TÉCNICO LISBOA

Distributed Real-Time Control Systems

Professor Alexandre Bernardino

Professor João Pedro Gomes

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

Aerospace Engineering

12 of January 2019

João Santos, **81510**

Miguel Luis, **81632**

Pedro Trindade, **81612**

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

João Santos, Miguel Luis, Pedro Trindade

Abstract

This report presents the technical work developed for the curricular unit of Distributed Real-Time Control Systems, in Instituto Superior Técnico. The project consist of a distributed control system for a smart office illuminance system, which ensures minimum illuminance requisites, dependent on the desk's occupancy, while optimizing energy consumption. A prototype of the system was built, using simple electronic components, such as LEDs, LDRs, resistors and capacitors, and development platforms, such as Arduino Uno and RaspBerry Pi. Each node, representing a desk, runs a local controller, tasked with the objective of tracking a reference illuminance value. Communication between the nodes, using I^2C protocol, was used in order to implement global optimization, using a distributed computation approach, by means of the ADMM algorithm, solving a consensus problem. Also, a TCP server was developed, to provide an interface to a client application that allows to supervise the system, and compute appropriate evaluation metrics.

Keywords— Real-Time control, Distributed Systems, Consensus, Raspberry Pi 3B, Arduino Uno, Optimization, C++ TCP Server, Threads, ADMM

Contents

1 Introduction

In the century's early years, illumination accounted for a large portion of energy consumption in office spaces. In the U.S., illumination accounted for 38% of energy consumption in office spaces by 2003. This registered a considerable decrease, and by 2012, statistics indicate this value decreased to about 17% [1]. However, this is still a considerable percentage. With the increase in energy cost, and the ever increasing environmental concerns, it is important to increase lighting efficiency in office buildings, in order to achieve greener solutions. This can be achieved in a variety of ways, such as, occupancy sensing, daylight harvesting, scheduling, or simply by the use of more efficient lamps [2].

There has been registered an increase in the percentage of compact fluorescent lamps (CFL) in office spaces, which provide a considerable increase in efficiency over the previously used incandescent light bulbs. Light Emitting Diodes (LEDs), available nowadays are more efficient than CFLs, however they are not yet widely used for this purposes, due to higher price. However, as this technology matures, a decrease in price is expected, and soon enough this might become the standard for office illumination systems. LEDs provide some advantages over CFLs. They provide an expected service life about 2 times longer, and have little to none warm-up time. Also, they can easily be integrated in a digital control system, which provides increased flexibility and allows to achieve a better solution.

Automation of LED light control systems has been a target of recent research [3][4][5]. This research bears some similarities with the problem at hand. The autors use distributed control in order to optimize illumination in an indoor space.

Our objective is to simulate an office space using simple hardware, in order to implement and test optimization and control algorithms for the illumination in this space. The office space is composed of several desks, with luminaires, on which we wish to control the luminaire's lighting, depending on the desk occupancy level. Due to the coupling effect of the several luminaires in each other, we wish to optimize the necessary power in each of them, in order to decrease global energy consumption, while guaranteeing adequate illuminance levels at each desk. It should also be able to adjust the LED dimming level, in order to adapt to external lightning. In order to achieve this goal, a prototype was built using a simple shoe box, coated with white paper to increase reflectivity, and simulated the nodes using Arduino boards with adequate circuits for driving led illuminance and reading illuminance values. Further, a TCP server was developed in a RaspBerry Pi, in order to enrich the system, thus allowing for a user to monitor it and extract information, using an application.

2 Concepts

In order to develop the system, it was necessary to produce a system prototype, i.e., small scale luminaries, to study and evaluate the system properties, and implement the developed algorithms. The developed system can be decomposed in two main components, these being

- the individual luminaires, or nodes;
- the data server, to monitor the system, collect data and provide an interface with the user.

2.1 Luminaires

Each luminaire consist of:

- a computational and communication system - for this work, this is comprised of an Arduino Uno,
- a light source - a Light Emitting Diode (LED) was used,
- an illuminance sensor - a circuit using a Light Dependent Resistor (LDR) was used,
- an occupancy sensor - This was not considered in the development of the project, and a switch was used to simulate the sensor.

For each luminaire i, values will be defined for minimum desired illuminance, i.e., an illuminance lower bound, L_i, which is defined according to its occupancy state, s_i. The goal for the system is to minimize global energy consumption. This is done using inter-node communication, via I^2C protocol [7], for cooperatively solving this optimization problem.

2.1.1 Light Sensor

In order to implement the light sensor, a LDR was used, as previously described. Within the illuminance region of interest, we can write

$$\log_{10}(R_L) = m \log_{10}(l) + b, \tag{2.1}$$

to relate the LDR's resistance value with the incidence illuminance, where R_L is the LDR resistance and l is the received illuminance, and m and b represent characteristics of the LDR. This way, after identifying the LDR characteristic, a light sensor can be implemented by measuring its resistance.

2.1.2 Light Source

For the used light source, a white LED, it can be said that $l_{led} \propto d$, where $d \in [0, 100]\,\%$ represents the LED dimming value, and l_{led} represents the produced illuminance. The LED dimming will be adjusted in the Arduino micro-controller, using an analog write port, which uses PWM modulation [1], thus, we are able to adjust the duty cycle of the rectangular wave, using Arduino's input variable $u \in [0, 255]$, which can be related to the dimming by $d = \frac{100}{255}u$, thus allowing us to control a luminaire illuminance value.

2.2 Data Server

The data server aims to monitor the system, "listening" to communications in the I^{2}C bus, and storing system data, as well as computing metrics to evaluate the system behaviour. It also aims to provide a user interface to the system, such that the user can access the computed metrics and current relevant values.

The data server was implemented in a Raspberry Pi 3B, in C++ [8], more concretely, C++11, which provides useful features not available in previous versions [9], such as lambdas, unordered containers and initializer lists, which where very useful in the development the data server. Also, the PIGPIO library[2] was used to simplify the use of the I^{2}C communication protocol, using Raspberry's GPIO pins. As for the implementation of the server itself, this was performed using TCP sockets, one for each client. In order to simplify this, Boost.Asio library[3] was used to implement an asynchronous server. Also, for ease of use, a static IP address as associated with the server (RaspBerry Pi), for easier access and use by the client.

3 Overall System Architecture

The overall system is represented in the diagram of figure 1, representing the main components, as well as the flux of information between the components.

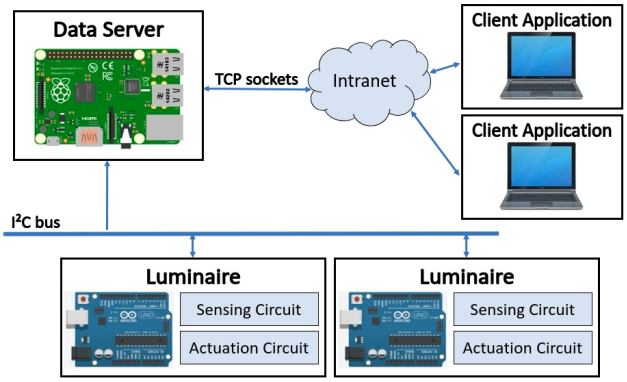


Figure 1: Diagram of the developed system architecture, illustrating the main components present.

This simple scheme allows to understand very simply how the system's components are linked together and how data is transferred among them. This components will be detailed in the remainder of the report, as well as communications among them.

In the next section of the report, we will begin by describing how the actuation and sensing circuits where implemented and identified, since this will be fundamental in the implementation of the control algorithm.

[1] described in `https://learn.sparkfun.com/tutorials/pulse-width-modulation/all`
[2] described in "The pigpio library", `http://abyz.me.uk/rpi/pigpio/`
[3] described in `https://theboostcpplibraries.com/boost.asio`

4 System Identification

In order to characterize our system, and develop the controllers to be used, it is necessary to study the system and identify the involved dynamics. In this section we present the methods used to perform the identification of the characteristics of the system.

4.1 Circuit Description

The circuitry used in combination with the Arduino micro-controller, in order to implement the small scale luminaires is illustrated in figure 14, in the appendix.

The illuminance sensor circuit is represented in sub-figure 14a and is composed of a LDR, R_L, a known resistor R_1 and a capacitor, C. The value of R_L can be obtained by measuring the voltage v at represented analog read pin, A_0, from Arduino, in stationary regime, v_s,

$$v_s = \frac{R_1 R_L}{R_1 + R_L} V_{CC} \Leftrightarrow R_L = R_1 \left(\frac{V_{CC}}{v_s} - 1 \right), \tag{4.1}$$

and therefore determine the incidence illuminance in SI units, lux, using equation 2.1.

The represented capacitor, C, is used to form a low-pass filter, in order to remove noise from the measurement but mostly to obtain the mean component of the PWM pulses. To do so, the frequency of the PWM pulses needs to be at least ten times greater than the filter cutoff frequency, so that its influence can be neglected.

Analyzing the circuit, we can conclude that the time constant, associated with the filter, changes with the incidence illuminance,

$$\tau(l) = \frac{R_1 R_L(l)}{R_1 + R_L(l)} C. \tag{4.2}$$

From equation 4.2, we can conclude about the magnitude of the time constant. The highest possible value for the time constant, happens when $R_L \to \infty$, i.e. when the incidence illuminance tends to zero, and is $\tau_{max} = R_1 C =$ 10ms, which corresponds to a cutoff frequency of 100Hz.

