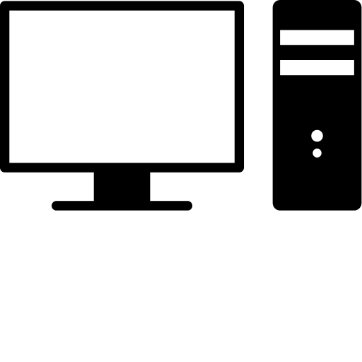
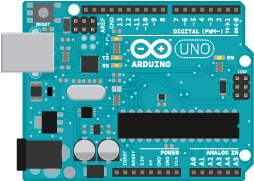
Robot swarm state analysis tool

This project is about developing a tool to handle a block of RGB LEDs; The block is an atomic unit. However, the tool must be able to handle several of these atomic units. The purpose of handle these units is producing different arenas with them and generate required LED patterns. Inside the arena are going to be robots, whose behavior will respond to the pattern of the arena.

The tool will be developed using several technologies as are:

* Arduino UNO
* LED strip APA102 IC
* Python
* XBee module

Figure 1. System diagram



Serial communication

Block unit with n RGB LEDs.

Arena

Arduino

Hardware Specifications

# Arduino UNO

Arduino Uno is a microcontroller board based on the. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller.



# LED strip APA102

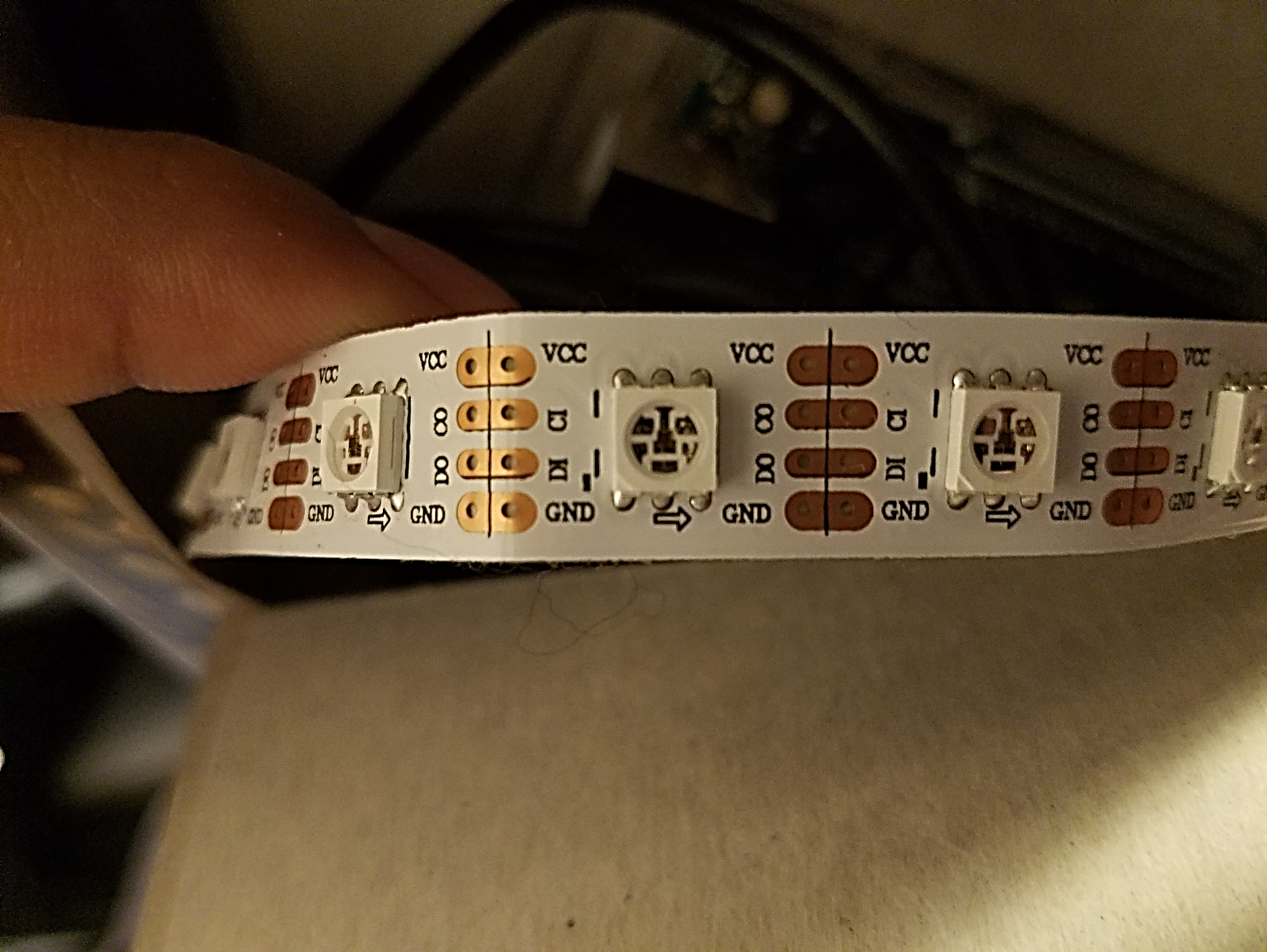
APA102 is a type IC for the three-color RGB control strip, This IC using the CMOS process to provide three-color RGB LED output driver to adjust the output with 256 gray scale and 32 brightness adjustment, APA with 2 signal output way, one is clock, another is data, the clock and data are synchronized, so that the crystal cascade piece of output movement is synchronized.

