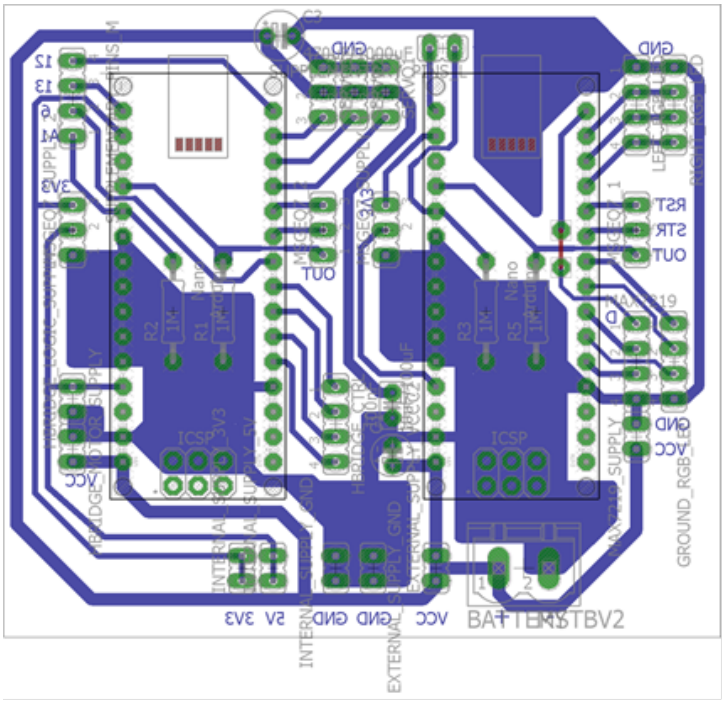
3.4 THE MAINBOARD

This is the circuit on which the two Arduino Nano are slotted and to which all other circuits described earlier converge.
It allows to make easy, steady and tidy connections between Arduinos and external peripherals that it has to drive through external connectors.

Here is the schema:



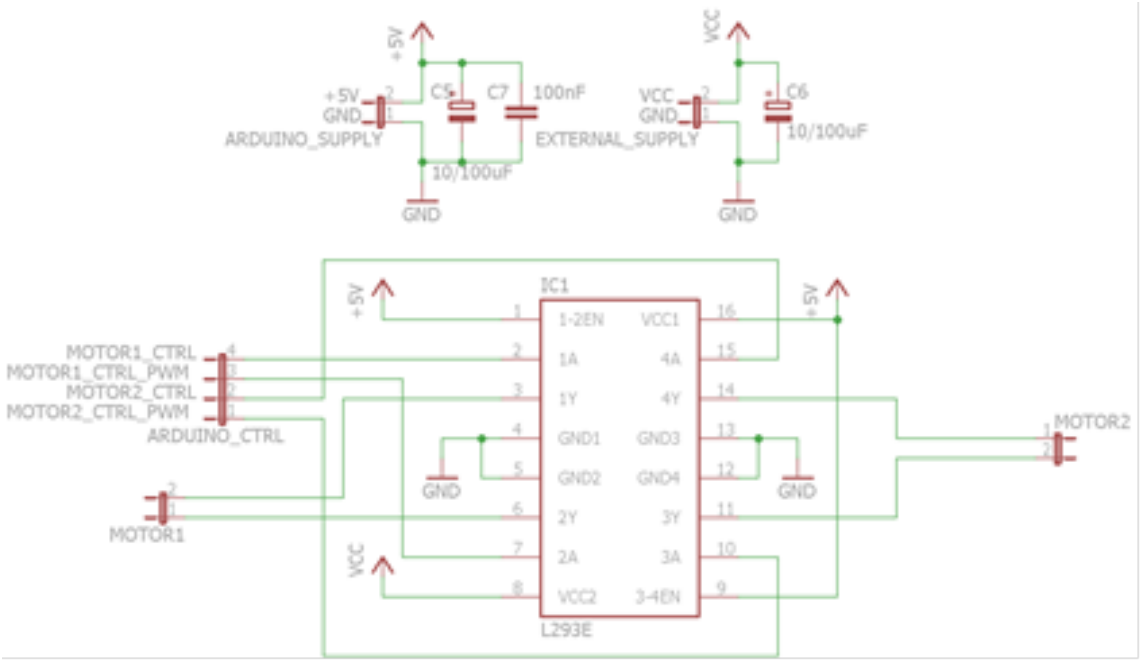
While the used master is reported below:



