Real-Time Cooperative Decentralised Control of Illuminance Systems

ALMEIDA, João*
joaotiago99@gmail.com
IST id: 90119

ROSA, Daniel†
dgd.rosa@gmail.com
IST id: 90041

VIEGAS, Guilherme‡
ggmviegas@gmail.com
IST id: 90090

February 19, 2021 Group 1

***Abstract*—The project consists on a real-time cooperative decentralised control of an illuminance system with several luminaires, representing an open working space. Each luminaire is composed by one LED (light-emitting diode), one LDR (light dependent resistor), one Arduino and one CAN-BUS. The luminaires in the network are connected by a CAN-BUS system which is used to make a collective communication. We implemented an intelligent lighting system for the distribution illuminance control, considering not only the value of one's luminaire but also its neighbourhoods. The formulated problem is based in the Consensus algorithm, that takes into account the power consumption of each luminaire and the minimum luminance when the desk is occupied or unoccupied. The algorithm evaluates the influence of all luminaires on each other and renders the best reference value for each node in the network in order to save as much energy as possible, fulfilling the constrains. In addition, we implemented an asynchronous TCP and UDP server that can be hosted by a Raspberry Pi, that communicates with an Arduino through Serial. Bearing in mind that the server can hold many asynchronous TCP and UDP clients, we defined two rules: 1) UDP connection is used to transfer large amounts of data (e.g. information of the last minute and live streaming), and a TCP connection for more precise commands (e.g. set or get occupancy at one desk); 2) multi-threading server to take the most advantage of the quad-core CPU that the hardware of the server has;.**

***Keywords*—** Illuminance Network, CAN-BUS System, Consensus Algorithm, Energy Minimisation, Asynchronous TCP and UDP Server and Client, Multi-Threading Server, Serial Communications.

I. Introduction‡

From the early days of energy production and consumption, people have tried to reduce the cost and waste of energy use, not only for economical and market competition reasons, but also for sustainability purposes. One way that was found was by creating intelligent luminaires that can not only adapt to real time changes in ambient light but also to people needs and wills. Although easy to make, these products do not consider their usage in a larger network, for example a school or a company, where many light sources are spread across the same room and where each light source as impact on all its neighbours. This report aims to solve those kinds of problems, where the utilisation of a real-time and decentralised control of an entire luminaire grid brings major improvements on the power consumed, while at the same time meeting the requirements for specific rooms or even desks.

A box simulates the office, and three *LED and LDR circuits* each with an Arduino are used to simulate three luminaires. A CAN-BUS system, using SPI communication, is used for the different inner-grid communications and a Raspberry Pi is connected to the system via *Serial*, implementing a server for displaying data and metrics useful for the users and for setting certain system parameters like occupancy, *LUX* lower bounds and cost of specific luminaires. The final goal is twofold: 1) to develop a controller that reacts, in real time, to disturbances and take into account the other luminaires in order to reduce the overall power consumption in a decentralised way; 2) to implement a server for displaying metrics and controlling certain aspects of the system and the implementation of external connections from the server to a client via *TCP*(transmission control protocol) and *UDP*(data stream service) network protocols for more than one client at the same time.

The project is divided into two main stages, with the first one only considering one luminaire, with the goal to develop a PI controller and a feedforward term, in a closed loop, capable of driving the system to a given reference value. In the second stage, the main goals are to develop a cooperative control system between more than one node to solve an optimisation problem for working reference solutions in order to minimise energy consumption and the development of the TCP/UDP asynchronous PC application.

In chapter II some previously needed concepts for the project development are described. In III it is described similar implementations. In IV it is explained some specifics of the project run and some difficulties and drawbacks on the project. The chapter V explains the implementation details and theory. VI shows the different experiments and results obtained to consolidate the chosen implementation, discussed also in Chapter VIII, conclusions, where it is shown some improvements that can be made too. Chapter VII is related with the work division for this project development.

II. Background/Concepts*†‡

In order to understand this project, the reader should have experience with first order circuits with a capacitor and resistance hence it is a crucial element in the illuminance system. In sptite of having basic knowledge of the communication via CAN-BUS between arduinos, we recommend to deepen your knowledge reading the Arduino MCP2515 CAN interface library [1]. In addition we recommend to take a glimpse at Serial Peripheral Interface (SPI) protocol since this allows a synchronous serial communication in the network, although it is mostly used to small distances. Furthermore it is also good to remember how state-machine works, which are very important not only to synchronous communication but also to understand the workflow of the asynchronous server. Both the client and the server are implemented with Boost.Asio [2] so it might be helpful to review same C++ concepts such as classes, lambda functions and pointers.

Notice that throughout this report, to ease the line of reasoning, it is used sometimes the terms luminaire A, B and C and in others instances luminaire 1, 2 and 3. They are respectively the same!

III. State of the art†

There is an article that was found with a similar implementation. The document [3] portrays a solution for the problem of illumination control in a networked lighting system wherein luminaires have local sensing and actuation capabilities, where it is used a distributed

control algorithm. However, the authors consider that the luminaires only communicate with a set of neighbours and the parameters of the controller are computed in the optimisation problem, which differs from the approach of this project. In addition not only the local controller of each luminaire is designed individually, but also each source can have different costs.

Due to the pandemic time when this project was made, we were not working together and so we have not done more commands than the ones from the assignment. However we are proud to say that our server is versatile enough to easily add incorporate commands for example: streaming the power and energy used in the system.

IV. Ease of use*

To use the developed system, despite the hardware equipment, the execution of the server is run after one node establish a connection with the computer. At this point, we run the executable of the client that asynchronously waits for a TCP connection and joins to a UDP connection automatically. In order to use them remotely it is also needed to do port mapping in the router of the server to both UDP and TCP ports, for the specific project defined ports, respectively 18701 and 18700. The system is super efficient since it only needs to be executed once at each computer, resulting in a stable connection and avoiding some mismatch related with inputs. Regarding the effectiveness of the system, it misses a few CAN-BUS and arduino messages however it quickly adapts due to implementation of acknowledge messages, which makes the gap time between one instruction and its response longer. Speaking about the engaging of the system, the use of it will be appreciated due to the fact that it successfully meets the goal. Bearing in mind that this system was only tested with 3 nodes, and one server (Raspberry Pi 3 model B), so it might present some errors when enlarging the network or using different components. To conclude, it is easy to learn in a client point of view, since there is a list of commands and its descriptions in the program (type *'Comds'* in client console).

V. Development*†‡

A. System Architecture

The system is composed by three nodes that are comprised by an arduino, a LED, a light dependent resistor (LDR) to read the illuminance and a set of components (resistors, capacitor, etc), all connected using a breadboard and wires. The luminaire can be divided into two circuits, one in charge of the LED's control and the other is responsible for the reading of the illuminance. In the LED driving circuit, in figure 1, a PWM output of the arduino is connected to the LED's positive and the negative pin is connected by a 100Ω resistor to the ground. The PWM output pin controls the illuminance of the LED by changing the duty cycle of the output signal.

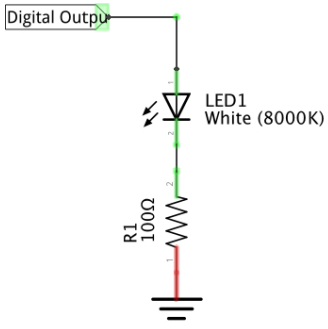


Fig. 1. LED driving circuit

The circuit for reading the illuminance is composed by a voltage divider with an LDR and a resistor of $10k\Omega$. With the objective of reducing the noise on the circuit, a capacitor of $1\mu F$ is used, which connects the LDR to the ground to remove the AC component of the signal. In figure 2 it is represented the illuminance reading circuit's scheme.

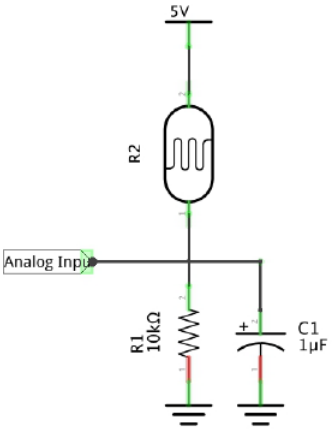


Fig. 2. Circuit to read the illuminance

Each node can send and receive messages from the others, while being connected to a Raspberry Pi working as a server, which communicates with the whole network.

B. System Description

The system has as objective to control the illuminance of an office, by creating a real-time cooperative distributed control system. As explained in the previous section, the PWM output signal controls the intensity of the LED's illuminance. This PWM is an integer between 0 and 255, therefore a PWM of 0 has a duty cycle of 0% and a PWM of 255 has a duty cycle of 100%. For the illuminance reading, the sensor (LDR) is connected to an analog input of the Arduino that measures the voltage in the $10k\Omega$ resistor. Notice that the voltage is read and stored using a 10 bits register, so this voltage is converted by the Arduino into an integer between 0 and 1023, where the supply voltage of the circuit, 5V, corresponds to the value of 1023.

C. System Model

In this section, the system is going to be described by mathematical expressions to model it. Firstly, the arduino's analog input requires to be converted into voltage, as explained previously. So since the arduino maps the voltages from 0 to 5V through integer values between 0 and 1023, the conversion of the analog input to voltage is

$$V_{R1} = \frac{5 \cdot AnalogInput}{1023}. \tag{1}$$

Now, analysing the circuit in figure 2 and applying the Kirchoff's voltage law (KVL), the voltage on LDR is

$$V_{LDR} = 5 - V_{R1}, \tag{2}$$

notice that the current in the resistor R_1 is the same as the current on the LDR, in DC, so the resistance of the LDR is given by

$$R_{LDR} = \frac{V_{LDR}}{I} = R_1 \frac{5 - V_{R1}}{V_{R1}}. \tag{3}$$

With the expression of LDR's resistance, the illuminance can be calculated. Although the relation between the illuminance measured and the LDR's resistance is not linear, in the datasheet [4] it can be seen that relation is logarithmic. In spite of being a logarithmic relation, using a logarithmic scale, it can be determined a straight line inside of the characteristic curve of the LDR. Now, converting it to a linear scale, the LDR's characteristic is represented by

$$log_{10}(R_{LDR}) = m\ log_{10}(L) + b, \tag{4}$$

where L is the illuminance measured by the sensor, m is the slope of the characteristic curve in the logarithmic scale and b is the interception of the characteristic curve with the resistance axis. From (4), the illuminance value is

$$L = 10^{\left(\frac{log_{10}(R_{LDR})-b}{m}\right)}. \tag{5}$$

The m and b are unique parameters of each LDR. Firstly, these parameters were set so that the linear characteristic of each LDR were in the datasheet threshold. To validate those values, the relation between the illuminance value measured by LDR and the LED actuation given by the PWM duty cycle was obtained. Now admitting a first order equation given by

$$G(x) = \frac{K_0(x)}{1 + s\tau(x)}, \tag{6}$$

$$L = Kd + o. \tag{7}$$

With the objective of validating the values of m and b, it was obtained the illuminance for different values of PWM for each luminaire.

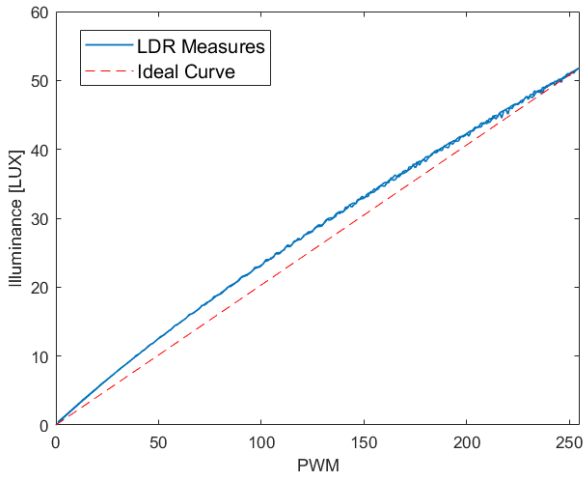


Fig. 3. Steady state characteristics of one LDR

In figure 3, it is represented the values of illuminance for different values of PWM of one LDR (note that it only shows values for one LDR because the behaviour is similar for every LDR). However, the relation between the illuminance and PWM is not linear, consequently the values of m and b were required to be adjusted manually.

