Real-Time Cooperative Decentralized Control of a Smart Office Illumination System

Group 9: Pedro Mendes, 81046 Miguel Malaca, 81702

***Abstract*—In order to control of an efficient way, the illumination of an office, an autonomous and cooperative real-time illumination system is created. A simple, smaller and cheaper version is implemented using 2 luminaires each of them composed by a LED, a sensor (LDR) and a unit (using an Arduino) to read data from the sensor, to process it and to control the LED according to the sensor data. Since it can be several luminaires, each of them with a controller is necessary to implement a distributed system where all nodes work together to achieve a common goal of minimizing the energy consumption but restricted to the occupancy state and the minimum requirements of each node that have to be satisfied. The nodes in a distributed system need to communicate between them so a protocol of communications has to be implemented in order to exchange data between luminaires and this data could be correctly recognized. The performance of the system is evaluated using the simulated office in order to validate this implementation. The solution proposed is tested not only with sources with same characteristics but also with different to prove the robustness and capability to minimize the energy.**

***Index Terms*—Smart illumination office, distributed illumination system, distributed controller, LED control, consensus algorithm.**

I. Introduction

PROGRESS in the semiconductor industry enabled the mass production of efficient, inexpensive and minimal light sources, such as LEDs. Ever since the appearance of these sources, people try to decrease energy consumption while having access to the same light conditions, by changing the old sources to these new ones. They also opened the doors to the research and creation of intelligent luminaires that are able to check when people need light (presence detection) and to control the source to create certain levels of illuminance in the environment. Similar things are now very common in the market when they detect presence they turn on the light when the illuminance level is not superior to a given bound. However, this solution doesn't consider the existence of many light sources and cannot control the given light to meet some requirement. In a study room where every desk has a luminaire the perspective of having a distributed control system for the lights seems very useful. If the room should have a constant value of illuminance bigger than 20 LUX (in this implementation) and for a person to be studying its desk should have a illuminance value greater than 50 LUX, a control algorithm that could take into account the contributions of all the lights in order to have the most efficient solution would be really helpful. This is the problem that this report aims to solve, to implement a distributed control algorithm that is able to follow standard values of illuminance minimizing the power consumption while giving the user methods to monitor the lighting conditions.

In this approach, a room is simulated by a box and two luminaires constituted by an Arduino with a LED and an LDR sensor. The I_2C communications are used to send messages between luminaires. Also, a Raspberry Pi is connected to the system to implement a server where the users can connect to get metrics about the illuminance in the room. The presence sensor is a push button that toggles the occupancy state of the desk. The objectives are to design and implement a controller with a satisfying performance that can react to disturbances and take into account the influences of other sources while minimizing the power consumption of the whole system that can have sources with different powers and to implement a server in which useful information is stored to evaluate the lighting conditions. The work is divided into two stages, the first one where only a luminaire is considered and whose objective is to develop a local controller capable to drive the system to a given reference value using feedback, feedforward, and some controller improvements. In the second stage, the other luminaire is added to the system and distributed methods are considered not only for 2 desks but creating a solution that comprises 127 possible desks. To solve the control problem an optimization technique will be used [2] and the comparison between using only the local controller or the distributed one will be made. Also, a TCP server that is able to store data is implemented in the Raspberry Pi.

In the chapter II a brief revision of the state of the art is made addressing a reference where a similar implementation is made. In III the implementation details for every part of the system are explained. It is divided into relevant tasks. Chapter IV has the experiments and results, organized by stage, that were made to confirm the implementation and performance of the solution proposed. Also, the results obtained previously are discussed and in chapter V conclusions about solution presented and some problems that appeared during its implementation are stated. Also in this chapter, some improvements for future work are suggested.

II. State-of-the-art

During the revision of the literature, an article with a similar implementation was found. In [1] a distributed control algorithm is also implemented. However, the authors consider that the luminaires can only communicate with a set of neighbors. Also, all the controller parameters are determined in the optimization problem, even the ones from the local controller. Furthermore, the optimization problem used serves the same

issue but needs to determine different variables and does not consider different sources that may have different costs. This implementation doesn't consider noisy light sensors. The article refers to the comparison between its controller and a situation without any controller where the LED dimming values are constant.

III. Development

A. System Architecture

The system is composed by two nodes, each of them composed by an Arduino, a LED, a Light Dependent Resistor (LDR) in order to read the illuminance, a set of components (resistors, capacitor, push button), all connected using a bread-board and wires.

Each node can be divided into two main circuits, one responsible by the LED's control and the other one responsible by the illuminance reading. The LED driving circuit connects one of the PWM outputs of the Arduino to the LED's positive pin. A resistor of 100Ω connects the other pin of the LED to the ground. The PWM output allows the control of the LED's illuminance by varying the duty cycle of the output signal. The scheme of the LED driving circuit is represented in figure 1.

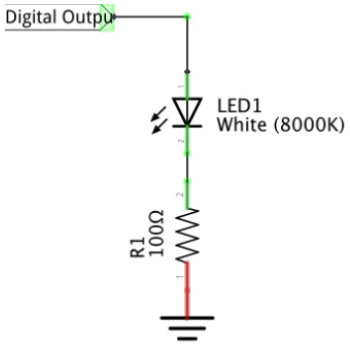


Fig. 1: LED driving circuit

The illuminance reading circuit is a voltage divider circuit constituted by an LDR and a resistor of $100k\Omega$. In order to reduce the noise on the circuit, a capacitor of $1\mu F$ is used, connecting the LDR to the ground to remove the AC component of the signal. The illuminance reading circuit's scheme is represented in figure 2.

The circuit is also composed by a switch, using a push button and a resistor. The push button is connected to the digital input port of the Arduino. The goal of this circuit is to toggle the state of the LED. While the push button is not pressed, the voltage of 0V is forced in the input port. When the push button is pressed, the input port is connected to the 5V.

Each node of the system is connected to a Raspberry Pi working as a server, through a common data bus, used to send data from the Arduinos to the Raspberry Pi. The Raspberry Pi only works as a slave node, receiving and processing data. The Arduinos can be a master node, sending data to the other nodes and to the server, and a slave node, receiving information from the other nodes.

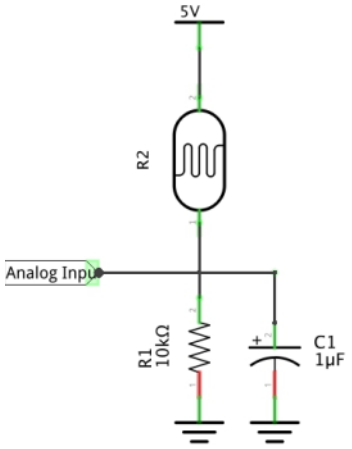


Fig. 2: Illuminace reading circuit

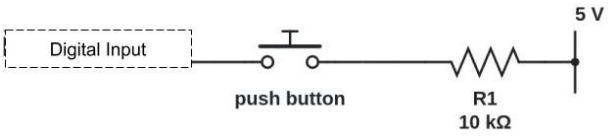


Fig. 3: Switch circuit

B. System Description

The goal of this system is to control the illuminance of an office, creating a real-time cooperative distributed control system. As explained in the previous section, the system can be composed of several nodes. Each node has an Arduino, whose PWM output signal is used to control the LED's illuminance intensity, changing the duty cycle of this signal. The PWM is an integer between 0 to 255. If the PWM value is 0, the duty cycle of the output signal is 0. If the PWM value is 255, the duty cycle of the output signal is 100%. A duty cycle of 50% corresponds to a PWM value of 127. The higher the duty cycle of the output signal is, greater the illuminance intensity of the led is.

The illuminance of the space is read using a sensor (LDR) connected to an analog input port of the Arduino. The voltage is read by an Arduino and store using 10 bits register. So, the voltage is converted in an integer value between 0 and 1023, where 5V corresponds to a value of 1023.

C. System Model

The system can be described using mathematical equations in order to model the system. Firstly, the analog input read from the Arduino needs to be converted in voltage. As previous explain, the Arduino maps the voltages from 0V to 5V using integer values with 10 bits (values from 0 to $2^{10} - 1 = 1023$). Knowing that 5V corresponds to a value of 1023, the conversion of the analog input in voltage is

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

Analyzing the circuit of figure 2 and applying the Kirchhoff's voltage law (KVL), the voltage on the LDR is

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

The current on the resistor R_1 and on the LDR is the same (in DC) and can be determined through the Ohm's law.

$$V_{R1} = R_1 I \Leftrightarrow I = \frac{V_{R1}}{R_1}. \tag{3}$$

The resistance of the LDR is given by

$$R_{LDR} = \frac{V_{LDR}}{I} \Leftrightarrow R_{LDR} = \frac{5 - V_{R1}}{V_{R1}} \cdot R_1. \tag{4}$$

Using the LDR's resistance determined, it's possible to calculate the correspondent value of illuminance. The relation between the illuminance measured and the correspondent LDR's resistance is not linear. Analyzing the datasheet, it's possible to see that there is a logarithm relation. Using this logarithmic scale, a straight line can be determined, inside the region of the characteristic. Converting to a linear scale, the equation (5) of the LDR's characteristic is obtained

$$\log_{10}(R_{LDR}) = m \log_{10}(L) + b, \tag{5}$$

where L is the illuminance measured by the sensor, m is the slope of the characteristic in the logarithmic scale and b is the interception of the characteristic with the origin in the logarithmic scale (logarithm of the resistance at 1 LUX). The illuminance value is

$$L = 10^{(\frac{\log_{10}(R_{LDR}) - b}{m})}. \tag{6}$$

The m and b are unique parameters of each LDR. These parameters are calibrated, firstly, using a luximeter to determine the illuminance and then using a multimeter to measure the voltage of the LDR. For some different illumination situations, the true value of the illuminance is measured and the value of R_{LDR} is calculated. This is done for both luminaires since the LDRs used are different. The results are shown in table I.

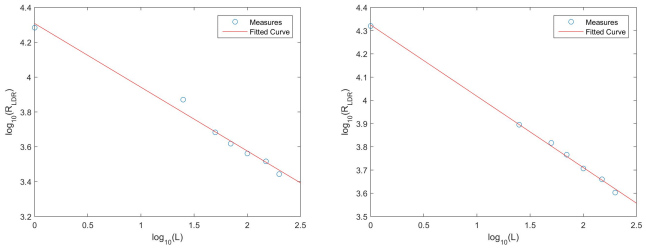
Illuminance	R_{LDR}	Illuminance	R_{LDR}
200	2771.536	200	4013.699
150	3285.714	150	4572.65
100	3640	100	5088.496
70	4149.378	70	5835.913
50	4826.087	50	6553.398
25	7427.597956	25	7853.403
1	19228.57143	1	20906.34

TABLE I: Results for the both illuminaires

In order to find the values of m and b, based on (5), the logarithms of the results in the table I were computed and organized in graphs. By (5) those values should evolve linearly with each other and m and b can be obtained by the linear regression.

From figures 4(a) and 4(b) one can see that the linear behavior predicted in (5) is verified in both luminaires. The values for the linear regression are $m = -0.3659$ and $b = 4.308$ for the first luminaire and $m = -0.3069$ and $b = 4.3246$ for the second one.