The left Arduino is the one designated to lights control. It drives the equalizer and reads its output through the pins linked to the connector MSGEQ7_1; through pins connected to MAX7219 it controls the Led matrixes; finally, through pins connected to GROUND_RGB_LED it manages the RGB ring placed under the robot.
The SUPPLEMENTARY_PINS_L connector has been inserted for eventual future upgrades, just as LEFT_RGB_LED and RIGHT_RGB_LED connectors, linked to the same pins and useful for adding new RGB leds.

The right Arduino, instead, is designated to motors' management. It also has a MSGEQ7_2 connector for driving and reading from the equalizer and then it has connectors SERVO1, SERVO2, SERVO3, through which it controls the 3 servos responsible of robot's movements.
The SUPPLEMENTARY_PINS_M connector, as for the other Arduino, has been inserted for eventual future upgrades, while the HBRIDGE_CTRL has been put to leave the possibility of adding two DC motors driven by an H bridge.

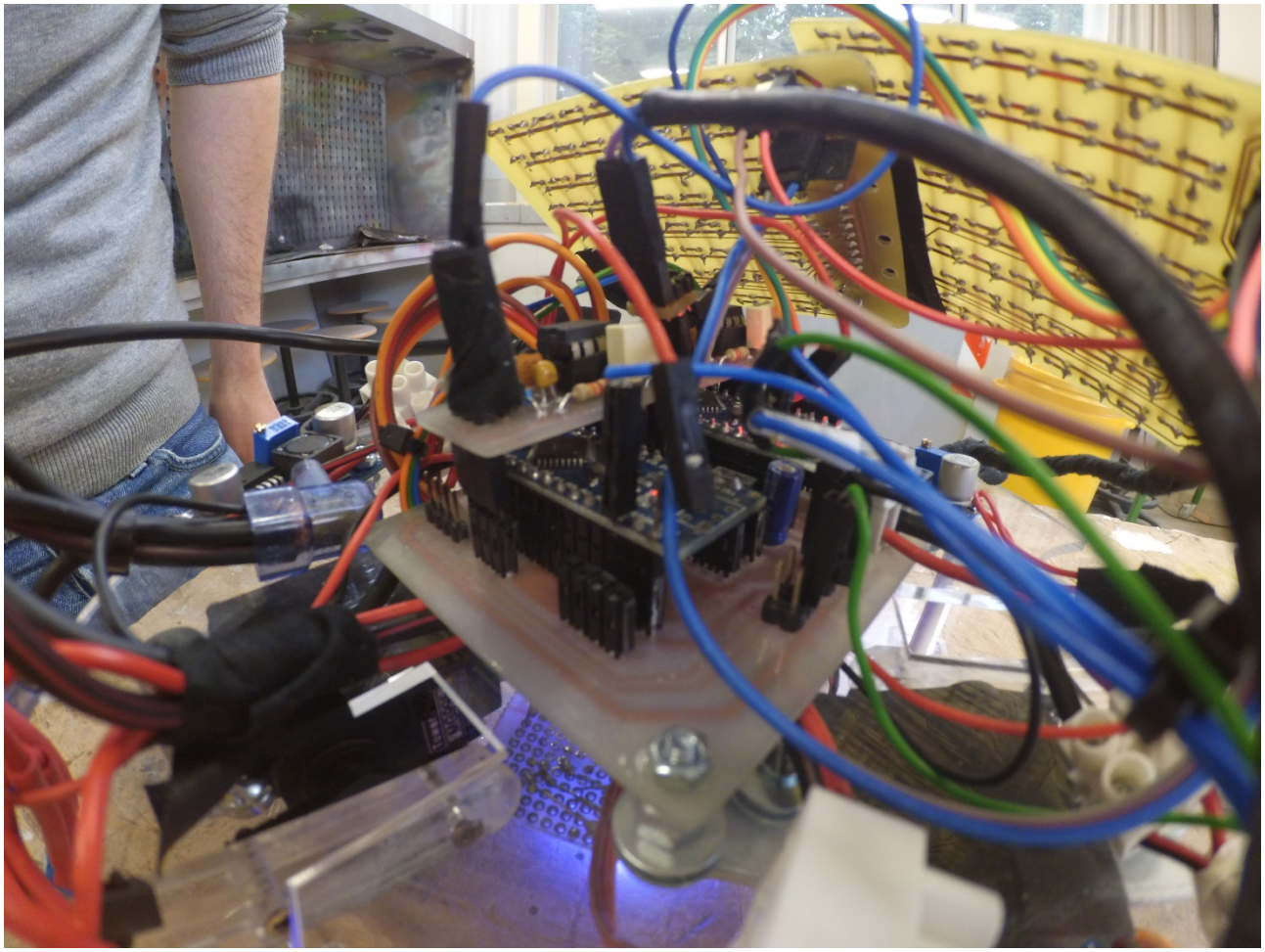
The circuit schema is the following:



