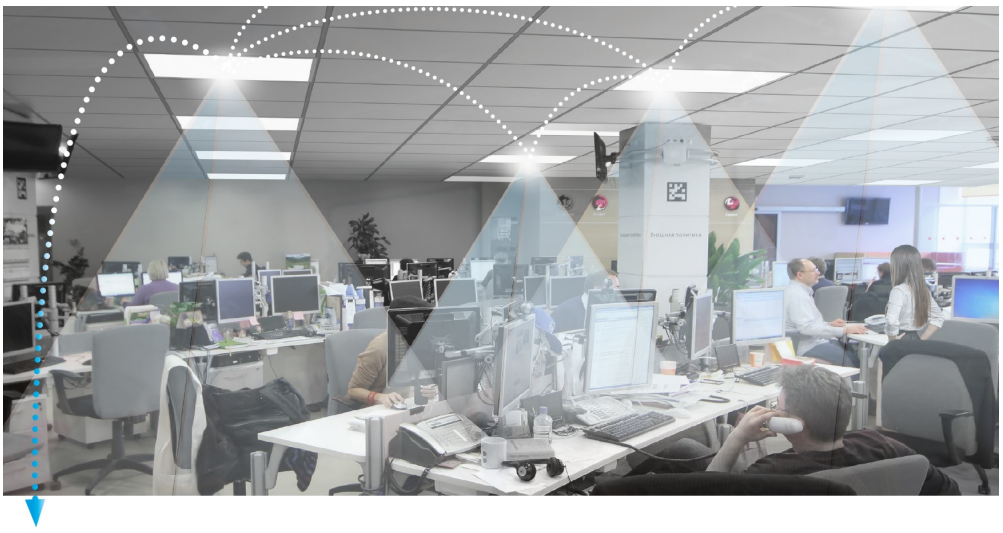
Instituto Superior Técnico,
Universidade de Lisboa

Sistemas de Controlo Distribuido em Tempo Real

Relatório Final

Prof. Alexandre José Malheiro Bernardino
Prof. Ricardo Adriano Ribeiro



Daniel Chagas	José Coelho	Miguel Fazenda
90043	90124	90146

June 29, 2021

Contents

1 Resumo

Neste relatório é proposta uma solução para um sistema de controlo distribuído em tempo real de um sistema de iluminação que permite manter a luminosidade num certo intervalo de luz desejado, em cada área de um escritório. O sistema pretende-se que seja autónomo, no sentido em que se auto-regula em função de perturbações externas, e cooperativo, no sentido em que todos os intervenientes trabalham individualmente para atingir a sua iluminação ideal mas de forma coletiva para atingir o objetivo de minimização de energia. Estas características levantam alguns desafios relacionados com a comunicação e sincronização entre intervenientes, manter a coerência do sistema e a gestão dos processos distribuídos.

Um dos objetivos principais do sistema é minimizar a energia gasta durante o seu funcionamento. De modo a medir a satisfação deste aspeto foram desenvolvidos métodos de medição da energia e potência gastas. A necessidade de bem-estar e comodidade do utilizador são avaliadas segundo as métricas de erro de visibilidade (desvio da referência desejada) e erro de *flicker* (amplitude da oscilação de luminosidade num dado nó).

A solução apresentada foi desenvolvida num modelo em escala reduzida de um escritório. Contudo, foi desenvolvido esforço no sentido de se puder replicar facilmente a solução numa dimensão real (em termos de área do escritório e número de luminárias).

2 Introdução

Numa altura em que tanto se fala e trabalha no sentido de criar uma geração mais verde e ecológica as atenções estão mais do que nunca viradas para a consciência energética. É assim pertinente o surgimento de novos sistemas capazes de gerir a iluminação de espaços, de modo a garantir a segurança e conforto dos utilizadores e a iluminação desejada, ao mesmo tempo que garanta a minimização de custos e gastos energéticos.

É nesta vertente que este projeto surge. É pretendido que o sistema proposto seja implementado, por exemplo, num escritório. Nesta abordagem foi considerado que o escritório possui várias secretárias, cada uma equipada por uma luminária. Cada luminária representa um atuador do sistema distribuído, sendo composto por um LED e um sensor de luminosidade (LDR). Possui ainda um micro-controlador, responsável pelas tarefas computacionais e um dispositivo de comunicação (MCP-2515). A este conjunto de componentes é dado o nome de nó. Ao longo da utilização do sistema, é possível definir o estado de ocupação de cada nó individualmente. O caso em que um nó é definido como ocupado corresponde à situação em que a a respetiva secretária está ocupada e por isso o nível de iluminação desejada deve ser superior a uma referência pré-definida e confortável ao trabalho a desenvolver. O caso em que um nó é definido como desocupado corresponde à situação em que a respetiva secretária está desocupada, e por isso o nível de iluminação mínima deve ser superior a uma referencia, correspondente a uma luz de presença ou de segurança. Estas iluminações de referencia foram proposta na norma Europeia EN 12464-1. Contudo, estes valores foram redimensionados para o modelo de menor dimensão utilizado no desenvolvimento deste projeto.

Tendo em conta o grande objetivo de minimizar o gasto energético é importante que o sistema implementado seja cooperativo. Deste modo, ganha-se um novo grau de liberdade no sentido em que poderá ser benéfico privilegiar o aumento de luminosidade numa das luminárias em detrimento de outro, caso o custo energético de utilizar a primeira for menor do que o custo de utilizar a segunda (por exemplo quando se utiliza na mesma sala lâmpadas LED e incandescentes). O algoritmo escolhido para este efeito foi um algoritmo do tipo *Consensus*, composto por varias iterações, em que cada em cada iteração cada nó participa na nova estimativa fornecendo uma possível solução, segundo o seu ponto de vista.

Pelos facto já mencionados é dedutível que o algoritmo implementado é adequado a um sistema de controlo distribuído (uma vez que cada nó tem um controlador com capacidade de computação

independente), em tempo real (uma vez que as ações devem ser tomadas e implementadas numa janela temporal restrita), descentralizado (uma vez que não existe nenhum nó que explicitamente tenha uma posição superior na hierarquia) e cooperativo (uma vez que, apesar de cada agente ter o seu objetivo individual, os agentes tem a liberdade para alterar o seu comportamento caso isso ajude outro agente). Este tipo de sistema de controlo apresenta algumas vantagens, nomeadamente uma melhor organização modular que facilita não só o crescimento do projeto em número de agentes como também uma gestão e manutenção de componentes menores. Contudo, surge com isto a necessidade de comunicação e sincronização entre os diversos nós, o que leva à necessidade de definir um protocolo de comunicação. No que diz respeito ao algoritmo descentralizado de *Consensus*, para a determinação das referencias de cada luminária, é de notar que tal implica um acréscimo de comunicações necessárias, e por isso um acrescimento também no tempo necessário para a obtenção da solução, quando comparado com a solução centralizada (em que um nó obtém num número bastante mais reduzido de passos a solução a atribuir a cada agente).

Para efeitos de manutenção, reparação ou despistagem de problemas no sistema foi ainda desenvolvido uma aplicação para computador com capacidade de se ligar a um dos nós, que passa deste modo a ser denominado como *hub*. Este nó é responsável por, em adição às suas tarefas habituais, fazer ainda a comunicação entre a aplicação e todos os agentes do sistema.

Ao longo deste relatório será descrito primeiramente a abordagem que foi feita de forma mais detalhada, e serão ainda analisados em detalhe alguns aspetos e ferramentas utilizadas. Será também feita uma analise dos resultados alcançados e, por fim, será dada uma análise geral da solução obtida.

3 Abordagem Proposta

3.1 Definição do Problema

O problema a solucionar envolve um sistema de controlo de iluminação distribuído, em tempo real, descentralizado e cooperativo. O sistema tem como objetivos:

- Gerir de forma independente a luminosidade em cada nó.
- Manter a luminosidade em cada nó superior ou igual a uma referência especificada
- Fazer com que o sistema seja impermeável a perturbações ou ruídos luminosos externos (por exemplo luz solar)
- Garantir a minimização dos gastos energéticos no sistema, ao mesmo tempo que se maximiza o conforto do utilizador. Estes dois objetivos devem ser avaliados segundas métricas previamente definidas.
- O sistema deverá ser passível de interação por parte do utilizador, através de uma aplicação. A comunicação sistema $\leftrightarrow$ aplicação deve ser efetuada através de um único nó (Hub), que deverá garantir a comunicação entre a aplicação e os restantes nós. Esta interação poderá também ser feita remotamente, funcionando a aplicação ligada ao nó como Servidor, e outras ligando-se a ela pela rede.

3.2 Solução Proposta e Respetivo Diagrama

Neste sentido, optou-se por utilizar as seguinte ferramentas (física e computacionais):

- Para gerir de forma independente a luminosidade em cada nó utilizou-se um Arduino Uno em cada nó. Deste modo garante-se que cada agente possui poder computacional independente e é capaz de atuar de forma imediata sobre o respetivo nó.
- De modo a garantir que a luminosidade em cada nó é escolhida tendo em conta uma referência pré-definida, e é impermeável a perturbações ou ruídos luminosos externos, foi criado um

controlador local com um termo de *feedback* e de um termo de *feedforward*.

- Para cada nó deve existir a possibilidade de definir o custo enérgico do tipo de lâmpada utilizado na respetiva luminária.
- Para garantir a minimização dos gastos energéticos foi utilizado um algoritmo do tipo *Consensus* aplicado à minimização de uma função de energia, restringida ao limites e características físicas do sistema: limites dos atuadores (luzes LED), caracteristicas físicas do sensor de luminosidade, interdependência entre os diferentes agentes.
- A aplicação foi desenvolvida em C++ utilizando a biblioteca Boost Asio para implementar a comunicação entre o computador e o *hub*, bem como para a comunicação TCP a ligação de clientes ao servidor.

As ferramentas anteriormente referidas são visíveis no esquema na figura 3.1.