To validate the model, the relation between the illuminance value measured by the LDR and the LED actuation given by



(a) Results for the first LDR (b) Results for the second LDR

Fig. 4: Characteristics of the LDRs ($log_{10}(LUX)$ - $log_{10}(R)$)

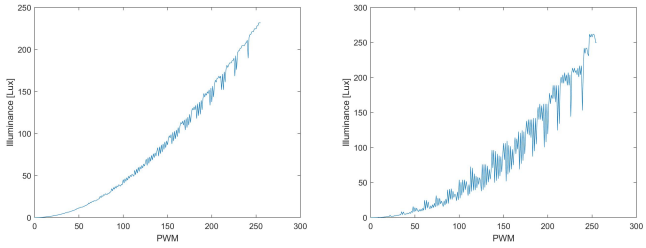
the PWM duty cycle was obtained. Admitting a first order system given by

$$G(x) = \frac{K_0(x)}{1 + s\tau(x)}, \tag{7}$$

one can see that the DC gain ($x = 0$) is constant $G(0) = K_0(0)$. Therefore, it is expected that the relation between the illuminance and the PWM stated is a linear one given by

$$L = Kd + o. \tag{8}$$

In order to do that, for different values of PWM in steady state, the illuminance level was obtained. This was done for both luminaires since serves as a validation of the values of m and b.

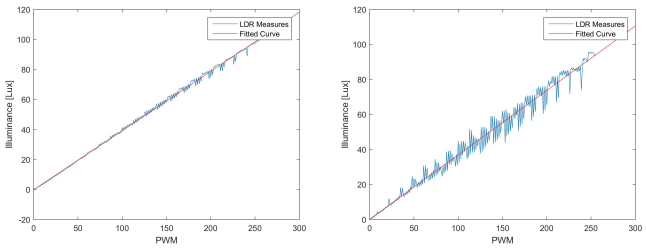


(a) Relation between Illuminance and PWM for the first LDR (b) Relation between Illuminance and PWM for the second LDR

Fig. 5: Steady state characteristics of the LDRs (PWM - LUX)

In the figures 5(a) and 5(b), one can see that the values proposed don't preserve the linear relation between the PWM and the illuminance measured. However, this relation needs to be verified in order to properly solve the problem. This way, the values of m and b were adjusted manually.

The figures 6(a) and 6(b) show that, with new values of m and b the expected behavior is obtained. These new values, even though not according to the ones measured with the luximeter, make possible to use the steady gain and proportional relation between illuminance and the PWM duty-cycle. The values are $m = -0.71$ and $b = 4.85$ for the first system and $m = -0.68$ and $b = 4.8$ for the second one. It was also seen, that these values were not very robust since with the movement of the box, a lot of things changed. Therefore, they don't preserve the reality about the illuminance measure but



(a) Relation between Illuminance and PWM for the first LDR

(b) Relation between Illuminance and PWM for the second LDR

Fig. 6: Steady state characteristics of the LDRs after changing values (PWM - LUX)

they guarantee the linear behavior that is crucial to properly solve the problem.

To test the response of the system and to determine the system's gain, an input step signal is generated. The system response obtained is plotted in figure 7 and analyzing it, it's concluded that the system response behaves like a system of the first order as in (7) where x is the target illuminance and it's possible to conclude that the system's gain ($k_0(x)$) and the time constant ($\tau(x)$) depend on the value of LED's illuminance.

To different values of PWM, an input step signal is created and the response obtained is analyzed. After the response stabilizes, the voltage of the LDR is measured. The results obtained are registered on figure 7.

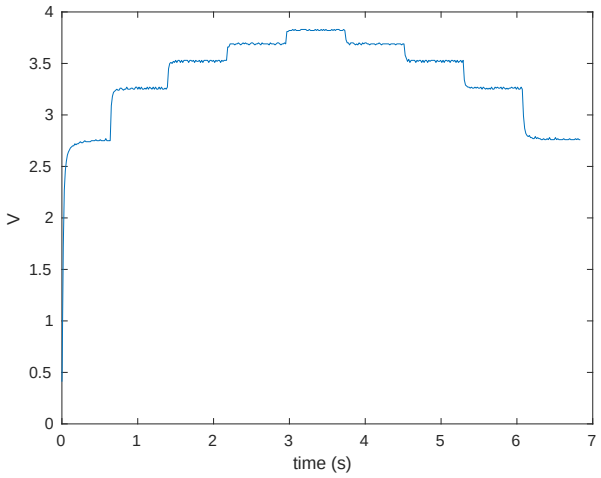


Fig. 7: System Response

Using this information and doing a plot of the static gain in function of the PWM (figure 8), it's possible to determine a mathematical equation of the trend line that fits the data. The equation (9) calculates the systems static gain in function of PWM.

$$K_0(x) = 0.015 + 0.085 \cdot 10^{-0.007x}, \tag{9}$$

where x is the PWM value. The static gain when using positive step signals are similar to the gains when using negative step signals to the input

The illuminance reading circuit of figure 2 has a capacitor that charge and discharge depending on the voltage of the

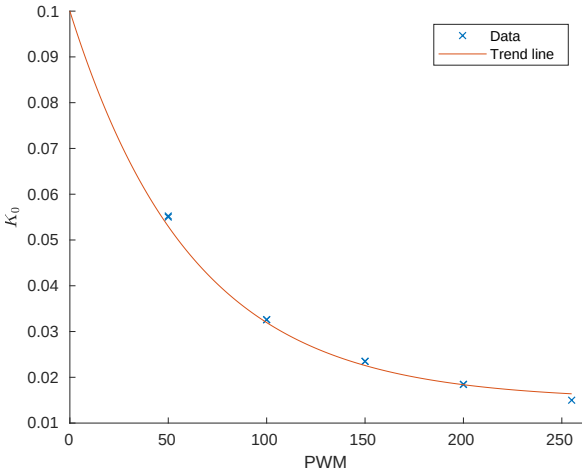


Fig. 8: Static Gain K_0 in function of PWM value

LDR. The response of this first order system is due to the charge and discharge of this capacitor. The time constant, τ, of a *RC* circuit is

$$\tau(x) = R(x)C, \tag{10}$$

where C is the capacity of the capacitor, R is the equivalent resistance of the parallel of the resistor R_1 and R_{LDR} and x is the illuminance measured by the LDR.

$$R(x) = R_1 // R_{LDR}(x) = \frac{R_1 \cdot R_{LDR}(x)}{R_1 + R_{LDR}(x)} \tag{11}$$

Substituting the result of (11) on (10), the time constant is given by

$$\tau(x) = \frac{R_1 \cdot R_{LDR}(x)}{R_1 + R_{LDR}(x)} \cdot C. \tag{12}$$

However, the time constant also can be determined experimentally, using the same procedure performed before to calculate the static gain. An input step signal is created for different values of PWM and the response is analyzed. The time constant corresponds to the instant of time when the response reaches 63% of the final value (when the input signal is a positive step). The results obtained are presented in figure 9.

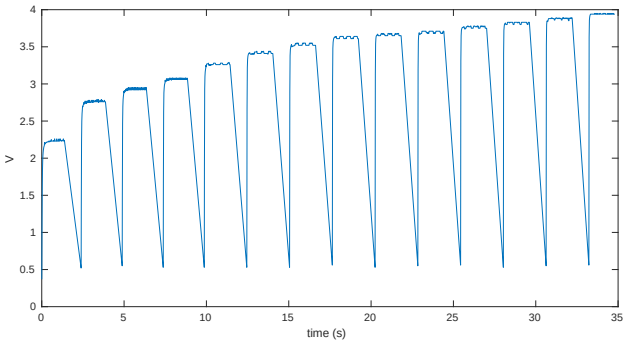


Fig. 9: System response to different step signals

Using the results obtained, the time constant is calculated in function of the PWM value. The results are register in figure 10. It's possible to determine a equation that fits the data.

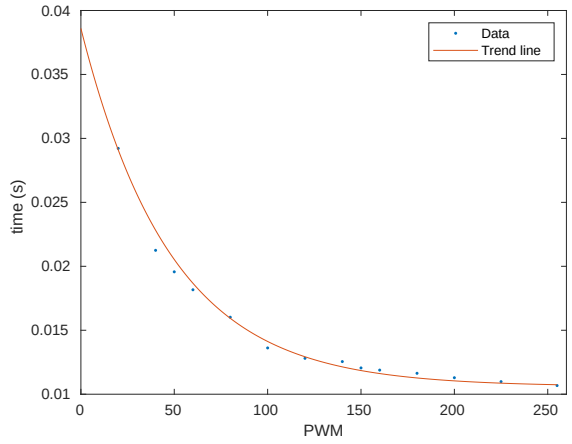


Fig. 10: Time Constant in function of PWM value

$$\tau(x) = 0.0106 + 0.0128 \cdot 10^{-0.009x}, \tag{13}$$

where x is the PWM value.

The system was also tested using negative input steps however the results of the time constant were similar to the positive steps.

The gain of the overall system is defined as the relationship between the illuminance, in LUX, and the PWM value (duty cycle), because as see before there is a linear relationship between both. The previous results can be converted from voltage units to illuminance, in LUX, using equations (4) and (6).

D. Feedforward Controller

The goal of the feedforward controller is to determine the PWM value of the input signal to reach a reference (desired) value of illuminance (figure 11).

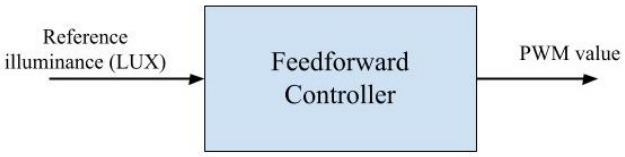


Fig. 11: Feedforward Controller

The system's gain is

$$K = \frac{illuminance}{PWM}, \tag{14}$$

so, the PWM value that produces a reference illuminance value is given by

$$PWM = \frac{illuminance}{K}. \tag{15}$$

E. Simulator

The simulator is a system responsible to calculate the true output illuminance value when a certain reference illuminance value is desired.

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

$$v(t) = v_f - (v_f - v_i)e^{-\frac{t-t_i}{\tau(x)}}, \tag{16}$$

where t_i is the initial time of the input signal, $\tau(x)$ is the time constant that depends on the illuminance read, v_f is the voltage value corresponding to reference illuminance value and v_i is the initial voltage of the step signal, both calculated through the manipulation of (6)

$$R_{LDR} = 10^{m \cdot log_{10}(illuminance)+b}, \tag{17}$$

and (4)

$$V = \frac{5}{1 + \frac{R_{LDR}}{R_1}}. \tag{18}$$

F. Feedback Controller

Using only feedforward control is a risky decision because if the input doesn't result in an output equal to the reference it won't correct it. The best way to do this is by implementing feedback control which determines the control input to give based on the error between the output and the reference. This controller gives the possibility to react to disturbances and adds robustness to model uncertainties with the downsides of adding some delay and introducing noise from the measurement.

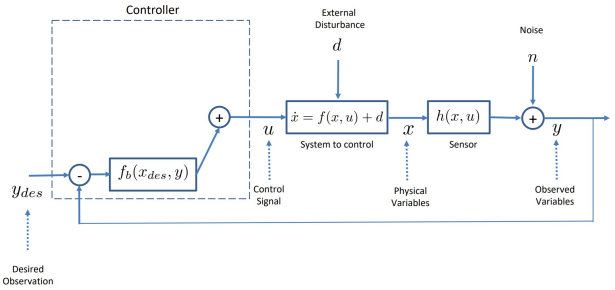


Fig. 12: Feedback controller structure

One of the most common implementations of a feedback controller is the Proportional-Integral-Derivative controller. This one is based on how the controller will react to the error between the reference and the output. It can be divided into three parts addressed separately.

1) Proportional: The proportional part of the controller does one simple thing: amplifies the error through a constant k_d and adds it in the input. The problem is that, depending on the system, the controller may not be able to drive it to the reference because the amplified error may not be enough to compensate the input. It also has the effect of increasing the overshoot and the frequency of oscillations. However, it is the fastest part of this controller since it doesn't need more than the actual measure to act.

