TÉCNICO LISBOA

Controlo de Iluminação Distribuído

Sistemas de Controlo Distribuído em Tempo Real (MEEC) - Grupo 5
Carlos Aleluia, 81038 — Filipe Madeira, 81076 — Mariana Martins, 80856

Projeto disponível em https://github.com/Mrrvm/SCDTR
Link para vídeo de funcionamento do sistema presente no GitHub

10 de janeiro, 2018

***Resumo*—Dado um sistema de iluminação, o objetivo deste projeto é controlá-lo distribuidamente de forma a maximizar o conforto do utilizador e a minimizar a energia gasta. Propomos a utilização do algoritmo *consensus* aplicado a um sistema de controlo, em que uma rede de agentes chega a um consenso da solução ótima, que gera a referência do sistema, ou determina que não existe uma solução viável. Definimos como a rede distribuída de agentes aplica o algoritmo e construímos meios para testar a sua eficiência. Obtivemos um sistema como pretendido inicialmente.**

Palavras-Chave: Consensus, PID, servidor TCP/IP, I2C sniffing, Controlo distribuído.

I. Introdução

A última década trouxe um nova forma de iluminação: LEDs (*Light Emitting Diodes*). Comparativamente à lâmpada incadescente convencional, o LED dura cerca de 50 vezes mais horas e usa muito menos potência [1]. A lâmpada fluorescente foi um avanço relativamente à anterior, mas o LED continua a ter mais duração e eficiência, e não tem os perigos associados ao mercúrio [2], apesar de ter perigos associados a outros metais. Permite, também, uma integração mais fácil com computadores e sensores. No entanto, esta tecnologia carece de optimização em termos de energia e conforto do utilizador. De acordo com o tipo de ambiente, com a iluminação externa existente e com a ocupação, deveria ser possível definir a luminosidade, frequência e côr dos LEDs que compõem o sistema de iluminação. Para que isto seja fazível, este sistema tem de ser definido como uma rede de agentes, em que cada agente tenta manter as características desejadas no seu LED, e podem comunicar entre si de modo a estabelecer o que é o melhor para todos.

Esta rede tanto poderia ser distribuída como centralizada, não obstante a versão distribuída apresenta alguns benefícios que tornam a solução mais facilmente escalável. Ao pôr o sistema embebido mais perto do ponto de atuação, os tempos de resposta são mais curtos, dado que um controlador central não tem de lidar com operações pequenas. Não é necessário fazer ligações com o nó central, que teria uma procura contínua por mais poder computacional, logo os custos de *hardware* são também mais baixos.

Este problema está atualmente a ser investigado [3], [4], [5], e que algumas soluções baseiam-se no uso do algoritmo *simplex*, que resolve problemas lineares e por si só não é distribuído. É neste contexto que propomos um sistema de controlo distribuído de forma coordenada por uma rede de agentes baseado no algoritmo *consensus* [6], que é monitorizado com estatísticas por clientes através de um servidor que as lê da rede. Para testar a nossa solução, estabelecemos a configuração apresentada na próxima secção. Todo o código do projeto está disponível no repositório de *github* indicado acima.

II. Configuração do sistema

O modelo em causa pretende simular uma sala com duas luminárias. Cada luminária tem um sensor, um microcontrolador (arduino UNO) e um LED. Todos os elementos se encontram inseridos dentro de uma caixa como se pode observar na figura 1. A caixa foi forrada a branco de maneira a refletir mais a luz, assim como os arduinos. As ligações foram feitas numa placa *breadboard*, que está colada ao chão da caixa. Na caixa existe também um motor servo e uma janela. O servo pode ser acionado por um dos arduinos, de maneira a abrir a janela. Por fora, a caixa incluí um Raspberry Pi 3B usado como servidor como se observa na figura 2.

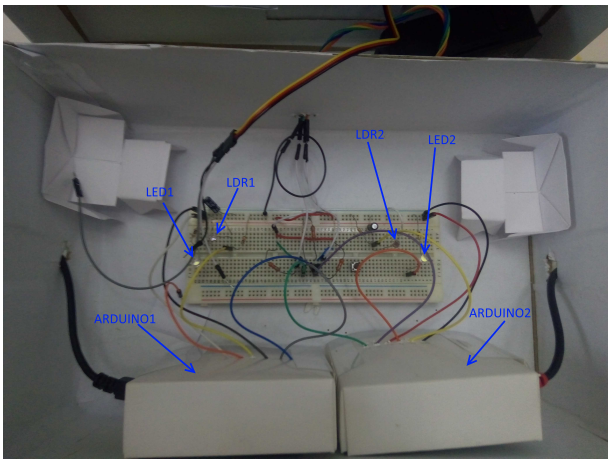

Figura 1: Caixa com luminárias.

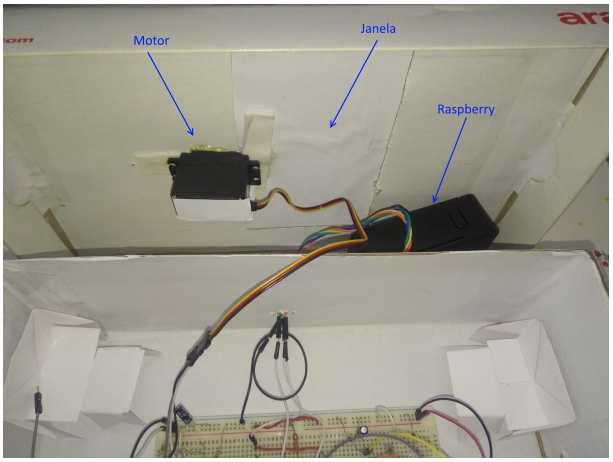

Figura 2: Motor, janela e raspberry.

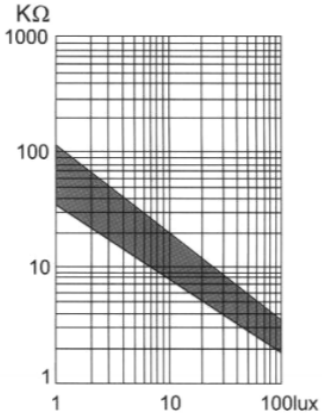

Figura 4: Resistência em função iluminação.

III. Abordagem

A. Calibração do LDR

O sensor de luz utilizado é um LDR (*Light Dependent Resistor*). A resistência diminui quanto maior for a intensidade de luz incidente. O sensor foi implementado de acordo com a figura 3. Com mais outra resistência é possível obter um divisor de tensão. Esta é depois lida pelo o arduíno e analisada. A tensão lida é dada pela expressão 1. O condensador permite a redução do ruído captado pelo sensor.

$$U = 5\frac{R_1}{R_1 + R_2} \tag{1}$$

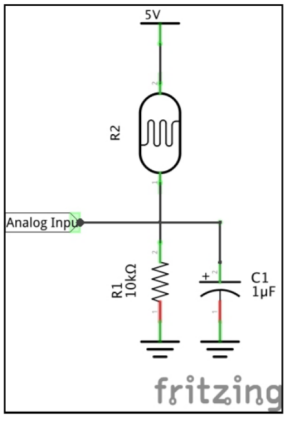

Figura 3: Circuito sensor de iluminação.

É necessário calibrar este sensor, pois existe uma gama de possíveis valores, como se pode observar na figura 4, com os quais se pode definir uma reta que indica o valor da resistência em função da iluminação. Para descobrir os parâmetros da reta foi usado um outro medidor de lux, onde se mediu a iluminação e de seguida com base nas equações da matriz em baixo calculou-se os parâmetros A e B, sendo o declive e a ordenada na origem respetivamente.

$$\begin{bmatrix} log_{10} * R_1 \\ log_{10} * R_2 \end{bmatrix} = \begin{bmatrix} log_{10} L_1 & 1 \\ log_{10} L_2 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}$$

B. Identificação individual do sistema

O sensor deteta a intensidade de lux atual e envia a informação para o arduíno. De seguida o arduíno ajusta a intensidade do LED de acordo com os requisitos impostos ao sistema. O ajuste é feito através de PWM (*Pulse Width Modulation*) e depende dos seguintes fatores:

- Luminosidade mínima com luminária desocupada (25 lux);
- Luminosidade mínima com luminária ocupada (50 lux);
- Intensidade de luz vinda do exterior;
- Intensidade de luz proveniente de outros LEDs.

O PWM varia entre 0 e 255 o que é equivalente a 0V e 5V em tensão contínua, mas é preciso saber a quantos lux equivale cada valor de PWM. Sabendo que a intensidade de luz varia linearmente com o PWM, determinou-se o ganho (declive da reta) que define essa função. Na figura 5 pode-se observar os lux em função do PWM e constatar que a variação é aproximadamente linear com um ganho de 0,3346.

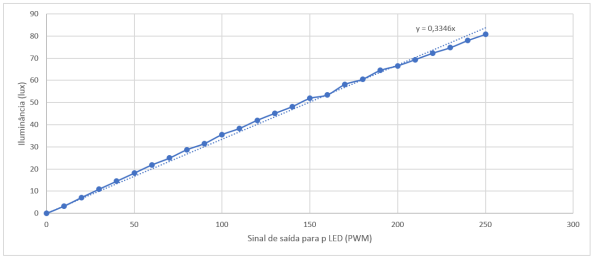

Figura 5: Lux em função do PWM.

De modo a converter o valor de PWM para lux chegou-se à equação final 2. O VDD é igual ao valor de 5V, PWM representa o valor que se quer converter e os parâmetros A e B representam a reta do LDR.

$$\left(\frac{(\frac{VDD}{PWM} - 1) * R1}{10^B} \right)^{\frac{1}{A}} \tag{2}$$

O sistema de controlo implementado está traduzindo na figura 6, sendo que tanto o valor de referência como o *feedback* são controlados pelo arduíno. Estes tópicos serão falados mais à frente. O *feedback* é essencial, pois garante que a luz proveniente do LED não é excessiva, diminuindo os consumos de energia e também porque permite que o sistema se adapte a uma nova situação de uma maneira mais confortável, ou seja, com transições menos bruscas.

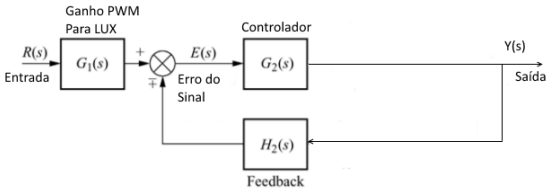

Figura 6: Esquema do controlador da luminária.

