Instituto Superior Técnico

MSc in Electrical and Computer Engineering

Distributed Real Time Control Systems

Real-Time Cooperative Decentralized Control of Illumination Systems

Filipe Parrado	83399
Paulo Bernardo	84157
Miguel Viegas	86335

Group 6

January 10, 2020

Abstract

This report entails the development of a distributed cooperative control system whose purpose is to control the illuminance in an enclosed space, using micro-controller technology, CAN (Control Area Network) bus and a consensus algorithm as well as an interface that allows both the monitoring and the alteration of a series of parameters related to the control system. The main challenges are a fast and reliable CAN bus internode communication, as well as the fine tuning of the controllers used in the non cooperative part of the control system. The work first began by developing a system of decentralized, non cooperative control and then using a consensus algorithm to implement the decentralized cooperative control system. A robust and fast internode communication was achieved, as well as the set of PID controllers that are fine tuned and respond quicky to changes in the overall system. The interface developed is capable of monitoring and altering a set of parameters, although not all of the commands requested are implemented.

Contents

1 Introduction

In today's world there is an ever increasing emphasis on energy efficiency, be it in the cars we drive, or the appliances we use. One area that has seen great improvements in efficiency has been artificial lighting, with the rise of LED technology, we can now light a significant area with a relatively low amount of power. With that said, in this project a further look is taken into efficiently lighting enclosed spaces, namely offices.

In offices, or any indoor space, there generally are a series of light bulbs or LEDs, typically equally spaced, that provide lighting to that room, typically, if somebody wants to illuminate said room, they would just turn a switch that toggles all of the lights on. In most situations this is very inefficient, since some spaces don't need as much illuminance as others, especially in an office where some cubicles, or desks, might be empty, where as others may be occupied. One solution to this might be to just have individual light switches for each desk, such that whenever someone sits down at said desk they would just turn the switch on and the bulb would turn on, at full power, otherwise, it would be off. Although this is more efficient, it still isn't optimal or practical, since some light may be needed on the general vicinity of those desks, despite the desk itself being vacant. There is also still room for optimization, since nothing guarantees that a light is needed at full power, even when the desk that light is under is occupied.

Another, better solution, would be to still have a switch, but instead of the switch controlling the light directly in a binary fashion (full throttle or off), it would change the illuminance level desired at that specific bulb (or luminaire). For example, suppose that a user defined that when they were on a specific desk, they would want **at least** 80 LUX of illuminance at the surface of their desk. Meanwhile, the owner of the building wanted **at least** 20 LUX at the surface of every single desk, in order to have proper illumination in that specific room. We now have 2 constraints, when any desk is occupied we need a minimum of 80 LUX on that desk, and whenever it is not occupied, we need a minumim of 20. Therefore, we can adapt our switch to, instead of just toggling the light at full power, signal a control system that we need at least 80 LUX on that desk, and if the switch is toggled off, the control system knows it only needs to guarantee 20 LUX.

For this solution to be implemented, sensors would be needed at the surface of every desk, which would also signal the control system. For even further efficiency, the control system would take into account light leaking from one luminaire to another, and compensate the power on all the LEDs to adjust to that, maybe one luminaire being set to 80 LUX is already enough to get 20 LUX on the desk next to it, therefore only that luminaire needs to be on. This would also imply some form of communication between the luminaires' controllers. Lastly, this control system needs to be resilient to outside light, or distrubances, otherwise opening a window wouldn't bode well for this system.

For this project, a very crude prototype of an office room with luminaires was assembled, using a shoe box and LEDs, the focus was on the control system for these luminaires. LDRs were used as desk iluminance sensors, and arduinos were used as micro-controllers for the luminaires. In order to achieve communication between the arduinos, CAN (Control Area Network) bus was used, with every single arduino possessing and CAN bus module. Using all of theses resources, a prototype was aspired of an office with a smart and efficient lighting setup.

2 Problem Definition

The problem at hand (the real-time cooperative decentralized control of illumination systems) will be defined as an optimal control problem. In other words, the solution to the problem is achieved when a control law for the dynamical system over a period of time is found such that an objective function is optimized [1]. As the main objective of the cooperative decentralization of the control system is to maximize efficiency (minimize total energy consumption), the objective function is a energy cost function, and the optimization of this function is the minimization of its output, hence the definition of the problem as an optimal control problem. Conceptually, the problem can be defined as:

"Given a set of $N \in \mathbb{N}$ luminaries, each with a possibly different energy consumption, a set of N desks, each approximately in the same of location of the luminaries, and considering the possibility of outside disturbances, what is the target illuminance of each luminary, where said target illuminance is defined by the occupation of M desks, where M is a subset of N, knowing that the objective is the achievement of the target illuminance specified whilst spending the optimal (minimal) amount of energy."

The solution to this problem can be broken down into two main and sequential parts. The first one is consists in finding the optimal illuminances in each luminary. These illuminances are found by computing the consensus algorithm provided in the course's lectures [2]. After each luminary has it's desired optimal illumiance, the second part of the solution consists in maintaining said illuminance level throughout the operating time of the system. This is achieved by the use of a PID controller. It is worthy to point out that the solution of the problem is then broken down into a decentralized cooperative stage (the computation of the consensus algorithm), and a decentralized, non-cooperative stage (the maintenance of the target illuminance provided for each luminary by it's own PID controller).

3 Proposed Approach

As stated in the section above, the solution to the problem can be broken down into two major parts, the development of a decentralized, cooperative control system (the consensus algorithm), and the development of a decentralized, non-cooperative control system (the PID controller). The development of the solution first began by developing the decentralized non-cooperative control, and then the decentralized cooperative control. The following subsections detail the methodology of each part.

3.1 Decentralized non-cooperative control, feedforward and the local PID controllers

The decentralized non-cooperative local control of each luminary is composed by two control systems, a feedforward control system and a feedback control system.