2) Integral: The Integral part of the controller accumulates the error during a time and this value will be amplified and added to the input. It doesn't normally act alone, instead, it works normally with a proportional part. Even if the instantaneous amplified error is not enough to drive the system to the correct value, the cumulative error can take care of that. However, if the integrator doesn't have a limit it may wind up if the actuators saturate. For example, when the box is opened and the actuators don't find a way to drive the illuminance measure to the desired one because of the external luminosity, the integrator will accumulate error and having really high values that are really slow to erase. This can be solved by implementing an anti wind-up technique.

3) Derivative: The derivative part does some prediction of how the system is evolving since it can access how the error is growing. It can make the system faster, reduce the overshoot and also reduce the frequency of the oscillations. However, it is very sensitive to the noise.

For this particular problem, only the Proportional and Integral parts will be implemented. The controller used is a discrete-time PI with the values

$$\begin{aligned} p &= k_p e, \\ i &= i_{ant} + k_p k_i \frac{T}{2}(e + e_{ant}), \\ u &= p + i, \end{aligned} \tag{19}$$

where e is the error between the output and reference, p the proportional part with k_p the proportional gain, i the integral part with k_i the integral gain, T the sampling period, the subscript ant stands for the value in the previous sample and u the control input determined by the feedback.

In addition, the anti-windup technique is also applied. It is a term that sums in the integral part. It basically detects when u saturates and when it happens this term will compensate the growing of the absolute value of the error. Special attention is required to the saturation limits of u because it will be summed to the control input determined in the feedforward part. It is known that

$$0 \leq u_{total} = u + u_{ff} \leq 255 \Leftrightarrow -u_{ff} \leq u \leq 255 - u_{ff} \tag{20}$$

where u_{ff} stands for the control determined in the feedforward part. Applying this saturation to u and subtracting the actual u by

$$u_{wdp} = u_{sat} - u, \tag{21}$$

adding it in the integral part by

$$i = i_a nt + k_p k_i \frac{T}{2}(e + e_a nt) + u_{wdp}, \tag{22}$$

the possible saturation of the actuators is compensated because i will never be too big or low.

Another thing that was implemented to improve the performance of the controller was a deadzone in the error. Due to the noisy measures made by the LDR the quality of the response was improved with aid of deadzones. They just make that an small error with a value different than 0, is considered 0. In this case, for a difference of under 1 LUX was considered 0.

Finally, the existence of flickering was detected when opening the box an exact quantity. This may happen because the actuator is really close to its low limit. Therefore, in the transition phase with values of 1 and 0 of the control input it flickers. To solve this, a kind o hysteresis was applied. Basically, the led turns off when the control input is 0 of duty cycle but only lights up again when the control input is 3. This gap makes the flickering disappear.

G. System's Frequency

The system clock is 16MHz. However, each Arduino has three different timers and they are all used in this implementation.

Timer0 which is responsible by the frequency of delays and time functions. The frequency of this timer is the default frequency ($f_{T0} = 976.5625$Hz).

The *Timer1* is used to control the interruption frequency. In order to generate an interruption every 10 milliseconds, the registers of the integrated circuit (*TCCR1B, TCNT1, OCR1A, TIMSK1 and TCCR1A*) are programmed. This implementation was used because is very accurate. The interruption is enable ever 10.016ms.

The *Timer2* is used to control the frequency of the digital input pin number 3. This frequency is set to the maximum ($f_{T2} = 31372.55$Hz) in order to attenuate the effect of charge and discharge the capacitor, i.e., to prevent the ripple effect and creates a constant signal.

H. Non Distributed System Architecture

An individual control system is implemented using the previous modules. The scheme of the system is represented in figure 13.

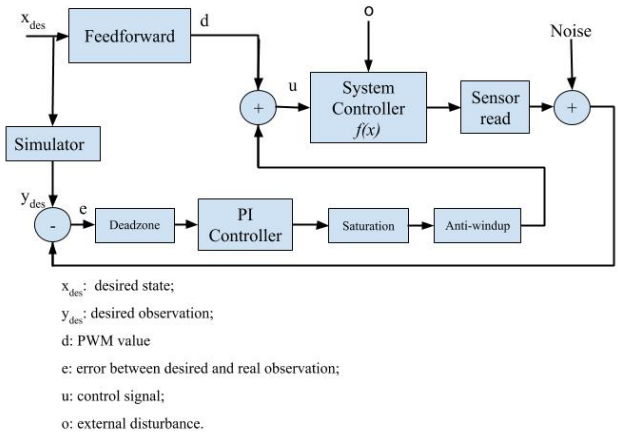


Fig. 13: System Scheme

I. Model of the Distributed System

The goal of the distributed system is to create a distributed controller for a network with a maximum of 127 nodes (there are 128 available addresses, however, address 0 is the broadcast address and also is the address of the server in the network). The addresses are given *a priori* to the program of each node and the address is stored in the EEPROM. The only requirement when giving the address is that all node must be a different address. This implementation was chosen instead

of a dynamically attribution of the addresses in the network because it requires fewer resources and time consumption (manual attribution of addresses has a complexity of $\mathcal{O}(1)$ and the dynamical attribution of the addresses has complexity of $\mathcal{O}(n^2)$, related the number of messages exchanged in all the network) and it's faster to compute and simple to implement.

Each node is equal to the one described in the previous sections. All nodes in the network work together to achieve the same goal, i.e., nodes are not independent of each other. The illuminance of one LED has influence in the other nodes sensor. The level of influence of one node in the other ones is given by the matrix of gains. The influence of one node in the other one is represented in the system's equation by adding a term that gives the relationship between the illuminance read (when only one node has the LED turn on) and the PWM value of this node. The new system of equations is

$$\begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_{126} \end{bmatrix} = \begin{bmatrix} K_{1,1} & K_{1,2} & \dots & K_{1,126} \\ K_{2,1} & K_{2,2} & \dots & K_{2,126} \\ \vdots & \vdots & \ddots & \vdots \\ K_{126,1} & K_{126,2} & \dots & K_{126,126} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{126} \end{bmatrix} + \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_{126} \end{bmatrix} \quad (23)$$

where L_i is the illuminance value read by node i, in LUX, $K_{i,j}$ is the gain of node i when only the LED of node j is turn on, d_i is the PWM value in percentage (from 0 to 100) of node i and o_i is the external illuminance observed in node i, in LUX.

J. Calibration

The system is constantly changing. Not only because there are new nodes entering, but also when considering a single luminaire, with the movement of the box and the changing of the external illumination conditions, the relation between PWM duty-cycle and illuminance measured may change. In order to overcome this problem and also give knowledge of how the LDRs measure the light of all the nodes in the network, the system should be calibrated every time that starts and when a new node enters, determining a constant gain that transforms the value of PWM in the illuminance measured.

The basis of the algorithm is simple, solving only

$$L = Kd + o, \quad (24)$$

in order to find K. The led should be completely turned off and the illuminance value measured in order to get the influence of the external illuminance o. then the PWM duty cycle, d, should be changed to a known value, 255 for example and the illuminance value measured L. Knowing all these values one just needs to compute using

$$K = \frac{L - o}{d} \quad (25)$$

Doing this for only one node is easy. However, when there are more than one, each one should solve as many equations as the nodes that exist determining all the values in (23). To solve this, a distributed algorithm should be implemented, using messages that can be transmitted between nodes either in broadcasting or unicast messages. All the nodes have in them an array of 128 integers which has zeros for the addresses of the nodes that are not in the network and ones for the others.

Firstly, a node that tries to enter in the network needs to have a certain behavior to inform the others that it's trying to join them.

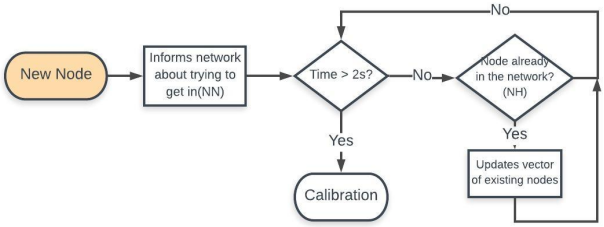


Fig. 14: Scheme for new node behavior

In the figure 14 this behavior is represented. This process indicates that when a new node wants to get in the network send a message to every node saying it (NewNode: NN) and waits during 2 seconds for answers from the nodes that were already in and, if there are any, the new node will update the vector representative of all the nodes. The nodes that were already in the network have the behavior represented in figure 15. They receive the NN message and send directly to the new node an NH message (NodeHere) saying that it is there. It also updates the vector in the new node address. It also waits some time before entering the calibration process to make sure the new node has time to reach it.

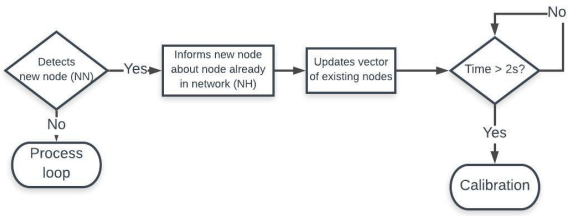


Fig. 15: Scheme for the old node behavior

The most complex part of the algorithm is the calibration itself whose fluxogram is represented in the figure 16. It shows that the algorithm runs until all the nodes are calibrated, which means, all the nodes have to turn the LED once for the others and itself to calculate the gains. This means that the Arduinos should have a variable that has the number of luminaires that have already been calibrated. When there are nodes to be calibrated, the luminaire has to check if it is its turn to be calibrated. It does that by comparing the number of calibrated nodes and the number of luminaires in the network and ordering them by address. For example, if a network has the nodes 1, 2, 5 and 8 and 2 nodes are already calibrated it's the turn of the node with the address 5. In the node's turn, it measures the external illumination and after turns on the led letting the others know by broadcasting an RML (ReadMyLed) message. The other nodes receive the message and do the computations to determine the K value with respect to the node turned on sending a message CYL (ConfirmYourLed) after,

informing that it has read the illuminance value successfully. Then, it increments the number of calibrated nodes and waits for the LED that was turned on to reach the end and informs the other luminaires. When the node that is being calibrated receives the information that the illuminance has been read successfully from all the other nodes it increments the number of calibrated nodes and calculates its own gain telling the others that it has turned off the LED. This implementation guarantees that the calibration works for a number of 127 nodes but it will only be tested with 1 or 2 due to hardware limitations.

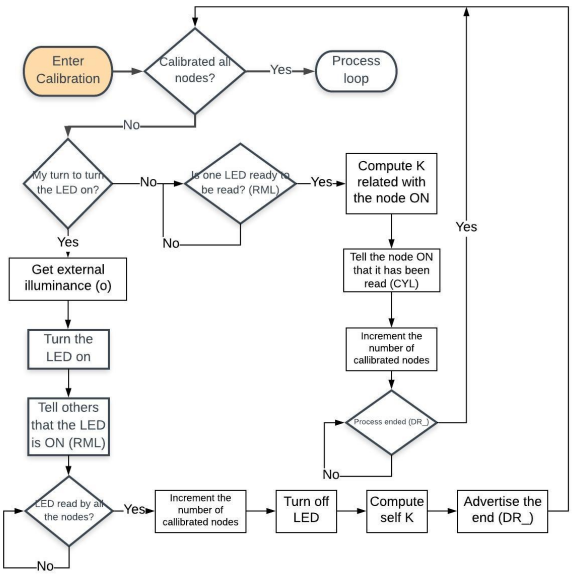


Fig. 16: Scheme of the calibration

K. External Illuminance

The external illuminance is measured every time at the beginning of the calibration process. All the LEDs are turned off and each node read the illuminance value. In this way, each node can determine the external illuminance.

