TÉCNICO LISBOA

Master's degree in Electrical and Computer Engineering

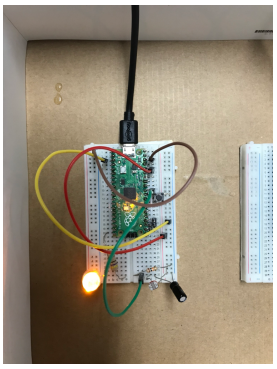
2024-2025

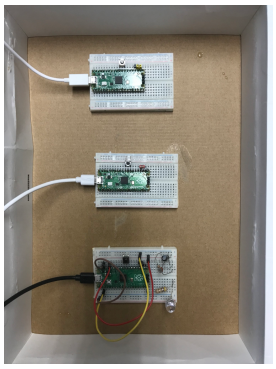
Spring Semester

Distributed Real-Time Control Systems

PROJECT GUIDE

Real-Time Cooperative Control of a Distributed Illumination System





Prepared by Prof. Alexandre Bernardino

With contributions and updates (since 2013) by Prof. José Gaspar, Prof. João Pedro Gomes, Prof. João Paulo Neto, Dr. Ricardo Ribeiro, and Dr. David Cabecinhas.

Instituto Superior Técnico

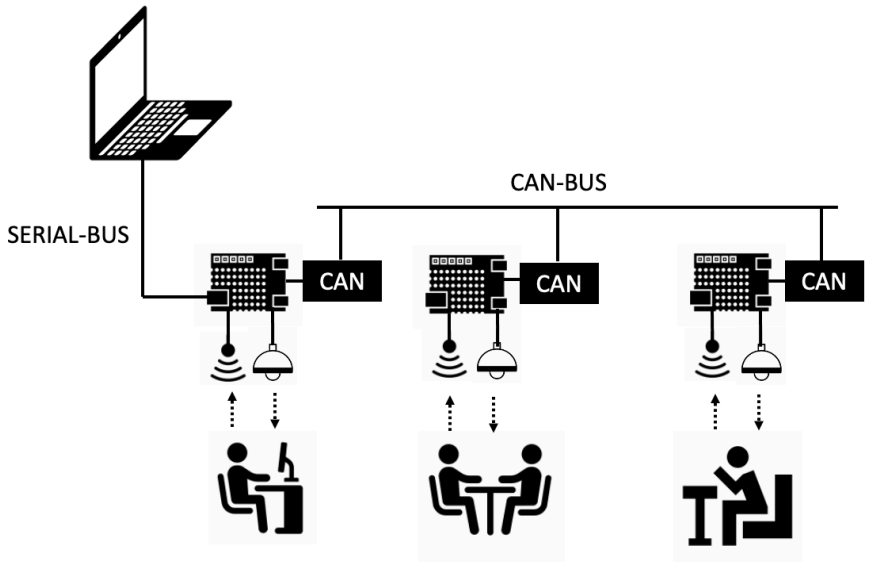
Department of Electrical and Computer Engineering

Scientific Area of Systems, Decision, and Control

Version 2.0

Abstract

The objective of this project is to design a networked real-time controller for a distributed illumination system on a small-scale model of an office space. Conceptually, each desk is served by a smart luminaire installed in the ceiling immediately on top of it. The luminaire has a light emitting source to illuminate the desk, a luminance sensor to measure the light reflected from the desk, a presence sensor to determine if the desk is occupied or free, and computational and communication elements to perform the control of the system. In the project, we will simulate the office with a small opaque cardboard box. Each luminaire consists of a breadboard with a microcontroller that drives a light-emitting diode (LED) and measures illuminance with a light-dependent resistor. All luminaires will be connected to a control network via the CAN-BUS for message exchange. The objective of the project is to control the dimming level of each LED in a coordinated way to maximise the comfort of users while minimising the energy consumption.



Laboratory Sessions

There are weekly lab sessions of 3 hours each, where students will receive guidance from the teaching staff to execute the project.

In the first week of the course, students receive and assemble individual kits with the equipment for the execution of the project that they can take home and keep until the end of the period. This will allow for greater autonomy and time of contact with the project hardware.

The remaining lab sessions are mandatory as they are part of the project's continuous evaluation process. Students must bring the equipment to demonstrate their progress, get answers to their questions, and learn best practices to execute the different phases of the project.

Equipment must be returned before the exam.

Important notes:

1. It is recommended that students use their own laptops in the lab for interfacing with the project's hardware. This facilitates switching from working at home to working in the lab and vice-versa.

2. In the first lab session bring a plastic bag to carry the take-home equipment.

3. In the second lab session bring a shoe box (or similar) to assemble the office illumination model.

Project Schedule

The project is divided into two phases. The first phase will have a duration of 4 lab sessions. The second phase will have a duration of 3 lab sessions. There are lab sessions every week.

In the first phase, each student will work individually in the construction of infrastructure of the distributed system: assembly, modelling, local control, communications. At the end of this phase, each student will provide a written report and the developed code.

In the second phase, students will work collaboratively in groups of at most three students to develop the cooperative distributed controller and a PC application to interface with the control network. Each student in the group may specialise in a particular component of the system and be evaluated for that: distributed control algorithms, networked implementation, communication protocols. A final demonstration of the project will be provided during exam preparation week. The developed code must also be delivered.

Important note:

Although the second phase of the project and final demonstration will be made in group, each student must take responsibility for a fair part of the work and there may be grade differences is large asymmetries are verified.

Project Evaluation

The final grade of the project takes into account (i) the quality of the mid-term report (including the code), (ii) the quality of the demonstration (including the code), and (iii) the continuous performance assessment during the lab sessions. It has a minimum grade of 8 for approval.