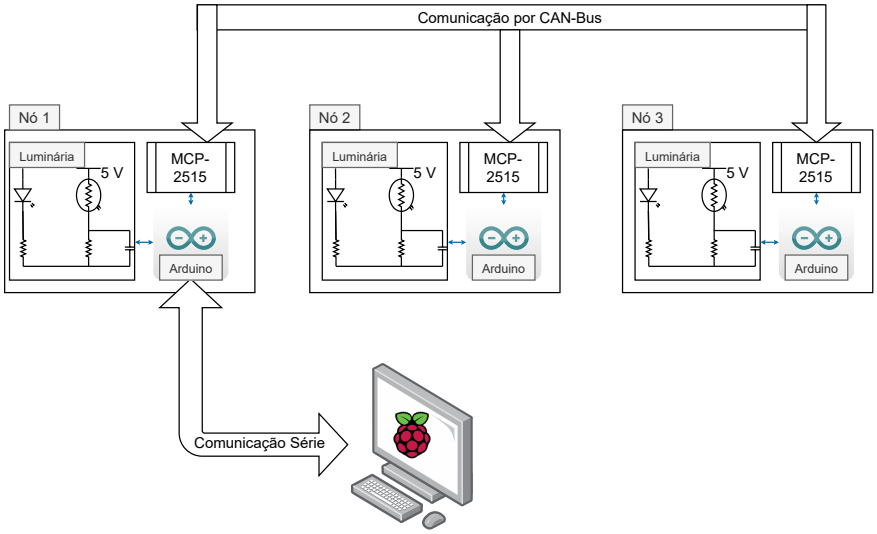


Figure 3.1: Arquitetura geral da solução proposta

É assim percetivel que os componentes interagem da seguinte forma:

- Luminária - Responsável pela interação entre o ambiente físico e o Arduino, que possui o algoritmo de controlo
- Arduino - Responsável pelo controlo de luminosidade e pela iniciativa do nó em comunicar com outros agentes do sistema, gerando e interpretando mensagens. Troca informação com a placa MCP-2515 e com a aplicação de computador (usando comunicação série).
- MCP-2515 - Responsável pela condução de toda a comunicação entre nós. Comunica com o arduino do seu nó e com as restantes placas MCP-2515. A comunicação com as outras placas utiliza o protocolo CAN-BUS.
- Rasperry Pi - Responsável pela interface utilizador <-> sistema. Corre a aplicação criada e comunica com um único nó (conectado através de comunicação série), de modo a recolher e receber a informação desejada.

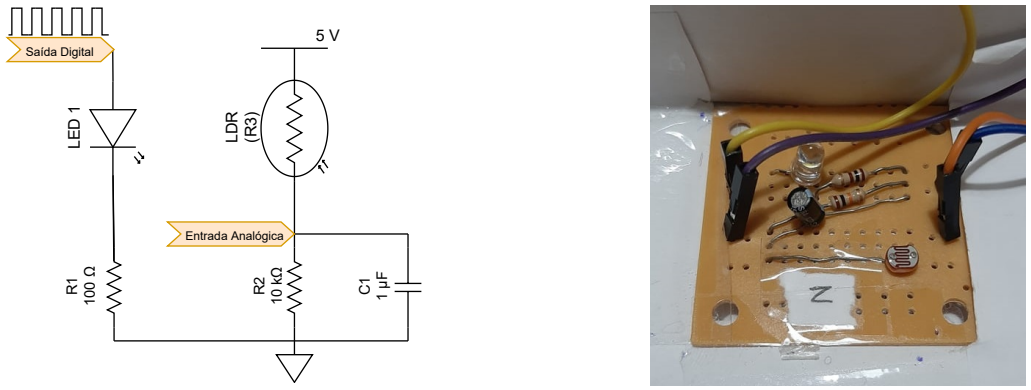
4 Abordagem experimental

4.1 Luminária e modelo utilizado para 1 nó

Como já referido, o sistema pretende-se que seja utilizado num local de *coworking*, como um escritório. Foi considerado que o espaço possui diversas secretárias e que cada secretária possui uma luminária equipada por um lâmpada e por um sensor de luminosidade. Contudo, foi utilizado no

desenvolvimento deste projeto um modelo de um escritório de pequena dimensão, com 3 luminárias. O modelo é assim composto por uma caixa paralelepipédica, como é visto nas figuras do Apêndice A.

Cada nó pode ser dividido em dois componentes gerais: a lúminária e os elementos de computação+comunicação. Na luminária, a já referida lâmpada (no modelo utilizou-se um LED) e o sensor de luminosidade (LDR) estão inseridos num circuito elétrico, como indicado na figura 4.1.



(a) Diagrama elétrico de uma luminária

(b) Exemplo de luminária utilizada no modelo

Figure 4.1: Luminárias

O paralelo entre a resistência R_2 e o condensador C_1 implementa um filtro passa alto, que tem como objetivo filtrar o ruído no circuito.

A atuação no sistema é feita através da tensão aplicada na saída digital, ou seja no LED. Idealmente o sinal deveria ser um sinal de tensão DC variável, contudo o Arduino não tem essa funcionalidade disponível. Assim, foi utilizada uma saída digital com modelação *Pulse-Width Modulation* (PWM). Deste modo, consoante o *duty-cycle* de um sinal quadrado a tensão eficaz no LED varia e este apresenta mais ou menos brilho. O brilho do LED é proporcional ao valor do *duty-cycle* da saída digital.

É pertinente que se modele a relação existente entre a atuação que se faz no espaço físico do sistema (através da luminosidade do LED) e a leitura de luminosidade que se obtém do mesmo. Contudo, é de notar que a leitura direta que se faz do sensor de luminosidade é em tensão e não em LUX. Assim, é necessário também saber a relação entre a tensão lida e o valor de luminosidade que está a incidir sobre o LDR. A figura 4.2 apresenta todo o modelo agora referido.

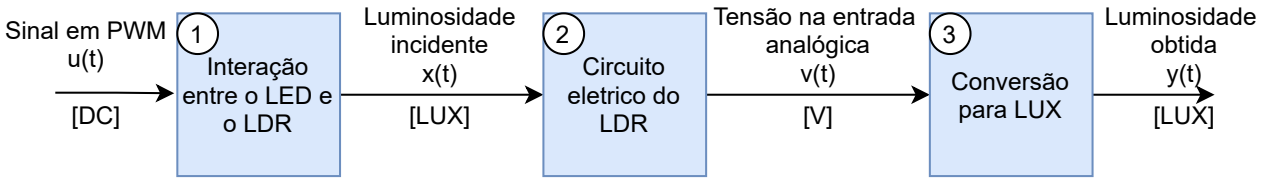


Figure 4.2: Modelo utilizado na abordagem ao problema

Para cada bloco da figura 4.2 podem ser feitas as seguintes considerações:

(1) Tendo em conta a reduzida dimensão da caixa, é possivel considerar que o tempo de propagação entre o LED e a superficie do LDR é imediata. Considera-se ainda que o tempo de propagação dos sinais nos circuitos elétricos é também nula, é assim possível concluir que a relação entre o sinal PWM e o sinal de luz incidente no LDR é um ganho de valo constante. Isto é, ignorando agora uma possivel leitura residual de luminosidade, trata-se de um sistema de ordem zero

do tipo:

$$x(t) = Gu(t). \tag{4.1}$$

(2) O circuito eléctrico de medição da tensão no LDR pode ser interpretado como um divisor de tensão entre a resistência R2 e a resistência variável R3 do LDR (confrontar figura 4.1). Considerando que a resistência R3 varia instantaneamente com a alteração na luminância e que esta é aproximadamente constante por degrau (é constantes excepto em certos instante em que varia em degrau), retira-se a seguinte equação diferencial para a tensão na entrada analógica do Arduino:

$$\dot{v}\tau(x) = -v + 5\frac{R_2}{R_2 + R_3(x)}, \tag{4.2}$$

onde $\tau(x) = C_1 \cdot \frac{R_2 R_3(x)}{R_2 + R_3(x)}$. A dependência da resistência R_3 com a luminosidade será analisa de seguida.

(3) Assume-se agora que a conversão entre a tensão lida e o valor da luminosidade obtida representa uma função não linear. isto é:

$$y(t) = f(v(t)). \tag{4.3}$$

4.2 Calibração do luxímetro e Identificação do Sistema

Como já referido, o LDR é afetado pela luminosidade incidente, variando a sua resistência $R_3(x)$. Neste sentido, algumas calibrações devem ser realizada, uma vez que cada dispositivo apresenta características físicas distintas. Recorrendo ao *datasheet* deste dispositivo sabe-se que a resistência varia com a luminosidade externa de forma inversamente proporcional. Isto é, quanto maior a luminosidade, menor será o valor de R_3. De forma a prever os valores típicos para a expressão matemática que traduz esta dependência recorre-se à figura 4.3, de onde se obtém a expressão

$$\log(R_3) = m\log(x(t)) + b \Leftrightarrow R_3 = 10^{m log_{10}(x)+b}. \tag{4.4}$$

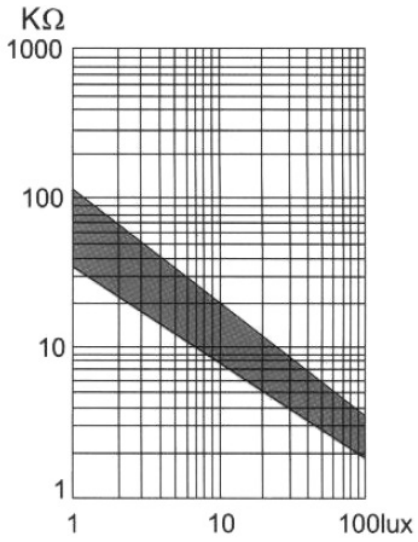


Figure 4.3: Variação típica da resistencia do LDR em função da luminosidade incidente

Assim, as variáveis do luxímetro a calibrar serão as variáveis m e b.

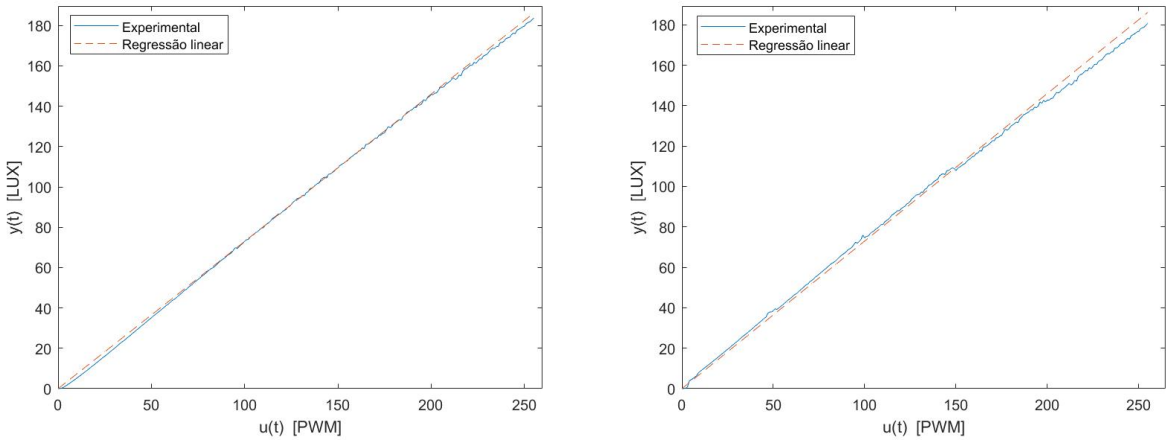
Caso o sensor de luminosidade esteja bem calibrado considera-se que, em regime estático, a leitura obtida será igual à luminosidade incidente no LDR, ou seja $y(t = \infty) = x(t = \infty) = Gu(t = \infty)$.

