COMPUTING PROJECT REPORT

The DLA
Dynamic Lighting Arena for swarm robotics

Keneth Efrén Ubeda Arriaza
Directed by Prof. Mauro Birattari
Supervised by David Garzón Ramos & Ken Hasselman
IRIDIA - CoDe
Université Libre de Bruxelles , Brussels

June 12, 2018

Abstract

The study of collective behavior in robot swarms has led to the developing a variety of scenarios in which the robot swarm behavior can be studied. This project was worked in cooperation with IRIDIA's Lab that already has an arena in which they performed different experiments. This report presents the implementation of a Dynamic lighting arena for swarm robotics as an extension of the current IRIDIA's arena. The report explains the main purpose of this implementation as well as the requirements. The implementation is divided in hardware and software sections since the goal is to design and build a system from scratch. The design section describes the design decisions taken and the implementation section, how this decisions have been executed. In the proof of concept section an scenario to validate the requirements is explained describing the properties of the build system.

Contents

1 Introduction

Swarm robotics studies how to design groups of robots that are able to operate without depending on any external infrastructure. Robot swarms are designed based on swarm intelligence principles that aims to make the robots fault tolerant, scalable and flexible systems. Some tasks that could be tackled using swarm robotics are search and rescue, underwater exploration and surveillance among others. A robot swarm is a self-organizing multi-robot system whose sensing and communication capabilities are local and robots do not have access to global information. Basically the robots' behavior is governed by the interactions of each individual robot with its peers and by the environment. Nowadays a large part of the research effort related with this topic is focused on the study of collective behaviors such as spatially organizing behaviors [1] and navigation behaviors. There are many scenarios in which a robot swarm can interact. There are simple scenarios that involves task of low complexity such as aggregation and dispersion, Then one can think in more complex scenarios such as sorting and self-assembly and even more complex scenarios such as collective construction, spatially organization and collective transport. This project has been worked in cooperation with IRIDIA, which is the Artificial Intelligence research laboratory of the Université Libre de Bruxelles. IRIDIA is also a world leader in two of the most successful areas of research in swarm intelligence Ant colony optimization and swarm robotics. They recently completed SWARM-BOTS [2] project which has succeeded in producing a viable distributed robotic system based on the swarm robotics methodology. Their continuous researching within this field has been given a chance to developing a variety of projects in order to extend their actual scenarios, such as Dynamic lighting arena for swarm robotics project that aims to extend the functionality of the robot swarm arena. The arena is a polygonal shape compounded by edges and the edges are compounded by wood blocks of a fixed size that surrounds the robot swarm in order to define the robot swarm environment. So this project aims to create a lighting arena using blocks of RGB LEDs handled with Arduino and a top layer built in Python which are going to be discussed in detail in the following sections.

2 Project Description

The Dynamic Lightning Arena is a modular system aimed to extend the richness of the workspace in experiments with robot swarms at IRIDIA. Beyond being a workspace delimiter, the DLA allows researchers to interact with the robots by displaying colored signals in their environment. In this section I formally describe the objective of project and the definition of the requirements for developing the DLA.

2.1 Objective

The project objective is to develop a modular, flexible and dynamic system for displaying colored signals in the walls that delimit the workspace in experiments with swarms of E-Puck robots.

2.2 Requirements

I structured the development of the DLA on the basis of the functional and non-functional requirements provided by IRIDIA's researchers. In this section I describe the specifications that guided the design and implementation of the system.

The main functional requirements are listed below.

- The DLA should be designed on the basis of an atomic unit defined as block.
- The blocks should have the property of changing their color in response to the commands provided by the user.
- The users should be able to combine the blocks in any way to form a polygonal structure.
- The users should interface with the polygonal structure at any level (i.e. from atomic units to the whole arena).

The main non-functional requirements are described in the following.

- The system should be scalable, as it could be composed by any number of blocks.
- The system should be flexible, as it could be integrated in different software architectures.
- The system should be portable, as it could interface with different operating systems.

3 Design

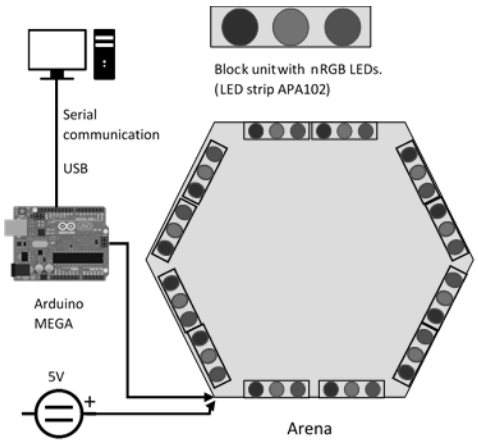


Figure 1: Hardware architecture.

