Instituto Superior Técnico

Distributed Real-Time Control Systems

1st Semester 2018/2019

Master Degree in Electrical and Computer Engineering

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

---

Authors:

Lourenço Vaz Pato - 80931

Duarte Dias - 81356

Pedro Faria - 81468

Group 3

January 12, 2019

1 Introduction

In the last decade the use of LEDs has increased greatly. They have a lower energy consumption and are more versatile than the other common available technologies, incandescent and fluorescent light. Due to the versatility of LEDs - for instance, its small size and dimming capacity - LEDs are now the prime choice of illumination in homes, cars, offices and smart spaces. They allow a new approach to energy optimization and comfort control.

In this project, a smart office will be simulated. The office will be constituted by two desks that have an illuminance sensor, a luminaires made of an LED and communication capabilities, so that synchronization between desks can achieved. The objective is to design a real-time cooperative decentralized control system for the distributed illumination system. The office was simulated by a white opaque cardboard shoe box, and each smart luminaire by a Arduino and a breadboard with LED and LDR circuits (smart office only has two luminaires).

The goal of the distributed system is to minimize energy consumption while ensuring the user comfort. This can be achieved by controlling the dimming level of each LED such that the desks' illuminance is above a minimum illuminance level of comfort while avoiding flickering. Energy consumption is optimized using a distributed optimization algorithm. Using the luminaires' communication capabilities, negotiation messages are exchanged in order to distributively compute the optimal actuation of each LED.

This report was divided as follows: Section 3.2 (Initialization and Calibration) and Section 3.5 (I2C communications) were written by Pedro Faria. Section 2 (Background) and Section 3.6 (Data Server) were written by Duarte Dias. Section 3.1 (System Architecture), Section 3.3 (Local Control) and Section 3.4 (Global Control) were written by Lourenço Vaz Pato. Section 4 was jointly written by Duarte Dias and Lourenço Vaz Pato. All other sections were written by all the group members.

2 Background

Illuminance One of the key goals of this project is to achieve a desired illumination on each desk. To measure it, we used the measure of the illuminance. The illuminance measures the total luminous power (luminous flux) incident on a surface. Its unit is lux (lx), which is equivalent to the amount of lumen per square meter, in SI units $\mathrm{cd\,sr\,m^{-2}}$.

Arduino An Arduino UNO [4] board is a microcontroller based on the microchip ATmega328P [3] and has 14 digital I/O pins and 6 analog pins. Its operating voltage is 5 V, its clock speed is 16 MHz and it has 1 kB of EEPROM memory.

Raspberry Pi A Raspberry Pi [5] is a single-board computer, a complete package in a very small format. It includes a quad-core CPU, Wi-Fi and bluetooth connectivity, ethernet, USB and SD ports and 40 I/O pins.

PWM Stands for pulse-width modulation, it is a rectangular pulse signal where the duty-cycle is modified in order to control the power supplied to electronic devices. In this project we will be using PWM to control the intensity of the LEDs in the system in a range of $[0, 5]$.

I2C Stands for Inter-Integrated Circuit, it is a synchronous, multi-master, multi-slave serial communications bus used to transfer data between integrated circuits using 2 lines: SCL and SDA. In this work, we use the I2C protocol to implement communications between the Arduinos themselves and to the Raspberry Pi.

3 Development

3.1 Overall System Architecture

The system was assembled inside a shoebox with cut out holes for the USB cables as shown in Figure 1. To set up the simulated smart office the following components were used:

- PC w/ Arduino IDE software
- 2 Arduino UNO w/ USB cable
- Raspberry Pi 3B + charger + cable
- 2 mini breadboards
- 2 superbright LED's
- 2 LDR's (LDR GL5528)
- 2 100 Ω resistors
- 2 3.3 kΩ resistors
- 2 100 kΩ resistors
- 2 1 µF capacitors
- Several colored jumper wires

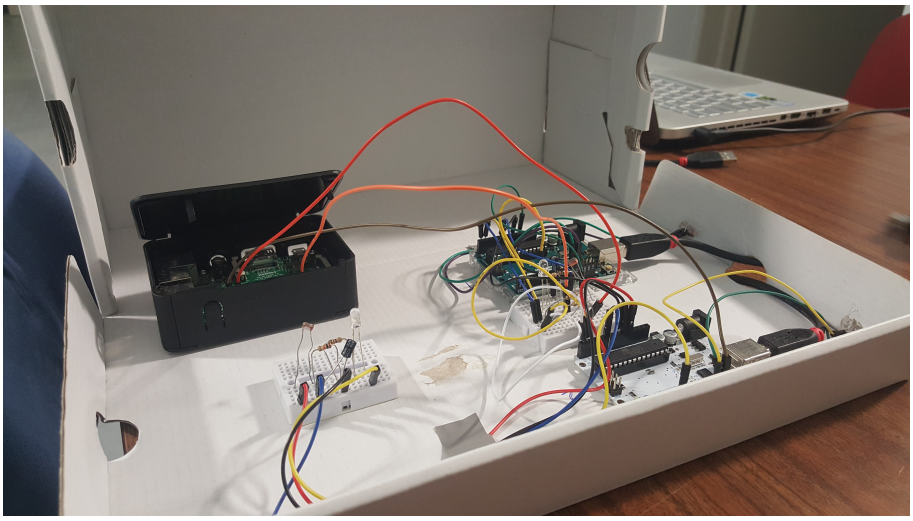


Figure 1: System overview

The overall architecture follows the architecture in Figure 2.