Note that the time constant decreases as R_L becomes smaller, i.e., as illuminance increases, and so, for higher illuminance, the measurement will be noisier.

The LED driving circuit, represented in sub-figure 14b is composed by the LED, D_{LED}, in series with a resistor, R_3, and is connected to the PWM capable pin 3 from Arduino. Due to the low-pass characteristic of the measuring circuit, we can conclude that increasing the frequency of the PWM wave, will yield better measurements. This way, the PWM frequency was set to the maximum frequency available to this port, 32150Hz.

4.2 LDR characteristic identification

In order to characterize the system, it is necessary to calibrate the sensor, i.e., determining the LDR characteristic. As described in subsection 2.1.2, the LED produced illuminance should be proportional to its dimming value. This way, incidence illuminance in the LDR, should be

$$l = l_{led} + o(t) = G_0 u + o(t), \tag{4.3}$$

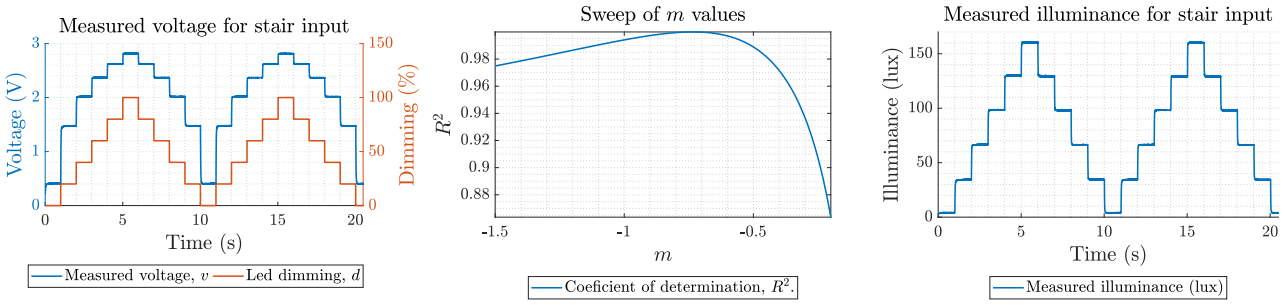
where $o(t)$ represents an external disturbance. It is assumed that this disturbance is constant, i.e., $o(t) = o$.

In order to determine the LDR characteristic, we began by setting values for u, and measuring the resulting illuminance, using a luxemeter. However, this measurements were not accurate, and results were far from linear. In order to overcome this problem, we decided to set the parameters such that this evolution was linear. Rewriting equation 2.1, we get

$$l = 10^{\frac{\log_{10}(R_L) - b}{m}} = \left(10^{-b} R_L\right)^{1/m} = A R_L^{1/m}, \tag{4.4}$$

with $A = 10^{-b/m}$. As we can see, b only afects the value of A, and so, it acts as a gain. In order to achieve linearity, we need to determine m. This was performed by changing the value of u, in a stair pattern, and measuring the voltage v, in stationary regime. From this voltage, the LDR's resistance, R_L, was computed. Then, a sweep over values of m was performed, computing the transformation of equation 4.4, and performing a linear regression, in order to determine the value of m that achieves the best regression. The process is illustrated in figure 2, which has represented in 2a the stair input applied to the LED and the measured voltage, and in 2b, the sweep over m values, which allows us to choose the m that maximizes the coefficient of determination, R^2.

As for the determination of the coefficient b, the luxemeter values where used. For this, the LED dimming was



(a) Representation of the measured voltage and the corresponding input.

(b) Coefficient of determination, R^2, as function of the slope m.

(c) Conversion of measured voltage to illuminance values.

Figure 2: LDR characteristic identification process.

set to the maximum, and the luxemeter used to measure the resulting illuminance level. Doing this, we were able to adjust b so that the maximum achieve illuminance coincides with the measured one. The results after conversion to lux are illustrated in sub-figure 2c.

4.3 Static model identification

Having identified the LDR characteristic, it is now possible to use the measured illuminance values from the stair pattern illustrated in sub-figure 2c, to determine the static gain, G_0, and external disturbance, o, described in equation 4.3. This can be performed by a simple linear regression. This process, however, is conducted in the luminaires, every time the system restarts, this way, accounting for possible changes in the system.

4.4 Dynamic model identification

As described in subsection 4.1, the sensing circuit has a capacitor, thus performing a low-pass filtering of the measurement. As for the actuation circuit, it is assumed to be fast enough that its influence can be neglected (order of nanoseconds). In order to obtain a better performance in the controller, the equivalent model must be identified. By analyzing the sensing circuit, we can derive

$$\tau(l)\dot{v} = -v + \frac{V_{CC}}{R_1 + R_L(l)}, \tag{4.5}$$

where τ represents the equivalent system's time constant, previously described in equation 4.2, and v represents the voltage measured at the analog read pin. This can be considered a linear time varying system. This equation could be converted into lux, which is actually the system output, and the variable we wish to control. However, the system considering the output in lux, would become non-linear. As for the system described in voltage, v, for a fixed illuminance level, l_s, this becomes a first order linear system, with fixed time constant, and its evolution for a step input can be described by

$$v(t) = v_i + (v_s - v_i)\left[1 - \exp\left(-\frac{t - t_i}{\tau(l_s)}\right)\right], \tag{4.6}$$

where v_i represents the initial voltage, at time t_i, when the illuminance was changed, and v_s represents the final voltage, i.e. the stationary value of v. These values should be changed every time the reference illuminance, l_{des}, changes. The final voltage, v_s, can be computed from equation 4.5 by setting $\dot{v} = 0$, and computing $R_L(l_{des})$ from the determined LDR characteristic.

Finally, in order to characterize the system, it is important to determine how the time constant, τ, varies with the illuminance level. From equation 4.2, and equation 2.1, we can conclude that the variation of the time constant with the illuminance is not linear. However, from equations 4.1 and 4.2, we conclude that the evolution of the time constant is linear with the target voltage, i.e., the voltage in stationary regime, v_s

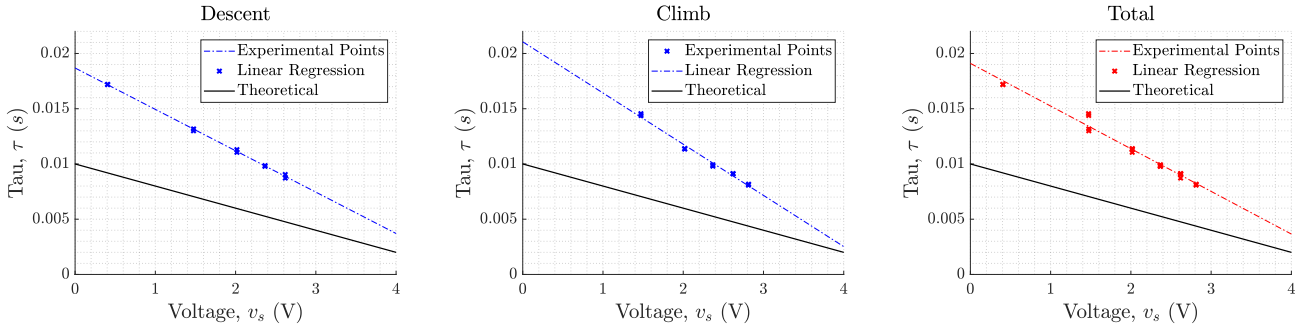
$$\tau(v_s) = \frac{R_1 C}{V_{CC}}(V_{CC} - v_s). \tag{4.7}$$

This way, the time constant will be determined as a function of the target voltage, which can easily be related to the target illuminance, l_{des}. The time constant determination can be conducted using the same stair pattern experiment, which will allow to obtain the time constant for different target voltages. The time constant can be

determined, for each stair step, by determining the time passed since the step input until the voltage rises 63.2% of the total rise $(v_s - v_i)$, i.e., by replacing t by $t_i + \tau$ in equation 4.6 we obtain

$$v(t_i + \tau) = v_i - (v_s - v_i)\left[1 - e^{-1}\right] \Longrightarrow \frac{v(t_i + \tau) - v_i}{v_s - v_i} = 1 - e^{-1} = 63.2\%. \tag{4.8}$$

The results of these determination are illustrated in figure 3. In the graphs, there are represented the determined time constants, as well as the linear regression performed with these time constant values, and the theoretical value, described by equation 4.7. It was distinguished between the cases where the target voltage decreases (subfig. 3a) and where it increases (subfig. 3b). A difference is observable between these two cases. The time constant values



(a) Descending steps in voltage. (b) Increasing steps in voltage. (c) All determined points.

Figure 3: Time constant determination as a function of the measured static voltage, v_s.

appear larger in the case where there is an increase in voltage. This implies that the capacitor takes longer to charge than to discharge, which might be caused by the capacity of the Arduino's source pin to provide the current requested by the circuit.

Finally, in order to perform all of these tasks in a succinct manner, an Arduino program and a MatLab script were developed, which run the stair pattern experiment, and process the data.

5 Control

In this section, the control algorithms are presented. These algorithms were used to sustain the real-time cooperative control and were implemented in two stages. In a first approach, a Feedback + Feedforward (FB+FF) architecture was implemented for a single luminaire system. This system was intended to follow any request from the user with a considerable speed and a small error. In the second and final approach, a second luminaire was introduced to the simulated room becoming a two (coupled) luminaires system. Since the two luminaires influence each other, a single (FB+FF) controller implemented in each station may not be efficient and may lead to instability. For this reason, in this stage, a cooperative distributed control system was implemented. The chosen distributive control algorithm is called Consensus and will be presented later in this section.