On the basis of the requirements defined in the past section, I decomposed the development of the DLA in its hardware and software constituents. The design process of both elements is summarized in this section.

3.1 Hardware design

I structured the hardware architecture using three components: blocks, an Arduino Mega board and a computer. A graphical representation of the architecture can be seen in Fig. 1.

The first and most basic component are the atomic units defined as blocks. The blocks are entities which function is to display color lights in the workspace of the robots. With this purpose, IRIDIA's researchers and I conceived a small wooden box that integrates a LED strip APA102, its electronic connectors, and a light diffuser. The DLA controls the LED strip APA102 to display different colors on the block, and the diffuser prevents a high saturation of the light.

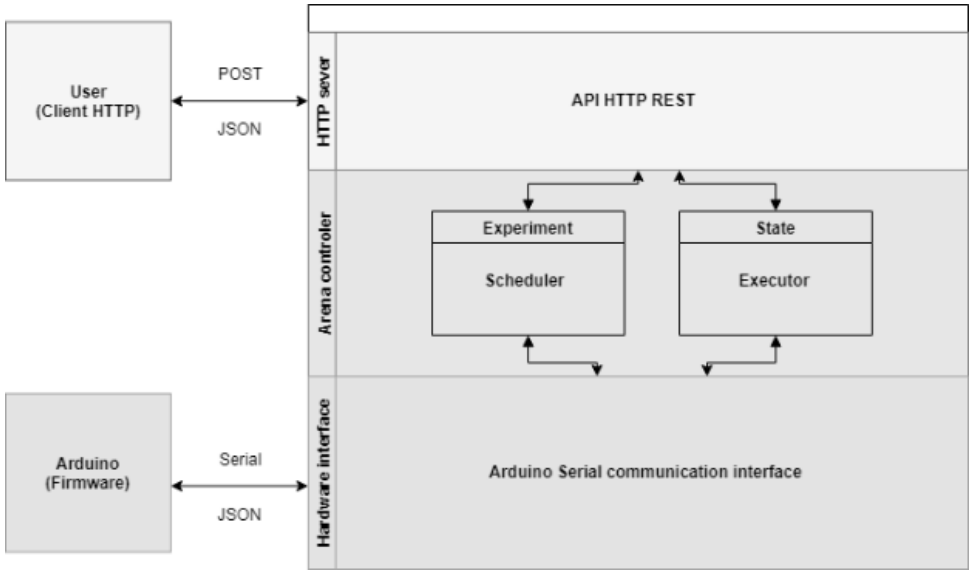
The next component of the hardware architecture is the Arduino Mega. I chose this development board for interfacing with the LED strip APA102 due to its simplicity, flexibility and reliability. Arduino Mega is capable of controlling the LED strip through the SPI protocol, and boards with more complexity were not required(e.i. Raspberry Pi). Moreover, Arduino developers have provided several libraries to control the LED strip.

Finally, the computer host the top layer of the sofware architecture and allows the user to interface with the rest of the software. This part will be described in more detail in the next sections.

3.2 Software design

Since Arduino is a micro-controller thought to interact with hardware and done very specific tasks one can found some limitations in terms of memory. However because this project aims to develop such a flexible and enriched system to handle an lightning arena it is necessary to build a upper layer in order to extend the firmware developed in the Arduino. This system has been developed using Python and based on the following architecture.

Figure 2: Software architecture.



There are basically 3 layers: Hardware interface that consists in the interface to communicate via serial with the Arduino, The HTTP sever in which the services are exposed and the Arena controller in which most of the logic is found. Also a firmware for the Arduino is needed. As mentioned as a programming language Python is used in its version 3.6.4. In order to develop the web server the aiohttp 3.2.1 is used and to achieve the serial communication the pyserial 3.4 library is used. Since an experiment can have many states and theses states change through the time an scheduler is used which is in the standard libraries Python already include. It is important to mention that the http server handles the request asynchronously, that means that since the request is well formed the server responds with an received status and execute the instructions in a different thread.