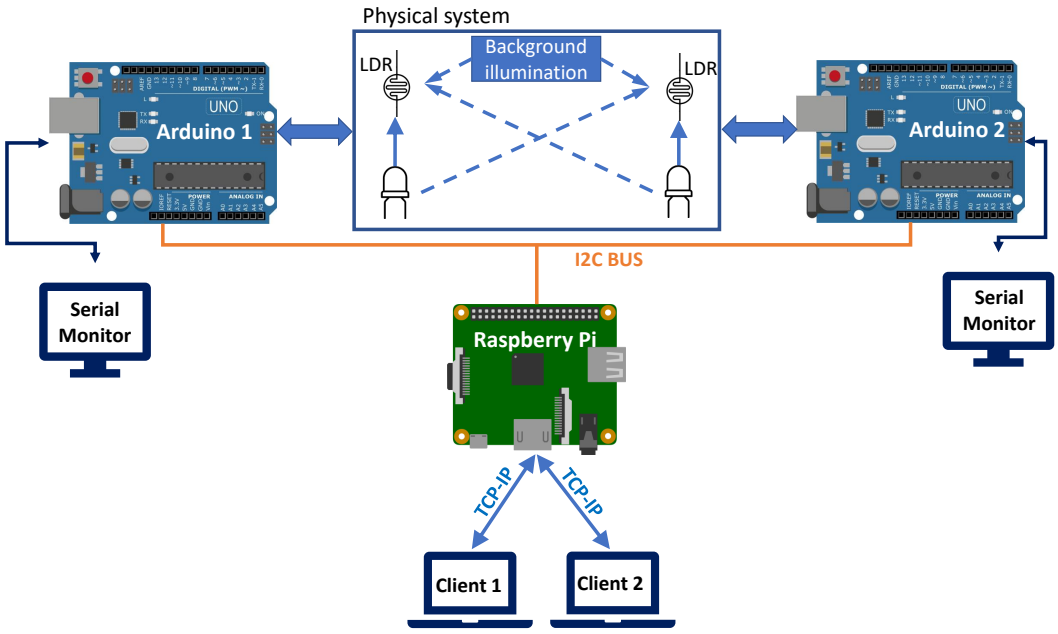


Figure 2: System architecture

Luminaire. Each luminaire kit is composed of an Arduino board, a LED circuit, a LDR circuit to measure illuminance and is connected to the PC console. In each desk the microcontroller implements a local controller that measures the illuminance and compares it with the illuminance reference. It then applies a corrective PWM signal

to the LED in order to obtain the desired illuminance value. The local controller is executed at a sampling rate of 200 Hz. At each cycle, the measurements are sent to the I2C Bus.

Serial monitor interface. Each Arduino is connected to the serial monitor from which the user can interact with the desk. When the system is started, the user is prompted to "press ENTER" to start the calibration routine. Then, the user should input the desired illuminance reference when the desk is occupied. The illuminance reference for an empty desk is 20 lux. During the execution the user can still interact with the system with the following commands:

- "1" - sets the desk state to occupied
- "-1" - sets the desk state to unoccupied
- "c <val>" - sets the local energy cost to the integer <val>
- "r <val>" - sets the local reference (when occupied) to the integer <val>

Calibration routine. At each system start a calibration routine is executed to measure the background illumination at each desk and to model the coupling gains between desks. The measure the coupling gains, a luminaire is turned on at a time, at full power, and the others will measure the illuminance on their desk. At the end of this routine, each computes the influence of the other LEDs' dimming on its own measurement. The calibration routine takes $N + 1$ seconds to execute, where N is the number of desks. Every time a new desk enters the network, the calibration routine is executed again to recalibrate the system.

Data Server. The Raspberry Pi is connected to the I2C bus and is sniffing the communications sent by the arduinos. It is collecting the measurements and signals applied by each node and computes statistics. The server then is also listening for requests from clients and replies with the requested statistic through TCP-IP.

3.2 Initialization and Calibration Procedures

3.2.1 Illuminance measurement system

The illuminance was measured via a LDR (Light Dependent Resistor) in a voltage divider circuit, as depicted in figure 3). The LDR is a non-linear element, whose gain (ratio of the illuminance with the measured voltage) depends on the operating point. However, the characteristic curve of the LDR (resistance vs illuminance) is linear in a logarithmic scale, as illustrated in its datasheet [2]. Since the LDR datasheet gives a range of resistance values for each illuminance, a set of experiments (measurement of the tension for various illuminances) was done in order to have a

one-to-one mapping for the tension and illuminance. By plotting $\log\left(\frac{V_{cc}}{V_{in}} - 1\right)$ vs $\log(L)$ we obtain a linear relation (figure 4), that is, identical increments in the LED actuation correspond to identical increments in lux, whose parameters are:

- For desk 1: $m_1 = -0.4934, b_1 = 0.6969$
- For desk 2: $m_2 = -0.4906, b_2 = 1.1133$

After determining the parameters m_i and b_i, we can then compute the illuminance at desk i as a function of the input voltage, V_{in}:

$$L_i = 10^{-\frac{b_i}{m_i}} \left(\frac{V_{cc}}{V_{in}} - 1\right)^{\frac{1}{m_i}} \tag{1}$$

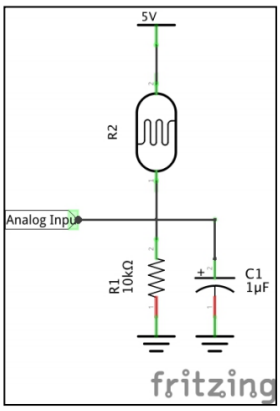


Figure 3: Illuminance reading circuit

3.2.2 System Dynamical Model

The illumination system dynamical model shows us how the LED actuation (system's input) influences the LDR measurement (system's output), in amplitude and time/frequency. As suggested in the handout, it was assumed that the system behaves locally as a first order system. The Laplace-Transform of the transfer function of this system is:

$$G(s) = \frac{G_0(x)}{1 + s\tau(x)} \tag{2}$$

where x is the desired final illuminance, $G_0(x)$ the static gain (ratio between illuminance [lux] and the applied PWM value) and $\tau(x)$ [s] the time constant.