È assim possível que a calibração das variáveis do sensor podem ser feita de modo a atingir uma relação linear entre $y(t=\infty)$ e $u(t=\infty)$.

Caso a sensor de luminosidade esteja bem calibrado considera-se que, em regime estático, a leitura obtida será igual à luminosidade incidente no LDR, ou seja $y(t=\infty) = x(t=\infty)$. Deste modo, é possível proceder à identificação do bloco (1) recorrendo a uma conjunto de medições em regime estacionário do tipo $x(t=\infty) = y(t=\infty) = G \cdot u(t=\infty)$. A identificação do sistema passa então pela obtenção do ganho G. Para tal foi utilizado o seguinte protocolo em cada luminárias:

1. Fixar m e b com valores iniciais na expressão 4.4, partindo dos dados da figura 4.3
2. Obter a leitura de tensão para vários valores de atuação de PWM no LED
3. Converter a leitura de tensão (leitura imediata do pino analogico) para uma leitura em LUX, utilizando a expressão 4.4
4. Criar um grafico com os pontos obtidos
5. Encontrar a reta de declive G que melhor se adapta aos pontos obtidos e avaliar se os pontos se encontram suficientemente perto dessa reta
6. Repetir os pontos 2 a 5 até que os pontos de encontrem adequados à reta

É de notar que apenas a variável m acaba por influenciar a curvatura do gráfico obtido, enquanto que a variável b apenas aumenta ou diminui a amplitude destas leituras de forma proporcional (maior ou menor amplitude de valores lidos). A figura 4.4 representa o processo iterativo de calibração destes valores. A figura 4.4b representa uma fase mais inicial da calibração do que a figura 4.4a, uma vez que apresenta uma maior curvatura do gráfico que se quer linear.



(a) Iteração final da calibração de uma luminária. Foram obtidos os valores $m =$, $b =$, $G = 0.7$

(b) Exemplo de uma iteração na fase inicial da calibração

Figure 4.4: Calibração das luminárias e obtenção do ganho G

O ganho G de um nó, uma vez que está muito dependente das reflexões existem nas superfícies do escritório (como mesas, paredes, pessoas...), deverá ser calculado de forma regular no uso do sistema. Contudo, o protocolo anteriormente descrito não é o mais imediato para obter este valor. Assim, faz-se uso do conhecimento de que este tem de ser uma gráfico linear pelo que basta obter duas medições $((PWM_1, LUX_1)$ e $(PWM_2, LUX_2))$ do mesmo para que se saiba o seu declive. Isto é

$$G = \frac{|PWM_1 - PWM_2|}{|LUX_1 - LUX_2|}. \tag{4.5}$$

A necessidade de se utilizar 2 pontos distintos e não 1 vem do facto de ser possivel existir uma componente residual de luz, que tem origem numa ou várias fontes desconhecidas ao sistema, que provoca com que a reta não passe na origem do referencial.

4.3 Simulador e Obtenção da Variavel $\tau(x)$

Uma vez que se pretende utilizar um controlador local em cada nó para garantir que a luminosidade incidente no LDR é constante e impermeável a ruídos e perturbações, é necessário ter em atenção a referencia que se entrega ao controlador. Como já referido, o circuito de receção de luminosidade contem um condensador que tem a capacidade de armazenar energia, pelo que o circuito terá uma constante de tempo $\tau(x)$ que irá influenciar o tipo de resposta da leitura de tensão. Ou seja, apensar de se puder considerar que $y(t=\infty) = x(t=\infty)$, o mesmo não se pode considerar para instantes perto do alterar de luminusidade no LDR.

No caso de uma alteração em degrau desde v_i até v_f no instante t_i, a tensão no circuito LDR corresponderá a solução da equação diferencial 4.2:

$$v(t) = v_f - (v_f - v_i) \cdot e^{-\frac{t-t_i}{\tau(x_f)}}, \tag{4.6}$$

onde x_f corresponde à luminosidade final do degrau.

A obtenção prática da variável $\tau(x)$ para cada LDR foi feita, uma vez que esta expressão depende de características física do sensor. Para tal foram efectuados vários degrau de diferentes amplitudes e mediu-se o instante em que a valor da tensão ultrapassa os 63% da diferença entre a tensão inicial e final do degrau. Para uma das luminárias a curva de $\tau(x)$ obtida por este método está representada na figura 4.5.

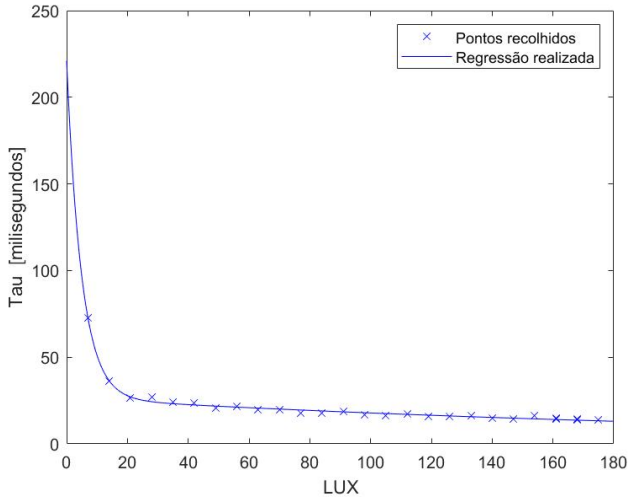


Figure 4.5: Pontos obtidos pelo processo de obtenção da curva $\tau(x)$ e respetiva regressão.

Um simulador de tensão foi criado e utilizado entre a referencia "original" e a entrada do controlador (confrontar figura 5.1), de modo a que a referencia de entrada (agora em tensão) se adequa melhor ao que de facto ocorre no sistema físico. É assim prevenido o aparecido de algum *overshoot* com origem numa referencia mais elevada do que a possibilidade de velocidade da resposta do sistema física. O simulador resolve, em cada instante a equação 4.6.

4.4 Modelo com várias luminárias

Quando vários nós coabitam no mesmo modelo existe, inevitavelmente, interações entre luzes de um nó e os sensores dos restantes nós. Assim, é pertinente obter não só o ganho entre o LED e o LDR do próprio nó ($G = K_{ii}$), mas também o ganho entre um LED e um LDR de outro nó (K_{ij}). A figura 4.6 representa a interação entre os LEDS e os sensores de luminosidade.

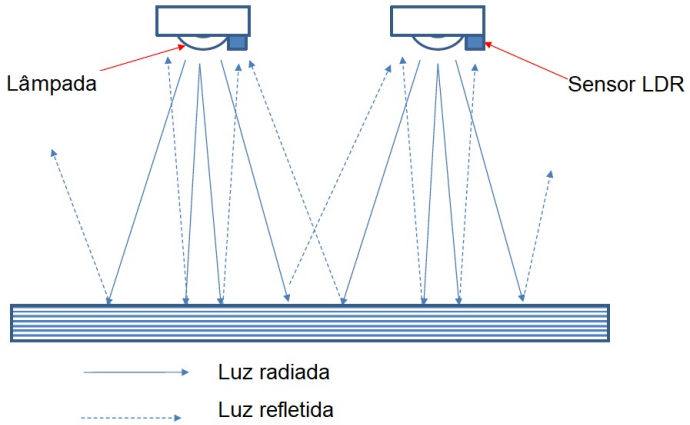


Figure 4.6: Influência cruzada das luminárias

Pela análise da figura 4.6, é percetivel que a luminosidade num certo ponto, x, é igual ao somatório da fração da luminosidade emitidas por cada fonte de luz que chega a esse ponto, $k_{ij}u(t)$, mais uma componente residual de luz, que tem origem numa ou várias fontes desconhecidas ao sistema, o. Isto é

$$x = \sum_i k_{ij}u + o_i = \sum_i k_{ij}d_i + o_i, \tag{4.7}$$

onde d_i representa o duty-cycle do LED da luminária i, no intervalo $[0,1]$.

5 Controlador local

5.1 Formulação

De modo controlar a sua luminária, cada nó tem implementado um controlador local que procura que a luminusidade medida por este tenda para uma referencia que varia consoante o seu estado (ocupado ou desocupado). Para isto, o controlador foi implementado com recurso a *feedback control*, feito por um controlador PI (*Proportional Integral*), e a *feedforward control*.

O controlo em *feedforward* permite colocar a iluminação rapidamente na referencia desejada. Para isto o Ganho estático (G) do sistema deve ser conhecido. O sinal de saída u_{ff} em PWM é dado pela expressão

$$u_{ff}(t) = \frac{lux_{ref}(t) - o}{G}, \tag{5.1}$$

em que lux_{ref} é o valor de referencia para a iluminação expresso em Lux e o é o valor de iluminação externa ao sistema, também expresso em Lux. É de notar o Ganho estático (G) do controlador i corresponde ao ganho K_{ii} da matriz dos ganhos.

O controlo em *feedback*, usando o controlador PI, permite rejeitar ruído e perturbações exteriores em regime estacionário e permite ainda o controlo preciso do sinal de saída para a referencia. Este controlador é composto por uma componente proporcional, que garante uma maior velocidade de resposta do controlador, no entanto só esta componente não eliminava completamente o erro em regime estacionário. Então, é composto também por uma componente integral, que garante que a saída do sistema tende exactamente para a referencia, eliminando completamente o erro. O sinal de controlo obtido pelo *feedback* u_{fb} é dado por

$$u_{fb}(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)d\tau, \tag{5.2}$$

em que K_p é o ganho proporcional, K_i é o ganho integral e e é o erro, em Volts, dado por

$$e(t) = r(t) - y(t), \tag{5.3}$$

em que r é o sinal de referencia e y é o sinal de saída, ambos em Volts.

O controlador local é composto pelo acoplamento do controlador de *feedforward* com o controlador de *feedback*. No entanto, é de notar que este funciona só com o *feedforward* ou só com o *feedback*.