5.1 Individual Controller

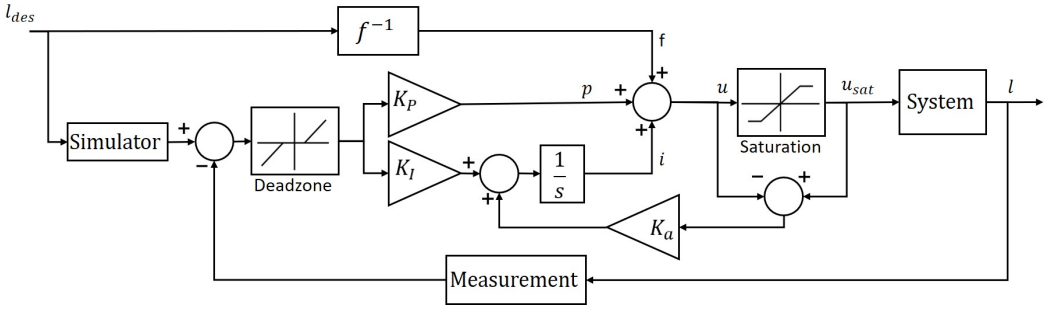


Figure 4: Individual controller block diagram.

5.1.1 Feedforward Control

The feedforward control strategy will consist on computing the system input that will, in stationary regime, take the illuminance value to a desired value. This implies the need for a system static model. Let this be described by $l_s = f(u)$. This way, the feedforward controller consist on setting the input such that

$$u = f^{-1}(l_{des}), \tag{5.1}$$

where l_{des} represents the desired illuminance value, i.e., the control reference. In our case, function $f(u)$ is given by $l_s = f(u) = G_0 u + o$, as previously described in equation 4.3.

This however is not able to assure tracking, nor rejecting disturbances. This way, a feedback controller is used alongside the feedforward controller.

5.1.2 Feedback Control

The digital feedback control consists of a proportional and integral term, given by:

$$p(t) = K_p e(t) \qquad\qquad i(t) = i(t-h) + K_p K_i T \frac{e(t) - e(t-1)}{2} \tag{5.2}$$

with $e(t) = r(t) - y(t)$ the measured error, $r(t)$ the reference signal and $y(t)$ the measured value of the illuminance. It is important to mention that, in this architecture, the reference signal is not l_{des}, but instead the output of a predictor. This predictor (or simulator) is used to predict, at each time sample, the output of the system given an initial and a final state.

5.1.3 Simulator

As it was stated in the previous subsection, the simulator is used to obtain the theoretical output of the system at each time sample. For its implementation, as previously described, equation 4.6 was used, to determine the expected measured voltage at each time sample. As described before, the value of v_s is only updated if there is a change in the reference illuminance (introduced by the user), and therefore values of v_i and t_i must be updated as well.

5.1.4 Controller Features

In order to improve the performance of the closed loop FB+FF system response, a few features were introduced. In this subsection, these features are presented as well as the problems they help mitigate.

Feedfoward + Feedback architecture

The architecture of the controller is itself considered as a feature since it is different from the usual and simpler feedback controller architecture that we are used to. This architecture combines the benefits from each of the controllers, leading to a better performance, when compared to the simpler approach. As it was stated before, a simple feedforward controller does not assure disturbances rejection, and requires an accurate knowledge of the static model. If these do not stand, then the resulting system will not perform well. The PI controller is applied to reduce the error caused by the accuracy of the model and reject the external disturbances.

Anti-windup

The windup is a phenomenon that occurs in a integrative controller when the control signal saturates and an error still exists. In this case, the integral part keeps accumulating error. When a change occurs afterwards, a considerable response delay will result, due to the accumulated error. An example of this occurrence in the context of this project is when the reference is set to a low illuminance level and the box is opened. The LDR will measure a high illuminance value and the LED will turn off completely. In the meantime, the integrator term will accumulate the error over time. If the box is closed once again, the controller will take a substantial amount of time before resuming a normal behaviour.

In order to overcome this, the controller structure was changed by introducing the controller output saturation and adding a feedback loop to the integrator term to incorporate the difference between the unsaturated and saturated input. Changing equation 5.2, this results in:

$$i(t) = i(t-h) + K_p K_i T \frac{e(t) - e(t-h)}{2} + K_a(u_{sat} - u); \tag{5.3}$$

Error Deadzone

Regarding the measurements, it is important to notice that Arduino's ADC (analog to digital converters) only outputs 1024 possible values (0 when 0V are measured and 1023 when 5V). This results in a resolution of $5/1024 = 4.88mV$. Although small, the resolution might lead to quantization errors in the system. Other quantization errors may occur in the system output, since PWM signals only have 8bit resolution. These errors are likely to add up and cause variations in the system input, even for a fixed output. This can result in flickering, as the controller will continually try to compensate for these errors.

A simple way to overcome this problem is by introducing an error deadzone. This way small errors caused by quantization and noise will be ignored and flicker will be mitigated.

5.2 Parameter Tuning

In order to get the required behaviour, the controller gains have to be tuned. According to the presented architecture, the gains that require tuning are K_p, K_i and K_a. In order to do so, the group followed empirical rules that were presented in the classes [10]. The results where then adjusted in order to get the best performance.

5.3 Distributed Controller

As it was mentioned before, when another luminaire is introduced in the room, the performance of the controller is reduced due to coupling effects. In order to solve this problem, each controller must be aware of the impact of each luminaire on one another, and coordinate their actuation, using communication. This is basically the function of a distributed controller architecture. In this section, the algorithm used for coordination is presented, and in addition, the expected results of its implementation on the described system are shown.

5.3.1 Consensus

In order to implement our distributed controller, a Consensus algorithm was used, and coordination is to be achieved using optimization. The goal is to obtain a global optimal solution for the output value of each node, provided that each of them fulfills some restrictions. In addition, it is pretended the result to be produced by distributing computation across all nodes. After reaching a possible solution, each node has to communicate its conclusion to its neighbours that use this information to obtain another possible solution and communicate it. In the end, all nodes reach an agreement.

In order to apply the Consensus algorithm, the optimization problem that refers to the current problem must firstly be defined. For this optimization problem, it was decided that it should minimize energy consumption and maximize lamp longevity. For the first objective, the cost function must include a linear term that penalizes the energy consumption at each node. In order to accomplish the second objective, the controller must prevent lamps going maximum power. For this purpose a quadratic term is introduced. This results in the cost function

$$J(\mathbf{u}) = \mathbf{c}^T\mathbf{u} + \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u}, \tag{5.4}$$

where $\mathbf{c}$ represents a cost vector for the node power, $\mathbf{u}$ represents the input vector and $\mathbf{Q}$ represents the lamp cost. It is important to notice that matrix $\mathbf{Q}$ is diagonal. Regarding the restrictions, the first restriction is applied to guarantee that the coupled illuminance created by the set of all luminaires in station i, $\sum_{j=1}^{N} K_{ij}u_j$, plus external disturbances o_i is larger than the lower bound L_i. Writing this relation in vector form results in

$$\mathbf{K}\mathbf{u} \geq \mathbf{L} - \mathbf{o}, \tag{5.5}$$

where square matrix $\mathbf{K}$ represents the coupling gains of and from all luminaires, $\mathbf{L}$ is the vector of illuminance lower bounds and $\mathbf{o}$ is the vector of external disturbances. In addition, actuators saturation must also be included as restrictions. This way, the following relations are included in the set of restrictions

$$u_i \geq 0, \quad u_i \leq 255. \tag{5.6}$$

The optimization problem is then stated as

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{u} + \frac{1}{2}\mathbf{u}^T\mathbf{Q}\mathbf{u} \\ \text{subject to} \quad & \mathbf{K}\mathbf{u} + \mathbf{o} \geq \mathbf{L} \\ & \mathbf{0} \leq \mathbf{u} \leq \mathbf{255}. \end{aligned}$$

This optimization problem is quadratic and of simple solution for a central processor with information about all

nodes. However, in this system there is no central unit capable of performing optimization. Instead, it is required that each node perform the cost function optimization with no information regarding the other nodes. By combining each result and iterating, it is expected that the optimal solution for the problem is reached.

In order to convert the global optimization problem into separated problems, the function and the restriction are separated and the problem is rewritten as:

$$\underset{u\in\Omega}{\operatorname{argmin}} \quad \sum_{i}^{N} J(\mathbf{u}_i).$$

$$J(\mathbf{u_i}) = \begin{cases} \mathbf{c}_i^T\mathbf{u}_i + \frac{1}{2}\mathbf{u}_i^T\mathbf{Q}_i\mathbf{u}_i, & \text{if } \left\{0 \le u_{ii} \le 255 \ \wedge \ \sum_{j=1}^{N} k_{ij}u_j \ge L_i - o_i\right\}, \\ +\infty, & \text{otherwise.} \end{cases}$$

$$\mathbf{u}_i = \begin{bmatrix} u_{i1} & \dots & u_{ii} & \dots & u_{iN} \end{bmatrix}^T, \qquad \mathbf{c}_i = \begin{bmatrix} 0 & \dots & c_i & \dots & 0 \end{bmatrix}^T \qquad \mathbf{Q}_i = \operatorname{diag}(0,\dots,Q_i,\dots,0)$$

Notice that $\mathbf{u}_i$ corresponds to the input values of all nodes computed from node i.