The mid-term report has a maximum of 6 pages + figures, and will be delivered at week 5.

The final demonstration will be made during the exam preparation week. The contribution of each student to the work done should be clearly stated. The grades may be individualised if the quality is inhomogeneous throughout the group.

Software, hardware schematics, and other material developed to execute the project should be submitted jointly with the mid-term report and the final demonstration, within the time limits.

The assiduity and punctuality of the student in the lab sessions is also considered for assessment purposes. Unjustified absences will be penalised.

Important Note:

The reports and the software developed must be original. All forms of plagiarism will be pursued to the full extent of IST/UL regulations and Portuguese law.

The use of LLMs is allowed only to improve language and grammar and should always be double checked by the students, which assume full responsibility for the contents of the report.

Take-home Kits per Student

- 2 RPI pico microcontrollers with GPIO headers
- 2 USB 2.0 A – micro USB cables for RPI pico
- 2 small breadboards
- 2 light-emitting Diodes 10mm 85000-100000 mcd
- 2 light-dependent resistors type PGM5659D
- 2 resistors 47 Ohm for LED driving circuit
- 2 resistors 10 KOhm for LDR reading circuit
- 2 capacitors 10 microF for LDR reading circuit
- 2 CAN-BUS Joy-It MCP2515 Drivers with terminations
- 2 wires for the CAN-BUS line
- 16 MF jumper wires for Pico to MCP2515 connections
- Other wires by request

Important notes:

1. In case of gross misuse of the equipment or violation of its operational limits, you will be requested to replace the damaged equipment.

2. You must always bring to every lab session the equipment kits since it is required for the lab work.

3. In the lab are available tool sets, multimeters, soldering irons, and other equipments that you may request to assemble and debug your system.

Project Description

1. Introduction

In this project, we consider an office-like scenario where each desk is served by a smart luminaire: a device with an illumination sensor, an LED driver with dimming, and communication abilities to coordinate the actuation with their neighbours.

The luminaires have three states: OFF (turned off, no one in the office), UNOCCUPIED (no one in the desk, low light), and OCCUPIED (desk is busy). When the luminaires are turned on, they have the job of controlling the illumination on the respective desk to a value equal to or superior to the recommended levels (either for UNOCCUPIED or OCCUPIED). Luminaires should have control algorithms to adapt to variations of the outside light during the day and to the influence of neighbour luminaires, trying to save energy. In addition, they should prevent flickering and other light variations that are uncomfortable to the user.

As a user of a desk sets its occupation state to UNOCCUPIED or OCCUPIED, the luminaire should quickly change the light intensity to the desired level. However, in response to external effects, it should change the light intensity slowly, without flicker.

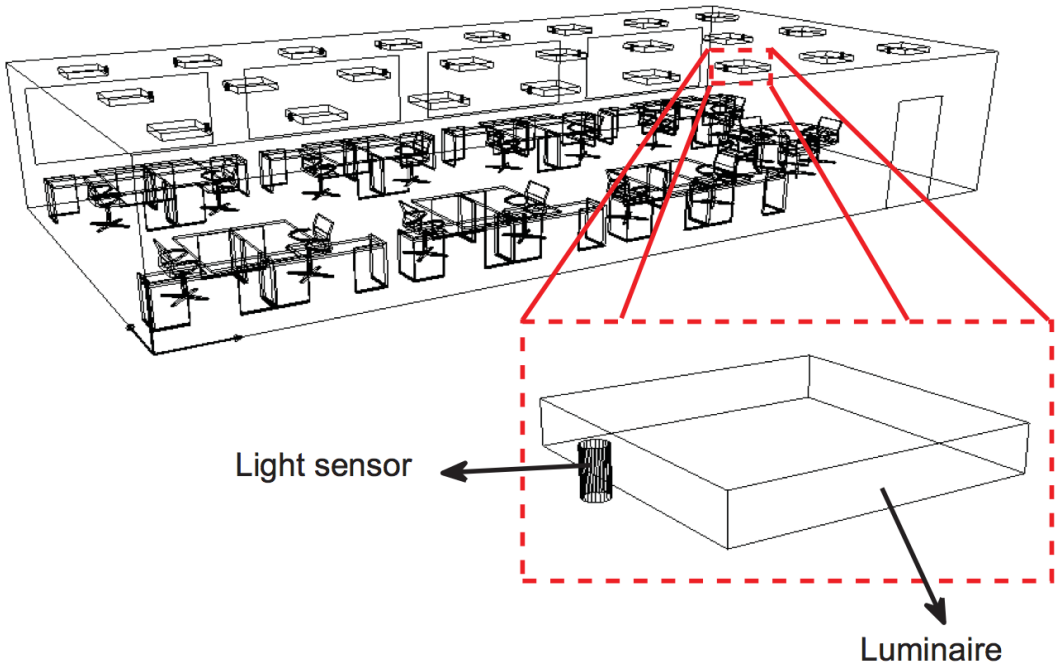


Figure 1. Scenario envisaged in the project. Each desk is served by a luminaire containing a light sensor, a presence sensor, and communication links with its neighbours. Note that the sensor measures the reflected light on the table and not the direct light from the lamp. Control of the lighting attains fixed levels of illumination at the desk plane (high for occupied desks and low for unoccupied desks) while minimising the global energy consumption and considering daylight illumination and disturbances from neighbouring luminaires.

When a luminaire changes its illuminance level, either due to commands of the user or due to adaptations to external effects, it will influence the illumination levels in the

neighbour desks. Therefore, the neighbour desks must adapt their own illumination levels, and this again affects the illumination level of the first desk. In this project, we will implement controllers to deal with this "coupling" effect. If luminaires can send messages to each other, they can coordinate their actions and potentially reach better solutions.

