Dynamic lighting arena for swarm robotics

Technical documentation

Keneth Ubeda Arriaza
Computer Science and Engineering, Université Libre de Bruxelles

I. Hardware architecture

THe hardware architecture consists essentially in an Arduino and blocks of LEDs extracted from a LED strip APA102 the figure below shows an overview of the hardware architecture.

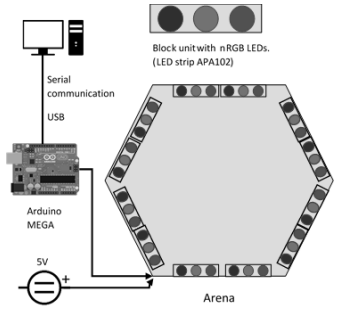


Fig. 1. Hardware architecture.

A. LED strip APA102

Blocks are constructed using this LED strip. To be more precisely each block constructed consists in 12 RGB LEDs extracted from the LED strip. The first block is connected to the Arduino and the other blocks that compound the arena are connected in series using an external power source of 5 Volts.

TABLE I
LED STRIP APA102

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
7	**current**	24.5 mA / RGB LED

B. Arduino MEGA

The Arduino is connected to the computer via USB in order to have a serial communication and change the incumbent information.The first block of the arena is connected directly to Arduino using the following inputs:

- Data: input 51
- Clock: input 53
- Ground: gnd

The recommended baud rate is 57600 and a pause between execute each instruction recommended is 0.0625 seconds.

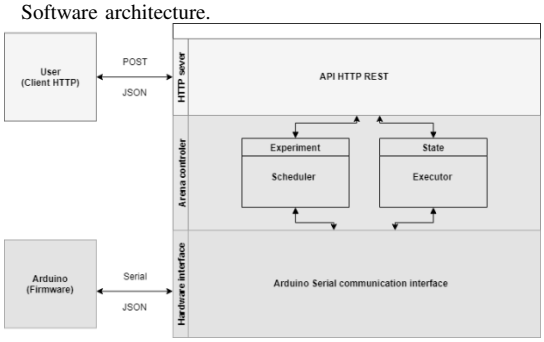
TABLE II
ARDUINO MEGA SPECIFICATIONS

Microcontroller	ATmega2560
Operating Voltage	5V
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

II. Software architecture

The software architecture is divided in 3 layers: the HTTP API, the arena controller and the hardware interface which have been implemented i Python 3.6.4. Additionally a firmware for Arduino has also been developed. The figure below shows a general view of the software architecture.

Fig. 2. Software architecture.



A. Arduino firmware

This file contains the code to handle a LED strip APA102 using the FastLED library and ArduinoJson. The Arduino essencialy receives a JSON string in which the instruction is encoded, via its serial port. Each instruction received represents a block that consits in a fixed amount of LEDs. Using an Arduino MEGA is possible to manage up to 960 LED strip where Each LED uses 3 bytes of memory.

Module name: ledstriphandler.ino.

Dependencies: FastLED, ArduinoJson.

setup(): setting up the baud rate for the serial communication, the input pins, brightness and the array of LEDs to handle. It is important to mention that depending

on the LED strip used the order of the RGB configuration can change, which for this project is actually BGR.

serialEvent(): This is method receives the serial string which is parsed firstly as a JSON to subsequently be interpreted and executed using the FastLED library. Each time the method execute the instruction, a messaege to the serial port is printed as an aknowledge of the request the same if any error ocurrs.

B. Hardware Interface

This module is going to be used as an Interface bewtween the Arduino and the top layer application which is in charge of manage the LED strip. The serial module is used to make the serial connection and the sleep from the time module to enhace it.

Module name: ArduinoInstruction.py.

Dependencies: pyserial.

start_connection() function: starts the serial connection with the Arduino.

send_instrunction(instruction): This sends the instruction to the Arduino. The parameter is string which contains a JSON.

C. Arena controller

This is the fundamental module which parses and interprets the states and expe- riments that are comming from the http requests. This module utilises an sched- uler in order schedule the different states within an experiment. If there are task scheduled but an simple state execution is received then all the scheduled task are canceled. The module contains the implementation for the differente level of control offered by the language definiton.

Module name: experimentctrl.py.