The figure 4 shows the linear relation between illuminance and PWM for one LDR, after m and b being adjusted. The new values of m and b are represented in table I for each LDR (A, B and C). Although the values were not accurate to the reality, this linear behaviour is crucial to solve the problem.

TABLE I
NEW VALUES OF m AND b FOR EACH LDR

	m	b
A	-0.678	4.7593
B	-0.576	4.1577
C	-0.668	4.6535

To test the response of the system, it was generated a *stair-like* input. In the figure 5 is represented the response of the system, in particular the LDR's voltage, to that type of input. Analysing it, one can conclude that the system is like the first order system in (6) where x is the target's illuminance.

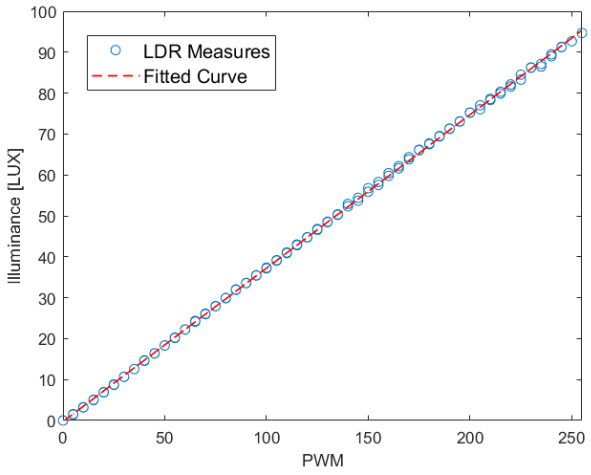


Fig. 4. Steady state characteristic of one LDR after calibration

The circuit built to read the illuminance in figure 2 has a capacitor that charges and discharges according to the voltage in the LDR. This capacitor is responsible for the response of this first order system. The time constant in (6), τ, of an *RC* circuit (with a resistance and a capacitor) is

$$\tau(x) = R(x)C, \tag{8}$$

where C is the capacity of the capacitor, R is the total resistance of the parallel of the resistor R_1 and R_{LDR} and x is the illuminance read by the LDR. Replacing (9) on (8), the theoretical time constant can be given by (10).

$$R(x) = R_1//R_{LDR}(x) = \frac{R_1 R_{LDR}(x)}{R_1 + R_{LDR}} \tag{9}$$

$$\tau(x) = \frac{R_1 R_{LDR}(x)}{R_1 + R_{LDR}} C. \tag{10}$$

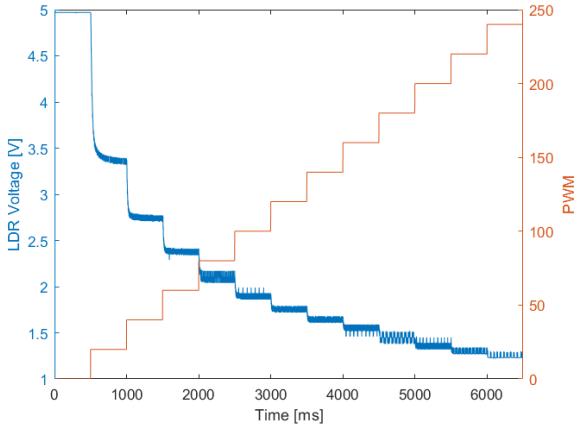
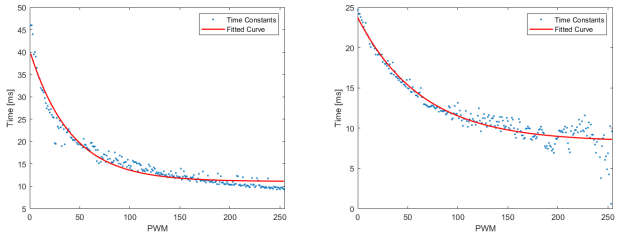


Fig. 5. LDR's voltage for a stair-like input

However, in reality, the time constant is not always precise using (10). As consequence of this fact, the way used to calculate the time constants was to put a step signal as input, for different values of PWM, and then wait for the response to stabilise. Finally, the time constant corresponds to the instant of time when the response reaches 63% of the interval between the final value and the initial value, for positive steps (37% for negative steps).

In figure 6 are represented the time constants calculated for positive and negative steps in function of the PWM value. Looking through

the time constants values, one can see that it is possible to find an equation of a curve that fits the data.



(a) sub caption 1 (b) sub caption 2

Fig. 6. Time Constant in function of the PWM value

Theses time constants were computed for each luminaire in the system. In (11), it is represented the expression for all the relations between time constants and PWM values, for negative and positive steps, meanwhile in the table II are represented all the parameters for the time constants of each node in milliseconds.

$$\begin{cases} \tau_{negative}(x) = a_n e^{-b_n x} + c_n \\ \tau_{positive}(x) = a_p e^{-b_p x} + c_p \end{cases} \tag{11}$$

TABLE II
PARAMETERS FOR τ EXPRESSIONS

	a_n	b_n	c_n	a_p	b_p	c_p
A	16.37	-0.02	7.27	28.81	-0.02	9.53
B	14.52	-0.01	7.39	15.05	-0.02	5.91
C	15.40	-0.02	8.31	29.21	-0.02	11.09

The gain of the overall system is defined as the relation between the illuminance, in LUX, and the PWM value. The previous results can be converted from voltage to LUX, manipulating (3) and (5).

D. Simulator

THE simulator is a tool that calculates the true output illuminance value when a reference for the illuminance is set.

The response of the first order system to a step signal in the instant time t is

$$v(t) = v_f - (v_f - v_i)e^{-\frac{t-t_i}{\tau(x)}}, \tag{12}$$

where t_i is the time when the system receives the step signal, $\tau(x)$ is the time constant that differs according to the illuminance read, v_f is the voltage value corresponding to the reference illuminance value and v_i is the initial voltage of the step signal. In order to calculate both voltages, (5) is manipulated

$$R_{LDR} = 10^{mlog_{10}(L)+b}, \tag{13}$$

as well as (3)

$$V = \frac{5}{1 + \frac{R_{LDR}}{R_1}}. \tag{14}$$

E. Feedfoward Controller

THE main goal for the **Feedforward** controller is to drive the system, in a relatively fast time, to an initial reference value, in open mesh which, because it does not account for real-time external disturbances, will be different than the final reference. The **Feedforward** term is computed based on the gains vector and offset, described in a simplistic way in (15).

$$FF = \frac{L - o}{K} \tag{15}$$

With FF being the **Feedforward** term, L the pretended illuminance, o being the offset and K being the specific gain for that node (K_{ii}). Although it does not react to disturbances, it is still a very important term because it drives the system to a more or less desired value with a relatively fast response.

F. Feedback Controller

THE Feedback controller enables a correction to the value of the Feedforward term, driving the system to the desired reference, based on the difference between the output and the reference values. Although having some downsides, like adding some delay to the system response, it enables the system to react to external, real-time, disturbances. A common type of feedback controller is the PID controller (Proportional-Integral-Derivative), described in (16), which, in this project case, was only used the Proportional and the Integral term, because although the Derivative term enables an anticipated correction to the error, reducing the overshoot, it is very sensitive to the error.

$$u(t) = \underbrace{k_p e(t)}_{\text{Proportional}} \underbrace{- K_i \int_0^t e(t')\,dt'}_{\text{Integral}} + \underbrace{K_d \frac{de(t)}{dt}}_{\text{Derivative}} \tag{16}$$

This controller can now be divide into its three main parts, namely:

1) *Proportional*: This controller has the ability to give a fast response to the system by amplifying the error through a constant k_d. However, it also increases the overshoot and frequency of oscillations.

2) *Integral*: The Integral adds an accumulated error, during time, to the input. It is usually used with other components like the proportional term, because even if the instantaneous error added by the proportional term is not enough to drive to the correct reference value, this integral term adds a cumulative error that may push it to the wanted reference. A big problem with this controller is the fact that, being cumulative over time, if not well though it may wind up when the system saturates, for example when there is more light entering the box than the one the luminaires can diminish this controller will always be accumulating over time being slow to drive the system to the correct reference when the excess light disappears.

3) *Derivative*: The derivative term has the ability to predict the system evolution, having the capability of making it evolve faster and reducing its overshoot, but has the big problem of being very sensitive to noise.

The specific controller for this project is the total sum, u, of a *proportional* term and an *integral* term as seen in (17), with e being the error between the system's output and the desired value, k_p and k_i the respective proportional and integral constants and T the sampling time due to the need for a discrete-time controller.

$$\begin{cases} u = p + i, \\ p = k_p e, \\ i = i_{prev} + k_i \frac{T}{2}(e + e_{prev}) \end{cases} \tag{17}$$

To denote here the use of a *Tustin* method in the integral term because this trapezoidal approximation was thought as a best estimate for the signal integral.

Because of some of the already seen downsides of the use of a PI controller, seen for example on system speed, flicker response and windup effect, some solutions added to the simple PI controller were made.

- **Anti-Windup:** With the ability to detect when the control signal saturates, it compensates the error growth by blocking it from growing too much. The control signal for this anti-windup effect is computed by $u_{awp} = u_{sat} - u$, with u being the control signal without anti-windup effect and $u_{sat} = -u_{ff}$ for a downward control signal trend and $u_{sat} = 255 - u_{ff}$ for an upward control signal trend.
- **Dead Zone:** Because only 10 bits are used for the analog to digital converter, ranging from 0 to 1023, and having a reach of 5V, the LDR readings are not precise, more specifically

there is only a 5mV precision on readings. This origins a flicker noise on the computations and on the final signal output, which was reduced by implementing a dead zone solution where the logic $VCC/MAX\_DIGITAL == 2 * 5/255$ represents the necessary voltage to increase or decrease at least 1 PWM. For the Lux reference equal or less than 2 Lux, in order to prevent accentuated flicker, the threshold is $2 * VCC/MAX\_DIGITAL$ because for this illuminance, the gap between voltages read for sequential values is bigger so the system is more sensitive. This condition prevents the led from turning on and off quickly which is uncomfortable for Human eye. If the Lux reference is more than 2 Lux, the dead-zone is also applied when the error is greater than $0.5 * VCC/MAX\_DIGITAL$, in order to minimise the error.

G. System Frequency

THE clock of the Arduino runs at 16MHz, and it allows to perform a task at a specifically timed interval regardless of what else is going on in your code. In the system, it is used only the *timer1* that calls the interruption to computed the feedback gain and stream values to the server every 10 milliseconds. Since this implementation is set directly at the registers it is very accurate.

H. Non Distributed System Architecture

THE scheme of the system involves the previous models, is represented in figure 7.

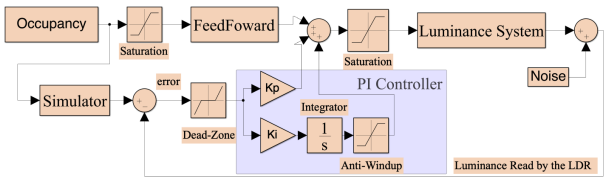


Fig. 7. Non distributed system model

I. Model of the Distributed System

THE implemented network are programmed to hold until 255 nodes since that one byte has that amount of different combination plus the one reserved for broadcast communication. Each luminaire has written its unique values on the EEPROM such as the address on the network, and the linear parameters *m*, *b* and the different τ values. This implementation, rather than a dynamical one, allows a solution that not only is more feasible but also requires fewer computational resources and time consumption since it has a complexity of $\mathcal{O}(1)$.

