TÉCNICO LISBOA

Real-Time Cooperative Control of a Distributed Illumination System

Project Report

Authors:

Duarte Sá Morais (100163) duarte.morais@tecnico.ulisboa.pt

1 Description of the problem

1.1 Introduction

The project's objective is to design a decentralized and collaborative real-time control system for lighting in an open workspace. The system features a network of luminaires, each consisting of an LED, an LDR sensor, an Raspberry Pi Pico, and a CAN-BUS interface for network communication.

In its initial stage, the project concentrates on setting up a controller for each light and establishing a CAN-BUS communication framework. These actions provide a foundation for moving towards an integrated, real-time control system across the network.

The first phase of the project involves creating a serial communication protocol that connects the Raspberry Pi Pico microcontrollers to a computer interface. This protocol allows for real-time adjustments and monitoring of the system's parameters and performance.

Moving to the next phase, the previously established CAN-BUS protocol becomes operational, allowing for coordinated control among all the microcontrollers. Then, utilizing a consensus algorithm, the system intelligently balances energy use with lighting needs by considering the impact of each luminaire on its surroundings. This method aims to minimize energy consumption while ensuring adequate lighting levels, whether desks are in use or not.

1.2 Objectives

So, the primary goals of this project are the following:

- **Minimize Energy Consumption:** Achieve energy efficiency by adjusting the dimming level of each LED to the minimum necessary to satisfy comfort requirements.
- **Maximize User Comfort:** Ensure the illumination is always at or above visibility lower bounds, respond quickly to user commands and minimize fast illuminance variations (flicker).

1.3 System Design

1.3.1 Simulation Environment Setup

The implementation utilizes a reduced-scale model within an opaque box, designed to simulate an office illumination system.

This setup, covered with white paper internally for optimal light reflection, contains three luminaires and is covered to minimize external light interference. Each luminaire, simulated using provided equipment, comprises a LED driving circuit and an illuminance reading circuit (LDR).

1.3.2 Architecture

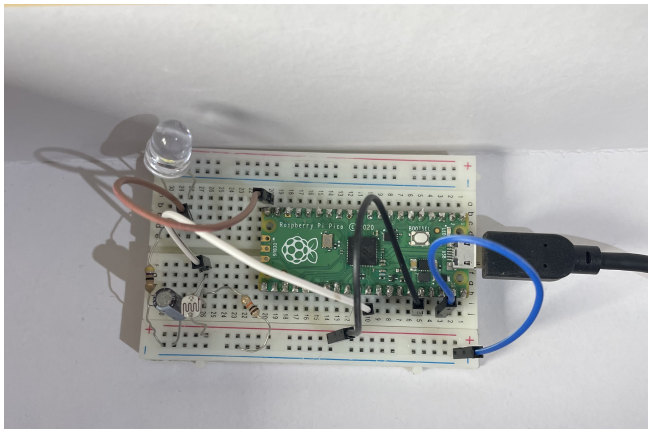

Figure 1: Raspberry Pi Pico assembly with LED and LDR

Figure 2: Opaque box covered with white paper

1.4 System Identification

1.4.1 Luxmeter

The device used to measure the intensity of light inside the box is an LDR (Light Dependent Resistor) which has a resistance that changes with the light intensity it is exposed to. The equation that expresses the relationship of the light intensity in LUX and the resistance of the LDR in ohm's is the following:

$$log_{10}(LDR) = mlog_{10}(LUX) + b \tag{1}$$

To determine the parameters m and b, the resistance of the LDR is first calculated using the following equation:

$$\dot{v}\tau(x) = -v + V_{cc}\frac{R_1}{R_1 + R_{LDR}} \tag{2}$$

Here, $R = 10k\Omega$ and $V_{cc} = 3.3V$.

Using the equation 2 to calculate the step response and assuming steady state, $\dot{v} = 0$, we obtain:

$$R_{LDR} = R \cdot \frac{V_{cc} - V_{ss}}{V_{ss}} \tag{3}$$

To determine the parameter m from the equation 3, it's crucial to verify the direct proportionality between the logarithmic values of luminance and the measured LUX, as shown in figure 3.

Data points are then extracted from LDR resistance measurements and corresponding LUX values. A linear regression is applied in *MATLAB*, resulting in m=-0.8. However, it's important to note from figure 4, that the relationship between $log_{10}(L)$ and $log_{10}(r_{LDR})$ may not be linear. Various values of m were tested, and it's concluded that m=0.8 best represents the linear behavior between LDR resistance and LUX in logarithmic scale.