Dependencies: json, copy, sched, time, asyncio and threading.

runState(state): receives the request from the API, receives a Dictionary containing the state configuration.

runExperiment(experiment): receives the request from the API. receives a Dictionary containing the experiment configuration.

generateArdInsForArena(arena): receives the arena object which contains the color configuration. This function executes the arena configuration and its Edge, Block and lEDs instructions.

generateArdInsForEdge(edge, arena, aIns): Receives three parameters: The edge object which contains the color configuration, The arena object which contains the general configuration, The serial connection to send the instructions to Arduino. This function executes the Edge configration and its Block and lEDs instructions.

generateArdInsForBlock(block, arena, aIns): Receives three parameters: The block object which contains the color configuration, The arena object which contains the general configuration, The serial connection to send the instructions to Arduino. This function executes the Block configuration and its lEDs instructions.

generateArdInsForLed(led, arena, aIns): Receives three parameters: The Led object which contains the color configuration, The arena object which contains the general configuration, The serial connection to send the instructions to Arduino. This function executes the LEDs configuration.

fromRelPosToAbsPos(edIdx, bckPerEd, bckIdx, space): Receives four parameters: Index of edge in which the idnex to transform is, Number of blocks that form an edge, Index to transform, The space within the index should be. This function transform the relative position of an index using the relative index, the number of blocks per edge, the index block and the space in which this index should be. The function accepts negative indexes.

fromNegToPosEq(space, number): Receives two parameters: The negative number to transform, The space within the index should be. This function transform a negative number to its positive equivalent.

rangeOrSingleEdge(edges, arena, aIns, fromArena): Receives four parameters: An array of edges which lenght can be 0, 1 or more, The arena object which contains the general configuration, The serial connection to send the instructions to Arduino, Specify if the range or single edge comes from the arena enviroment. This function execute a range of Edges or a single Edge.

rangeOrSingleBlock(blocks, arena, aIns, fromArena): Receives four parameters: An array of blocks which lenght can be 0, 1 or more, The arena object which contains the general configuration, The serial connection to send the instructions to Arduino, Specify if the range or single edge comes from the arena enviroment. This function execute a range of Blocks or a single one.

rangeOrSingleLed(leds, arena, aIns, fromArena): Receives four parameters: An array of leds which lenght can be 0, 1 or more, The arena object which contains the general configuration, The serial connection to send the instructions to Arduino, Specify if the range or single edge comes from the arena enviroment. This function execute a range of Leds instructions or a single one.

rangeToList(rng): receives a Python range. and returns a List which contains the respective range values. This function

transform a python range to a list of values.

D. API HTTP

This module uses the aiohttp library in order to create a web server which will expose the service to run experiments and states. The asyncio is also used to achivied asynchronous services and the negociation between the client and the server is done via JSON messages.

Module name: apiserver.py.

Dependencies: aiohttp, asyncio, json, time, argparse.

runState(request): receives State to execute according to the defined language. returns a JSON to confirm the reception of the State. runState service is a HTTP POST request described in the following table

TABLE III
STATE SERVICE

Name	Execute State
URL	http://localhost:8080/arena-handler/api/v1.0/state
Method	POST
Content type	application/json
Response	application/json

runExperiment(request): receives Set of states and time for the corresponding experiment. according to the defined language. JSON to confirm the reception of the experiment. runExperiment service is a HTTP POST request described in the following table

TABLE IV
EXPERIMENT SERVICE

Name	Execute Experiment
URL	http://localhost:8080/arena-handler/api/v1.0/experiment
Method	POST
Content type	application/json
Response	application/json

III. DEPLOYMENT

In order to set the serial port in which the Arduino is connected and the baud rate for the serial communication a config.json file is found in the projects folder. Once this file has been modified according to the current environment the deployment can be done using the following commands:

apiserver.py [–port] [–host]

host means the host where you want to bind the web server and port means the port where you want to bind the web server. The command can change depending if you work on Linux or Windows so one can deploy it as follows:

Windows:

python apiserver.py –port=8080 –host=localhost

Linux:

./apiserver.py –port=8080 –host=localhost

Note: By default the host and port are 0.0.0.0, 8080 respectively, however it is possible to change it using the port and host argument at the time of execute the command. Also the ”python” command corresponds to the version 3.6.4 of Python.

One final remark is that the colors accepted it is defined in the Color.py file inside the project, it is possible to add more color but at this point the following list comes by default.

TABLE V
DEFAULT COLORS

Color	**RGB Representation**
NONE	0,0,0
RED	255,0,0
GREEN	0,255,0
BLUE	0,0,255
YELLOW	255,255,0
WHITE	255,255,255
OMIT	-1,-1,-1

Note: the omit color simply omit this instruction while the none set the LEDs into black which is equivalent to turn off the LEDs. The colors can be written either lower and upper case.