Um ponto importante para o bom funcionamento do sistema é a frequência de funcionamento, ou seja, a frequência de amostragem. O sistema tem que garantir a oscilação do LED não é detetável pelo olho humano, logo tem de ser inferior a 40ms. No entanto não faz sentido a frequência ser o mais baixo possível, pois implica um maior gasto de energia. Existe ainda um problema aderente ao condensador existente no circuito do sensor. O condensador requer um determinado tempo a carregar e portanto existe uma constante de tempo associada. De forma a medir esta constante foram retiradas amostras com o PWM a aumentar e obteve-se o gráfico da figura 7. O PMW foi aumentado apenas de meio em meio segundo de maneira a que tensão lida tivesse tempo de ficar estável.

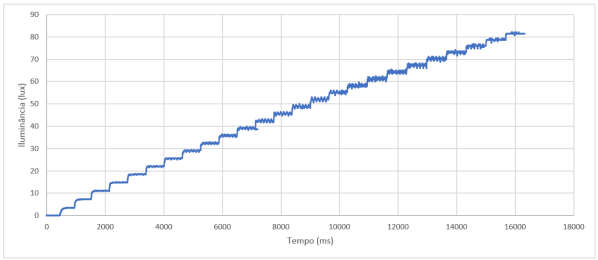

Figura 7: Lux em função do tempo.

Como era de esperar o condensador demora mais tempo a carregar (a chegar um valor de tensão constante) quando o PWM se encontra inicialmente a 0. Isto porque não só o condensador encontra-se completamente descarregado, mas também porque a constante é dada pela formula 3 e esta depende de R_2, sendo que é maior quanto menor for a intensidade de luz, como é o caso.

$$\tau = R_2 C_1 \tag{3}$$

Na figura 8 pode-se observar com mais pormenor a constante de tempo para o pior caso descrito em cima. A constante de tempo é determinada pelo o tempo que demora a atingir $\frac{1}{\sqrt{2}} = 70\%$ do valor em regime constante. Neste caso o valor obtido foi de 81ms. Este é um valor bastante alto, no entanto é um caso particular, que só acontece uma vez e não tem influência no sistema. Verificou-se que para mudanças de valores típicos, como por exemplo de 25 lux para 50, a constante é de 8ms. Como tal optou-se por escolher uma frequência de funcionamento tendo em conta a sensibilidade do olho humano e uma margem de segurança. O valor escolhido foi de 30ms.

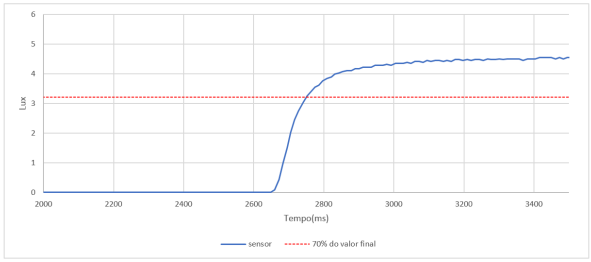

Figura 8: Lux em função do tempo para o pior caso.

Com a constante de tempo e o ganho obtidos, o sistema individual em ciclo aberto é dado por $\frac{K_0}{1+\tau}$.

C. Identificação do sistema com múltiplos arduínos

A rede de agentes é o aglomerado de vários sistemas individuais descritos anteriormente, sendo que os ganhos relativos de cada um não se alteram. No entanto, passam a existir ganhos entre eles, porque o LED de uma luminária pode ter influência na intensidade de luz recebida por outra. N luminárias implicam a existência de uma matriz de ganhos com dimensão N x N, que neste caso é de 2 x 2.

$$\begin{bmatrix} Y_{11} \\ Y_{22} \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{bmatrix} X_{11} \\ X_{22} \end{bmatrix}$$

Na matriz Y é o valor de saída, K é o ganho de uma luminária em relação a outra e X é a luminosidade de uma mesma luminária.

D. Controlador

O controlador é responsável por garantir que a intensidade de luz numa dada luminária se mantém dentro dos parâmetros estabelecidos. É constituído por 2 componentes: *feedfoward* e *feedback*. Este último tem uma componente proporcional e uma componente integral. Cada arduino equivale a um controlador local e a única informação que tem sobre os outros arduinos, caso existam, é o ganho de cada arduino relativamente a si próprio, ou seja, a matriz discutida na secção III-C.

Na figura 9 pode-se observar, mais detalhadamente o sistema de controlo. Todas as componentes irão ser explicadas de seguida. A componente derivativa foi retirada, pois não só impedia que o PID funcionasse em tempo real, mas também porque para este caso, esta componente não tem qualquer efeito.

Para minimizar o *flickering*, na medição do erro face à referência do controlador implementou-se uma *dead zone*, ou

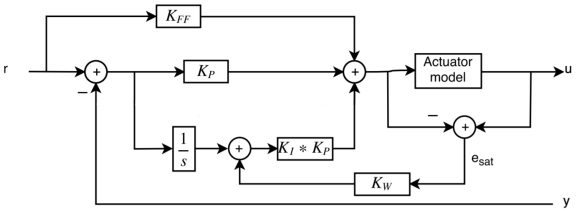

Figura 9: PID.

seja, o erro medido apenas era considerado caso fosse superior a um patamar, em módulo. Uma representação gráfica da *dead zone* implementada pode ser vista na figura 10, com parâmetro ERROR_MAX=1.

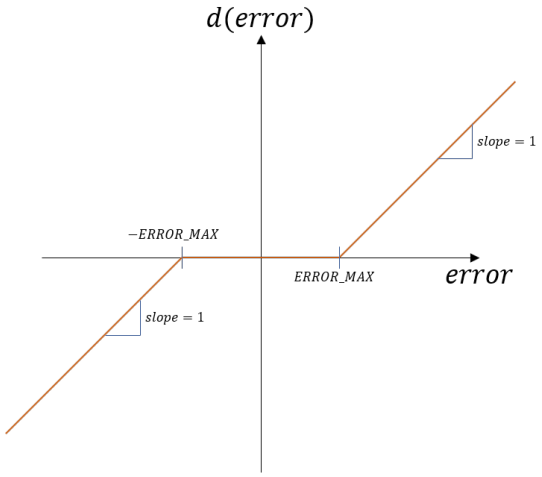

Figura 10: *Deadzone* implementada: erro fornecido ao PID ($d(error)$) em função do erro medido ($error$).

1) Feedfoward: Consiste em enviar um sinal de controlo do arduíno para o LED, de modo a acender o LED com uma certa intensidade de luz. Neste caso o controlo é feito com PWM. O feedfoward permite ter uma melhor precisão, assim como uma melhor estabilidade do sistema e permite ainda atingir o valor de referência mais rapidamente. É controlado através da constante K_{FF} equivale ao ganho K medido anteriormente.

2) Feedback: Componente proporcional: Produz um sinal de saída que é proporcional à amplitude do erro e(t), sendo K_p a constante de proporcionalidade. Este método possui a vantagem de eliminar as oscilações do sinal de saída.

3) Feedback: Componente integral: Produz um sinal de saída que é proporcional à magnitude e à duração do erro, ou seja, ao erro acumulado. Isso fornece uma alternativa para corrigir o erro de *offset* gerado pela ação proporcional e acelera a resposta do sistema, permitindo-o chegar ao valor de referência mais rapidamente.
Contudo, o termo integral também pode levar muitas vezes a instabilidade ou a problemas de *windup*. O *windup* consiste na acumulação de erros muito elevado (devido a saturação de actuadores, por exemplo), e caso ocorra leva a atrasos na resposta ou a *overshooting*. No caso deste projeto, se a iluminação externa for maior do que a iluminação mínima pretendida e caso esta seja a referência dada ao PID, como o sistema não consegue retirar luminosidade (saturação dos atuadores), então o termo integral vai acumular erros negativos durante um largo período. Se, nestas condições, a iluminação externa voltar a diminuir, como o termo integral do *feedback* é tão negativa o sistema vai permanecer apagado até descarregar o erro acumulado. Um raciocínio semelhante podia ser feito para o caso inverso, levando a *overshooting*.
Portanto, implementou-se uma medida de *anti-windup*: em cada ciclo de controlo, se o termo integral for superior a uma constante W ou inferior a $-W$, então este é saturado, ou seja, o módulo do termo integral é sempre mantido no intervalo $[-W, W]$.

Existem diferentes formas de associar o *feedforward* e o *feedback* no sistema. Neste projeto, foi utilizada a técnica de *decoupled feedforward*, que consiste em utilizar o *feedfoward* para obter a resposta pretendida em condições ideais e o *feedback* apenas ter interferência para rejeitar perturbações ou não idealidades do sistema. Para isso, é necessário um conhecimento mais profundo e uma boa modelização do sistema, pois consiste em fornecer como referência ao *feedback* o comportamento esperado do sistema quando fornecido o *feedforward*.

No nosso caso, em boa aproximação (como aliás foi visto na secção III-B), a resposta do sistema ao *feedforward* pode ser modelada por um escalão de amplitude correspondente à iluminação de referência pretendida com um atraso de um intervalo de *sampling*. Portanto, num primeiro ciclo de controlo após a alteração de referência (e, portanto, de valor de *feedforward*) o termo de *feedback* fornecido ao sistema é nulo; nos ciclos seguintes, tudo se desenrola normalmente, com referência igual ao valor que se pretende que a luminária imponha através do LED.

E. Calibração do sistema

Como foi referido na secção III-C, numa rede de vários arduinos existem efeitos de acoplamento, traduzidos em primeira ordem por ganhos K_{ij}. Como qualquer alteração de iluminação externa ou de organização no interior da caixa usada implica mudanças nesses ganhos, era fundamental a existência de um método de calibração em cada inicialização do sistema (ou reinicialização, mediante a ordem do cliente).

Portanto, cada vez que o sistema é iniciado ou reiniciado, todos os arduinos na rede iniciam um protocolo que permite cada um deles armazenar um *array* de ganhos relativos a todos os outros arduinos. Este protocolo consiste num ciclo em que, um arduino de cada vez, percorre todos os valores de *duty cycle* (0-255) e todos os outros (incluindo o próprio que está a acender) medem a iluminação no seu LDR. Os ganhos são então calculados linearizando a curva:

$$K_{ij} = \frac{l_i}{d_j}, \tag{4}$$

em que l_i é a iluminação medida na luminária i e d_j o *duty cycle* imposto na luminária j. No final de serem percorridos todos os valores de d_j, toma-se como ganho a média dos declives.

Para minimizar erros aleatórios, permitiu-se ainda definir uma variável C que indica o número de amostras que serão retiradas (ou seja, cada arduino percorre todos os *duty cycle* de 0-255 C vezes).