In order to explain as best as possible the feedforward, the PID controllers and the local control system of each luminary, it is important to specify some of the characteristics of the hardware used, namely the relation between the LED (luminary) and the LDR (the desk's light sensor). The LED produces a level of illuminance that is proportional to it's input duty cycle. The LDR is a electrical component that detects the level of illuminance surrounding it and converts said level into a voltage. This LED/LDR pair is the plant of our control system, it's composition can be found in the project's handout and is here presented in Figure 1

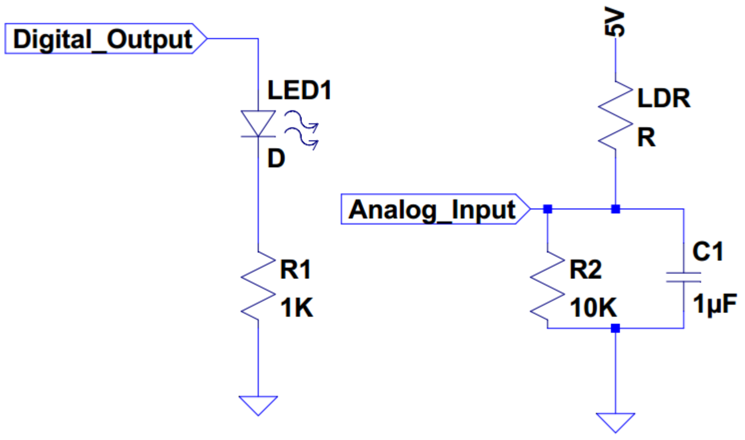

Figure 1: *LED driving and illuminance reading circuit.*

The `Digitial_Outout` and `Analong_Input` tags denote the inputs and outputs of the arduino, where the computations for the control will be made. In a very succinct manner, `Analog_Input` will be the voltage read from the LDR, which will be the input variable of the control system. The output variable of the control system is the duty cycle to be injected into the LED (`Digital_Output`). This continuous loop will be controlled by the PID controller and the feedback control system.

The feedforward control system is simply the injection of the desired illuminance into the system's plant. Since the plant is prepared to receive a duty cycle, a conversion from LUX to duty cycles must be computed, which will be discussed in the Development section.

The final control system consists on the unification of the feedforward and the feedback control systems. In Figure 2 the local control system is presented.

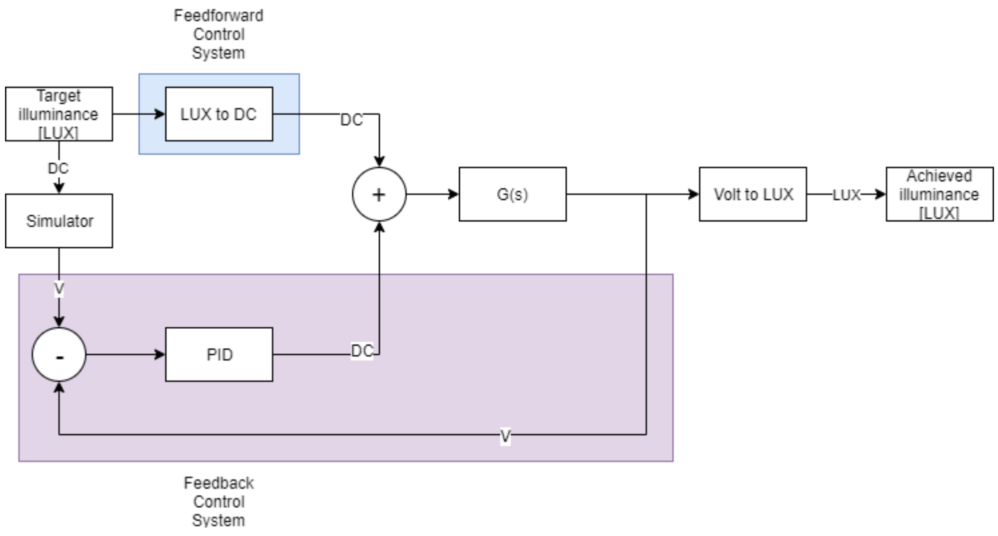

Figure 2: *Local control system.*

In Figure 2 the `G(s)` denotes the plant of the system (the LED/LDR pair), and the `Simulator` block computes the theoretical (ideal) illuminance value at each time step, and is used to compute the error to be corrected by the `PID`.

The PID controller boasts, along it's standard qualities, anti-windup and a deadzone. Given that the the control system is computed the PID controller has to be a digital PID. This imposes the consideration of quantization error that is meant to attenuated by the deadzone implemented. The anti-windup is meant to stop the over-accumulation of the integrative term of the PID controller such that if an outside disturbance is detected for a long period of time the integrator would not continue to rise indefinitely.

It is worthy to point out that the main difficulties related to this part of the control system are the calibration of the conversion functions, the calibration of the PID constants and the design of the PID controller, namely the addition of a deadzone and anti-windup. All these aspects will be detailed in the Development section of the report.

3.2 Decentralized cooperative control, the consensus algorithm

The need for a decentralized cooperative control system stems from the fact that the illuminance emitted from each LED can be captured by more than it's own LDR. Figure 3 represents this phenomena for three LED/LDR pairs, the number of pairs provided to develop the project.

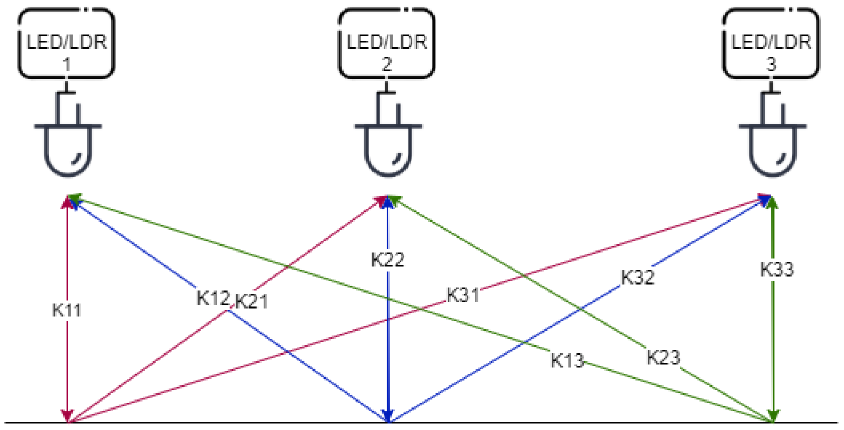

Figure 3: *The effect that each LED has in each LDR.*

Figure 3 exemplifies the influence of each LED in each LDR. The K constants quantify the illuminance that each LDR gathers from each LED. These constants are computed each time that the consensus algorithm is run, which again will be explained in further detail in the development section of the report.

4 Development

The following section details all the specific components that allow the successful implementation of the project.