5.2 Implementação

A implementação do controlador local está representada no diagrama de de blocos da figura 5.1.

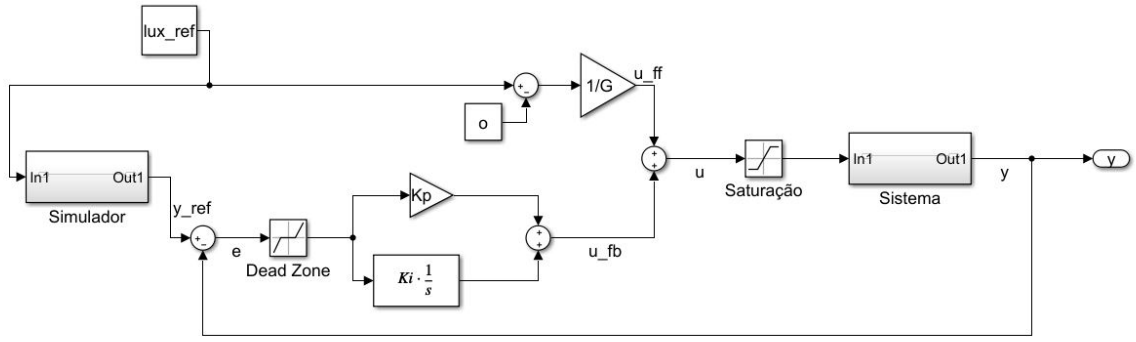


Figure 5.1: Diagrama de blocos da implementação do controlador local

De modo a calibrar o sistema foi necessário obter os ganho K_i e K_p do controlador PI. A calibração destes consistiu um processo iterativo de ajusto dos ganhos, aumentando gradualmente os mesmo, de modo a encontrar o par de valores que garantem uma resposta rápida do sistema mas sem ocorrência de overshoot.

Para alem do referido anteriormente, foi adicionada uma *deadzone* após o calculo do erro do controlo em *feedback*. Esta serve para diminuir a atuação do controlador sobre os erros de quantização e o ruído em regime estacionário, no entanto leva a um aumento do erro estático.

Foi ainda acrescentado um bloco de saturação de modo a contrariar o Windup-Effect. Isto leva a que, quando o sinal de controlo de saida do contrulador (u) sature, o termo integral do controlador seja limitado, impedindo que este (u) saia dos seus limites.

Por ultimo, foi colocado também o simulador, referido na secção anterior, antes do sinal de referencia do controlador PI. O simulador recebe à entrada o valor da referencia da iluminação e calcula y_{ref}, a tensão de referencia do controlador em *feedback*. Isto permite que à referencia prever a o sinal de saída, aproximando o erro de 0.

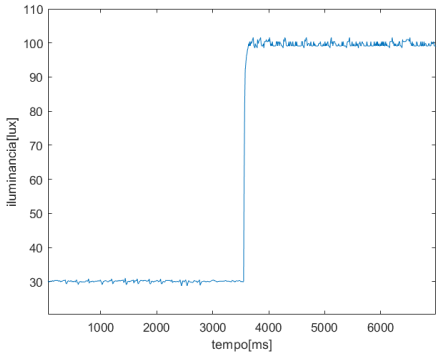
5.3 Resultados

Ao aplicar o processo iterativo de ajuste dos ganhos, para a calibração do controlador, foram obtidos os resultados na figura 5.2.

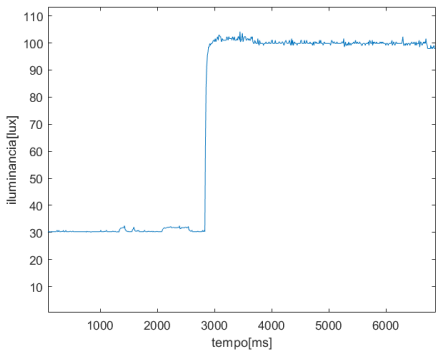
Ganhos	Valor
Kp	10
Ki	800

Figure 5.2: Ganhos K_p e K_i do controlador

Na figura 5.3 estão representadas as transições da luminância medida, do estado desocupado da luminária para ocupado, para diferentes valores de K_p e de K_i.



(a) Controlador sem *overshoot*. $K_p = 10$, $K_i = 800$



(b) Controlador com *overshoot*. $K_p = 20$, $K_i = 1500$

Figure 5.3: Variação da iluminancia para uma transição de estado desocupado para ocupado para diferentes ganho de K_p e K_i

A imagem da esquerda representa a acção do controlador local dimensionado para este projecto. É de notar que este não possui *overshoot* e que apresenta uma rápida resposta à mudança de estado entre os valores exatos de referencia dos mesmos (desocupado a $30lux$ e ocupado a $100lux$). Existe algum ruído que poderia ter sido atenuado aplicando uma *moving average* ao sinal de saída. Na imagem da direita encontra-se um exemplo de controlador com valores de K_p e K_i superiores, podendo verificar-se que este apresenta algum *overshoot* devido aos seus ganhos serem elevados. Este é um exemplo de um comportamento que o controlador não deve ter.

6 CAN-BUS

6.1 Protocolo

Como foi referido anteriormente, a troca de mensagens entre as luminárias é feita via CAN-BUS. Estas mensagens são identificadas pelo seu ID (*messageID*), necessitando este dos seguintes campos: informação sobre o tipo de mensagem que está a transmitir, o ID do nó que envia a mensagem e o destinatário da mensagem. Deste modo a comunicação será mais eficiente. Para isto foi definido um protocolo representado na figura 6.1.

Byte	Descrição	Valor
0	Tipo da Mensagem	0-255
1	ID do Nó	0-255
2	Destino da mensagem	0-255

Figure 6.1: Protocolo do ID das mensagens de comunicação por CAN para *extended frames*

De modo a implementar este protocolo recorreu-se ao uso de *extended frames* que possuem um identificador de mensagem com 29 bits (ao invés dos 11 bits das *standard frames*).

O primeiro byte está reservado para o tipo de mensagem que está a ser enviado. Isto permite que existam 255 tipos de mensagens diferentes que podem ser trocadas, garantindo flexibilidade do sistema à implementação de funcionalidades que recorrem à comunicação entre luminárias.

O segundo byte está reservado para o identificador do remetente da mensagem, enquanto que o terceiro byte está reservado para o identificador do seu destinatário. Isto permite uma grande flexidade no numero de luminárias que o sistema pode ter a trocar mensagens, havendo um máximo de 255. É de notar que não são utilizados todos os bits do identificador da mensagem, havendo 5 bits que não estão a ser utilizados.

6.2 Implementação

O Can é controlado pelo MCP2515 que possui 2 *buffers* para armazenamento de mensagens priorizado, o RXB1 e o RXB0, este ultimo tem maior prioridade. RXB1 possuí 4 filtros e uma *Mask* para filtrar as mensagens que recebe, enquanto que RXB0 tem apensa 2 filtros e uma *Mask*. Estes filtros permitem verificar se a mensagem recebida por Can é suposto ser lida ou ignorada. A *Mask* é responsável por definir quais os bits dos filtros que devem ser comparados com os respectivos bits do identificador da mensagem. Neste caso, as *Masks* de ambos os *buffers* definem que os bits a comparar com o filtro são os 8 bits do terceiro byte que guardam o destinatário da mensagem. Os filtros de ambos os *buffers* definem que as mensagens lidas são as mensagem cujo ID do destinatário corresponde ao próprio ID da luminária e também as mensagens cujo destinatário é 0, que corresponde a uma mensagem de *Broadcast* que deve ser lida por todos os nós.

De modo a não haver overflow de mensagens recebidas foi implementado em cada arduino um *buffer* para guardar mensagens recebidas, não estando assim limitado às duas mensagens guardadas por RXB0 e RXB1.

Como referido anteriormente, foram definidos vários tipos de mensagens diferentes para que quando um nó receba um tipo especifico de mensagem, consiga-a interpretar e realize uma acção especifica. Os tipos de mensagens utilizados estão definidos na tabela da figura 11.7.

ID	Descrição	Dados enviados
1	Broadcast Wakeup	-
2	Nó pronto para calibrar	Ganho ou valor residual
3	LED acesso na calibração	-
5	Executar consensus	-
6	Dados do consensus	Dutycicle
8	Pedido de execução de comando	Comando
9	Resposta de execução de comando	Valor de resposta
10	Indicação do novo Hub	-
11	Indicação de cessação do Hub	-
12	Envio dos dados regulares	Dutycicle e lux

Figure 6.2: ID dos tipos de mensagem usados na comunicação por CAN-BUS

Estes tipos de mensagem estão representados por ordem de prioridade, sendo os tipos de mensagem que tem ID 1 as mais prioritárias. O primeiro tipo é utilizado na inicialização do sistema e permite a cada nó saber que outros nós existem, sendo que cada nó envia esta mensagem quando é inicializado. O segundo e terceiro tipo estão relacionados com a calibração do sistema distribuído que vai ser explicado na próxima secção, sendo que a segunda envia o ganho calculado ou o valor residual lido para os outros nós e a terceira é enviada por uma Luminária para avisar que o seu LED acendeu.

O tipo com ID 5 e 6 são referentes ao processo de otimização do controlo distribuído, também explicado na próxima secção. Sendo a primeira enviada pelo hub para as outras luminárias a indicar que devem iniciar o *consensus* e a segunda corresponde ao envio de dados necessários para o *consensus*.

Os tipos com ID 8 e 9 correspondem ao envio de um comando por parte do Hub para outro nó executar um comando e à resposta depois de executado o comando por parte do nó.

O tipo 10 é enviado quando há um novo Hub, para todos os outros nós saberem qual o novo Hub, enquanto que o tipo 11 é enviado quando o Hub deixa de ser Hub. Por fim, o tipo 12 corresponde a um tipo de mensagem de envio de dados regulares. É feita por todos os nós exepto o hub (uma vez que este envia diretamente estes dados para o PC). Este tipo serve para as luminárias enviarem regularmente a iluminação e o *dutycicle* que está a ser obtido para o Hub, para este depois enviar para o PC.

6.3 Benchmarking

De modo a avaliar a performance do envio de mensagens foi enviada uma mensagem de um nó para outro que responde imediatamente e foi medido o tempo desde que o nó envia a mensagem até que recebe a resposta. Este processo foi iterado varias vezes e foi feita uma média com todas as medições e esta foi dividida por 2 para obter o tempo médio do envio de uma mensagem. Foi obtido um tempo médio de 1.6[ms] para o envio de uma mensagem.

