TÉCNICO LISBOA

Master Degree in Electrical and Computer Engineering

Distributed Real-Time Control Systems

Project

Distributed Lighting Control

Group: 10

Frederico Lourenço 78645
Daniel Sousa 79129
Jorge Gonçalves 79154

January 9, 2018

Contents

1 Abstract

This project consists on designing a distributed control system coupled with a server with the objective of controlling the illumination level of a given room. The system implemented allows the user to set different illumination levels for different areas of the room using an external TCP client. It is equipped with the Consensus algorithm for decentralized coordinated control between all the defined areas, using bus network communication, in order to find a common optimal solution, while also being able to respond to direct commands and requests from the TCP client. This report provides a compilation of the results obtained during the execution of the project so as to show the differences between coordinated and uncoordinated control, the impact of using Feedforward and PI control and some results from the system during operation.

Keywords: PI controller; Feedforward; Decentralized Coordinated Control; Consensus Algorithm; ADMM; I2C communication; TCP/IP Client

2 Introduction

The problem being addressed in this report is relevant nowadays because most of the time the energy consumption in an office is well below optimal. This leads to unnecessary spending to the company and, because of the of the increase in energy demand, harm to the environment.

In order to simulate a room as described, a model for a small office with only two luminaires was considered. To represent the physical boundaries of the office, a shoe box was used with two Arduinos UNO as controllers of the two luminaires considered and a Raspberry Pi (RPI) to implement the TCP server. The two Arduinos are connected to a breadboard, each controlling a circuit with a LED and a Light Dependant Resistor (LDR). The 5V pins in each Arduino are connected to each other in order to allow the system to operate with only one power source. This connection can be made between only two Arduinos, since the power needed for one Arduino can be supplied by the other but more than that is not recommendable (for a bigger network more power sources would be required). Figure 1 shows a photograph of the project's model seen from above.

The Arduinos communicate using the I2C protocol, and the RPI uses Serial communication and an I2C passive "Sniffer" as well. Although the system has only two luminaires it has the potential to be applied to a situation with many more without having to be subjected to many changes.

During the development of the project, different types of local controllers were studied: Feedforward, Feedback and Feedforward & Feedback controllers. The advantages from each controller are discussed in this project.

With the local controllers designed, a distributed control algorithm based on communication between the luminaires was implemented to improve the efficiency of the overall system. This algorithm allows the controllers to agree on reference levels in the different regions of the room, based on the user's requested levels.

The TCP server was implemented in a C++ environment. It is able to receive requests from various clients, change the requested luminaires to the wanted levels and retrieve information asked by the client. It is also described, how the server deals with parallel tasks and the methods implemented to guarantee that the concurrency of tasks does not

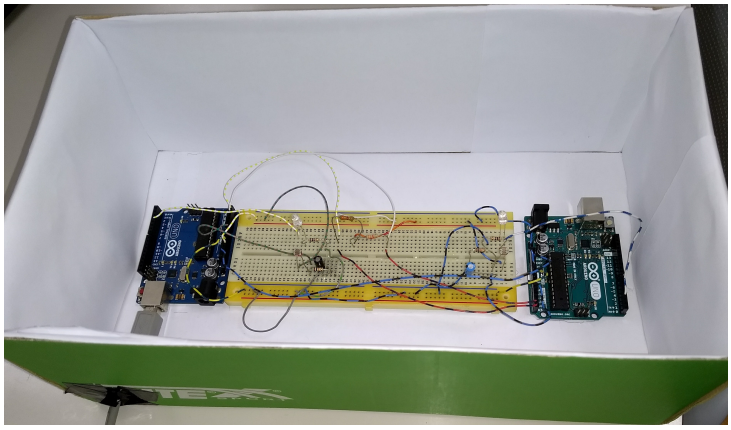


Figure 1: Project's model of an office with 2 luminaires.

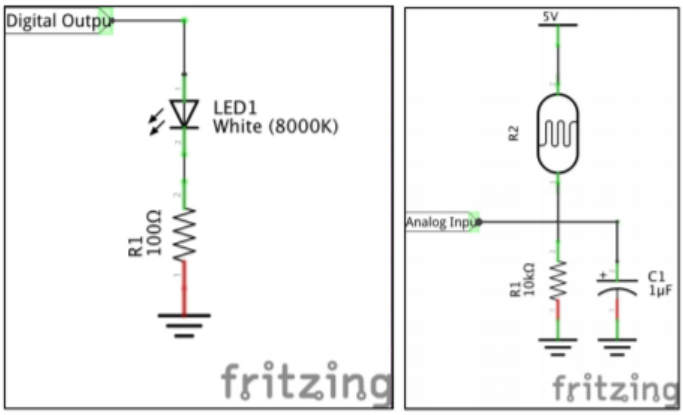


Figure 2: Luminaire circuitry. Source: [1]

compromise the good functioning of the server.

3 Approach

The Arduino's based circuit used for each luminaire is shown in Figure 2. The *Digital Output* is connected to the digital port number 3 of each Arduino. This port was used because it supports Pulse Width Modulation (PWM) and it does not use the *Timer1* so it does not affect the *delay* function.

The *Analog Input* is connected to the Arduino's Analog Input port *A0*. For Arduino to measure an analog signal, an Analog to Digital Converter is used. It outputs values from 0 (0 Volt) to 1023 (5 Volt) with unit steps. So a tool, based on the circuits of Figure 2, that converts the values measured by the Arduino ADC to lux is required, i.e., a luximiter.

3.1 Lux reading calibration [luximiter]

Analyzing the datasheet [2] of the LDR used in this project, one can obtain the characteristic of the illuminance vs. resistance, as seen in Figure 3. The line in red seen in the Figure represents an approximation for the resistance of the LDR in function of the

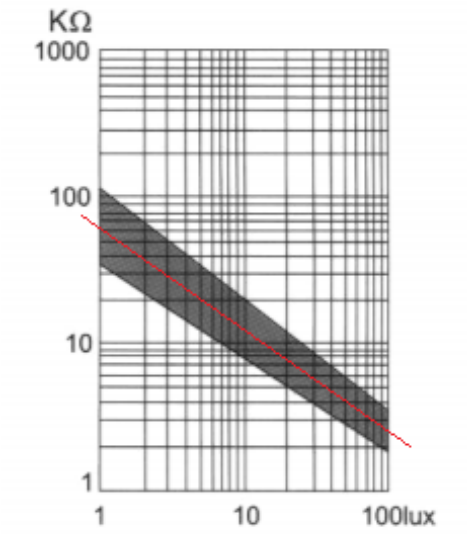


Figure 3: LDR Characteristic Illuminance vs. Photo Resistance and linear approximation (red line). Source: [2]

illuminance it is subject to. Since the graph has logarithmic scale the line is defined by

$$R\ [k\Omega] = L\ [\text{lux}]^m \times 10^b, \tag{1}$$

where L is the measured illuminance, $m = -0.692$ and $b = 1.808$. In a linear scale the line is defined by $R\ [k\Omega] = -0.692 \times L\ [lux] + 1.808$.