Several methods can be applied to solve this distributed optimization problem. In this project the alternated direction method of multipliers, ADMM [6] was used. ADMM is an algorithm that is intended to gather the decomposability of dual ascend and the superior convergence properties of the method of multipliers in order to get a get a solution for the decomposed optimization problem. The ADMM requires that all local copies $\mathbf{u}_i$ in each node should be identical. Beyond its local copy of the solution, each node i maintains at each time a local copy with the average solution of all nodes $\bar{\mathbf{u}}_i$ and a local variable of Lagrange multipliers. Then it performs the following iteration until convergence is reached:

1. $\mathbf{u}_i(k+1) = \underset{\mathbf{u}_i\in\Omega_i}{\operatorname{argmin}} \left\{ \mathbf{c}_i^T\mathbf{u}_i + \frac{1}{2}\mathbf{u}_i^T\mathbf{Q}_i\mathbf{u}_i + \mathbf{y}_i^T(k)(\mathbf{u}_i - \bar{\mathbf{u}}_i(k)) + \frac{\rho}{2}\left\|\mathbf{u}_i - \bar{\mathbf{u}}_i(t)\right\|_2^2 \right\}$
2. $\bar{\mathbf{u}}_i(k+1) = \frac{1}{N}\sum_{j=1}^{N}\mathbf{u}_j(k+1)$
3. $\mathbf{y}_i(k+1) = \mathbf{y}_i(t) + \rho(\mathbf{u}_i(k+1) - \bar{\mathbf{u}}_i(k+1))$

where k denotes the iteration index, ρ is the augmented Lagrangian method penalty parameter (ADMM uses an augmented Lagrangian whose role is to augment the Lagrangian with a quadratic term to promote the convergence of the method) and Ω_i is the local constraint.

$$\Omega_i : \left\{0 \le u_{ii} \le 255 \ \wedge \ \mathbf{k}_i^T\mathbf{u}_i \ge L_i - o_i\right\}, \qquad \mathbf{k}_i = \begin{bmatrix} k_{i1} & \dots & k_{ii} & \dots & k_{iN} \end{bmatrix}^T$$

As a result, each node i needs to know l_i, o_i, c_i, k_i, ρ and the actuator bounds at each time sample, in order to compute iteration k.

5.3.2 Primal Iteration

So far only the iteration procedure was stated. However, in order to obtain the optimal value, the first part of the iteration must be solved. To do so, it is important to notice that the local optimization problem can be rewritten as

$$\mathbf{u}_i(k+1) = \underset{\mathbf{u}_i\in\Omega_i}{\operatorname{argmin}} \left\{ \frac{1}{2}\mathbf{u}_i^T\mathbf{B}_i\mathbf{u}_i - \mathbf{u}_i^T\mathbf{z}_i \right\}$$

Where $\mathbf{B}_i = \mathbf{Q}_i + \rho\mathbf{I}$ and $\mathbf{z}_i = -\mathbf{c}_i - \mathbf{y}_i + \rho\bar{\mathbf{u}}_i$. This way, the problem is simplified into a convex optimization problem, that can be easily solved. It is important to notice, however that the solution to this problem can be found inside the region allowed by the restrictions or in its boundary. For this reason, the solution is computed in six different domains:

- In the interior of the region
- In the illuminance lower bound domain $\mathbf{k}_i^T\mathbf{u}_i = L_i - o_i$ (ILB)
- In the lower bound on its actuation domain $d_{ii} = 0$ (DLB)
- In the upper bound on its actuation domain $d_{ii} = 255$ (DUB)
- At the intersection of ILB with DLB
- At the intersection of ILB with DUB

The resulting expressions of the solution $\mathbf{u}_i$ in each of these domains is presented in the Appendix section. It is important to notice that in each iteration of the consensus algorithm, all six possible solutions are computed. To

choose the optimal solution among all six, first the feasibility check must be performed. If the solution is feasible (the solution is inside the domain set by the restrictions), then the cost of each is computed. Choosing the lowest cost solution, the local optimal solution is obtained and sent to all neighbor luminaires.

5.4 Final Controller

As it was stated before, after reaching the local optimal solution, the node exchanges information about its solution with its neighbours. As a response, it receives its neighbors solutions in order to compute the average (step 2). Since all luminaires compute their local solution and receive the solution from the others, the average dimming value vector is suppose to be the same in all nodes.

After several iterations, convergence should be met. In that situation, the resulting $\mathbf{u}$ is used as a reference to the local controller. The consensus algorithm is then used to replace the feedforward controller described earlier: it sends the reference actuation value directly to the actuator and is used as a reference for the expected illuminance to be used in the predictor.

As a final remark, care must be taken regarding the lower bound parameter. This parameter sets the lower limit for the illuminance of the local station. In this system it only has two possible values: 100 Lux (if the station is "occupied") or 20 Lux (if the station is "not occupied"). In order to simulate this and to allow a change between the states, a switcher was integrated in each luminaire. Every time the switch is set on, the mode is changed into "occupied". In case of a change, the consensus algorithm must start iterating in order to reach an optimal solution for the actuation that must be set to reach the lower bounds in each station.

6 Data Server

In this section, the methods used to develop and implement the data server in the Raspberry Pi, using C++11 language are described. We begin by describing the server architecture. This is illustrated in figure 5, where it is represented the running threads, and the transmission of data among them, as well as outside communication. Note

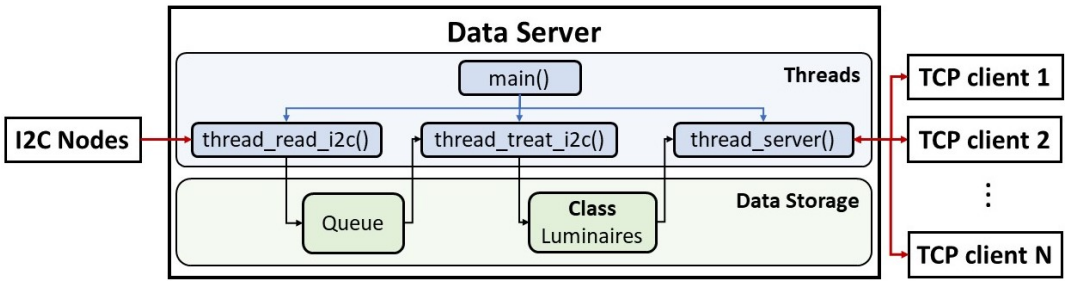


Figure 5: Diagram of the server's threads and flow of data. Red arrows: communications with the exterior; black arrows: flow of data in the server; blue arrows: creation of threads.

that one way arrows implies that communication occurs one way. It is also represented some of the "containers" used for data storage, such as the class `Luminaires`. The important classes used in the server coding are:

- `Read_I2C` - This class implements an abstraction of the PIGPIO library for I^2C, simplifying bus monitoring;
- `Connection` - An object of this class represents a connection to a client. It holds the connection's attributes, and handles client requests;
- `Server` - An object of this class runs an acceptor, waiting for new client connections and creates a new instance of the class `Connection` for each accepted connection;
- `Luminaires` - This class holds the server data as private variables, and provides methods for updating and accessing the data. It also implements the synchronization, and mutual exclusion, to avoid multiple threads accessing the data simultaneously.

Lets now begin by addressing the way data is acquired from the system nodes.

6.1 Data Acquisition

In order to implement our server, it is necessary to establish a communication link with the luminaires, in order to receive data. This is done using I^2C communications, where the Raspberry Pi acts as a slave, and listens to

communications on the bus. In order to guarantee that 3.3V reached the Raspberry Pi, instead of the Arduino's 5V, it was necessary to use pull down resistors, as in the scheme illustrated in figure 15 in appendix. This enables the Raspberry to read the line, however, it is not possible for it to write in the line. It might even be harder to set the line to LOW, due to the external resistor. This way, there is no guarantee that the Raspberry can make an acknowledge (ACK). In order to overcome this, we decided to use the I^2C broadcast property for the Arduinos to write in the line. This way, the remaining nodes will also perform the ACK, and receive the message, thus guaranteeing that communication is coherent.

In order to continuously read the bus, a thread was used, represented in the diagram of figure 5 by `thread_read_i2c()`, which performs an active wait for communications in the bus, to assure no messages are lost. Every time a new message is read from the bus, the server data must be updated. However, there might be other operations being performed in the data, and thus the thread might be blocked waiting to update the data, and therefore, loose messages. To assure this does not happen, this thread will not handle the job of updating new data. Instead, it will associate a time stamp to the message and insert it into a queue, for another thread to process, avoiding to block. The thread that processes the data, `thread_treat_i2c()`, will then pull messages from the queue, and update the server data. If this thread gets blocked, some messages will accumulate in the queue, and after unblocked, it will manage all of this, emptying the queue.

As previously described, the class `Read_I2C` was created to simplify the interaction with the PIGPIO library. The class constructor handles the initialization of the GPIO pins and setup of I^2C communications. An object of this class is created in `thread_read_i2c()`, and then a class method can be used to simply perform the bus monitoring operation.

6.2 Data Storage

As previously described, a class named `Luminaires` was created to store the data and manage data access. An object of this class is created as a global variable, so that it can be accessed by all threads.