2. Objectives

The objective of the project is to minimise the energy consumption of the system and maximise the user comfort. Energy minimisation will be achieved by controlling the dimming level of each LED to a minimum value that satisfies the user comfort requirements. User comfort should be maximised by (i) keeping the illumination always above or equal to the lower bounds of visibility (OCCUPIED lower bound for occupied and UNOCCUPIED lower bound for unoccupied desks), (ii) by reacting quickly to user commands, and (iii) minimising the fast up-and-down variations of the illuminance (flicker). These variations may be due to noise, external disturbances, or interference caused by other luminaires in the shared space.

3. The Plant

The office illumination system, illustrated in the figure below, should be simulated inside in a reduced-scale model.

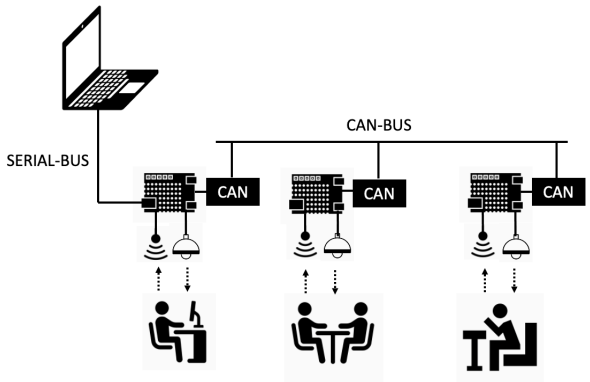


Figure 2. Overall architecture of the office illumination system to implement.

An opaque box with a cover should be used to block external illumination as much as possible. The box should be large enough to contain the 3 luminaires but not too large. If the box is too large, the intensity of the LEDs may be insufficient to properly illuminate the LDR (note that the LDR does not receive direct light from the LED but from the light reflections on the sides of the box). To improve light reflection, the interior of the box should be covered with white paper. Small openings should be made in the box to pass cables and wires. Try to insulate these openings as much as possible to prevent uncontrolled light from entering the box. A window might be cut into the box to test the system under external light, or one of the luminaires can be used to impose controlled disturbances on the system.

Attach the components to the box so they will always remain fixed. Use fixation materials that do not damage the components and allow for easy disassembly. You can use double face tape if necessary. DO NOT USE THE STICKER TAPE OF THE BREADBOARDS DIRECTLY – USE EXTERNAL FIXATION MATERIALS THAT DO NOT DAMAGE THE BREADBOARD REAR SURFACE.

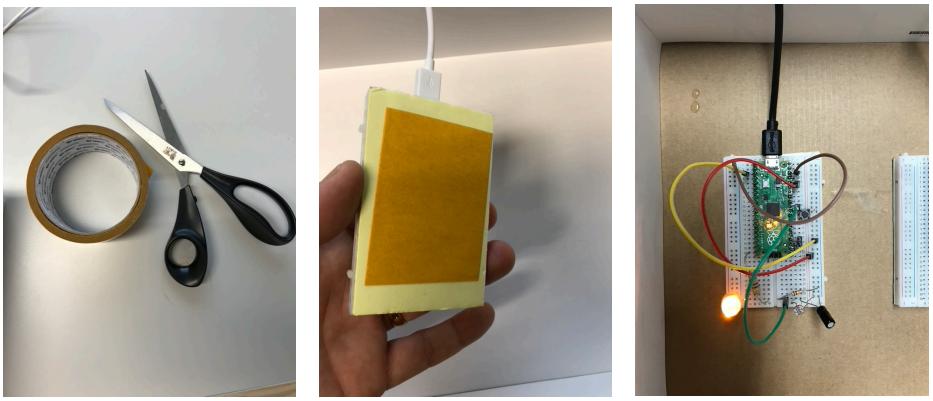


Figure 3 - Proper assembly of the components in the box is essential to prevent problems in cabling and calibration due to objects motions.

Luminaire nodes. Each luminaire will be simulated with the equipment provided. The following diagrams illustrate a possible LED driver circuit and an illuminance reading circuit.

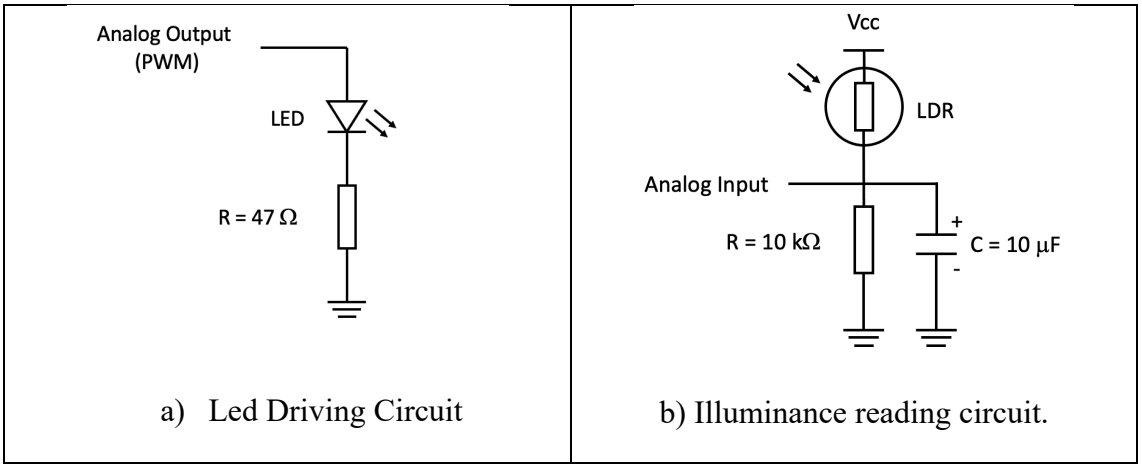


Figure 4 – Schematics for the luxmeter (a) and LED driver (b) circuits.