So to complete the luximiter one needs to invert (1) and find the value of illuminance for a given LDR resistance. The equation then becomes

$$L\ [\text{lux}] = 10^{\left(\frac{\log_{10}(R\ [k\Omega]) - b}{m}\right)}. \tag{2}$$

The value of the LDR resistance can be obtained through circuit analysis:

- Voltage measured by the Arduino: $v = ADC \times \frac{5}{1023}$;
- Current in the LDR: $i_{LDR} = \frac{v}{R_1}$;
- Voltage in the LDR: $v_{LDR} = 5 - v$;
- Resistance in the LDR: $R_{LDR} = \frac{v_{LDR}}{i_{LDR}} = \frac{5-v}{\frac{v}{R_1}} = \frac{R_1(5-v)}{v} = \frac{R_1(5-ADC\times\frac{5}{1023})}{ADC\times\frac{5}{1023}}$.

Since R_1 is known, for each reading of the Arduino ADC the resistance of the LDR is computed and using equation (2) the value of the illuminance level (lux) is obtained.

To reduce measurement errors, a simple "filter" was developed. Five measurements from the Arduino are taken at each sample time and the average is computed. The average value is then used to compute the lux value.

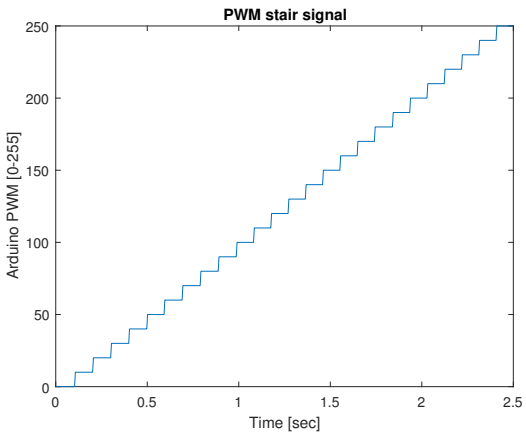


Figure 4: PWM input stair signal

3.2 Individual System Identification

To develop a controller for this system, one needs first to characterize and produce its mathematical model. Starting by the characterization of each luminaire, an analysis of the steady-state response to a "stair" signal input is required. This signal is meaningful because it produces the response for a wide range of possible inputs. In this case, steps with amplitude 10 and duration 0.1 sec were used, going from 0 to 250, as depicted in Figure 4.

The duration in each step allows stabilization, in order to avoid erroneous readings in transient periods between steps. Since the system in study has, as possible input values for the LED PWM, duty cycle values between 0 (0% duty cycle) and 255 (100% duty cycle), the stair signal illustrates the response of the system for a wide portion of possible inputs, giving a precise characterization for this system. Also, multiple readings were made in each step in order to increase the data used for the system's identification.

The response of the system to the stair signal can be seen in Figure 5a. It shows the relation between duty cycle and the voltage drop in the LDR, as read by the Arduino's ADC. To convert this values to voltage (in Volt), one only needs to compute a simple equation,

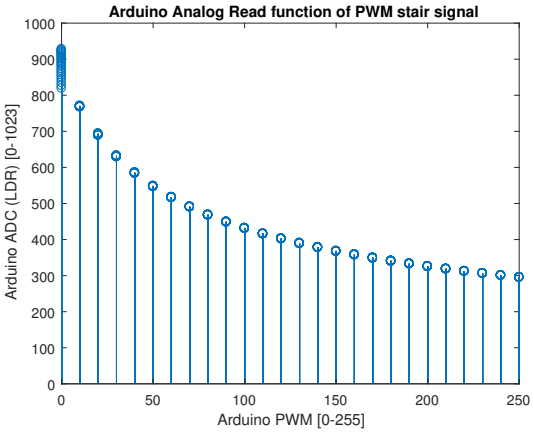
$$LDR_{Volt} = LDR_{ADC} \times \frac{5}{1023}, \tag{3}$$

since the Arduino can only measure up to 5 Volts. The constant $\frac{5}{1023} = 4.9$ mV, corresponds to the Arduino's quantization interval.

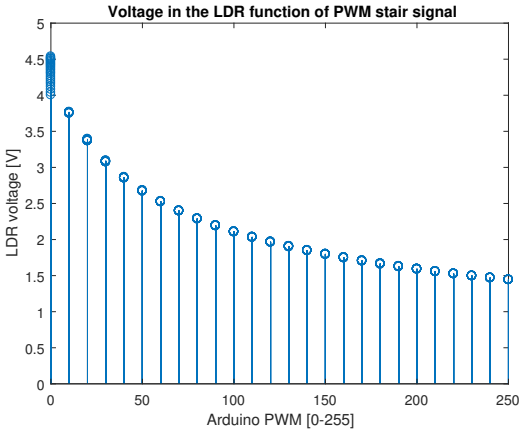
Using this conversion, one can plot the values of the voltage in the LDR as a function of the PWM stair signal, as is depicted in Figure 5b.

Both 5a and 5b show that the values obtained directly from the LDR vary non-linearly with the variation in the LED duty cycle (PWM values). In order to linearize this relation one can convert the LDR values to lux as described in section 3.1 and shown in Figure 5c.

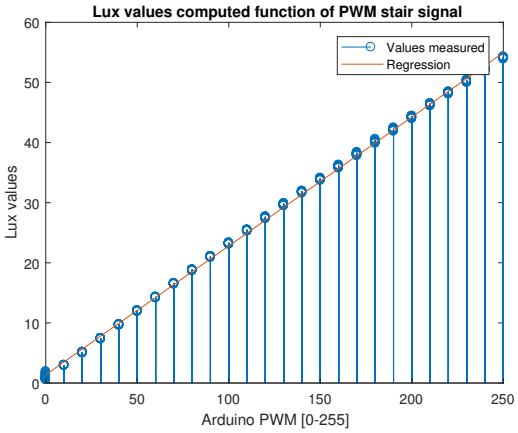
Now, as it is depicted in Figure 5c, the relation between duty cycle and lux is linear. This makes it possible to approximate the parameters of the line through a linear regression of the data measured. The Figure also depicts the line obtained, which is defined by: $y = 0.2143x + 1.3721$. This linearization simplifies the conversion between duty cycle



