Mestrado em Engenharia Electrotécnica
e de Computadores

Sistemas de Controlo Distribuído
em Tempo Real

Project Report

Distributed Lighting Control

Group 1

Gonçalo Ribeiro
73294

João Almeida
73198

Prof. José Gaspar
Prof. Alexandre Bernardino

January 4, 2016

Contents

Abstract

We present our solution to the Distributed Lighting Problem. The goal is to control in a distributed manner a set of lights keeping them at a desired level while minimizing the energy spent and maximing confort to the people using the light. This solution is tested and implemented in a physical setup consisting of a wooden box with 3 Arduino Uno Boards controlling each one white LED and reading the values of illuminance with the help of an LDR.

In each Arduino a Local Controller keeps the illuminance at a desired level. To interact with the system a client connects to a TCP/IP server that transmits the commands to the Arduino and sends statistics about the system operation in the other direction.

The problem is represented in a Linear programm form to allow us to apply the Simplex algorithm. The results of this are then fed to the local controllers to speed up their response.

With our experiments we show that our approach is reasonable and achieves good results, however the Simplex algorithm seems not to improve system performance.

1 Introduction

A real efficient lighting solution must be able to accomodate different lighting requirements in different areas, must take into account external lighting in each area and must be able to understand the light interferences between different areas.

The Distributed Lighting Problem consists of finding the optimal way to control a set of illuminaries to keep the illuminance at a desired value. These illuminaires may correspond to different desks or areas which migh have different lighting requirements. The system must present optimal behaviour not only in terms of energy efficiency but also in terms of response speed to changes and confort to the final user. On top of all this a distributed solution when compared to a centralized one presents many benefits. This is the problem we prupose to solve.

There has already been research on this problem [5], [6], [8]. Our approach tries to define the problem as a linear programm that would then be solved with the Simplex Algorithm. In [4] a distributed version of this algorithm is explored. In our solution we don't use this version of the simplex algorithm but this might be an interesting improvement.

We created a physical setup to simulate an office environment and test our approach to this problem. It consists of three illuminaires inside a box which could have external lighting. Each illuminaire is made of a LED and a small circuit with an LDR to measure the illuminance, the illuminaire is controlled by an Arduino UNO board. The arduinos communicate between themselves and with a server, this serves as the interface to clients, allowing them to analyze and change the state of the system.

All our code is available publicly on `https://github.com/goncalor/SCDTR`.

2 Approach

In the next section we describe our solution for the distributed lighting problem. The main components of our system are the Local Controller, used to maintain the illuminance at each desk at the desired reference, the Simplex Algorithm, that calculates the optimal solution to the linear programm that describes this problem, and the TCP/IP Server, which serves as an intermediary between the clients and the system. An overview of the system is depicted on Figure 1.

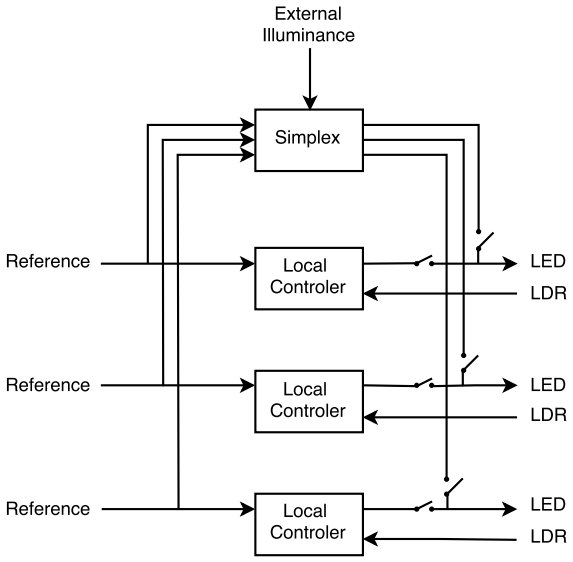


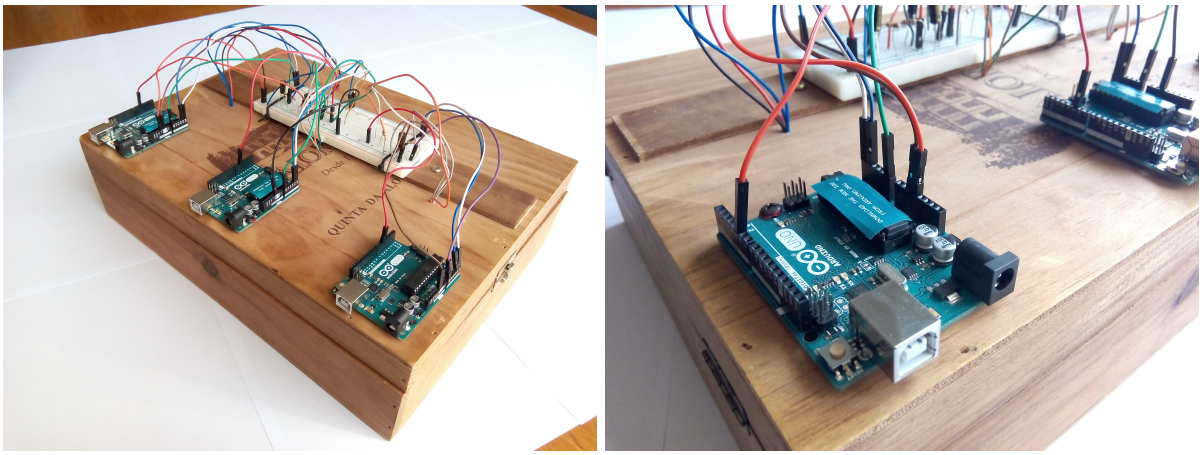
Figure 1: Block diagram of the implemented system

2.1 Physical Setup

The setup must model a room with luminaires on its ceiling. Each luminaire has a controller that keeps the light below it above some standard level of illuminance. For this project it was pre-established that the room contained three luminaires.

For the physical setup a wooden box was used. This was preferred to a cardboard box because of the increased sturdiness. Since the controllers (Arduinos) are too big to be included inside the luminaires in our model they were put on the outside of the box, and all the wiring is done outside the box using a breadboard (see Figure 2a). Both the breadboard and the Arduinos are screwed to the box preventing them to move, which could easily disconnect the jumpers. Tight holes were made to pass the wires needed to light the LED and read the sensor, while

preventing light from entering the box (see Figure 2b).