L. Consensus Algorithm

The system implemented is a distributed cooperative system, where each node works in coordination with the other nodes to perform a given task and achieve the same goal of minimizing the cost and the energy consumed by the system.

The consensus algorithm is implemented in order to achieve an agreement that fulfills all the requirements and constraints of all nodes. Each node computes and minimizes its own cost function given by

$$\begin{aligned} \underset{d\in\mathbf{R}^N}{\text{minimize}} \quad & c^T d \\ \text{subject to} \quad & Kd + o \geq L; \\ & 0 \leq d \leq 100; \end{aligned} \tag{26}$$

where

$$d = [d_1 \quad \dots \quad d_N]^T, \tag{27}$$

is the vector of PWM values (in percentage) of all nodes,

$$c = [c_1 \quad \dots \quad c_N]^T, \tag{28}$$

is the vector of energy costs of the LED, K is the vector of coupling gains from the other luminaires to itself (converted in percentage), o is the external illuminance and L the reference value of the illuminance.

The cost function is subject to two constraints, the first one responsible for the LED achieved the lower bound, L, and the second one, responsible for the PWM is a value in percentage (between 0 and 100). Each node holds a local copy of the variables [2].

The algorithm tries to reach an agreement in the PWM value of each node. This algorithm was implemented with fifty iterations because in this implementation the network only has two nodes and a high number of iterations is not required to achieve a consensus. Each node maintains a local copy of d with the average solution of all nodes and the Lagrange multipliers [2]. In each iteration, the algorithm tries to evaluate and minimize the cost function, checks its local feasibility, verifies if the problem's constraints are satisfied, sends its own copy of array d to all the available nodes in the network and updates its own Lagrangian with the average of all d values. At the end of the iterations, all the local copies of variable d should be identical.

In this implementation, all nodes have the same LED's type, so the cost is identical in all nodes and in this case the cost is set to one for all nodes in the network. The result of the consensus algorithm can dictate that a luminaire may exceed its own minimum required value in order to help the other nodes to achieve their goal.

After solving the optimization problem and the algorithm has determined the PWM value applied in each LED, in order to determine the desired illuminance in each node, the system (23) is solved.

M. Distributed System Architecture

With the implementation of the consensus algorithm, the system determines the reference illuminance and the PWM value that each luminaire has to accomplish. Therefore, the feedforward controller previously developed is not needed and it's replaced by the consensus algorithm. The scheme of the overall system implemented is represented in figures 17.

Three different classes of C++ were used to implement the system in the Arduino. The first class is responsible for all the calibration. The second one is used to implement the consensus algorithm. The third C++ class contains all the controller functions.

N. Server

The network has a server (performed by the Raspberry Pi) that is responsible to acquire data from all the available luminaires and at the same time give the information requested by clients.

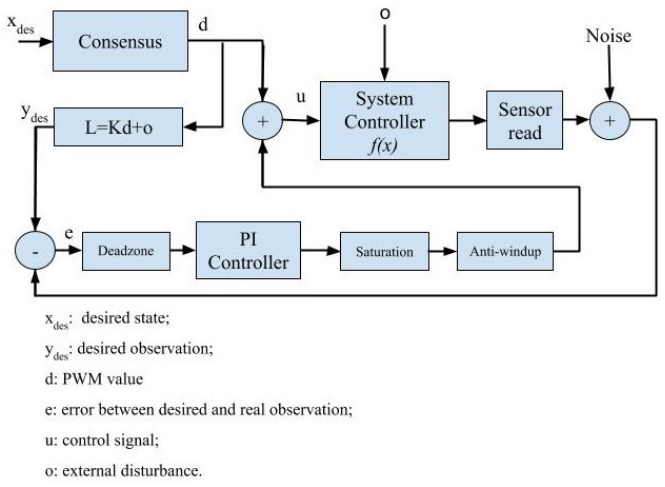


Fig. 17: System Scheme

1) Implementation: An asynchronous TCP/IP server was implemented in the Raspberry Pi. This server is collecting information from the Arduinos about the luminaires and after storing and processing the data it's able to reply to the requests of clients.

The implementation of an asynchronous TCP server has some advantages. The transport protocol used was TCP/IP instead of UDP because even though the UDP protocol is faster this can not ensure that the messages are delivered. The TCP protocol creates a direct connection, to a specific port, between the server and the client, and ensures that all the messages are correctly delivery, without errors and in order. An asynchronous server has some advantages comparing with synchronous servers. One of the most important is the fact that the server does not block when is receiving a message, processing the request and replying to the client. In this implementation, the same process handles all the communications with all the clients (without using a thread to handle the communication for each client). There is a queue that stores the order to process the sending and the receiving messages for all the clients. The tasks of reading and writing stored in the queue are processed asynchronously by an auxiliary mechanism and due to that, the main process does not block. When the receiving or sending data operation ends, an auxiliary mechanism notifies the process on the completion of the task [3]. The asynchronous server requires auxiliary tasks to handle the read and write operation.

The server was implemented in C++ language using the *Boost* Library. Two threads were used, one to read, store and process the data of I_2C bus, and another one to handle the communications and request from clients.

2) Data Acquisition: The thread that listens to the I_2C bus behaves as a slave node because only reads data from the Arduinos (it cannot send data to the Arduinos). The address of the server (in the network of luminaires) is 0, so the Arduinos, in order to send the data to the server, need to address a message to address 0. To read the data from the I_2C bus, it's used the *PigPio* Library. When there is a message available on the I_2C bus, it's read and saved in a vector of chars. The protocol used sends data in byte format, so this message needs to be decoded. The first byte gives the type of message and the reaming bytes are separated in groups. The protocol of messages used is explained in the following section.

Each Arduino sends a message to the client when finishes the calibration, after running the consensus algorithm and every 10 milliseconds, after the control interruption being triggered.

3) Data Structure: The server has an array of pointers with size of 128 corresponding to the possible valid addresses in the network. When a new node is detected, i.e., when a message is received and there isn't any information about this new node, a data structure is created dynamically containing all the information about the node and the pointer to this new data structured is assigned to the same address in the array of pointers.

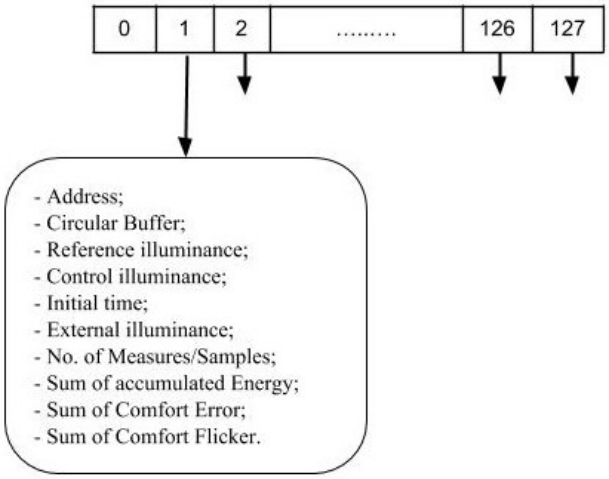


Fig. 18: Data Structure

On this way, the server only allocates memory when there is a new node in the network.

Each structure is implemented in a C++ class. The circular buffer is implemented using the *Boost* Library and it has a length of 6000 in order to store the measured illuminance and the PWM value of the last minute. The server receives data from the Arduinos to store in the circular buffer every $10ms$. So to be possible to store data of the last minute the buffer needs to have a size of 6000. When the buffer is full, the oldest item is overwritten, with a policy of First In, First Out (FIFO). The circular buffer is of the type of a data structure containing two floats (measure illuminance and PWM).

Every time new data is available in the I_2C, the message is read and processed and if the node already exists in the array the information is updated in the structure corresponding to the sender address.

The thread that reads data from the I_2C only writes in this database and the thread that handles the request of clients only have read access to this database.

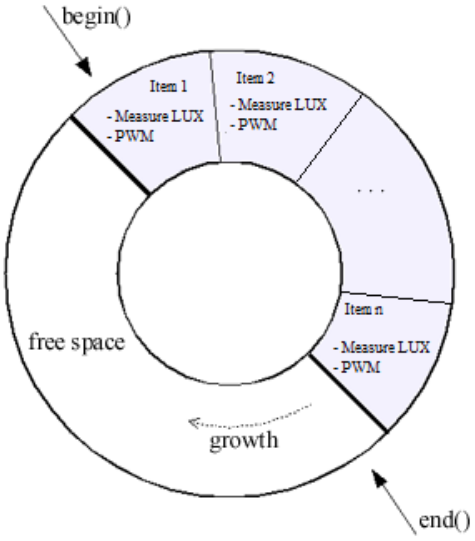


Fig. 19: Circular Buffer Implemented

4) Server Operations: The server can handle multiple clients and request. These requests follow the rules given in the project statement.

In order to save memory in the server, only the available nodes have the structure allocated and the requested statistics (energy consumption, comfort error and comfort flicker) are calculated cumulatively (in one variable for each).

When the client is asking for the current measure illuminance or current duty cycle (PWM), in percentage, of desk i, the server gets the value of the last positions of the circular buffer with the address i. When the request of the client is the current occupancy state at desk i, the value of the reference illuminance is consulted. If this illuminance value is 50, the desk is occupied (1), if the reference illuminance is 20, the desk is free (0). When the client requests the current illuminance lower bound, or the external illuminance, or the illuminance control reference of desk i or the elapsed time since the last restart, the corresponding variables stored in the data structure with address i are read and sent for the client in a proper message. If the client asks for the power consumption, this is calculated using (29) and knowing that the power is the derivative of the energy in order to the time.

$$P_j = \frac{dE_j}{dt} = P_{j-max} d_{\%}, \tag{29}$$

where P_j is the power consumption of desk j, E_j is the energy consumed at desk j, P_{j-max} is the LED nominal power and is equal to 1W and $d_{\%}$ is the PWM value in percentage. That means, when the value of PWM is maximum (equals to 255), the power consumption is also maximum and is equal to 1W. The client can also request the total power consumption of all desks and in this case, the server searches for the available nodes and calculated the sum of the power consumption of each node.

When the client solicits the energy consumption

$$E_j = P_j \sum_{i=1}^{N} d_{i-1}(t_i - t_{i-1}), \tag{30}$$

where N is the number of measures/samples, P_j is the maximum power (1W), d_i and t_i is the duty cycle and the time of measure i, respectively, or the comfort error

$$C_{error} = \frac{1}{N} \sum_{i=1}^{N} max(l_{ref}(t_i) - l_{meas}(t_i), 0), \tag{31}$$

where $(l_{ref}(t_i)$ and $l_{meas}(t_i)$ is the illuminance of reference and measured, respectively, or the comfort flicker

$$C_{flicker} = \frac{\sum_{i=3}^{N} f_i}{N} \tag{32}$$

where f_i is given by

$$f_i = \begin{cases} (|l_i - l_{i-1}| + |l_{i-1} - l_{i-2}|)/(2T_s), \\ \qquad if (l_i - l_{i-1}) \cdot (l_{i-1} - l_{i-2}) < 0, \\ 0, \qquad otherwise \end{cases} \tag{33}$$

the server already stores the accumulated sum of these performance metrics and only needs to divide by the number of measure (N). Posteriorly, the result is sent to the client.