(a) LDR [ADC] vs PWM input signal



(b) LDR [voltage] vs PWM input signal



(c) LDR [lux] vs PWM input signal

Figure 5: System's response to the input stair signal

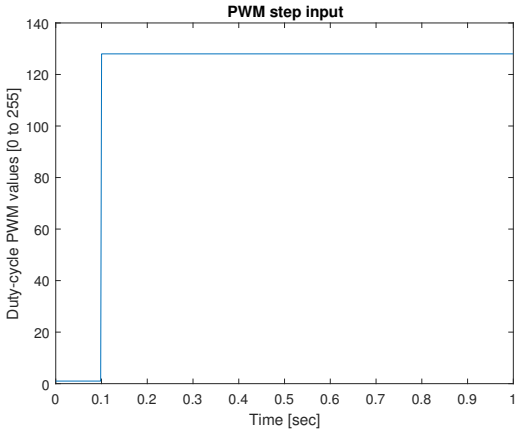
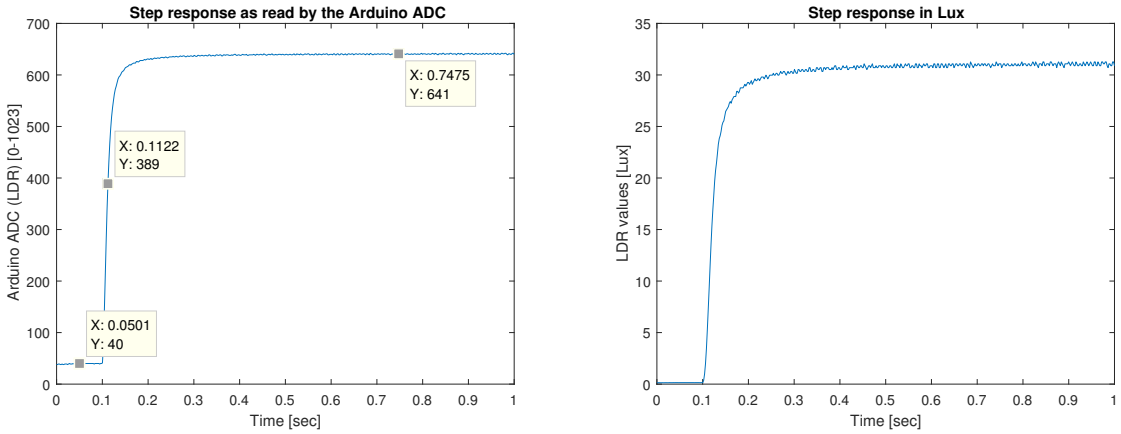


Figure 6: PWM step input



(a) Step response ADC

(b) Step response in lux

Figure 7: System's response to the input step signal

and lux values read in the LDR in each luminaire,

$$\text{lux} = 0.2143 \times \text{PWM}[0 \text{ to } 255] + 1.3721. \tag{4}$$

After characterizing the system, to conclude the individual system identification, one needs to analyze the step response.

In this case, a step signal with duration of 1 second and with the change in amplitude at 0.1 seconds was used. The amplitude of the step was equal to $128 - 1 = 123$. The step used has 1 as a lower value because it was noticed that the LED had some delay turning ON, and this was introducing error in the measurement of the time constant of the system. The upper value 128 was used since it represents the duty cycle at 50%. This system has different responses for different duty cycle levels, so the middle of the duty cycle scale was used as an approximation for the whole system. This input can be seen in Figure 6.

It is assumed that the local step responses observed in lux can be modelled as first order systems $G(s) = \frac{K_0}{1+s\tau}$, where K_0 is the static gain and τ is the time constant of the system.

The static gain can be obtained with the line defined in (4), K_0 is approximately the

slope of this line, $K_0 \approx 0.2143$.

The time constant τ is the time that the system takes to reach 63.2% of the steady state value. In order to obtain the time constant of the system correctly one needs to analyze the ADC values (0 to 1023), since the lux values were computed through a non-linear operation, so the 63.2% of the steady state value has no meaning for the lux values. As it is possible to see in Figure 7a, the steady state value is 641, the initial value is 40, so $(640 - 40) \times 0.632 = 379.83$. The system reaches the value 389, which is the closest value in the data, at time $t_2 = 0.1122$ s, but since the step was only made at time $t_1 = 0.1$ s, so $\tau = t_2 - t_1 = 0.1122 - 0.1 = 0.0122$ s $= 12.2$ ms. This means that the bandwidth of the system is $f_{BW} = \frac{1}{2\pi\tau} \approx 13.05$ Hz.

3.3 Coupled System Identification

In order to obtain the values for the coupled system identification, a calibration of the system is required. For this, three illumination measurements are required: when both LEDs are OFF; when one LED is ON and the other is OFF; and vice-versa. This is achieved through a synchronized combination of measurements and LED blinks.

It is already known from section 3.2 that the lux measured changes linearly with the duty cycle of the LED. So, in this case, it was only observed the case of the duty cycle equals to 255, i.e, 100% duty cycle (maximum brightness of the LED). The synchronization is achieved by connecting the *reset* port of the Arduinos to each other, so only one luminaire needs resetting to restart the whole system. The calibration process is the starting point of the system so it is guaranteed that this process is synchronous.

The values obtained with the calibration are the cross-gains matrix $(k_{11}, ..., k_{22})$ and the external luminance levels (o_1, o_2), that define the equations of the system in (5),

$$\begin{bmatrix} l_1 \\ l_2 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} \times \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} + \begin{bmatrix} o_1 \\ o_2 \end{bmatrix}, \tag{5}$$

where l is the lux measured at each luminaire and d is the duty cycle of each luminaire.

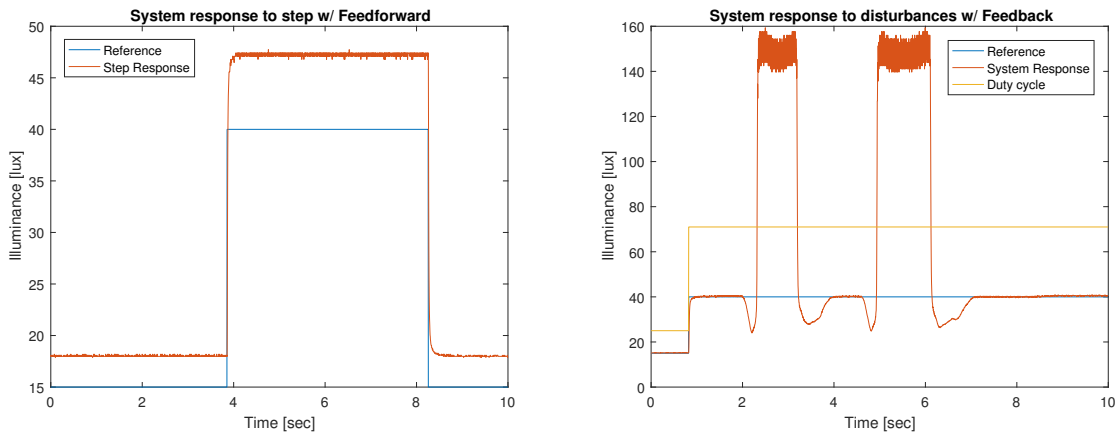
3.4 Individual Feedforward Control

The feedforward controller used was based in the system identification described in section 3.2. As seen, the relation between lux and duty cycle can be approximated by (4), so the controller has as input the reference value intended (in lux) and outputs the duty cycle value through