(a) Arduinos and breadboard on top of the box

(b) Detail showing the connections on the Arduino and one hole in the box

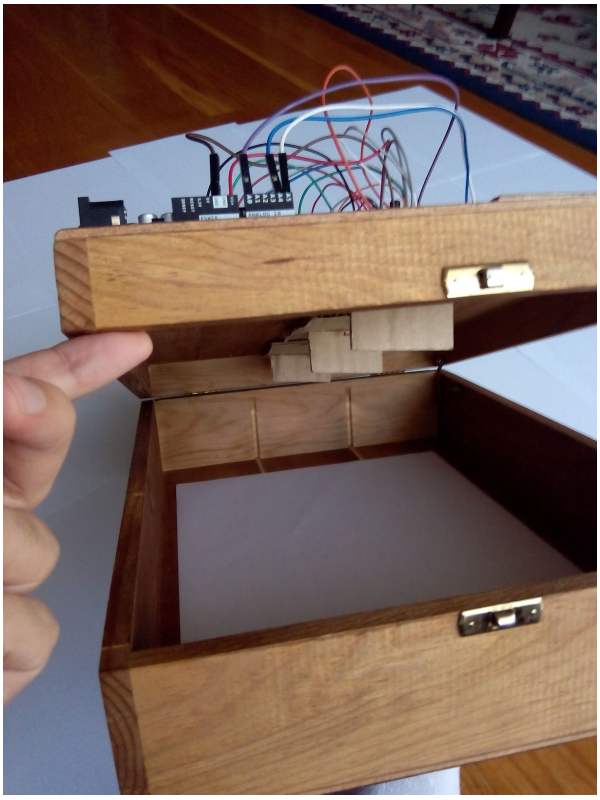
Figure 2: The outside of the box

On the inside of the box only the luminaires are present. Each luminaire is assumed to include both a LED light and a light sensor (LDR). The LED and the LDR are separated by a piece of cardboard so that the light from the LED does not directly affect the LDR reading (Figure 3a). Only reflected light should be read. In order to increase the light reflection inside the box — so that the LDR senses more light — a white sheet of paper was placed on the bottom of the box (see Figure 3b).

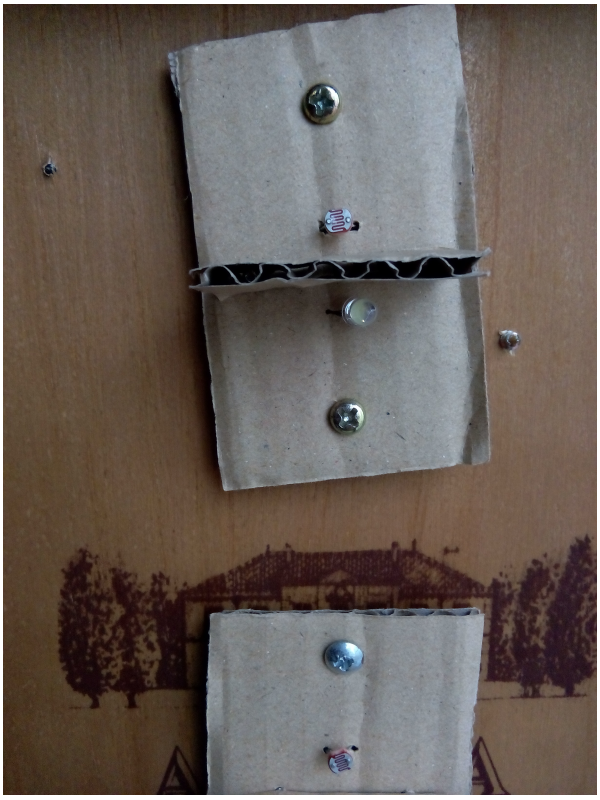
The assignment requires that the model has a window. The used box has hinges and the top of the box can be opened (see Figure 3b). Therefore this was used as an window. The top should only be opened in small angles so that we can consider (by approximation) that the LDR is still facing the ground and that the light of the LED is mostly reflected back at the LDR.

The emitter-detector circuit for each luminaire is shown in Figure 4. The emitter is made of an LED in series with a current limiting resistor (Figure 4a). The typical forward voltage of the used LED is 3.0 V and its maximum forward current is 20 mA. Since the Arduino is powered with 5 V the voltage on pin `D5` is at most 5 V. Therefore, using a 100 Ω resistor makes the LED be at its maximum power when the duty cycle on pin `D5` is 1. For the light detector circuit a voltage divider is used, consisting of a normal resistor and a photoresistor. A capacitor is also included in parallel with the resistor to act as a low-pass filter to eliminate noise from the readings. An analysis of the characteristic of the LDR is made in 2.2.2.

A I^2C/TWI bus is used to implement the communications between the controllers of each luminaire. The physical implementation of this bus is attained by connecting the `A4` pin of all



(a) Luminaires on the ceiling and white sheet on the floor



(b) Detail of a luminaire. The LDR on top, LED on bottom and cardboard in between

Figure 3: The inside of the box

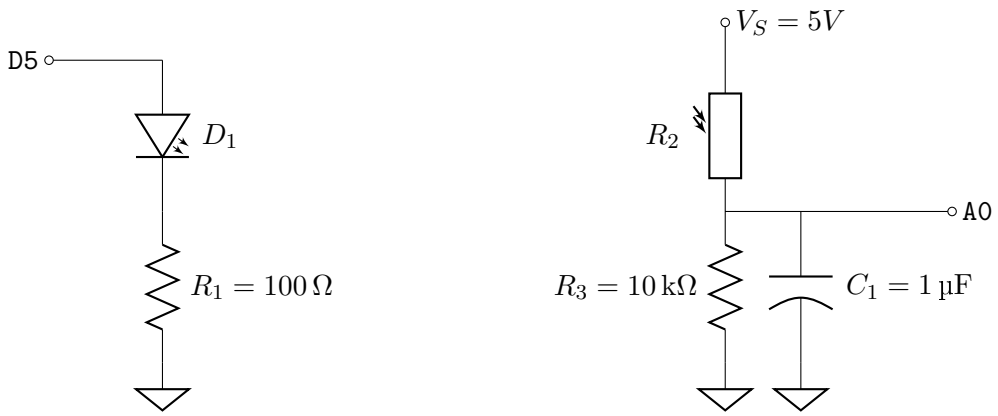
the Arduinos as well as the `A5` pins — SDA and SCL respectively.