O processo de troca de informação entre os arduinos (envio do *duty cycle* imposto de cada vez pelo arduino a acender o LED) será explicado mais detalhadamente na secção III-G, e todo o processo de calibração em termos de *software* será descrito na secção III-H.

F. Controlo Distribuído e Otimização

Quando existem vários controladores em jogo, há várias topologias distintas de como organizar o controlo distribuído, mais concretamente, a capacidade de tomada de decisões ou a forma como se comportam para atingir os seus objetivos. No caso específico deste projeto, a topologia utilizada foi de controlo distribuído descentralizado, pois não existia nenhum controlador central que toma todas as decisões globais, mas sim vários controladores capazes de comunicar entre si as suas decisões.

Quanto à cooperação ou não entre os diferentes nós, é possível o sistema funcionar das duas formas. Para agir de forma não cooperativa, basta colocar os controladores individuais a operar segundo a descrição feita na secção III-D1: cada arduino coloca o *duty-cycle* necessário para atingir a luminosidade mínima estabelecida, ignorando a existência do outro arduino. Nesta forma de atuar, cada agente apenas se preocupa em cumprir o seu próprio objetivo, e trata qualquer ação de um outro agente como perturbações externas e desconhecidas.

Para funcionar de forma cooperativa, é necessário um procedimento mais complicado, em que se procede à otimização de uma função de custo tendo em conta todas as variáveis, por exemplo, efeitos de *coupling* entre os dois controladores. Assim sendo, é fundamental que tenham um sistema de coordenação e comunicação eficaz para que cheguem a um consenso acerca das variáveis que cada um tem que manipular, a fim de atingirem a solução ótima global.

Como foi referido nos objetivos do projeto, pretende-se minimizar o consumo de energia, maximizando o conforto de utilizadores (mantendo a iluminação acima de certos limites e reduzindo o *flicker* ao mínimo possível). Portanto, a função de custo que se pretende minimizar vai ter dois termos: um relacionado com o consumo de energia, ou seja, com dependência linear nos *duty cycle* impressos pelos controladores; e outro termo quadrático nos $\boldsymbol{d}$, que representa um custo adicional por desgaste da lâmpada, isto é, ao penalizar soluções com *duty cycle* perto do máximo, aumenta a longevidade do LED. Na verdade, o problema de otimização pode ser formulado da seguinte forma:

$$\begin{aligned} &\underset{d_1,...,d_N}{\text{minimize}} && \boldsymbol{c}^T\boldsymbol{d} + \boldsymbol{d}^T\boldsymbol{Q}\boldsymbol{d} \\ &\text{subject to} && \boldsymbol{K}\boldsymbol{d} + \boldsymbol{o} \geq \boldsymbol{L}, \\ &&& 0 \leq d_i \leq 255,\ i = 1,...,N. \end{aligned} \tag{5}$$

Como é possível observar, a função de custo tem os dois termos mencionados, cuja influência relativa é determinada pelos coeficientes c_i (termo linear) e Q_{ii} (termo quadrático). Nota-se que a matriz $\boldsymbol{Q}$ é diagonal pois não faz sentido a existência de termos quadráticos cruzados. Já acerca das restrições, a primeira garante que a iluminação imposta pelo conjunto de todos os arduinos ($\sum_{j=1}^{N} K_{ij}d_j$), acrescentada à iluminação externa existente (o_i), é superior ao limite mínimo (L_i). A segunda restrição impõe os limites físicos aos *duty cycle* impostos. Imaginando o caso $N = 2$, uma forma fácil de entender as restrições relacionadas com a iluminação mínima é escrevê-las na forma: $K_{11}d_1 + K_{12}d_2 \geq L_1 - o_1 \leftrightarrow d_2 \geq \frac{L_1-o_1}{K_{12}} - \frac{K_{11}}{K_{12}}d_1$, e o mesmo para a restrição de iluminação mínima no segundo controlador. Na figura 11 está representada a região delimitada pelas restrições para o caso $N = 2$. Uma forma gráfica de entender ao que corresponde o mínimo que se procura pode ser visualizada na figura 12.

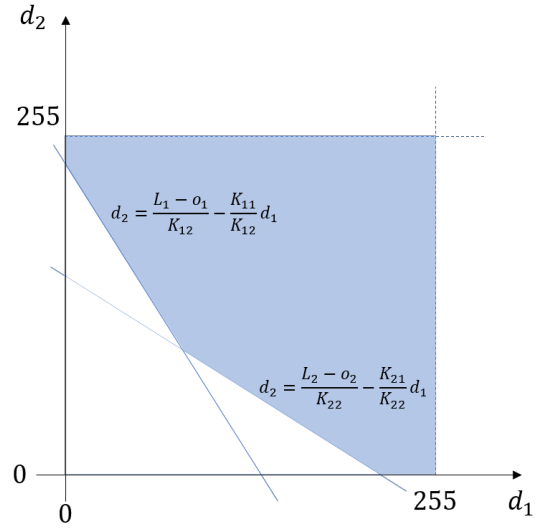

Figura 11: Representação, para o caso $N = 2$, da região delimitada pelas restrições.

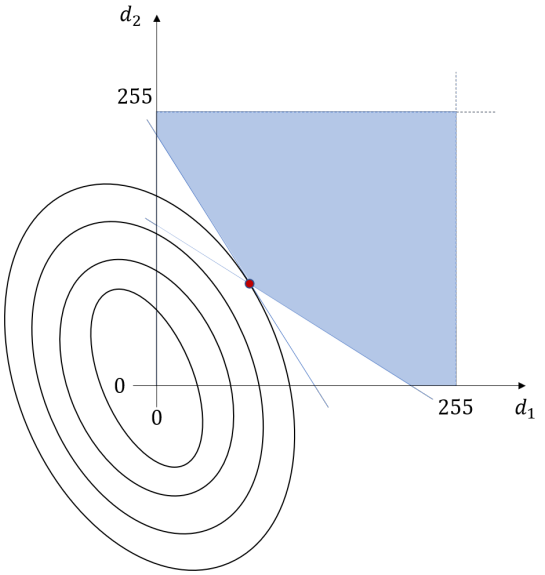

Figura 12: Representação, para o caso $N = 2$, do mínimo da função de custo, sujeita às restrições.

O problema descrito é um problema quadrático, de resolução fácil caso o problema fosse resolvido por um processador central com informações de todos os nós. Contudo, como neste caso se pretende uma solução distribuída, cada controlador deve calcular a sua variável de controlo de forma a minimizar a função de custo global, desconhecendo, na sua totalidade, a informação dos outros controladores. Dependendo de algumas propriedades da função de custo e das restrições, esta distribuição pode ser mesmo impossível. No entanto, para o problema em estudo, uma vez que a função a minimizar é totalmente separável e as restrições podem ser parcialmente decompostas, isto é, o problema pode ser reescrito na forma:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{N} f(d_i) \\ d_1,...,d_N & \\ \text{subject to} & \sum_{i=1}^{N} h(d_i) \leq 0, \\ & d_i \in \Omega_i,\ i = 1,...,N, \end{array} \tag{6}$$

existem métodos capazes de resolver problemas de otimização distribuídos como este. Neste projeto, foi utilizado o método de *alternating direction method of multipliers*, ADMM, uma ferramenta para resolver problemas de otimização distribuídos e que podem ser decompostos até certo ponto, como o presente [6]. Mais concretamente, vai ser aplicado a um problema de otimização de *consensus*, isto é, onde existem variáveis globais - variáveis partilhadas por todos os nós - que devem refletir as variáveis presentes em cada um dos controladores. No caso apresentado, estas variáveis globais vão ser os valores médios de cada d_i entre todos os nós. Resumindo, o algoritmo vai impôr a restrição de que, em cada controlador, as cópias do $\boldsymbol{d}$ sejam iguais ao valor médio que é partilhado por todos, ou seja, que sejam iguais entre si! Desta forma, cada um pode proceder à otimização com restrições, envolvendo os *duty cycle* dos outros, sabendo que todos possuem o mesmo valor. [7]

O método utilizado é, então, um método iterativo, em que a condição $d_i - \bar{d}_i = 0$, onde $\bar{d}_i$ é a média sobre todos os nós de d_i, é satisfeita utilizando penalizações de Lagrangiano aumentado, e em que cada iteração é atualizada a variável dual (multiplicadores de Lagrange), de acordo com o *dual ascent method* [7]. Ou seja, em cada nó i são feitas as seguintes operações, na iteração t:

$$\begin{cases} \boldsymbol{d_i} = \arg\min_{\boldsymbol{d_i} \in \chi_i} (\frac{1}{2}\boldsymbol{d_i}^T \boldsymbol{Q} \boldsymbol{d_i} + \boldsymbol{c_i}^T \boldsymbol{d_i} + \boldsymbol{y_i}^T(\boldsymbol{d_i} - \bar{\boldsymbol{d}}_i) + \frac{\rho}{2}||(\boldsymbol{d_i} - \bar{\boldsymbol{d}}_i)||_2^2) \\ \bar{\boldsymbol{d}}_i^{\,t+1} = \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{d_j}^{t+1} \\ \boldsymbol{y_i}^{t+1} = \boldsymbol{y_i}^t + \rho(\boldsymbol{d_i}^{t+1} - \bar{\boldsymbol{d}}_i^{\,t+1}), \end{cases} \tag{7}$$

com $\boldsymbol{y_i}$ sendo os multiplicadores de Lagrange e ρ o parâmetro de penalização de Lagrangiano aumentado. Nota-se que os índices t não foram colocados na primeira equação para não sobrecarregar a notação, mas naturalmente é calculado o $\boldsymbol{d_i}$ utilizando os parâmetros da iterada anterior. Como a otimização é distribuída, os parâmetros da função de custo apenas consideram o efeito do nó atual, ou seja, $\boldsymbol{c_i}^T = [0 \cdots c_i \cdots 0]$ e $\boldsymbol{Q} = \text{diag}(0 \cdots q_i \cdots 0)$, e as restrições são locais:

$$\chi_i = \{\boldsymbol{d_i} \in \mathbb{R}^N : 0 \leq d_{ii} \leq 255\ , \sum_{j=1}^{N} K_{ij} d_{ij} \geq L_i - o_i\}. \tag{8}$$

Já relativamente à resolução do problema com restrição em cada nó, mostra-se facilmente que se reduz a (referência!!):

$$\begin{array}{ll} \boldsymbol{d_i}^{t+1} = & \text{argmin}(\frac{1}{2}\boldsymbol{d_i}^T \boldsymbol{R_i} \boldsymbol{d_i} - \boldsymbol{d_i}^T \boldsymbol{z_i}), \\ & \boldsymbol{d_i} \in \chi_i \end{array} \tag{9}$$

em que $\boldsymbol{R_i} = \boldsymbol{Q_i} + \rho \boldsymbol{I}$ e $\boldsymbol{z_i} = -\boldsymbol{c_i} - \boldsymbol{y_i} + \rho \bar{\boldsymbol{d}}_i$. Portanto, trata-se de um problema de otimização convexo. A solução pode estar no interior da região permitida pelas restrições ou pode estar na sua fronteira, como pode ser visto na figura 13 para o caso $N = 2$. Nota-se que nesta imagem já se aplicou o facto de apenas condicionarem a solução as restrições locais.