The H bridge has been realized with the IC L293. Capacitors are useful to suppress eventual noise on the supply. The circuit is driven by Arduino connecting ARDUINO_CTRL to HBRIDGE_CTRL on the mainboard respecting pin positioning based on their names.

This circuit has not been implemented on the real robot in order to respect the time scheduling of the course. Code writing, management of heat dissipation produced by this ciruit and motors, and management of power source for working with higher peak currents would have required much more time that, for having a finished working robot, we had not.

Inside the mainboard has been provided a terminal board MSTBV2 to power up the whole board, a V_IN connector for powering Arduinos by the Vin pin, a RESET connector for the connection of a reset button, external to the robot, connectors for powering all other external peripherals and more other connectors to supply different tensions present on board (5V, 3V3 output from Arduino, VCC input) to eventual future extensions.

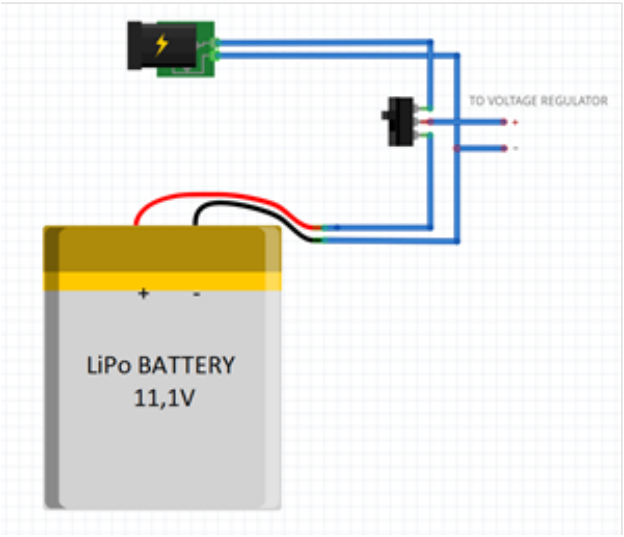


3.5 Power Source

In order to correctly power up all the circuits described earlier, it has been necessary to design a network that can supply to each circuit the right tension in a stable way and that guarantees the distribution of the maximum current needed.