Bearing in mind that all luminaries work together to achieve minimum consumption of energy on the network hence they have influence on each other. Therefore this relation can be expressed by a matrix that takes into account the influence of every LED in every LDR. Starting from (7) where it is represented the influence of the duty cycle *d* (LED) and the output illuminance *o* at each LDR, now both variables can be rewritten to incorporate the values of all nodes. The new system of equations is given by

$$\begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_N \end{bmatrix} = \begin{bmatrix} K_{1,1} & K_{1,2} & \dots & K_{1,N} \\ K_{2,1} & K_{2,2} & \dots & K_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ K_{N,1} & K_{N,2} & \dots & K_{N,N} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix} + \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_N \end{bmatrix}, \quad (18)$$

where N has a upper bound of 255. The L_n and o_n are the computed illuminance and external illuminance at node n in Lux. The $K_{n,m}$ represents the influence of the illuminance of node m on n. Finally, the d_n is the work effort in PWM, in the range from 0 to 255.

J. Calibration

WHEN the system changes, so the relation between PWM duty-cycle and illuminance, in particular when new nodes enter the network. In order to overcome this problem, the system is calibrated every time it starts and a new node enters the network, therefore it determines the constant gain that transforms PWM in to illuminance values by solving (7) in order to compute K. Although this is an easy task to solve for a lonely node, it might be more challenging to solve for more nodes, because each node needs to solve as many equations as the number of nodes in the network for determining all the values in (18). For that reason a distributed algorithm is implemented, using messages between nodes, with the objective to coordinate all the nodes in the calibration. All the nodes have an integer that counts the number of addresses in the network, but also an array that has all the nodes addresses and the broadcast address (that has value of 0). Furthermore, it is important to notice that not only that the maximum number of nodes is 3, which could be easily increased by dynamically allocating the array, but also that the addresses in the array are sorted in crescent order to facilitate the process.

Firstly, a node that tries to enter the network needs to inform the others that it is trying to join them. In figure 8 is represented the procedure that the new node has. After it sends the *Hello* broadcast message to the other nodes, it waits 2 seconds in order to receive the *olleh* messages from the other nodes. Lastly, it updates and sorts the array of the addresses and also the number of addresses in the system.

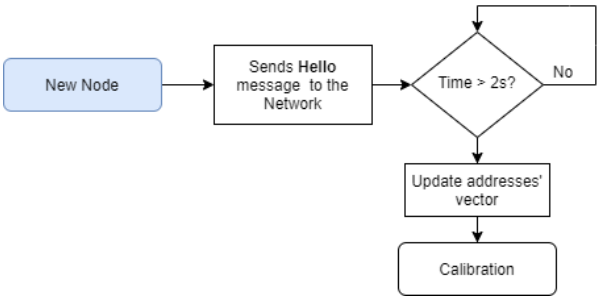


Fig. 8. Scheme for new node procedure

The nodes already in the network has a different procedure represented in figure 9. When it receives a **Hello** message of a new node, it sends an **olleh** message to inform the new node that it is not alone, meanwhile it updates and sorts the addresses' array.

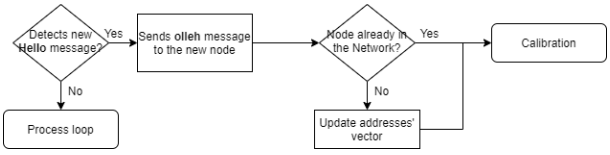


Fig. 9. Scheme for the old nodes procedure

The calibration's fluxogram is represented in figure 10 that shows the algorithm that runs until the process of calibration is finished. Firstly, each LED needs to be turned on once for the others nodes to be able to calculate the gains. When the calibration starts the first address in the sorted array of addresses, excluding the broadcast address, is the *Master*, so it sends a **turn_off_led** message to the other nodes and waits until it receives the **ACK**s (acknowledge) from them. Then it reads the external illuminance (offset) and sends a **read_offset** message for the other nodes and waits for all the **ACK** messages. Finally, it turns on its LED and sends a **read_gain** message to the other LEDs for computing the K value relative to the *Master*, meanwhile the *Master* computes its own K. Lastly, the *Master* turns off its LED and send **your_time_master** message to the next node in the array of addresses, making that node the *Master*. It proceeds

the same way as previous, stated until the turn off the last node of being *Master*, to conclude the calibration. After the calibration, the system starts the consensus algorithm.

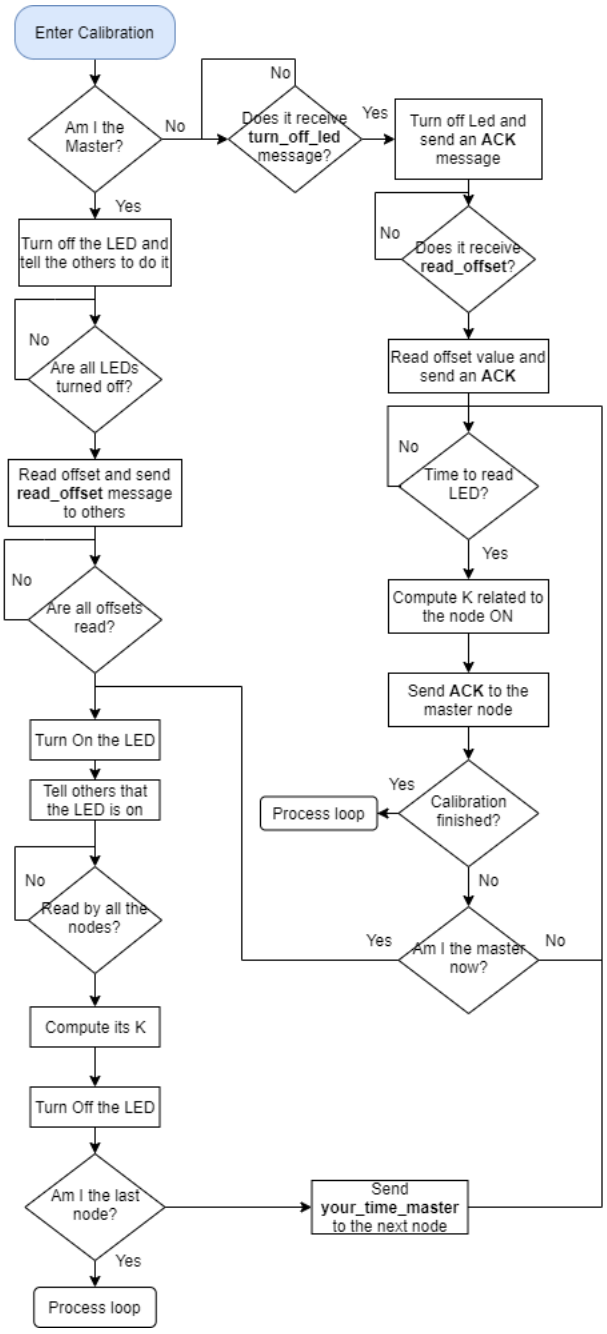


Fig. 10. Scheme of the calibration

K. External Illuminance

THE external illuminance, or the so called offset, is measured at each calibration start and only once, by order of the first node that appears in addresses vector. This works because every node's address vector is sorted the same way.

In this stage, all the LEDs are turned off and each node reads their illuminance value, storing it.

L. Consensus Algorithm

AS referenced in early chapters, each luminaire light impacts the others around it, so a coupling effect exists between all the system luminaires. With this in mind, before turning on each one of them, a process to determine what would be the best reference values, in order to minimise the energy consumed, is needed.

This process is done through a Consensus algorithm using the **ADDM** method (alternated direction method of multipliers), after the calibration stage, where instead of having a single node having to much computations to do, the computational work is spread across all nodes and where all parties, in the end, achieve an agreement on the reference values to use.

For this, each luminaire, having in mind their own constraints, solves its own optimal minimum, without thinking about the other nodes and sends it's own solution after each iteration, updating its own Lagrangian according to the received solutions computed by the other nodes and averaging them. So every node starts, in each iteration, with the same average best solution.

In this specific situation each node minimises its own cost function given by (19)

$$\begin{aligned} \underset{d \in R^N}{\text{minimise}} \quad & c^T d \\ \text{subject to} \quad & Kd + o \geq L \\ & 0 \leq d \leq 100 \end{aligned} \tag{19}$$

where $d \in R^N$ is the vector of duty-cycles (in percentage) of all nodes and $c \in R^N$ is the vector of power costs of each LED, representing the energy consumed by the LED, o is the offset, L the illuminance value pretended for each luminaire and K the vector of coupling gains from the other luminaires to itself. Each node is subject to three constraints, with the first two forcing the duty-cycle value to be between 0 and 100 percent and the third one responsible for the luminaire achieving the defined lower bound, showed in figure 11 for one example node [5].

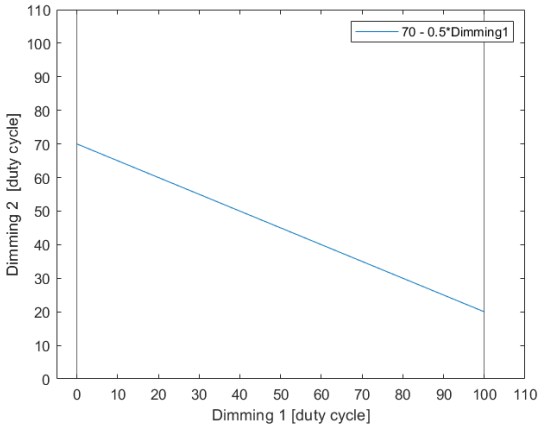


Fig. 11. Constraints for one single node

The **ADDM** method uses an augmented Lagrangian as a cost function for finding the solution and also updates the average solutions vector and the Lagrange multipliers vector, these three steps are described in (20) with ρ being the Augmented Lagrangian penalisation parameter.

$$\begin{cases} d_i(n+1) = \underset{d \in C_i}{\text{argmin}}(c_i^T d_i + y_i^T(n)(d_i - \overline{d_i}(n)) + \frac{\rho}{2}||d_i - \overline{d_i}(n)||_2^2 \\ \overline{d_i}(n+1) = \frac{1}{N}\sum_{j=1}^{N} d_j(n+1) \\ y_i(n+1) = y_i(n) + \rho(d_i(n+1) - \overline{d_i}(n+1)) \end{cases} \tag{20}$$

which as described in [5] origins (21)

$$d_i(n+1) = \underset{d \in C_i}{\text{argmin}} \frac{1}{2}\rho d_i^T d_i - d_i^T z_i(n) \tag{21}$$

with the new cost function being (22)

$$f^{+}(d_i) = \frac{1}{2}\rho d_i^T d_i z_i(n), \\ with z_i(n) = \rho\overline{d_i}(n) - c_i y_i(n) \tag{22}$$

Because it is a convex function, the solution is either inside the feasible region (global minimum) or in its frontiers.

The algorithm first computes the global minimum of the Augmented Lagrangian and checks its feasibility by checking if it fits to its local constraints, if the global minimum fits the constraints the algorithm for that iterations is finished because it is the best solution, if not, the 3 solutions for the constraint boundaries and the 2 solutions for the boundaries intersections are computed, checked for their feasibility and the best one is sent to the other nodes. Finishing the consensus a solution for each node is found, which is used in the **Feedforward** term.