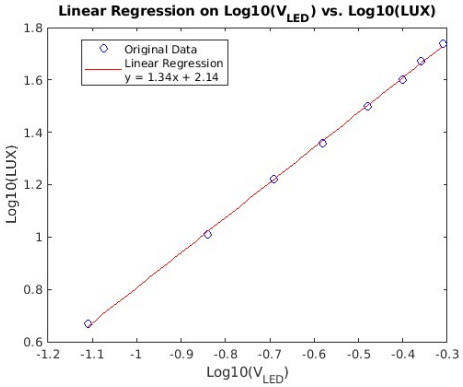

Figure 3: Linear regression for several PWM values (ranging from 0 to 4095) on logarithmic scale

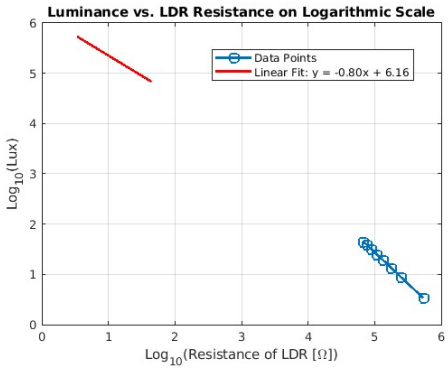

Figure 4: Linear regression for values of LDR resistance and luminance on Logarithmic Scale

The value of b is chosen using the LDR's datasheet, which states that at 10 LUX, the LDR resistance should be between $150 - 300k\Omega$. The median value of this interval, $R_{LDR} = 225k\Omega$, is selected, resulting in $b = 6.15$.

1.4.2 Capacitor

Each Raspberry Pi Pico assembly requires a capacitor because Luxmeter's ability to measure light accurately over time can be affected by rapid fluctuations in the light intensity, which can cause noise in the electrical signal.

The capacitor solves this issue by introducing delay in the circuit's response to changes in light levels. This ensures that the Luxmeter's output reflects a stable and accurate representation of the ambient light.

1.4.3 LED actuation

The LED actuation is controlled through pulse width modulation (PWM), a method where the Raspberry Pi Pico generates voltage pulses of varying duration to mimic an analog input signal.

Modulation occurs by adjusting the duty cycle, represented by u, which ranges from 0 to 4095. This value indicates the proportion of time the LED receives power at 3.3 volts, as determined by the system.

An immediate response of the LED to changes in the duty cycle is crucial for examining the system. The brightness inside the box, denoted as $X(t)$, has a direct relationship with the duty cycle, which can be explained by the equation:

$$X(t) = G \cdot u(t) + o(t) \tag{4}$$

In this context, G represents the system's gain, highlighting how the setup of the LED and the box amplifies the input from the control. Meanwhile, $o(t)$ refers to the impact of outside light on the total brightness.

To determine G, the o voltage (zero light inside the box) is first measured, followed by selecting a high value for x (in this case, 3000 PWM). The following code snippet illustrates this process:

```
void get_gain(int value){ //value is a PWM of 3000
    float x_lux;
    float o; //voltage for zero light
    float x; //voltage for 3000 PWM
    float gain;

    delay(1500);
    analogWrite(my()->LED_PIN, 0);
    delay(3000);
    o = analogRead(my()->LDR_port)*3.3/4095;
    my()->o_lux = Volt2LUX(o);
    analogWrite(my()->LED_PIN, value);
    delay(3000);
    x = analogRead(my()->LDR_port)*3.3/4095;
    x_lux = Volt2LUX(x);
    my()->gain = (x_lux - my()->o_lux) / value;
    Serial.print("Gain: "); Serial.println(my()->gain, 10);
    delay(1000);
}

```

Listing 1: Function getgain()

To calibrate G, measurements are taken when the LED is deactivated ($u = 0$) and at a high duty cycle (e.g., $u = 3000$) during system initialization. This calibration provides an accurate value of G and depends on the reflective properties of the box.

2 Individual Luminaire Controller

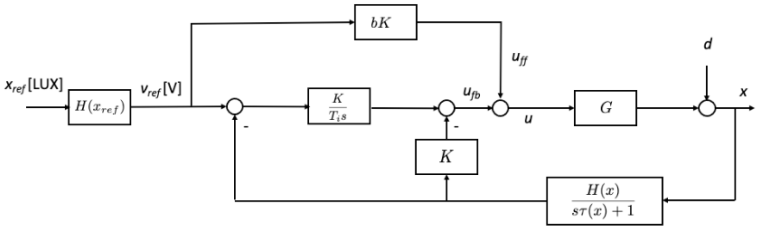

Figure 5: Control System

2.1 Definition of Control Variables for the PI Controller

For the PI controller to be defined, certain parameters are required to be established: the integral time T_i, the derivative time T_t, the b parameter $b_{controller}$, the proportional gain K, and the sampling time h.