The system must implement two types of storage. In order to store the data in a manner that simplifies access to it when a request is made to the server, `unordered maps` where used. This allows to map a variable using some other variable. In this case, strings were used to map the variables. Two unordered maps where used, one for the current time data, which maps strings to floats, and other to the last minute buffer data, which will map strings to a C++ container[4]. This way, when a request of the type "g l $\langle i \rangle$" is performed (get lux at luminaire i), for example, data could be accessed by simply using the "l $\langle i \rangle$" sub-string.

In order to implement the last minute buffer, a dynamic container was used. To simplify implementation, and use an higher level of abstraction, C++ standard library's sequence containers were considered. At first glance, a `queue` container seemed appropriate, since we are able to remove older values from the end, and add new values to the beginning. However, the `queue` implementation does not allow iteration. Instead, a container of type `deque`, or double ended queue[5], was implemented, thus simplifying the process. Due to the type of implementation, `deque` also provides faster iteration than `list`[6].

In order to manage the last minute buffer, a method was defined to remove elements from the `deque`, if they are older than one minute. In order to do so, the data stored in the `deque` keeps a `timepoint`, from `chrono` library. Every time a new value is added, or a read operation to the buffer is performed, older values are checked.

Whenever the system is initialized, or a reset occurs, the data containers are erased, and a `timepoint` from the `Luminaires`' class object, holding the last reset time, is updated with the current time.

Lastly, notice that due to the implementation, the server is easily scalable. It is only necessary to change the type of message received in order to be able to serve more luminaires.

6.3 Data Processing

Data processing is done in the `Luminaires` class object. Whenever a new message is available, the thread `thread_treat_i2c()` calls a method from the `Luminaires`' class object, to update the data. This method receives the message and will update the adequate variables, held in the object. Mutual exclusion is implemented inside the class, in the methods that perform data update and data access, to prevent multiple threads accessing the data simultaneously.

[4] described in `http://www.cplusplus.com/reference/stl/`
[5] described in `http://www.cplusplus.com/reference/deque/deque/`
[6] see `https://baptiste-wicht.com/posts/2012/12/cpp-benchmark-vector-list-deque.html`

6.4 Server-Client communications

As previously described, in subsection 2.2, the server-client communications were implemented using TCP connections, which ensure reliable packet transmission.

A thread was created to run the server, `thread_server()`, as illustrated in the diagram of figure 5. This thread creates an I/O service object from Boost.Asio library, and then creates an object from the class `Server`. This object, accepts connections to port 17000, and creates a new object of class `Connection` every time a client connects to this endpoint. Every `Connection` class object contains a socket, and other attributes, specific to the respective client. This way, the number of sockets used to implement the server is one per client. Furthermore, these objects employ Boost.Asio library functions, in order to asynchronously handle requests from clients, thus allowing to handle multiple clients simultaneously, using a single I/O service object.

When a client request is received, a lambda function is executed in order to handle the request. Firstly, the request is validated, using a method from the class `Luminaires`. If this request is invalid, an appropriate response is sent to the client. In case it is valid, the request string is passed to the `Luminaires`' class object, which will return the appropriate response to be sent back to the client. In the case a streaming request is made, condition variables were used in order to signal when a new value was ready to be sent to the client. In this case, after each write operation, the callback lambda will call another write operation, until a request to stop streaming is received. To prevent the server from blocking in the case of system shutdown (blocks waiting for new variable), the condition variable's `wait_for()` method was used instead of the standard `wait()`, in order to implement a timeout. If there is no new value for a long period of time, the thread unblocks and an error message is sent back to the client, stopping the stream.

In order to simplify the connection to the server, as described before, a static IP address was defined for the Wi-Fi connection. This also simplified working with the RaspBerry Pi, since there was never the need to plug in the Ethernet cable. A possible way to improve this would be to setup an access point in the RaspBerry, so that it would not be dependent on external Wi-Fi network. A client could the connect to the RaspBerry's standalone Wi-Fi network and simply run the client application. Also, the server could be defined as a startup application in the RaspBerry, making this device, "plug-and-play".

7 Communication

Communication from Arduino board to Arduino board is essential to the correct execution of the calibration routine and of the control algorithm. Furthermore, the server requires information from each luminaire which needs to be sent from each Arduino. In this section the methods used on the data transmission between Arduino boards and Raspberry Pi are described.

7.1 Communication protocol

The controllers and the server are connected via I^2C (Inter-Integrated Circuit) bus. Each controller is considered a master allowing direct communication between them. However, as already referred in subsection 6.1 the Raspberry Pi acts as a read only slave.

The original Wire library for the Arduino was used to perform the communications via I^2C. Every communication is made through broadcast, i.e., the messages are sent to address 0x00. Even though this is not needed in a two luminaire system, since data relevant to the other controller only needs to be sent to one address, with N luminaires only one message with a specific information would be needed to reach every Arduino instead of N messages for each. The reason for broadcast transmissions to the Raspberry Pi were previously explained in subsection 6.1.

The library used has the clock frequency in the I^2C set as default to 100 kHz. This value was left unchanged. It was verified that in a 10 millisecond period, the number of times an Arduino sent a two bytes message and received an answer with the same size was in average 12. This way we can assume that the process of sending and receiving an answer takes approximately 0.83 ms. This test was performed with messages of two bytes as is the type used in the consensus algorithm for this project.

The maximum number of bytes per message when using the Wire library is 32.

The system is programed so that it resends a message sent between Arduinos if it detects an error. An error is detected either by a non-acknowledgment signal or by zero iterations made in one sample time. Regarding messages to the server, it is not possible to confirm if each message was received with the configuration used in the I^2C bus, making every message that is not received considered as lost. The server in its run shown negligible number of

packet losses, proving good enough behavior for this project.

There is a known problem with the Wire library running in a multi-master configuration[7]. This problem causes both Arduinos to stop execution of the loop when the function used to send data through the bus I^2C is called, making the project only able to run for a limited time. Introducing delays of $1\mu s$ before the sending process improved the operational time. However, no definite solution for this problem was implemented.

7.2 Message structures

Types of data packets were defined for the following tasks: calibration, consensus algorithm and data server. The last one can either be constant, i.e. only needed to be sent after the calibration routine is completed, or change every sampling time. In figure 6 the different types of message structures are represented.

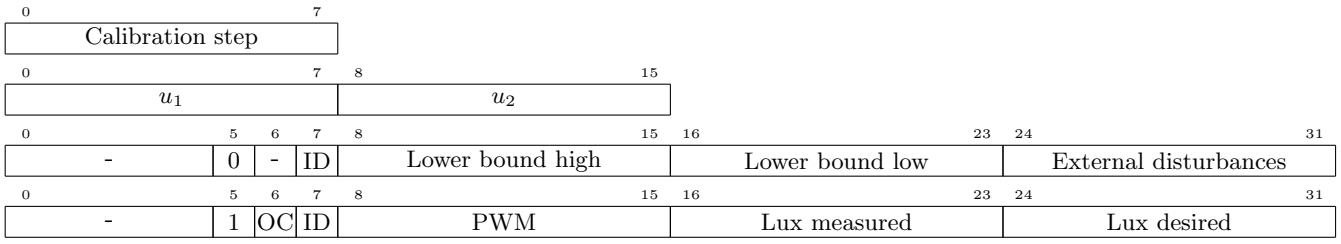


Figure 6: I^2C message structures for each four possible types: calibration, consensus, constant data server and variable data server. The numbers represent the bit indices. ID and OC refer to the sender identifier and the occupancy of the desk respectively

The size of the message received is used to identify its type and if it is relevant to the receiver since every message is sent in broadcast.

7.2.1 Calibration

During calibration each arduino computes the influence of each luminaire on its LUX sensor and the external disturbances. Each message transmitted in this procedure consist of one byte with the corresponding instruction. The codification of this procedure is implemented according to table 1.

Table 1: Calibration routine codification with the Arduino that executes the step and respective action.

Calibration step	Arduino	Action
0	1	Starts calibration. Turns LED OFF
1	2	Turns LED ON. Reads sensor.
2	1	Reads sensor. Turns LED ON.
3	2	Turns LED OFF. Reads sensor
4	1	Reads sensor. Turns LED OFF. Reads sensor. Computes **K** and **o**.
5	2	Reads sensor. Computes **K** and **o**.

To assure correct measurements, delays were introduced between a change in one LED and a sensor read. This happens once per each luminaire in the system plus once before external disturbances are measured, making this process slow if the system is composed by many LEDs. Since only two LEDs are used, only three times this delay is needed, taking each delay the duration of 1 second acceptable. After the computation of gains and disturbances, one delay of 1 second occurs, finishing the process of calibration in about 4 seconds.

The code developed for this process was the same for the two arduinos. Any arduino that restarts sends a one byte message with value 0, executing the procedure as arduino number 2. However, with an increase in the number of luminaires, scalability would be a problem due to the need to program each individual message.

7.2.2 Consensus algorithm

As seen previously in subsection 5.3, with each iteration of the consensus algorithm the vector of dimming values **u** needs to be exchanged between controllers. This is accomplished using messages with two bytes. Each

[7] https://github.com/arduino/Arduino/issues/1313

byte contains an entry of the vector. Each entry is an unsigned integer, between 0 and 255, since it corresponds to the PWM number given as output of the feed-forward controller. The algorithm only continues iterating after receiving the message containing **u** computed by the other Arduino.