M. Distributed System Architecture

In the distributed system the values for the feedforward controller and for the reference are computed by the consensus algorithm, before the LED's turn on. Therefore, the feedforward controller in the non-distributed scheme is replaced by the consensus routines, with the new distributed system architecture being described in 12. For this implementation, a main code was implemented for the controller inheriting the first stage controller classes and a new class for the consensus was created. The calibration functions are implemented in the main code. The controller makes use of a state machine that prevents it from blocking or stopping in any part of the program run.

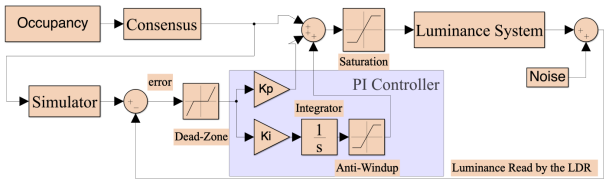


Fig. 12. Distributed system model

N. Client-Server PC Application

The server can be connected to any node via serial connection. When it successfully happens, the server is responsible to acquire data from all nodes in the network. It also makes a bridge to a remote connection with the network, allowing to control and evaluate the performance of the network.

1) Architecture and Concurrency of the Server: Once the server is connected to a micro controller via serial and with other computer via internet communication TCP and UDP, the implementation has to be non blocking. All connections are implemented in C++ language using the Boost.Asio Library [2] and are based examples from the official site and from the set of slides [5]. The figure 13 is the schematic of the architecture of the PC application, involving the most important classes and connections of it.

The asynchronous functions copies the information on the channel to a buffer and then moves to a callback instruction, passing the error flag and the size of the incoming message. Each protocol has its unique characteristics:

a) Serial: Before each message, it is sent the character '+' as a sign that the following bytes contain the message. Every 10 milliseconds the micro controller sends to the server (stream) the information of the duty cycle and illuminance of every node in a 6 bytes message. Therefore the baud rate, that corresponds to the maximum number of bits transmitted per second, is given by

$$Baud\ Rate = N_{micro\ controllers} * message\ size * \\ frequency * byte\ size. \tag{23}$$

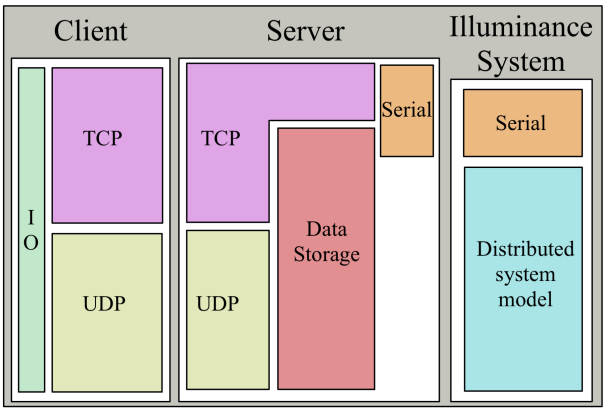


Fig. 13. Architecture of the PC Application

Not taking into account the get and set instructions since its lower frequency comparing to the stream values, the message size is 7 bytes, hence for a network at full capacity the minimum baud rate value is 1428000 bits per second. However in this project the baud rate programmed is 230400 bits/s once the number of arduinos is 3 and therefore the minimum value is 16800 bits/s. This value is the most reliable, once it presents less misreads comparing to higher rates. The serial communication is bidirectional, since the server can set new lower bound values for both occupied and unoccupied state and also change the occupancy and cost. After the micro controller performers the instruction, it uses a ricochet effect to transmitting the result of the operation, with the most significant bit expressing if an error has raised. Finally the C++ class that involves serial communication is the responsible to write the incoming values in the database, once it is the only with that privilege. The TCP and UDP connections can only read values already stored. Finally in order to prevent the Arduino to auto restart after a serial connection, the stimulated current is embezzled from the restart pin to the ground through a $10\mu F$ capacitor (a very important finding for arduino debugging).

b) TCP: The TCP implementation allows many clients to use the network independently, because each one has an unique bind socket. The transmission control protocol (TCP) can be seen in [5]. Nevertheless its slow sequential data transfer, guarantees that each message is correctly sent and received due to its acknowledge messages after an unsuccess, partial success or a successful transmission. This protocol is implemented for the get and set instructions due to their importance regarding not only the control of the network but also because of its short message size. The get instructions correspond to the ones that the server has already the pretended information, although the set instructions involves asking information to the micro controller. Regarding this last set of commands, the server has a queue of 'waiting for acknowledge instructions' that prevents from duplicated instructions and assures that the response containing whether acknowledge messages or values is sent to the right client. This particular implementation is based on an asynchronous operation involving a timer. There are three vectors (sockets' address (id), command, micro controller's response), and each index corresponds to a different valid command. Every 1500 milliseconds the server checks if the client has a message on the id's vector and if so, checks if a new value is available (micro controller's response is different from the initialisation). This is possible because of the ricochet effect since the server and the micro controller reach an agreement about the message thread. The function used to send data is $tcp :: socket :: async\_send()$, to receive $tcp :: socket :: async\_read\_some()$ and to manage the timer it is used the typedef $boost :: asio :: steady\_timer$.

c) UDP: On the other hand, there is also an implementation of an UDP server for the operation of streaming the duty cycle and illuminance of a node every 10ms, and also to receive the last minute buffer information (30KB - 4 bytes for the float value and newline).

This protocol allows a faster data transfer with packets delivered in flow, however its integrity in receiving messages is lower than TCP. Despite the fact that for this project the amount of transferred data does not justifies this implementation (in case of a scientific work), the academic purpose of this project does. When the streaming transmission is enabled, every time the server detects a new available value for streaming it sends it, nevertheless the previous values have been received or not. When transferring the last minute buffer is about, the servers copies the newest version of the data to an array as soon as the command is received in order not to lose any data. To sum up this implementation is simultaneously waiting for new messages from the client using ($udp :: socket :: async\_receive\_from()$ - which includes the endpoint of the client) and sending values to the client using ($udp :: socket :: async\_send\_to()$ - specific client). The server can handle multiple UDP instructions because when the UDP server receives a new message, the endpoint associated to it is passed by value later so it is not lost when other connection arrives. Therefore it allows to have countless streams regardless if the command is repeated from other client.

To conclude, the classes that are most relevant for coding the PC interface are displayed in the figure 13, such that the associate class to light green is udp_server, the *communications* is represented in orange, the class where all the values are stored (*office*) in red and finally in purple is represented both classes tcp_server and $new\_connection$. About these last two classes although the TCP connection is assured by tcp_server with the function ($tcp :: acceptor :: async\_accept$), the socket of the client is in the $new\_connection$ in order to have more than one connection simultaneously with this protocol.

2) Architecture and Concurrency of the Client: The client program also runs asynchronously divided in three main tasks. When it is initialised an UDP connection is started at a predefined port (17801), additionally one TCP connection is wanting for an acknowledge from a different endpoint also at a predefined port (17800). When a connection is successfully, the program will start interpret the commands written int the console. Finally when the TCP connection detects that the servers goes down it stops both connections, and shuts down the client.

There is also an option to save in a text file the streamed and last minute values, if it is typed the word "*file*" after the desired command. Besides if it is typed the word "*Comds*" the program will print the list of available commands and for the word "*close*", the program is going to shut down securely.

Regarding the text file, it is created at maximum one file per client, and it has an unique name format of:

$$< command > \; - day : month : year : \\ hour : minute : second.txt$$

Since that the stream contains both measured value and time since last restart, and as the server allows to have many streams at the same time, it is possible to make use of the data for statics or graphics. Nevertheless, some attention might be needed regarding the time matching because of there is no safety that all data is sequentially corrected (UDP).

3) Data Acquisition: The process of acquisition the data from the serial connection consists of decoding the incoming bytes according the message in the first byte. At this point the message is interpreted, however if the message is not recognise the program does not recover some potential bytes.

Although the address of each node is different from the micro controller and server's point of view, the arduino has a decode vector that matches both parts.

4) Data Struct: The data struct of the projects consists of one class containing the general information about the office such as the number of nodes in the network or the time since the last restart of the decentralised system. Moreover this class holds an array of the class *lamp*, where unique information of each node is stored such as:

- State (Occupied or Unoccupied);
- Lower Bound for Occupied State;
- Lower Bound for Unoccupied State;
- Accumulated Energy Consumption;
- Instant Power Consumption;
- Accumulated Visibility error;
- Accumulated Flicker error;
- Number of samples.

Most of this metrics are going to be explain later in this section. Additionally this class also has access to a third class that design particularly for circular buffer management. Instead of be used the boost available struct [6], for academic purpose it was design a Template class that behaves as a circular array that can operate with generic types of data. This implementation consists on having stored the index position of the newest element on the head and th oldest on the tail and follows the golden rule that both index can not be crossed and the only instant that they are equal occurs when the buffer is empty. This implementation can be imagine has the figure 14 shows. The policy of this struct is first in first out (FIFO), which in orders

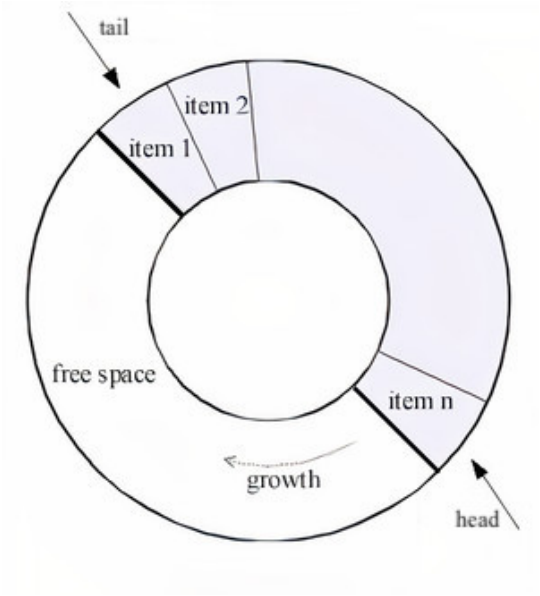


Fig. 14. Circular Buffer

means the oldest element (first of all which had entered in the array) is overwritten when the buffer has no free space. This implementation can be seen at class $circular\_buffer$. This data struct is used by the *lamp* class twice in order to save the illuminance and duty cycle values of the last minute information making a total of 6000 values.

5) Performance metrics: Bearing in mind that the micro controller already has to deal with the decentralised control, the server is responsible to computed all the performance metrics.

Firstly, throughout this section lets consider the following notation: t_n the time instant; T the time since last restart; i the desk; N the number of nodes at the system.