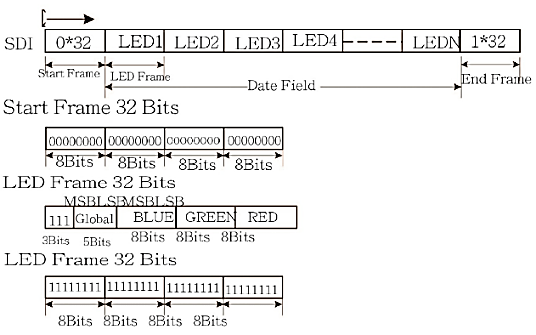
**Electrical Characteristics**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Symbol** | **Parameter** | **Condition** | **Min.** | **Typ.** | **MAX** | **Units** |
| **VDD** | **Supply Voltage** | **5.0** | **5.5** | **V** |  |  |
| **VIH** | **Input High Voltage** | **0.7VDD** | **VDD +0.3** | **V** |  |  |
| **VH** | **Input Low Voltage** | **VSS-0.3** | **0.3V DD** | **V** |  |  |
| **LOL** | **Sink Current Voltage(RGB)** | **VDD=5V, VOL>1V** | **22.5** | **24.5** | **26.5** | **mA** |
| **RIN** | **Pull High** | **VDD=5V** | **570** | **kΩ** |  |  |
| **VREG** | **Regulator Voltage(VREG)** | **VDD=5V** | **4.4** | **4.5** | **4.7** | **V** |
| **FOSC** | **Oscillator Frequency** | **800** | **1200** | **KHz** |  |  |

|  |  |  |
| --- | --- | --- |
| **No.** | **Symbol** | **Function description** |
| **1** | **DI** | **Data input** |
| **2** | **CI** | **Clock input** |
| **3** | **DO** | **Data output** |
| **4** | **CO** | **Clock output** |
| **5** | **GND** | **-** |
| **6** | **VCC** | **+ 5V** |

LED component



**Data format**

For further detail please

Consult the datasheet.

Project planning

We are going to divide the project in 4 stages:

1. Master the LED strip by means of Arduino UNO.
2. Master the LED strip from the computer via serial communication.
3. Master the LED strip from via wireless serial communication
4. Develop the interface to communicate with the simulator

# Master the LED strip by means of Arduino UNO

This stage consists in connect the LED strip to the Arduino and start discovering the different tools that already exists to manage the LEDS, also consists in learn the right way to manipulate the LED strip (power supply, how to weld it, how to cut it).