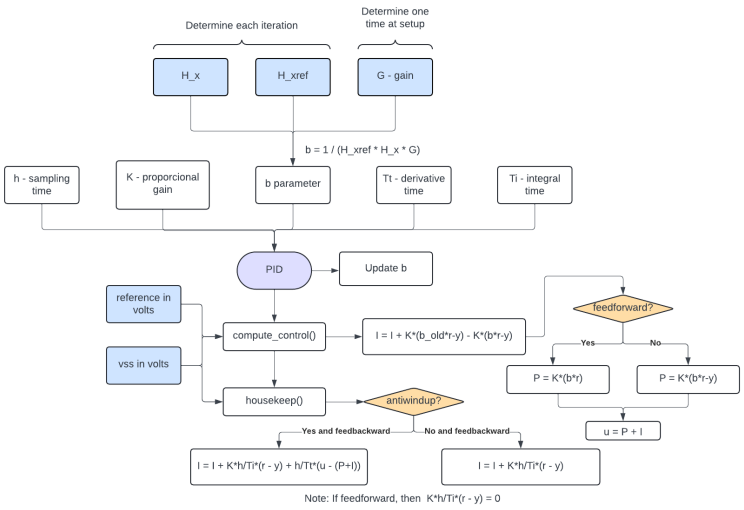

Figure 6: Fluxogram of the controller

As shown in figure 5, the transfer function from the reference illuminance x_{ref} to the actual illuminance x is characterized by:

$$\frac{X}{X_{\text{ref}}} = \frac{1}{G \cdot bK \cdot H(x_{\text{ref}})} \tag{5}$$

The determination of b and K is enabled by the following equations for $H(x_{\text{ref}})$ and $H(x)$:

$$H(x_{\text{ref}}) = \frac{v_{\text{ref}}}{X_{\text{ref}}} = \frac{V_{cc} \cdot R_1}{R_1 + 10^m \cdot \log_{10}(x_{\text{ref}}) + b} \tag{6}$$

$$H(x) = \frac{V_{ss}}{X} = \frac{V_{ss}}{10^{\frac{\log_{10}(R_1 \cdot (\frac{3.3}{V_{ss}} - 1)) - b}{m}}} \tag{7}$$

With R_1 set at $10k\Omega$ and V_{cc} at $3.3V$, and with V_{ss} representing the voltage measured by the ADC circuit, these relationships are used to compute the feedforward control loop. The calculation of $H(x_{ref})$ and $H_(x)$ is used to fine-tune b in real time, ensuring the light levels meet what is needed in the model.

2.2 K and b control variable

A analysis from the figure 5 reveals the relationship:

$$H(x_{\text{ref}}) \cdot b \cdot K \cdot G = 1 \tag{8}$$

Through the equation 8, it is shown that the product of b and K can be calculated once $H(x_{\text{ref}})$ and G are known. Thus, with a chosen value for K, the corresponding b value is determined.

This experimental approach, supplemented by real-time adjustments based on the values of $H(x_{ref})$ and $H(x)$, ensures the lighting control algorithm remains accurate under varying conditions. Here, K remains a constant factor, while b is dynamically updated in each iteration to reflect the current lighting conditions precisely.

Various K values were experimented with to observe their effects on the system. In the results section, it is shared which K value leads to reduced overshoot and a better response of the system.

2.3 Ti control variable

The integral time Ti in a PID controller is related to the integral action, which is responsible for eliminating steady-state error by integrating the error over time and applying a correction based on this cumulative error.

So, the value of Ti assumed is the value of τ in order to create a balance between steady-state accuracy and dynamic responsiveness. So the purpose is to match the controller's corrective action to the system's rate of change.