2.2 System Characteristics

2.2.1 Steady State

The steady state response of the system was determined by setting the duty cycle of the LED to all values in the PWM range 0–255. Each sample was acquired 200 ms after the previous one so that the system had time to reach a steady state. This is an adequate time between samples because RC constant of the detector circuit in Figure 4b is $\tau = R_2C_1 = 60\,\text{ms}$ and 400 ms amounts to more than 3τ. The reason why R_2 is in the time constant and not R_3 is because the PWM value was incremented and hence C_1 charges via R_2. If the data was acquired while decrementing the PWM value R_3 would be used instead.

A graph of the acquired data can be seen in Figure 5. We conclude that the resistance of



(a) The LED circuit. Pin `D5` controls the power on the LED using PWM.

(b) The light detector. An LDR is used and readings are done on pin `A0`.

Figure 4: Emitter-detector circuitry of each luminaire

the LDR does not vary linearly with the duty cycle on the LED. Nonetheless it is possible to obtain a linear function relating the input an output of the system, as described in 2.2.2.

2.2.2 Modeling the LDR

Since the steady state plot in Figure 5 reveals that the system is nonlinear it is useful to understand why. The main sources from nonlinearity should result from the LED, the LDR or a combination of both.

With that in mind the LDR datasheet was analysed. On it the graph in Figure 6a can be found. This graph gives a range of values for the resistance of the LDR for each value of incident illuminance. To produce an equation that models the LDR resistance as a function of illuminance the resistance value used for the model is the mean of the maximum and minimum values given for each illuminance. The final result should be a function which produces a line that crosses the middle of the dark region in Figure 6a. Note the axis of the figure are both in logarithmic scale. The considered points to obtain this linear function (in a log-log reference) were $(L_1, R_1) = (100\,\text{lx}; 2.5\,\text{k}\Omega)$ and $(L_2, R_2) = (1\,\text{lx}; 60\,\text{k}\Omega)$. The model will be an equation in the form

$$\log_{10} R_{LDR} = a \log_{10} L + b. \tag{1}$$

We can substitute (L_1, R_1) and (L_2, R_2) in (1) to obtain the system of equations

$$\begin{bmatrix} \log_{10} R_1 \\ \log_{10} R_2 \end{bmatrix} = \begin{bmatrix} \log_{10} L_1 & 1 \\ \log_{10} L_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}. \tag{2}$$

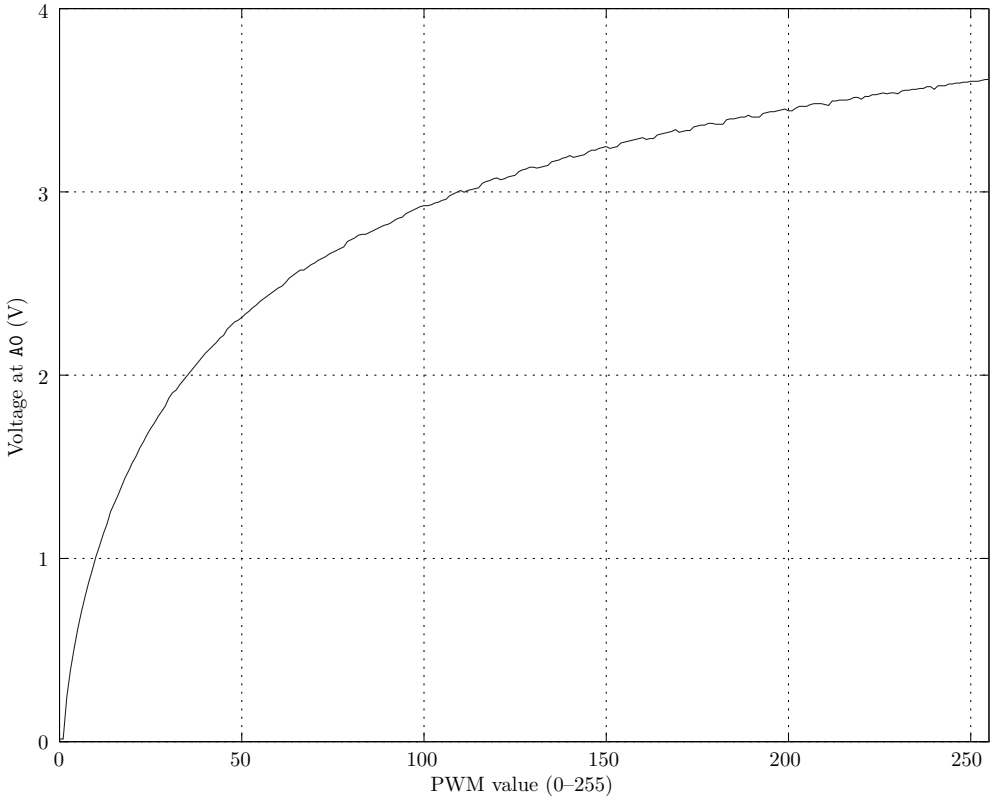


Figure 5: Steady state voltage at pin A0 as a function of the duty cycle of the LED

By solving the system the values $a \approx -0.6901$ and $b \approx 4.7782$ are obtained for this case.

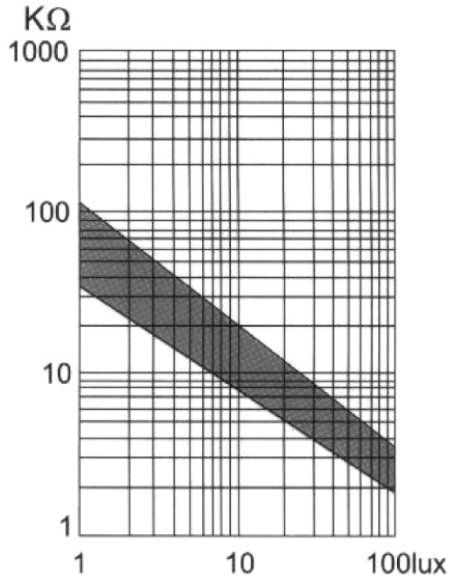
With the values for a and b and (1) it is possible to plot the graph in Figure 6b which models the LDR characteristic, as desired. Using (1) it is possible to obtain the incident illuminance (L) from the resistance of the LDR R_{LDR}