To determine the parameters of the two dynamical models (one for each desk), a set of experiments was done: step changes in the LED actuation were performed and the voltage on the LDR measured.

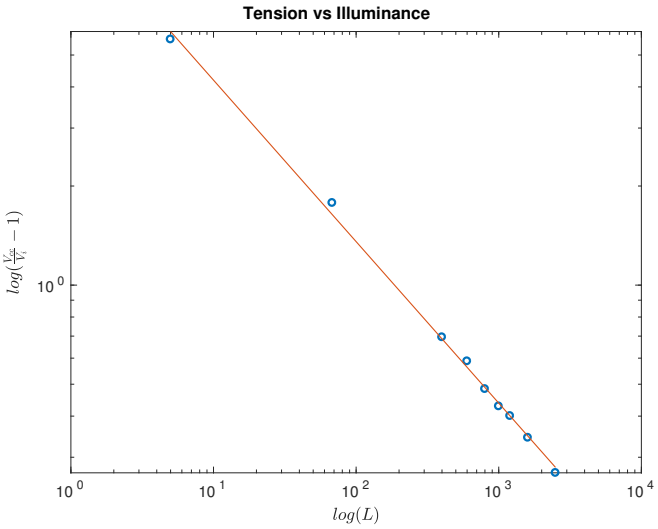


Figure 4: Mapping between tension, $log(\frac{V_{cc}}{V_i} - 1)$, and illuminance, $log(L)$

Using equation (2) we can compute the illuminance in lux and then the gain G_0. By plotting G_0 vs illuminance (figure 5), we obtain a linear relation, whose parameters are:

- For desk 1: $m_{G_1} = 0.1001, b_{G_1} = 44.9773$
- For desk 2: $m_{G_2} = 0.1227, b_{G_2} = 13.3269$

The static gain for desk i for desired illuminance x, $G_{0_i}(x)$, can then be computed by:

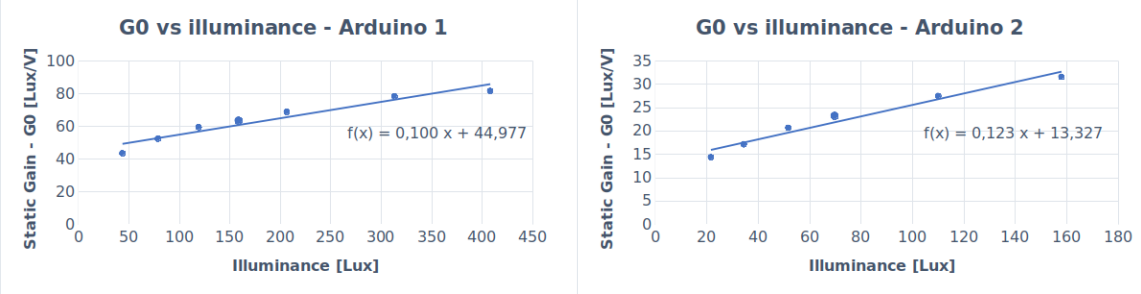
$$G_{0_i}(x) = m_{G_i}x + b_{G_i}. \tag{3}$$

After measuring the time constant, τ (time to reach 63.2% of the final step voltage value), for each one of the step changes, we plotted $\tau(x)$ vs illuminance (figure 6) and obtained a linear relation again, with the parameters:

- For desk 1: $m_{\tau_1} = -1.4449 * 10^{-5}, b_{\tau_1} = 0.0119$
- For desk 2: $m_{\tau_2} = -9 * 10^{-5}$, $b_{\tau_2} = 0.0237$

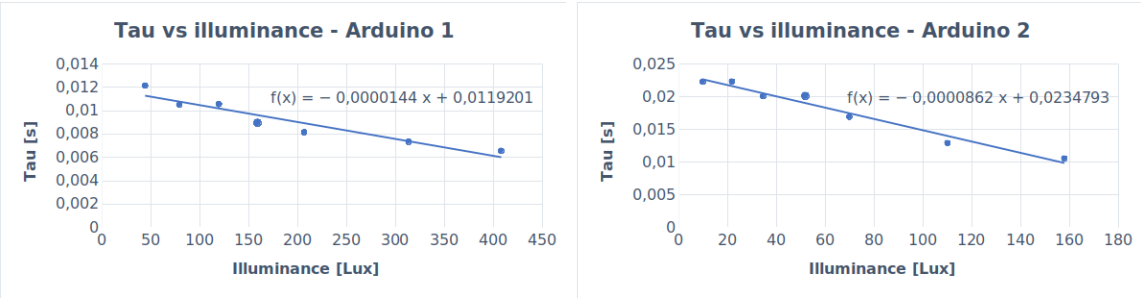
The time constant at desk i, $\tau_i(x)$, can then be computed by:

$$\tau_i(x) = m_{\tau_i}x + b_{\tau_i} \tag{4}$$



(a) Desk 1 (b) Desk 2

Figure 5: G_0 vs illuminance



(a) Desk 1 (b) Desk 2

Figure 6: τ vs illuminance

3.3 Local Illumination Control

The local luminaire controller was implemented by coupling a feedforward controller with a feedback PI controller. This approach combines the zero lag characteristics of the feedforward control with the disturbance rejection of feedback control. Figure 7 shows the block diagram of the controlled system.