2.4 τ value

The time constant τ is key for understanding how quickly the lighting system adjusts to changes. Experimentally, τ is found by first turning off the LED to establish a baseline, then turning it on high to simulate a sudden change. The light level, detected by an LDR, is recorded over time. When the light level stabilizes, indicating the system has reached its steady state, the time taken to reach 63.2% of this final value is identified as τ.

```
void calculate_tau(int ledOff, int ledOn){
    int number_of_samples = 400;
    int i, current, my_time[number_of_samples];
    float vss, new_vss, voltage[number_of_samples], tau;

    memset(voltage, 0, sizeof(float) * number_of_samples); //clear array
    memset(my_time, 0, sizeof(int) * number_of_samples);
    analogWrite(my()->LED_PIN, ledOff); // set inicial led PWM
    delay(1000);
    vss = analogRead(my()->LDR_port);
    delay(5000);
    analogWrite(my()->LED_PIN, ledOn); // set final led PWM
    int start_time = millis();
    while(i < number_of_samples)
    {
      new_vss = analogRead(my()->LDR_port)*3.3/4095;
      if(abs(vss - new_vss)/ new_vss <= 0.001f){ break; }
      voltage[i] = new_vss;
      my_time[i] = millis() - start_time;
      vss = new_vss;
      delay(20);
      i++;
    }
    float voltage_tau = voltage[i-1] * 0.63;
    for(i = 0; i < number_of_samples; i++){
        if(voltage[i] > voltage_tau){
            tau = my_time[i];
            break;
        }
    }
    Serial.print("Tau is (s): "); Serial.println(tau / 1000, 30);
    delay(5000); //5s
}
```

Listing 2: Function calculate$_tau$()

The value of τ used was 0.19 after several experiments of initial and final value for the LED.

2.5 Experimental Results

2.5.1 Digital Filter Implementation

To mitigate the noise present in the measurements captured by the LDR, a digital filter was employed.

```
float get_adc_digital_filter(const int n_size, int delay_microseconds) {
    float readings[n_size];
    int median_index;
    float median_value;
    // Collect a set of measurements with a delay between each reading
    for (size_t i = 0; i < n_size; i++) {
        readings[i] = analogRead(my()->LDR_port);
        delayMicroseconds(delay_microseconds);
    }
    // Sort the collected measurements to find the median
    std::sort(readings, readings + n_size);
    median_index = n_size / 2;
    // Compute the median value
    if (n_size % 2 == 0) {
        median_value = (readings[median_index - 1] + readings[median_index]) / 2.0;
    } else {
        median_value = readings[median_index];
    }
    return median_value;
}
```

Listing 3: Median Filter Implementation for LDR Measurements

This function first collects a series of readings from the LDR, introducing a specified delay in microseconds between each reading to reduce the influence of transient noise. It then calculates the median of the gathered data, which provides a measure of the central tendency that is less susceptible to outliers compared to the mean. In both figure 7 and 8, time is shown on the x-axis in milliseconds, and light intensity is shown on the y-axis in LUX.

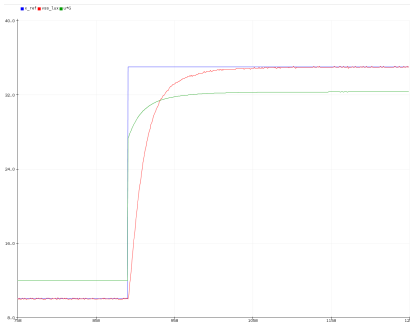

Figure 7: PI controller for $K = 500$, T_i=τ and T_t=1 with a digital filter

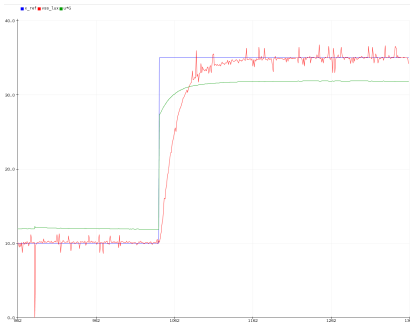

Figure 8: PI controller for $K = 500$, T_i=$\tau/10$ and T_t=1 without a digital filter

2.5.2 Results of feedback control

During the PI controller parameter tests, it was found that setting the integral time T_i to the system's time constant τ resulted in a slower response. Reducing T_i to one-tenth of τ significantly speed up the system's reaction. However, decreasing T_i to one-hundredth of τ made the system start to oscillate, which is a sign of too much correction that could lead to instability.

As for the proportional gain K, increasing it from 500 to 1000, with T_i set at $\tau/10$, introduced more overshoot in the system's reaction. After evaluating the results, the PI controller was set to a proportional gain K of 250 and an integral time Ti of $\tau/10$, where τ is 0.19. This combination was chosen because it gives a quick and stable response without too much overshoot.