The dimming of the LED is implemented via PWM in one of the digital output ports. Illuminance can be measured via the LDR in a voltage divider circuit. The capacitor in the voltage divider helps reduce noise in the sensing circuit. The PWM frequency should be configured to further reduce the noise on the analogue input. Both the LED and the LDR should point 'vertically', the sensor should measure the reflected light on the desk and not the direct light from the lamp. Note that the sensing electronics is a low-pass filter, so light measurements will be delayed and smoothed with respect to their true variations.

4. First stage

4.1. Description

During the first stage of the project, the students will implement the basic infrastructure of the distribute control system: assemble the luminaires and the control network, program the microcontroller to perform local control and basic interface with the PC, and test the CAN-BUS communications among the nodes.

At this stage, the local controller is unaware of the existence of other luminaires and has the objective of keeping the illumination level as close as possible to the desired value despite external disturbances. The controller should have a fast response to changes in the desired illumination level and avoid flickering and overshoot in response to disturbances.

The software to implement in each luminaire can be decomposed into the following modules: (i) the illuminance measurement system (luxmeter), (ii) the LED actuation system (driver), (iii) the individual luminaire controller (local controller), (iv) a simple interface with a PC to receive commands and send back status data, (v) data storage and computation of performance metrics, and (vi) routines to send and receive messages in the CAN-BUS.

4.1.1. The Illuminance Measurement System (Luxmeter)

The purpose of this system, described in Fig. 3 right, is to measure the illuminance in LUX units. The LDR is a nonlinear element, i.e. its gain (ratio of the variation of the illuminance to variation of the measured voltage or current) varies with the operating point. However, its relationship to the standard illuminance unit - the LUX - is known: in the log-log scale, resistance and illuminance have an affine relationship, graphically illustrated in the following figure:

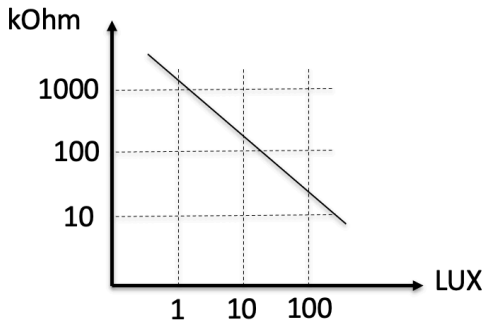


Figure 5. Typical illuminance-to-resistance characteristic of a LDR. Note that the axes are in logarithmic scales.

The equation that expresses this relationship is as follows:

$$\log_{10}(\text{LDR}) = m \log_{10}(\text{LUX}) + b \qquad (1)$$

Parameters *b* and *m* can be computed by knowing the resistance value at a certain value of illuminance (see value at 10 LUX in the datasheet) and the slope (sensitivity) of the characteristic curve (see value of the parameter γ in the datasheet).

Thus, routines for computing the illumination read by the LDR in LUX should be programmed in the microcontroller. Note that the LDR datasheet indicates a range of possible resistance values at 10 LUX. This is due to manufacturing tolerances. As we do not have a high-precision tool to measure luminance, you can consider any value inside the range specified in the data sheet.

4.1.2. The LED Actuation System

The microcontroller does not have a pure analogue output. Instead, it emulates one using a switching digital signal with pulse width modulation (PWM), whose ratio of the duration of 1 to the duration of 0 (duty cycle) is proportional to the analogue voltage required. The frequency of the PWM signal should be at least 10 times higher than the cutoff frequency of the input filter (implemented by the input capacitor), so that the switching does not have a significant influence on the illuminance measurement signal. Note that the human eye is a low-pass filter that is insensitive to flicker at frequencies above around 100Hz, so we cannot notice the LED pulsing. However, the LDR sensor is sensitive to higher frequencies and may inject noise into the reading circuit.

4.1.3. The Individual Luminaire Controller

The individual luminaire controller (local controller) should be implemented as a PID controller class in C++. Fine-tune the gains to have a quick response to step changes in reference and smooth responses to external disturbances. You can simulate changes in the reference via a user command, and changes in external illumination by opening a window in the box or using other luminaries as external sources. **Use a sampling rate of 100Hz,** as stable as possible. Since the system is non-linear, tune the PID controller so that it has a good performance both at low and high illuminances.

Note that the gains that you tune are valid for the used sampling rate. If you change the sampling rate, you will need to re-tune the gains.

Do not forget to implement adequate integrator anti-windup functions, to cope with actuator limits. Also, check that the PWM frequency for the LED is high enough to prevent unnecessary noise in LDR measurements. It may also be useful to implement a digital filter in the LDR readings to reduce noise, e.g., acquire many samples during a time interval and compute the mean or the median of the values.

4.1.4. Interfacing with PC

To read and write data from/to the microcontroller, a simple PC interface can be implemented using the “serial monitor” programme in the Arduino IDE. The students should implement a simple character-based protocol to read LDR values in LUX, set LED PWM values, set OCCUPIED and UNOCCUPIED lower bounds for the local

controller, set desk occupancy, turn on/off parts of the controller (e.g. feedforward, feedback, anti-windup), continuously stream a variable, get a batch of buffered data (e.g last minute), or perform any other operations that you may find useful for testing/debugging or reporting.

A possible list of commands is presented in Table 1 in the Appendix. You may extend the list with other commands you find useful.