$$\text{PWM [0 to 255]} = \frac{\text{REF [lux]} - 1.3721}{0.2143}. \tag{6}$$

The response of the system with only this controller is depicted in Figure 8. As it would be expected, Figure 8a shows that this open loop controller alone can not satisfy the reference, even without disturbances, since the system's model is only an approximation, with some imperfections, of the real system.

Furthermore, this controller cannot respond to external disturbances. For example, an increase in external lighting (created by simply opening the box where the luminaires are), due to the fact that it is an open loop controller and therefore cannot perceive these disturbances. This can be seen in Figure 8b where, despite the change in illuminance,



(a) Feedforward without disturbances (b) Feedforward with disturbances

Figure 8: System's step response with only Feedforward with and without disturbances

the duty cycle does not change (the LED stays ON), which would not be necessary since the disturbance illuminance is above the reference.

3.5 Individual Feedback Control

Since the controller described in section 3.4, is not able to follow the reference accurately and could not respond to external disturbances, a closed loop controller needs to be used. A PI controller was chosen since the system needs to interact with people and does not need to react too fast. Fast changes in the lighting will cause flickering which is not comfortable for the user. The changes in the system can be relatively slow and therefore the Derivative part of the PID is not necessary, hence a PI controller is sufficient in this case.

Firstly, the sampling time needed for this controller was defined. For that it was used the rule of thumb which says that the sampling time needs to be greater than 20 times the bandwidth of the system. Since in this case the bandwidth of the system, as seen in section 3.2, is $f_{BW} = 13.05$ Hz,

$$f_s > 20 \times f_{BW} \Leftrightarrow f_s > 260.91 \text{ Hz}. \tag{7}$$

The sampling time then has to be less than

$$T_s = \frac{1}{f_s} = \frac{1}{260.91} \approx 0.0038 \text{ s} = 3.8 \text{ ms}. \tag{8}$$

For simplicity a sampling time of 3 ms was used. This sampling time was assured in the Arduinos through a *delay* function at the end of each control loop.

Then, using Tustin's method, the expression of the controller was defined by

$$u_{controller} = P + I, \tag{9}$$

where

$$P = K_p \times (\underbrace{\text{ref} - \text{lux}_{\text{lido}}}_{e}) \tag{10}$$

and

$$I = I_{\text{previous iteration}} + K_p \times K_i \times \frac{\text{sample time}}{2} \times (e + e_{\text{previous iteration}}). \tag{11}$$

This controller needs to be tuned to find the parameters K_p and K_i. The tuning method implemented is described in [3], and the values obtained were $K_p = 0.6$ and $K_i = 60 \times 10^{-6}$.

To further improve this controller, an Anti-Windup loop and a Dead-Zone were added. The Anti-Windup prevents the integrator to grow indefinitely. For instance, when there is so much external illuminance that the measured illuminance is far above the reference, the error would be large, but the actuator would saturate so the integrator would start to accumulate to a very large number. The opposite can also occur for the absence of illuminance, the integrator accumulates to a large negative number. When this external disturbance vanishes the integrator would take a long time to return from the large accumulated value, which would introduce unwanted delays in the system's response. To prevent this from happening, the controller has an Anti-Windup algorithm that checks if the output of the controller in the previous iteration would saturate the actuator, i.e., above 255 or below 0. If this happens, the value of the integrator remains the same as in the previous iteration, otherwise the integrator is computed with (11). This method prevents the integrator from accumulating to large numbers, reducing delays in the response.

A Dead-Zone between -0.5 and 0.5 was implemented to reduce the influence of quantization errors from the Arduino's ADC. A Dead-Zone has the drawback of introducing error in the response at steady state. For example, a reference of 15 lux will produce a response between 14.5 lux and 15.5 lux due to the Dead-Zone, however the overall response was improved.

The response of the system with only this Feedback controller is depicted in Figure 9.

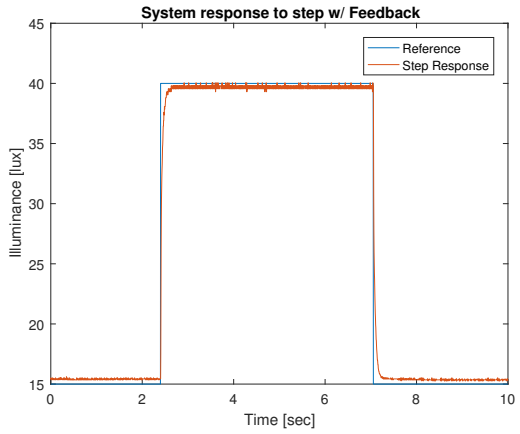
Unlike the results in section 3.4, the Feedback controller allows the system to follow the reference with an error close to zero at steady state, as can be seen in Figure 9a. Even with the presence of external disturbances, the system now responds as intended, as depicted by the duty cycle curve in 9b. For example, when the illuminance in the system was greater than the reference (Figure 9b), the LED was turned OFF completely. Otherwise, the LED adjusts its power to reach the illuminance level of the reference required.

3.6 Individual Feedforward & Feedback Control

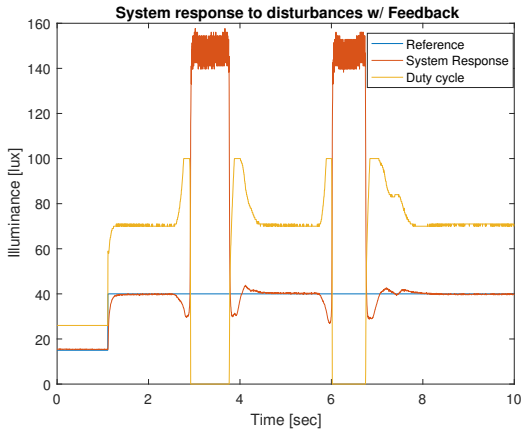
For an improved performance, a controller that is the combination of the previous two was implemented, combining the advantages of both. The system now has a faster response time to a change in the reference (characteristic of the Feedforward controller) and the rejection of disturbances and minimum error at steady state (characteristics of the Feedback controller). The control signal sent to the system is the sum of the two signals, that is then saturated between 0 and 255, since the actuation values are within this range.