The power at the time instant *t* at desk *i*, is given by the relative power (d - duty cycle between 0 and 1) compared to its nominal power (c - value in Watts) (cost at the consensus algorithm)

$$p_i(t) = c_i(t) \cdot d_i(t), \quad [W]. \tag{24}$$

Consequently, the total power consumption at the system at *t* is

$$P(t) = \sum_{i}^{N} p_i(t), \quad [W]. \tag{25}$$

The energy consumption during the time span $\Delta t \in [t_{n-1}, t_n]$ represents the power consumption during that time span at desk *i*,

$$e_i(\Delta t) = \int_{t-1}^{t} p_i(t)\, dT \quad [J]. \tag{26}$$

Since the duty cycle during consecutive instants is constant, the equation (26) is rewritten as the time span times its respectively power consumption and leads to the total accumulated energy consumption at the system since last restart

$$E = \sum_{i}^{N} \sum_{n}^{T} c_i(t_{n-1}) \cdot d_i(t_{n-1}) \cdot (t_n - t_{n-1}) \quad [J]. \tag{27}$$

The visibility error of the network represents a subjective comfort criteria concerning the periods of illumination below the lower bound value defined by occupied or unoccupied. This metric is defined as the average error between the reference illuminance (L_i) and the measured illuminance (l_i), at desk i, for the periods when $l_i < L_i$. The server computes the accumulate value of this metric for both a single desk and for all system towards the time. They are, respectively, given by

$$v_i(t) = \frac{1}{T} \cdot \sum_{n}^{T} max(0, L_i(t_n) - l_i(t_n)) \quad [Lux], \tag{28}$$

$$V = \sum_{i}^{N} v_i(t) \quad [Lux], \tag{29}$$

Lastly other comfort metric that allows to evaluate the decentralised system performance is the flicker error. When there is a change of the signal of the illuminance towards the time, the accumulated is flicker is updated. Defining the flicker at time t_n as $f(t_n)$

$$f(t_n) = \begin{cases} (|l(t_n) - l(t_{n-1})| + |l(t_n - 1) - l(t_{n-2})|)/(2T_s) \\ \quad if\ (l(t_n) - l(t_{n-1})) + (l(t_n - 1) - l(t_{n-2})) < 0 \\ 0 \quad otherwise, \end{cases} \tag{30}$$

where T_s is the sampling time (10 ms). Finally this metric is also available in the range of operations of the system, and can be obtain both as the accumulated flicker at desk i or at the system since last restart. Both commands are given, respectively, as

$$f_i(t) = \frac{1}{T} \sum_{n}^{T} f_i(t_n) \quad [Lux], \tag{31}$$

$$F = \sum_{i}^{N} f_i(t) \quad [Lux]. \tag{32}$$

Regarding this last metric, that are some parameters that might have influence on it. Firstly, lets take out from the equation the time when the system is in the dead zone because it is not taken into account in the server in order to match the expression in the assignment. Secondly the principal reason of the flicker is concerning the movement of the light that may no be perceptible to an human eye although is detected by the photoresistor plus the white noise. In addition, the PWM signal for the selected pins in arduino has a frequency of 490Hz so is not an harmonic of the 100Hz system's frequency, therefore is a small chance of additional noise. This problem is as more relevant as the network size increases, once the time to compute the feedback gain and the communication in the network makes even harder the synchronism of the nodes when adapting its illuminance. Finally, although it has not been detected throughout the development of the project, there is a slightly chance of occurring flotations on the LDR's voltage supply.

6) Memory Management: The memory management of the program is a complex topic that involves memory allocation, passing values, freeing memory. Starting off with the memory allocation, every time a TCP client joins in a new connection it dynamically creates a new space for that client and adds the socket's address in a vector of the TCP server class. Despite the fact that this information is not used for more than freeing memory when the destructor is called, it could have been used for statics.

Secondly, throughout the program it is always passed the referenced to the objects, avoiding a new copy of the variable hence the program executes faster specially when passing object of large sizes such as the circular buffer. Another advantage is the possibility to change multiple values inside the function. This can be easily in the constructor of the *office* class where it is created a vector of address to a *lamp*. In order to work with different techniques, sometimes it is used smart pointers for example to the circular buffer. This implementation is more elegant since the programmer does not need to concern about freeing it later.

Another important fact is to avoid leaks in the memory, which can diminish the performance of the computer by reducing the amount of available memory. The implementation assures that the program frees the allocated memory, when it catches operating system commands, e.g. Command/Control + C, segmentation fault, preventing a premature exits. The solution used consists on using the native C++ function *signal* [Signal]. This implementation is available for both client and server programs.

In order to test this implementation, there is an option to limit the amount of time that the server is up. Regardless how many clients joins the server and how many closed or opened sessions exits when the time ends, the system presents any leak of memory. It is also a valid state that the program presents the same result when the decentralised system restarts, after had received the proper command, hence it automatically frees the circular buffer (unique pointer) memory when the code is manually deleting each *lamp*.

Finally, once that the micro controller used (Arduino UNO) has only 2KB memory the only necessarily information to the decentralised control is save there and the rest is sent to RPi, which its limit of memory depends on the characteristic of the SD Card.

7) Multithreading: The implemented process consists on creating a defined number of threads that are controlled by the principal thread. For development purposes the server is a Raspberry Pi with a quad core CPU so there the maximum number of simultaneous threads is one of each core therefore 4. In each thread there is run the boost $io\_service$, in other words the server can hold until 4 different communications (TCP, UDP, serial). Practically during the tests there were any notorious delay when the server was asked to hold 4 streams of different clients. On the other hand for streaming the illuminance values and duty cycles for all available nodes - 6, although the server had performed well there was a slightly difference comparing to the previous case.

8) Mutual Exclusion: Bearing in mind that we can hold more than one thread called in the principal process, all the available memory of the program can be access by them. In other orders two or more threads can access to the same information in memory at the same time, so when the server uses the *database*, is entering in a critical region of the program. Therefore it has to have a mechanism to protect itself from collisions that prevents one thread to get a specific information when other thread is accessing it. The solution of this problem is mutual exclusion mechanism in each class of the *database*. The implemented mutexes work like a traffic light, when a thread, lets call thread 1, access to any function of one class, its mutex locks preventing other threads (queue) to access to any data of the class. This process holds until the thread finish its work on that class when the mutex becomes unlocked, and than one thread on the queue can enter in the class and the lock process repeats. Every class of the server has only one mutex, instead of an hierarchy where one mutex has priority over other once in the queue.

9) Port Forwarding: Lastly, it is implemented the possibility to port forward the server. In order to accomplish the demand, the client has defined the endpoint of the TCP or UDP connection with the the public IPv4 of the router that the server is connected. The auxiliary function to this process is $ip :: address :: from\_string(< IPv4 > , port)$ and the port is the virtual place in the computer where the information arrives or is delivered. In addition the router of the server needs to be configured, where a port mapping has to be made. What it means practically is that the client drops a letter (message) in a virtual public point (public IPv4 of the router) encoded (port), and by

mapping the port the router can decode the receiver (server with local ip) of the letter. Additionally, a firewall might have to be disabled regarding this ports.

O. Communication Protocols

THE communications represents an important of this project once it allows the nodes work as a team and being managed from a computer outside the network. There is used two different protocols in the implementation, CAN-BUS system for the communications inside the network and Serial to transmitting values to the server. Despite of the differences between the two protocols, they both use throughout the code a unique representation of a float number. Bearing in mind that a float number is represented by 4 bytes with limit by the order of magnitude of 38. For this project such values are not needed because of the illumimance capacity of the LED. The maximum illuminance in the box does not exceed the 100 Lux when all duty cycles are maximum, although with a flash the photoresistor detects approximately 1000 Lux. Furthermore the precision of the LDR are mainly in the most significant decimal bit, hence the others are more affected by the noise. Therefore it was defined that the decimal part could be represented in 4 bits and the integer part with 12 bits. To conclude with this implementation can be summed up fourfold:

- 2 byte representation (- 2Bytes);
- unsigned values (+1bit);
- integer part in [0 to 4095];
- decimal part in [0 to 9].

In addition, there are also two advantages, one is the four decimal bits from '10' to 'F' can represent code as special messages such as warnings, or specific errors. Moreover, the most significant bit represents, in the response to the set commands, the occurrence of an error in the system. The encode and decode functions are called '$float\_2\_bytes$' and '$bytes\_2\_float$', respectively.

1) Serial: Firstly all valid messages in this channel starts with a delimiter '+' that tells to one endpoint that there is a possible new message. The protocol starts with a synchronism in the server when it sends the greeting message "+RPiG" and waits for the micro controller message in the format "+A<network's size>:)". Later the server allocates the space with memory to store all values and send the message start message "+RPiS", that gives permission to the micro controller to start sending the values of occupancy, lower bound values, the time since last restart and finally it starts to send the illuminance and duty cycle values. To end the conversation the server sends the message "+RPiE". Besides the starting message all the all communication is done asynchronously with the functions $boost :: asio :: read()$ and $boost :: asio :: write()$ from boost [2]. On the arduino, the functions used are $Serial.available()$, $Serial.readBytes()$ and $Serial.write()$. This last function is preferable to $Serial.print$ reads the argument as a byte instead of an ASCII character. There are 9 different messages:

- "+A_:)" - (Arduino) or "+rrrr" (server) - 5 bytes;
- "+slldd" - Streaming values - 7 bytes;
- "+o_b" - Occupancy as a boolean - 5 bytes;
- "+O_lb" - Occupied lower bound - 5 bytes;
- "+U_lb" - Unoccupied lower bound - 5 bytes;
- "+x_ei" - External illuminance - 5 bytes;
- "+r_ic" - Reference illuminance from consensus - 5 bytes;
- "+tslr" - Elapsed time since last restart - 5 bytes;
- "+c_pc" - Power cost - 5 bytes;

where the underscore represents the address of the node from the server's point of view.

2) Canbus: The Canbus' messages are sent in a format of *bytes*. Each message has 4 *bytes*, where the first represents the message's type, the second the sender's address and the last 2 *bytes*, as explained previously, represent a float number. In some cases, there are messages that do not need to send a float number, therefore that last 2 *bytes* are zero.

The following messages can be sent during the calibration process and during the consensus algorithm:

- Type *hello*: when a new node is booting;
- Type *olleh*: when it receives an *hello* message from a new node in order to tell the new node that it is not alone;
- Type *ACK*: when it receives an message, the node send an acknowledge to the sender;
- Type *turn_off_led*: during calibration to tell the others to turn off the led to begin the calibration;
- Type *read_offset_value*: during the calibration to notify the others that the offset can be read;
- Type *read_gain*: during calibration to warn the others that its LED is on and the network can compute the gain referent to this node;
- Type *your_time_master*: sent to the next node to turn on the LED in calibration;
- Type *start_consensus*: to start consensus;
- Type *send_consensus_val*: during the consensus algorithm to send the result of the dimmings computed;
- Type *turn_max_led*: when the client asks an illuminance value superior than the possible, it notifies the others to turn on the LED on maximum illuminance.

However, the nodes also communicate between each other not only to send information to the RPi, but also to receive information from the RPi. These messages are all between the HUB node and the rest of the network, which are:

- *set_occupancy*: to set the receiver node on occupied or unoccupied mode;
- *set_bound_occupied*: to set the occupied bound of the receiver;
- *set_bound_unoccupied*: to set the unoccupied bound of the receiver;
- *set_cost*: to set a new power consumption value on the receiver;
- *request_stream*: request stream of illuminance and duty-cycle;
- *sending_stream_lux*: send the illuminance value for the stream;
- *sending_stream_dimming*: to send the duty-cycle value of the node for the stream;
- *get_reference*: ask for the current reference of the receiver;
- *get_external*: ask for the current external illuminance;
- *sending_reference*: send the current reference;
- *sending_external*: send the current external illuminance;
- *reset*: reset all the nodes variables and goes to the calibration state.

VI. EXPERIMENTAL RESULTS*†‡

A. Circuit Description

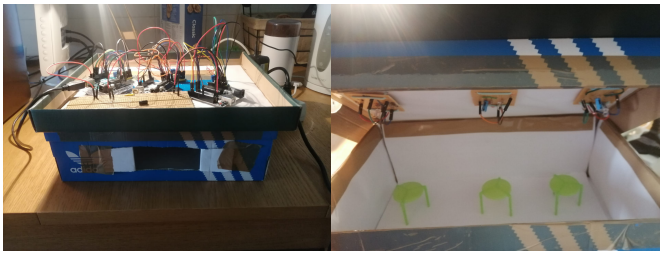
FOR simulating a room or a office, a shoe box was used 15(a) and the luminaires were strapped at the box ceiling to really simulate what would be real office lights 15(b). The overall controller circuits and rpi are stored in a box above the "office" 15(c).