3.3.1 Feedforward controller

Using the system's model $G(s)$ characterized in Section 3.2.2, it is possible to implement a feedforward controller. Knowing the desired illuminance level r, and the system static gain at the desired illuminance level $G_0(r)$ (from Equation 3), the feedforward term, $u_{FF}(r)$, is simply computed by:

$$u_{FF}(r) = \frac{r}{G_0(r)}. \tag{5}$$

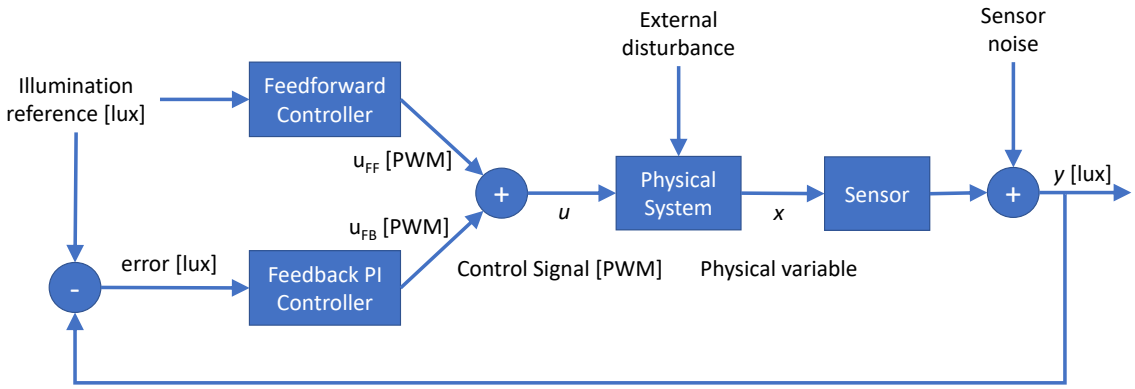


Figure 7: Block diagram of the controlled system.

3.3.2 Feedback Controller

In order to improve the performance of our controller, the feedforward term is coupled with a feedback PI controller. At each iteration k, the feedback control signal $u_{FB}(k)$ is given by:

$$u_{FB}(k) = p(k) + i(k) \tag{6}$$

The proportional term $p(k)$ is used to compensate the tracking error and the integral term $i(k)$ is used to reduce the static error. The $p(k)$ term is computed by

$$p(k) = K_p e(k), \tag{7}$$

and $i(k)$ term is given by

$$i(k) = i(k-1) + K_2(e(k) - e(k-1)), \tag{8}$$

where $e(k)$ is the tracking error in time instant k,

$$e(k) = r(k) - y(k), \tag{9}$$

and

$$K_2 = K_p K_i \frac{T}{2}. \tag{10}$$

The PI controller gains were adjusted to $K_p = 3$ and $K_i = 12$. These values were fined-tuned to minimize the transition time, reduce flicker and have a satisfactory static error.

3.3.3 Improvements to the basic controller

Anti-windup. By using the previously described PI controller the system would face the problem of integrator windup. There are 2 cases where integrator windup happens: one is when the external illuminance is too high and the desired illuminance is lower (e.g. the box lid is open); the other when the desired illuminance is too high for the LED to achieve such illuminance. In both these cases, the feedback controller simply could not compensate the error and the integrator would infinitely accumulate this error. When the box lid was closed again it would take a long time for the LED to turn back on.

To solve this problem, a anti-windup solution was adopted by limiting the values the integrator can take (saturation). The integrator limits were tuned to the values of -10 and 10.

System output predictor. Another problem faced by this controller is reacting to a step input. When the illuminance reference changes as a step signal, the system has some transition delay. As shown in Figure 7, the error is computed using the current measurement and the current reference. There is no accounting for the system delay. The solution to this problem was implementing a block that predicts the expected output given the previous inputs. The current error would be computed by

$$e(k) = y_{\text{expected}}(k) - y(k), \tag{11}$$

where y_{expected} is the result of the predictor.

We tried to implement this solution, but were unsuccessful in doing it. However, the response obtained without the predictor was already quite satisfactory, and so we decided to not use this solution.

3.3.4 Timing analysis

Sampling rate. The controller is programmed to sample the illuminance in the system with a 5 ms sampling period using Arduino timer interrupts. After implementing the local controllers, the sampling period of each unit was tested and is illustrated in Figure 8.

With the gathered distributions we can compute the sampling period and the jitter, given by the standard deviation of the distribution. For both luminaires, the average sampling period is 5 ms. For luminaire 1, the jitter is 21 µs and for luminaire 2, the jitter is 6 µs. The two arduino boards are from two different vendors and thus it is expected for them to have different characteristics. From this analysis we can conclude that the controller computations should take at most 4.9 ms.

Computation time. To analyze the the fitness of the sampling rate, we measured the time for the computations for the controller of each luminaire, shown in Table 1.

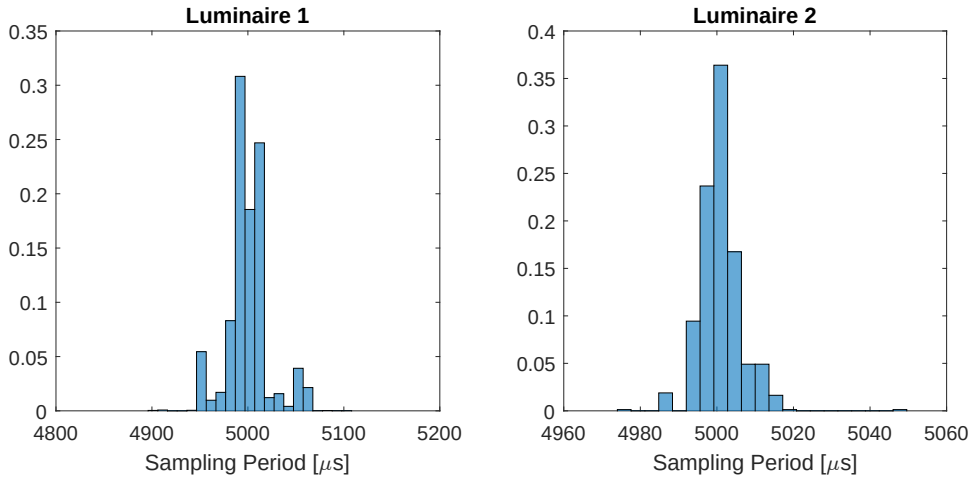


Figure 8: Sampling period for the 2 luminaire systems.