The step response without disturbances is shown in Figure 10a and with disturbances in 10b.

It is notable that this response is faster than the previous but it shows an overshoot of about 20% which can be significant. This overshoot is introduced by the Feedforward

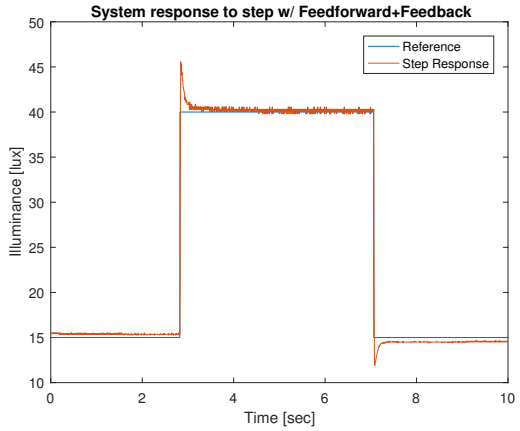


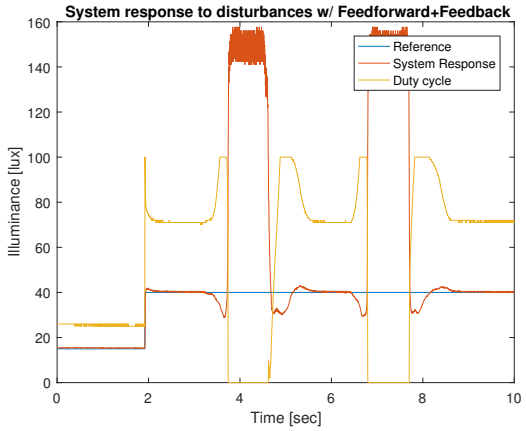
(a) Feedback without disturbances



(b) Feedback with disturbances

Figure 9: System's step response with only Feedback with and without disturbances





(a) Feedback + Feedforward w/o disturbances (b) Feedback + Feedforward w/ disturbances

Figure 10: System's step response with Feedback and Feedforward with and without disturbances

term and can be reduced by the tuning of the controller parameters (K_p and K_i). One can also reduce the overshoot by developing an internal method in this system that removes almost completely the Feedback term when there are no disturbances, or by decoupling both Feedback and Feedforward terms with a more complex controller.

3.7 Distributed Control

In sections 3.4, 3.5 and 3.6, a controller was developed considering only a single luminaire. This project aims to create a distributed coordinated control lighting system, so for that it is required an algorithm that computes each luminaire's reference value based on the reference of all luminaires, trying to optimize a cost function described below. The method used was the Consensus Algorithm described in [4]. Together with this coordinated control, each luminaire has also the local controller described in section 3.6 to reject small local disturbances and some imperfections in the model used to compute the reference levels.

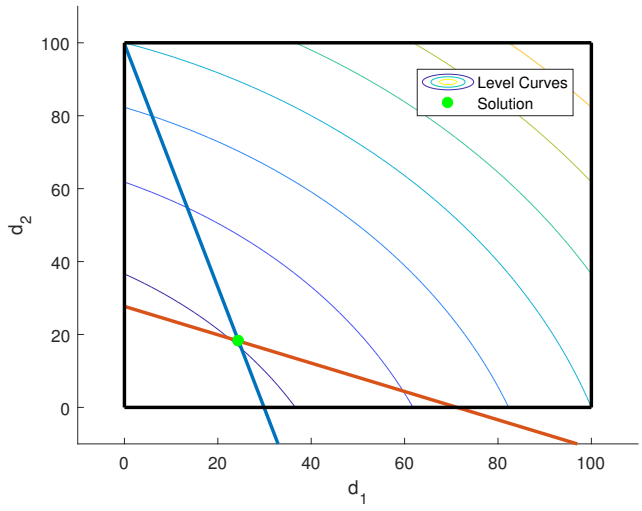


Figure 11: Optimization problem's constraints (black, blue and red lines) and cost function's level curves representation. (Two luminaires example)

This solution is useful in real systems to input some restrictions to the system, for example, optimizing the energy consumed by the luminaires or minimizing the number of times that a LED is turned ON, increasing its lifetime and reducing monetary costs of the overall system.

3.7.1 Optimization Problem

The optimization problem addressed can be formalized for the case of two luminaires as

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \frac{1}{2} d^T Q d + c^T d \\ \text{subject to} \quad & k_{11} d_1 + k_{12} d_2 + o_1 \geq L_1, \\ & k_{21} d_1 + k_{22} d_2 + o_2 \geq L_2 \\ & 0 \leq d \leq 100 \end{aligned} \tag{12}$$

where c is a vector of the energy cost per unit of duty cycle in each luminaire and Q is a diagonal matrix where each diagonal entry represents the cost related to the wear of each luminaire per unit of duty cycle.

Note that the value of the LED's PWM are used in percentage ($0 \leq d \leq 100$) for simplicity in the computations, this value is converted to the 0 to 255 scale simply by multiplying the result by $\frac{255}{100} = 2.55$, before using it in the controller. The same reasoning was applied in section 3.3, where in the calibration process the values of k_{ij} are obtained considering this scale.

The constraints of this problem are the limitations of the system discussed in section 3.3, considering that the lux measured has to be greater than or equal to an intended reference L in each luminaire. The feasible region, for the case of two luminaires, and this example's solution are depicted in Figure 11.

3.7.2 Consensus Algorithm

The optimization problem in (12) is separable, meaning the computation can be distributed in each node (luminaire), only some information needs to be exchanged in each iteration due to the constraints. Each node only has a portion of the global problem and only needs to know the values of the d variables computed by the neighbour.

Then the algorithm uses the Alternated Direction Method of Multipliers (ADMM) as described in detail in [4]. This method finds the six possible solutions: the unconstrained solution and five solutions in the boundaries of the feasible region (black square and red and blue lines in Figure 11). Then it checks which solution leads to the smaller value of the cost function and saves that solution. After the two nodes finish their computations, they exchange the solution of that iteration and start a new iteration with the solution received from the other node. The process is repeated in 50 iterations to guarantee that the method converges, but it was noted that the method converges on average after no more than 30 iterations. A stopping criteria could be implemented to avoid redundant iterations, for example if the difference between iteration's solutions were below a certain threshold.

When the algorithm converges, the values for the duty cycles from each node to one another are equal, i.e., both nodes reach a consensus on the value of d each one must set to fulfill the constraints. Then these values are set as the respective references of the local PI controllers.