7 Controlo Distribuído

7.1 Formulação

Como já referido, o sistema desenvolvido tem a característica de ser distribuído, o que faz com que cada nó tenha capacidade de computação própria, o que envolve a necessidade de comunicação de certos aspetos individuais de um nó para os restantes utilizando o protocolo CAN-BUS. Existem 2 processos importantes que são efetuados em paralelos por todos os nós: A calibração inicial e o algoritmo *consensus*.

A calibração inicial tem como objetivo a obtenção dos ganhos K_{ij} e das leituras residuais o_i e necessita de uma elevada coordenação entre todos os nós.

O grande objetivo de minimização da energia gasta por parte do sistema pode ser resolvido pela solução do problema

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & f(\mathbf{d}) = \mathbf{c}^T\mathbf{d} \\ \text{subject to} \quad & 0 \le d_i \le 100, \quad \forall_i \\ & k_i \cdot \mathbf{d} \le L_i - o_i, \quad \forall_i, \end{aligned} \tag{7.1}$$

onde $c = [c_1, ..., c_n]$ é o vetor de custos de cada luminária, $d = [d_1, ..., d_n]$ é o vetor de duty-cycle de cada luminária, $k_i = [k_{i1}, ..., k_{in}]$ é o vetor de ganhos do LDR da luminária i em relação às restantes luminárias, L_i e o_i são respetivamente a referência e a leitura residual, em LUX, do nó i.

Contudo esta formulação apresenta 2 problemas: seria necessário todos os nós possuirem informação sobre, por exemplo a referencia L_i de todos os restantes (o que não não há necessidade num sistema distribuído), e a formulação apresenta demasiadas restrições que fazem com que o problema cresça de complexidade muito rapidamente com o aumento do número de nós. De modo a solucionar estes problema a formulação foi distribuida por todos os nós, de modo a que cada nó resolve o problema utilizando apenas as 3 restrições correspondentes à sua luminária: $d_i \ge 0$, $d_i \le 100$ e $k_i \cdot \mathbf{d} \le L_i - o_i$. Por sua vez é ainda possível realizar o arranjo da função de custo de modo a retirar estas restrições do bloco *subject to* e passa-las para a função de custo. Define-se assim a função $f_i^+(\mathbf{d})$:

$$\begin{cases} f_i(\mathbf{d}) & \text{,se } 0 \le d_i \le 100 \ \& \ k_i \cdot \mathbf{d} \le L_i - o_i \\ +\infty & \text{, caso contrário} \end{cases} \tag{7.2}$$

utilizar o método dos multiplicadores de *Lagrange* de modo a retirar estas restrições do bloco *subject to* e passa-las para a função de cuto. Deste modo passa-se a ter um problema sem restrições.

O algoritmo *consensus* surge como forma de juntar todas as sugestões de solução dada por cada um dos nós, forçando a que as soluções de todos os nós convirjam para um mesmo valor, gerando assim consenso. Contudo voltamos assim a ter um problema com uma restrição: $d_i = \overline{d}_i$. A resolução deste problema é feita recorrendo ao método ADMM (alternated direction method of multipliers), que utiliza um lagrangiano aumentado (y_i) para garantir a convergência do método.

7.2 Implementação

O sistema de controlo distribuído foi implementado através de uma maquina, representada na figura 7.1

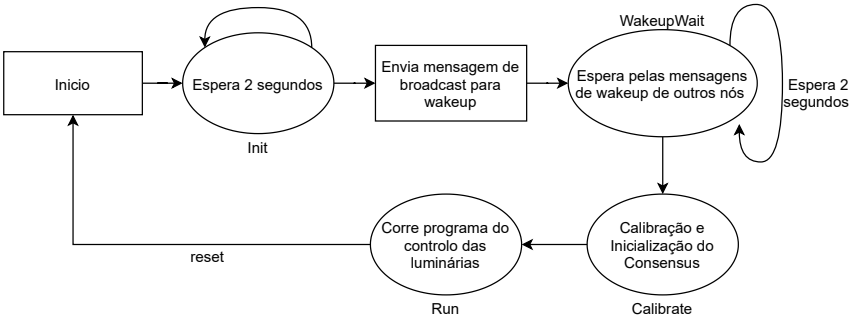


Figure 7.1: Máquina de Estados do programa de controlo distribuído

No inicio do programa cada nó começa por esperar 2 segundos para dar tempo aos outros nós para inicializarem. Depois enviam uma mensagem do tipo *broadcast wakeup*, para que os outros nós, ao lerem esta mensagem, registem a sua existência. O programa entra então num estado em que volta a esperar mais dois segundos para garantir que tem tempo que os outros nós enviem a sua mensagem de *wakeup* de modo a que todos os nós conheçam os identificadores dos outros nós que pertencem ao sistema. Depois transita para o estado em que faz a calibração seguida da inicialização

Para a implementação da calibração fez-se uso de uma máquina de estado de modo a que o programa não fique preso só na calibração, podendo assim haver troca de mensagens entre os nós. Esta máquina de estados está representada na figura 7.2.

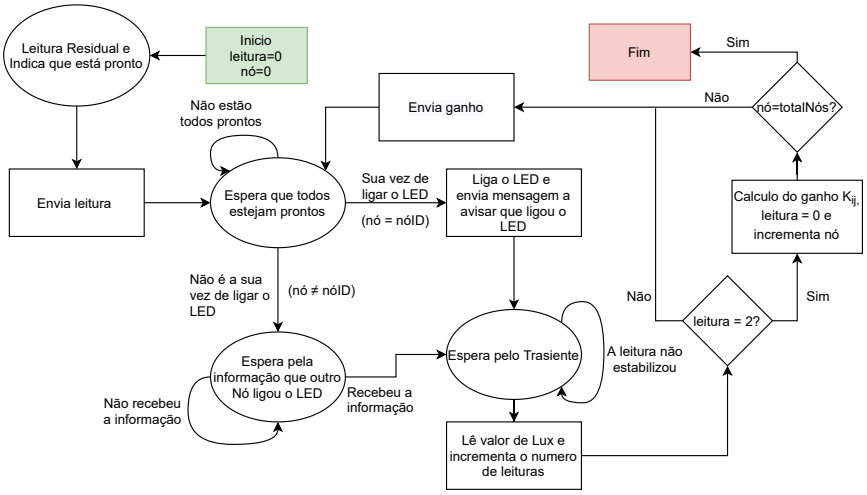


Figure 7.2: Máquina de Estados do programa da Calibração

Resumindo, o programa da calibração começa por fazer a leitura da luz residual e envia uma mensagem aos outros nós com a sua leitura e a avisar que está pronto para continuar. De seguida, muda para um estado em que espera que os outros nós estejam prontos. Quando todos os nós estiverem prontos cada nó verifica se é a sua vez de ligar o LED. Se for a sua vez, o nó liga o LED com um valor de PWM baixo e envia uma mensagem para os restantes a avisar que ligou o LED. Caso não seja a sua vez, o nó muda para um estado em que espera pela mensagem de um outro nó a dizer que ligou o LED. Depois, tanto se for como se não for a sua vez de ligar o LED, o nó muda para um estado em que espera por um trasiente, de modo a que a leitura não seja feita durante o

período de carga do condensador. De seguida é feita a leitura da iluminância, e é incrementado no número de leituras. Caso não seja a segunda leitura o nó envia uma mensagem a dizer que está pronto e volta a repetir o processo para o mesmo nó que ligou a luz, mas agora para um pwm mais alto. Se o número de leituras for 2, é calculado o ganho K_{ij}, em que i é o ID do nó que ligou o LED e j é o ID do próprio LED, depois é enviado o ganho calculado e verifica-se se já todos os nós foram percorridos de modo a ver se a calibração termine. Se não for para terminar a calibração, o numero de leituras é colocado a 0 e muda o nó que é suposto ligar o LED.

É de notar que cada nó calcula uma coluna da matriz dos ganhos e que para calcular o ganho é assumido que o gráfico $lux(pwm)$ é linear e não passa na origem e o ganho corresponde ao declive do mesmo, então são necessárias no mínimo 2 leituras para calcular o ganho. Terminada a calibração, é iniciado o algoritmo *consensus*, que funciona em paralelo com esta máquina de estados, e transita para o estado de controlo da lúminaria que é feito pelo controlador local descrito na secção 5. O sistema só sai deste estado quando o computador envia um comando de *reset*, fazendo o programa voltar ao estado inicial.

7.3 Algoritmo *Consensus* e ADMM

O algoritmo *conensus* ADMM foi implementado recorrendo a uma máquina de estados com os seguintes passos representada nas imagens da figura 11.5, em anexo. As seis zonas de procura referidas na figura 11.5(b) são:

1. Zona interior da região das restrições
2. Fronteira inferior de atuação do LED $L_i = 0$
3. Fronteira superior de atuação do LED $L_i = 100$
4. Fronteira da referencia escolhida pelo utilizador $k_i \cdot \mathbf{d} \leq L_i - o_i$
5. Intercepção da zona 2 e 4
6. Intercepção da zona 3 e 4

A função a minimizar é a função $f_i^+(\mathbf{d}) + y_i^T(\mathbf{d} - \overline{d}) + \frac{\rho}{2}||\mathbf{d} - \overline{d}||_2^2$.

7.4 Resultados

Como resultado dos processos agora referidos apresenta-se a figura 7.4, que apresenta o andamento do algoritmo de obtenção das referência otimas para o caso em que todas as luminárias devem apresentar luminosidade superior a 30 LUX e $c_1 = 1$, $c_1 = 10$, $c_1 = 1$. A figura 7.3 apresenta o resultado do processo de calibração típico para a caixa fechada, no o modelo utilizado.