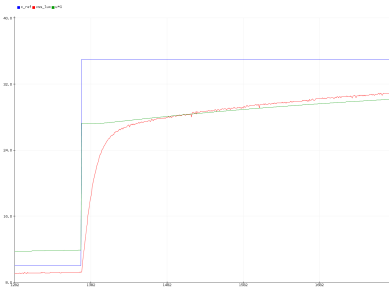

Figure 9: PI controller with $K = 500$, $T_i = \tau$ and $T_t = 1$

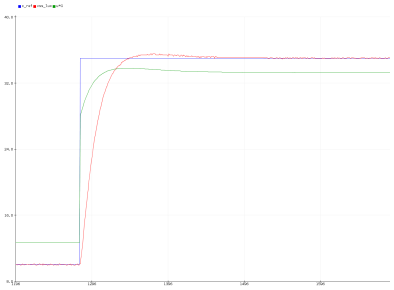

Figure 10: PI controller with $K = 500$, $T_i = \tau/10$ and $T_t = 1$

Figure 11: PI controller with $K = 500$, $T_i = \tau/100$ and $T_t = 1$

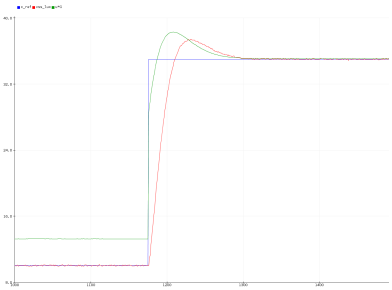

Figure 12: PI controller with $K = 1000$, $T_i = \tau/10$ and $T_t = 1$

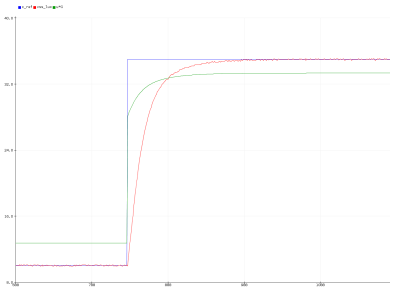

Figure 13: PI controller with $K = 20$, $T_i = \tau/100$ and $T_t = 1$

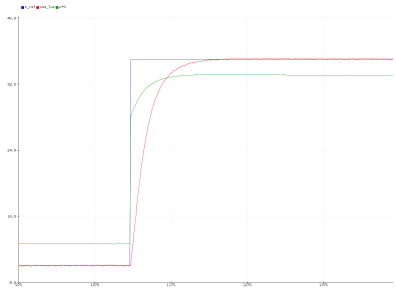

Figure 14: PI controller with $K = 250$, $T_i = \tau/10$ and $T_t = 1$

2.5.3 Response to Disturbances and Results of feedforward control

Looking at figure 15, when the system faces disturbances, the control action, shown in green, appears smoother compared to the LED's response. For the feedforward control system, figure 16 demonstrates that simply using proportional control is not sufficient to match the light intensity in LUX to the target level.

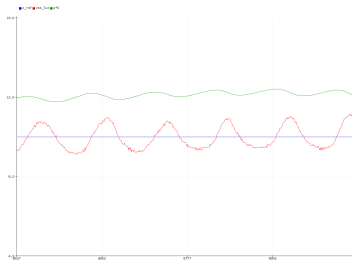

Figure 15: Feedback control reaction when a flashlight is turned on and off inside a small part of the box, causing changes

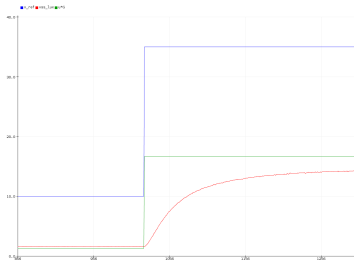

Figure 16: Representation of the feedforward control system's response. The x-axis indicates time in milliseconds

2.5.4 Anti-windup

Anti-windup measures were successfully incorporated to prevent the controller's integral term from exceeding its operational limits, which can occur when the process variable is outside the control range. This ensures that the control action remains within achievable bounds and the system can recover more quickly to setpoint changes.

The figures bellow show that, without anti-windup, the system struggles to return to the initial setpoint after being changed to a reference outside it's normal operating range. The derivative time constant, denoted as Tt, was chosen to be 1 second.