The algorithm is run every time that one node changes its reference. This could be done for every disturbance but it would introduce unnecessary delays and computation in the system, so the local controllers are responsible to deal with local disturbances as said before.

The response of the system with and without the coordinated control, for both luminaires, is shown in Figure 13, section 5.1, as a result of an experiment described in section 4.1.

3.8 Arduino-to-Arduino Communications

The communication protocol used in the Arduino's network was I2C, using the default clock frequency of 100 kHz. In order to test the average time needed to send and receive a message in this protocol, it was sent a request message with one byte. On average the delay of the request and the response was 236μs.

As for the I2C addresses the method used was to create a piece of code to write each address in a position of the EEPROM memory of the Arduino, which saves the values even if the Arduino is turned OFF. For that it was used Arduino's function *EEPROM.write* from EEPROM library. In the beginning of the program each Arduino starts by reading the position of the EEPROM memory with function *EEPROM.read* setting its own address and computing the other Arduino's address with simple logic.

This protocol is used to send information between Arduinos that can be read by the RPI, as it will be explained later in section 3.12, as well as commands from the Arduino which is connected to the RPI to the other Arduino.

3.9 Arduino-to-RPI Communications

In this network the Raspberry Pi represents a node which controls the other nodes, the Arduinos. The RPI consists on a cloud which stores the relevant information about the functioning of the program which can be accessed by several clients, which can be seen in the overall scheme of the project in Figure 12. The RPI can also send commands to each individual Arduino to change some of its parameters. This cloud listens to the I2C bus through a sniffer program (described in section 3.12), this sniffer lets the RPI listen to what messages the Arduinos are changing between each other.

To send the client's commands to any of the Arduinos in the network, the RPI is connected by its serial port (USB port) with one of the Arduinos'. When a client wants to change a functionality of an Arduino, the RPI sends a message through the serial port to the connected Arduino. Although the connected Arduino might not be the recipient of said message, the RPI message has the address of the receiver. This message is passed by the I2C bus in a broadcast in order to reach its goal. It is important to note that the *baudrate* of the serial connection was set to 115200 (bits/cycle). This value was chosen to be this high in order to assure the quickness in sending messages to avoid any unwanted delays in the Arduino Program while sending data through the serial port.

3.10 RPI-Sockets Communications

The implemented TCP server is asynchronous, meaning the program does not block when a call to a usually blocking function is made, the program simply verifies if there exists anything to read and if not, continues. Using this we can use just one thread to run the TCP server, where it checks several clients if a message is received. All the used class and objects of the TCP server were obtained using the Boost Library [5].

To successfully implement this, one needs to assign a port and a descriptor to a TCP socket, this socket will receive any incoming connections from different clients. When this socket receives a new call a new file descriptor (or socket) is returned and it will identify the caller and its connection, any received messages or sent will be through that file descriptor. This way a TCP server can attend multiple clients at the same time.

In our program the TCP class is one of more fundamental ones since it receives the incoming calls and fetches all the data requested by the client. All the other classes used to keep the data and to process it are explained later in section 3.12.

3.11 Arduino Software Architecture

For the software architecture of the Arduino, a function based program was used. The only Interruption Service Routine (ISR) used was for the I2C protocol with the *Wire* library [6], that called a function (*receiveEvent*) when an I2C message is received.

This function creates a buffer to write all the bytes received and then parses the data to retrieve the values sent from the other Arduino. The values sent by the Arduinos are separated by semicolons, so on receive the function searches for all the semicolons and converts each value to a float that can be used in the computations. The values read need to be char type, due to the limitation on the Arduino UNO's I2C protocol which does not support floats to be sent.

All the other features of the program are implemented through functions that are called in the main loop whenever needed, like the function to read the serial commands,

which uses the same reasoning and structure as the function *receiveEvent* from the I2C protocol.

The main loop also includes all the computations for the illuminance control loop described in section 3.6. By adding the functions described above, the sample time needed to be increased. A value of 15 ms was chosen instead of 3 ms as before. This was caused by the delays introduced by the function calls. To refine the controller after this change one would need to retune the control parameters (K_p and K_i). However, the old parameters' values maintained the performance level, so retuning was not required.

3.12 RPI Software Architecture

The Raspberry Pi Architecture is separated in three main classes such as Arduino, circular buffer and server. Nonetheless, it also has file functions that do not belong to any of these classes but are used in the overall program: *serial_comm* and *pig2i2c*.

To begin with, the file *serial_comm* deals with the serial communication between the RPI and the connected Arduino, it sets the used baud rate (115200 bits/s) and opens the corresponding serial port to send messages. Duly note that, as previously said, the serial connection between RPI - Arduino is only used to send unidirectional messages from the Pi to the Arduino and it handles the client's request, for example, to switch a desk's occupancy state or reset the whole program. Afterwards, the pig2i2c function ("Sniffer" function) is utilized in order to "listen" to messages between Arduinos. This function "spies" on the I2C bus and keeps the data that is important since in the end of every duty cycle a message is sent from one Arduino to the next, informing its relevant information.

To store the information from the Sniffer, the class Arduino is used. This class is the most important of the RPI Software, since it stores all the relevant information and does all the needed computation to save every value that could be requested by the client. Apart from all the simple computations when reading a new set of values or the usual *setters* and *getters*, this class has a queue (First In First Out - FIFO) where the sniffer function pushes the newly caught data and while the queue is not empty the class Arduino pulls information from it and processes all the data. This can be useful for the real time stream, since all the Arduino needs to do is pop from the queue and send it to the client through the TCP server. It is important to note that for this to be possible without any conflicts in the program (data races or deadlocks), *Mutex* lock/unlock functions are needed when pushing and pulling from the queue.

Some client requests need special types of structures such as a circular buffer. This class is implemented in the Arduino class and it is used to store all of the lux and duty values from the last minute. The size of the circular buffer is set according with the sample time of the program as to be big enough to store all values from the last minute. When the capacity of this buffer is reached the program starts writing over the oldest values and, as soon as the client requests it, the buffer is all sent with the corresponding values, organized from new to oldest.

As discussed before in section 3.10, the server class creates the objects needed to implement a TCP server and to receive several clients. This class simply receives the client's request and gets the required information, sending it back to the desired user.