Para verificar se o mínimo está no interior, calcula-se quando o gradiente da função de custo anterior se anula, ou seja,

$$\boldsymbol{d_i}^* = \boldsymbol{R_i}^{-1} \boldsymbol{z_i}, \tag{10}$$

e testam-se as restrições.

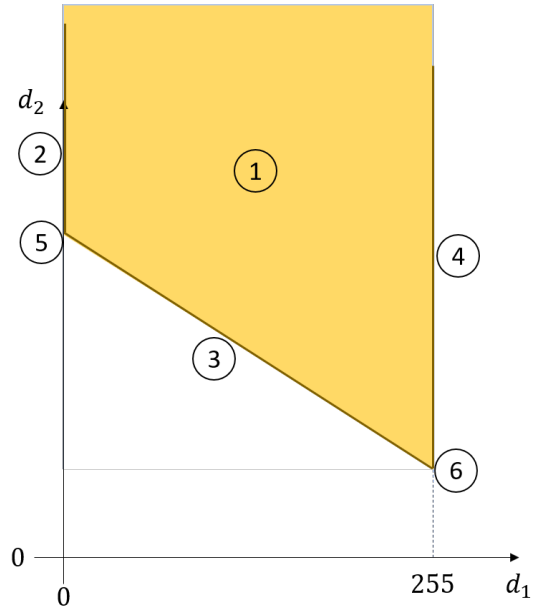

Figura 13: Representação, para o caso $N = 2$, dos diferentes tipos de soluções possíveis: no interior do conjunto, nas fronteiras e nas suas interseções.

Caso a solução esteja na fronteira, pode estar na fronteira de qualquer uma das 3 restrições ou na interseção de quaisquer pares possíveis (ver figura 13). Se as fronteiras l possíveis estiverem escritas na forma

$$\boldsymbol{A_i}^{(l)} \boldsymbol{d_i} = \boldsymbol{u_i}^{(l)}, \tag{11}$$

então prova-se [6] que a solução nessa fronteira é dada por:

$$\boldsymbol{d_i}^* = \boldsymbol{R_i}^{-1} \boldsymbol{z_i} + \boldsymbol{R_i}^{-1} \boldsymbol{A_i}^T (\boldsymbol{A_i} \boldsymbol{R_i}^{-1} \boldsymbol{A_i}^T)^{-1} (\boldsymbol{u_i} - \boldsymbol{A_i} \boldsymbol{R_i}^{-1} \boldsymbol{z_i}). \tag{12}$$

A título de exemplo, para a solução 4 representada na figura 13 (ou seja, restringida à fronteira representada por $d_1 = 255$), $\boldsymbol{A_i}^{(4)} = [1\ 0]$ e $u_i^{(4)} = 255$.

Numericamente, em cada iteração, calculam-se as 6 soluções possíveis, testa-se a sua viabilidade recorrendo às restrições e escolhe-se a que estiver associada a um valor da função de custo menor (utilizando a expressão completa,

presente na equação 7). Em termos de código, foi utilizada uma biblioteca que definia produtos matriciais e vetoriais, de forma a que o cálculo das soluções fosse totalmente genérico e escalável, apenas fazendo os produtos presentes em 10 e 12. A biblioteca utilizada foi a MatrixMath [8].

No final de cada iteração, os nós procedem à troca dos $\boldsymbol{d_i}$ calculados por cada um, de forma a que calculem a média $\bar{\boldsymbol{d_i}}$ e seja igual em todos. Este processo de troca de informação também é totalmente genérico e escalável, e será melhor descrito na secção III-G.

No final, quando o algoritmo convergir (ou seja, quando todos os nós tiverem a mesma solução final), o termo de *feedforward* que cada arduino passa a colocar é o seu *duty cycle* respetivo da solução final. Além disso, a referência que os termos de *feedback* passam a seguir também é alterada, por duas razões. Em primeiro lugar, podia acontecer que, devido a custos assimétricos, a solução ótima implicasse um deles colocar mais intensidade do que seria necessário caso estivesse isolado, para auxiliar os outros. Nessas condições, o *feedback* ia tentar baixar a iluminação medida para o mínimo pois apenas tenta manter a referência. Em segundo lugar, caso existisse iluminação externa mais do que suficiente para garantir o mínimo, o *feedback* também podia deixar de tentar baixar o seu valor até à referência antiga. Resumindo, a expressão que foi utilizada para determinar a nova referência foi a seguinte:

$$L_i' = \max\{L_i, \sum_{j=1}^{N} K_{ij} d_{ij} + o_i\}, \tag{13}$$

em que L_i' é a nova referência. O facto de se utilizar o máximo é só para garantir que, se por algum tipo de imprecisão numérica o segundo termo for ligeiramente inferior ao mínimo permitido nessa luminária, L_i, então a referência para o *feedback* faz subir a iluminação do que faltar até L_i, em vez de se contentar com iluminação um pouco abaixo do mínimo.

G. Comunicação I2C

A comunicação entre os arduinos e a recolha de informação pelo Raspberry Pi foi feita utilizando o protocolo de comunicação série síncrono *Inter-Integrated Circuit*, normalmente designado apenas por I2C. Este protocolo permite não só comunicação entre vários *masters* (quem controla as linhas de comunicação) e vários *slaves* (para quem a mensagem do *master* é endereçada), mas também permite comunicação entre um número muito elevado de dispositivos utilizando apenas 2 fios: *clock line* (SCL) e *data line* (SDA).

Segundo este protocolo, os dispositivos nas linhas devem estar associados a um endereço de 7 bits. As comunicações são iniciadas pela condição de *START* e terminadas pela condição de *STOP*, impostas às linhas SCL e SDA pelo dispositivo a atuar como *master*. Cada conjunto de informação é composta por 8 bits (1 byte), tendo o *master*, após a condição de *START*, ter que comunicar os 7 bits de endereço do *slave* a quem se dirige, além do bit final, que indica se pretende fazer *write* ou *read* do *slave* [9].

Cada arduino permite a comunicação por I2C nestes dois modos de funcionamento, ou seja, em *write* ou *read*, como referido atrás:

- MT/SR - *Master Trasmitter, Slave Receiver*: o dispositivo a operar como *master* envia dados para o dispositivo a servir de *slave*, que apenas recebe;
- MR/ST - *Master Receiver, Slave Trasmitter*: o dispositivo a operar como *master* solicita dados do *slave*, que responde.

Neste projeto apenas foi utilizado o primeiro modo de funcionamento descrito (MT/SR) pois é o mais simples e menos suscetível de provocar erros. A biblioteca *Wire* do Arduino permitiu facilitar a implementação do protocolo, apenas tendo o dispositivo *master* que iniciar a transmissão, especificando o endereço do *slave* a quem se pretendia dirigir.

No âmbito do projeto, todas as comunicações por I2C foram feitas em modo *broadcast*, ou seja, enviadas para todos os dispositivos, que não o *master*, receberem. Isto foi implementado dirigindo as transmissões para o endereço "0". De qualquer forma, para cada arduino poder participar nas comunicações, foi necessário atribuir um endereço a todos. Para isso, escreveu-se na EEPROM um inteiro que foi interpretado como o seu endereço de comunicação. Este número também foi utilizado para outras aplicações, sempre que era necessário tomar ações diferentes consoante o índice do arduino (por exemplo, na calibração e no algoritmo de controlo distribuído). Relativamente à frequência da linha SCL, utilizou-se a predefinida, ou seja, 100kHz. Contudo, testou-se até 400kHz e o sistema continuava a funcionar corretamente.

Em todas as comunicações iniciadas pelos arduinos, antes de se enviarem os dados propriamente ditos, é sempre enviada uma *label* (um caracter ASCII) para facilitar o processo de interpretação dos dados enviados não só pelos restantes arduinos, como também pelo Raspberry Pi na tarefa de recolher dados do sistema. De seguida encontram-se detalhadas todas as comunicações empregues por I2C no sistema:

- **Calibração**, *Label*: "k": enviada por um arduino enquanto estão todos a efetuar a calibração. O arduino que está a acender o LED envia o *duty cycle*, de 0 a 255, que está a colocar no LED para os outros calcularem o ganho cruzado.
- **Consensus**, *Label*: "c": enviada por um arduino enquanto estão todos no algoritmo de otimização distribuído. Um arduino, de cada vez, envia 2 inteiros: um para identificar qual o índice do d enviado (na verdade, trata-se do índice-1, para facilitar) e outro para enviar o valor em si. Por exemplo, se a solução ótima calculada, pelo arduino 2 para o arduino 1 (1-1=0), for $d_1 = 100$, então a sequência será "c","0","100".
- **Restart**, *Label*: "r": enviada por um arduino quando recebe um comando, via *Serial* do Raspberry Pi, a indicar para recomeçar o sistema e recalibrar. Os arduinos que receberem essa *label* ficam com uma *flag* "calibrated"=0, portanto a próxima vez que iniciarem o *loop* entram na calibração, todos coordenados.
- **Recalibração**, *Label*: "b": enviada por um arduino

quando é reiniciado. Esta comunicação é necessária porque, por exemplo, ao ligar a *Serial* do Raspberry Pi, o arduino que lhe estiver ligado faz automaticamente *reset*, ou seja, precisa de voltar a fazer a calibração. Uma vez que a calibração apenas funciona se todos estiverem a realizá-la simultaneamente, então é necessário enviar esta informação para que os restantes arduinos na rede voltem à etapa de calibração e se mantenham todos coordenados.

