Sistemas de Controlo Distribuído em Tempo Real

Controlo Luminoso Distribuído

João Ramiro José Miragaia Diogo Góis
{ joao.ramiro, jpedromiragaia,diogo.gois}@tecnico.ulisboa.pt
Instituto Superior Técnico

***Abstract*—Neste projeto é proposto um sistema distribuído que resolve o problema do controlo automático e eficiente de iluminação em salas com vários postos de trabalho. Através de um sistema de luminaires com uma fonte de luz e um sensor da mesma é controlada e monitorizada distribuidamente, de forma a reduzir a energia total consumida. Ao mesmo tempo o sistema é monitorizado por um servidor capaz de relatar informações a clientes que se conectem.**

I. Introdução

Nos dias de hoje, a minimização do consumo de energia em sistemas de iluminação em escritórios é um tópico bastante abordado. Para isto, são estudadas várias estratégias com o intuito de achar a opção mais eficiente e com menor custo de manutenção que leve a um sistema de controlo de iluminação mais aceitável e inteligente. Tem-se notado que estratégias de manutenção distribuídas e automáticas são mais privilegiadas a um nível pessoal visto que apresentam melhores recomendações dos níveis de luz. Assim é necessário conseguir conciliar iluminação existente em cada área de trabalho (secretária) e o consumo de energia.

De modo a implementar o sistema, foi simulada esta sala à escala através de uma caixa como representado em figura 1, onde dentro se encontravam díodos de emissão de luz (LEDs), servindo de fonte de iluminação pois sistemas baseados na utilização de LEDs apontam uma grande capacidade de conservar energia devido à sua facilidade e flexibilidade de afinar através do duty-cycle. Neste trabalho, são tomadas em consideração fontes de luz externas ao sistema e a ocupação das secretárias de modo a minimizar o consumo de energia.

Na secção II, descreve-se a metodologia, detalhando cada fase do projeto. As experiências realizadas de forma a confirmar todo o trabalho realizado e encontram-se na secção III. Por fim existe uma conclusão na secção IV.

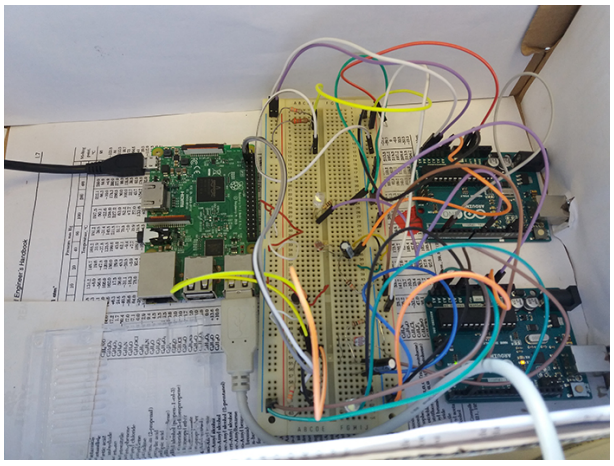


Fig. 1: Simulação de sala, com duas secretárias, luminaires, e servidor.

II. Abordagem

A. Visão geral

O sistema implementado tem como objectivo controlar a intensidade de luz numa sala com diversas secretárias (espaços de trabalho), podendo também monitorizá-las. O controlo é feito através de luminaires que comunicam entre si de modo a encontrar a maneira mais eficiente de iluminar a sala, e as secretárias que esta contém.

A cada uma das secretárias vai estar associado um custo de energia por utilização da lâmpada e um custo de desgaste da mesma, por isso para além de se satisfazer os valores de iluminação em cada uma das secretárias deverá também ser procurado a solução que leve a um menor custo físico.

Assim, é possível controlar as intensidades de luz em cada um dos luminaires através de *dimming* de os LEDs presentes nestes. Assim, vão ser escolhidas os valores de intensidade de luz em cada uma das secretárias tendo sempre em conta a potência consumida em cada LED e o desgaste a cada LED, encontrando intensidades que satisfazem as requisitos do sistema com o menor custo possível.

Cada secretária irá então estar associada a um luminaire descrito na figura 2. Além disso este sistema é monitorizado através de um servidor que envia comandos e recebe informações para os luminaires e que pode ser acedido remotamente por wi-fi.

Para satisfazer as condições apresentadas,(fornecer a luminosidade pedida em cada uma das secretárias) é necessário criar um sistema de controlo da actuação do LED, que mantém a luminosidade medida próxima do valor requerido. Para isso, em cada luminaire existirá um controlador responsável por concretizar isto. Para que a solução encontrada seja a de menor custo é necessário saber quais os valores dos duty-cycle que minimizam os custos, sendo estes os pontos de partida para o sistema de controlo (consensus). Por fim, para que seja possível mudar ou consultar as condições actuais do sistema, ocupação, mais concretamente os valores de iluminação necessária em cada um dos espaços será adicionado um esquema de comunicações que tem como objectivo ter um acesso fácil a todas os elementos que pertencem ao sistema, e confirmar que está a funcionar mediante o esperado através de um cliente que se consegue conectar a ele.