Residual:

0.00000000 0.00000000 0.00000000

Gains:

1.58551621	0.30206861	0.44321336
0.11784492	1.81627998	0.08356681
0.30116596	0.16152157	1.19797658

Figure 7.3: Matriz dos ganhos e vetor de leituras residuais obtidos numa das experiências com a caixa fechada

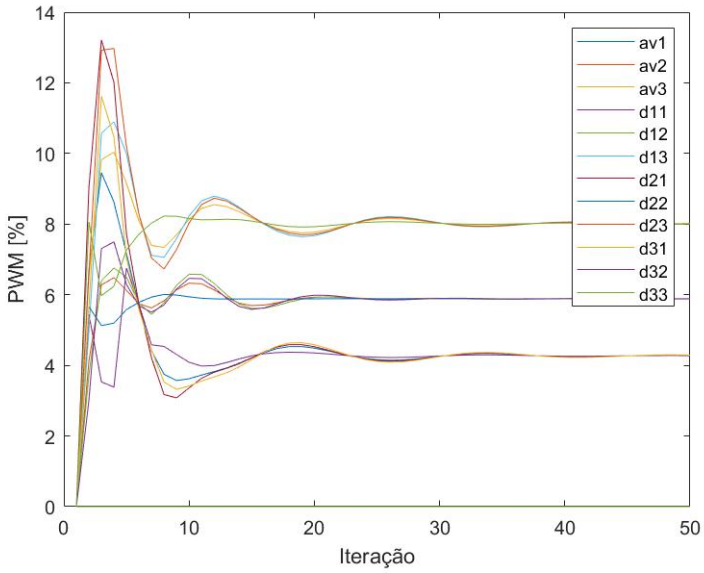


Figure 7.4: Andamento das soluções de cada nó, em cada iteração

Apesar dos gastos energéticos serem facilmente avaliados, por serem um grandeza quantificavel, o conforto do utilizador já não. Neste sentido foram criadas duas métricas que avaliam o conforto visual do utilizador e 2 métricas que avaliam a energia e potencia gastas:

- Erro de visibilidade - Como se sabe o sistema tem deve manter a luminosidade superior a um referência especificada pelo utilizador. Assim, não é desejável que a luminusidade lida pelo LDR se seja inferior a esta referência. Define-se assim a metrica de erro de visibilidade:

$$V = \frac{1}{N}\sum_{i=1}^{N} max(0, L(t_i) - l(t_i)) \quad [LUX]. \tag{7.3}$$

  Onde $L(t_i)$ corresponde à referência do nó no instante t_i e $l(t_i)$ é a luminosidade realmente medida pelo LDR no mesmo instante. Esta métrica pode ser avaliada para um nó ou para todos em simultâneo (somando o valor obtido para cada nó). Para o sistema implementado, esta métrica apresenta valores típicos nas casa das décimas para os nós 1 e 3 (≈ 0.2), enquanto que chega à ordem das unidades para o nó 2.

- Erro de flicker - Num ambiente de trabalho é pretendido que a luminosidade seja não só a adequada, mas constante, evitando o piscar das luzes. De modo a medir o quando as luzes deste sistema piscam de modo indesejado foi criada a métrica do erro de flicker:

$$F = \frac{1}{N}\sum_{i=1}^{N} f_i \quad [LUX \cdot s^{-1}] \tag{7.4}$$

$$f_i = \begin{cases} (|l_i - l_{i-1}| + |l_i - l_{i-1}|)/(2T_s) & \textbf{, se} (l_i - l_{i-1}) \times (l_{i-1} - l_{i-2}) < 0 \\ 0 & \textbf{caso contrário} \end{cases} \tag{7.5}$$

  Onde T_s é o tempo entre 2 amostras seguidas de luminosidade. A métrica é assim uma média móvel do valor médio da alteração de luminosidade medida. Para o sistema implementado, esta métrica apresenta valores típicos nas casa das unidades para os nós 1 e 3 (≈ 20), enquanto que chega à ordem das centenas para o nó 2 (≈ 400).

- Energia e Potência - Com estas métricas é possível analisar ao longo do tempo os gastos do sistema e avaliar a performance do algoritmo de minimização de energia.

8 Comunicações Série

A comunicação entre o hub e o PC é feita, como já foi mencionado, por comunicação série. Esta é feita a uma baud-rate de 1000000 baud.

8.1 Protocolo

8.1.1 Protocolo Arduino -> PC

Foi definido um protocolo para as mensagens enviadas tanto do PC para o Arduino, como do Arduino para o PC via porta Serie. Este protocolo foi dividido em 2 partes: as mensagens enviadas do Arduino para o PC, e as mensagens enviadas no sentido oposto.

Mensagem "PC Discovery"

Byte	Descrição	Valor/Tipo
0	Sync byte	255
1	"PC Discovery"	'D'

Mensagem "List of node IDs"

Byte	Descrição	Valor/Tipo
0	Sync byte	255
1	"List of node IDs"	'L'
2	Number of nodes	1-255
3	1st node ID	0-255
...		
n+2	nth node ID	0-255

Mensagem "Response"

Byte	Descrição	Valor/Tipo
0	Sync byte	255
1	"Response"	'R'
2	Value	32-bit integer or float
3		
4		
5		

Mensagem "Frequent data"

Byte	Descrição	Valor/Tipo
0	Sync byte	255
1	"Frequent data"	'F'
2	Node ID	0-255
3	Illuminance	unsigned short int
4		
5	PWM	0-255

Figure 8.1: Protocolo das mensagens enviadas pelo Arduino para o PC

Na figura 8.1 está representado o protocolo Arduino -> PC, onde existem 4 mensagens em que cada uma tem um numero de bytes, e o que cada um deles representa. Todas têm em comum o primeiro byte, com valor 255, que é um byte de sincronismo, ou seja, permite ao PC saber que ao receber um byte de valor 255, os seguintes bytes recebidos pela UART representam uma mensagem. O segundo byte permite ao computador perceber qual o tipo de mensagem que está a ser recebida.

- A mensagem "PC Discovery" é enviada por cada Arduino de 2 em 2 segundos. Quando o PC recebe esta mensagem, responde ao Arduino com outra mensagem (cuja implementação ainda será discutida neste capítulo), de modo ao Arduino perceber se está ou não ligado a um PC, consoante receba a resposta do PC. Quando não estava ligado a um computador e passa a receber resposta a esta mensagem, esse Arduino para a ser o *Hub Node*, quando deixa de receber resposta mensagem enviada periodicamente, deixa de ser o *Hub Node*. A ocorrência destes 2 eventos é comunicada aos outros nós do sistema via CAN Bus.
- A mensagem "List of node IDs" contem a lista de IDs dos nós, para que o PC saiba que nós existem, de modo a validar os comandos inseridos pelo utilizador. É enviada logo que um Arduino passa a ser *Hub Node*, ou seja, que tenha recebido resposta à mensagem "PC Discovery", ou quando um novo nó é inserido no sistema enquanto o Arduino esteja ligado ao PC, de modo a que o computador tenha sempre a lista atualizada de nós existentes. Esta mensagem contem o numero de nós (n), e os n bytes seguintes contêm os IDs dos nós presentes.
- A mensagem "Response" trata-se da resposta a um comando enviado pelo PC. O valor da resposta é enviado em 4 bytes, que regra geral é um valor do tipo *float*, mas pode ser um

inteiro, por exemplo para enviar um valor booleano (0 ou 1).

- Por fim, o propósito da mensagem "Frequent data" é enviar dados cujo envio tem de acontecer frequentemente, e não só quando o utilizador os pede (com a execução de um comando). Esta mensagem contem o ID do nó que enviou os dados ao hub(ou o ID do prório hub), o byte 3 e 4 contêm a iluminância medida pelo nó, e o byte 5 o valor PWM da luminária (0 a 255 para indicar 0 a 100%). A iluminância é enviada como um inteiro de 2 bytes (unsigned short int).

Um aspeto importante a referir é que a iluminância é enviada num inteiro de 2 bytes na mensagem "Frequent data", enquanto que na resposta ao comando "g I <nodeID>" (obter a Iluminância de um nó), a mensagem "Response" envia em 4 bytes. Foi implementado desta forma porque a mensagem "Frequent data" é enviada com grande frequência ($100 \times n$ por segundo, sendo n o numero de nós), logo é importante torna-la o mais curta possível para uma transmissão rápida, em que duas casas décimais são suficientes. A conversão é feita multiplicando o valor por 100 de virgula flutuante. Um *unsigned short int* pode representar numeros no intervalo [0, 65535]. Sendo que os 2 digitos menos significativos representam as casas décimais, é possível enviar iluminâncias no intervalo [0; 655,35] lux. Pelo contrário, como a resposta a um comando é algo que acontece esporadicamente, pode ser enviado o valor em *float*, tal como é guardado no Arduino, não fazendo sentido tentar evitar enviar mais 2 bytes.

8.1.2 Protocolo PC -> Arduino

Commando

Byte	Descrição	Valor/Tipo
0	Sync byte	255
1	Cmd	char
2	Destinatário	0-255
3	Value	32-bit integer or float
4		
5		
6		

Commando "get"

Byte	Descrição	Valor/Tipo
0	Sync byte	255
1	Cmd	char
2	Destinatário	0-255
3	Get command type	char
4	-	-
5		
6		

Figure 8.2: Protocolo das mensagens enviadas pelo PC para o Arduino

Todas as mensagens enviadas pelo PC ao Arduino são enviadas no formato representado na figura 8.2, que são descodificadas para uma estrutura de dados que representa um comando. Os comandos são enviados da mesma forma, mas ou são interpretados como os últimos 4 bytes a representar um valor, ou só o byte 3 a representar uma letra.

Quando é enviado um comando do tipo "get" (g <tipo_de_valor> <nodeID>)

- cmd='g'
- destinatário=<nodeID>
- "get command type" (byte 3) vai conter o caráter <tipo_de_valor>.

Quando é enviado um comando do tipo "set" (<tipo_de_valor> <nodeId> <val>)

- cmd='<tipo_de_valor>' ('O', 'U', 'c' ou 'o'),
- destinatário=<nodeID>
- value (bytes 3 a 6) vai conter o valor a ser definido. Será um *float* para todos exceto para o 'o' (definir a ocupância), será um inteiro, que só pode assumir os valores 0 e 1.

O valor cmd pode ainda assumir o valor 'r', quando é executado o comando para reiniciar todos os nós, e 'D', que é a resposta à mensagem "PC Discovery" enviada pelo Arduino. O destinatário pode assumir o valor '0' que corresponde a executar que afete todos os nós, como os comandos de total, ou o reset, que deve ser feito por todos os nós.

Algumas partes da estrutura são inutilizadas para algumas mensagens (exceto os bytes 0 e 1). Esta não é a forma mais eficiente de enviar mensagens, no entanto, como são enviadas esporadicamente, entre otimizar a comunicação PC->Arduino e ou tornar o código mais complexo, ou um código mais simples mas menos otimizado, optou-se pela segunda opção.

8.2 Implementação no Arduino

A função *readSerial* é corrida em cada ciclo do *loop* principal do Arduino, é verificado se o nó recebeu um byte de sincronismo(valor 255), se não, verifica se algum dos bytes seguintes o é. Se for, o arduino lê os 6 bytes seguintes e converte-os para a estrutura de dados que representa um comando.