4.1 Overall System

The overall system used in the development of the project consists in three LED/LDR/arduino sets, even though the software developed is ready to accept the insertion of more of the same sets. All the sets are connected to each other via a CAN bus module that allow the sets to communicate between them. Each set has a local control system like the one seen in Figure 1 although the PID constants differ from set to set. The overall diagrams of the whole system of the local assembly can be see in Figures 4 and 5

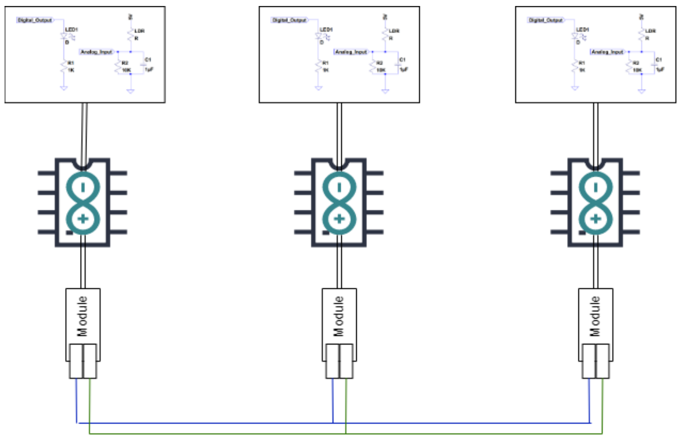

Figure 4: *Overall system diagram.*

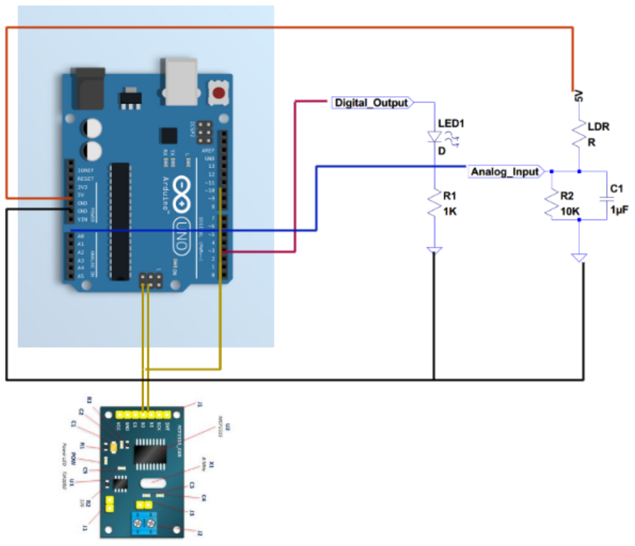

Figure 5: *Local system diagram.*

In Figure 5 is represented all the of the wires connection, namely the connections between the CAN bus modules and the arduino board. In order to avoid any misinterpretations, all the wire connections are listed bellow.

- The `Analog_Input` is connected to pin `A0`
- The `Digutal_Output` is connected to pin `3` , which is a PWM pin
- The grounds and $5V$ are connected respectively to the `5V` and the `GND` pins
- The CAN-bus modules pins are connected to the arduino in the following order
  - The `VCC` pin is connected to pin `10`
  - The `GND` pin is connected to pin `2`
  - The `CS` pin is connected to `ISCP[2,3]`
  - The `SO` pin is connected to `ISCP[1,3]`
  - The `SI` pin is connected to `ISCP[2,3]`
  - The `SCK` pin is connected to `ISCP[1,2]`
  - The `INT` pin is connected to `ISCP[2,1]`

There are other connections present in the arduinos, namely the wires that connect the arduinos to each other in order to power up the whole system, more precisely, 2 connections between arduino 1 and arduino 2, and other 2 between arduino 2 and arduino 3. These connections are provided below.

- The `ICSP2[1,1]` pin, from arduino 1, is connected to `ISCP2[1,1]` pin from arduino 2
- The `ICSP2[3,1]` pin, from arduino 1, is connected to `ISCP2[3,1]` pin from arduino 2
- The `IOREF` pin, from arduino 2, is connected to the `IOREF` pin from arduino 3
- The `GND` pin, from arduino 2, is connected to the `GND` pin from arduino 3

These are all the connections that allow the assembly of the project, the following subsections detail the development of each of the project's core components.

4.2 The luminaire: construction and modeling

As stated before, the luminaire model consists of a LED/LDR pair. Perhaps the most important of this component is the relations between all the variables, namely the LED's input duty cycle, the LDR's input illuminance and the LDR's output voltage. These relations are of the utmost importance as they allow the calibration of the system, creating the expressions that allow the conversion between variables present in Figure 2. These relations were obtained by running the software and printing to the arduino's IDE serial output the values of the variables along a certain number of iterations. After these data sets were gathered, MatLab's curve fitting toolbox was used to obtain the functions that express the relations between the variables.

The first relation obtained is the one that converts the photo resistance of the LDR into a illuminance. To obtain this relation, the LDR's `Illuminance Vs. Photo Resistance` plot present in it's datasheet [3] and in Figure 6 was used, along with the expression present in the course's slides [4], $L = 10^{\frac{log_{10}(R)-b}{m}}$.

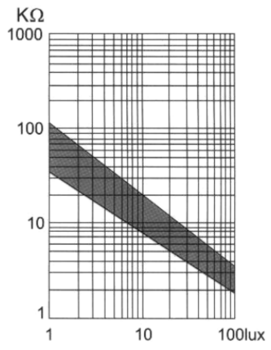

Figure 6: *Illuminance VS Photoresistance of the LDR.*

Using Figure 6 and the expression provided, it is possible to compute values for m and b that convert photo resistance into illuminance. These values are constant for all arduinos and are shown in Table 1.

Table 1: *Computed values for m and b*

m	b
-0.76	5.0337

Now that it is possible to compute the input illuminance value of the LDR, the next of the relations are made possible and are shown, along with their functions in Figures 7 to 9.