4.1.5. Performance metrics

To properly validate an engineering solution, it is fundamental to define performance metrics that quantitatively express the requirements of the system. For this project, we aim to minimise the energy spent on illumination while providing comfort to users. Energy is the accumulation (integral) of instantaneous power over time. Suppose that the maximum power of a luminaire is denoted as $P_{\max}$. Then, a formula to compute the energy consumed at each desk is:

$$E = P_{\max} \sum_{k=1}^{N} d_{k-1}(t_k - t_{k-1})$$

where k is the index of the control samples, t_k is the time in seconds of the k-th sample, and d_k is the value of the led duty cycle (between 0 and 1) at the sample time t_k. The units of this metric are Joule [J].

Although energy minimisation is simple to formulate, comfort criteria are more subjective. We can consider the following rules:

a) The system should prevent periods of illumination below the minimum settings defined by an occupation state. A metric to assess this criterion can be defined as the average error between the reference illuminance (L) and the measured illuminance (l) for periods when the measured illuminance is below the reference. Let us call this quantity the average **visibility error:**

$$V = \frac{1}{N} \sum_{k=1}^{N} \max\left(0, L(t_k) - l(t_k)\right)$$

where N is the total number of samples used to compute the metric and t_k are the sampling times. The previous expression refers to a single desk. The total average error should be calculated as the sum of the average errors at each desk. Units of this metric are [LUX].

b) The system should prevent frequent ups and downs of illuminance (flickering) while the reference is at a constant value. A metric to assess this criterion can be defined as the average magnitude of the signal derivatives when it changes sign during periods of constant occupation. Let us first define the flicker at time t_k as f_k

$$f_k = \begin{cases} |d_k - d_{k-1}| + |d_{k-1} - d_{k-2}| & \text{if} \quad (d_k - d_{k-1})(d_{k-1} - d_{k-2}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

where d_k is the duty cycle at time t_k, and $|A|$ denotes the absolute value of A. Transient periods due to explicit variation of the reference should be excluded from the formula.

We can now define the average **Flicker** as:

$$F = \frac{1}{N} \sum_{k=1}^{N} f_k$$

Again, the previous expression refers to a single desk. The total average flicker should be computed as the sum of the flicker error at each desk. Units are [s^{-1}].

The computations should be done at all control cycles, i.e. with a frequency of 100Hz.

Some more commands regarding these metrics can be implemented to interface with the PC. See Table 2 in the Appendix.

4.1.6. CAN-BUS communications

In cooperative distributed control systems, each node typically takes the initiative to send messages to its neighbours and is permanently listening for incoming messages from other nodes. Messages can be directed to any node, or they can be broadcast to all nodes. Some messages can indicate a change of state that requires control actions by the neighbouring nodes, while others may be transmitted for logging purposes. The student should program routines on the microcontroller to be able to send and receive messages asynchronously between any nodes.

To uniquely identify a node in the network, you can use the RPI pico C/C++ high level API “pico_unique_id”.

4.2. Implementation Tips for the First Stage

- For the report, it is very important to collect data from your system to make plots of the different signals in the control system (references, control values, measurements) and compute metrics that show the correct operation of the system.
- You can use one of the other luminaires to inject deterministic disturbances into the system and test the ability of your controller to attenuate them.
- Note that serial communications use precious microprocessor time. Choose messages of a short size. Compute the communication times of the serial and CAN-BUS packets and verify that they can be accommodated within the available control loop period.
- You can copy-paste text from the serial monitor. If you format your data correctly, you can use the copied text to visualise the data graphically in Matlab or Excel.

Note that the Arduino IDE has a serial Plotter interface that can display data graphically if the messages are properly formatted.

4.3. Milestones for each lab session of the first stage

Although it is not mandatory to follow a strict agenda in the execution of the project, there are some recommended objectives that should be met each week to ensure the timely execution of the project. Note that objectives are quite hard to achieve during the 3 hours of the session, so some items must be prepared at home before the session, and others completed autonomously after the session.

Session 1 – Intro to Hardware and Software:

- Installation of the Arduino IDE on the PC that interfaces with the microcontrollers.
- Loading the first Arduino IDE sketch into the assembled luminaire. The first upload requires holding the BOOTSEL button before powering up the microcontroller (connecting to USB). You can use the example 'FADE' included in the rp2040 examples to see the built-in RPI pico LED fade in and out, or the example 'Dimmer' included in the communication examples to set the dimming level of the built-in led given user input in the SerialMonitor.
- Assemble the luminaires. Build the LED drive and LDR measurement circuits on the breadboard.
- Development of basic programmes to (i) repeatedly read values from the LDR and send those values to serial output; (ii) read values from the serial input and set the duty cycle of the LED to this value). See examples 'AnalogReadSerial' and 'Dimmer'.
- Format the output data so that you can see real-time plots of the variables of interest.

Session 2 – Implementation of the Office Model and LUX sensor

- Use a shoe box (or similar) to create a scaled model of an office space. It is important that the box can isolate the external light well.
- Implement a function that converts the voltage at the analogue input port to LUX (you must use the expressions of the voltage divider and the LDR log-log characteristic curve). Start with nominal values for the parameters m and b of equation (1).
- The measurements in LUX should be related linearly to the received luminous power received by the LDR. If the box is well insulated, this power is linearly related to the power emitted by the LED, which is proportional to the duty cycle. Use this fact to design experiments that allow to fine tune the parameters m and b in equation (1).

Session 3 – Local Control and User Interface:

- Local control. Implement a C++ PID controller with set-point weighting and anti-windup for one luminaire. Check how it behaves both in response to step changes in

the illuminance references and in the presence of external disturbances. Visualise plots to verify accuracy, overshoot, and oscillations (flicker).