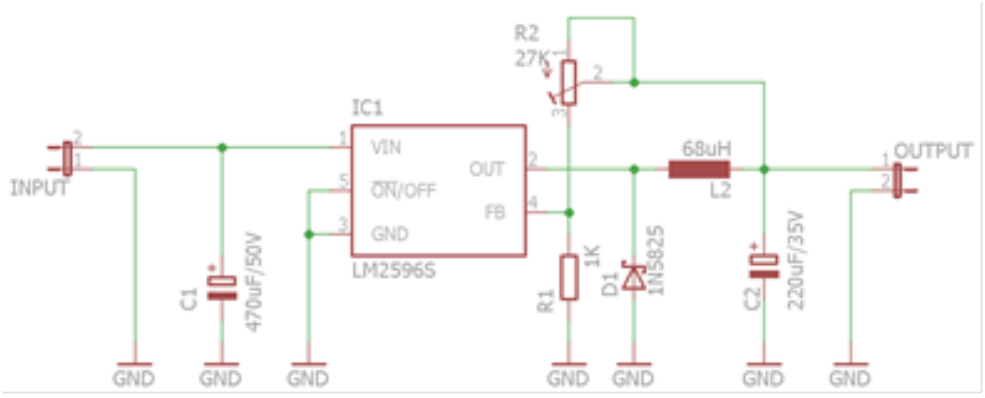
The robot can be powered both through an external power supply with an output voltage between 9V and 45V and a max current output higher than 3A (the one that we use is 90W, with a 19V tension and 4,73A), or from a LiPo battery with a nominal tension of 11,1V.

We use a 3-way switch that lets us turn the robot on selecting the type of power source or to leave it in off state (ON NETWORK, OFF, ON BATTERY).
The circuit is as follows:



NOTE: in case of using a LiPo battery, in order not to damage it, for safety reasons and also for right robot operation, it is necessary to put a control circuit between the battery and the connector that can turn off the alimentation if the battery tension goes below 9V.
Because of time constraints we couldn't design this circuit. There are many similar circuits in commerce that emit a sound/visual signal in case of low battery. In our case this would have been useless, because it would have been difficult to catch those kind of signals given the nature of our robot.
To avoid this eventual problems we prefered to use the external source.

We have used 3 switching step down DC-DC voltage regulators linked in parallel to supply the right tension to each circuit of the robot. The schema of each one is the following:



Given the great diffusion on the market of this circuit, with low prices, we opted to buy them ready-made:

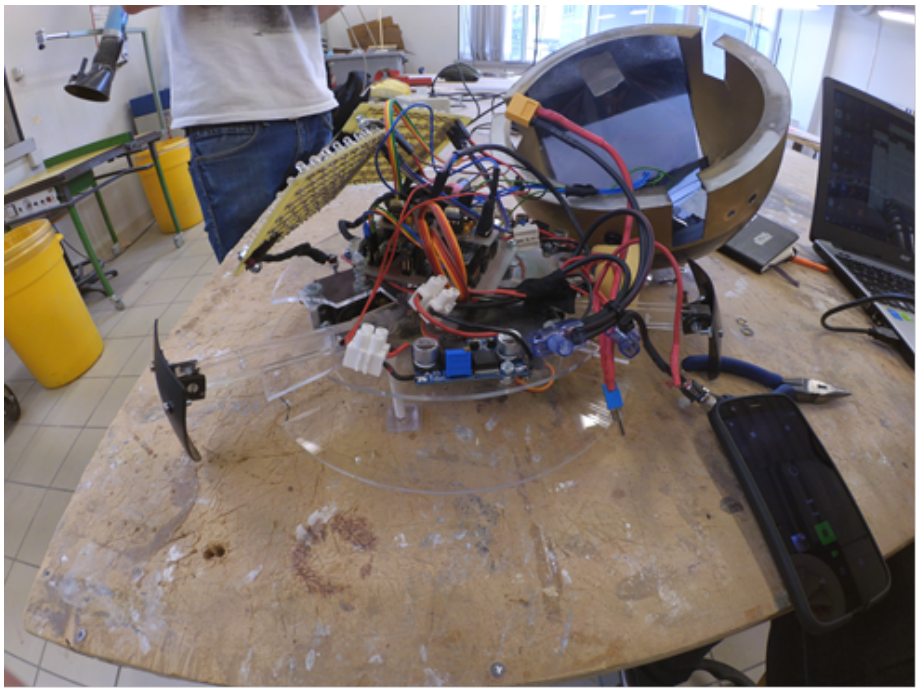
This kind of regulators, with respect to classic linear regulators (e.g. 7805, lm317), let us lower the input voltage to the desired values (acting on a trimmer), yet without having a high heat dissipation that, in a plastic robot, could have led to problems.