To control the LED strip from the Arduino we’ve considered the follow libraries:

* **FastLED 3.1**, https://github.com/FastLED/FastLED
* **APA102/SK9822** library for Arduino, https://github.com/pololu/apa102-arduino

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characteristic** | **FastLED 3.1** | | **APA102/SK9822** | |
| **Good** | **Excellent** | **Good** | **Excellent** |
| **Documentation** |  | ✓ |  | ✓ |
| **Support** | ✓ |  |  | ✓ |
| **Instructions set** | ✓ |  |  | ✓ |
| **Platform compatibility** | ✓ |  |  | ✓ |
| **LED strip compatibility** | ✓ |  |  | ✓ |
| **Speed** | ✓ |  |  | ✓ |
| **Friendly** |  | ✓ |  | ✓ |
| **manageability** |  | ✓ |  | ✓ |

Due FastLED 3.1 is mature library manipulating LED strips comparted to APA102/SK9822, for this project FastLED 3.1 is going to be used. This library allows us to manipulate brightness, color (HTML Color, RGB, HSV), data rate, output pins, etc.

Here is an example of the code, working on Arduino, and a briefly explanation of it.

#include <FastLED.h>

The first line includes the library to the project, the next lines are constant definitions.

#define NUM\_LEDS 12

#define DATA\_PIN 13

#define CLOCK\_PIN 12

CRGB leds[NUM\_LEDS];

Creating the logic LED strip, whose size has been defined above.

void setup() {

//FastLED.addLeds<APA102, DATA\_PIN, CLOCK\_PIN, BGR, DATA\_RATE\_MHZ(12)>(leds, NUM\_LEDS);

FastLED.addLeds<APA102, BGR>(leds, NUM\_LEDS);

LEDS.setBrightness(25);

}

FastLED.addLeds you can set the configuration for: LED strip type, Data pin, Clock pin, RGB order, Data rate. Also, you have to set 2 parameters: CRGB array of leds and the Number of leds in your strip, which you have defined before.

void loop() {

This loop is going to set all the LEDs of the string with some color.

for (int i; i < NUM\_LEDS; i++) {

if (i % 2 == 0) {

leds[i] = CRGB::Orange;

}

else {

To illustrate they can be managed independently, there is a condition that will set the even LEDs to one color and the odd ones to a different color.

leds[i] = CRGB::Cyan;

}

}

FastLED.show();

delay(15000);

You can access to each LED of the strip individually by means of the logic array using an index. You can set a color using this notation “CRGB::” plus the name of the color in web/HTML color code (v2).

for (int i; i < NUM\_LEDS; i++) {

if (i % 2 == 0) {

leds[i] = CRGB::Green;

}

else {

leds[i] = CRGB::HotPink;

}

}

FastLED.show();

This function shows the result on the LED strip. The delay is to wait for a given amount of time and start again or pass to the next instruction.

delay(15000);

}

This code basically starts showing the LED strip with the even LEDs in orange, the odd ones in Cyan for 15 seconds then changes to Green and HotPink for 15 seconds. This process is repeated within an infinite loop.

Demo link: https://youtu.be/mSmZi78Zdxw.

# Master the LED strip from the computer via serial communication.

## First attempt

One can send information from the computer to the Arduino by means of serial communication, this is important for the developing of the project because we need to manage the LED strip dynamically and not just one pattern inside the Arduino. The idea is to send an instruction to the Arduino from the computer to change the current pattern. To develop this idea is necessary put in to the Arduino a program that must be able of interpret the instruction that comes from the computer and therefore update the LED stirp with the new pattern.

To reach the goal of communicate both parts, it is necessary to establish a convention due the program in the Arduino is going to be static. To solve this issue, we have selected a JSON with the following structure.