- Verify if the same gains are good for the other luminaire. If not, make fine adjustments to correct.
- Implement C++ methods to store data in the microcontroller (last-minute buffer) and compute the performance metrics Energy, Visibility Error, and Flicker.
- Implement a user interface with the suggested commands (See Appendix I) and others that you may find useful.

Session 4 – CAN-BUS communications.

- Assemble the CAN-BUS network. Connect all luminaires to a CAN-BUS network. Do not forget to twist the cables to reduce electromagnetic noise and use the terminator jumpers in the CAN-BUS controllers at the extremities of the bus.
- Write and test a programme to send and receive simple messages between the microcontroller nodes. Demonstrate its correct operation, e.g. one node streaming to the PC the illuminance values read in the other node.
- Make sure your microcontroller code is nonblocking, using adequate concurrent programming primitives and exploiting the multiple cores of the microcontroller machine to properly deal with the several running tasks (user interface, CAN-BUS communications, local control, …).

4.4. Guidelines for the mid-term report

a) The report should describe the problem, the solutions adopted, the results obtained, and discuss the pros and cons of your design choices.
b) The report must be complete but succinct, with fewer than 6 pages of text – you may include as many figures as you wish. Avoid redundancies and overly technical content. Summarise as much as possible, but do not leave out important issues.
c) Graphics should be self-contained, i.e., fully labelled and with complete captions.
d) Take pictures of the interior and exterior of the box enclosing the luminaires. Make sure that the positions of the LED, the LDR and the emission / reflection paths are visible in the photo.
e) Show plots of the steady state characteristic of the system. Show step responses of the system under different illuminance conditions.
f) Characterise the jitter in your control system. How much does the sampling rate deviate from the desired one?
g) Illustrate the operation of the PID controller features - anti-windup, set-point weighting, etc) with plots of the time responses. In the plots always include the sensor measurement and the control signal – both are essential to characterize your controller.
h) Measure the processing time taken by the control computation, serial communications, CAN-BUS communications, and other computations.
i) Always use SI units for pertinent quantities. Check the data sheets to verify conversions from electrical to physical units.

5. Second stage

5.1. Description

During the second stage, the students will collaborate within a group of 2 or 3 students to implement a distributed cooperative control system. To interact with the control system you can use the USB connection from your computer to one of the microcontroller and use terminal or console facilities provided by your operating system or the Arduino IDE Serial Monitor.

In distributed cooperative control, all nodes cooperate to achieve a common objective. There is no central node, and no node should play a special role in the network. The nodes communicate using a control network (CAN-BUS). The nodes can take the initiative to start communications at any time. Ideally, all luminaires should have the same code and implement 'boot' mechanisms to recognise each other and be able to operate in a network with a variable number of nodes. Nodes can exchange messages whenever necessary to jointly minimise energy consumption and maximise user comfort. User comfort is defined as having a luminance on its desk always above or equal to the reference set points (visibility criteria) and low variations of high frequency noticeable to the human eye (flicker criteria).

The PC will connect to one of the luminaires via serial communications. This luminaire will operate as a hub to exchange information between the network and the PC. The hub luminaire has two main purposes: (i) collect information from all control nodes (dimming levels, occupation levels, etc) and send it to the PC; (ii) receive commands from the PC (set points, working modes, etc) and route them to the target control nodes. Extend the command-line protocols established in stage 1 according to Table 3 in Appendix.

5.2. Workpackages

We can identify the following main functionalities to implement in this second stage: (i) communications between the PC and the control network (hub function); (ii) network start-up; (iii) system calibration; (iv) distributed control algorithm.

Hub Function

Any node, when connected to a PC, should be able to route information to and from any node in the network. Let us call this functionality the Hub Function. This function adds, to the normal operation of a node, the role of connecting the control network to the PC. Communication with the PC is made via serial link, whereas communication with the other nodes is made using CAN-BUS.

Network start-up

When the system is powered up, all microcontrollers and Can-Bus drivers engage in their boot sequence. Each node must then understand which other nodes are becoming

active in the network to initialise the resources for the distributed control algorithm. This is a nontrivial process because the boot order may be different every time the system is started. A suitable networked ‘boot’ procedure should be programmed to ensure that all nodes are recognised and there are no deadlocks.

System Calibration

When a luminaire changes its actuation, it impacts neighbouring luminaires (coupling effect). To realise the global controller, it is necessary to model the effects of the intensity of a luminaire on the measured illuminance of the other(s). Because this coupling between luminaires depends on many factors, a calibration procedure should be made before the controllers actuate. For example, turn on one luminaire at each time and measure the illuminance in all other luminaires. This effect is almost linear when LUX units are used for the illuminance, and only a small number of measurements are required to model it. This method should also allow for the estimation of the current background (external) illumination.

Distributed Control Algorithm

The controller should be 'cooperative', i.e, there is no central master. This improves overall reliability because the whole system can still operate even if one node stops working. Each node will be an independent decision-making unit with a shared 'goal': minimise a global cost function (energy consumption), while satisfying minimal illuminance levels according to the occupation of the desks. Two cost functions will be considered: one with luminaires with identical power consumption, and the other with luminaires with different powers. The coefficients of the cost function for each luminaire, which are proportional to its power consumption, can be set via the PC interface. A few distributed optimisation algorithms will be given in the course lectures, and the groups will have to choose which ones can be used and discuss the pros and cons of each alternative.

5.3. List of commands to be processed by the hub node.

In addition to the list of commands defined in Part 1, the commands listed in Table 3 in the Appendix, specific for Part 2, should be implemented.