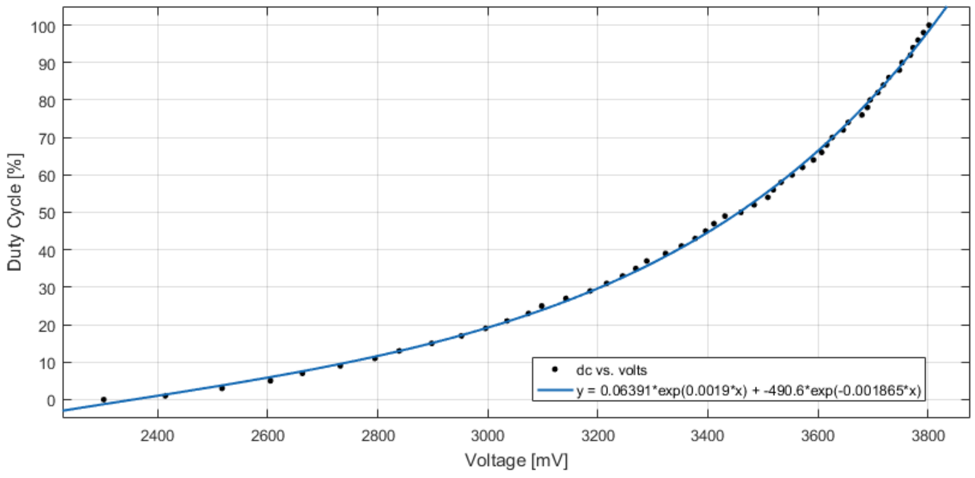

Figure 7: *Voltage to duty cycle.*

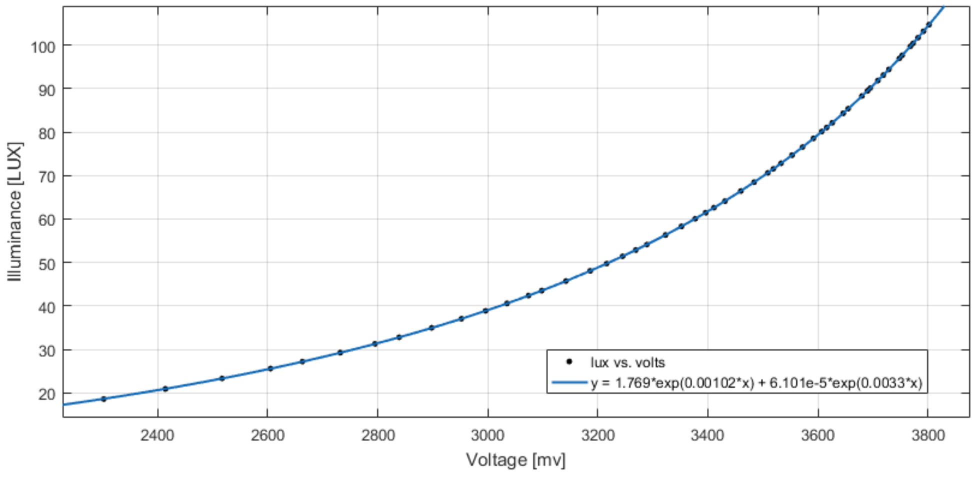

Figure 8: *Voltage to illuminance.*

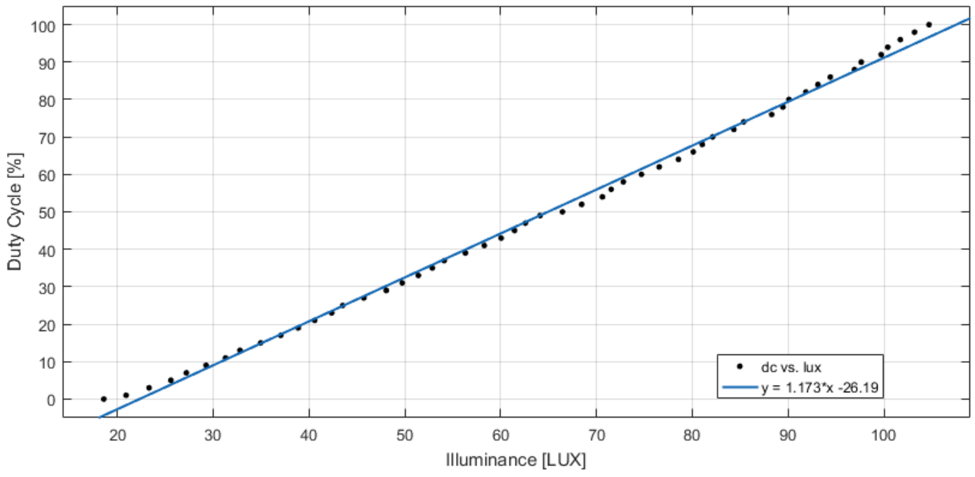

Figure 9: *Illuminance to duty cycle.*

These are the relations that were used throughout the code although many more can be computed from the data sets obtained. Along the development of the project many other relations were used but ultimately these were condensed into the relations presented above.

4.3 The local controller

As stated before, the local controller consists in two major parts, the feedforward control system, and the feedback control system, which can be seen in Figure 2. The feedforward control system is simply the injection of the desired reference value into the system's plant. The feedback control system consists in a feedback loop, in which is present a PID controller that boast both anti-windup and a deadzone. In order to develop the local controller, first a basic PID controller was designed and then added anti-windup and deadzone.

A PID controllers is a controller that is the unification of a proportional controller, a derivative controller and a integral controller and whose diagram is shown in Figure 10.

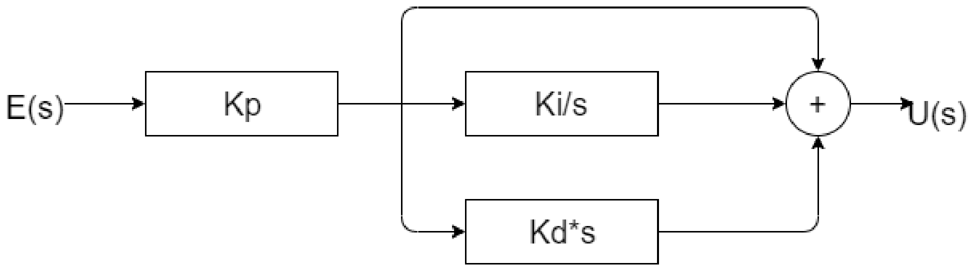

Figure 10: *A basic PID controller.*

Each component of the controller has its upsides and drawbacks. The proportional term makes the system reaction faster, but it increases the overshoot and the frequency of oscillations, for systems without integrators it may not lead to a zero static error and for systems with significant delays it may not stabilize the system. The integral term alone decreases the system's stability and it is usually combined with proportional control. This combination however, is slow for systems with integrators and it may wind-up if the actuators saturate. The derivative term can do some prediction about the output evolution. PD control is effective in stabilizing systems with large delay and PID control is able to speed up response and keep low overshoot and oscillations frequency, however it's very sensitive to noise [5].

Because the system has a small delay and is supposed to be noise resistant, a PI control (proportional, integrative) was implemented. This then imposed the need to implement anti-windup in case the actuators saturate. In order to find the K_p and K_i constants, MatLab's PID tuner toolbox was used. It is important to mention that the total latency of the system was set to 10 ms. This value was chosen because it is enough time to compute all the values needed to ensure that the system behaves correctly. After some iterations of the controllers constants, the group achieved the constants that provided the best system response whose values are present in Table 2.

Table 2: *The PID controller's constants*

Kp	Ki	Kd	a	T[s]
0.003	100	0	0	0.01

The anti-windup and the deadzone limits were found iteratively, by attempting several limits and finding the ones that produced the best results. These limits can be seen in Table 3.