4 Implementation

4.1 Hardware implementation

In this section the Hardware implementation is described in detail mentioning important remarks about the choices taken, functionality and troubleshooting throughout the development process.
As it is mentioned in the design section one of the fundamental components in this project is the LED strip APA102 which has been cut in order to form blocks of 12 LEDs. 12 LEDs has been selected since the required size of each block is 20 cm. Work with blocks instead of work directly with edges or even the complete arena as a LED strip is a decision taken due to the molecularity and flexibility mentioned in the non-functional requirements. Work with blocks help us to replace pieces in case of failure and at the same time isolates the risk to little units. Since this arena is designed to robot swarms which cameras can be very sensible to the color, the blocks have a plexi-glass front layer to not saturates it.
Let us know described the Arduino which is also a fundamental part of this system. In this project the Arduino MEGA, as micro-controller Arduino provide us a set of inputs and outputs in which the device we want to handle is connected. Hence the block has 4 different inputs: data, clock, ground, voltage; three of them are connected to the Arduino: data in the 51 input, clock in the 53 input and it is also connected to the Arduino ground input. The voltage supply for the block is not directly connected to the Arduino but to an external source about 5 volts and 1.5 amperes. These blocks are going to be interconnected to form edges and the edges will form the actual arena. So the arena has to be connected to the Arduino by means of the first block of the first edge that forms the arena. All the instructions executed are transmitted to destination passing throughout the preceding blocks.
The hardware system as a whole works as follows. Firstly all the blocks are connected one after the other forming the whole arena, the first block is connected to the Arduino and the power source. Secondly the Arduino is connected via USB port to the computer that runs the upper software layer in order to receive the instructions via serial communication, and execute them doing the arena change depending on the received instruction. Some issues were found during the implementation. For instance the implementation started using the Arduino UNO which actually lacks of memory to handle enough amount of LEDs since per each LED, using the library mentioned before, 3 bytes are needed.

4.2 Software implementation

The main functionality of this projects has been implemented as software layer. which basically consists in: the Arduino firmware, the serial communication interface, and an interpreted language which can be used by means of a web API.
Let us start explaining the built firmware. As is known a micro-controller needs and embedded system which is in charge of execute an specific given task. So Arduino has its own programming language to build the firmware system and can easily be uploaded using the Arduino IDE. However since the goal of this project was to built a flexible and configurable system Arduino presents some disadvantages. Due to that it was very important to delimit the functionality the Arduino would have. In this implementaton Arduino essentially is able to manage individual blocks, this decision has been taken due this is our atomic piece in the arena and is the perfect balance between manage everything LED by LED or edge by edge. This gives us some advantages in the software layer since build the instructions block by block is more efficiently and also give us better performance results at the time of execute the instructions in the hardware since we are not sending the information LED by LED but in blocks. To reach this behavior the two libraries mentioned in the design section has been used. FastLED allow us to handle the LED strip APA102 as an array of LEDs in which each LED uses 3 bytes memory, also allows us to set the color of the LEDs in different systems in which RGB system is used due to can be easily manipulated as three integer variables. Since the communication between the Arduino and the computer is serial a protocol has been defined using the ArduinoJson library to parse the incoming instruction. The JSON string is the follwoing.

Figure 3: Serial instruction

```
{
    "brightness": 25,
    "block": "0,12,0,0,255",
    "led": [ "0,0,255,0" ]
}
```

The figure above shows how a instruction looks like where in "brightness" a integer value between 1 and 100 inclusive is set, in "block" the string is formed by five different parameters separated by commas: the first parameter represents the index of the block which must be a number from 0 to n where n is the number of blocks in the arena, the second one the number of LEDs that compound a block, and the last three corresponds to the RGB color configuration. And in "led" represents an array that can contains as many elements as LEDs has a block. the value of each element of "led" is formed also as an string representing five

parameters the same as "block" with the only difference that the index is pointing at the position of the LED inside the block. Let us now explain the software top layer built with Python. First of all an interface to communicate was needed, this interface was developed using the pyserial library that allows us given a port and a baud rate send serial messages to the connected device, which in this case is the Arduino. This first layer basically includes the functions to establish the connection, send a message and close the connection. In the middle the core of this software is done, it can be seen as an interpreter of a JSON-based language. As in the firmware JSON as a markup language can gives us a light, well formed, understandable and well structured properties to define a intuitive and flexible language to use in order to handle the lighting arena. An alternative to JSON was XML however XML is a heavy markup language that evidently cannot be used as serial message to sent to Arduino, knowing that one of the limitations of a micro-controller is its memory capacity. So in order to be consistent through all levels of the implementation and because working with Arduino implies memory limitations JSON has been selected.