From these results we can conclude that the computation of the controller takes about 1.5 ms from the sampling period, leaving 3.4 ms available for other computations.

Table 1: Analysis of the controller computation times for the two luminaires

	Avg. computation [ms]	Std. deviation [μs]	Serial Comms. [μs/char]
Luminaire 1	1.444	19.4	70
Luminaire 2	1.474	7.5	70

3.4 Global Controller

The described Local Controller achieves the goal of following a local illuminance reference. However, this implementation is not enough to solve the problem of minimizing energy consumption when costs are different for each desk. The global controller can be obtained by using a distributed and cooperative optimization algorithm.

In this section we describe the formulation of the optimization problem, the distributed consensus algorithm and detail the implementation of the distributed controller.

3.4.1 Problem Formulation

The goal function to minimize is the total energy consumption of the system, by choosing the dimming levels that minimize energy consumption, while maintaining

the illuminance above the reference levels. We denote $\mathbf{d}$ as the vector of the dimming of all N luminaires and $\mathbf{c}$ the vector of energy costs. The coupling matrix $\mathbf{K}$ is a $N \times N$ matrix that measures the influence of the dimming of each luminaire on the other luminaires' illuminance. The vectors $\mathbf{o}$ and $\mathbf{L}$ denote the background illuminance and illuminance lower bound at each desk. Vector $\mathbf{o}$ and matrix $\mathbf{K}$ are both measured during the calibration routine. The cost function is given by:

$$f(\mathbf{d}) = \mathbf{c}^T\mathbf{d}. \tag{12}$$

This problem is constrained in 2 ways:

- The dimming of each luminaire i is bounded by $d_i \in [0, 5]$.
- The illuminance at each desk must be above the illuminance lower bound, $\mathbf{k}_i d_i + o_i \geq L_i$.

The problem can be formulated by the following optimization problem:

$$\begin{aligned} \underset{\mathbf{d}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{d} \\ \text{subject to} \quad & \mathbf{K}\mathbf{d} + \mathbf{o} \geq \mathbf{d} \\ & 0 \leq \mathbf{d} \leq 5. \end{aligned} \tag{13}$$

3.4.2 Consensus algorithm

The consensus algorithm works by distributing the computation of the minimum by all the nodes. Each node i holds a copy of the global variable $\mathbf{d}_i$ and must know the local illuminance lower bound L_i, the local background illuminance, o_i, the local energy cost c_i, the coupling gains from the other luminaires to itself, $\mathbf{k}_i$ and the global optimization parameter $\rho = 0.07$.

The consensus algorithm uses an augmented Lagrangian method, described in more detail in [6] and [7]. Until convergence is reached, at each iteration the node:

1. Recomputes the optimal solution for the current state, inside the feasible set (Primal Iterates):

$$\mathbf{d}_i(t+1) = \arg\min_{\mathbf{d}_i}\{\mathbf{c}_i^T\mathbf{d}_i + \mathbf{y}_i^T(t)(\mathbf{d}_i - \bar{\mathbf{d}}_i(t)) + \frac{\rho}{2}||\mathbf{d}_i - \bar{\mathbf{d}}_i(t)||_2^2\}. \tag{14}$$

2. Receives the local solutions from other nodes and computes the average solution:

$$\bar{\mathbf{d}}_i(t+1) = \sum_{j=1}^{N} \mathbf{d}_j(t+1), \tag{15}$$

3. Updates local lagrangians:

$$\mathbf{y}_i(t+1) = \mathbf{y}_i(t) + \rho(\mathbf{d}_i(t+1) - \bar{\mathbf{d}}_i(t+1)). \tag{16}$$

3.4.3 Implementation of the distributed controller

The first part in the development of the distributed controller was defining the communication protocol between luminaires, detailed in Section 3.5.

The communication between luminaires allowed for the development of a finite state machine for negotiation, illustrated in Figure 9. When the system enters

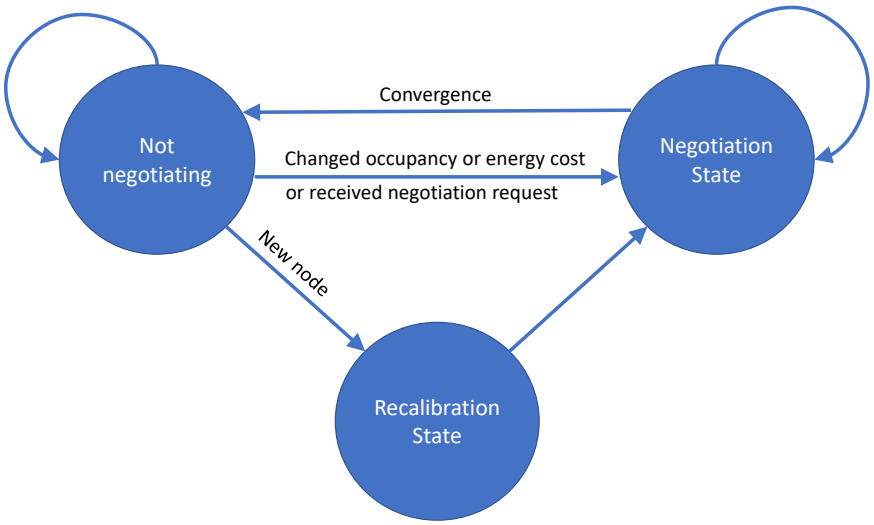


Figure 9: State machine for the negotiation between nodes.

the negotiation state, it executes the computations described in Section 3.4.2, until convergence.