Table 3: *Anti-windup and deazone limits*

Anti-windup	Deadzone
± 150	± 50

The anti-windup takes effect every time the integral term goes either above or below 150. The deadzone applies every time that the input value is between +50 and - 50 mV. It is important to point out that the PID's inputs is voltage and it's outputs duty cycles. It is also important to recall that the PID designed must be converted into a digital PID. This conversion is made possible by the expressions that convert the controller's constants into their digital counterparts. This can be achieved with the expressions below, present in the courses slides[6]:

$$K_1 = K_p \tag{1}$$

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

$$K_3 = \frac{K_d}{K_d + aT} \tag{3}$$

$$K_4 = \frac{K_p K_d a}{Kd + aT} \tag{4}$$

In order to compute the error that is the input of the PID, a simulator was developed. This simulator allows the comparison at each time step between the predicted output voltage and the real output voltage that the system is outputing. The simulation can be computed by implementing the expressions found in the courses lectures[7] and are as follows:

$$v(t) = v_f - (v_f - v_i)e^{-\frac{t-t_i}{\tau(x_f)}}, \tag{5}$$

$$v_f = V_{cc}\frac{R_1}{R_1 + R_{LDR}(x)}, \tag{6}$$

$$\tau(x) = R_{eq}(x)C_1, \tag{7}$$

$$R_{eq}(x) = \frac{R_1 R_{LDR}(x)}{R_1 + R_{LDR}(x)}, \tag{8}$$

$$R_{LDR}(x) = 10^{m log_{10}(x)+b}. \tag{9}$$

All these components combined provide the local controller, whose diagram can be seen in Figure 2. It is important to remind that to adapt the local controller to the decentralized cooperative, the feedforward is switched to the consensus algorithm output.

4.4 The distributed controller

In order for one luminaire to compensate for the excess light of another, all of the individual controllers need to agree with every other controller's duty cycle. This is so that the total energy consumed by the system is minimized.

To achieve this minimization, an iterative consensus algorithm[2] is used. This algorithm allows for every luminaire to have its own constraints, i.e. the maximum and minimum LUX. Every micro-controller will have two arrays of duty-cycles, with size N, where N is the number of luminaires/controllers, one which will be local: $\mathbf{d}$, and another one which all of the controllers have to agree on: $\bar{\mathbf{d}}$. The overhead bar denotes the average over all $\mathbf{d}$ arrays, i.e.:

$$\bar{\mathbf{d}} = \sum_{i=1}^{N} \frac{\mathbf{d}_i}{N} \tag{10}$$

Where i denotes a node.

In order for them to agree on the same value of $\bar{\mathbf{d}}$, on the end of every iteration, every arduino sends their $\mathbf{d}$ array to all others. In a single iteration, all of the arduinos/nodes will attempt to minimize a cost function, and afterwards send its own solution to all others, so they can calculate the average. Beyond this, every node also maintains a local variable of Lagrange multipliers: $\mathbf{y}_i$, and a quadratic penalty parameter: ρ.

The quadratic component can be adjusted using its previously mentioned parameter in order to achieve more accurate solutions in less iterations. The work group found $\rho = 0.07$ to be sufficiently fast, therefore this value was used for all nodes.

For further comprehension of the algorithm used, refer to [2].

4.5 Initialization and calibration

At the beginning of each run of the program, there is a need to calibrate each of constants of illuminance from every arduino (each constant effect can be observed in the Figure 3). There's also the need of a synchronization event so that every arduino connected to the network (CAN-BUS) "knows" the ID's of all the arduinos connected to the network. The implementations of these functionalities are not trivial due to the fact that every arduino run the same program (line by line).

The full initial routine is present in the Flowchart represented in the Figure 11. This routine's flowchart corresponds to a simplified version of the project's first function - *setup*. As the aforementioned figure suggests, this function is in charge of the masks creation and the variables initialization (namely each of the arduinos reads its own ID from the EPROM), the list creation and the k and o constants. The last two mentioned functionalities are implemented with two other functions, respectively, *build_ID_list()* and *init_sync()*. Both of these functions are far too complex to explain by words, therefore flowcharts are presented for each of the functions previously mentioned.

During the testing phase of the initial routine, some synchronization problems were found, namely, different arduinos were in different parts of the routine. The solution for these problems was the placement of several delays (most of them shorter than 0.1 s) between each key component of the aforementioned functions.

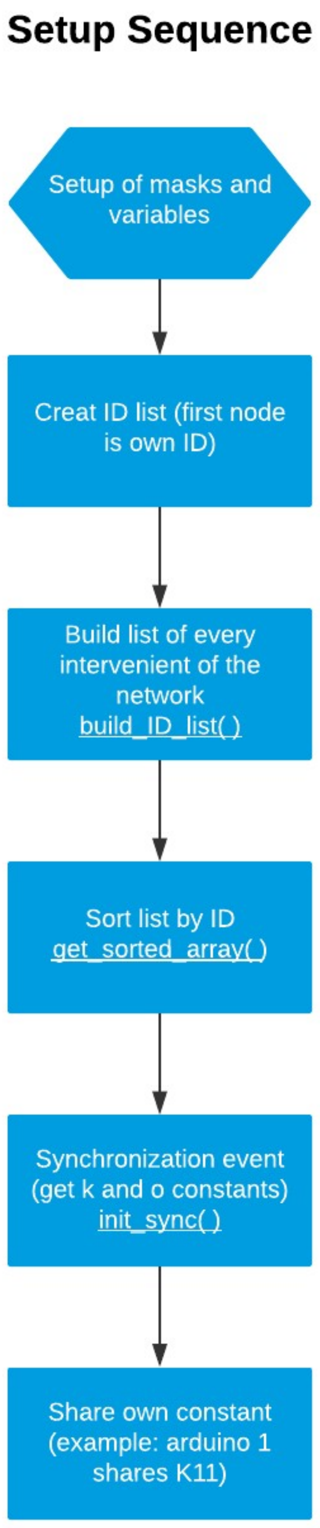
Setup Sequence
Setup of masks and variables
Creat ID list (first node is own ID)
Build list of every intervenient of the network build_ID_list( )
Sort list by ID get_sorted_array( )
Synchronization event (get k and o constants) init_sync( )
Share own constant (example: arduino 1 shares K11)