Figure 4: Base structure

```
{
    "arena": {
        "edges": 3,
        "blocks": 2,
        "leds": 2,
        "color": "red",
        "brightness": 5,
        "edge": [],
        "block": [],
        "led": []
    }
}
```

The figure above shows the base structure of the language where "edges" represents the number of edges that formed the polygonal arena, "blocks" represents the number of blocks per edge, "leds" represents the number of leds per block, "color" is the string of a limited set of colors predefined, "brightness" is the brightness and bust be a number between 1 and 100 inclusive. Since the arena is physically and conceptually compounded by edges, blocks and LEDs, it is possible to handle it at those levels which are represented as arrays of objects since conceptually arena is compounded by many of them.
The JSON showed in the figure below is the one that can be included many times inside the arrays mentioned before. "color" as before represents the string value of a predefined color and "index" is an array that accepts either 1, 2 or 3 different integer positive and non positive values separated by commas. When only one integer is found this means directly the index of the edge, block or LED as applicable with respect of the arena, if two parameter are found this

represents a range of edges, blocks or LEDs as applicable and if three parameters are found the third parameter indicates the step which represents the difference between each number in the sequence. This last syntax is an analogy to the range function in Python.

Figure 5: Base structure

```
{
    "color": "yellow",
    "index": [ 2 ]
}
```

An important remark here is the fact that the index values can be negative values which depending on the context the object is found it points at different positions in the arena. Related with the index it is important to mention that 0 index cannot be used, an example of the index functionality will be given in the proof of concept section. Since an edge conceptually also is compounded by blocks and LEDs, one can include its respective instruction inside the object, the same with a block that is compounded by LEDs. So the middle layer interprets this instructions and converts them into Arduino instructions to subsequently send them by serial. A web API server has been also implemented on top of the controller to let the final user consume this language as a services. Two services were created one to execute static changes of states and the other to execute experiments.
Summarizing, the flow of the system is the following: A user write any possible instruction based on the language described above and sent it by means of a HTTP client request, the HTTP server receives this information and notifies the client of received the it parses the JSON to subsequently convert to Arduino instructions to finally be executed in the lightning arena. In order to set the serial port, baud rate, log level and log format a configuration file is used, so the system will run according to information set on that file.
During the implementation some troubles related the serial communication were found. Since many instructions can be sent a pause between each of the instructions is done, such pause depends on the set baud rate, the recommended values for this implementations were found by empirical tests which gave as a result the following values: a wait time of 0.0625 seconds between messages and as baud rate of 57600 baud.

5 Proof of concept

The tests to validate the well functioning of the system has been done over the scenario described in the figure 6.

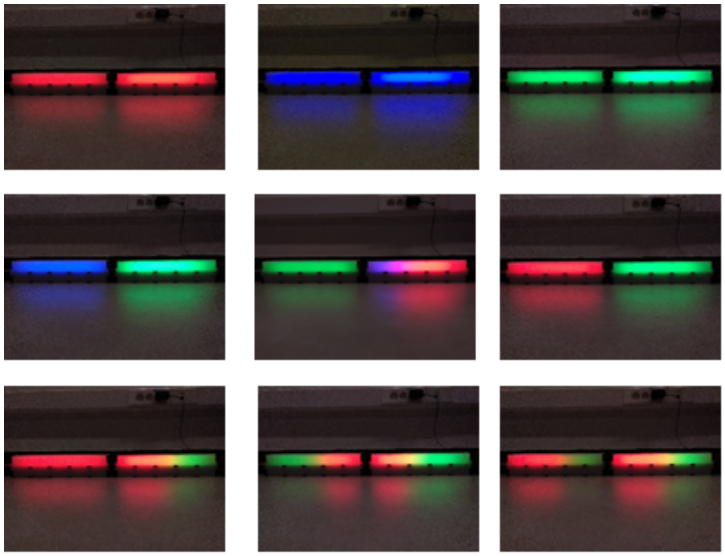


Figure 6: Validating scenario.

Since an important requirement of this project was to be accessible from external applications an HTTP API was done. The API includes two different services, one to execute changes in the color state and the other to schedule experiments including changes of state throughout the time. Basically to validate the functioning of the system one can use any HTTP client like curl, wget if you are working with Linux or one can use specialized tools to make HTTP requests as POSTMAN is. In this validation scenario the last tool mentioned was used. Under the rules explained in the implementation section one can construct states, experiments and send them via the HTTP POST method using the preferred HTTP client. An example is going to be explained in order to show the modularity and flexibility reached through the implemented of the JSON-based language.