When the client sends a message to request the restart of the system, all the information of all nodes is "clean". If the client request the values of measured illuminance or PWM recorded during the last minute, the information saved in the circular buffer is sent back to the client.

When the user wants to receive a stream of some measured variable, the information about the stream requested is stored in a linked list that stores the number of the desk requested, the variable and the number of a sample. Periodically, the server verifies if there is a request for some stream. If positive, the server sends the value of the variable that it's stored in the last position of the circular buffer and the time of that sample. When the server receives the message to stop the stream, the list is verified to determine if the stream exists and in a positive case, the stream information is deleted from the list.

5) Shared Variables - Mutual Exclusion: The array of pointers containing the data structures of the available nodes is written by the thread that reads data from the Arduinos and is read by the thread that handles the requests of clients, i.e., different threads can access to the same resource in memory. So, when the server uses the database is entering in a critical region. For this reason, the server needs to have protection for collisions when it wants to read and write data at the same time. Mutual exclusion mechanisms have to be used in order to prevent concurrent access to critical regions.

In order to accomplish this demand, it's implemented one Mutex. If a Mutex is unlocked, the thread can enter in the critical region and locks the resource for all the other threads. If a Mutex is locked and a thread wants to enter in the critical region, this one needs to wait until the Mutex is unlocked. Always that a thread wants to read or write in the database, a Mutex is locked and only the current thread can execute the critical region.

6) Communication with Clients: When a client wants to connect to the server, it starts a TCP connection to the port 17000. When the server receives a message on this port, it knows that a new client wants to establish a connection. Then a socket is created in the server that is responsible for all the messages received and sent to the client.

The server's communications were implemented with two different C++ classes, called *tcp_server* and *connection_client*. The first one constructs an acceptor that listens to TCP connections on the port 17000 and calls the constructor of *connection_client* that creates a socket. This second class handles all the requests of clients. After the socket is created, it's initialized an asynchronous accept operation to wait for a new connection of a client. When a connection is established, the server handles the request by calling a function of class *connection_client* that sets a timer of 1 second and during this period the server is asynchronously waiting for the client to establish a TCP connection between these two. When the connection is correctly established, the server sends a message of welcome to the client and then waits for requests of clients. At the same time, the server continues to accept connections of new clients for the port 17000.

7) Fault Tolerance: A Fault Tolerance policy is implemented in the server in order to be impossible the crash of the server and this can continue its operation when there is some error or system failure, giving only a warning or an error message, rather than failing completely.

The messages sent by the clients are always checked if they have the correct format, if is a valid request and if the desk requested exists. In this way, the server prevents the access to memory that is not allocated or does not exist and also prevents handle requests that are not correct.

Always that an error or an anomaly in the process is detected when running a determine function, an error signal is returned and it's handled in order to try to solve the situation. If the error happens when the server is handling the request to the client and because of that the request cannot be correctly answered, the server sends a message to the client in order to warn the client about the error.

O. Protocols Implemented

1) Between Arduinos and Raspberry Pi: The Raspberry Pi works always as a slave node, so it only can receive messages from the Arduinos and cannot send messages back. In order to do not send a big number of bytes, when sending a string, all the messages are converted in the type *bytes*. The first byte corresponds to the type of message and the second byte has the address of the node that sent the message. All the bytes are stored in big-endian, i.e., the most significant byte of a sequence is stored first (at the lowest address).

There are three different types of messages:

- Type *F* (10 bytes): contains the message type in the first byte, the sender address in the second byte, four bytes for the initial time and other four bytes for the external illuminance. It's sent after the calibration process.

0	1	2	3	4	5	6	7	8	9	Vector of bytes
Type F	Adrr	Initial Time				External Illuminance				Content
int	int	unsigned long				float				Variable type

Fig. 20: Type *F* message scheme

- Type *C* (7 bytes): contains the message type in the first byte, the sender address in the second byte, four bytes for the control illuminance and one byte for the reference illuminance. It's sent after the consensus algorithm.

0	1	2	3	4	5	6	Vector of bytes
Type C	Adrr	Control Illuminance				Reference Illuminance	Content
int	int	float				int	Variable type

Fig. 21: Type *C* message scheme

- Type *I* (7 bytes): contains the message type in the first byte, the sender address in the second byte, four bytes for the measured illuminance and one byte for the PWM value. It's sent every 10 milliseconds, after the control interruption.

0	1	2	3	4	5	6	Vector of bytes
Type I	Adrr	Measured Illuminance				PWM	Content
int	int	float				int	Variable type

Fig. 22: Type *I* message scheme

2) Between Arduinos: The messages exchanged between Arduinos are also sent in the format of *bytes* for the same reason as the messages sent to the Raspberry Pi. Each message has a header of three bytes where is saved the type of message and another byte to store the sender address.

The messages between Arduinos can be sent during the calibration process, during the consensus algorithm and the reference illuminance change in one node and the nodes need to run the consensus algorithm again. There are 6 different message types:

- Type *RML* (READ_MY_LED): during calibration to warn that the sender node has the LED turn on and the other nodes can read the illuminance value;
- Type *RYL* (READ_YOUR_LED): during calibration to notifies the receiver node that the sender node wants to read the illuminance value to calibrate the gains;
- Type *CYL* (CONF_READ_YOUR_LED): during calibration to warn the node that has the LED turn on that the illuminance value was read;
- Type *DR_* (DONE_READ): notifies the node to end of calibration;
- Type *TC_* (TURN_ON_CONSENSUS): to start the consensus algorithm when the reference illuminance of the

sender node change and the network needs to calculate the new values of consensus;
- Type *SRD* (SEND_RESULT): during consensus algorithm to send the result of iterations to the other nodes.

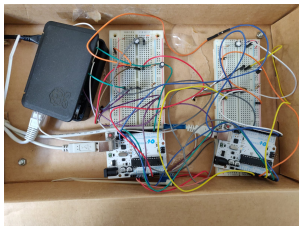
IV. Experimental Results

A. Circuit Description

To simulate the office, a box was adapted and the luminaires were enclosed there (Figure 23(a)). The sensors are located on the top of the box in order to have a model closer to the reality and a window was created to see the interior (Figure 23(b)). The electrical circuits are in a different box attached to the main one (Figure 23(c)).

(a) Box exterior with window

(b) Box interior



(c) Secondary box with circuits

Fig. 23: Box used in the model

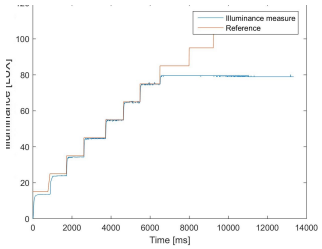
B. First Stage

In the first stage, only the controller of one Arduino was implemented. In the figure 24 the emission/reflection path is represented.

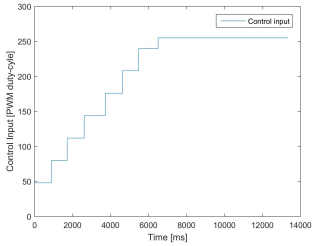
Fig. 24: Light reflection path

The steady state characteristic of the system was already stated and shown. Every time the system begins, K and o in (24) are calculated. For this time values obtained were $K = 0.31$ and $o = 0.01$. From the external illuminance value, one can conclude that the box is well isolated, measuring only 0.01 LUX when the LEDs are off.

1) Feedforward: The first part to test in the system is the feedforward controller acting alone. As predicted before, it will be a fast controller but not very accurate in some situations. To test that, the reference is a signal that when a button is pressed increases 10 LUX and the measured illuminance and control input are analyzed.



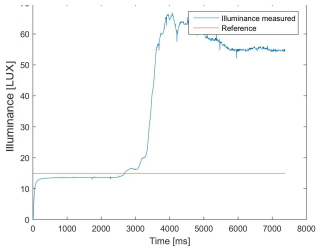
(a) Measures and reference



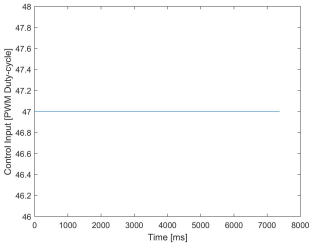
(b) Control Input

Fig. 25: Results for feedforward control

In the figure 25 the results of the analysis are shown. For values of reference of until 80 LUX, the system performs fairly good. The measure has a behavior very identical to the reference. However, for the lowest values of reference, one can see that the measured value has a difference compared to the reference and can't correct it. This is characteristic of a feedforward controller since it can't generate control input to compensate the error. Furthermore, after 80 LUX the illuminance becomes always constant meaning that the actuation limit was reached, as seen in the figure 25(b). This only means that no value bigger than 80 LUX can be reached. Considering only the behavior until the 80 LUX the average mean squared error between the output of the simulator and the measurements can be computed. The value in this case is 1,4342 LUX which is very small. To check how robust is this controller to disturbances, the flash of a camera's mobile phone was used to create one.



(a) Measures and reference



(b) Control Input

Fig. 26: Results for feedforward control with disturbance

Through the analysis of the figure 25(a) it can be seen that the flash was inside the box from the 4^{th} second. If the controller was robust to the disturbance it should act upon the LED in order to try to maintain an illuminance near the reference. In this case, the LED intensity should be lower when the flash is in the box because it makes the illuminance level bigger. However, in the figure 26(b) the control input never changes which means that the controller can't react to the disturbances. To sum up, a feedforward controller would be

good enough if the system was disturbance free. Since those types of system are rare and this one is not one of them, another technique needs to be used: feedback.

2) Feedback: As seen in the implementation section, the proportional integral feedback controller can solve the problems that were stated for the feedforward controller. To see how, the system was tested with only the proportional part and both. The respective gains were empirically determined by many trials which made possible to get an idea of how different gains influence the system. Firstly, the system was tested to see the differences between having integral part or not.

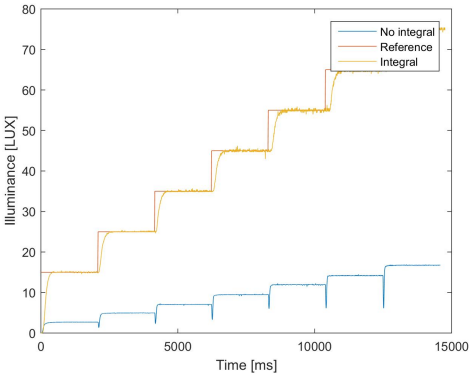
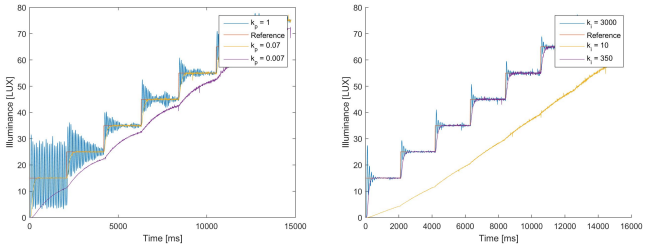


Fig. 27: Difference between using or not the integral part

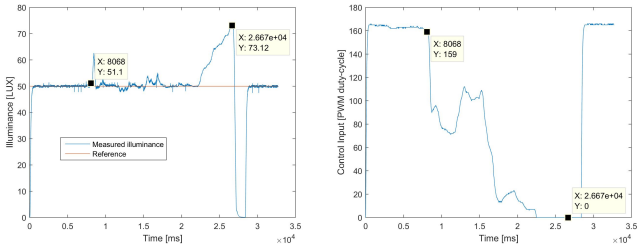
From the figure 27 one can see that with only proportional part the controller cannot drive the system to the reference, as it was predicted in the implementation. The proportional gain is not enough to, multiplied by the error, create an input that is high enough to make the system converge to the reference. By joining both parts, one can see that the system follows the reference as intended. In order to get the best response possible, the integral and proportional gains were tuned.