Figure 11: *Flowchart of the initial routine (setup function)*

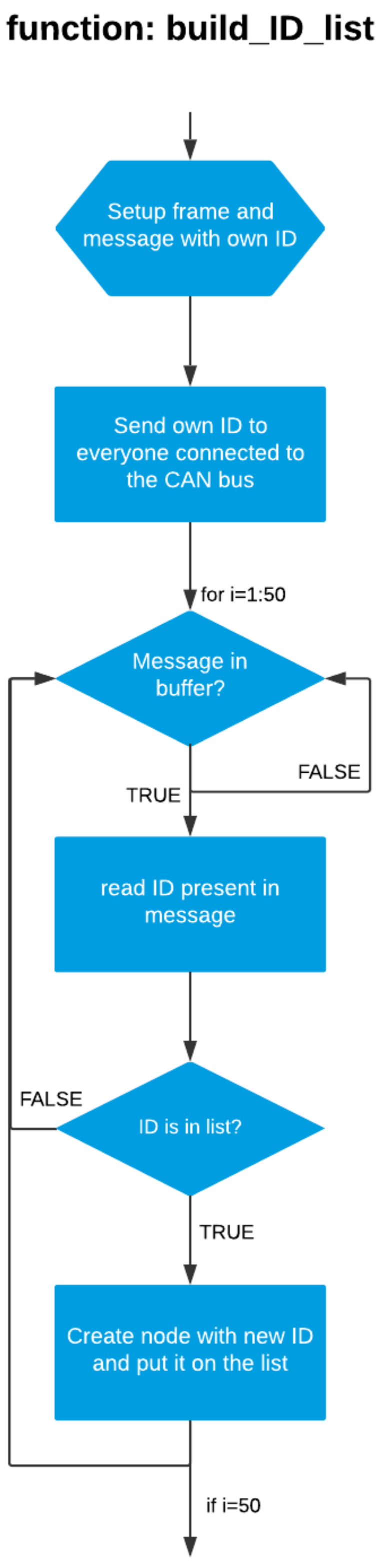
function: build_ID_list
Setup frame and message with own ID
Send own ID to everyone connected to the CAN bus
for i=1:50
Message in buffer?
FALSE
TRUE
read ID present in message
FALSE
ID is in list?
TRUE
Create node with new ID and put it on the list
if i=50

Figure 12: *Flowchart detailing build_ID_list function*

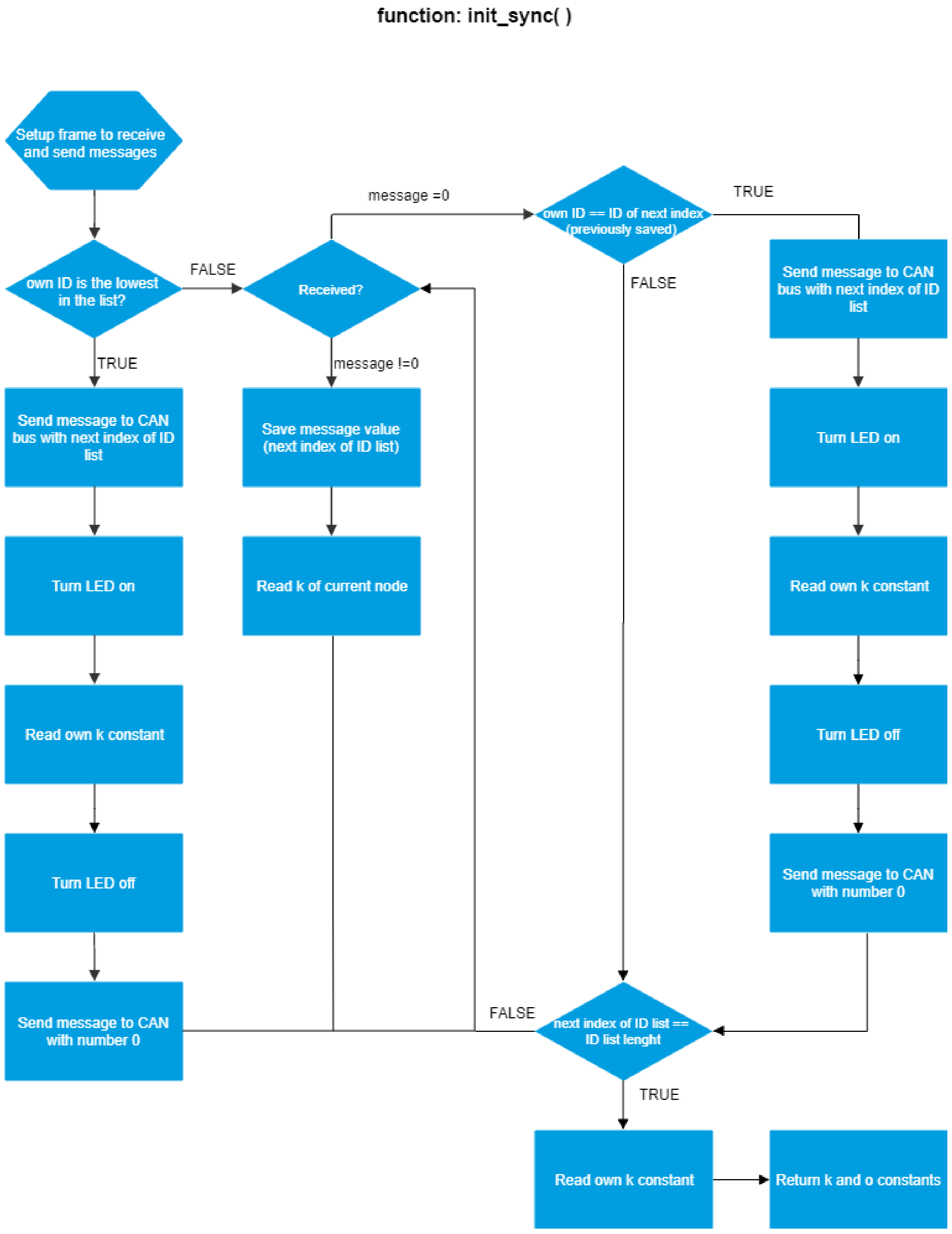
function: init_sync( )
Setup frame to receive and send messages
own ID is the lowest in the list?
FALSE
TRUE
Received?
message =0
message !=0
own ID == ID of next index (previously saved)
TRUE
FALSE
Send message to CAN bus with next index of ID list
Turn LED on
Read own k constant
Turn LED off
Send message to CAN with number 0
Save message value (next index of ID list)
Read k of current node
Send message to CAN bus with next index of ID list
Turn LED on
Read own k constant
Turn LED off
Send message to CAN with number 0
next index of ID list == ID list lenght
FALSE
TRUE
Read own k constant
Return k and o constants

Figure 13: *Flowchart of the synchronization sequence (init_sync function)*

4.6 Communications

As mentioned previously, the controllers utilize CAN bus to communicate with one another.

The `arduino-mcp2515`[8] library was used to interface with the network. Both of the terminating 120Ω resistors were placed on the open ended CAN bus nodes (first and third arduino).

In a CAN bus frame, there is an ID field with 11 bits of length, and a data field with varying byte size (0-8 bytes), among others. For the sake of standardization of code, in the project, every frame sent has 4 bytes of data. The ID field is used to identify which arduino is sending the message, i.e. whenever a message is sent, the ID field contains the unique identifier for the sending arduino. Whenever a message is read, the receiver then knows which arduino sent it.

In regards to the data field, what is sent varies in regard to the circumstance, in the case of the function `init_sync`, it sends the next ID in the ordered list. When it comes to the PC application, the data field consists of 3 characters (3 bytes, no spaces). This is to say that, due to the frame size limitations, and variety of uses for the data field, there is not a consistent protocol for the data field.