B. Single System

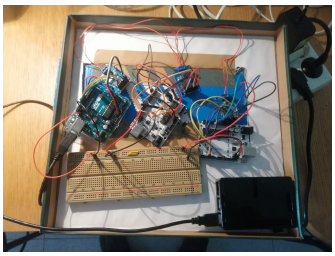
IN the first stage, only one controller was implemented for one luminaire. For each system boot, a new gain and offset is calculated. The gain in one example execution was equal to 0.31 and the offset equal to 0.00, meaning not only that the box is well isolated but also that the computed gains make sense for the calibrated m's and b's previously computed.

1) Feedfoward: The first part tested was the system only with the feedforward controller. As predicted before, the feedforward controller reacts fast to a change of the reference, but it is not very precise.

In figure 16, there are represented the results for 3 different reference values. One can say that the measured illuminance has an identical behaviour as the reference. However, it can not compensate the error between to the measures and the reference since that is the weakness of the feedforward controller. Afterwards, to test the

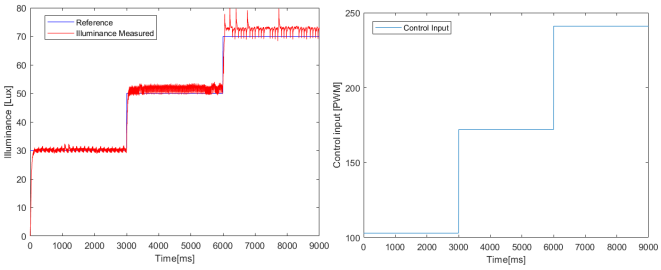


(a) Box exterior (b) Lights inside Box



(c) Circuits above Box

Fig. 15. Office simulation the Box



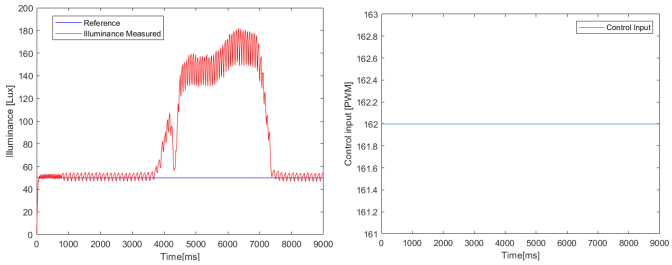
(a) Measures and reference (b) Control input

Fig. 16. Results of the feedforward control

robustness of this controller towards disturbances, it was used a mobile phone's flash.

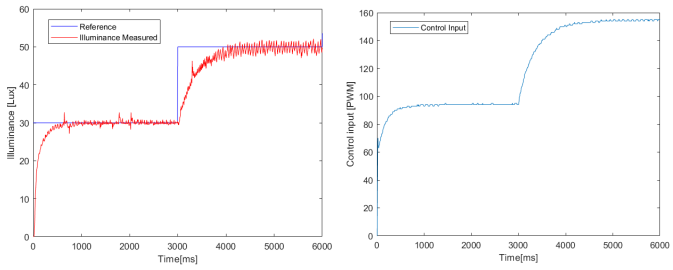
Analysing the figure 17(a), the flash was inside the box in the beginning of the 4^{th} second. It can be seen that towards a disturbance like this flash, the control input in 17(b) does not change, but with the increasing of the illuminance in the box, the input signal in the LED should be decreasing in order to make the illuminance closer to the reference. To conclude, the feedforward alone is not robust for responding to disturbances. It would be a good choice for a system without disturbances, but that is mostly impossible to happen. So to fix this problem, it was used a feedback controller.

2) Feedback: As previously explained, the feedback controller was designed with a proportional part and also an integral part. In the set of figures shown in 18, it is represented the illuminance measured and the reference as also the control input through time, when it was only implemented a feedback controller. Analysing this figure, one can see that the system is slower than the system with the feedforward controller, but it is better following the reference, which has a smaller error.



(a) Measures and reference (b) Control input

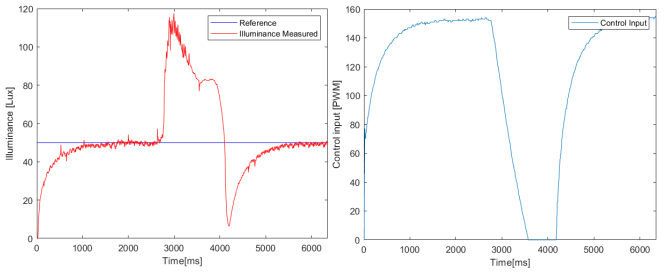
Fig. 17. Results of the feedforward control with a disturbance



(a) Measures and reference (b) Control input

Fig. 18. Results of the feedback control

To test this controller towards a disturbance, as for the feedforward controller, it was used a flash from the mobile phone. In the set of figures shown in 19, it can be seen the the illuminance measured and reference, as also the control input through the time. In the beginning of the 3^{th} second, the flash is turned on and in the 4^{th} second it is turned off. Comparatively to the feedforward controller, the feedback can readjust the input in order to follow the reference through disturbances. Notice that the LED is turned off in some interval of time, because the flash overwhelms the reference.



(a) Measures and reference (b) Control input

Fig. 19. Results of the feedback control with a disturbance

In conclusion, the feedback controller is better following the reference through disturbances than the feedforward controller, however it is slower. So the controller can be improved with using both designs, but also using other extras that will be stated in Improvements.

3) Improvements: The first major imporvement that can be seen at first sight is to join the feedforward controller with the feedback in order to make the system faster. In figure 20, the responses of the system is expected, because with the feedforward controller component the system is faster to react to changes, which is what is wanted!

The next improvement that can be done is implementing some engine that reduces the noise in the measurements. This noise happens due to the fact that the system always tries to react to the error, even for small values. In order to fix that, it is used a margin of 0.1 Lux

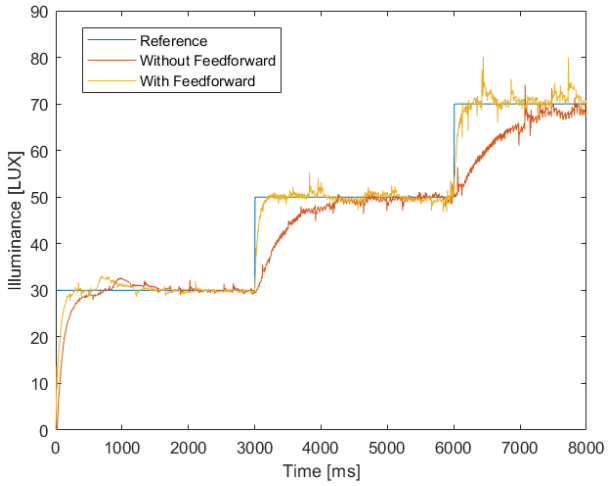
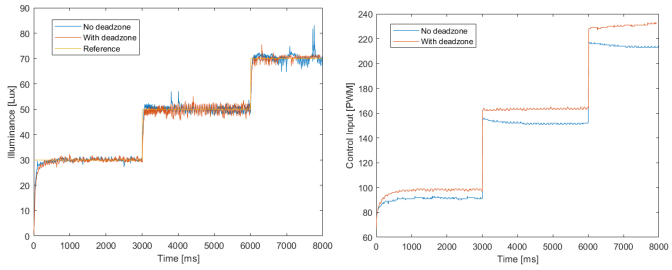


Fig. 20. Difference between the usage or not of feedforward

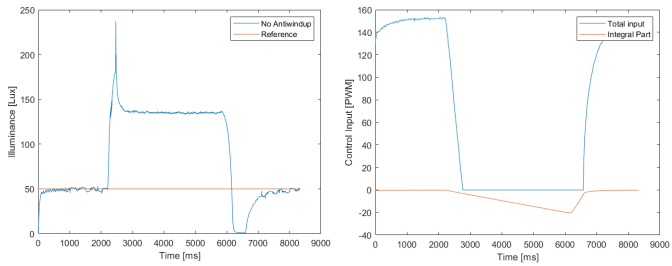
that if the absolute error is equal or less than 0.1 Lux the error is 0 (the deadzone that was explained in the Development Section).



(a) Measures and reference (b) Control input

Fig. 21. Difference between using or not using the deadzone

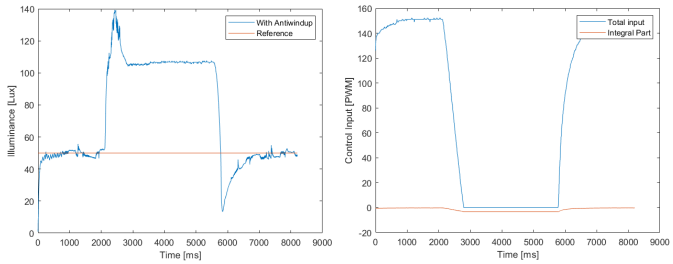
Now, analysing the figure 21(a) one can not see much of a difference as the noise is almost the same. Looking to the figure 21(b), the control input with the deadzone has less changes, so the noise is due the measurements of the LDR and not because of the control input. Therefore this implementation is not that handy in this system.



(a) Measures and reference (b) Control input

Fig. 22. Windup effect

The windup effect happens when the integrator accumulates error values with no limits, which then makes it challeging to get it back to 0. In the set of figures in 22, it is shown the windup effect, after a mobile phone's flash disturbs the system. One can see that after the flash, the system takes a lot of time to get back to the reference, because, in figure 22(b), one can see that the integral term keeps growing and takes some time to go to 0 again. In order to make the system faster, the anti-windup engine, which was explained in section V, is implemented.



(a) Measures and reference (b) Control input

Fig. 23. Results with the anti-windup engine

In the set of figures 23, the response of the system with the anti-windup engine is portrayed, while a flash disturbs the system. One can see that not only the system is way faster with the antiwindup than without it, but also that the integral part does not reach enormous values. In particular, the systems becomes 1 second faster, which is a huge improvement of the controller.

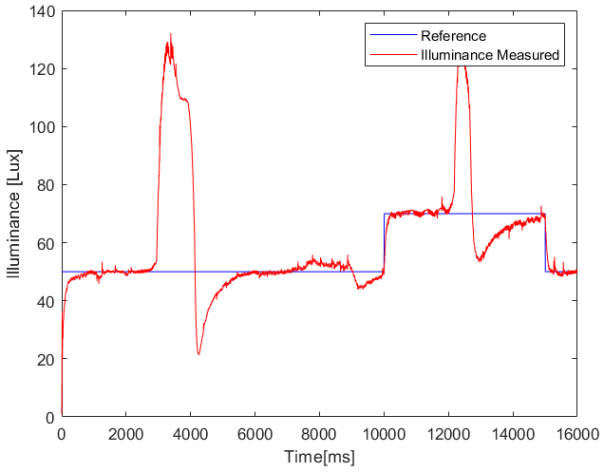


Fig. 24. Results of full controller

To conclude, the controller with all the improvements previously state was tested to reference changes and to disturbances. In figure 24, it is shown the system response.

Additionally, one can say that the controller has a pretty good performance, because it has no overshoot and it is very quick to react to disturbances. So the controller design is completed.

4) Computational Cost: It is important to also check the times that some program routines take to ensure the robustness of the program. A very important time is the sampling time, intended to be 10ms. 25 shows the elapsed time between many interruptions proving a good robustness in the interruption times, being equal to exactly 10ms.

It is also showed in III some elapsed times to compute important subroutines in the program, more specifically the Feedback controller function that takes 1080.4 us, the loop which takes 2277.8us and the use of *Serial Prints*, important for debugging and plotting results, that take more or less 2363.4 us. This proves that each of these main functions take less time than the sampling period of 10ms, satisfying the computational requirements to work without run time errors.