{

"colorConf": {

"numLeds": 12,

"unitSize": 3,

"brightness": 15,

"patternSize": 3,

"colorPattern": [

{

"rgb": [0, 0, 255]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [255, 0, 0]

}

]

}

}

|  |  |  |
| --- | --- | --- |
| **Label** | **Description** | **Value** |
| colorConf | This object contains all the configuration parameters. | It has the parameter objects inside. |
| numLeds | This object corresponds to the total number of LEDS in the arena. | Must be an Integer value. |
| unitSize | This object corresponds to the size of the atomic unit used in the color pattern. | Must be an Integer value. |
| brightness | This object corresponds to the percentage of brightness of the entire LED strip. | Must be an Integer value between 0 and 100. |
| patternSize | This object corresponds to the quantity of colors used in the color pattern. | Must be an Integer value and match with the size of the colorPattern. |
| colorPattern | This is an array of objects that contains the rgb colors of the color pattern. | It can have as many rgb objects as it is needed. |
| rgb | This is an Integer array that represents the color in the RGB systems. | Must be a three lenght array, where the first position represents red, the second green and the third one blue. |

It is important to mention that the pattern is going to be repeated until the size of the LED strip, for instance Let the numLeds = 12, unitSize = 3, patternSize = 3 and the colorPattern = [blue,green,red], the LED strip will show, 3 blue LEDs; 3 green LEDs; 3 red LEDs,(here the pattern starts again) 3 blue LEDs.

To process this JSON in the Arduino program, the ArduinoJson library is going to be used, its documentations can be found in the following address:

* ArduinoJson 5.13.1, <https://github.com/bblanchon/ArduinoJson>

With this library one can be able to decode the values of the JSON into Arduino variables and therefore change the behavior of the static program deployed in the Arduino. Here there is and example of the JSON decoding.

int numLeds = root["colorConf"]["numLeds"];

int unitSize = root["colorConf"]["unitSize"];

int brightness = root["colorConf"]["brightness"];

int patternSize = root["colorConf"]["patternSize"];

Once defined the communication convention, one can start to think about the channel of communication in this case is simpler because the Arduino provides a serial communication interface which can be initialized as follows, where SRATE is the rate in baud:

void setup()

{

Serial.begin(SRATE);

}

Then one can start receiving data using a while statement:

void serialEvent()

{

while (Serial.available()) {}

}

Here there is some examples of inputs and they respective outputs.

{

"colorConf": {

"numLeds": 12,

"unitSize": 3,

"brightness": 15,

"patternSize": 3,

 "colorPattern": [

{

"rgb": [0, 0, 255]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [255, 0, 0]

}

]

}

}

{

"colorConf": {

"numLeds": 12,

"unitSize": 2,

"brightness": 25,

"patternSize": 3,

 "colorPattern": [

{

"rgb": [0, 0, 255]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [255, 0, 0]

}

]

}

}

{

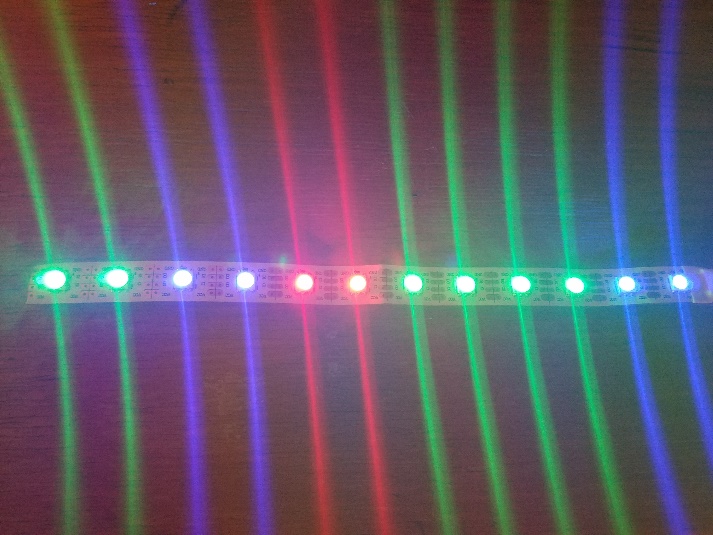
"colorConf": {

"numLeds": 12,

"unitSize": 2,

"brightness": 25,

"patternSize": 4,

 "colorPattern": [

{

"rgb": [0, 0, 255]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [255, 0, 0]