$$L = 10^{-\frac{b}{a}} R_{LDR}^{\frac{1}{a}}. \tag{3}$$

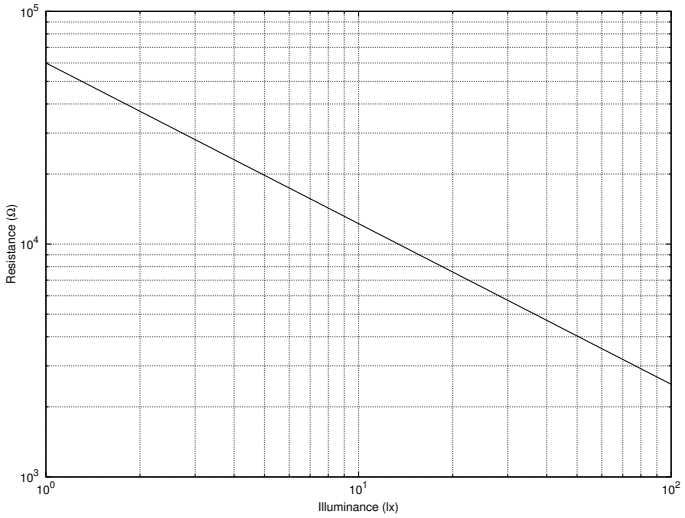
And since in a steady state

$$R_{LDR} = \frac{V_S R_3}{V_{\texttt{A0}}} - R_3, \tag{4}$$

we can substitute (4) in (3) and apply the new equation to the plot in Figure 5 to produce the plot in Figure 7 which relates the duty cycle of the LED to the illuminance measured by the LDR. We can conclude that the illuminance varies linearly with the duty cycle applied to the



(a) LDR resistance as a function of illuminance according to the GL5528's datasheet



(b) Model for the resistance of the LDR as function of the incident illuminance

Figure 6: LDR characteristic

LED.

2.2.3 Step Response

To plot the step response of the system an Arduino program was written that to produce a step and acquire data to be ploted. In this script the LED is first turned off for 1 s so that the capacitor "fully" discharges. Then a step is applied to the system by setting the LED duty cycle to 0.5. A sample is acquired and sent to the computer approximately every 1.35 ms. The readings are converted to Volt and the result is Figure 8a.

An inflection point can be perceived early in the step response. From this follows that the system's order must be at least second degree. The detector in Figure 4b includes a capacitor, which must introduce a pole. Therefore an experiment was made which consisted simply in removing the capacitor and acquiring the step response again. The result is shown in Figure 8c where it can be seen that the aforementioned inflection does not exist. It is also noticable that introducing the capacitor does filter high frequency noise, as expected. Therefore introducing the capacitor improves the readings but makes the system order increase. Nonethess the capacitor value can be changed allowing to control the pole position and cutoff frequency.

Assuming that the system can be approximated to a first order system of the form $G(s) =$

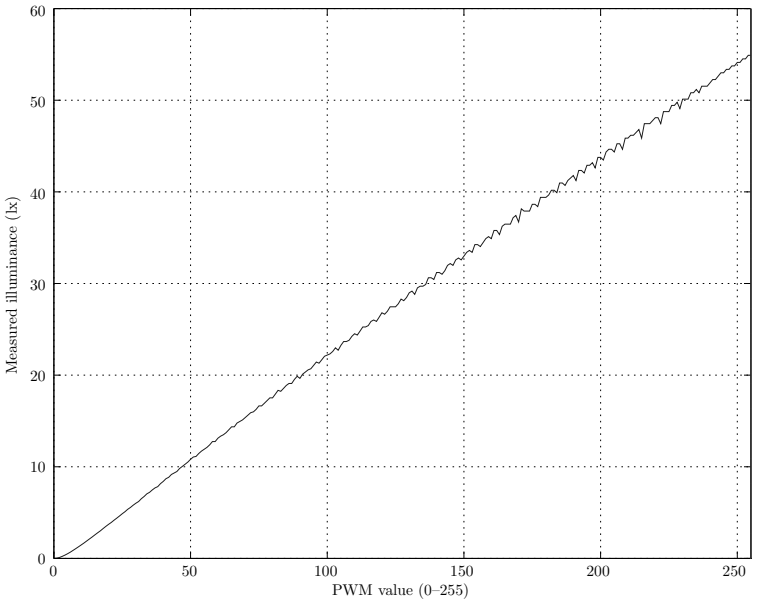


Figure 7: Illuminance detected by the LDR in function of the duty cycle of the LED

$K_0/(1+s\tau)$ the system can be modelled by finding the static gain K_0 and the time constant τ. K_0 is calculated as the ratio of the output and the input under steady state; the time constant is the time the system takes to reach $1 - e^{-1} \approx 63\%$ of its final value. Therefore for Figure 8a the static gain is $3.08/2.5 = 1.232$ and the time constant is approximately 23 ms.

If the system is linearised by using the transformation presented in 2.2.2 Figure 8b is obtained. In this case the static gain is $26.4/2.5 = 10.6$ and the time constant is approximately 47 ms.

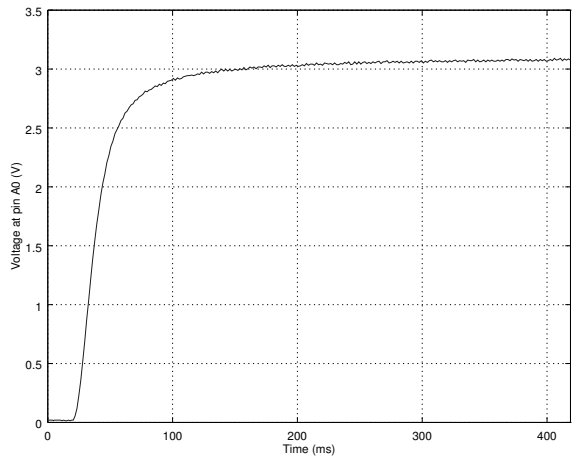
2.2.4 Incremental Response

An incremental step response was also produced (Figure 9). For this, incremental steps of 10 PWM units (3.92% duty cycle) were used every 400 ms. A sample was acquired approximately every 1.35 ms.

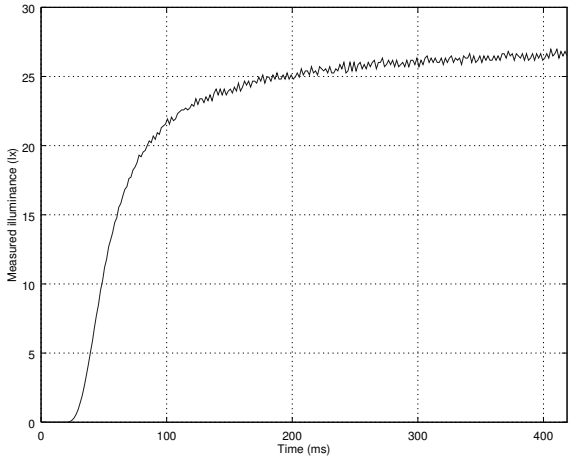
The acquired data is noisy and therefore it is difficult to measure the time constants for most of the steps.