Overall, the designed program works on four threads: one to run the TCP server, to receive incoming calls and to send information; one for each Arduino, to process received data and to do all the computations needed; and lastly another one for the sniffer, so the

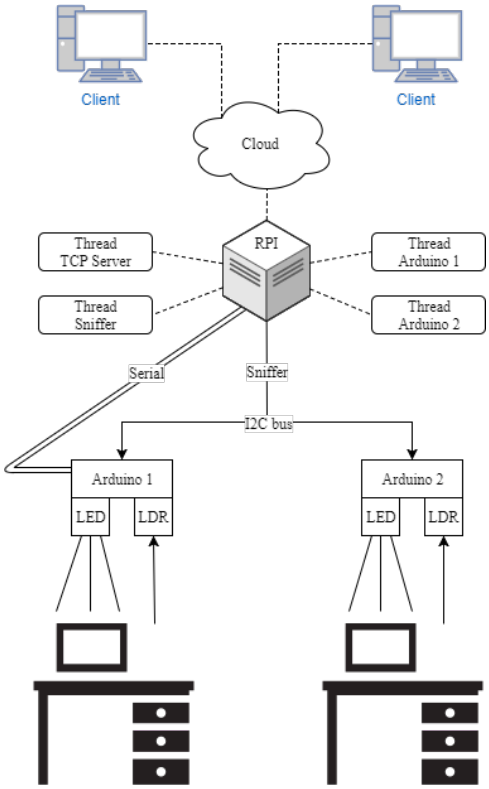


Figure 12: Overall scheme of the project

information can be processed faster. The program is scalable as it can have several clients connected and several objects of the Arduino class can be instantiated when dealing with more than two Arduinos. As for the memory usage by the program, one can conclude that it is constant since everything is allocated and acessed in boundaries. The circular buffer has a maximum size, writing over the oldest value when a new one is read. The only case of memory leaking would be having a full queue but this never happens since when one thread is pushing the other one is waiting to pop.

4 Experiments

4.1 Coordinated control response

In order to test the system response with the coordinated control, an experiment with the following instructions was designed: luminaire 1 reference level is increased to the higher value (40 lux) and decreased to the lower reference value (15 lux) while luminaire 2 is at the lower reference value and at the higher reference value. This was done with and without the coordinated control being active in the system and the results can be seen and are discussed in section 5.1.

4.2 Coordinated control impact

Beyond just the system response, there is a need to test the impact the coordinated control has in the overall values of energy consumed by the system, the comfort error the users experience and the comfort variance in the values of illuminance.

The total energy consumed was calculated as

$$E_{total_i} = \sum_{n=1}^{N} P_{total_i}(n) \times T_s, \tag{13}$$

with n representing one sample time (the sum of all the sample times occurred gives the total time passed), T_s being the sample time and $P_{total_i}(n)$ being the total instantaneous power being consumed at desk i in that same period of time. This power was calculated by multiplying the duty cycle of each luminaire by 1 W, which was the assumed maximum power of the luminaires, and adding the results.

The total comfort error at each desk was defined as the continuous sum of error between the reference illuminance at desk i (I_{ref_i}) and the measured illuminance at desk i (I_{meas_i}), for the periods when the measured illuminance is below reference, and then adding all the desks,

$$C_{error_i} = \sum_{n=1}^{N} max\left\{I_{ref_i}(n) - I_{meas_i}(n), 0\right\}. \tag{14}$$

As for the total comfort variance at each desk, it was defined as the continuous sum of sudden variations in the illuminance (flicker). This was approximated to the second derivative of the illuminance,

$$V_{flicker_i} = \frac{\sum_{n=3}^{N} \left|I_{meas_i}(n) - 2I_{meas_i}(n-1) + I_{meas_i}(n-2)\right|}{T_s^2}. \tag{15}$$

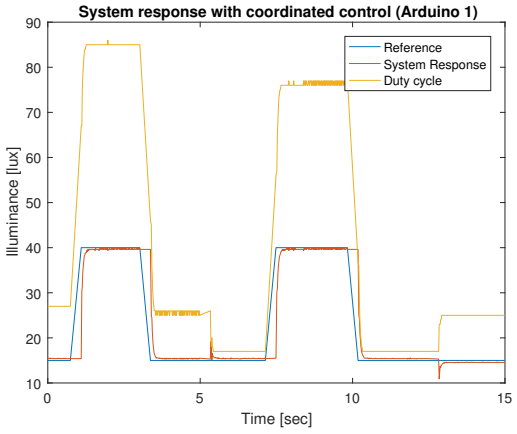
This equations compute the metrics of each luminaire, to get the metrics for the whole system, one only needs to sum the metrics of the individual nodes.

A test was designed to see the impact the coordinated control had on all three values: starting with both luminaires at the lower reference, the reference level of luminaire 1 was increased 10 seconds after the start of the test; 50 seconds later the values were taken using client requests to the server. The test was the same for all the values and the results can be seen in section 5.2, as well as a result discussion.

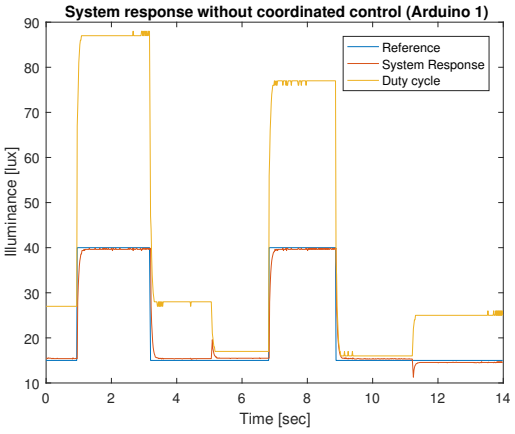
4.3 System Functionalities

The final test, to show some of the functionalities of the completed system, can be seen in the video [7]. It shows the server retrieving reference and illuminance values right after start-up and after a reference change in desk 2 and the energy consumed (in each desk and the total) since start-up. Then the system is restarted, the reference and duty cycles in each desk are retrieved and finally the time since restart is tested, before and after a second restart command.

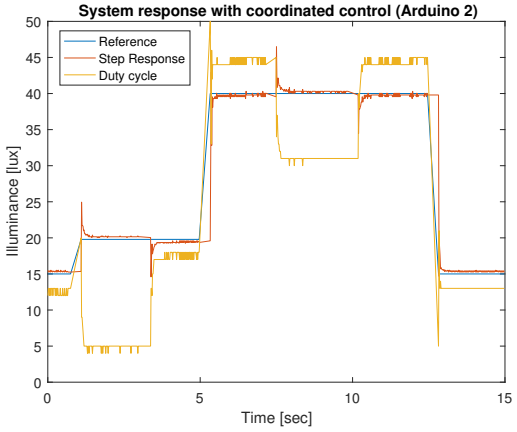
The video [8] shows the calibration process and a small demonstration of the system reacting to changes in external illuminance.