}

]

}

}

{

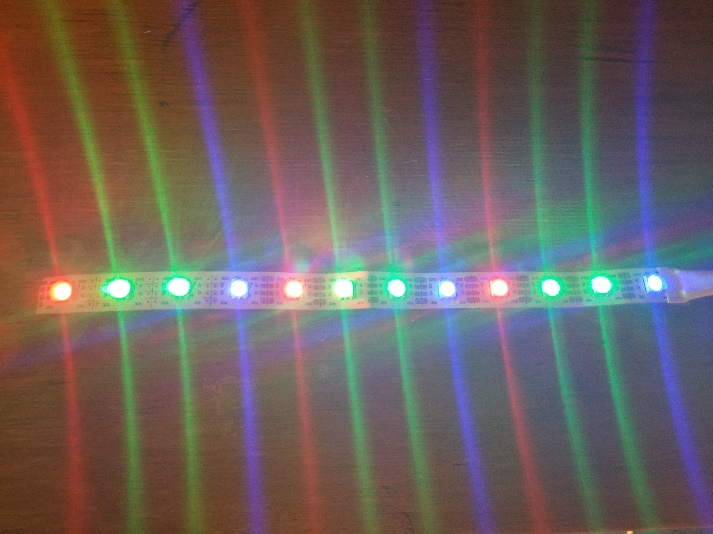
"colorConf": {

"numLeds": 12,

"unitSize": 1,

"brightness": 25,

"patternSize": 4,

 "colorPattern": [

{

"rgb": [0, 0, 255]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [0, 255, 0]

},

{

"rgb": [255, 0, 0]

}

]

}

}

## Second attempt

After the first attempt to build the communication between the Computer and the LED strip by means of Arduino, we have built a first prototype using JSON. However, it is necessary to make some changes due to two important reasons: The memory management in Arduino and the main goal behind this project. It is needed to propose a solution for both. The changes due to the main goal are going to be explained first followed by the memory management.

### Main goal

The main goal of this project is build a system to handle an Arena compounded by blocks. Where each block consists in a led strip about 12 LEDs, 20 cm long. To really understand this idea the concept of the Arena must be clear therefore is going to be explained in more detail.

Basically, the Arena is a polygon, this means an Arena is going to have edges. For example, we can have a Triangle, a Square, a Pentagon, a Dodecagon, etc. This is important because once we now the arena has a specific number of edges, the idea is to form this edges with blocks, for instance one edge could be formed by 2 blocks. Therefore, if there is a Triangle (3 edges), 2 blocks per edge and 2 LEDs per block, we are going to have, 6 Blocks and 12 LEDs in total. let’s illustrate this concept with an image.

Edge 2

Edge 1

5

4

Block 3

Block 2

6

3

7

2

Block 4

Block 1

8

1

12

11

10

9

Block 5

Block 6

Edge 3

Triangular Arena.

Now that the concept of Arena an its components is clear, the proposed language to handle the Arena is going to be explained. This is going to be explained from the essential structure to more complex trying to cover the most important uses of it.

{

"arena": {

This is the minimum element that the language can understand. The three first parameters are the conceptual definition of the Arena, “edges” represents the number of edges it has, “blocks” the number of blocks per edge and “leds” the number of LEDs per block. This configuration

"edges": 3,

"blocks": 2,

"leds": 2,

"color": "red",

"brightness": 10,

"edge":[],

"block":[],

"led":[]

},

}