- **Set**, *Label*: ”s”: enviada por um arduino quando recebe um comando, via *Serial* do Raspberry Pi, a indicar alterar a ocupação de um arduino qualquer do sistema. Após a *label*, envia exatamente o inteiro que recebeu por *Serial*, que será interpretado pelo arduino alvo, através da codificação: $(address - 1) * 2 + 1 \rightarrow$ ocupação=0 ; $(address - 1) * 2 + 2 \rightarrow$ ocupação=1. Esta codificação permite minimizar a informação enviada e maximizar o número de arduinos na rede a quem esta mensagem pode chegar. A título de exemplo, caso se pretenda colocar a ocupação do arduino 2 a 1, o inteiro enviado por *Serial* e retransmitido por I2C para a rede inteira é $(2 - 1) * 2 + 2 = 4$.
- **Refazer o consensus**, *Label*: ”z”: enviada por um arduino se receber, por meio da comunicação anterior, ordens para alterar a ocupação e, portanto, a sua iluminação mínima. Quando os restantes arduinos recebem esta *label* realizam todos o algoritmo de controlo distribuído de forma coordenada.
- **Ligar/Desligar controlo distribuído**, *Label*: ”p”: enviada por um arduino quando recebe um comando, via *Serial* do Raspberry Pi, a indicar alterar o estado do controlo distribuído. Se receber uma instrução para desligar o algoritmo de controlo distribuído, então envia, além da *label*, um ”0”; caso seja para o ligar, envia um ”1”.
- **Informação para o Raspberry Pi ler**, *Label*: ”a”: enviada por todos os arduinos após cada ciclo de controlo, caso sejam efetuados cálculos. Esta mensagem é totalmente ignorada pelos arduinos, é apenas enviada para fins de coletar os dados pelo Raspberry Pi. Cada arduino envia os dados seguindo a forma: ”a”,<endereço>,”l”,<iluminação medida>,”d”,<duty cycle>,”o”,<ocupação>. Nota-se que o tempo que cada arduino demora a enviar esta mensagem é ligeiramente inferior a $1ms$, pelo que caso a rede crescesse acima de 25 nós, alguma alteração teria que ser feita (retirar os caracteres auxiliares ou aumentar a frequência do *SCL*, por exemplo).
- **Referência e iluminação externa para o Raspberry Pi ler**, *Label*: ”i”: enviada por todos os arduinos após terem efetuado a calibração e o controlo distribuído. Esta mensagem é totalmente ignorada pelos arduinos, é apenas enviada para fins de coletar os dados pelo Raspberry Pi. Cada arduino envia os dados seguindo a forma: ”i”,<endereço>,”t”,<referência>,”x”,<iluminação externa>.

H. Arquitetura do software do Arduino

O funcionamento do programa nos arduínos está representado na 14. Ao reinicializar o sistema ”r”, mudar a abertura da janela ”w”, fazer *reset* a um dos arduínos ”b”ou simplesmente iniciar o sistema, é feita a calibração representada na figura 15. O código respetivo à mesma está contido nas funções *loop_calib()* e *calib()* em *arduino/main/Arduino_final.ino*. Começa por medir a iluminação externa na caixa, de acordo com o nº de amostras definido, para a poder descontar mais tarde à luminosidade lida do LDR. A variável *step* representa o endereço da lumiária que está a calibrar, isto é, que está a iterar o *duty-cycle* do LED de 0 a 255, e que a cada iteração envia esse valor para todas as outras luminárias e obtém o valor dessa luminosidade medido no seu próprio LDR. Este processo é repetido consoante o nº de amostras, quando estiver completo aumenta o *step* para representar a calibração da próxima luminária. Enquanto isto, as outras luminárias foram recebendo valores de *duty-cycle* da que estava a calibrar, leram esses valores nos seus LDRs e calcularam o ganho relativo a essa luminária com todas as amostras, como foi explicado na secção III-E. No final, guardaram o ganho e incrementaram o *step*, pelo que agora a próxima luminária a calibrar é a que tiver o endereço correspondente ao mesmo. Para se definir o que é o final, foi criada uma variável *count* que conta o nº de medições feitas. No entanto, como os arduínos podem estar desfasados, o minímo a partir do qual se considera o fim não é exatamente 255 vezes o nº de amostras, mas sim o valor definido experimentalmente perto disso. Adicionalmente, todas as luminárias verificam se o arduino da que está a calibrar morreu através do tempo que está a demorar, em caso positivo assumem que o ganho relativo é zero.

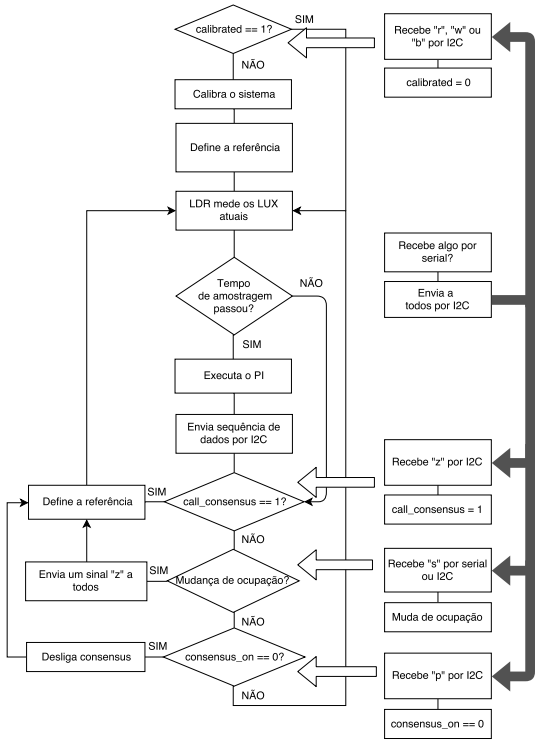

Figura 14: Fluxograma principal

Voltando ao programa principal (figura 14), feita a calibração, é definida a referência do sistema. Todas as funções relaci-

onadas com o sistema de controlo estão organizadas numa classe (*PID*) que está definida em *arduino/main/PID.h* e *arduino/main/PID.cpp*. Para este caso é utilizada a função *PID::SetReference()*, que define a referência como o resultado do *consensus* caso este esteja ligado ou caso não esteja, como o valor definido de *"HIGH"* ou *"LOW"* de acordo com a ocupação. Seguidamente, o LDR da luminária mede os *lux* atuais na caixa, e caso tenha passado o tempo de amostragem fixado (25 ms), executa os cálculos de controlo PI e envia a sequência de carateres indicada na III-G necessária ao *Rapsberry Pi*.
A seguir, se a variável *call_consensus* estiver a 1, a referência é definida novamente. Esta variável está sempre a 0, a não ser que o arduíno receba por o I2C a *label* "z", que está explicada na secção anterior. Se houver uma mudança de ocupação das luminárias, é como dito, enviada essa *label* a todos os arduínos através da função *PID::SetOccupancy* e definida no próprio arduíno uma nova referência. Se o sistema de controlo distribuído for desligado, é determinada mais uma vez a referência agora sem o mesmo. Repete-se o todo processo, novamente.
Quando o arduíno que está ligado por *serial* ao *Raspberry Pi*, recebe algum carater, a mensagem é enviada a todos os arduínos por I2C, e são estas mensagens que definem valores para variáveis capazes de alterar o funcionamento do sistema com base na interação do cliente.
O sistema de controlo distribuído está definido como um método da classe, nomeadamente *PID::Consensus()*. As operações de matrizes estão definidas à parte nos ficheiros *arduino/main/MatrixMath.cpp* e *arduino/main/MatrixMath.h*, que foram obtidos no repositório de *github* `https://github.com/eecharlie/MatrixMath`.

I. Arquitetura do software do Raspberry Pi

O *software* implementado no Raspberry Pi é composto por 3 partes principais representadas na figura 16. O processo da direita faz *sniffing* da comunicação I2C entre os arduínos, analisa a informação e escreve a que é relevante para um FIFO. O processo da esquerda está dividido em duas threads. Uma delas lê a informação do FIFO e guarda-a numa estrutura partilhada. A outra recebe ligações de clientes, e cria uma sessão para cada um.

- A **thread que aceita ligações de clientes** foi implementada usando a biblioteca de C++ BOOST [10], que permite a abstração de toda a criação de sessões por cliente aceite quando usada assincronamente. O servidor é assim capaz de lidar com pedidos de vários clientes paralelos (cada um usando o seu *socket*) e emitir as respostas respetivas para cada um. Para este fim, foram implementadas 2 classes (*tcp_server* e *conn* contidas em *raspberry/main/server.cpp*). A primeira é responsável por aceitar as ligações, a segunda, que representa a sessão do cliente, é responsável por receber pedidos e responder-lhes adequadamente. Estes últimos tanto podem ser pedidos de interação com o sistema, *sets*, ou pedidos para obter estatísticas, *gets*.

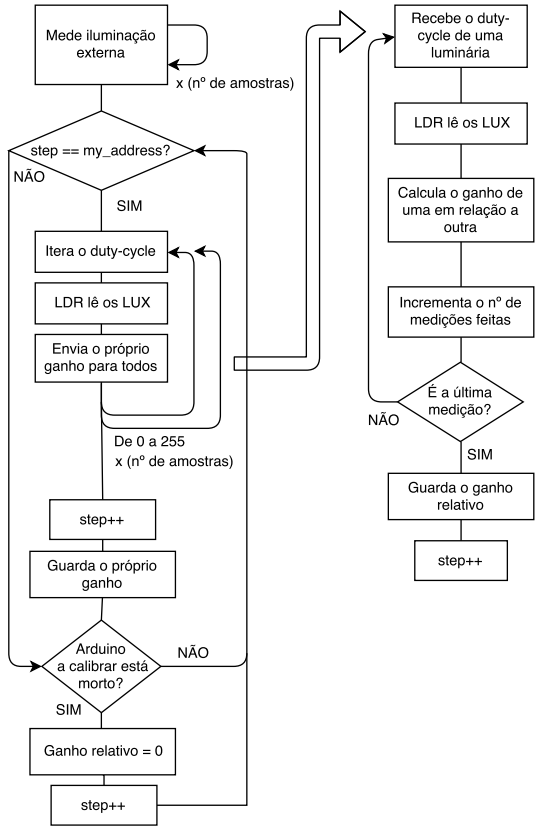

Figura 15: Fluxograma da calibração

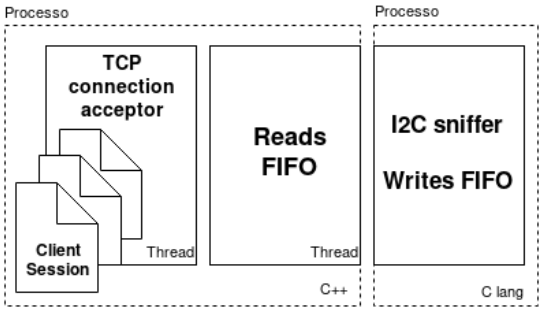

Figura 16: Arquitetura do *software* do *Raspberry Pi*

No primeiro caso, o servidor manda via *serial* (USART convertido para/de USB) um inteiro para um do arduínos que o comunica aos outros como explicado na III-G, e manda para o cliente um *acknowledgement* como prova que o pedido foi enviado ao sistema. O servidor permite que o cliente faça uma reinicialização ao sistema, defina a ocupação de cada luminária, defina se o *consensus* está a ser aplicado ou não, e defina o ângulo de abertura da janela.
Quando o cliente quer **obter estatísticas do sistema**, estas são retiradas da estrutura de dados partilhada que tem guardada essa informação desde a inicialização e adequadamente enviadas de acordo com o pedido. As respostas do servidor estão implementadas em *raspberry/main/resolve_get.cpp*). De forma, a responder

ao pedido de *streaming* a tempo real, foi implementado um temporizador que verifica de 5 em 5 segundos se há informação que deve ser enviada e caso seja esse o caso envia-a. O *streaming* pode ser composto pelo *duty-cycle*, pela luminosidade ou por ambos de arduínos diferentes ao mesmo tempo.