(a) Changing k_p (b) Changing k_i

Fig. 28: Tuning of feedback parameters

In the figure 28(a) one can see the influence of changing the proportional gain with the integral fixed. If the value is too low the system does not have time to converge and if it too big, then the oscillations are big and it may not stabilize for certain values (ref = 15 LUX). For the integral part, the case is similar as can be seen in figure 28(b). To check if this controller really outperforms the feedforward one, the robustness to disturbances must be analyzed. The reference is fixed in 50 LUX and during the process, the flash of the mobile phone is once again used to simulate a disturbance.



(a) Measures and reference (b) Control input

Fig. 29: Feedback results with dsturbance

In the set of figures shown in 29 the behavior of the system when a disturbance happens is shown. The pins in the images represent the moment when the light from the flash starts to be detected by the LDR and when it stops. Approximately until the 22.5 seconds, one can see from figure 29(a) that the system is able to follow the reference even with the disturbance. During the same time, the control input in figure 29(b) adapts to create the proper response. What can also be seen through the figures is that, after the disturbance stops, the system takes a while to converge again. This is called the windup effect. In order to have information about the convergence speed and other measures, the controller was tested with a reference that changed between 20 and 50 LUX when the button was pressed and the time response was analyzed.

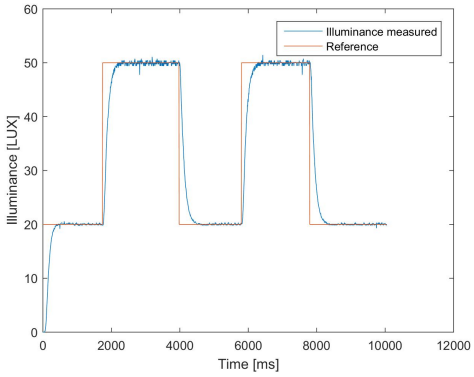


Fig. 30: Step response analysis

From the figure 30 some conclusions about the characteristics of the system can be taken. Firstly, no overshoot is seen in all transitions. This means that the damping ratio is greater or equal to 1. The settling time can also be analyzed and it is approximately 0.3 seconds for all the transitions. With the analysis from this subsection, it can be concluded that the feedback controller suits this problem better than the feedforward one mostly because it can react to disturbances and can correct the values when a model error is made. However, it can be very improved using the feedforward part and some extras stated in the implementation.

3) Improvements: The first improvement that can be done to try to make the system faster is to join the feedforward

controller to the feedback. This means that the feedforward will be always there and the feedback will compensate the differences. That way it is expected that the system will converge faster to the reference possibly creating overshoot.

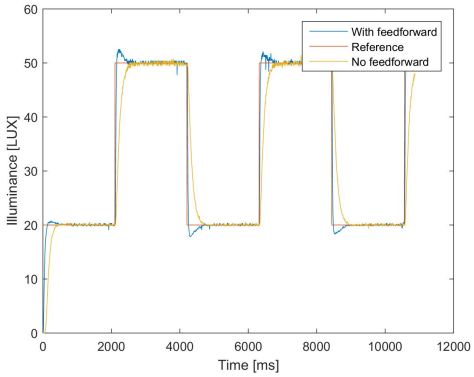
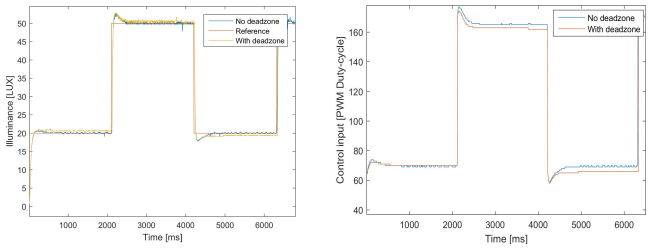


Fig. 31: Difference between the usage or not of feedforward

From the figure 31 one should understand that the predicted behavior is verified. The system now has an overshoot and therefore, the damping factor is lower than 1. Since the overshoot is not that big, approximately 5 percent of the reference value, the settling time becomes smaller, approximately 0.15 seconds. From this, it can be said that the system becomes faster.

The second improvement that can be made is trying to reduce the noise that is visible in every test shown until now. This happens because the system tries to react whenever there is an error, even if it 0.1 LUX. Given that it is very difficult for the LDR to measure exactly the value of the reference and it has some noise associated, a margin can be given to the error by a deadzone.

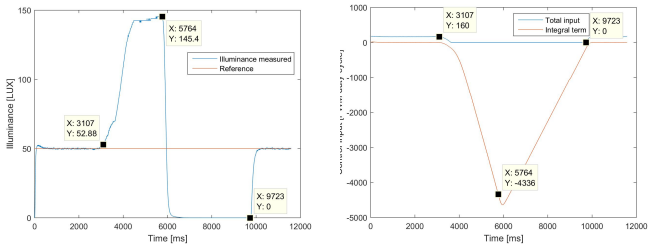


(a) Measures and reference (b) Control input

Fig. 32: Difference between using or not deadzone

From the figure 32(a) one can't say that the situation became better. It seems as if the noise is more or less the same. However, looking at the figure 32(b) the control input seems to have fewer changes. The noise seen before was, therefore, created by the measurements of the LDR and not because of different inputs. In this case, this method doesn't seem that useful but it can be with different conditions when the noise is more noticeable and makes the input needs to change more.

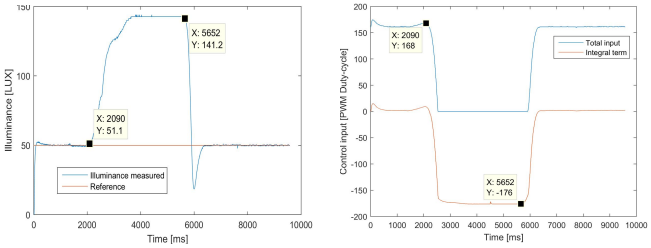
One of the most important improvements that can be made is to solve the windup effect that happens accordingly to the set of figures 29. The reason why it happens was already discussed in the implementation section, but briefly, the integrator keeps accumulation error values with no limits and after it will be hard to get back to 0. To show that happening a flash was pointed to the LDR during some time and after turned off, always with a constant reference.



(a) Measures and reference (b) Control input

Fig. 33: Windup effect

The results for the system when the windup happens are in the set of figures 33. They show three pins that are representative of when the disturbance started when it ends and when the system started to converge again. In the figure 33(b) one can see how the integral term of the controller keeps growing and the time that it takes to go to 0 again, 4 seconds for a 2 seconds disturbance. For the sake of the controller velocity, the anti-windup addressed in the implementation must be used.



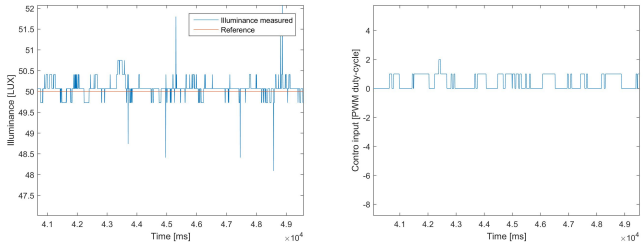
(a) Measures and reference (b) Control input

Fig. 34: Results with anti-windup

In the set of figures 34 the response of the system is shown after applying the anti-windup method. From the figure 34(b) one can conclude that the method successfully limits the integral term not letting it reach enormous values. With this, the system will be faster to react to the disturbances, less than 1 second. The anti-windup effect is consequently a big improvement on the controller.

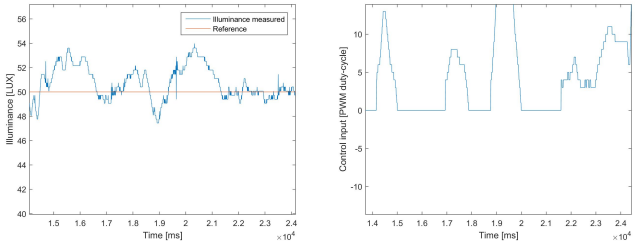
As mentioned in the implementation section, another problem that happened was flickering. It happens when the disturbance that is happening is almost sufficient to make the LED turn off. In those cases the LED turned on and off a couple times until the disturbance was big enough to make it turn off at all or small enough needing the LED to be on to achieve the reference illuminance value. This event is hard to translate in a plot because is more perceptible by looking at the LED.

In figure 35(b) one can see that for the time interval, the control input has a lot of changes between 0 and 2 creating the flicker .



(a) Measures and reference (b) Control input

Fig. 35: Results without hysteresis



(a) Measures and reference (b) Control input

Fig. 36: Results with hysteresis

The set of figures 36 show the results after applying the hysteresis addressed in the implementation. One can see in the figure 36(b) that no variations of the control input near 0 exist. This is, therefore, a detail that doesn't make much of a difference but makes the system more pleasant when disturbances are present.

Finally, with all the improvements stated, the final controller is determined. It comprises the feedback and feedforward controllers, the simulator and the additional components. To check how it performs, a reference that changed between 20 and 50 LUX was given and some disturbances were simulated. The results are in the figure 37.

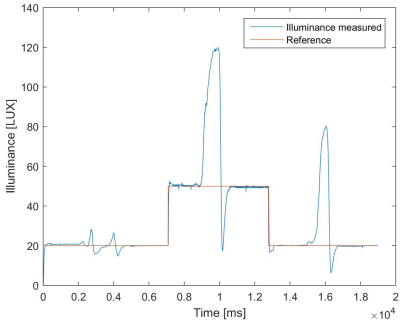


Fig. 37: Results for full controller

From the results, the controller produced has a very good performance. Not having big overshoots - maximum 2.5 LUX - and having short 95% settling times - around 0.1, 0.15 seconds it shows an adequate behavior in a disturbance free environment. Considering disturbances, it can have a pretty fast answer to them taking approximately 1 second for a big one. Therefore, the task of implementing a capable controller is completed.

4) Computational factors: To finalize the analysis of the control algorithm developed, it is important to check its complexity and computational time. Firstly a sampling period of 10ms is intended. To check whether this value corresponds to the truth the time elapsed between two interruptions is obtained. This is done for many following interruptions to guarantee the equality between them.

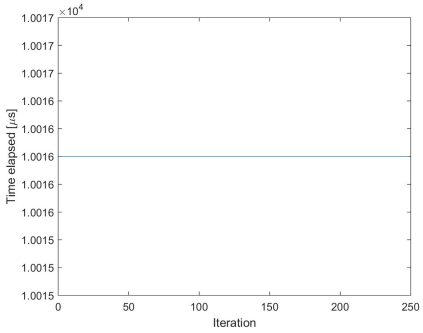


Fig. 38: Time elapsed between interruptions

From figure 38 one can affirm that the elapsed time between interruptions is always the same and equal to $10.016ms$. It only has a deviation of $16\mu s$ from the wanted value. Now, in order to guarantee that the controller can work properly, all the computations that are occurring in the program must take less than 10ms being all the values ready for the interruption where the feedback control takes place. In order to analyze that, the approximated elapsed time between beginning and conclusion is obtained.

Task	Elapsed Time [μs]
Feedback controller(Interruption)	141
Prints to Serial	1315
Controller algorithm	2284
Loop	4989

TABLE II: Time elapsed in different tasks

From the times shown in the table II it can be concluded that the program has plenty of time to run between two interruptions. Firstly, the feedback controller lasts in average 141 μs which means that the jitter is very low because almost no deviation from the wanted sampling time is seen. The Baud rate chosen is the maximum possible (2000000 bps) in order to be the fastest possible. The prints addressed in the table are four with the conversion from voltage to LUX in the middle since these are the ones needed to have good information in the Serial plotter. All in all, the system satisfies the computational requirements to work properly.