If the number of luminaires would be increased to N this type of data packet could be adapted to contain an identification number in one byte of the sender plus the vector **u**. The final size of the message would be $N + 1$ bytes, with a maximum of 31 luminaires using only one message.

7.2.3 Server data

To communicate between one Arduino master and the Raspberry Pi slave two types of messages with four bytes were implemented. The third least significant bit of the first byte indicates if the message contains constant values or variables. Constant values are sent when the calibration routine ends. Variables are sent once every sampling time. Every value is sent as an unsigned integer, between 0 and 255.

Common to both types of message is the ID of each arduino. Since we only use two arduinos only one bit is necessary to identify the sender. As five bits of the first byte are not used, this could be used to increase the number of IDs supported, resulting in up to 32 luminaires at the same time.

8 Experiments

In order to evaluate the characteristics of the described system, several experiences were performed. In this section, all experiments and their objectives are described. In the first set of experiments, the local characteristics are tested, using the first control setup to show the correct behaviour of the system when a certain amount of illuminance is requested by the user. The second part consists on testing the distributed controlled system with two luminaires. For this case, the equations that were reached in the second part of the report are simplified (in the Annex section the reader can analyze the resulting equations for the consensus algorithm). Also, the communications will be simplified for the two luminaires system. As for the resulting system, the objectives of the tests are to prove the benefits of using this control architecture (cooperative) over the non-cooperative control architecture.

8.1 Local Control

In this section, the experiments performed on the local systems are described. The objective of these experiments is to guarantee a good performance of the single luminaire system, which is the result of a quick response, a small (or nonexistent) overshoot and an effective tracking of the reference illuminance. For these purposes a set of features such as the FB+FF architecture, the anti-windup and the deadzone were introduced, as described earlier. The result of this integration was tested in order to prove the effect stated before. In addition, a test of the predictor effectiveness, disturbance rejection and controller jitter are also included.

8.1.1 Predictor effectiveness

In order to test the predictor effectiveness, different illuminance levels were requested to the open loop system (feedforward only). In this experiment, the values read by the sensor were compared to output values of the predictor which are suppose to represent the expected values of illuminance (in Volts). This was quantitatively evaluated by computing the sum of squared error throughout the experiment

8.1.2 Feedback + feedforward architecture

In order to demonstrate the benefits of this architecture, three controllers were developed and tested. First, the feedforward controller was applied to the system, and several requests were performed, to evaluate the accuracy of the open loop system. In addition, a disturbance was introduced to verify the behaviour in these conditions. After this experiment, a feedback controller (PI) was tested, in order to observe the behaviour of the closed loop system and the disturbance rejection properties. Finally, feedback and feedforward were combined into the desired control architecture and the resulting system properties were tested, in order to verify the desired performance and to adapt the feedback gains to this control architecture. The objective of these tests is to prove the advantages of this architecture over the single FF and single FB controllers.

8.1.3 Windup vs Anti-windup

Regarding the windup effect, the anti-windup feature was tested by requesting an illuminance level that was impossible to reach. After a few seconds, the request was changed to 100 lux (a value that can be easily reached). The windup effect was compared for systems with and without the anti-windup feature.

8.1.4 Deadzone

Regarding the deadzone, its inclusion in the measured error was tested by comparing the behaviour of the system with and without this feature. The experience consisted on observing the step response of both systems and the actuator output variation in steady state.

8.1.5 Disturbance Rejection

As previously described, the feedforward system does not reject disturbances (caused from external light). In order to simulate this disturbance, a window was included in the box. When opened, the exterior light influenced the illuminance level inside the room, changing (or not) the system behaviour. This experience was performed for the FF, FB and FB+FF controllers.

8.1.6 Controller Jitter

The jitter is defined as spurious variations of the pre-defined sampling time. This can be caused by non adequate use of temporal primitives or preemption of the real-time clock due to higher priority interrupts and can introduce random errors in the system's output value. Since the local system is quite simple and it uses interrupts and proper time primitives, jitter is expected not to be significant. Nevertheless, it was measured in order to validate this statement. To do so, the time between sensor readings was measured. This is expected to be equal to 10ms, or at least to oscillate around this value.

8.2 Distributed Controlled System

The second phase of the experiments regards the distributed cooperative control system. Their objective, as stated before, is to validate the advantages of this system over a non-cooperative architecture. In order to do so the two architectures were tested in order to determine which performs best. This test consists of successive changes in the state of occupancy of both stations, one at a time. During this transition and consequent steady state, three metrics were computed in the data server. These were used to quantitatively measure the comfort and efficiency of each station, in particular, and inside the room, in general.

8.2.1 Energy consumed

For this project, an important objective is to minimize the energy consumption while providing comfort to the users. The energy consumed is defined as the accumulation of the instantaneous power along time. Supposing the maximum power of luminaire j is denoted as P_j, then the energy consumed at each desk is

$$E_j = P_j \sum_{i=1}^{N} d_{i-1}(t_i - t_{i-1}), \tag{8.1}$$

where N is the number of samples since the last reset. Also, the maximum power was assumed to be $1W$.

8.2.2 Comfort Error

While energy minimization is simple to formulate, comfort criteria are more subjective. The system should prevent periods of illumination below the minimum settings defined by the occupation state. To access this criterion, the average between the reference illuminance and the measured illuminance when the measured illuminance is below the reference, was computed:

$$C_{error} = \frac{1}{N} \sum_{i=1}^{N} \max\left(l_{ref}(t_i) - l_m(t_i), 0\right) \tag{8.2}$$

Where N is the number of samples since the last reset.

8.2.3 Comfort Flicker

Another way to measure the comfort is by testing the frequency the number of ups-and-downs of illuminance (flickering) while the reference is at constant value. To do so, a metric can be defined as the average magnitude of the signal derivatives when it changes sign, during periods of constant occupation:

$$C_{flicker} = \begin{cases} (|l_i - l_{i-1}| + |l_{i-1} - l_{i-2}|)/(2T_s), & \text{if } (l_i - l_{i-1}) \times (|l_{i-1} - l_{i-2}|) < 0. \\ 0, & \text{otherwise.} \end{cases} \tag{8.3}$$

Both the Comfort Error and the Comfort Flicker were computed and evaluated using data stored in the server.

9 Results

In this section the plots corresponding to the results of the experiments are presented. In each subsection the results of the corresponding experiment described in the previous section.

9.1 Local Control

9.1.1 Predictor effectiveness

The results for the test of the predictor effectiveness are presented in figure 7. Note that the predictor is quite accurate as the difference between the prediction and the measure is only slightly different. With an accurate predictor, the performance required can be easily obtained.

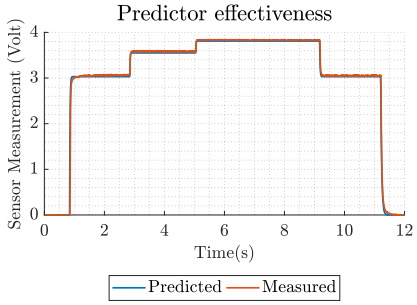
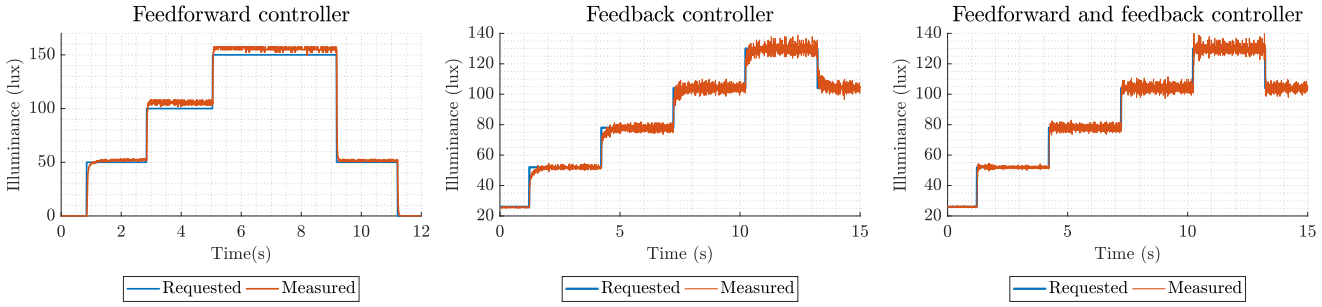


Figure 7: Predictor measurements

9.1.2 Feedback + feedforward architecture

In this section three different plots are presented, corresponding respectively to the experiments performed over the feedback controller, the feedforward and the combined architecture.