- A outra thread está assincronamente a **ler nova informação do FIFO** (usando a biblioteca citada acima) e a guardá-la no seu lugar respetivo na estrutura de dados. Esta informação aparece na forma de um *string*, que pode ser uma sequência de dados sobre uma luminária, por exemplo a1l100d50o1 (luminária 1 tem luminosidade 100, *duty-cycle* 50 e ocupação 1) ou uma sequência recebida apenas uma vez no fim da calibração/*consensus*, i1t50x40 (luminária 1 tem referência 50 e iluminação externa 40). O *timestamp* da sequência é originado quando esta é lida do FIFO. Esta parte está implementada numa classe (*sniff*) contida em *raspberry/mainserver.cpp*.

- O **processo que faz *sniffing*** foi implementado em C usando o código *I2C Sniffer* presente nos exemplos da biblioteca *pigpio* [11]. Uma vez que este programa corre a uma frequência mais alta que a do I2C (100kHz), é necessário determinar se o byte que está a ser recebido ainda é o mesmo ou não. Já que pode haver a possibilidade de bytes correspondentes a dados diferentes terem o mesmo valor, os arduínos enviam carateres antes de cada valor para a garantir que não se perde informação. Assim, a cada sequência de dados detetada (relativa à luminária ou ao fim da calibração/*consensus*), é escrita a mesma para o FIFO em formato *string*. O código implementado está contido em *raspberry/main/I2Csniff.c*

A **estrutura de dados** é um *vector* de objectos da classe Data (contida em *raspberry/main/data.h*) em que cada objecto representa uma luminária e é composto por 5 vetores principais (tempo, luminosidade, *duty-cycle*, ocupação e energria) e outras variáveis que ajudam nos cálculos a serem feitos. A energia instantânea é calculada assim que nova informação chega ao vetor, de forma a diminuir o tempo de computação em futuros cálculos. Dado que a estrutura é partilhada entre threads, está protegida por um *mutex*. A implementação pode ser consultada nos ficheiros *raspberry/main/data.cpp*.

IV. Experiências

De forma a analisar qualitativamente os vários aspetos do sistema, foram realizadas diversas experiências, cada uma para testar e validar os conceitos descritos na Abordagem. As experiências realizadas e as suas configurações foram as seguintes:

- **Abertura da janela e mudança de referência**. A primeira experiência realizada foi somente observar o funcionamento do sistema em duas condições usuais: abertura da janela passados a meio do controlo do sistema, de forma a observar a resposta do sistema a esta perturbação externa; e alteração da ocupação de uma das luminárias (portanto, alterando a sua referência). Foram feitas medições da iluminação, em lux, e do *duty cycle*, ambos em função do tempo para um dos nós, com ambos ligados. A análise e discussão dos resultados está presente na secção V-A.
- **Energia gasta pelas luminárias**. Mediu-se, ao longo do tempo, a primeira métrica de avaliação de desempenho do sistema: a energia gasta em cada nó. A energia das luminárias foi obtida calculando a potência instantânea e somando ao longo do tempo (supondo que 1 LED gasta, no máximo, 1 W de energia):

$$E = \sum_{i=2}^{N} d_{i-1}(t_i - t_{i-1}). \tag{14}$$

  Obtiveram-se então os gráficos de energia em função do tempo para ambas as luminárias, presentes na secção V-B.
- **Erro de conforto**. De seguida, mediu-se a segunda métrica de avaliação: o erro de conforto. Esta métrica permite quantificar a capacidade do sistema evitar descer abaixo do mínimo de iluminação, calculada usando a seguinte a expressão:

$$C_{error} = \frac{1}{N}\sum_{i=1}^{N} max(l_{referencia}(t_i) - l_{medido}(t_i), 0). \tag{15}$$

  A evolução desta métrica de avaliação está na secção V-C.
- **Variância de conforto**. Por último, mediu-se a terceira métrica de avaliação: a variância de conforto (para quantificar o *flickering*), calculada seguindo a expressão:

$$V_{flicker} = \frac{1}{N}\frac{\sum_{i=3}^{N} |l(t_i) - 2l(t_{i-1}) + l(t_{i-2})|}{T_s^2}, \tag{16}$$

  Em que $T_s = 25ms$ é o período de amostragem e l é a iluminação medida numa dada luminária. O resultado desta experiência encontra-se na secção V-D.
- **Comparação do funcionamento com e sem feedforward**. Para estudar o efeito do termo de *feedforward* em cada controlador, foram feitas medições da iluminação ao longo do tempo num controlador (com o outro desligado, porque neste caso apenas se pretendia estudar o efeito do *feedforward* no seguimento da referência de um nó), com e sem termo de *feedforward*. Foram feitas medições de iluminação, em lux, em função do tempo, cujos resultados podem ser visualizados na secção V-E.
- **Comparação entre *additive feedfoward* e *decoupled feedforward***. Estudou-se a diferença no comportamento de um controlador isolado caso a interação entre os termos *feedforward* e *feedback* seja do tipo aditiva ou *decoupled*. Foram feitas medições de iluminação ao longo do tempo, presentes na secção V-F.
- **Comparação do funcionamento com e sem anti-windup**. De forma analisar o comportamento do sistema com e sem anti-windup, realizou-se uma experiência em

que o sistema começou com a janela fechada e, de seguida, abriu-se a janela por um período de tempo de cerca de 30 s (para acumular um erro elevado no termo integral). Por ultimo fechou-se a janela para perceber o tempo que o sistema demorou a regressar à iluminação desejada. Foram feitas medidas de iluminação numa luminária, deixando a outra desligada, em função do tempo. O limite máximo imposto ao integral foi de $W = 75\ lux$ Os resultados desta comparação estão apresentados na secção V-G.

- **Resolução do problema de controlo distrubuído**. Procedeu-se à alteração do código de *Matlab* fornecido (presente no *GitHub*) para que replicasse as operações feitas por ambos os controladores e assim analisar as soluções alterando certos parâmetros. Introduzindo os ganhos e as iluminações externas obtidas nos nós, foi possível obter as diferentes soluções. Os ganhos utilizados foram: $K_{11} = 0.445$, $K_{12} = 0.033$, $K_{21} = 0.021$ e $K_{22} = 0.333$. As iluminações externas e referências dependeram do teste, como se pode ver na secção V-H. Os dois últimos testes envolveram alterar os parâmetros da função de custo.
- **Comparação entre controlo cooperativo ou não cooperativo**. Analisou-se também o impacto do controlo entre os dois nós ser cooperativo ou não, isto é, se cada um deles funcionava interpretando o outro como uma perturbação externa ou se realizavam o algoritmo de controlo distribuído descrito na secção III-F. Para isso, retiraram-se valores de iluminação e energia total gasta em ambas as luminárias, eem função do tempo, para ambos os casos, analisados na secção V-I.

V. Resultados e discussão

A. Abertura da janela e mudança de referência

Analisando os dados presentes na figura 17, vê-se que quando a janela é aberta (aproximadamente aos $4\ s$), existe uma diminuição do *duty cycle* e um maior *flickering* na iluminação. Isso é devido à acção do termo de *feedback* a tentar seguir a referência. Contudo, em termos qualitativos, o comportamento foi muito positivo pois a variação máxima da iluminação foi apenas cerca de 3% e o tempo de estabelecimento foi menos de $0.5\ s$ (considerando que a abertura da janela não é instantânea, o que ainda dificulta mais a tarefa do controlador pois encontra-se num regime transitório). Naturalmente, depois desta fase, o *duty cycle* imposto foi bastante menor para atingir a referência, fruto da iluminação externa. Cerca de $10\ s$ mais tarde, alterou-se a ocupação da luminária, o que levou à realização do algoritmo de controlo distribuído entre ambos os nós. Após a realização do mesmo, o sistema obteve a iluminação esperada praticamente em apenas um ciclo de *sampling* (ou seja, sem ter que recorrer ao *feedback*), o que comprova a qualidade do algoritmo implementado. Em relação à sobreelevação, na iluminação foi praticamente nula enquanto no *duty cycle* foi aproximadamente 5%. Mais tarde, voltou-se a colocar a ocupação dessa iluminária a 1, e a transição ocorreu rapidamente e sem *overshooting* em qualquer variável.

Um facto importante de salientar é que o *flickering* observado, a maioria é devido a imprecisões no ADC do arduino, ou seja, são variações virtuais no sentido em que o controlador está sempre a impor o mesmo *duty cycle* (porque devido à *deadzone* o controlador interpreta o desvio como deveria ser, nulo) só que na conversão da tensão aos terminais do LDR o arduino interpreta como valores ligeiramente distintos. Para corrigir esta imperfeição no futuro, o valor de luminosidade devia ser enviado recorrendo ao erro obtido no PID que, devido à *deadzone* implementada, é nulo nestas situações.

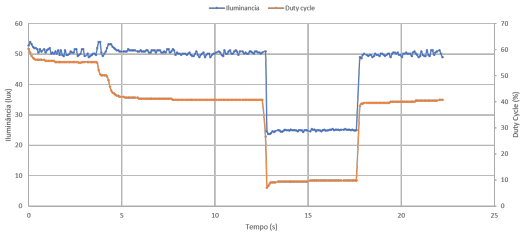

Figura 17: Abertura da janela e alteração de ocupação e respetiva resposta do sistema.