The three regulators are used as follows:

- One regulator to supply 7V to the Vin input of Arduinos without burning them and without heating their internal (linear) regulators.
- One regulator to supply 6V necessary to control servo motors.
- Last one to supply 5V useful to pilot led matrixes and rgb ring.

Initially we thought about powering all external peripherals at 5V, included servo motors. So there should have been only two regulators, one for Arduinos, and the other one for the rest. This one would have been connected directly to the power supply on the mainboard, that in fact has been designed with a unique VCC line where all peripherals connect.
Later on we decided to add another regulator to provide 6V and to separate alimentation of servos and lights. The reasons behind this are two:

- Increasing the speed and torque of servo motors;
- Isolating the lights from eventual interference made by servo motors.



4 CODE

4.1 Leds

What we wanted to achieve were nice light games following the music rhythm, like the real daft punk helmets. Our original idea was close to the end result, even if the main problem is the lack of different colors on the matrixes (we ended up only having blue and green, red leds were not powerful enough).
We have two fully addressable 8x8 matrixes and a 9 RGB leds ring.

The code for lights management is organized in different files:

- LedController is the main file, it contains global variables, the setup and loop funcitons.
- Applause contains the code to recognize an applause and the winner function
- MSGEQ7 has the function to read the frequencies values from MSGEQ7 component
- VisualEffects contains the functions to animate the matrixes
- help_rgb manages the rgb led on the bottom of the robot

To manage the led matrixes we have used the LedControl library, written by Eberhard Fahle for the MAX7219 Led Display drivers.

The matrixes have 3 different visual effects:

- The first one is a simple VU meter, that reflects the input taken from the msgeq7. This component outputs 7 bands, while the matrix has 8 columns. In order not to have an unused column, we decided to make the 4th column the mean between the 3rd and 4th band's values. Since with the MAX7219 the LEDs are addressable one by one, we made an array of bitmasks, each representing an increasing number of leds lit. For example, the bitmask 'B11110000' lits the first 4 LEDs.
- The second effect is a happy face, drawn by using the right bitmask for each column, that is like drawing on a squared paper. This effect has a simple animation that moves the eyes up and down every 200ms.
- the last one is the 'wave' effect, that takes a random index between 1 and 5 (the central audio bands) and every 20 milliseconds prints the value of that band in the last column and shifts all the other columns to the left, making a kind of moving wave shape.

The flow of the program is as follows: in the setup() function we reset the MAX7219 drivers, clear displays and set the intensity of LEDs. Then we also reset the MSGEQ7, so that we are sure to read the right values in order (from first band frequency to last). In the loop() function we first read the values from MSGEQ7, we check for applause, and manage RGB LEDs. After that the program starts calling VUMeter(). The loop goes on this way until 20 seconds have passed, then show the happy face for less than 1 second and, after that, it restarts, this time calling the wave() effect instead of VUMeter. This swap goes on every 20 seconds.