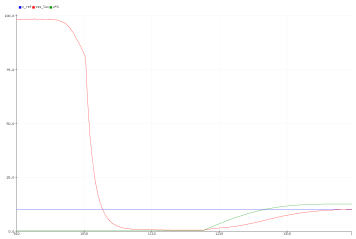

Figure 17: Controller with anti-windup

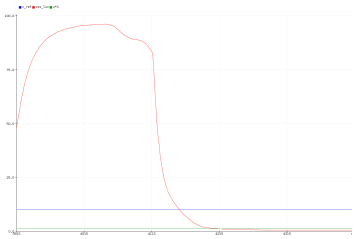

Figure 18: Controller without anti-windup

2.5.5 Steady-State System Characterization and Step Response

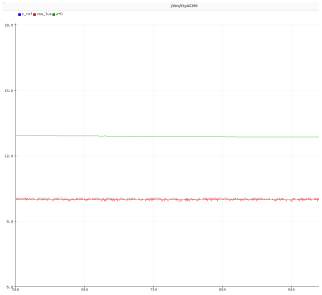

Figure 19: System steady-State

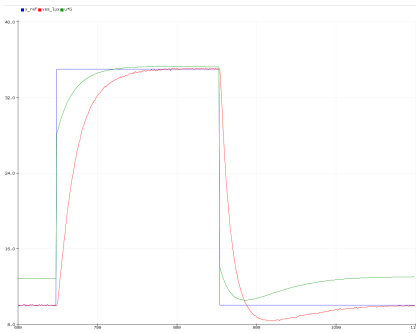

Figure 20: System step response

2.5.6 Jitter, Sampling Rate Evaluation and Metrics

To calculate the jitter, a timer was used that updates only after the control loop is executed. When there isn't much interaction from the user, the average jitter is about 1 millisecond.

The time taken to calculate the control is about 0.237 milliseconds, and the time for receiving data through serial communication is estimated at approximately 2.31 milliseconds.

In the project, the required rate for checking the system and getting new data, known as the sampling time, is set at 100 times per second (100Hz). This target is met without any noticeable jitter because the work done in each control loop cycle is completed within the time frame set for one sample.

For the performance metrics, the system calculated energy usage, visibility error, and average flicker in real time. The data shown in figure 21 are examples, actual results may differ based on the specific conditions and the duration of observation, since these values accumulate over time.

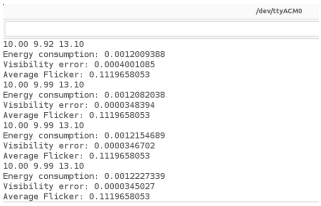

Figure 21: Real-time metrics calculation

2.5.7 Computer Interface

A serial interface was developed, allowing users to adjust settings and view metrics according to the project's requirements. This interface lets users change light settings based on desk occupancy and view system performance.

3 Communication System

3.1 CAN-BUS Network Configuration and Message Handling

The CAN-BUS, allow microcontrollers and devices to communicate with each other's applications without a host computer and it was effectively established in the project setup. It was implemented by setting up each Raspberry Pi Pico to send and receive messages at a specified bitrate of 1 Mbit/s. An interrupt service routine was configured to flag when new data is ready for processing, streamlining the handling of incoming messages. This setup was tested and found to be successful, ensuring reliable communication between the three nodes shown in the figures 22, 23 and 24.

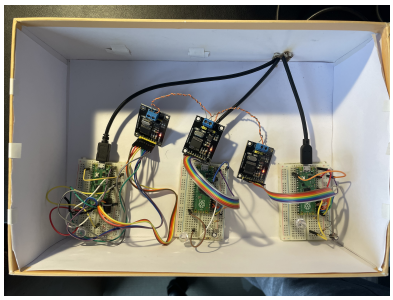

Figure 22: CAN-BUS configuration

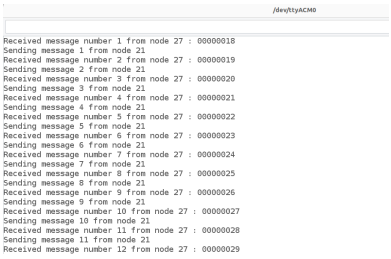

Figure 23: Sending and receiving messages with CAN-BUS protocol

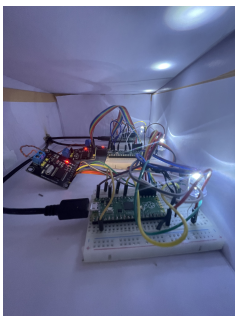

Figure 24: CAN-BUS protocol working in three Raspberry Pi's Pico

4 References

[1] Alexandre Bernardino. Distributed real-time control systems - lecture slides, 2023-2024.

[2] Alexandre Bernardino. Real-time cooperative control of a distributed illumination system - project assignment, 2023-2024.