B. Energia gasta pelas luminárias

Através do gráfico da figura 18 verifica-se que, apesar dos LED's estarem ligados durante 40s, eles gastaram menos de 10J. Isto porque o duty cycle de cada LED é ajustado de forma a não gastar energia desnecessária. O facto de uma iluminária consumir mais energia que a outra deve-se ao facto do controlo distribuído ajustar, de acordo com vários parâmetros, o duty cycle de cada LED da forma mais eficiente.

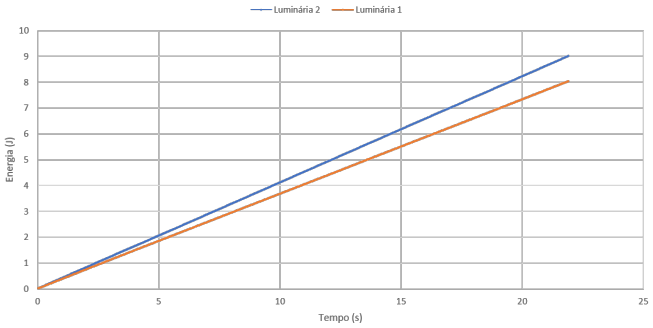

Figura 18: Energia gasta pelas luminárias em função do tempo.

C. Erro de conforto

Pode-se verificar que na figura 19 erro é bastante elevado ao inicio, pois a luz ainda está a convergir para o seu valor de referência. O erro estabiliza ao fim de 0.5s, sendo que as luminárias convergem de maneira ligeiramente diferente, pois têm valores de referência diferentes.

D. Variância de conforto

Observando o gráfico da figura 20, conclui-se que, como no erro de conforto, a variância varia bastante ao inicio. à medida que se atinge o valor de referencia, a variância

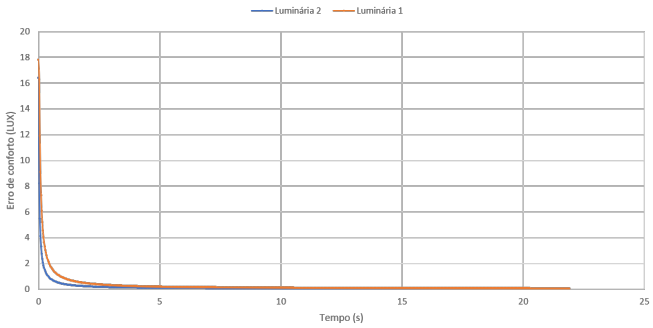

Figura 19: Erro de conforto das luminárias ao longo do tempo.

diminui estabilizando à volta de 40. Este valor engloba o *flckering* devido ao ADC descrito na secção V-A. Futuramente, a variância de conforto também devia passar a ser calculada com base no erro do controlador.

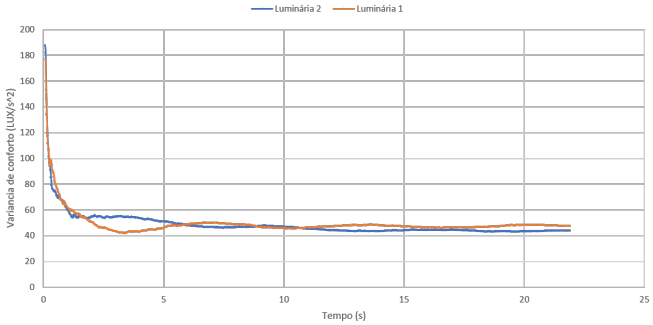

Figura 20: Variância de conforto das luminárias ao longo do tempo.

E. Comparação do funcionamento com e sem feedforward

Como se pode observar da imagem 21, a existência do termo de *feedforward* acelera a resposta do sistema pois impõe logo após o primeiro ciclo uma estimativa muito próxima do *duty cycle* que a luminária terá que impôr. A meio do teste alterou-se a referência HIGH→LOW e depois voltou-se ao estado inicial, e verificou-se que em ambas as alterações o *feedforward* acelerou a resposta. Em termos qualitativos, o tempo de estabelecimento sem *feedforward* era cerca de $0.4\ s$, enquanto com *feedforward* era apenas um intervalo de *sampling*, $0.025\ s$, muito inferior. Um aspeto negativo pode ser o aumento ligeiro de *overshooting* nalgumas situações, mas de qualquer forma é um efeito ainda menor devido à interação entre o *feedforward* e o *feedback* do controlador: *decoupled feedforward* (ver secção V-F).

F. Comparação entre additive feedfoward e decoupled feedforward

Analisando os resultados da figura 22 verifica-se que com uma interação simplesmente aditiva, o *overshoot* é muito superior ao caso utilizado (com *decoupled feedforward*). Concretizando, chegava a valores perto de 20% de *overshoot*,

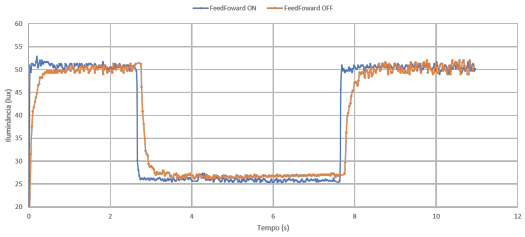

Figura 21: Comparação da iluminação ao longo do tempo com e sem termo de *feedforward* num único controlador.

contrastanto com a quase inexistência caso o *decoupled feedforward* estivesse ativo. Isso verificou-se tanto nas transições de referência HIGH→LOW como as inversas.

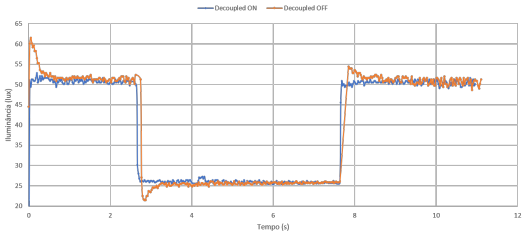

Figura 22: Comparação da iluminação ao longo do tempo com e sem *decoupled feedforward*, num único controlador.

G. Comparação do funcionamento com e sem anti-windup

Ao olhar para a figura 23 verifica-se que o controlador sem anti-windup demora 20s a regressar à iluminação desejada após o fecho da janela, enquanto que, com anti-windup ele tem uma queda acentuada, mas volta quase imediatamente para o valor de referência. Isto quer dizer que o anti-windup tem um impacto bastante importante no sistema, pois este impede que o integral tome valores elevados ao ponto de impedir o controlador de colocar a referência correta. Ou seja, o integral tem um limite máximo, possibilitando a recuperação do sistema dentro de um curto de espaço de tempo. Nota-se que a iluminação externa medida para as duas situações não foi exatamente igual como se pode ver na figura 23, porque é impossível recriar igualmente medições de luz externa tão intensa como a usada (a luz utilizada foi intensa para garantir que era suficiente para saturar o controlador). Contudo, os resultados aceitam-se na mesma pois a diferença não foi muito acentuada.

H. Resolução do problema de controlo distribuído

Para os primeiros 3 testes, utilizaram-se custos $C_i = 1$ para todos os nós e $q_{ii} = 0$. Em primeiro lugar, resolveu-se o problema mais simples: ambos os nós com a mesma referência $L = 50\ lux$ e com iluminações externas nulas. Os resultados encontram-se na figura 24. Cada figura mostra as restrições às variáveis, as curvas de nível da função de custo, a solução ótima calculada pelo *Matlab* e a solução após cada iteração do *consensus*. Os *duty cycle* nos dois controladores evoluíram

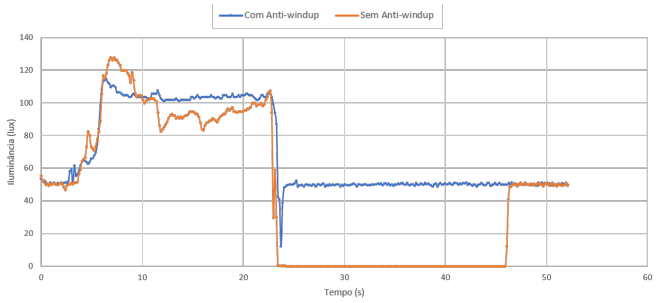

Figura 23: Comparação da iluminação ao longo do tempo com e sem windup.

para valores ligeiramente distintos devido aos ganhos serem distintos (como os ganhos relativos à luminária 2 são mais baixos, o algoritmo levou a um d_2 maior). Procedeu-se então

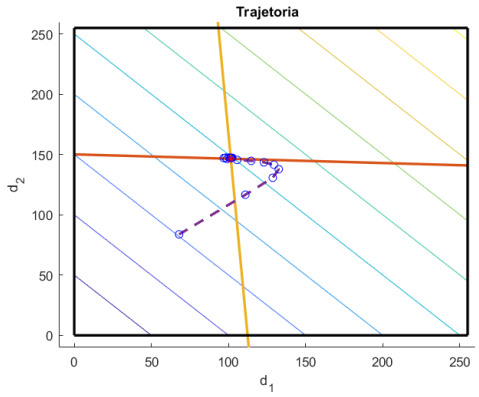

Figura 24: Solução do problema de controlo distribuído com $L_1 = 50$, $L_2 = 50$, $o_1 = 0$, $o_2 = 0$.

à resolução tendo em conta iluminações externas: primeiro, com uma iluminação externa que não fosse acima do mínimo (25) e de seguida com uma que fosse acima (26). Na primeira, foi definido $L_1 = 25\ lux$ para ver o impacto de referências distintas. Como era de esperar, no caso da figura 25 a

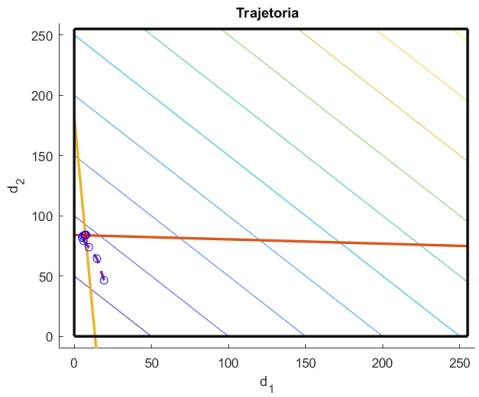

Figura 25: Solução do problema de controlo distribuído com $L_1 = 25$, $L_2 = 50$, $o_1 = 19$, $o_2 = 22$.

luminária 1 teve que acender pouco pois a iluminação externa

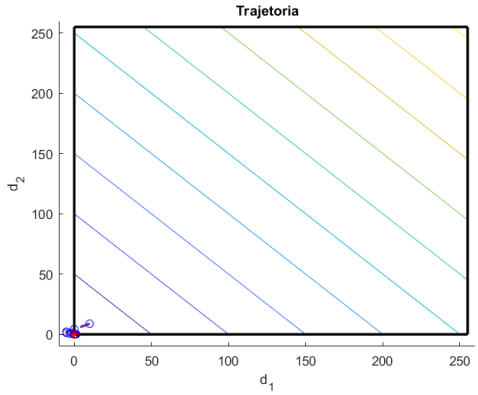

Figura 26: Solução do problema de controlo distribuído com $L_1 = 50$, $L_2 = 50$, $o_1 = 70$, $o_2 = 70$.

era quase igual à sua referência mínima, e no caso da figura 26 ambos os *duty cycle* tenderam para zero pois não era necessário acender os LEDs.