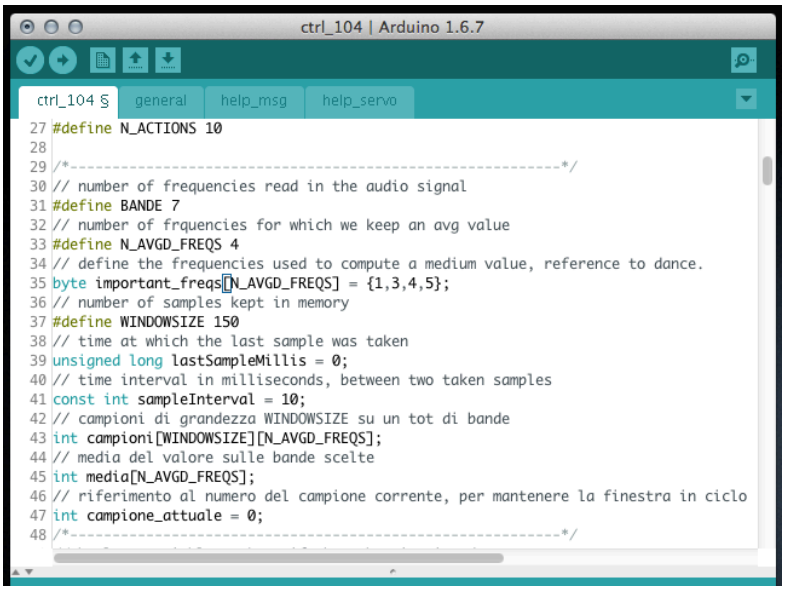
When an applause is detected, the matrixes show the happy face for about 10 seconds and after that goes back to the normal flow.

4.2 Motors

Here we are going to explain the fundamental points of the motors controller. The whole code is pretty long and we will deliver it as an attachment to this document.
We organized the code in different files, in order to distribute many helper functions related to different fields of the project and keep clean the main program.
We describe here the general content of the files, but many functions will be explained later:
• help_msg contains the functions to read the input signals, initializes some variables used by the controller and updates the average values.
• help_servos contains the basic moves and the function to check if the robot is not doing any action.
• general contains general purpose functions, like the applauseCounter() (function that tries to understand if the robot is receiving applauses) and winner() (the function that shows a happy behavior).
• ctrl_104 is the main program to control the servos, containing the setup() and loop() functions.

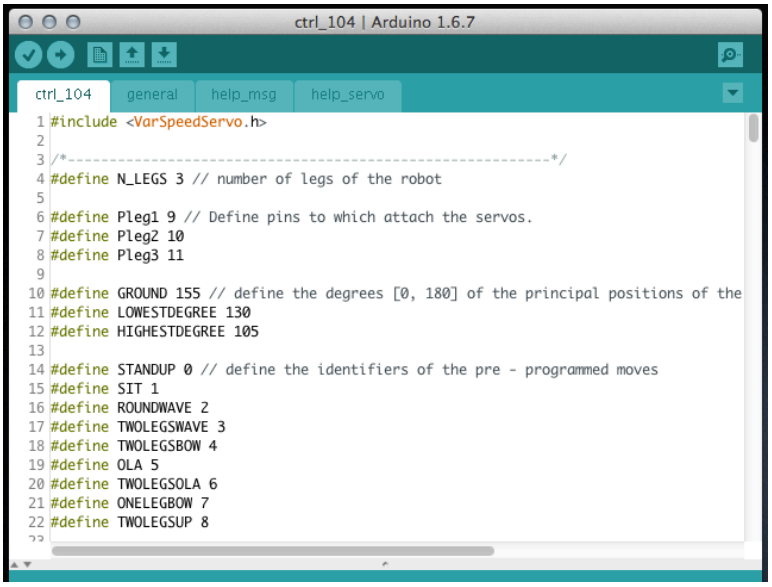
The key concept is to keep the average value of some of the available frequencies over a certain interval of time in order to be able to identify peaks of the music in that frequencies, and move the robot's legs as a reaction to these peaks.
To implement this functionality, we sample the music every t milliseconds in a matrix of size [n x m] where n is the number of values averaged for each frequency, and m is the number of frequencies for which we keep a track. This matrix represents a sort of array of queues and allows us to compute the average value over the last (t * n) milliseconds. These numbers are the result of a

trade-off between the need to know the mean values for the highest possible number of frequencies and the small size of Arduino memory.
As shown in sketch 1 we came out with an analysis of 4 frequencies (over the 7 available), sampling each of them every 10 milliseconds and keeping 150 samples for each one, so that we can evaluate peaks with respect to a dynamic threshold, based on the last 1,5 seconds of music.