2.3 Local Controller

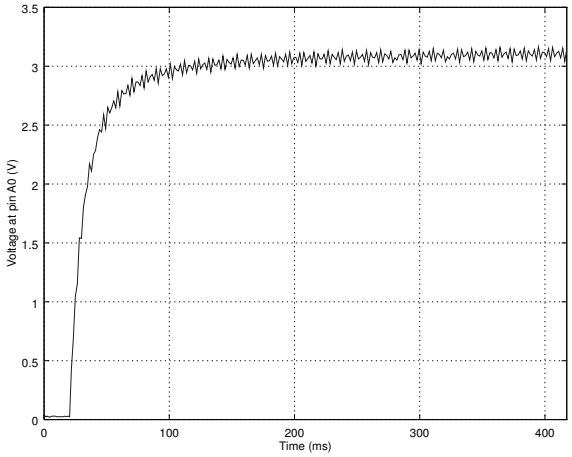
The function of the Local controller is to maintain the illuminance at each *desk* as close to the desired state as possible. This controller doesn't have any information about the global



(a) With capacitor, not linearised



(b) With capacitor, linearised



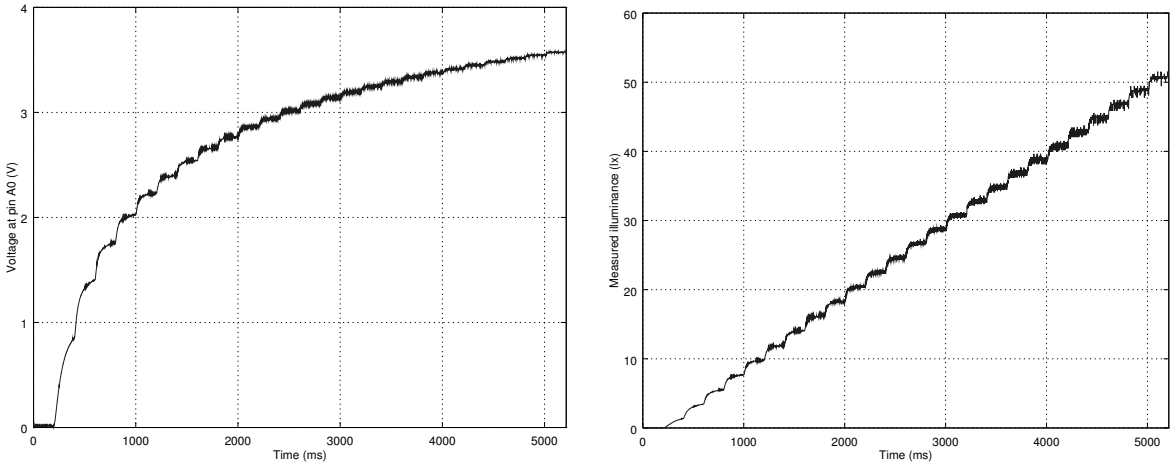
(c) No capacitor, not linearised

Figure 8: Response of the system to a step with 50% duty cycle applied at time 20 ms

system and only uses local information to operate.

The local controller consists of a Proportional-Integral-Derivative Controller (PID) that receives as input a integer reference value (r) from 0 to 1023, the LDR readings from 0 to 1023 (y) and has as output the PWM duty cycle (u) in the range 0 to 255.

The block diagram of the local controller is on Figure 10. The error (e) is defined as $e = r - y$. The signal u is calculated as the sum of the proportional, integral and derivative components and the feedforward term.



(a) Not linearised (b) Linearised

Figure 9: Incremental step responses with increments of 3.92% duty cycle. First step is at 20 ms.

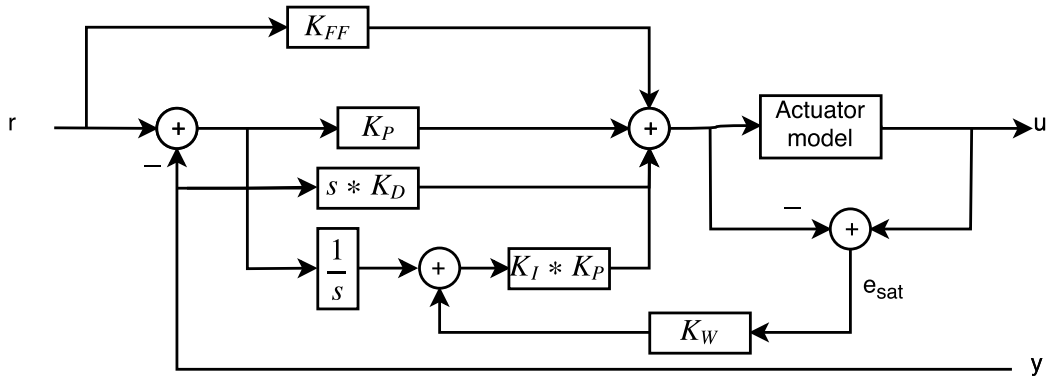


Figure 10: Local controller block diagram

2.3.1 Proportional Component

The proportional component (P) of the controller is just a gain multiplied (K_p) by e. $P = K_p * e$.

2.3.2 Integral Component

To approximate the integral of the error (I) at sample k we use the formula $I_k = I_{k-1} + \frac{e_k - e_{k-1}}{2} * T_s$. We need to make this approximation as we don't have the continuous time values for the error.

2.3.3 Derivative Component

The derivative component (D) is calculated as $D = -K_d/T_s * (y_k - y_{k-1})$. This component is calculated with signal y instead of u to avoid differentiating discontinuities, which would cause the signal u to saturate.

2.3.4 Anti-Windup

The output of the integral component of the controller can be outside the range of values acceptable by the Arduino output. This would lead the integrator to accumulate an error that would only disappear with the error being negative for multiple samples. This causes a poor performance on the system. To counter this effect we implemented a Anti-Windup block. This block uses the difference between the expected controller output and the constrained value multiplied by a gain (K_W) to clean the integrator.