Our implementation uses a synchronous version of the consensus, i.e., a node computes the local solution, then waits for the local solution from other nodes and, after receiving each reply, recomputes the average and the lagrangians. Convergence is obtained when the solution computed by the node is approximately equal to the average, $\mathbf{d}_i(t) \approx \bar{\mathbf{d}}(t)$. When this happens, the node announces to other nodes that it reached a solution and all other nodes exit the negotiation state. The other breaking conditions on the negotiation state is when the number of iterations goes over the 50 iteration limit (no convergence) or when, for some reason, other nodes stop replying and a timeout is triggered. The flowchart of the algorithm is illustrated in Appendix A.

3.5 I2C Communications Protocol

For the nodes to communicate with each other the I2C communication protocol was used. It was used the MT/SR mode (Master Transmitter/Slave Receiver), where the master sends data to the slave, and the slave only receives - in this mode the master releases the bus as soon as it finishes writing. On the MT/SR mode,

a multi-master approach was implemented, where all nodes are master and slaves simultaneously.

Each arduino has an address associated with that is used as identifier when sending a message - the addresses start at 1 and then increase from there (with only two luminaires the addresses will be 1 and 2). Each node's address was set through EEPROM registers, which brings the advantage of being able to use the same code for all nodes. For the purpose of collecting data, each node broadcasts its state permanently to address 0, i.e., the current master sends data to all slaves periodically (at the sampling period of 5ms).

In order to the different nodes communicate with each other, a communication protocol had to be established. In this protocol were defined 7 different messages, being two of them send periodically (a illuminance and a duty cycle), and the others when an event happens. We decided to divide the necessary communications in various different messages, so the length of each message would be the shortest possible.

In Appendix B is described the I2C communication protocol implemented. Each message has 3 or 4 bytes (except for message 2). In all messages the first byte indicates the address of the node that is sending the message, the second byte is a identifier of the type of message, and the remaining bytes are the content of the message.

3.6 Data Server

Components To allow remote communication between our controller and a remote client a data server had to be implemented. This data server is, as described in figure 2, a Raspberry Pi connected to the Arduinos through the I2C bus where it only reads data sent to the channel by the Arduinos, it doesn't send any data to the I2C bus itself.

Communication The server is set up as an asynchronous TCP server in order to avoid the need for multiple communication threads for multiple clients. This way the server only needs a single running process to be able to give response to multiple client request at the same time. All communication variables and methods used for communicating between the server and the clients are based on the Boost asio library. In order for the data server to be able to communicate to incoming client requests it needs to store meaningful data received from the arduinos. In order for this to happen, several data structures were created in order to store in incoming data form the controllers.

Classes Two classes were defined: the database class and the I2C communications class. The database class contains all the data structures where the received information from the arduinos is stored as well as all of the necessary getter and setter methods. The I2C communication class contains all the methods necessary for communication with the arduinos. It also contains a method for continuously

reading information from the I2C channel and store all the received information in the object of the database class previously described.

Storage To store the latest received values of illuminance and duty cycle a 2D vector of floats (vector of vector of floats), from the std library, was implemented for each one of these two types of data. The number of rows correspond to the number of connected arduinos while the columns correspond to the received values ordered by time received, being the latest received value in the beginning of this column/sub-vector, each column can have, at most, 1200 values, which corresponds to all the received values for the mast minute for the frequency of 200Hz. Each time a new value is received it is inserted in the beginning of the column/sub-vector and all the other values are moved to the next index. When inserting a new value, if the column/sub-vector has all its 1200 elements allocated it will delete the oldest received value in order to make space for the new one.

Processing The server needs to be able to continuously perform the following task: read data from the I2C bus and store it in the database; be on the listen for client requests and respond to them. To perform these 2 tasks we use 2 processes running in parallel. A thread is created for reading data from the I2C bus and storing it in database and on the main process of the server we receive and respond to incoming client request.

Sockets A single socket is used in and its address corresponds to the localhost address (127.0.0.1) and it is running on port 123. No other sockets are necessary since the server is asynchronous.

Concurrency When it comes to concurrency we didn't implement any mutexes or other tools to deal with multiple memory accesses due to the simplicity of the project. On projects of larger scale we could implement mutexes that would temporarily block access to the data structures in the database whilst they are being written on.

4 Experiments and Results

NOTE: The final implementation still has a bug that could not be solved in time. This problem occurs when the system is running and the luminaires are sending messages to the I2C bus. After a few seconds, the two senders collide and one of them gets stuck while sending a message. This results in the stuck agent holding the SCL line and the other nodes are stuck reading the I2C bus. This is a bug in the Arduino I2C library. One way increase the execution time without getting stuck is to reduce the sampling frequency, but the communications still got stuck after a minute. The final adopted mitigation was to change the code in the library to allow the execution to not get stuck. What happens in this case is, when the error occurs, I2C communications are disabled and the system keeps running using only the local controllers.