Sketch 1

Sketch 2 shows parameters related to the movements of the legs. The main idea is to have 3 basic positions for each leg, and a number of preprogrammed actions, defined as coordinated movements of one, two or three legs in a sequence of these positions.

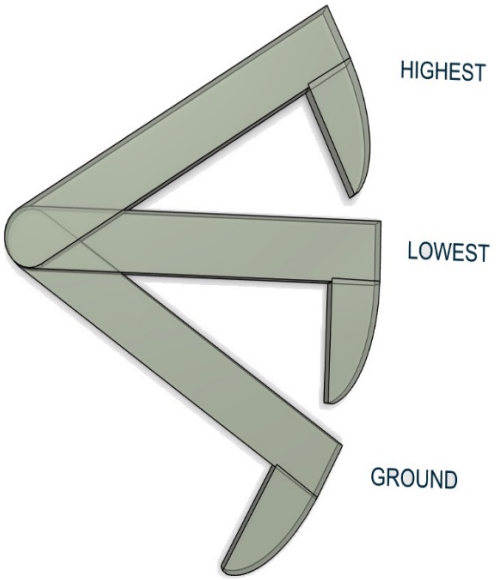


Sketch 2

Sketch 3 shows the positions of the legs and a sample code of these moves. Simple moves, like stand or sit, are completely done in 'false' mode, in order to not stop the loop and continue to sample the music during the move, while coordinated moves require sometimes to move servos in a synchronized way, waiting for the motor to finish his run ('true' mode).

ctrl_103 | general | help_msg | help_servo

```
74
75 //-----------------------------------------------------------------
76 //-----------------------------------------------------------------
77 /**
78  * FUNCTION TO MOVE THE ROBOT STARTING FROM POSITION WITH LEGS
79  * TOUCHING THE GROUND TO POSITION WITH WHEELS TOUCHING THE GROUND
80  */
81 void sit(int sit_speed) {
82   // set the status in 'currently doing action SIT'
83   actions[SIT] = 1;
84   setInterval(SIT, sit_speed);
85
86   // check if the robot was actually on the ground position.
87   if  ((legs[0].read() == GROUND) && (legs[1].read() == GROUND) && (legs[2].read() == GROUND)) {
88     // sit down on the wheels moving legs simultaneously
89     legs[0].write(LOWESTDEGREE, sit_speed, false);
90     legs[1].write(LOWESTDEGREE, sit_speed, false);
91     legs[2].write(LOWESTDEGREE, sit_speed, false);
92   }
93 }
  :
132 /**
133  * FUNCTION THAT MAKES THE ROBOT OSCILLATING ON TWO OF ITS THREE LEGS
134  */
135 void twoLegsWave(int excludedLeg, int wave_speed) {
136
137   // set the status in 'currently doing action SIT'
138   actions[TWOLEGSWAVE] = 1;
139   setInterval(TWOLEGSWAVE, wave_speed);
140
141   for (int l = 0; l<3; l++) {
142     if (l != excludedLeg) {
143       legs[l].write(LOWESTDEGREE, wave_speed, true);
144       legs[l].write(GROUND, wave_speed, false);
145     }
146   }
147 }
148
```



Sketch 3

We also developed a system to not overwrite a previous move (written in 'false' mode) with another during the loop. We keep an array with the time needed for each action to complete, and a boolean array saying for each action if it is running or not (only one of the actions can be running at a certain time). At every loop we save the current time, and compare it with the time the last action started, in order to see if it has finished or not.