2.3.5 Feedforward

The feedforward term(F) depends only on the reference and is used to speed up the controller. It is calculated as $F = r * K_{FF}$, where K_{FF} is the feedforward gain.

2.3.6 Implementation Details

The readings from the sensor are done with the average of three samples.

This controller was implemented as a interrupt routine to guarantee its execution in fixed intervals.

Todos os cálculos do controlador são feitos com valores de 0 a 1023, sendo convertidos quando necessário: para 0 a 255 para actuar o LED; e para lux quando são pedidos valores de iluminância.

2.3.7 Parameters

The gains of the controller were tuned by hand. Following the guidelines from *PID Without a PhD* [9] we set the coarse values for each of them. Then we a set of tests to both the step response and the incremental response we fined tuned the parameters. The sampling time was

chosen experimentally as the lowest value that allowed the rest of the code in the Arduino to exectude between each controller update.

The final values we use are:

Proportional Gain $K_P = 10$;

Derivative Gain $K_D = 0.01$;

Integral Gain $K_I = 10$;

Feedforward Gain $K_F F = 0.1$;

Anti-Windup Gain $K_W = 0.05$;

Sampling Time $T_s = 1500\mu s$;

2.4 Server

The Server in this system has 3 main goals:

- Allow a client to send commands and change the state of the lights.
- Allow a client to inspect the system, sending them statistics about energy consumption and confort of the user.
- Run the Simplex implementation and send the results as commands to each arduino.

The Server is an interface between clients and the arduinos controlling each light, with the clients it communicates via TCP/IP and with the arduinos it uses a serial connection.

The Server was implemented in C++ using the C++ Standard Library and the *Boost ASIO* Library [2].

Although the Server was implemented using an asynchronous library it is only able to service the request of one client at a time. We considered it would only be connect to a lighting system at each moment. As the system is only able to service one request at a time we considered making the server asynchronous might lead to problems with overlaping communications. So the server is only able to answer to one client, if another client connects, it acepts the connection but only processes the request after the first client disconnects.

The client can request the information about a variable or statistic with three different modes: actual values (just the current reading), last minute values and stream (instead of a real stream the server outputs the next 100 readings).

As the assignment didn't specify how to change between these modes we implemented our own commands. The actual value commands were implemented exactly as in the assignment.

To indicate last minute or stream mode a character *'G'* was added before each command and then if the letters were capital it would mean stream and last minute otherwise.

At each received command over TCP/IP the server would send exaclty the same command to the Arduino it was connected. That Arduino would then be responsible for retrieving the information from the system. If it didn't have the information then it would communicate with it's neighbours via I^2C/TWI to collect it.

When there is a change in the occupancy state of the server runs the Simplex algorithm with the new configuration. It then takes the results and sends them to the arduinos. They turn off the controller take the results and apply them directly on the LEDs and only after that they change the references and restart the controllers.

2.5 Simplex

The Simplex algorithms solves Linear programms (LP's), it finds a optimal solution if it exists. We implemented the simplex following the pseudo-code on the book *Introduction to Algorithms* [7]. This implementation solves Linear programms of the form:

$$\max c * x$$

$$A * x \geq b,$$

$$x \geq 0$$

. We can describe the problem as a Linear programm of this form. First we define O as the array with the external illuminance at each *desk*, l as the array with the desired illuminance for each *desk* and E as the 3 by 3 matrix of influence between each light. The problem we want to solve is to find values for d, the PWM values normalized from 0 to 1, minimizing the energy spent and that make the illuminance be equal or greater than l. This can be formalized as:

$$\min \sum d$$

$$E * d \geq l - O,$$

$$d \geq 0,$$

$$d \leq 1$$

. If we define $c = [-1, -1, -1]$ we get:

$$\max c * d$$

$$E * d \geq l - O,$$

$$d \geq 0,$$

$$d \leq 1$$

. And to remove the last condition we concatenate an identity matrix to the matrix E and an array of 1's to the *l-O* array and get:

$$\max c * d$$

$$\begin{bmatrix} E \\ I_3 \end{bmatrix} * d \geq \begin{bmatrix} l - O \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$d \geq 0$$

With this form we can apply our Simplex algorithm directly.

The Simplex has three possible outcomes. It may find one of the optimal solutions, the ideal case. It may detect that there is no possible solution that satisfies all constrains, this happens in our case when we have the illuminance lower than l at at least one desk even if all ligths are set to their maximum value. Or it may declare the problem to be unbounded, where the objective function can have an infinite value, this won't happen in our problem as the maximum value for the objective function would be 0, when all lights are turned off.

Our Simplex implementation has three main routines similarly to the pseudo-code in [7]. The main *Simplex* routine, the *Initialize-Simplex* routine and the *Pivot* routine.

The main routine calls the *Initialize-Simplex* to generate a feasible solution and evaluates it, then makes successive calls to the *Pivot* until it reaches an optimal solution.

The *Initialize-Simplex* routine tests if the initial basic solution (setting x to all zero's) is feasible, if not it creates an auxiliary Linear Programm that if it has a solution there is a solution to the initial LP, it then solves the auxiliar LP.

The *Pivot* operation transforms the problem generating a new the solution.

The *Initialize-Simplex* is responsible for detecting if the problem is impossible while the main *Simplex* routine is responsible for detecting if it is unbounded.

Our implementation was able to solve all LP we tested it with, even impossible and unbounded LP's. The results were compared to the *MATLAB: Linear Programming Toolbox*.

2.5.1 Simplex Implementation Details

The Standard C++ library was used widely in the implementation.

Due to possible numerical problems all comparisons within the Simplex implementation are done within a defined *delta*.

To throughly test the Simplex implementation a series of simple unit tests were constructed using the *Boost Test* Library [2].

2.6 Arduino Communications

Each luminaire has two communication components: the communications with the computer, if one is connected; the inter-luminaire communications. The first is accomplished via serial (USART converted to/from USB through the on-board ATmega16U2); while the second is implemented with I^2C/TWI.

The `loop()` in the program multiplexes the two communication channels, by checking if there is incoming data for each of them. Two functions (`wire_process_incoming()` and `serial_process_incoming()`) process data according to its origin and act on it. These functions are called only if new data is available on the respective interface. This is accomplished by checking the output of `serial_read_str()` for serial and a variable that is set on wire interrupts for TWI.