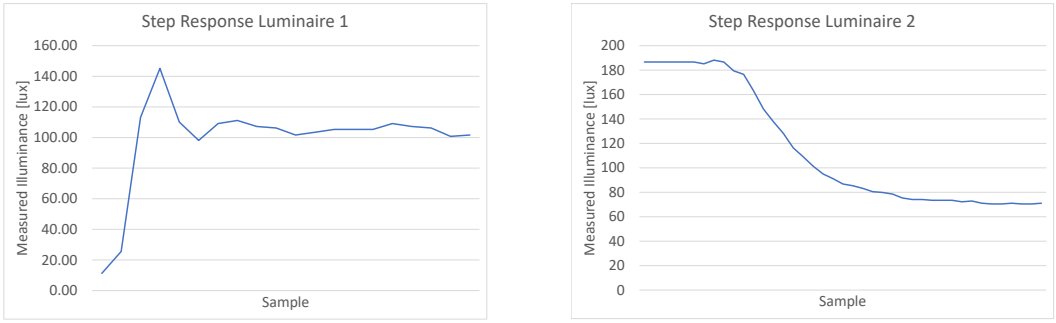
A video was produced to show the correct functioning of the different elements of the project. In this video we can see:

- the user interface with the Serial Monitor;
- the calibration routine;
- the consensus algorithm;
- the local controllers reducing the dimming in the presence of external light;
- the client interaction with the data collection server;
- the I2C communications being lost (erro 9).

Video link: ttps://youtu.be/dU1YCfsOGf4

4.1 Dynamic characteristics of the local controllers

To validate the performance of the designed controllers, the step response of each luminaire was tested and is illustrated in Figure 10.



(a) Luminaire 1 (b) Luminaire 2

Figure 10: Step response of the luminaires.

From this figure we can see that the controllers have different behaviors in each system. The first one reacts and stabilizes faster, but it has a some overshoot. The controller for luminaire 2 is slower. In both cases, the systems stabilize over about 20 iterations which is about 100 ms, a very acceptable value. As can be seen in the video, with the naked eye, there is very few flickering and the system's response is quick for an office usage.

4.2 Advantage of Combined Feedback/Feedforward control

A deeper analysis could be made on the advantages of the use of the combined feedback/feedforward control. This was not accomplished due to lack of time. However, the video is enough evidence to prove that using a feedback controller can help reduce energy costs and guarantee the tracking of a reference illuminance level. When the box lid is opened, the LEDs react to the disturbance and reduce their power consumption, maintaining the illuminance at the desired level.

4.3 Advantage of Global vs Local Control

Due to the I2C communications error, it was not possible to read the metrics with the 2 luminaries and thus we could not obtain any experimental results on this comparison. We can, however, motivate for the expected comparative results of these two approaches by comparing the expected energy costs of both approaches.

The experiments shown in Table 2 used the system's static gain G_0 from Equation 2, and the values $o_1 = 0.03, o_2 = 0.01, k_{12} = 3.5$ and $k_{21} = 3.3$.

Table 2: Comparison between energy costs using Feedforward and Consensus

Costs		References		Feedforward sol.			Consensus sol.			Energy savings (%)
c_1	c_2	L_1	L_2	d_1	d_2	cost	d_1	d_2	cost	
20	1	100	100	1.82	3.907	40.28	1.5	5	35	13.115
10	1	100	100	1.82	3.907	22.09	1.582	3.702	19.5	11.617
5	1	100	100	1.82	3.907	13	1.582	3.702	11.6	10.657
1	1	100	100	1.82	3.907	5.725	1.582	3.702	5.28	7.7013
1	2	100	100	1.82	3.907	9.632	1.582	3.702	8.99	6.6969
1	5	100	100	1.82	3.907	21.35	1.582	3.702	20.1	5.8931
1	10	100	100	1.82	3.907	40.89	5	3.262	37.6	7.995
1	10	20	100	0.43	3.907	39.49	5	3.262	37.6	4.7501
1	1	20	100	0.43	3.907	4.332	0.135	3.889	4.02	7.1101
1	1	20	20	0.43	1.267	1.693	0.336	1.197	1.53	9.4842
1	1	100	20	1.82	1.267	3.086	1.761	0.899	2.66	13.823

We can conclude that the use of a distributed solution can consistently provide energy savings, both for symmetric and asymmetric costs. The energy savings are substantial, in this example they can range from 5-13%.

5 Conclusions

In this projected we managed to create a functioning distributed control system for controlling illuminance in simulated office environment. This system also includes a data collection server and a user interface.

We can conclude that systems like this one can provide a good energy saving solution for modern offices. Our experiments show that the use of a distributed controller can reduce energy costs by at least 5-13%. Furthermore, with the use of the adaptive brightness feature of our system (feedback control), the luminaires reduce their power consumption when there is an external source of light.

This project can be a viable product to market. In order to do so, a few improvements must be done. We suggest the following improvements:

- create a user-friendly interface for interacting with luminaires and data collection server;
- improve local controllers: fine-tune parameters and use a system output predictor for faster stabilization;
- improvements on communications: use another I2C library and reduce load on the I2C bus.
- improvements on data server: enable concurrent client requests.

References

[1] Alexandre Bernardino and João Pedro Gomes. Distributed Real-Time Control Systems project handout, IST, 2018.

[2] GL5528 LDR Datasheet, `https://fenix.tecnico.ulisboa.pt/downloadFile/845043405474933/GL5528.pdf`.

[3] ATmega328 Datasheet, `https://www.microchip.com/wwwproducts/en/ATmega328`.

[4] Arduino, `https://www.arduino.cc/`

[5] Raspberry Pi, `https://www.raspberrypi.org/`

[6] Alexandre Bernardino. *Solution of Distributed Optimization Problems: The consensus algorithm.* Real-Time Distributed Control Systems, IST, 2018.