The loop code is organized in these steps:
1 - reading of the signals
2 - sampling when necessary and update averages
3 - try to reveal applauses
4 - peaks detection
5 - check if previous action is finished
6 - proceed with another, depending on the combinations of peaks in the frequencies under analysis

The code is quite linear and commented.

We just want to clarify the way applauses are revealed: at some point in the loop we call the applauseCounter function passing to it the array with the spectrum values. This function checks if the values correspond to a certain pattern we verified to happen as a consequence of a human hands clapping and if the case, it increments a counter. The same happens at the next loop and if there is not a clap within a certain interval of time, the counter is reset. When the counter reaches a threshold after a continuous clapping (about 4 or 5 seconds in practice), the winner function is called, that makes the robot bow to the public.

5 PROBLEMS

We have had some issues during the 2 months we worked on DAFT DRUNK, but we managed to solve them briefly and deliver a fully working robot for the project deadline.

5.1 Component Problems

To assemble the final product, we had to buy some external components:

2 Arduino Nano
1 microphone
2 Msgeq7
2 Max7219
3 Servo RG995
129 Leds
9 RGB Leds
PCB materials
L2965 Voltage regulators

Some of the components we ordered got stolen, others were faulty but fortunately in the end everything worked.

5.2 Software Problems

The software code has been developed mainly by Damiano, Alessandro and Carlo and has been written with a hybrid approach between Bottom-Up and Top-Down. This means that we had a general idea about how the Arduinos and the components should interact but we started coding simple functions starting from each component separately, at first with basic functions/movements and then with more complicated ones, always keeping in mind how they interacted with the Arduino they were connected to.

Despite the usual programming issues (variable initialization, missing semicolons, function parameters, etc.) we encountered two “funny” problems:

1. OVERFLOW: as we explained before in section 4.2 we wrote a function to calculate and detect samples in a dynamic way. While the function was running we noticed that at a certain point the Matrix stored negative values. At first we couldn’t understand why an averaging calculus of a sum produced negative results (It was impossible since we divided a positive quantity by an integer). So we came to the conclusion that we were in front of the informatics behavior known as overflow.

2. ROUNDING: Daft Drunk makes dancing moves based on the frequency peaks it detects. We decided to look at 4 frequencies and find the peaks on those. We stored these information on a 4 bit array, ranging values from 0 to 15 (0000 - 1111 in binary), and then assign the moves based on cases of a *switch* statement.
   In order to get the base-10 value of the number stored in the array, we needed the power operator. This gave us many problems because we eventually discovered that the Arduino power operator didn't work as we expected. For example, 2^3 gave 7.99 which was rounded to 7 since we declared the variable for the base-10 value as integer. This error messed up the behavior of the robot, so we wrote a power function by ourselves.

6 BILL OF MATERIALS

Table 1 - Electronics

Arduino nano	€ 13,80
L293D	€ 3,56
Microphone	€ 7,93
MSGEQ7 (robotics-3d)	€ 11,46
MAX7219	€ 3,40
Servos (x4)	€ 33,00
DC Motors (x2)	€ 10,16
Led (50 x 3 colors)	€ 12,00
Tin	€ 5,00
Iron Chloride	€ 7,50
Shield	€ 2,50
Doubleface Shield	€ 3,20
L2965 Regulators	€ 5,00
Wires	€ 6,00
Connectors	€ 1,20
Switch	€ 1,00
Other Electronics	€ 9,00
Shipping	€ 21,90

Table 2 - Design materials

Plexiglass	€ 12,80
Helmet's plastic	€ 8,20
Laser cutting	€ 36,50
Spray color (x3)	€ 15,30
Spray color (smoked black)	€ 21,46
Screws	€ 2,20
Paper and polistyrene for the box	€ 4,40

7 CONCLUSIONS

The project required commitment and participation from every team member, and although it was demanding in terms of experimentations, meetings and work, we had fun in developing the robot.

We enjoyed taking this course for how it's structured and because it is quite unusual, but also because we worked together the most of the time coordinating the different jobs and, most important thing of all, we got along really well.