During the first part of the project the serial communication used a baud rate of 115200. For the second part the Arduino has many more tasks running, one of which is computing and storing the metrics. This made the communication unreliable with the baud rate previously used. Therefore the baud rate was reduced so that serial communication kept working reliably. The final baud rate is 19 200 bps. Each command that can be sent to the Arduino is made of isolated characters separated by spaces that identify the command followed by the parameters of that command. Once a command is received a custom `split()` function is applied to the incoming data buffer. This command separates the buffer on spaces and creates a list (an array) of the words that constitute the command. This list can then be used to figure which command was received and if the required arguments for that command were provided. Invalid commands can be detected and reported. If the command is correct then the corresponding actions are performed. These actions can be changes to the luminaire itself or they can involve sending commands to the other Arduinos via TWI and getting back replies.

The TWI communication is handled by using the methods from the Arduino's Wire library. These methods have some drawbacks – for example `Wire.write()` is a blocking function – but it sufficed to perform the needed tasks and therefore a new TWI library was not implemented (this was also a time-wise decision). Each time there is incoming TWI data an interrupt is triggered. The interrupt routine asserts via a variable that wire data was received and then copies this data to a buffer for later processing. The communication protocol implemented over

TWI is similar to the one used for serial, only the commands change. The default bit rate for the Wire library was used. This bit rate is of 100 kHz [1]. There is some overhead for each TWI communication since a start bit, the address and mode (1 B), ACKs (1 bit for each byte of data) and a stop bit must be written to the bus, apart from the data that is actually to be sent [2]. Therefore the nominal transmission rate is lower that the frequency the TWI is operating at. The impact of the overhead decreases with bigger packets of data. Nonetheless the Arduino's Wire library limits transmissions to payloads of 32 bytes [3]. The bit rate could be increased up to 400 kHz.

3 Experiments

To analyze the resulting system 3 metrics were implemented (as specified on the assignment):

- Energy: the energy consumed at each desk assuming each LED has a nominal power of 1W
- Comfort error: the average error between the reference illuminance and the measured illuminance for the periods when the measured illuminance is below the reference
- Comfort variance: the average variation of the illuminance during periods of constant occupation computed by and approximation to the second derivative of the illuminance.

We used these metrics to evaluate the system's performance on our experiments. First to test the Local Controller operation with multiple Arduinos working simultaneously we ran two experiments. The results of these experiments are presented and discussed on section 4.

With the box closed (no external lighting) and with all desks set as unoccupied (reference of 15 lx) we analyzed the three metrics on the first 6 seconds after system startup.

With the box slightly open (some external lighting) and with the first and the third desks set as occupied (reference of 30 lx) we analyzed the metrics in the same way.

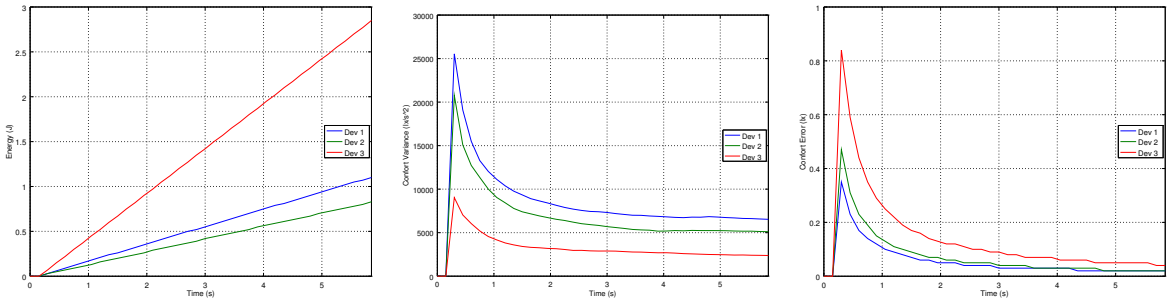
The values for the occupied and unoccupied reference were chosen taking into consideration Figure 7.

4 Results & Discussion

In this section the obtained results are shown and discussed. First we present the results for the analysis of the system through the metrics and with only the local controller. In Figure 11

[1] Arduino's GitHub, `twi.h`, line 28
[2] I^2C's Bus Specification
[3] Arduino's GitHub, `Wire.h`, line 28

and Figure 12 we show two cases of the evolution of the three metrics after the startup of the system.



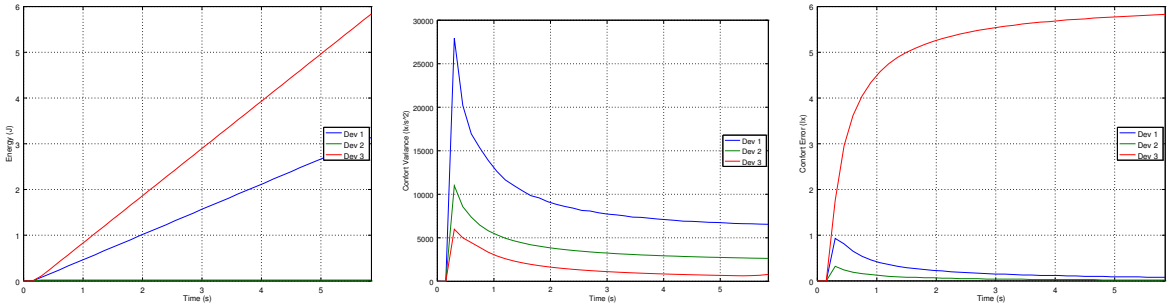
(a) Energy over time of the 3 luminaires.

(b) Comfort variance over time of the 3 luminaires.

(c) Comfort error over time of the 3 luminaires.

Figure 11: Metrics after system startup with the box closed and all desks as unoccupied.

The first figure corresponds to the system without external light and with all the references to the lower value 15 (unoccupied state). The energy spent rises in a linear way in all illuminaires, with desk 3 rising faster. This can be explained if we consider all luminaires to be at a fixed, but different, duty cycle, with luminaire 3 being the one with the LED at a higher intensity. Both the comfort error and the confort variance present the same behaviour, with the difference that the error tends to 0, while the variance tends to a small, but larger than 0, value. The initial spike in both metrics is explained by the startup of the Local controller, when all LED's are off. The remaining value in the variance shows that there is some flicker in the LED's that is however unnoticeable with a naked eye. This flicker can be explained by noise on the reading of the illuminance or due to not enough precision with the PWM or even some lack of tuning in the PID.



(a) Energy over time of the 3 luminaires.