5.4. Milestones for each lab session

The milestones of the second phase can be made in parallel by the different elements of the group or made in a different order, as suggested here. The order given is the one most synchronised with the contents given in the theory lectures, but if the group wants to progress faster, the relevant information will be given per request. Again, the objectives are too challenging to complete during session time (180 min) and autonomous work will most likely be required.

Session 5 - System Wake-up and Calibration

- Develop network 'wake-up' procedures so that all nodes are aware of each other and start up properly.
- Calibrate the gains of the distributed system model. Write a programme to be deployed in all controller nodes to make them synchronously turn on and off the luminaires. This code should then be used to calibrate the self- and cross-coupling gains between the several nodes in the distributed system. This procedure should be done every time the system wakes up to prevent errors in calibration due to movement of the elements inside the box.

Session 6 – CAN-BUS Protocol and Hub Function

- Define an appropriate communication protocol to exchange messages the between nodes.
- Implement the Hub function to serve the listed commands. Write and test a microcontroller programme that routes messages between the PC and other controller nodes. Test this programme first with the Serial Monitor.

Session 7 – Distributed Controller

- Implement one of the distributed controllers given in the theory lectures using C++ classes.
- Deploy and test your code on the microcontrollers.
- Show how the controllers react to changes in desk occupation and external disturbances. Consider two cases: identical or different costs in the energy minimisation criteria.
- Compare this with the non-coordinated controller when no communications are available.

5.4. Guidelines for Presentation

- Bring to the demonstration diagrams that illustrate the important components of your systems.
- Bring plots, photos, or videos that illustrate the operation of the system (in case Murphy's law applies, you can still show the results of your project.
- Prepare to demonstrate in a quick and easy way:
    - the ability of the system to properly wake up and calibrate.
    - the ability of the system to properly control the illuminance of each desk in a cooperative way.
    - the ability of the system to compute energy saving solutions for both identical and different energy costs in luminaires.
    - the ability of the system to react properly to external illumination.
- Be prepared to show the implemented code and answer questions about the implementation.

6. Project Assessment

The grading of the project is based on several points:

1. Quality of the work done: this point is evaluated based on the implemented code, quality of the developed applications, and experimental results obtained. The self-assessment points described in Section *6.1.* indicate the evaluation criteria considered and the requirements for each level of grading.
2. Quality of the report
3. Quality of the demonstration
4. Continuous assessment

6.1. Self-Assessment

In this section are described the assessment criteria for the quality of the project implementation. These points can be used to monitor your progress and manage your expectations about your grading. Note that the quality of the report, the quality of the project demonstration and the continuous assessment in the lab classes are other items considered for the project grade.

The minimum requirements to pass the project are presented in the *Sufficient Items* (must do) section, assuming that the report, demonstration, and participation in the lab are also satisfactory. The *Good* (should do) and *Very Good* (can do) targets grant higher grades. To reach top grades, students can apply for *Excellent* on each point by discussing with the teaching staff the implementation and analysis of some other nontrivial function related to the point.

The self-assessment points and grading levels are the following.

- LUX measurement
    - Sufficient (must do): Implement a function that when invoked returns the illuminance measured in LUX.
    - Good (should do): The function takes repetitive measures to filter the noise present in the individual measurements.
    - Very good (can do): The function is implemented in a way that optimises its computational time and accuracy.
- LED driver
    - Sufficient (must do): Implement a function that, when invoked, sets the power of the led to some required value.
    - Good (should do): Implement functions that allow for specification of the LED dimming in different formats (e.g. duty cycle, percentage, PWM range, etc).
    - Very good (can do): The time range and resolution of the PWM are configured properly for the requirements of the system.
- System Identification
    - Sufficient (must do): Implement a function that computes the static gains of the system at initialisation.

    - Good (should do): Perform experiments to gain understanding of the main characteristics of the system (non-linearities, time-constants, delays, etc).
    - Very Good (can do): Implement a system simulator that can predict its future output given a sequence of inputs.
- Luminaire PID controller
    - Sufficient (must do): Implement a C++ class with functions to perform feedback/feedforward control of a luminaire that can set its illuminance to a desired value.
    - Good (should do): The controller should be robust to external disturbances and tuned to have dynamical characteristics compatible with the requirements. Perform experiments showing that your controller is improved with respect to a baseline solution.
    - Very good (can do): Perform selected experiments on different types of PID controllers and possible add-ons that illustrate the improvement of control quality.
- Can-Bus communication protocol:
    - Sufficient (must do): Implement a network protocol where one node can send messages to any other node and broadcast messages to all nodes. Characterise the communication delays involved in your implementation.
    - Good (should do): Implement error-checking mechanisms so that messages are issued if there are errors.
    - Very good (can do): Optimise the communication protocol to have the most efficient message encoding possible. The objective is to minimise communication delays and maximise scalability.
- Embedded Concurrent Programming
    - Sufficient (must do): Implement a concurrent application on the microcontroller with a control task, a Can-Bus communication task, a PC to Can-Bus message routing mechanism (hub function), and a user interaction task. The concurrent application must be of the non-blocking type, so no task can prevent other tasks from executing.
    - Good (should do): Analysis of the time spent on each task should be performed to ensure that the real-time requirements are met. All required user commands should be implemented.
    - Very good (can do): Exploit adequately the several real-time facilities of the used microcontroller to make the application efficient and concurrency safe (multiple cores, interrupts, mutual exclusion mechanisms, etc).
- Network Initialisation:
    - Sufficient (must do): Implement a boot procedure in which the different nodes recognise each other, calibrate the system gains, and synchronise to start operating the distributed control system.
    - Good (should do): Consider that the network can be composed of an arbitrary number of luminaires, not fixed *a priori*, up to the limit of the designed Can-Bus messaging protocol.
    - Very good (can do): Optimise the time of the boot and calibration procedure.
- Distributed Control:
    - Sufficient (must do): Implement and present results of at least one of the distributed solutions to the project presented in the course that require communication between control nodes.