C. Second stage

For the second stage, the full distributed system represented in figure 23(b) needs to be tested. Firstly, the calibration must be done in order to determine the values of K and o. Even though the calibration is scalable for 127 different nodes, the system has only 2 and, therefore, only two gains shall be

computed by each node. In a particular test, the equation that translates the system after calibration, based on (23) is

$$\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} K_{1,1} & K_{1,2} \\ K_{2,1} & K_{2,2} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} + \begin{bmatrix} o_1 \\ o_2 \end{bmatrix} \Leftrightarrow \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} 0.24 & 0.13 \\ 0.07 & 0.25 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} + \begin{bmatrix} 0.35 \\ 0.28 \end{bmatrix} \quad (34)$$

The values obtained are according to the expected since both LEDs have a higher effect on the associated LDRs than on the LDR of the other node. This is predictable because of the positions of each node (Figure 23(b)). Also, the interaction between different nodes is detected because of $K_{1,2}$ and $K_{2,1}$ are not 0. Then, the controller needs to be tested.

1) Distributed vs. Non Distributed Controller: Performance Metrics: As stated in the implementation, a distributed controller, based on the algorithm consensus, is implemented. Its performance needs to be compared with the performance of having individual controllers in each node. In order to evaluate the performance, three different measures are used. The energy consumed by each luminaire, given by (30), the Comfort Error, given by (31) that evaluates the periods of illuminance below the lower bound and the Comfort Flicker, given by (32), which evaluates the changes in illuminance (ups-and-downs) while the reference value of illuminance is constant.

The system was tested and the following results were obtained using the server implemented. In this test, it was not used external disturbances. It was evaluated the performance of the distributed controller and of the simple controller for desk 1 and desk 2. The results obtained are the same for both desks using the same controller since the components are equal. In figure 39 and figure 40 are plotted the measured and reference illuminance and the respective PWM value for desk 1 using the distributed controller and the non distributed controller.

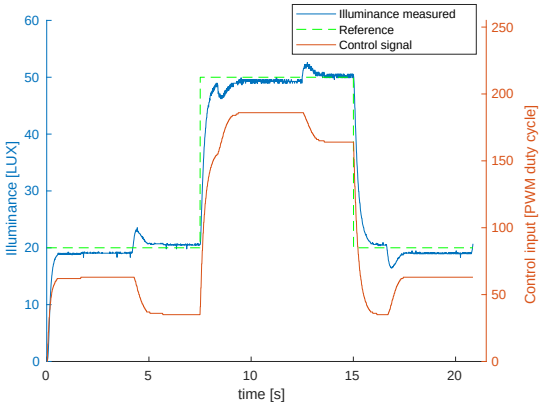


Fig. 39: Measures of desk 1 using a distributed controller

Then the energy consumed by the luminaire of desk 1 was calculated for each instant of time. The results are registered in figure 41 for a distributed controller and a non distributed controller, respectively.

As it was expected because the energy consumed corresponds to the energy accumulated, the energy value is always increasing, because the LED is always turn on. When the PWM value is smaller, the instantaneous energy consumption

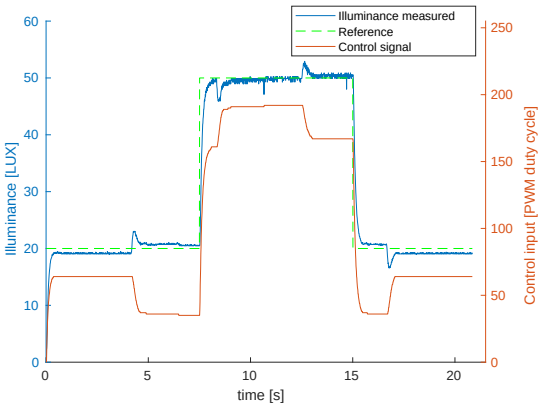


Fig. 40: Measures of desk 1 using a non distributed controller

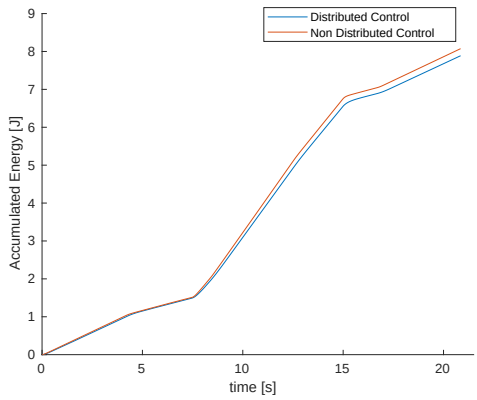


Fig. 41: Energy consumption of desk 1

is also smaller, as it's possible to observe in figure 41, because when the PWM value is smaller the slope of the line is smaller, i.e., the energy accumulated increases more slowly comparing when the PWM value is higher that corresponds a higher instantaneous energy consumption.

The other performance metric was evaluated. Using the same data, the comfort error was determined and the results are represented in figure 42.

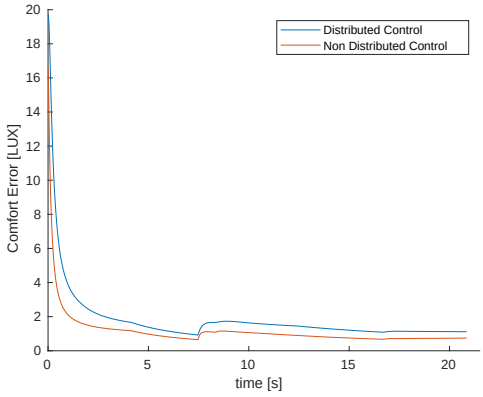


Fig. 42: Comfort Error of desk 1

The comfort error tends to decrease and stabilize because the system follows the reference, oscillating between this

value, so the difference between the reference (constant) and the measured illuminance is small, so the comfort error decreases. When the value of the reference changes from 20 LUX to 50 LUX, the comfort error suffers an increase because the reference has a higher value than the measured illuminance. When the reference changes from 50 LUX to 20 LUX, the comfort error decreases because the measured illuminance is above the lower bound.

The last performance metrics evaluated was the comfort flicker that is registered in figure 43.

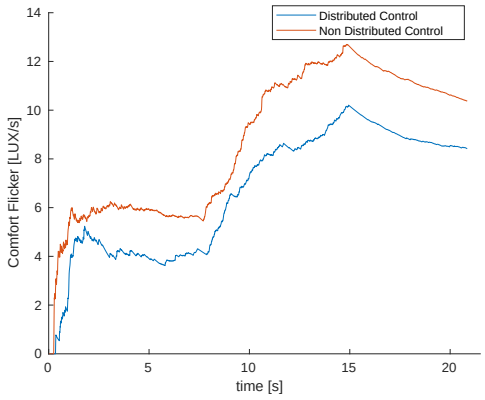


Fig. 43: Comfort flicker of desk 1

When the reference illuminance is 20 LUX, the value of the comfort flicker stabilizes approximately on 6 LUX/s using a distributed controller and around 4 LUX/s using the non distributed controller. When the reference illuminance increases to 50 LUX, the comfort error also increases a lot due to the fact that for the higher illuminance the noise measured by the LDR increases and there are more oscillations of the signal measured. This provokes an increased of the comfort flicker as it's possible to observe in figure 43.

2) Uneven costs: In order to better compare the distributed and non distributed control, two different lamps are considered. In the first desk, the source wastes 8 times more energy than the second desk. This is considered in the Consensus algorithm as the cost of the node. A similar test to the previous one is made to see the differences in the wasted energy and behavior of the system considering the costs

$$c = \begin{bmatrix} 8 \\ 1 \end{bmatrix} \tag{35}$$

instead of using 1 for the two costs.

In the figures 44 and 45 the measurements and actuation with the distributed controller are represented. The desk 1 shows levels of actuation lower than before. This happens because in this case providing light by this desk is far more expensive. To compensate this it should use the light source in desk 2 to maintain the reference values of illuminance, having its actuation in the maximum value most of the time. This makes the illuminance measured in 2 to be a lot greater than the reference values but since they are only lower bounds for the algorithm, it is a feasible solution. This may not be the best solution in terms of comfort but it is in terms of energy saving.

To understand that the values of the energy consumption in both desks is obtained and showed in 46 and 47 respectively. For the first desk, the values are really big when no distributed controller is used because it doesn't take into consideration the cost of the energy. The opposite happens when the distributed controller is used, it can be seen that the values of consumption become really smaller. The desk 2 shows a really low value of energy when using the local controller. It makes sense because it has no need to use more power to compensate the other luminaire. With the distributed controller the energy grows because it is using more power to compensate the other desk. In tables III and IV the scalar values of the energy wasted are shown. With the distributed approach a saving of around 15J was done which is very positive.

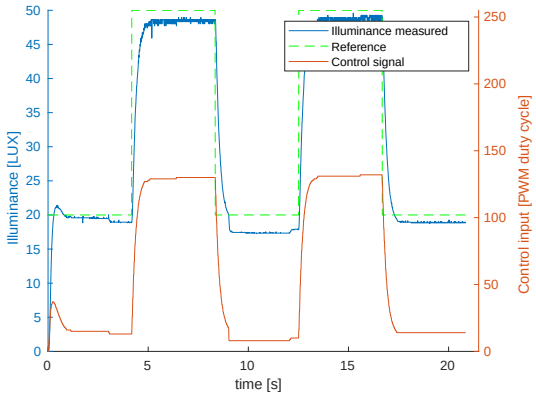


Fig. 44: Measures of desk 1 using the distributed controller

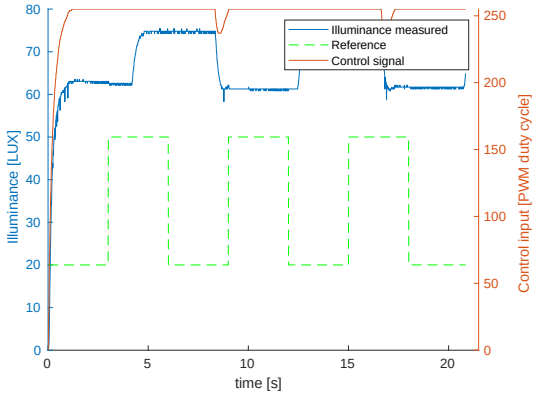


Fig. 45: Measures of desk 2 using the distributed controller

Desk	Energy
Desk 1	68.2140
Desk 2	7.1845
Total	75.3985

TABLE III: Energy consumption of the Non Distributed System

3) Distributed Controller Testing: After the analyses and evaluation of the performance metrics comparing a distributed controller with a non distributed controller, it's possible to conclude that the first one has some advantages because each

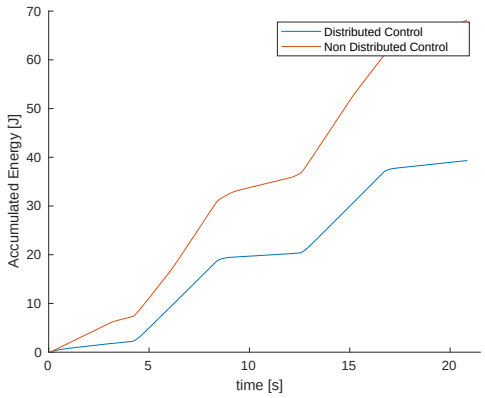


Fig. 46: Energy consumption of desk 1

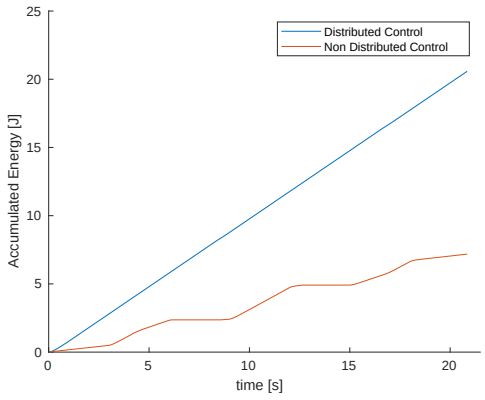


Fig. 47: Energy consumption of desk 2

Desk	Energy
Desk 1	39.344
Desk 2	20,5962
Total	59.9402

TABLE IV: Energy consumption of the Distributed System

node is cooperating to achieve the goal of ensuring that the minimum illuminance at each desk is guaranteed using less power consumption. Analyzing figure 41, the energy consumed by the distributed controller is less than the individual controller.