4.7 The PC application

In order to interface with the illumination network and log information for diagnostics and evolution a PC application was developed. This application was implemented using asynchronous I/O services provided by the Boost ASIO library. The application developed uses two threads, one for receiving the information from the network, and another for reading user input and sending the request to the network. There were two major considerations when developing the application. The first was that any arduino could be connected to the PC, which implies that both the application and the arduino's code should be ready to accept the commands and send them to the serial monitor (where the data is written and read). For example, it is possible to request the measured illuminance at any given desk. That is, if the PC is connected no arduino 1, but the request was made for arduino 2, the application and the arduino's code have to be able to comply and return the value requested. The second was that the client requests are formatted in a certain way, and some string manipulation was used so that the arduino's code could correctly interpret the users commands. Many request were provided in the projects handout [9]. Unfortunately, the group was not able to implement all of the requested commands. The last three commands (Get last minute buffer of variable ⟨x⟩of desk ⟨i⟩, Start and Stop stream of real time variable ⟨x⟩of desk ⟨i⟩) were not implemented as the group was not able to allocate the time needed for this implementation.

5 Experiments

In order to evaluate the system's performance, three performance metrics were provided in the project's handout [9], the total energy consumption at each desk, E_j, the visibility error at each desk, V and the flicker error at each desk, F. The sum of each of these metrics for each desk provided the total performance metrics. The expressions for the aforementioned performance metrics are:

$$E_j = P_j \sum_{i=1}^{N} d_{i-1}(t_i - t_{i-1}), \tag{11}$$

where P_j denotes the maximum power of the luminaire j, i is the index of the control samples, t_i is the time in seconds of the $i-th$ sample, and d_i is the LED duty cycle value (between 0 and 1) at sample time t_i.

$$V = \frac{1}{N} \sum_{i=1}^{N} max(0, L(t_i) - l(t_i)), \tag{12}$$

where $L(t_i)$ is the reference illuminance and $l(t_i)$ is the measured illuminance at time instant t_i,

$$F = \frac{1}{N} \sum_{i=1}^{N} f_i, \tag{13}$$

where f_i is defined as

$$f_i = \begin{cases} (|l_i - l_{i-1}| + |l_{i-1} - |l_{i-2}||) \quad if \quad (l_i - l_{i-1})(l_{i-1} - l_{i-1}) < 0 \\ 0 \qquad otherwise, \end{cases} \tag{14}$$

where l_i is the measured illuminance at time t_i and T_s is the sampling period. These metrics can be requested by the user via PC interface.

5.1 Feedforward vs Feedback

In order to analyse the impact of the feedforward and feedback controllers in the system, each of said controllers was studied individually, evaluating the controllers performance along 66 iterations for two cases. The first case consists in the response of the system with said controller from 0 LUX to 80 LUX (the system turned off to the system's occupied state). The second case consists in the response of the system with each controller from 80 LUX to 20 LUX (occupied state to unoccupied state). Finally, the performance of the total control system (feedforward and feedback) is shown. The results for these experiments can be seen in Figures 15 to 18.