(a) Local control using FF control (b) Local control using FB control (c) Local control using FF+FB control

Figure 8: Feedforward control, feedback control and feedback+feedforward control

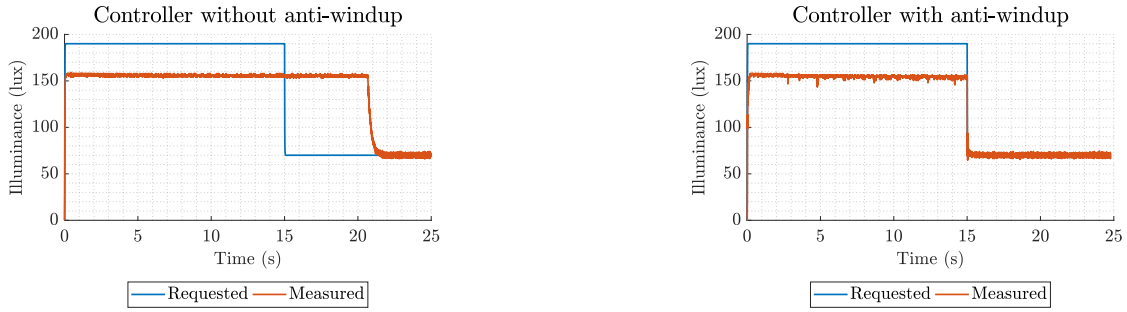
The feedback controlled system presents a slow response to a change in the reference illuminance. However, although it is not presented, this architecture presents robustness to disturbances. This property was tested using the window that was inserted on the box. By opening the window, external illuminance enters inside the box and influences the LDR measurements. As it can be noted, the system responds to this change by decreasing the LED's dimming value.

The feedforward controller have an almost instantaneous response (which goes along with the explanation provided in the respective section). However, it is not robust to disturbances. It is also important to note that the system is quite accurate, as it can be seen by the difference between the response of the system and the reference. This is the result of a well performed calibration of the system gains whose procedure was discussed earlier.

Finally the combined architecture performance is tested. As it can be seen it represents the best performance when compared to the others, as expected. The time of convergence is less then that of the feedback control and there is no sight of overshooting. In addition, it is robust to disturbances caused by external illuminance.

9.1.3 Anti-windup

The response of the system without the anti-windup is shown in figure 9. It can be noticed that after the second change in reference, the system took time a lot of time to react to the change. This is explained by the cumulative error in the integrative term of the PI controller, created by setting the reference too high. By including the anti-windup feature, it is shown that the response became faster.



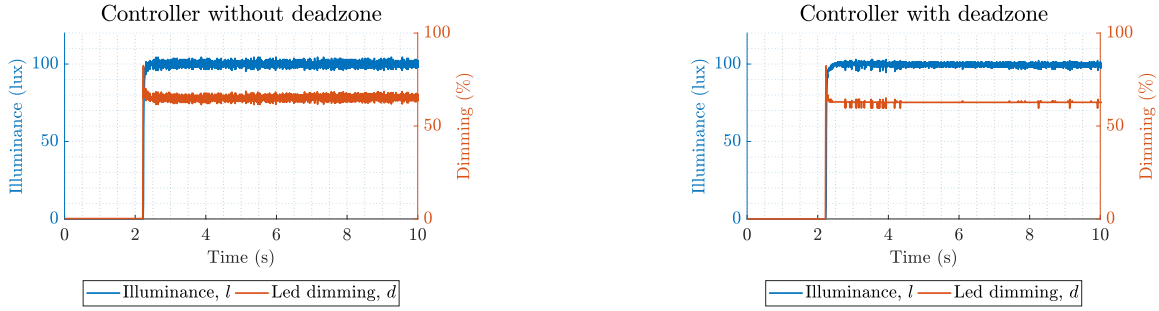
(a) Local control without anti-windup.

(b) Local control with anti-windup.

Figure 9: Experiment to evaluate the use of the anti-windup.

9.1.4 Deadzone

Just like the anti-windup feature, the deadzone was also tested. Following the procedure described in the previous section, the system was tested. The results shown in figure 10 prove that with the actuator deadzone, the system is less reactive to noise in steady state. This can be concluded, by noticing the dimming value keeps constant when the system is in steady state. As a consequence, the system decreases its oscillations in the illuminance, and decreasing the flickering.



(a) Local control without actuator deadzone.

(b) Local control with actuator deadzone.

Figure 10: Experiment to evaluate the use of the deadzone.

9.1.5 Jitter

In figure 11 the time interval between two actuator output is shown. It can be concluded that the time interval is almost constant throughout the experiment, oscillating around 10ms (which corresponds to the sampling time). The amplitude of the oscillations is low with the exception of a few exceptional situations. After reaching steady state, the jitter almost ends.

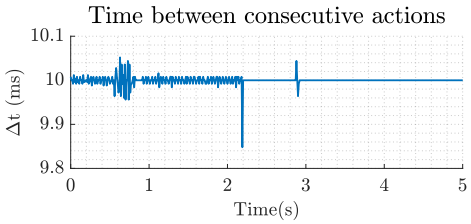
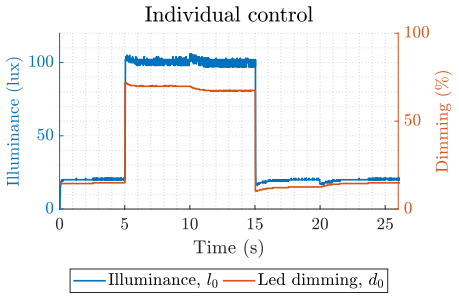


Figure 11: Time interval between actuator output

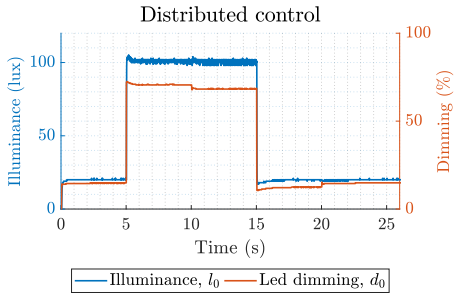
9.2 Distributed Control System Performance

As it was stated before, the main objective of this project is to conclude the advantages of a cooperative control system over the traditional PI non cooperative control system. In this section, a a comparison is developed and a final discussion regarding the advantage of this systems is presented.

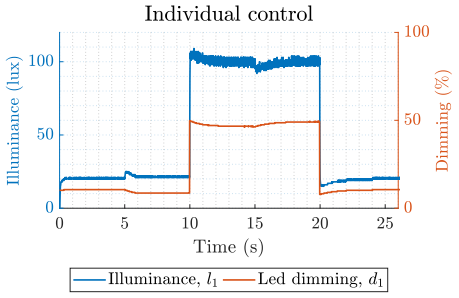
By observing the plots presented in figure 12 it is possible to notice that the the general behaviour is according to the expected good behaviour for both systems. However, the response developed by the cooperative design has less oscillations and has a fine actuation. This is expected according to a well designed architecture of both PI non-cooperative and cooperative system. However, the difference can be seen in the the comfort metrics. In general, the distributed control achieves a less comfort error, comfort flickering and energy consumption as it can be seen in figure 13. This fact goes accordingly to the expected behaviour discussed earlier, since the main goal of the cooperative control is also minimize the energy spent.



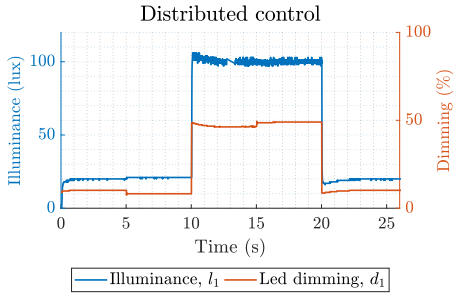
(a) Non-cooperative controller - response in node 0



(b) Cooperative controller - response in node 0

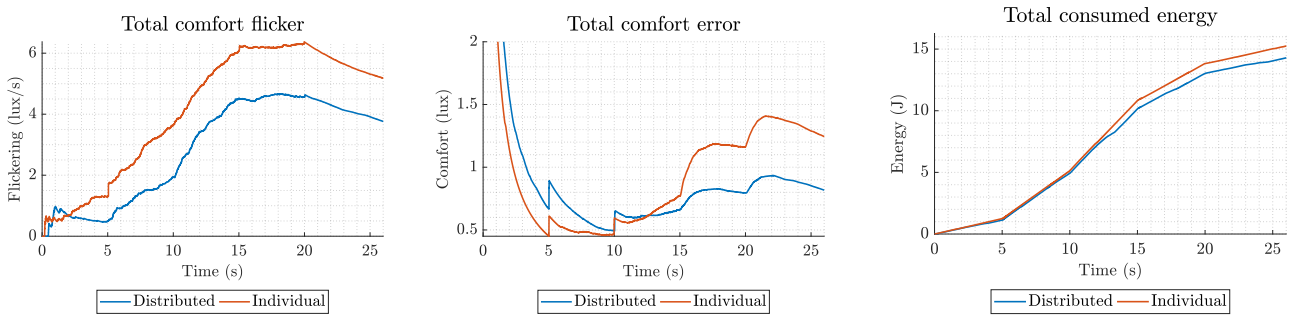


(c) Non-cooperative controller - response in node 1



(d) Cooperative controller - response in node 1

Figure 12: Representation of the response of the non-cooperative and cooperative control systems



(a) Comfort flicker for cooperative and non-cooperative controlled systems

(b) Comfort Error for non-cooperative and cooperative controlled systems

(c) Total consumed energy for non-cooperative and cooperative systems

Figure 13: Representation of the resulting metrics of the cooperative and non-cooperative control system

10 Conclusion

The implementation of an efficient distributed real-time control system applied to a simplified model of a small office was achieved with success. The change from individual control to distributed control using the consensus algorithm resulted in a significant improvement to comfort and a drop in energy consumption as these were the main objectives of the project.

These results were obtained starting with the identification of the system. Then an appropriate controller for one luminaire was designed using a feedforward and a feedback components. The feedback controller was then improved with features as the anti-windup and the error deadzone, resulting in a better response of a single luminaire. With the introduction of second LED distributed control was implemented using the consensus algorithm and communication between control boards. A server was implemented as to a client can access data relevant to system monitoring.