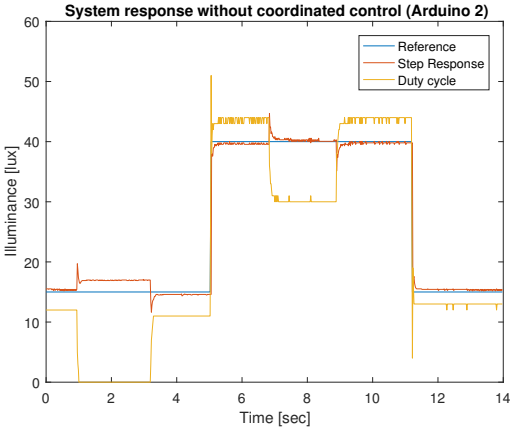
(a) Coordinated control luminaire 1



(b) Uncoordinated control luminaire 1



(c) Coordinated control luminaire 2



(d) Uncoordinated control luminaire 2

Figure 13: System's step response with decentralized coordinated and uncoordinated control

Table 1: Energy, Comfort and Variance levels after 1 minute of test, with and without Coordinated Control

	Energy [J]	Comfort [lux]	Variance [μlux/s^2]
CC_{off}	37.509	568.709	2.111
CC_{on}	37.100	539.109	1.756

5 Results

5.1 Coordinated control response

In this experiment, one can note the coupling of the luminaires, every reference change in one luminaire causes an overshoot of illuminance in the other. With the coordinated controller, the system will try to use this coupling to optimize the lighting control of the system.

For the case of the coordinated control (Figures 13a and 13c) one can see that when luminaire 1 changes its reference to 40 lux (1 second after the beginning of the test), luminaire 2 also changes its reference to around 24 lux, since this was the Consensus solution of this case. However, when the reference of luminaire 1 returned to 15 lux, the reference of luminaire 2 did not. This is due to the values used in the formulation of the optimization problem, it penalizes the variations in the duty cycle in a quadratic way, while penalizing the power consumption in a linear way, as in (12). This behavior can be adapted by changing the weights of the cost function (c and Q).

The uncoordinated controllers (Figures 13b and 13d) do not exchange information between them, so every reference change in one luminaire is dealt as an external disturbance in the other. This can be seen in Figure 13d between second 1 and 3, as in the coordinated case, luminaire 1 changed its reference to 40 lux. In this case, luminaire 2 was not able to reduce the light to 15 lux (its reference), since the measured illuminance is close to 17 lux due to the illuminance produced by the other luminaire, even with the LED of luminaire 2 turned OFF. So the controller cannot satisfy the reference and this creates an error that could have been perceived and nullified if the coordinated control had been used.

By comparing Figures 13a and 13b, one can see a slight increase in the duty cycle when the coordinated control is not active and the reference is increased. This shows that the Consensus algorithm is trying to reduced the wear in luminaire 1 by increasing the reference in luminaire 2, luminaire 1 can have a lower duty cycle.

5.2 Coordinated control impact

As can be seen by analyzing Tabel 1, the coordinated control does not yield very different results from the uncoordinated control, although it is better in all studied aspects.

The energy consumed shows a 1.1% drop which seems small at first, but it's not negligible from a large scale point of view. This is a system with only two luminaires and the test was run for only one minute, in a system with a lot more luminaires running for a whole year the difference would be considerable, making the coordinated control a better option than uncoordinated.

The comfort error also shows a slight drop of about 5.2%, meaning that the coordinated control has more impact minimizing the comfort error. Despite that, a 5.2%

difference in the illuminance level is too small a difference to be noticeable by the human eye. This result could be improved by decreasing the Dead-Zone of the local controller to decrease the steady state error.

Finally, the variance shows a more significant decrease, about 16.8%, which may indicate that the coordinated control is smoother in its transitions. However, this test should have had a lot more step changes than only one, in order for this result to be more reliable.

6 Conclusion

The main goal of this project was to develop and implement a lighting system consisting of LED luminaires, able to locally sense and control the illuminance at a desk, while globally optimizing the energy consumption and guaranteeing sufficient comfort for the users, as well as being able to take commands from an external entity.

Both the implementations of the local controllers and of the global control showed to be working adequately during demonstrations. The impact of the Feedforward and the Feedback are highly noticeable when assessing the overall quality of the local controllers.

The coordinated control is showing to be working correctly, taking into account the wear of the luminaires, as disussed in section 5.1. It also showed superior results from the uncoordinated control, in terms of energy consumption, comfort and variance. The differences are small (the energy consumed dropped by 1% when using the Consensus algorithm) but the project's system was also very small, with only two luminaires and the tests running for only 1 minute. The variance could have been tested differently from the other two metrics, with more step changes, in order to produce a more reliable result.

Unfortunately, the command for the real time stream of illuminance and duty cycle values at each desk asked for in the project description was not implemented. This was mainly due to time constraints of the project. However, the developed C++ server ran properly, i.e., was able to properly receive and store the information sent by the Arduinos, and responded adequately to all other client requests and commands.

In this project, the Consensus algorithm was used for its fairly adequate results and relatively simple implementation, but perhaps a different coordination algorithm could be used depending on the needs of the system to be implemented.

Maybe in a system with different characteristics a more complex cost function could be used. In this project only the wear of the luminaires and the energy consumed were taken into account but there is capability of adding more variables, which increases the complexity of the algorithm.

For example, in a system where the energy consumption is more problematic, different weights or expression for the energy in the cost function can be applied. The same can be applicable for the wear and other characteristics of the luminaires that one might use.

In order to improve this system one could create a graphic interface. This would enable a more friendly user interface, an advantage if the system was to be commercialized.

Given how well this project's system can be expanded, one can extrapolate this to a wider network of luminaires. This model can be implemented in an industrial setup, office lights, public lights, and others. By saving power consumption and reducing the wear in the light bulbs one saves money but also the environment because less power and less light bulbs end up being produced. This model could even be used in a completely different setup, like agriculture, the model would only have to switch from desks to be

illuminated to plants to be specifically lit.

References

[1] A. Bernardino, *Distributed Lighting Control*, Project guide.

[2] *GL5528*, CdS Photoconductive Cells. [Online]. Available: https://pi.gate.ac.uk/pages/airpi-files/PD0001.pdf

[3] T. Wescott, "Pid without a phd," Jan. 2000. [Online]. Available: http://www.wescottdesign.com/articles/pid/pidWithoutAPhd.pdf

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, Jan. 2011.

[5] *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] Wire library. Last accessed: 05/01/18. [Online]. Available: https://www.arduino.cc/en/Reference/Wire

[7] Project's server demonstrations. Accessed: 05/01/18. [Online]. Available: https://youtu.be/YitZQARzt44

[8] Project's overall performance demonstration. Accessed: 05/01/18. [Online]. Available: https://youtu.be/lx_5Uw84AJI