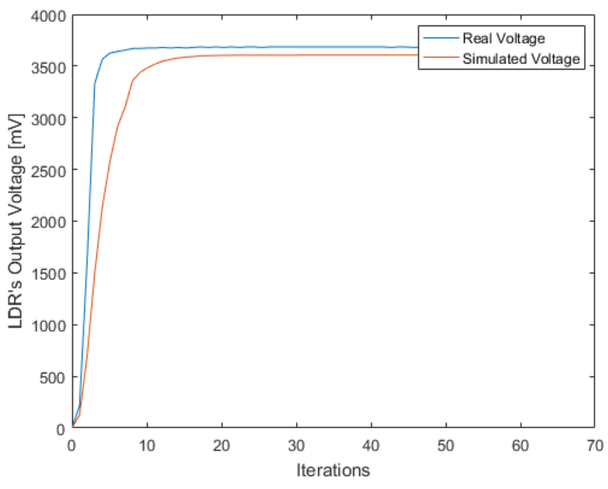

Figure 14: *Feedforward system from 0 LUX to 80 LUX*

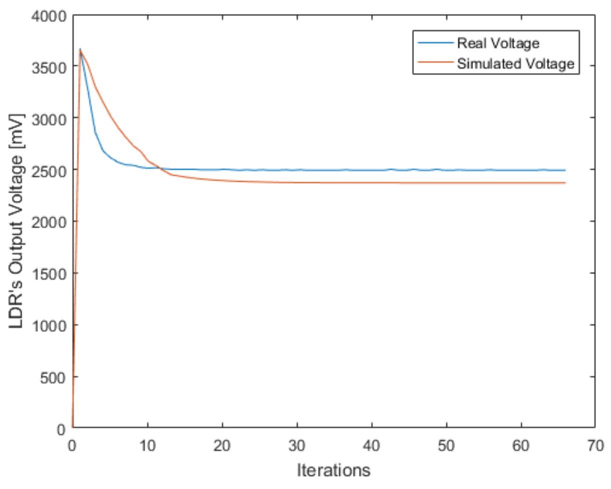

Figure 15: *Feedforward system from 80 LUX to 20 LUX*

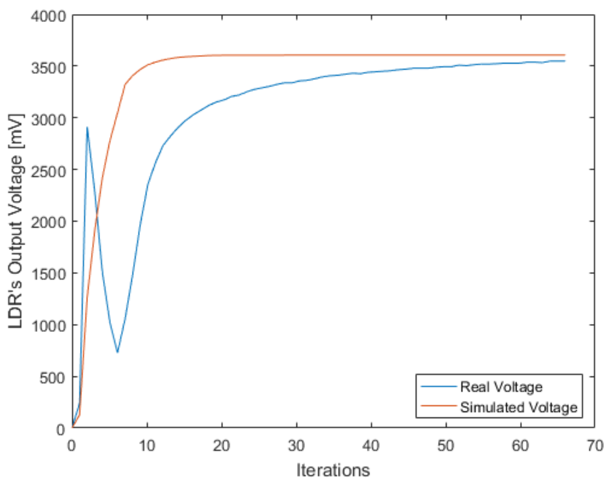

Figure 16: *Feedback system from 0 LUX to 80 LUX*

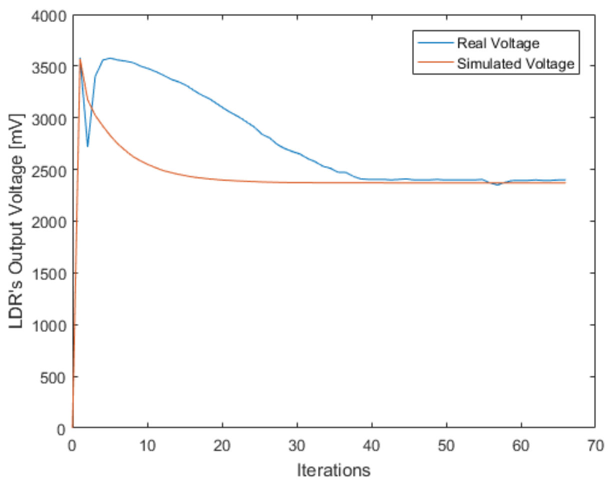

Figure 17: *Feedback system from 80 LUX to 20 LUX*

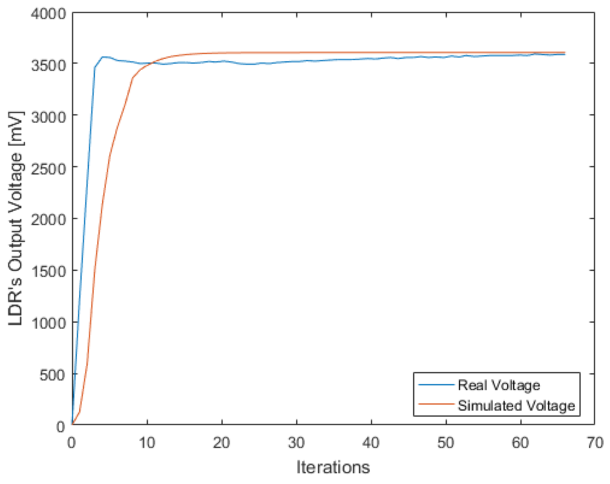

Figure 18: *Feedforward and feedback system from 0 LUX to 80 LUX*

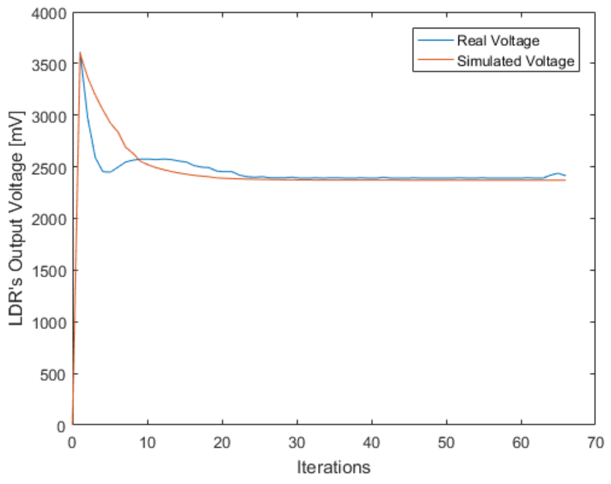

Figure 19: *Feedforward and feedback system from 80 LUX to 20 LUX*

As can be see in Figures 14 and 15, the feedforward has a very quick time response but boast a rather large static error. The feedback control system, whose responses are shown in Figures 16 and 15, has a virtually non-existent static errors, but has a very slow time response. Combining the best of both of worlds, we reach the final local controller (the union of the feedforward and feedback control systems), whose responses are shown in Figures 18 and 19. This controller boasts both the swift time response of the feedforward controller as well as the inconsiderable static error provided by the feedback control system.

5.2 Cooperative Control vs Non-cooperative Control

The next experiment made is used to show the clear advantage of using an implementation based on cooperative control versus non-cooperative control.

In order to do so, the metrics present in (11), (12) and (13) are used for both cooperative control and for non-cooperative control.The system is turned on for thirty seconds and the performance metrics for each arduino are collected. These metrics are present in Tables 4 and 5.

Table 4: *Performance metrics for the non-cooperatice control system*

	Non-Cooperative Control		
Metric	Total Energy Consumption [J]	Visibility Error [LUX]	Flicker Error [LUX/s]
A_1	1764	4.37	17.22
A_2	1610	0.01	38.46
A_3	1900	0.01	39.45
Total	5274	4.39	95.13

Table 5: *Performance metrics for the cooperative control system*

	Cooperative Control		
Metric	Total Energy Consumption [J]	Visibility Error [LUX]	Flicker error [LUX/s]
A_1	1022	2.49	9.58
A_2	1518	0.08	32.34
A_3	1326	0.05	40.18
Total	3866	2.62	82.1

As we can see, every performance metric present in the Tables are better for the cooperative control system than the non-cooperative control system. This was to be expected, namely in the total energy consumption, since the consensus algorithm computes a optimal solution that minimizes that very same metric.

6 Results and Discussion

Throughout the development of this project the group was able to develop both a robust decentralized, non-cooperative control system (the local control system) and a decentralized, cooperative control system (the addition of the consensus algorithm). The group was also able to develop a PC application to interface with the system, being able to gather valuable information about the system's performance as well as setting the different system states (for example changing the occupation state or a bound of a certain desk).

The main conclusions to be drawn from these projects are twofold. First, we can conclude that a control system that boasts both feedforward and feedback control performs better than a system that is only controlled by one of the mentioned parts. The second conclusion is that a decentralized cooperative control system performs better than a decentralized non-cooperative one. This is made clear by the values present in Tables 4 and 5, present in the Experiments section.

The group considers that the component that is deserving of more attention is the PC application developed. Since not all the functionalities were implemented (namely, the return last minute buffer and start and stop stream of a real time variable), it would be important to be able to implement said functionalities as they would mean the total completion of the project.

References

[1] `https://en.wikipedia.org/wiki/Optimal_control`

[2] `MODULE22-CONSENSUS.pdf` *in* Distributed Real Time Control System's course page 2019/2020, Alexandre Bernardino, November 29, 2019

[3] `https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/SEN-09088.pdf`

[4] `MODULE6-LIGHT-MODELS` *in* Distributed Real Time Control System's course page 2019/2020, Alexandre Bernardino

[5] `MODULE11-PID` *in* Distributed Real Time Control System's course page 2019/2020, Alexandre Bernardino

[6] `MODULE12-DIGITAL-PID` *in* Distributed Real Time Control System's course page 2019/2020, Alexandre Bernardino

[7] `MODULE9-DYNAMICAL-MODELS` *in* Distributed Real Time Control System's course page 2019/2020, Alexandre Bernardino

[8] `https://github.com/autowp/arduino-mcp2515/`

[9] `Microsoft Word - SCDTR1920-Project-V1.0` *in* Distributed Real Time Control System's course page 2019/2020, Alexandre Bernardino, João Pedro Gomes, Ricardo Ribeiro

Hardware Used

- 3 *Arduino UNO* Rev. 3 or equivalents with *ATmega328P* micro-controller
- 3 *MCP 2515* CAN bus modules
- 3 LEDs, 20mA, 3.4V
- 3 Light Dependent Resistors, *LDR GL5528*
- 3 Resistors 100 Ω
- 3 Resistors 10 KΩ
- 3 Capacitors 1 µF
- 2 120Ω CAN bus terminating shunts