The system has aspects were further development should be done. Calibration is one of them as the communication process is not practical for scalability. Improvements on the Arduino library used for the I^2C are needed as known problems interfered with the system communications.

Repository

`https://github.com/ptrindade96/SCDTR-Proj`

11 Contributions

	Report	**Project**
João Santos	Control; Experiments; Results	Local Controller; Consensus; Distributed Controller.
Miguel Luis	Overall System Architecture; Communication; Results; Conclusion.	System calibration; Arduino-Arduino Communications; Arduino-Raspberry Pi Communications.
Pedro Trindade	Introduction; Concepts; System Identification; Data Server.	Data Server; System Identification; Consensus; RaspBerry Pi setup.

References

[1] "Commercial Buildings Energy Consumption Survey", U.S. Energy Information Administration, 2012

[2] T. Dwyer, "Lighting control technologies and strategies to cut energy consumption", CIBSE Journal, 2010

[3] D. Caicedo, A. Pandharipande, G. Leus, "Occupancy-based illumination control of LED lighting systems", 2010

[4] D. Caicedo, A. Pandharipande, "Daylight integrated illumination control of LED systems based on enhanced presence sensing", 2010

[5] D. Caicedo, A. Pandharipande, "Smart indoor lighting systems with luminaire-based sensing: A review of lighting control approaches", 2015

[6] S. Boyd, N. Parikh, et all "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers", 2016

[7] J. Valdez, J. Becker, "Understanding the I^{2}C Bus", Texas Instruments, 2015

[8] B. Stroustrup, "The C++ Programming Language", Pearson Education, New Jersey, 2013

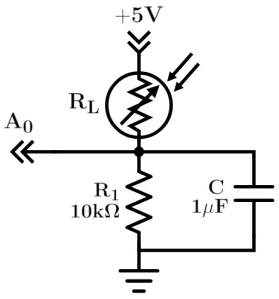
[9] D. Kalev, "The Biggest Changes in C++11 (and Why You Should Care)", 2011

[10] T. Wescott, "PID Without a PhD", Wescott Design Services, 2016

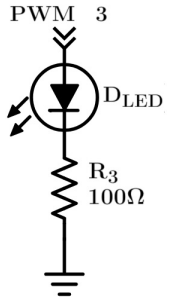
[11] R. J. Barnes, "Matrix Differentiation", 2006

Appendices

Sensing and Actuation circuits



(a) Illuminance sensor implementation: C is a capacitor, R_1 represents a known resistor, R_L represents the LDR and A_0 an analog read pin from Arduino UNO.



(b) LED driving circuit implementation: D_{LED} represents the LED, and R_3 represents a known resistor.

Figure 14: Representation of the circuits used with the Arduino UNO in order to implement the luminaries.

I^{2}C bus connection for the Raspberry Pi

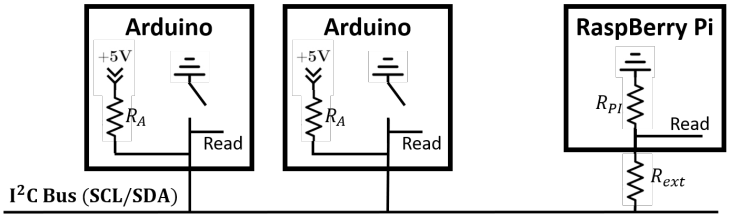


Figure 15: Diagram of the RaspBerry Pi's I^{2}C connection, using external resistors of value $R_{ext} = 3.3k\Omega$

Consensus local optimization computation

As described in subsection 5.3.2, the local optimization problem can be described by

$$\mathbf{u}_i(k+1) = \underset{\mathbf{u}_i \in \Omega_i}{\operatorname{argmin}} \left\{ \frac{1}{2}\mathbf{u}_i^T \mathbf{B}_i \mathbf{u}_i - \mathbf{u}_i^T \mathbf{z}_i \right\},$$

and thus we have a convex quadratic optimization problem. Also, as previously described, the solution may lie in the interior of the feasible region, or in its boundaries. Lets then define the solution in each of this regions.

Interior solution

For the interior solution, since we are dealing with a convex optimization problem, this corresponds to the global solution, or unconstrained solution. This is described by

$$\nabla J\left(\mathbf{u}_i\right) = \mathbf{0}.$$

Thus, using matrix differentiation properties [11]

$$\nabla J\left(\mathbf{u}_i\right) = \nabla\left(\frac{1}{2}\mathbf{u}_i^T\mathbf{B}_i\mathbf{u}_i - \mathbf{u}_i^T\mathbf{z}_i\right) = \frac{1}{2}\left(\mathbf{B}_i + \mathbf{B}_i^T\right)\mathbf{u}_i - \mathbf{z}_i,$$

and remembering the symmetry of matrix $\mathbf{B}_i$

$$\nabla J\left(\mathbf{u}_i\right) = \mathbf{B}_i\mathbf{u}_i - \mathbf{z}_i = \mathbf{0} \implies \mathbf{u}_i^{(\mathrm{un})} = \mathbf{B}_i^{-1}\mathbf{z}_i,$$

where $\mathbf{u}_i^{(\mathrm{un})}$ represents the global unconstrained solution. Also, remember $\mathbf{B}_i$ is diagonal, and therefore its inverse is of simple computation.

Boundary solution

Whenever the interior solution does not belong in the feasible solution, i.e., it does not obey to the constraints, then the solution should be in the boundary. Notice that, locally, a boundary solution may not be feasible, so the choice of correct solution resumes to verifying which of the solutions obtained in different bound constraints is feasible and provides the minimum cost solution. The boundary domains were also described in subsection 5.3.2. Lets then define the boundary as

$$\partial\Omega_i = \bigcup_{m=1}^{M} \partial\mathcal{B}_i^{(m)} \cap \Omega_i,$$

where M represents the total number of individual constraint regions and their intersections, in this case, there are three constraints, and their intersection provides for an extra three regions, however, one of them is not feasible ($\mathrm{DUB} \cap \mathrm{DLB} = \varnothing$), thus resulting in $M = 5$. Further, we can write

$$\partial\mathcal{B}_i^{(m)} : \mathbf{A}_i^{(m)}\mathbf{u}_i^{(m)} = \mathbf{b}_i^m,$$

and define for the involved constraints and intersections in matricial form

1. DLB : $\mathbf{A}_i^{(1)} = -\mathbf{e}_i^T$, $\mathbf{b}_i^{(1)} = 0$;
2. DUB : $\mathbf{A}_i^{(2)} = \mathbf{e}_i^T$, $\mathbf{b}_i^{(2)} = 255$;
3. ILB : $\mathbf{A}_i^{(3)} = -\mathbf{k}_i^T$, $\mathbf{b}_i^{(3)} = o_i - L_i$;
4. $\mathrm{DLB} \cap \mathrm{ILB}$: $\mathbf{A}_i^{(4)} = \left[-\mathbf{e}_i \middle| -\mathbf{k}_i\right]^T$, $\mathbf{b}_i^{(4)} = \left[0 \middle| o_i - L_i\right]^T$;
5. $\mathrm{DUB} \cap \mathrm{ILB}$: $\mathbf{A}_i^{(5)} = \left[\mathbf{e}_i \middle| -\mathbf{k}_i\right]^T$, $\mathbf{b}_i^{(5)} = \left[255 \middle| o_i - L_i\right]^T$.

Now, a solution in each of this regions must be computed. This can be done using the Lagrangian method. The solution comes described by

$$\begin{cases} \nabla L = \mathbf{B}_i\mathbf{u}_i - \mathbf{z}_i + \mathbf{A}_i^T\boldsymbol{\lambda} = 0 \\ \mathbf{A}_i\mathbf{u}_i - \mathbf{b}_i = \mathbf{0} \end{cases}, \qquad \text{with } L(\mathbf{u}_i, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{u}_i^T\mathbf{B}_i\mathbf{u}_i - \mathbf{u}_i^T\mathbf{z}_i + \boldsymbol{\lambda}^T\left(\mathbf{A}_i\mathbf{u}_i - \mathbf{b}_i\right),$$

and can therefore be written as

$$\begin{bmatrix} \mathbf{B}_i & \mathbf{A}_i^T \\ \mathbf{A}_i & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{u}_i \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_i \\ \mathbf{b}_i \end{bmatrix} \implies \begin{bmatrix} \mathbf{u}_i \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_i & \mathbf{A}_i^T \\ \mathbf{A}_i & \mathbf{0} \end{bmatrix}^{-1}\begin{bmatrix} \mathbf{z}_i \\ \mathbf{b}_i \end{bmatrix}.$$

Computing the matrix inverse and simplifying, one can obtain

$$\mathbf{u}_i = \mathbf{B}_i^{-1}\mathbf{z}_i + \mathbf{B}_i^{-1}\mathbf{A}_i^T\left(\mathbf{A}_i\mathbf{B}_i^{-1}\mathbf{A}_i^T\right)^{-1}\left[\mathbf{b}_i - \mathbf{A}_i\mathbf{B}_i^{-1}\mathbf{z}_i\right].$$

Further, scalar expressions for the $\mathbf{u}_i$ vector components, for each of the M regions previously described were determined, with the aid of MatLab's symbolic toolbox, in order to simplify programming implementation, and provide for a faster code.