(a) Red arena

```
{
    "arena": {
        "edges": 2,
        "blocks": 1,
        "leds": 12,
        "color": "red",
        "brightness": 100
    }
}
```

(b) Blue arena

```
{
    "arena": {
        "edges": 2,
        "blocks": 1,
        "leds": 12,
        "color": "blue",
        "brightness": 100
    }
}
```

Figure 7: Arena's code

For the most simple instruction one can observe the first three images in the figure 6 which

are done with the code in the figure 7, the code shows that we have an arena compounded by 2 edges, one block per edge and twelve LEDs per block. As it is shown one can easily colored the whole arena with a simple instruction and change the color changing the "color" parameter in the code.

```
{
    "arena": {
        "edges": 2,
        "blocks": 1,
        "leds": 12,
        "color": "red",
        "brightness": 100,
        "edge": [
             { "color": "green", "index": [1] },
             { "color": "red", "index": [2] }
         ]
     }
 }
```

Figure 8: Green and red edge

One can easily manage the edges independently as it is shown in the fourth and sixth image in the figure 6 having one edge colored in blue and other in green for the first case and for the second one in red and the other in green. The code used to get these results is shown in the figure 8. The code specifically shows the green and red combination, however changing these colors to green and red respectively we get the other combination. It is important to observe that color set in the arena level is overridden by the edge instructions.

```
{
    "arena": {
        "edges": 3,
        "blocks": 1,
        "leds": 8,
        "color": "green",
        "brightness": 100,
        "block": [ { "index": [2], "color": "red" } ]
    }
}
```

Figure 9: Red and green blocks

In the last three images of figure 6 is shown the results of handling the arena conceptually different. This arena is actually compounded by 3 edges of 1 block per edge and 8 LEDs per block. This show that the system is flexible and can be managed as the user wants to be defined. The code in the figure 9 shows specifically the instruction to get the result of the eighth image in the figure 6. It is important to mention that one can use the the color set in the arena level as a base and simply colored incumbent edge, block or LED. An other

important observation is that the block for this arena definition is acting as an edge since an edge is compounded by only one block.

```
{
    "arena": {
        "edges": 2,
        "blocks": 1,
        "leds": 12,
        "color":"red",
        "brightness": 100,
        "edge": [ { "color": "green", "index": [2] } ],
        "led": [ { "color": "blue", "index": [10,12] } ]
     }
 }
```

Figure 10: Blue LEDs

Send instructions to LEDs individually is also possible as it is shown in the fifth image in the figure 6 in which the last two LEDs of the first block are coloured in blue. This functionality can be achieved using the instruction shown in the figure 10 in which is important to remark that the "index" parameter accepts range of indexes. It is also important to mention that playing with more than one color within a physical block can cause mixture of colors as it is possible to observe in the LED instruction result in where the combination of red and blue is viewed as purple.

Figure 11: Experiment structure

```
{
    "experiment": {
        "totalTime": 30,
        "repeat": true,
        "clean": true,
        "states": [
            {
                "time": 2,
                "arena": {...}
            }
        ]
    }
}
```

Finally running an experiment uses the following structure is used, where "totalTime" represents the time in seconds in which the experiment is executed if the "repeat" parameter is true. The "clean" parameters cleans (no color) the arena at the end of the experiment if its set value is true. And inside "states" any arena configuration can be done, additionally a time for each arena state has to be set. Such "time" parameter is the time in seconds a state must last. One important remark is that if the "repeat" parameter is set to false the experiment will execute the corresponding states until reach the last one.

6 Conclusions

One can observe that at the time of defining a language the expressiveness of such a language is a critical aspect to take into account during the design. During this project we have reached the definition of the language through many meetings with the IRIDIA's team which are the principal stakeholders in this project. We can also observe that to build a system initially one can have a variety of tools to work with and we have to discard some of them, that doesn't mean that the selected ones are the best ones in general but are the ones that better fits to the objectives of the project. All the decisions must be taken for the good of the project and not forgetting the final user which is the one that has to feel comfortable with implemented system. So it is important to spend time thinking about the critical aspects of the systems and having frequent meetings with the stakeholders in order to end up with a useful system for the final users that fits with their necessity.

References

[1] A. Brutschy, L. Garattoni, M. Brambilla, G. Francesca, G. Pini, M. Dorigo, and M. Birattari, "The TAM: abstracting complex tasks in swarm robotics research," vol. 9, no. 1, pp. 1–22.

[2] "Swarmbots project summary." http://www.swarm-bots.org/. Accessed: 2018-06-01.