A funcionalidade dos comandos de total ("g <tipo_de_valor> T"), que obtêm o total de um valor de todas os nós do sistema foi implementado verificando se o destinatário está a 0, caso seja esse o valor, o *hub* executa o comando, e envia um broadcast via CAN Bus para que cada nó execute o comando. Quando forem recebidas todas as respostas, é enviada a resposta pela ligação série.

8.3 Benchmarking

Para fazer um *benchmark* da performance da comunicação entre o computador e o Arduino, foi registado na tabela 8.1 o tempo desde o envio de um comando até à receção da resposta. Para tal, foi executado o comando "g I <nodeId>", ou seja, o comando que obtem a iluminância de um nó. Os resultados foram divididos em 2: tempo para um comando a ser executado no nó hub que é o hub, que representa o tempo da comunicação série e o overhead da execução do comando no nó, e também o tempo para um comando a ser executado num outro nó, que inclui também o tempo de comunicação pelo CAN-Bus.

Nó	Tempo médio (6 amostras)
Hub node (g I <hubID>)	6.07 ms
Outro nó (g I <outroID>)	13.12 ms

Table 8.1: Tempo médio de execução de um comando

9 Aplicação PC Cliente-Servidor

A aplicação do PC foi implementada em C++ usando a biblioteca Boost Asio. A aplicação funciona em dois modos: Cliente e Servidor. O servidor inicia a comunicação série com o Arduino *hub* do sistema, interagindo com ele consoante os comandos inseridos pelo utilizador. O utilizador pode executar os comandos (como exemplificado na figura em anexo 11.6) tanto no Servidor, como noutro computador, executando a aplicação em modo Cliente, sendo que esta vai comunicar com o Servidor por uma ligação TCP.

A implementação do Cliente é bastante simples. O programa apenas se liga por TCP ao Servidor, envia o texto que o utilizador insere, e escreve na linha de comandos a respetiva resposta do Servidor.

O Servidor é mais complexo. As principais funcionalidades são a comunicação por série com o Arduino, a ligação TCP com os clientes e a interpretação dos comandos. Para tirar partido do processador *multi-core* que o Raspberry Pi 3 possui, o programa corre em 4 threads, sendo que foram adotados mecanismos para prevenir problemas que possam surgir da implementação da execução de tarefas de modo concorrente.

9.1 Execução de comandos

Na figura 9.1 está representado um fluxograma da funcionalidade de execução de comandos do Servidor. Quando o é inserido texto na janela do Servidor ou enviado do Cliente, o Servidor interpreta esse texto para interpretar o comando inserido, e se este é válido. Se for, é colocado numa fila de comandos por executar.

Esta fila tem o propósito de garantir que caso o Arduino demore algum tempo a executar um comando, caso um utilizador insira outro, o Servidor sabe que a resposta corresponde ao primeiro comando. Sempre que a fila não está vazia, o Servidor envia o pedido de execução ao Arduino, e assim que recebe o valor da resposta, cria uma *string* com o formato correto que contenha essa resposta, e envia para o Cliente que inseriu o comando, ou imprime na janela do Servidor, caso o comando tenha sido inserido no servidor. É importante referir que foi implementado um mecanismo de *timeout*, que previne que o Servidor fique bloqueado à espera de uma resposta no caso de o Arduino não funcionar corretamente e nunca a enviar. Assim que o Arduino não responda ao pedido por mais de 2 segundos, uma mensagem de erro é impressa, o comando é ignorado e enviado o seguinte na fila caso exista.

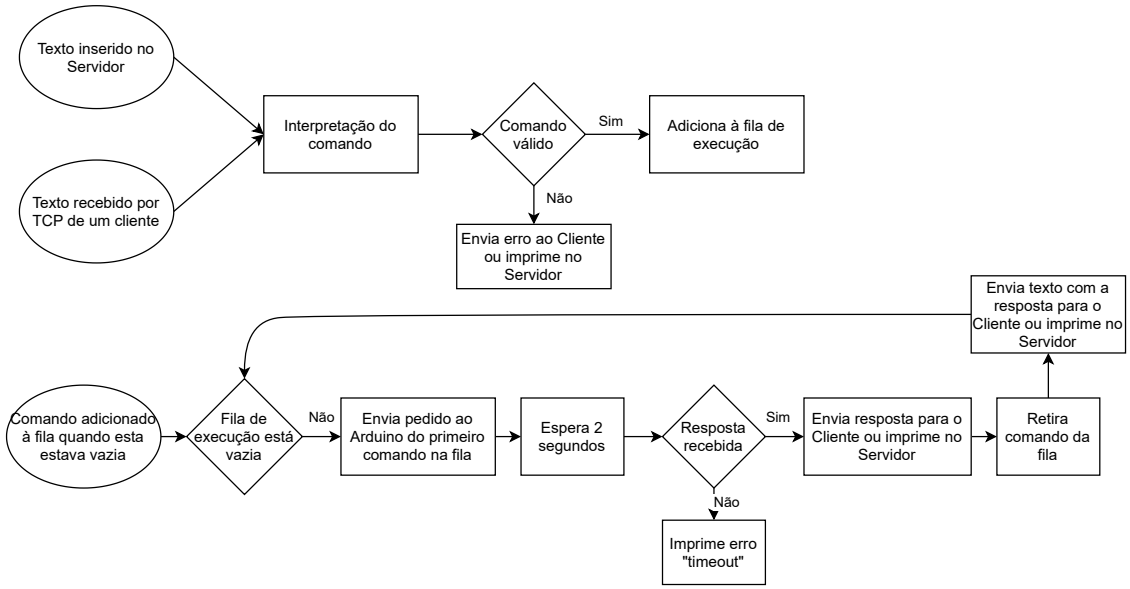


Figure 9.1: Fluxograma referente à execução de comandos no servidor

Existe um conjunto de comandos em que não é enviado nenhum pedido ao Arduino, nomeadamente os comandos "s", que inicia ou termina o modo *Streaming*, o comando "b", que imprime o *last minute buffer*, e os comandos "start save" e "stop save", que permitem que sejam guardados num ficheiro dados relativos ao comportamento do sistema, que serão mencionados de seguida.

9.2 Frequent data

Sempre que é recebido dados de *frequent data* (iluminância e PWM de um nó) vindos do Arduino, a aplicação toma duas ações:

- Se a funcionalidade de *streaming* estiver ativa nalgum Cliente para o nó a que os dados dizem respeito(ou no próprio Servidor), é enviado ao cliente o texto para ser impresso que contem o valor (ou escrito diretamente na janela do Servidor).
- São adicionados os valores ao *last minute buffer*

O last minute buffer guarda os dados de iluminância e PWM relativos a cada nó, para cada instante de tempo. Este buffer foi implementado com recurso a algumas estruturas de dados

disponibilizadas pela *standard library* do C++, nomeadamente o *std::map*, que permite associar chaves a valores, e o *std::vector* que permite criar uma lista dinâmica. Um primeiro *map*, em que a chave é o ID do nó, permite aceder a um buffer separado para cada nó. O buffer relativo a cada nó contem outro *map*, que permite uma lista(*std::vector*) com os valores por cada unidade de tempo. A unidade de tempo escolhida foi o segundo, ou seja, a chave do *map* é um valor que representa o segundo (Unix timestamp).

A cada segundo verifica-se se alguma das chaves que representam o instante de tempo são referentes a um instante há mais de um minuto, e esses são removidos. A remoção é assim bastante eficiente, sendo que basta eliminar essa entrada do *map*, em vez de eliminar amostra a amostra, caso não estivessem agrupadas pelo segundo do instante de tempo.

10 Conclusão

Em suma, conseguiu-se implementar todas as especificações previstas para o sistema. Nomeadamente os protocolos de comunicação, os comandos, a interface Sistema ↔ Utilizador, com a possibilidade de vários clientes ligarem-se ao servidor, e as métricas de desempenho.

As métricas que avaliam a performance do sistema podiam apresentar valores melhores, sendo que particularmente uma das luminárias apresenta valores um pouco maiores que o esperado.

Uma das limitações do sistema reflecte-se no facto de quando é feito o reset manual, no botão do arduino, este deve ser feito por todas as luminárias ao mesmo tempo. Caso isto não aconteça, o sistema não está preparado para receber um novo nó.

Uma das inovações do sistema criado é o facto de não haver a necessidade dos ID's dos nós ser sequencial. Assim, em caso de avaria de uma luminária, esta pode ser facilmente substituída por uma diferente, sem que se tenha de garantir que possui um ID igual à luminária que vem substituir. O programa está também preparado para trabalhar com mais do que 3 luminárias, o que facilita bastante o crescimento do sistema e a sua aplicação num contexto real. O custo de cada luminária, alteravel durante o programa, está a ser lido e escrito na EEPROM. Deste modo o custo de cada luminária mantém-se no programa entre utilizações, o que é benéfico pois é evitado a repetida definição de custo para cada luminária. Para uma maior liberdade na utilização da aplicação, é possível alterar o nó *hub* ao longo da execução do programa.

O sistema poderia ser melhorado, por exemplo, acrescentado funcionalidade úteis ao conforto do utilizador, como o controlo de temperatura. Outra melhoria implementável é a re-obtenção das leituras residuais ao longo do dia: é espectável que a luminosidade residual no escritório não seja a mesma de manhã (quando o sistema é tipicamente calibrado) ou a meio do dia.