- Good (should do): Implement and present results of one of the cooperative distributed control methods presented in the course and perform a comparative study of different configurations of the system and controller parameters.
- Very good (can do): Perform a quantitative analysis of a cooperative distributed control method with respect to other methods in different system configurations.

– ***Enjoy the project -***

Appendix – Tables with the list of commands

Table 1 - Basic luminaire commands

Command	Request	Response	Observation
Set directly the duty cycle (control value) of luminaire i.	‘u <i> <val>’	'ack' or 'err'	<val> is a number that reflects the duty cycle.
Get current duty cycle (control value) of luminaire i	'g u <i>’	‘u <i> <val>’	<val> is a number that expresses the duty cycle.
Set the illuminance reference of luminaire i	‘r <i> <val>’	'ack' or 'err'	<val> is a number that expresses the illuminance reference in LUX
Get current illuminance reference of luminaire i	'g r <i>’	‘r <i> <val>’	<val> is a number that expresses the illuminance reference in LUX.
Measure the actual illuminance (LUX sensor) of luminaire i	'g y <i>’	‘y <i> <val>’	<val> is a number that expresses the measured illuminance in LUX.
Measure the voltage level at the LDR at luminaire i	'g v <i>’	‘v <i> <val>’	<val> is a number that expresses the voltage at the LDR in the illuminance sensor.
Set the current occupancy state of desk <i>	‘o <i> <val>’	'ack' or 'err'	<val> is a Boolean flag: 0 – unoccupied, 1 – occupied.
Get the current occupancy state of desk <i>	'g o <i>’	‘o <i> <val>’	<val> is a Boolean flag: 0 – unoccupied, 1 – occupied.
Set anti-windup on/off on at desk <i>	‘a <i> <val>’	'ack' or 'err'	<val> is a Boolean flag: 0 – off, 1 – on.
Get anti-windup state of desk <i>	'g a <i>’	‘a <i> <val>’	<val> is a Boolean flag: 0 – off, 1 – on.
Set feedback control on/off on desk <i>	‘f <i> <val>’	'ack' or 'err'	<val> is a Boolean flag: 0 – off, 1 – on.
Get feedback control state of desk <i>	'g f <i>’	f <i> <val>	<val> is a Boolean flag: 0 – off, 1 – on.
Get current external illuminance of desk <i>	'g d <i>’	‘d <i> <val>’	<val> is a number that expresses the external illuminance in lux.
Get instantaneous power consumption of desk <i>	'g p <i>’	‘p <i> <val>’	<val> is a number that expresses instant power at the desk <i> in watts.
Get the elapsed time since the last restart	'g t <i>’	‘t <i> <val>’	<val> is a number that expresses the time elapsed in seconds.
Start the stream of the real-time variable <x> of desk <i>. <x> can be 'y' or 'u'.	‘s <x> <i>’	‘s <x> <i> <val> <time>’	Initiate a real-time stream of values. Every time a new sample of a certain variable is available, it is sent to the client as a string with the format indicated. <time> is an increasing timestamp in milliseconds.
Stop the stream of the real-time variable <x> of desk <i>. <x> can be 'y' or 'u'.	‘S <x> <i>’	'ack' or 'err'	Stops the real-time stream of values.
Get the last minute buffer of the variable <x> of the desk <i>. <x> can be 'y' or 'u'.	'g b <x> <i>’	‘b <x> <i> <val1>, <val2>, …<val_n>’	Values are returned in a string of comma-separated numbers. The string ends with the new-line character.

Table 2 – Luminaire Performance Commands

Command	Request	Response	Observation
Get the average energy consumption at the desk <i> since the last system restart.	'g E <i>'	"E <i> <val>'	<val> is a floating point number that expresses the accumulated energy consumption at the desk <i> in Joule.
Get the average visibility error at desk <i> since the last system restart.	'g V <i>'	'V <i> <val>'	<val> is the floating point number that expresses the accumulated visibility error in lux.
Get the average flicker error on desk <i> since the last system restart.	'g F <i>'	'F <i> <val>'	<val> is the floating point number that reflects the accumulated flicker error in s^{-1}.

Table 3 – Commands for the 2nd stage

Command	Request	Response	Observation
Get lower bound on illuminance for the occupied state at desk <i>	'g O <i>'	'O <i> <val>'	<val> is a number that expresses the lower bound of illuminance for the occupied state on the desk <i> in lux.
Set lower bound on illuminance for the occupied state at desk <i>	'O <i> <val>'	'ack' or 'err'	<val> is a number that expresses the lower bound of illuminance for the occupied state on the desk <i> in lux.
Get lower bound on illuminance for the unoccupied state at desk <i>	'g U <i>'	'U <i> <val>'	<val> is a number that expresses the lower bound of illuminance for the non-occupied state on the desk <i> in lux.
Set lower bound on illuminance for the unoccupied state at desk <i>	'U <i> <val>'	'ack' or 'err'	<val> is a number that expresses the lower bound of illuminance for the non-occupied state on the desk <i> in lux.
Get the current illuminance lower bound at the desk <i>	'g L <i>'	'L <i> <val>'	<val> is a number that expresses the lower bound of illuminance in lux.
Get the current energy cost at the desk <i>	'g C <i>'	'C <i> <val>'	<val> is a number that expresses the current cost of energy.
Set the current energy cost at the desk <i>	'C <i> <val>'	'ack' or 'err'	<val> is a number that expresses the current cost of energy.
Restart the system	'R'	'ack' or 'err'	Reset all values and recalibrate.