De seguida, aumentou-se o custo c_1 para 4 e encontrou-se a solução, presente na figura 27. Como se pode ver, esta alteração levou a um aumento do *duty cycle* da luminária 2 face à luminária 1 na solução final, o que faz sentido pois tinha um custo menor. Salienta-se a diferença nas curvas de nível, agora com inclinação diferente face aos testes anteriores.

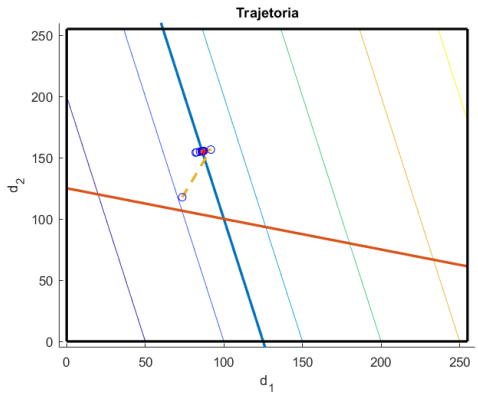

Figura 27: Solução do problema de controlo distribuído com custos lineares distintos.

Por último, testou-se a inclusão do termo quadrático na função de custo, ficando com $q_{11} = q_{22} = 0.1$. A solução deste caso encontra-se na figura 28. Observou-se que, devido ao termo quadrático ser baixo, não se verificaram grandes alterações na solução ótima obtida.

Para o funcionamento normal do sistema, utilizaram-se $c_1 = c_2 = 1$ e $q_{11} = q_{22} = 0$. Futuramente, seria positivo analisar valores para utilizar no termo quadrático da função de custo. Em relação ao número de iterações, utilizaram-se somente 10 iterações dado que era suficiente para o sistema convergir nas situações testadas, e era importante reduzir o tempo que os arduinos demoravam a realizar o algoritmo uma vez que crescia exponencialmente com o aumento da rede, caso esta crescesse. Contudo, para uma maior robustez, seria preferível colocar uma condição de paragem em vez de um

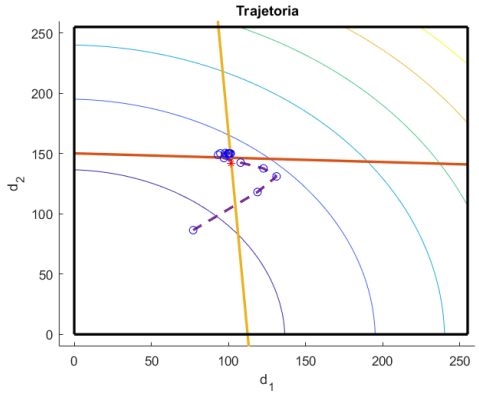

Figura 28: Solução do problema de controlo distribuído com custos quadráticos.

número de iterações máximas, como por exemplo, a função de custo diminuir menos de um $\delta << 1$ entre as duas últimas iterações.

I. *Comparação entre controlo cooperativo ou não cooperativo*

Como se pode retirar da figura 29, onde estão presentes as iluminações para um controlador, sem controlo cooperativo, existe um *overshoot* de cerca de 8%, enquanto que fazendo o algoritmo de controlo dstribuído é quase inexistente. Isto faz sentido porque caso um nó esteja ignorante da existência de outros e da sua influência, é natural que coloque mais iluminação por si. Contudo, quando a outra luminária acende o seu LED, devido aos efeitos acoplativos, isso vai levar a um excesso de lux face ao necessário. Embora esta sobreelevação seja rapidamente corrigida pelo termo de *feedback* do controlador, é notório que para o sistema é preferível um controlo cooperativo porque além de anular o *overshooting* ainda diminui o *flickering* e o tempo de resposta. Além disso, observando

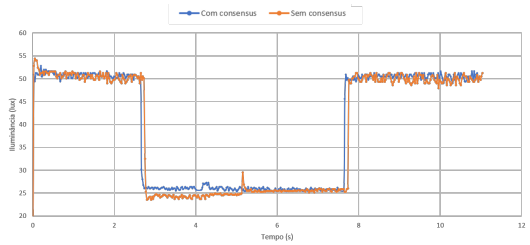

Figura 29: Comparação da iluminação ao longo do tempo com e sem controlo distribuído.

o gasto total de energia para os dois casos presente na figura 30, verificou-se que sem controlo distribuído o sistema gastou mais 1.2J ao fim de 20 segundos. Isto deve-se ao explicado atrás: quando um nó não sabe da existência do outro, coloca iluminação a mais, o que se refletiu no consumo maior de energia.

VI. Conclusões

Com a implementação e estudo do sistema de para controlar luminárias foi possível verificar que todas as opções escolhidas

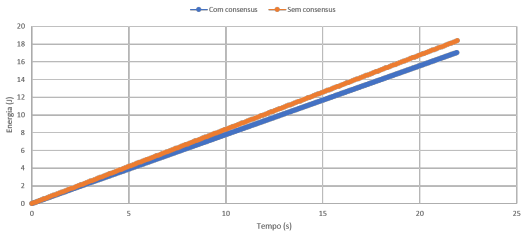

Figura 30: Comparação do consumo de energia total ao longo do tempo com e sem controlo distribuído.

ao longo da realização do controlador têm efeito no sistema e no seu desempenho.

Uma das características mais importantes é o controlo distribuído, que tendo em conta uma certa função de custo, escolhe a intensidade de cada lâmpada da forma mais eficiente possível. Face ao controlo não cooperativo, este acaba por reduzir os gastos de energia e ao mesmo tempo reduz o *overshoot* cada vez que há troca de referência, ou seja, também leva a um menor *flickering*.

Para além do algoritmo de PID básico, outras ferramentas foram implementadas de forma a que o sistema reagisse melhor a mudanças de referência ou de iluminação externa, como por exemplo o *decoupled feedforward* e o anti-windup. Todas estas implementações acabam por ter influência num maior conforto para os utilizadores.

Relativamente ao servidor criado, destaca-se a sua capacidade de armazenar dados uma vez que, de acordo com a sua memória de 1GB, estimou-se que era capaz de guardar a informação vinda do arduino durante cerca de 180 horas, ou seja, cerca de uma semana.

Contudo, vários aspetos do sistema ainda podem ser aperfeiçoados. Nomeadamente, as comunicações por I2C para fornecer informação do sistema ao Raspberry Pi podiam ser aceleradas retirando todas os caracteres auxiliares para leitura e armazenamento pela função de *sniff*. Caso o sistema fosse constituído por um número muito grande de arduinos, todas estas comunicações podiam ultrapassar a frequência de amostragem. Outro aspeto a melhorar seria criar uma *thread* apenas para o *stream* para diminuir o tempo máximo de resposta aos clientes.

VII. Contribuições

A. *Projeto*

Carlos Aleluia, 81038:

- Algoritmo de PID e respetivas funcionalidades;
- Calibração do sistema conjunto;
- Algoritmo do *Consensus*;
- Comunicações Arduino-Arduino (I2C);
- Comunicações Raspberry Pi-Arduino (Serial) - receção no arduino.

Filipe Madeira, 81076:

- Identificação do sistema;
- Algoritmo de PID e respetivas funcionalidades;
- Comunicações Raspberry Pi-Arduino (Serial) - receção no arduino;

- Armazenamento de dados no Raspberry Pi (classe Data);
- *Stream*.

Mariana Martins, 80856:

- Algoritmo de PID e respetivas funcionalidades;
- Servidor
- *Sniff* e FIFO;
- Comunicações Raspberry Pi-Arduino (Serial) - envio pelo Raspberry Pi;
- *Stream*.

Nota-se que, embora cada aluno tenha áreas onde tenha contribuído mais do que noutras, a maioria do trabalho foi feita na presença de todos os elementos do grupo pelo que todos estão familiarizados com o projeto inteiro.

B. Relatório

Carlos Aleluia, 81038:

- Calibração do sistema;
- Controlo Distribuído e Otimização;
- Comunicação I2C;
- Experiências;
- Resultados;
- Conclusões.

Filipe Madeira, 81076:

- Configuração do sistema;
- Calibração do LDR;
- Identificação do sistema;
- Controlador;
- Resultados;
- Conclusões.

Mariana Martins, 80856:

- *Abstract*;
- Introdução;
- Arquitetura de software do Arduino;
- Arquitetura de software do Raspberry Pi;
- Edição do vídeo;
- Conclusões.

REFERÊNCIAS

[1] Diffen: Incandescent Bulbs vs. LED Bulbs
https://www.diffen.com/difference/Incandescent_Bulbs_vs_LED_Bulbs.
Acedido a 30 de Dezembro de 2017.

[2] Diffen: CFL vs. LED Bulbs
https://www.diffen.com/difference/Fluorescent_Bulbs_vs_LED_Bulbs.
Acedido a 30 de Dezembro de 2017.

[3] D. Caicedo, A. Pandharipande Distributed Illumination Control With Local Sensing and Actuation in Networked Lighting Systems. IEEE Sensors Journal, Vol. 13, Nº 3, Março 2013.

[4] D. Caicedo, A. Pandharipande Daylight integrated illumination control of LED systems based on enhanced presence sensing. Energy and Buildings 43 (2011) 944–950, 2010.

[5] C. Fernandez, D.R. Distributed smart lighting systems : sensing and control. Technische Universiteit Eindhoven DOI: 10.6100/IR774336, 2014.

[6] A. Bernardino. Solution of Distributed Optimization Problems: The consensus algorithm. 2017

[7] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato e Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers.. 2010

[8] Arduino Playground - MatrixMath.
http://playground.arduino.cc/Code/MatrixMath.
Acedido a 30 de Dezembro de 2017.

[9] J. Valdez, J. Becker.
Understanding the I2C Bus. 2015

[10] BOOST C++ Libraries.
http://www.boost.org/.
Acedido a 30 de Dezembro de 2017.

[11] pigpio library Examples.
http://abyz.me.uk/rpi/pigpio/examples.html.
Acedido a 30 de Dezembro de 2017.