References

[1] Slides das aulas teoricas
`https://fenix.tecnico.ulisboa.pt/disciplinas/SCDTR77/2020-2021/1-semestre/lectures`

[2] Datasheet do LDR
`https://www.alldatasheet.com/datasheet-pdf/pdf/217129/EVERLIGHT/SIR333C.html`

[3] Biblioteca da placa MCP-2515
`https://github.com/autowp/arduino-mcp2515`

[4] Enunciado do projeto
`https://fenix.tecnico.ulisboa.pt/downloadFile/1689468335654974/SCDTR2021-Project-V1.0.pdf`

11 Apêndice

11.1 Apêndice A - Imagens do modelo construído em escala reduzida

Figure 11.1: Caixa utilizada para simular o ambiente de *coworking*

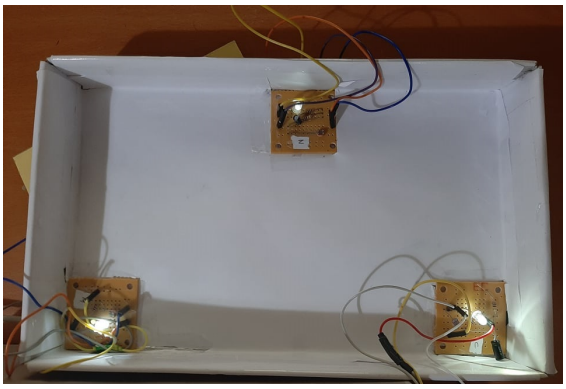


Figure 11.2: Disposição das luminárias pelo ”teto” da caixa

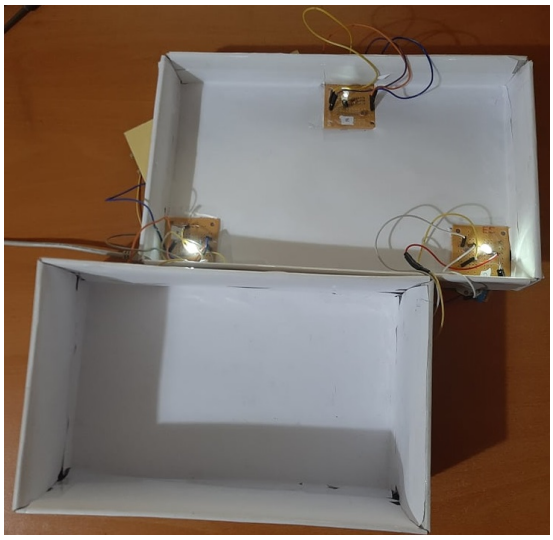


Figure 11.3: Caixa + luminárias

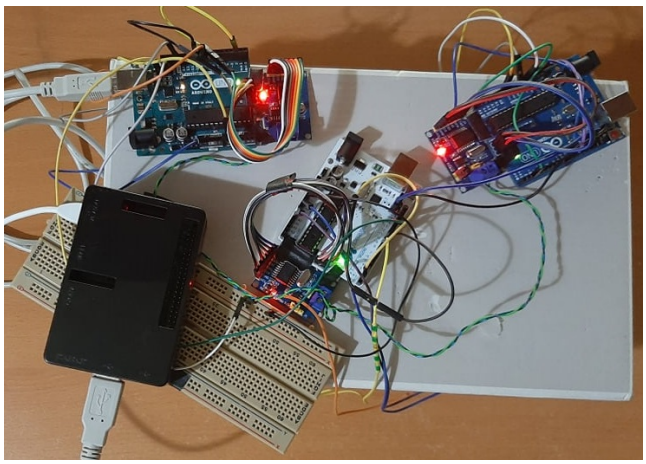
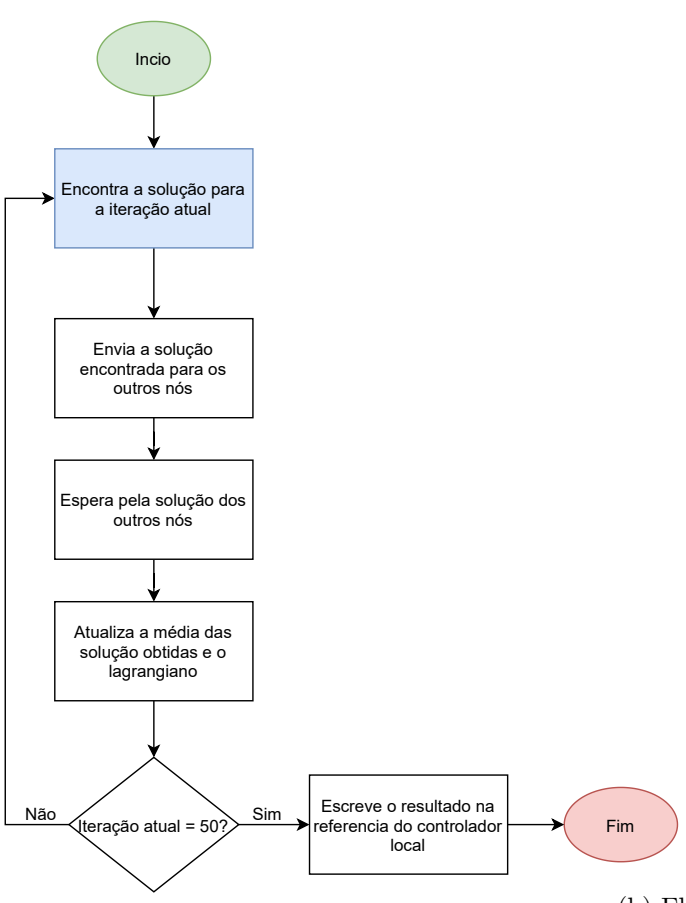
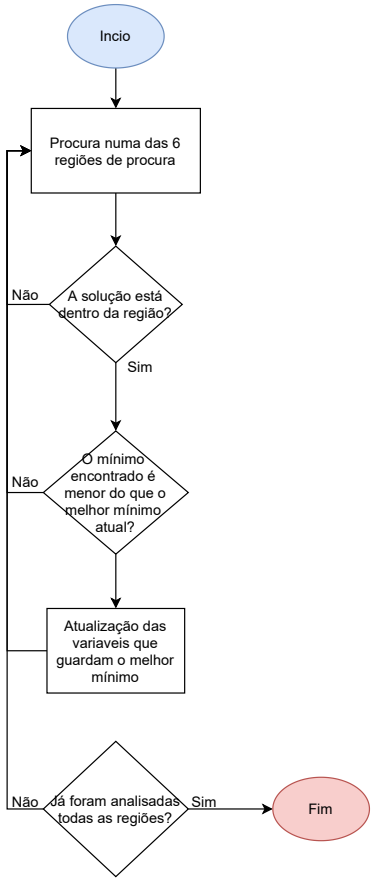


Figure 11.4: Fluxograma geral do algoritmo

11.2 Apêndice B - Fluxogramas do *Consensus*



(a) Fluxograma geral do algoritmo



(b) Fluxograma da função que obtém a nova estimativa em cada nó

Figure 11.5: Fluxogramas do algoritmo *Consensus*

11.3 Apêndice C - Interface do utilizador

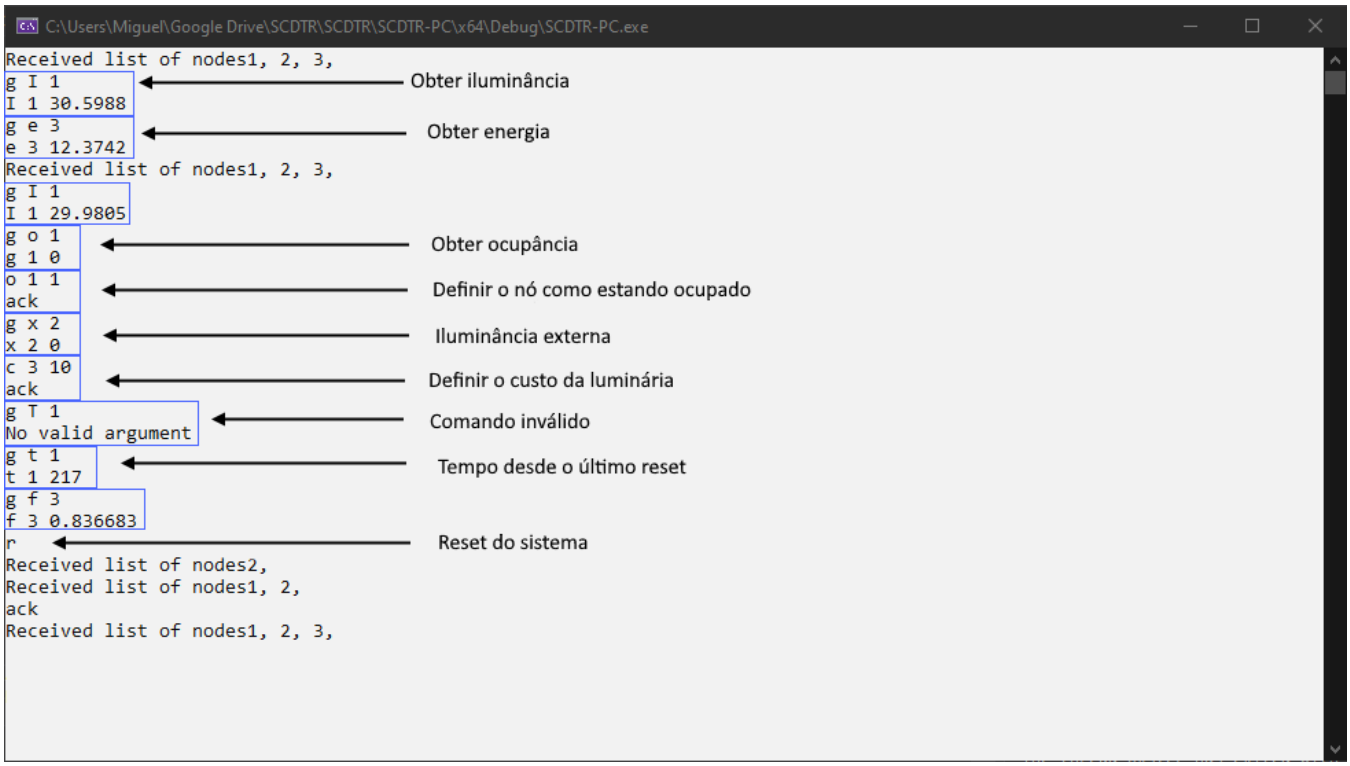


Figure 11.6: Exemplo de interação com o utilizador

11.4 Apêndice D - Tabela de constituições

	Daniel Chagas	José Coelho	Miguel Fazenda
Resumo	Reviu	Escreveu	Reviu
Introdução	Reviu	Escreveu	Reviu
Abordagem Proposta	Reviu	Escreveu, Ilustrou	Reviu
Abordagem Experimental	Reviu	Escreveu, Ilustrou, Adquiriu informação	Reviu
Controlo Local	Escreveu, Ilustrou, Implementou, Testou, Adquiriu informação	Reviu, Implementou, Testou	Reviu, Implementou, Testou
CAN-BUS	Escreveu, Ilustrou, Implementou, Testou	Reviu, Testou	Reviu, Implementou, Testou
Controlo Distribuido	Escreveu, Ilustrou	Escreveu, Ilustrou, Adquiriu informação	Reviu, Adquiriu informação
Comunicação Série	Implementou, Testou, Reviu	Implementou, Testou, Reviu	Escreveu, Ilustrou, Implementou, Testou
Aplicação PC Client-Server	Testou, Reviu	Testou, Reviu	Escreveu, Ilustrou, Implementou, Testou
Conclusão	Escreveu, Reviu	Escreveu, Reviu	Escreveu, Reviu

Figure 11.7: Tabela de contribuições de cada aluno