C. Multiple Systems

FOR the second stage the distributed system is now the main focus, using three luminaires. In the calibration stage, a gains matrix, **K**

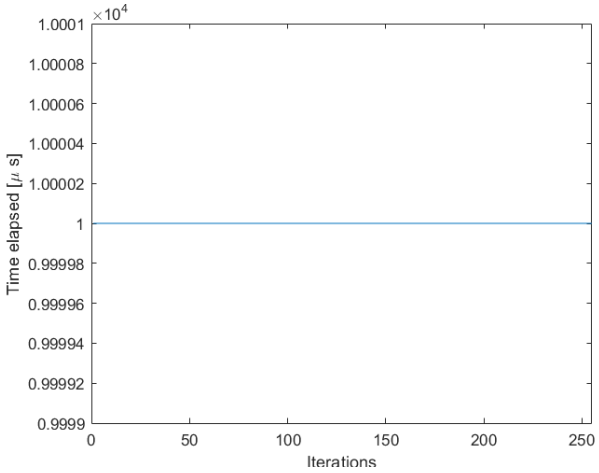
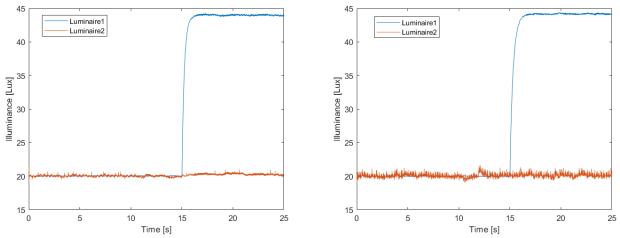


Fig. 25. Time between interruptions

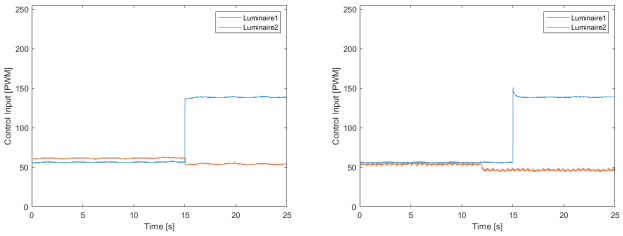
TABLE III
TIME TAKEN FOR DIFFERENT SUB-ROUTINES

Sub-routine	Elapsed Time(us)
Loop	2277.8
Feedback	1080.4
Serial Prints	1280.4
Overall	4641.2



(a) Distributed Control (b) Non Distributed Control

Fig. 26. Luminance evolution towards the time for both types of control.



(a) Distributed Control (b) Non Distributed Control

Fig. 27. Duty cycle evolution towards the time for both types of control

and an offset vector, **b**, are computed, originating the system in (33).

$$\begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} 0.3029 & 0.0646 & 0.0364 \\ 0.0284 & 0.3360 & 0.0619 \\ 0.0252 & 0.0641 & 0.3250 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} + \begin{bmatrix} 0.00 \\ 0.01 \\ 0.00 \end{bmatrix} \quad (33)$$

Just by looking to the diagonal values vs the other values one can conclude that these are expected values because the diagonal values, which represent the impact one led has on itself, are always bigger than the others, which represent the impact I have in the others or the others in myself.

1) Performance of Distributed vs Non Distributed System: Other advantage of the distributed control is that despite they do not have information about the others references, they have reach a decision in consensus. Therefore they will be focus to follow its own reference with the influence of the others in order to fulfil the lower bound value constrain. On the other hand, due to the ignorance of the others' nodes, the next moments is predicted with the actual environment, which are permanently changing so the stability in the non distributed control is difficult to be achieved. The 26 and 27 represents, respectively, the illuminance and duty cycle of the system with two nodes. In addition both nodes started with the unoccupied state (20 Lux), however approximately at 10 seconds the state of the first node was toggle to occupied (42 Lux). Node that the consensus take in average 5 seconds for two nodes. The goal of this section is to compare and point out the differences between a distributed and a non distributed controller with, so the network is reduced to two nodes with even costs, to minimise the possible lost message and maximise the accuracy.

Once again, as it can be seen in the figure 27, is clear the advantage of distributed control, whereas the non distributed contains overshoot in the control input after the chance of occupancy.

In addition, there are advantages specially regarding the energy savings. In the figure 28, from the moment that the system is turned on, the energy consumption with the distributed control is less than the results with the non distributed. This metric is computed according to (27).

Secondly it is also illustrated the accumulated flicker (see (32)) in the figure 29. As it would be expected, due to the self behaviour of the nodes in the non distributed system, the accumulated flicker is improving far more than in the distributed system. Furthermore, the two steps in the accumulate flicker of the distributed system are related to the consensus algorithm though it establish quickly after the new reference is defined. Finally the more luminance the office has the more difficulty the system has to stay reach the dead-zone, which can be seen for the higher scope of results in comparison to the ones with lower luminance.

Lastly, there is the analysis to the accumulated visibility error according to 29. Bearing in mind that the distributed controller have a slower response to a change of the reference. Whereas the non distributed controller reacts instantaneously to the error, the distributed controller starts the consensus algorithm which leads to an delay, therefore an higher slope during that period of time.

2) Uneven Costs: In order to a better comparison between the distributed and non distributed control, it was set uneven costs for each luminaire, where the second node (Luminare1) spends 2.5 times more energy than the other. This simulates a real situation where the lamps nearby the windows should be less used, to get advantage from the light outside (external). For this simulation the vector of cost is

$$c = \begin{bmatrix} 2.5 \\ 1 \end{bmatrix}. \quad (34)$$

In the set of figures 31 are represented the illuminance and duty-cycle of the actuation of the distributed controller. During the first 10 seconds all nodes are in the unoccupied state (20 Lux), but after 15 seconds, the state occupancy of the first node (Luminaire1) is toggled to occupied (42 Lux).

In figure 32, it is shown the energy consumption through time and, as expected, the distributed controller is more economic and therefore better in this case. In addition the figures 33 and 34 represent the accumulated flicker and the accumulated visibility for each type of control. Although the visibility and the flicker are not directly influenced by the change of costs, comparatively to the previous section, both are better for the distributed control with the uneven costs than with even.

In conclusion, although in this case it is not that significant in this system, it can be seen that the distribution control is better than the non distributed when the costs are uneven.

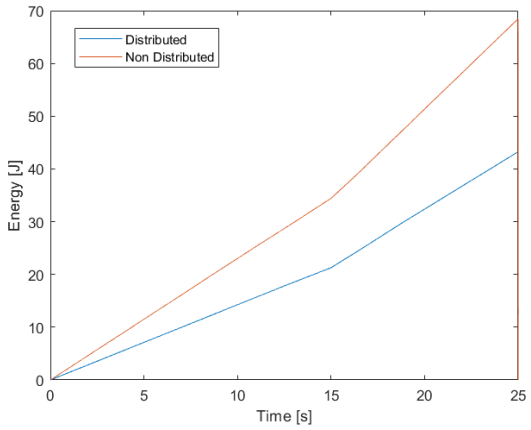


Fig. 28. Accumulated Energy for both control types

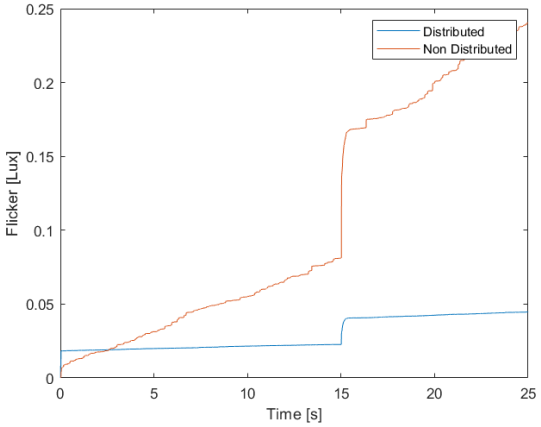


Fig. 29. Accumulated Flicker for both control types

D. Interface and Operation

THERE are three different types of commands, that allows the client to control, test, and evaluate the network remotely.

1) Get commands: This is the type of commands that the server has the response, with the exception of the first two commands of the figure 35. Regarding the last two commands they are requested in real time to the arduino. The has output has the following format:

$$< code >< address >< desired\ value >$$

In the figure 35 is available the commands of getting the external illuminance (x), the reference Lux from the consensus (r), the occupancy (o), the measured illuminance (l), the measured duty cycle (d), the time since last restart (t), the lower bound for both occupied (O) or unoccupied (U), the accumulated: energy consumption (e), flicker (f) and visibility (v), the energy cost (c) and the instant power (p).

2) Set commands: The set commands allow to change values in the arduino. There are four available commands that are also represented in figure 36, sets new lower bound value of Occupied (O) and unoccupied (U) state and also the occupancy (o), change the and set a new power cost (c). This commands are characterised by the response of the server whether 'ack' or 'err'. Additionally there is the response of the server facing bad command.

3) Real time and last minute minute buffer: Finally the real time stream is started by the command

$$< s >< illuminance(l)/dutycycle(d) >< address >$$

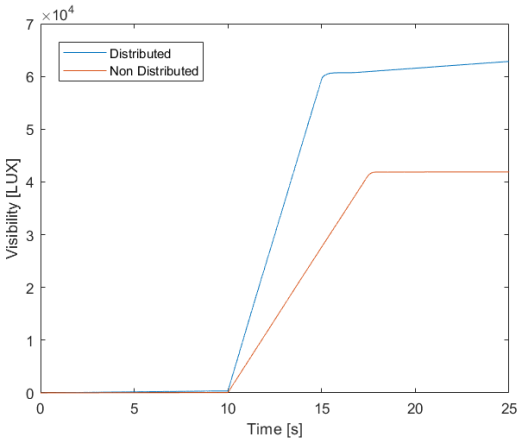
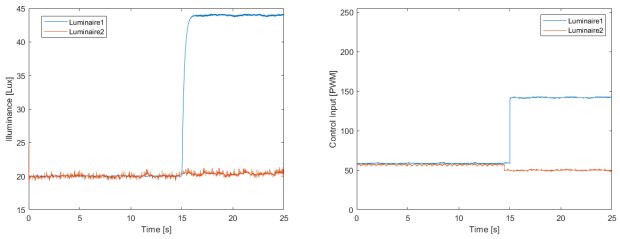


Fig. 30. Accumulated Visibility Error for both control types



(a) Illuminance Measures (b) Control input

Fig. 31. Measures of the distributed controller with uneven costs

and it only ends if the end command matches to the initial giving a acknowledge, any other stream command will be denied. This process is shown in the figure 37. The last minute buffer is directly printed in a text file, though there is no change of the terminal freezing.

4) Distributed Controller Testing: Finally it is shown in the figure 38 the evolution of the decentralised system throughout one test. The office was gradually submitted to external noise. After approximately 20 seconds one window was opened and a little later both, and sometime later one smartphone flash from outside the window pointing to the tables. The results could not has been more accurate since the first luminaire was in the occupied state (50 lux) and the others in unoccupied (20 lux).

The figure 39 shows the duty cycle from all luminaires during the test corroborates with the sequence of events.

This graph also allows to say that the PI controller is working well due to the quick response to the flash, and the stability during the test.

Bearing in mind the energy saving, the figures 40 evidences the achieved goal. When the system detects that the external light was enough to fulfil the constrains it presented a non energy consumption.

The analysis to the flicker error whether instantaneous or accumulated comes roughly from the both moments when the light was flushing the office. Besides that moments the flicker error was continuous which, as explained before, might be caused because there was not taking any threshold into account.