Then it was performed a test using the distributed system, where some external disturbances were done in order to see the system's reaction. The measured and reference illuminance and the PWM value of each desk are registered in figure 48 and figure 49.

Analyzing the previous figures, it's possible to see that the system responds in a proper way to all the disturbance and changes. During the 10 and 20 seconds, the reference illuminance changes and the control signal varies according expected. When the reference illuminance of desk 2 changes from 20 to 50 LUX, the PWM value of this desk increases in order to ensure the lower bound illuminance. At the same time, desk 1 that maintains its reference decreases the value of PWM. This occurs because as the LED illuminance of desk 2 increases and as it has influence on LDR of desk 1, the lower

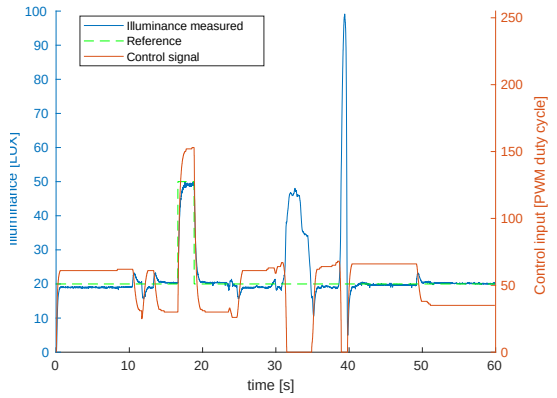


Fig. 48: Measures of desk 1 using a distributed controller

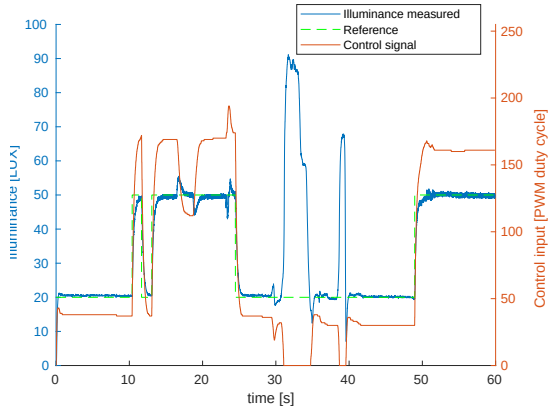


Fig. 49: Measures of desk 2 using a distributed controller

bound of desk 1 is guaranteed using a small PWM value. An equivalent behavior occurs when the reference illuminance changes from 50 to 20 LUX.

During 30 and 40 seconds, some external disturbances were performed. The window was open and a light was pointed inside the box. As it's possible to see, there is a big increase in the illuminance measured by both LDR. As the lower bound illuminance is guaranteed (using an external light), the system reacts and the control signal changes to a PWM value of 0. That means that the LED is turned off and the external light is enough to ensure the minimum illuminance in both desks.

When the window is closed and there are no high external disturbances, the controller responds increasing the PWM value in order to ensure the minimum illuminance.

Then the performance metrics are determined. The results are registered in the following figures.

The energy consumed in desk 1 during 60 seconds was approximately 11 J and in desk 2 approximately 19 J. The energy consumed by luminaire 2 is higher because the PWM value of desk 2 is higher than PWM value of desk 1 in almost every time (except during 40 to 50 seconds). It's also possible to see that when the LED is turned off, i.e, the PWM value is 0, there is no energy consumption.

The comfort error decreases when there are external disturbances because the lower bound illuminance is ensured using the external light. The comfort error of each desk is similar.

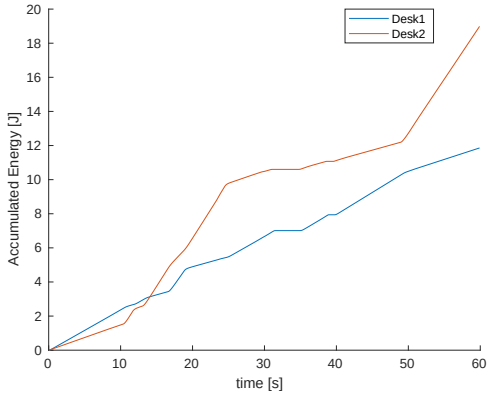


Fig. 50: Energy Consumption of the system

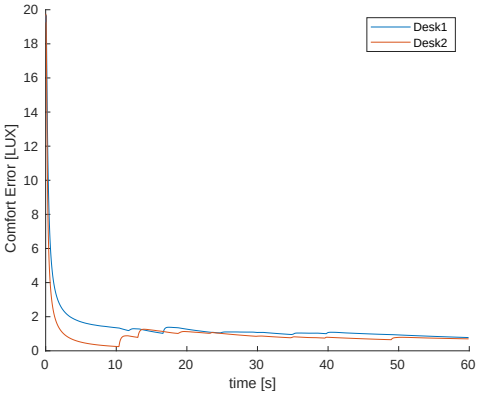


Fig. 51: Comfort Error of each desk

Analyzing the comfort flicker, it's possible to see that the value of desk 2 is much higher than the value of desk 1 due to the fact that LDR of desk 2 reads higher illuminance values and to higher illuminance values the LDR presents more noise. This provokes an increase of the comfort error.

The external disturbance does not have a big influence on the evolution of comfort flicker. In this test, the external light was constant (without any effect of flickering) and the external disturbance was high only for 2 seconds. So, as it's possible to see in figure 52, the external disturbance does not have a big influence on the comfort flicker.

4) Communications' Performance: The communication between Arduinos and the Raspberry Pi uses the I_2C bus. As seen before, the Arduinos can send and receive messages. However, the Raspberry Pi can only receive messages from the I_2C bus. The communications of Arduinos were implemented using *WSWire* library that uses timers to communications.

The network only has one bus available to send all the messages and it only can be one message on the bus each time. This leads to conflicts between messages that may be sent at the same time. The library used verifies (through a flag) if the bus is occupied by another message. If the bus is not occupied the message can be sent. Otherwise, if the bus is occupied, the message cannot be immediately sent. This message is stored in a queue and the program has to check periodically to verify when the bus is free. When the bus is free again, this message can be sent.

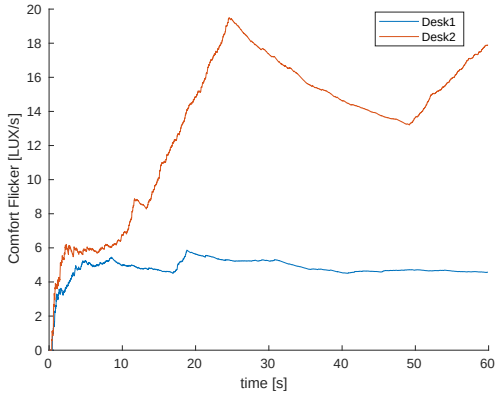


Fig. 52: Comfort Flicker of each desk

In order to determine the time necessary to send a message between Arduinos, a simple program was done where one Arduino sends a message of 8 bytes (the same length of messages used in the protocol implemented) and waits for a response of acknowledgment. Then it computes the time between the moment that it sent the message and the moment that it receives the acknowledgment. The other Arduino receives the message and sends immediately an acknowledgment, so there is almost no lost time to computing data. This program was run several times and an average of the time to send and received a message was $1817\mu s$. The time needed to send one message corresponds to half of this value, i.e, the time that it takes to send a message of 8 bytes is $908.5\mu s$.

The message sent has a length of 8 bytes. However, the program sends more than 8 bytes because all the messages have a header with a start and an end condition, the slave address, a Read/write bit to identify if the message to send data or to request and acknowledgment bits between every byte of data. At least a message with 8 bytes (64 bits) of data needs to send 83 bits (if the address field has 7 bits as in this case).

Start	Addr	R/w	Ack/ Nack	data	Ack/ Nack	data	Ack/ Nack	Stop
1 bit	7bits	1 bit	1bit	8bits	1bit	8bits	1bit	1bit

Fig. 53: Message I_2C

The bit rate of the I_2C is $100kbps$. This means that in one second 100000 bits are sent through the bus. So, theoretically, a message of 8 bytes is sent in

$$t = \frac{83 \cdot 1}{100000} = 0.83\text{ms}. \tag{36}$$

This result is similar to the test performed. In reality, the message takes more time, because there is time lost between receiving, processing and sending an acknowledgment. Also may have delays in communications due to wires used.

The messages sent from the Arduinos to Raspberry Pi, theoretically, will take more time, since there is an integrated

circuit converting the 5V to 3.3V voltages. However, there is not a good way to test the time that it takes to send a message to the Raspberry Pi since the clocks are not synchronized and the Raspberry only works in slave mode.

V. Conclusion

The distributed controller proposed was implemented using the Consensus optimization technique and also the PI local controller. For the local controller, different techniques were used such as feedforward and feedback control, deadzones, anti-windup or hysteresis. It showed good performance and robustness in the tests performed. The distributed implementation suggested showed decent performance when tests were made, being able to drive the system for the references given, or to higher values when it was not possible. The LDRs used proved to be a not so good sensor since they created lots of noise many times. When the box was moved abruptly it could make the measures to become poorer and making it harder to have a proper controller. The comparison between the distributed and local controllers showed that the difference is not that big when considering equal sources, but, as predicted, the distributed one wastes a less amount of energy. This was more noticeable when sources with different costs were used. Also, the calibration system proved effective for two luminaires but it is prepared for all the possible 127.

A TCP server was implemented in the Raspberry Pi to store data used to monitor the performance and quality of the lighting system. It showed fault tolerance and capabilities of stream and multiclient besides being able to store the data properly. Some problems with this derived from the communication between the Arduinos and the Raspberry since it crashed sometimes due to electrical problems in the connection between them.

As work for the future, we propose trying to implement this controller in a system that uses more reliable forms of communication and more reliable sensors. This could be an improvement in the controller and server performance. Trying to experiment it in an environment with the 127 desks would be a very interesting thing.

VI. Project Division

All this project was made with common effort from both students. However, to better organize and solve the problem in time one was more responsible for the control and communication between nodes and the other took care of the interruptions, consensus algorithm and server. During the elaboration of every part the ideas were discussed and only after the implementation was done.

References

[1] Caicedo, David and Pandharipande, Ashish, *Distributed illumination control with local sensing and actuation in networked lighting systems,* 2013.

[2] Alexandre Bernardino, *Solution of Distributed Optimization Problems: The consensus algorithm,* November 14, 2018.

[3] Alexandre Bernardino, *Slides of the Lectures of Distributed Real-Time Control Systems,* Instítuto Superior Técnico, 2018.