[7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato and Jonathan Eckstein. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Foundations and Trends in Machine Learning, Vol. 3, No. 1 (2010) 1122.*

A

Distributed control algorithm flowchart

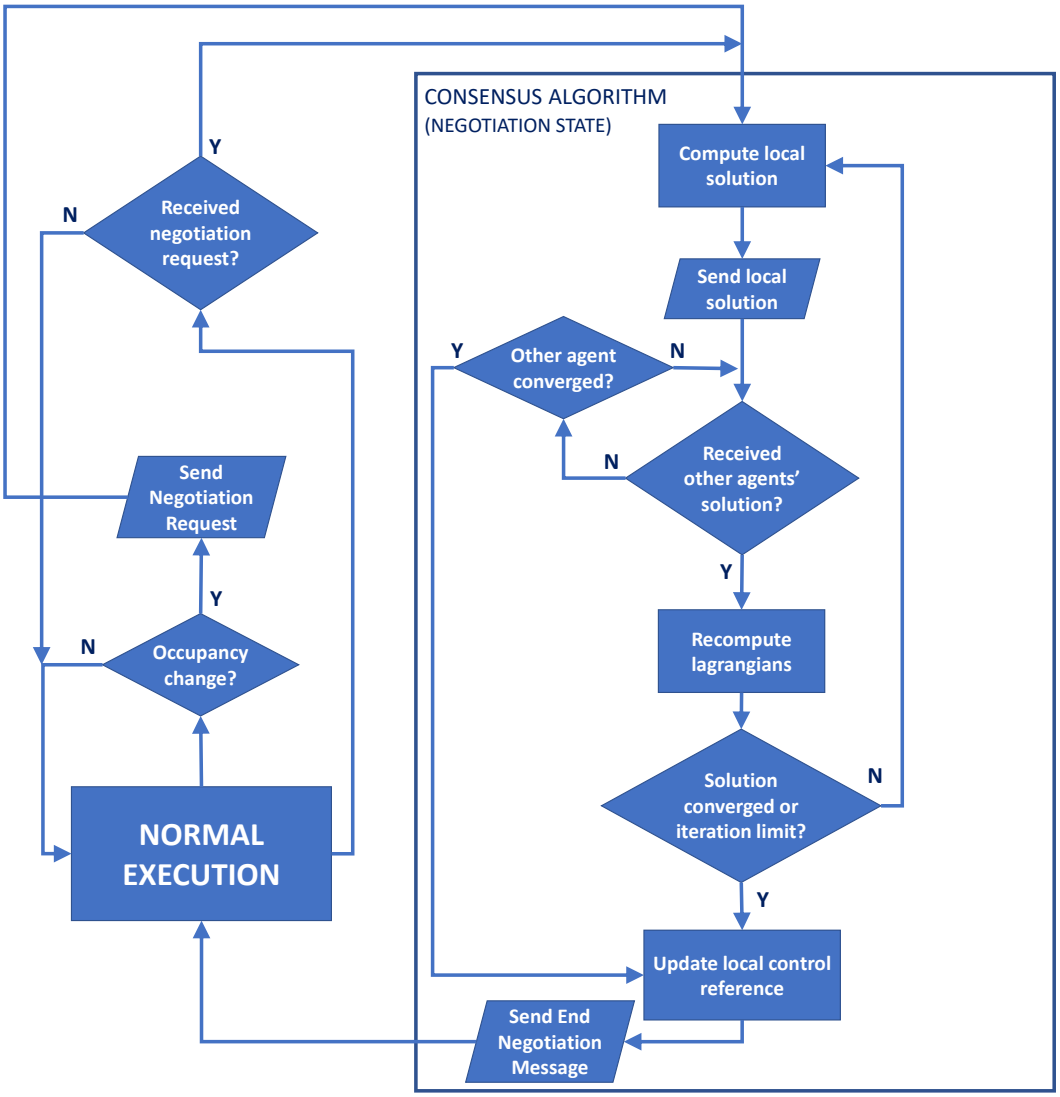


Figure 11: Flowchart of the implementation of the consensus algorithm.

B

I2C communications protocol

Command	Client Request	Server Response	I2C message	Amount of bytes	Observation	Arduino Function
Get current measured illuminance at desk <i>	g l <i>	l <i> <val>	<address>1<val>	4	<address> is the address of the sending node <val> is floating point number expressing measured illuminance in lux between 0 and 500	sendLuxReading(<val>)
Get current duty cycle at luminaire i	g d <i>	d <i> <val>	<address>0<val>	3	<val> is 1B representing a floating point number expressing duty cycle between 0 and 5	sendPwm(<val>)
Get current occupancy state at desk <i>	g s <i>	s <i> <val>	<address>3<val>	3	<val> is a Boolean flag: 0 – non-occupied, 1 – occupied	sendOccupancy(<val>)
Get current illuminance lower bound at desk <i>	g L <i>	L <i> <val>	<address>4<val>	4	<val> is floating point number expressing illuminance lower bound in lux between 0 and 500	sendLuxLowerBound(<v>)
Get current external illuminance at desk <i>	g o <i>	o <i> <val>	<address>5<val>	4	<val> is floating point number expressing background illuminance in lux between 0 and 500	sendLuxBackground(<v>)
Get current illuminance control reference at desk <i>	g r <i>	<i> <val>	<address>6<val>	4	<val> is floating point number expressing the illuminance PWM control reference (obtained by consensus) between 0 and 5	sendPWMRef(<v>)
Send negotiation message (local solution)			<add>2<N><val 1>...<val N>	>5	<N> is the number of nodes to be sent <val i> is a proposed PWM value for node i	sendNegotiation(<v>,N)
Send negotiation state			<address>7<val>	3	<val> is a flag: '1' - ready to begin negotiation '0' - not negotiating '2' - reset system setup	sendNegotiationState(<v>)

Figure 12: Communication protocol