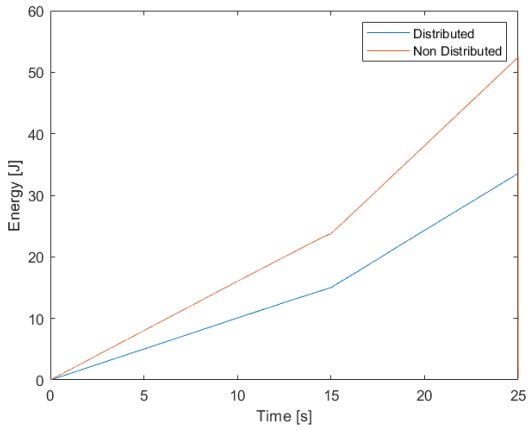


Fig. 32. Accumulated Energy for both control types with uneven costs

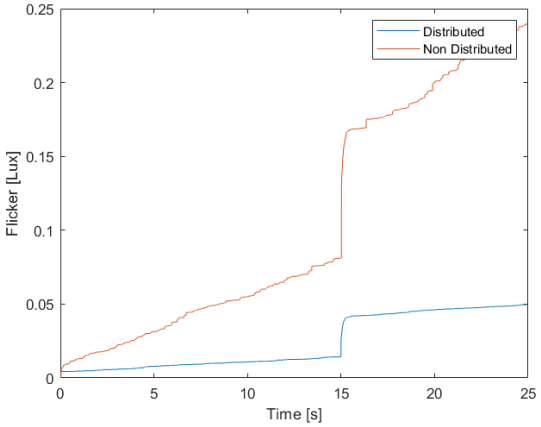


Fig. 33. Accumulated Flicker for both control types with uneven costs

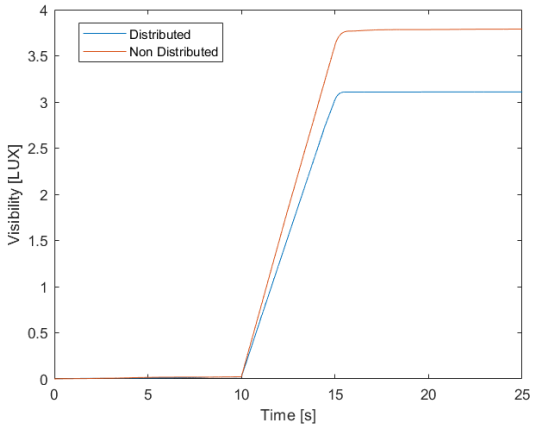


Fig. 34. Accumulated visibility for both control types with uneven costs

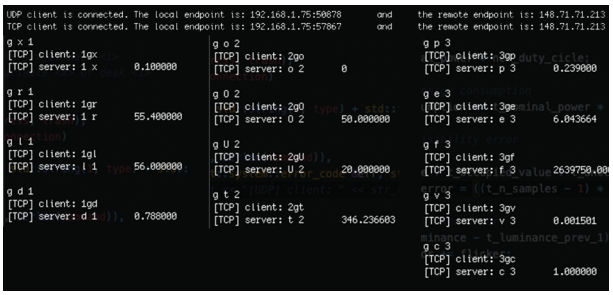


Fig. 35. Example of get commands

Fig. 36. Example of set commands

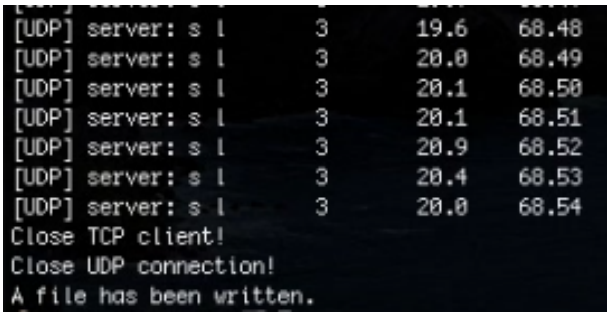


Fig. 37. Example of streaming

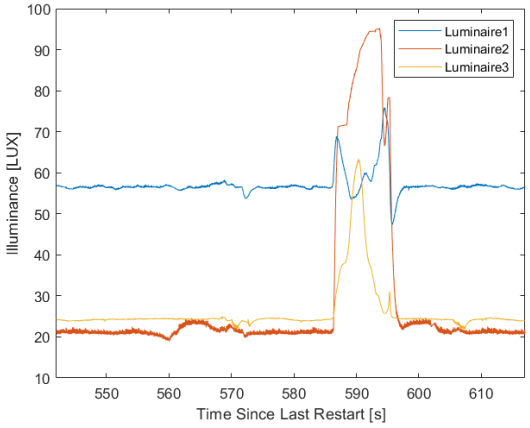


Fig. 38. Illuminance evolution for each desk

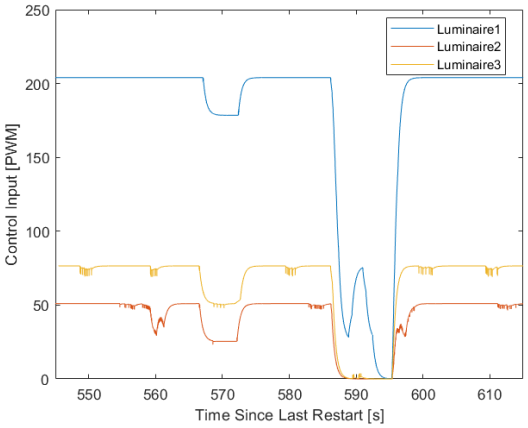
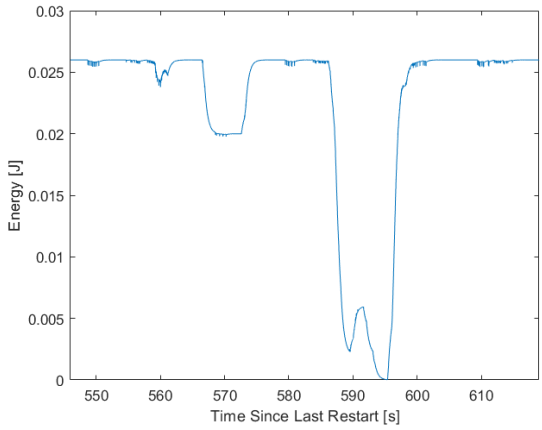
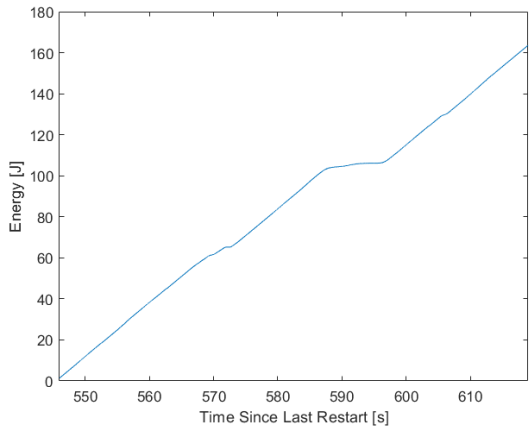


Fig. 39. Duty Cycle evolution for each desk

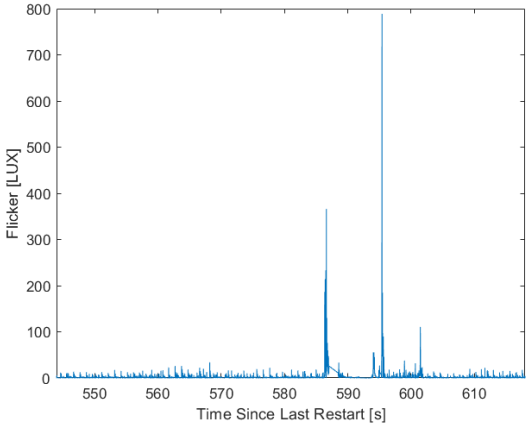


(a) Instant Energy

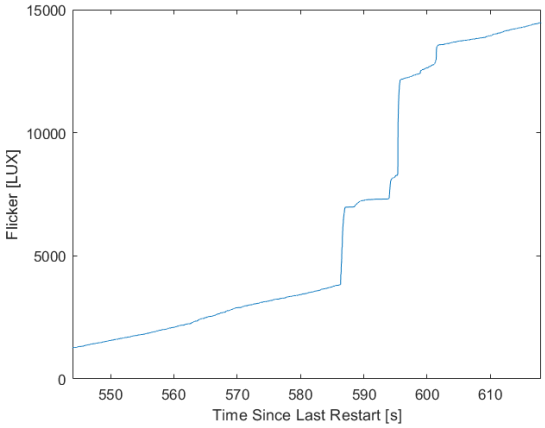


(b) Accumulated Energy

Fig. 40. Energy metrics of the system



(a) Instant Flicker Error



(b) Accumulated Flicker Error

Fig. 41. Flicker metrics of the system

VII. Division of work‡

All three students had similar workload in this project, but for a better project organisation João Almeida was responsible for the Client-Server PC application both in the code and on the report ans the video edition. On the other hand, Daniel Rosa was ahead of the calibration of the nodes and the CAN-BUS communication, regarding the code and reporting the first stage as well as all MATLAB programs in order to plotting graphs. Lastly, Guilherme Viegas was in charge of the Consensus Algorithm, Wake up function, and the Serial communications from arduino's point of view. This student also was the main contributor of reporting the experimental results. All in all we had worked as an united team throughout the entire project.

VIII. Conclusion†

The project needed a broad range of different concepts. Firstly, concepts of local control, specifically a feedforward and a feedback control using a PI controller were built. Various techniques were used for improving these controllers like *dead-zone* thresholds and *anti-windup* effect. It proved to be a pretty good implementation, driving the system to the specified reference values and reacting well to external disturbances. For the distributed system, a comparison between distributed vs non-distributed system was made, checking only a slight improvement in the energy consumption of the distributed system, but with better improvements in flickering values for example.

The wake up function and the calibration has proven to be very strong and robust for the three used Arduinos. Regarding the Consensus, the algorithm worked very well for two arduinos, but for three arduinos, seldom times it did not reach the end of the algorithm. This is well explained due to the big number of *can-bus* messages used for the consensus, and sometimes one could be lost or simply arrived at the same time as the other messages, therefore the arduino's interrupt could not detect. In this case, it was hard to implement some kind on acknowledge message because with such a number of messages, even the acknowledges could be lost. So, although both Arduino and Can-bus are pretty interesting systems, for this project case they may sin for lack of computational power and speed, specifically the existence of only two receiving message registers.

A TCP server was implemented in the Raspberry Pi to retrieve and send data from and to the Arduinos and an UDP server was implemented for the large data transfer purposes. This UDP implementation proved to be of great interest, because although some messages could be lost, being a stream the fluidity and velocity of the incoming data were preferable than some lost of few messages.

There is always room for improvement. Building a more robust *can-bus* message exchange system would be very beneficial as well as adding more security and redundancy for the incoming server messages to the Arduino. More reliable sensors, instead of just the *LDR* would also be very beneficial and would reduce a lot the experienced noise. For the server side it could even be proposed to enhance the user metrics presentation with plots of the energy consumption. Although a very dense project, this touched in many key aspects that will be, for sure, remembered in the future. All the group members are **consensu**ally satisfied with the project carried out.

The demonstration of the project is available here. Additionally the code of the project is available here.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

REFERENCES

[1] WheelsAge *Arduino MCP2515 CAN interface library*
[2] Kohlhoff, C. *Boost.Asio C++ libraries*
[3] Pandharipande, A., Caicedo, D. *Distributed Illumination Control With Local Sensing and Actuation in Networked Lighting Systems*.
[4] TruOpto, *GL5528 Light Dependent Photoresistor", GL5528 datasheet*
[5] Bernardino, A. *Slides of the Lectures of Distributed Real-Time Control Systems [18.Consensus and ADDM][19-Concurrency][20-Sockets][21-Function Objects][22-Memory Management][23-Threads][24-Mutual Exclusion]*, 2020/2021.
[6] Gaspar, J. *Boost.Circular Buffer*
[7] Open Source *C++ Signal Handling*