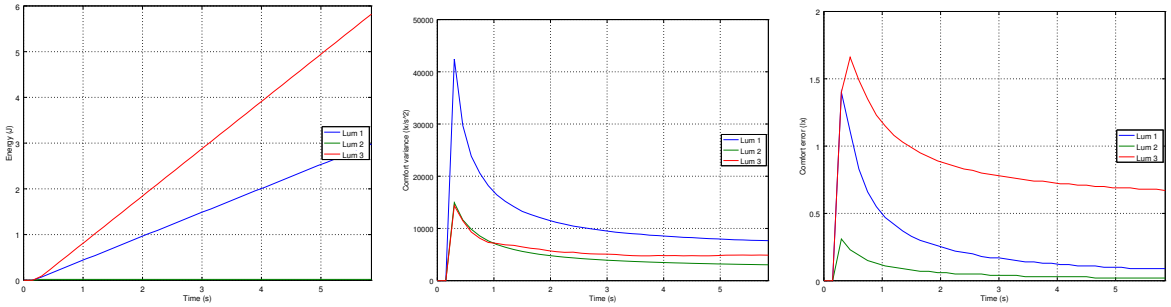
(b) Comfort variance over time of the 3 luminaires.

(c) Comfort error over time of the 3 luminaires.

Figure 12: Metrics after system startup with the box open and desks 1 and 3 as occupied.

On Figure 12 we see some similarities on the behaviour of the metrics. The energy for luminaires 1 and 3 rises linearly however energy spent by luminaire 2 is approximillarly 0. This means that the second LED is off and that it isn't necessary to reach the desired illuminance. The main difference in this result is that the confort error for the luminaire 3 doesn't converge to 0, but to 6. This means that the even with the luminaire at it's maximum intensity the sensor can't read the 30 lux value, only 24. This might seem weird, but there are many reasons to explain this. First the steady state plot on Figure 5 was taken for luminaire 1 and not 3 that might have different characteristics such as the reflection of the light. Also the 3rd LDR is the only one that doesn't have a LED to it's right. The most important factor is the fact that the box is slightly open, this causes the LED and the LDR to not be directly in their reflection path.

With the simplex already in use we ran another experiment with the results displayed on Figure 13. This experiement was run with the box closed and with desk 1 and 3 set as occupied.



(a) Energy over time of the 3 luminaires.

(b) Comfort variance over time of the 3 luminaires.

(c) Comfort error over time of the 3 luminaires.

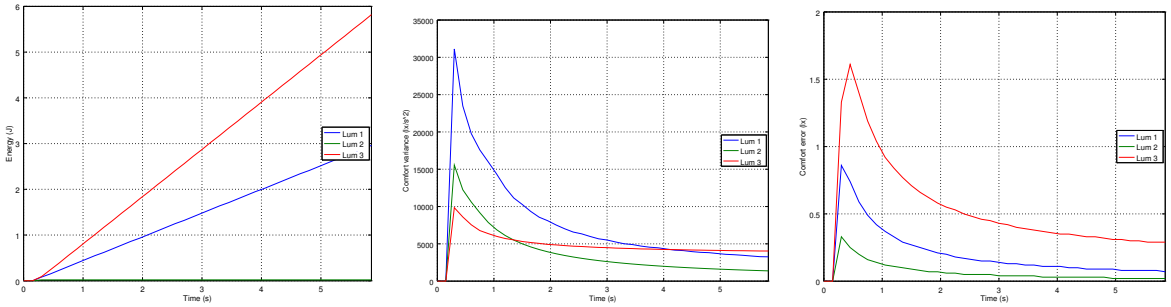
Figure 13: Metrics after system startup with the box close and desks 1 and 3 as occupied with the Simplex algorithm initializing the LED's intensities.

With the same system configuration but without the Simplex we obtained 14.

The results in Figure 13 are clearly worse than the ones in Figure 14. This is explained by the fact that the system doesn't converge to the same solution proposed by the simplex, causing more effort on the local controllers to satisfy the conditions. This causes the system to be inefficiont as the simplex's solution uses less energy. The final states of the duty cycle of the LED's are 52%, 0% and 100% while the simplex sets them to 85%, 0% and 62%.

5 Conclusion

The final results of this project constitute a control system that allows to control ambient light to meet desired levels specified according to the occupation of parts of a room. The system includes a server that can be connected to a network of luminaires allowing to control



(a) Energy over time of the 3 luminaires.

(b) Comfort variance over time of the 3 luminaires.

(c) Comfort error over time of the 3 luminaires.

Figure 14: Metrics after system startup with the box closed and desks 1 and 3 as occupied.

the references of each as well as acquired data to measure the energetic efficiency and comfort metrics of the illumination system.

One contribution of this work is the implementation of an interrupt based PID controller that runs on a common microcontroller. Another important contribution is the implementation of the Simplex algorithm according to the pseudocode presented in [7]. This implementation includes the needed transformations for cases where the initial solution is unfeasible. No previous implementation that follows this work was found to exist.

The use of the results provided by the simplex failed to improve the system. Even when the results of the simplex algorithm are applied the system evolves naturally to the state it was before. That state satisfies the illumination requirements but is proved not to be optimal energy-wise.

There is room to improve several aspects of the system. Namely the protocols used could be more robust; the ATMega ADC noise reduction mode could be used to acquire better samples; and the ADC readings could be made in the background by using the interrupts that exist for that purpose and a circular buffer instead of using `analogRead()` several times and make a mean of the values to filter some noise.

References

[1] Arduino's GitHub. `https://github.com/arduino/Arduino`. Accessed: January 3, 2016.

[2] *BOOST C++ Libraries*. `http://www.boost.org`. Accessed: January 3, 2016.

[3] K. Aström and R. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[4] M. Bürger, G. Notarstefano, F. Bullo, and F. Allgöwer. A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. *Automatica*, 48(9):2298–2304, 2012.

[5] D. Caicedo and A. Pandharipande. Distributed illumination control with local sensing and actuation in networked lighting systems. *Sensors Journal, IEEE*, 13(3):1092–1104, 2013.

[6] D. Caicedo, A. Pandharipande, and G. Leus. Occupancy-based illumination control of led lighting systems. *Lighting Research and Technology*, 43(2):217–234, 2011.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[8] A. Pandharipande and D. Caicedo. Daylight integrated illumination control of led systems based on enhanced presence sensing. *Energy and Buildings*, 43(4):944–950, 2011.

[9] T. Wescott. PID without a PhD. *Embedded Systems Programming*, 13(11):1–7, 2000.