B. Luminaire

Neste projeto um luminaire é constituído pelo seguinte esquema representado na figura 2. Neste esquema os principais componentes são um díodo de emissão de luz (LED) com a função de ser a fonte de iluminação, um arduino que serve de controlador principal para esse LED e uma fotoresistência LDR que varia em função de luminosidade que nela incide incide. Nota-se ainda a presença de outros componentes como um condensador com o papel de filtragem (filtro passa baixo) e duas resistências, uma de 10kΩ e outra de 100kΩ. A resistência de 100kΩ detém um papel importante no sistema visto que protege o LED, limitando a corrente que passa neste. A resistência de 10kΩ é dimensionada de forma a que ao variar a iluminação, seja observável uma variação clara da tensão à entrada do pino analógico 1. Assim esta resistência deve ter um valor semelhante ao do LDR quando este está exposto a níveis de iluminação normais.

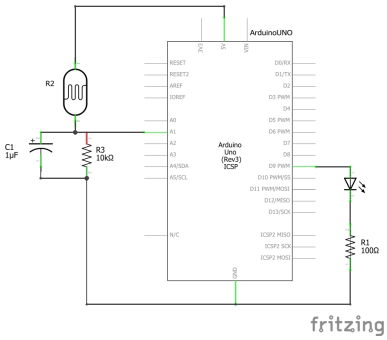


Fig. 2: Circuito de luminaire e sensor de luz.

Começou-se por obter a tensão no LDR. Para isto converteu-se de o valor obtido pelo adc de 10-bit ou seja de [0,1023] para um intervalo de [0,5] Volt. Ignorando os efeitos do condensador e visto a R1 ser conhecida foi calculado através de um divisor de tensão existente entre o LDR e a resistência, o valor de resistência atual do LDR.

Por fim, para converter o valor desta resistência para luxs, visto que a iluminância e a fotoresistência relacionam-se logaritmicamente, como se pode ver na figura 3, usou-se a seguinte fórmula para calcular os luxs:

$$\log_{10} luxs = \frac{\log_{10}(R2) - 4.8451}{-0.7186} \quad (1)$$

Em que R2 corresponde ao valor calculado anteriormente para esta resistência e os outros dois valores retirados da reta do gráfico da figura 12 proveniente de [2].

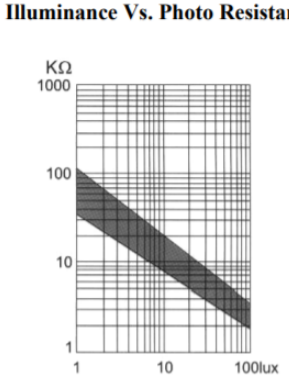


Fig. 3: Relação entre iluminância e a resistência do LDR.

C. Esquema I2C

Na figura 4 pode ser observado as ligações feitas relativas ao bus I2C.

Como se pode observar no esquema em 4, estão identificados o bus i2c, os pins ao quais o bus deve estar ligado tanto no raspberry pi como nos arduinos, e os componentes necessários para o bom funcionamento do bus. São usadas as resistências Rp1 Rp2, Rp3 e Rp4 como resistências de pull-up e estas têm valores baixos (todas de 3.3kΩ) de forma a impedir que o bus aja como uma antena quando não está a transmitir informação devido as pinos estarem em alta impedância neste caso, mas não tão baixos que estejam a consumir muita energia. Além disso estes valores afetam a velocidade das comunicações e por isso devem estar bem dimensionadas.

Os transístores são level-shifters usados para proteger o raspberry pois o logic high dos arduinos é 5V enquanto que o logic high do

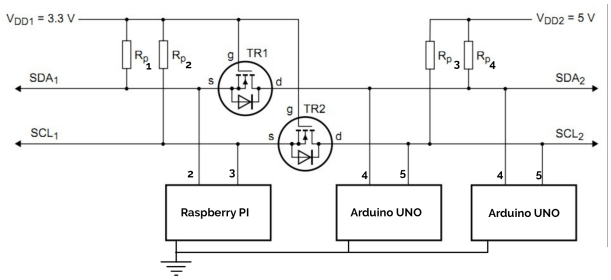


Fig. 4: Esquema i2c e com level-shifting.

raspberry é 3.3V. Desta forma quando os pinos SDA1 e SCL1 estão a low a source está a low, o gate a 3.3V e assim como $V_{gs} > V_{th}$ o mosfet vai estar na saturação e por isso o canal do mosfet abre permitindo a passagem de corrente do drain para a source convertendo efetivamente o sinal de 5V para 3.3V.

Para a informação passar da parte 3.3 para a parte 5v do bus, o bus de 5V está low e por isso a corrente (informação) passa a partir do díodo de corpo

Os endereços de cada um dos arduinos é especificado mediante a tensão que se encontra num dos seus pinos. Nesta situação foi dado o endereço 1 se nesse pino se a tensão era 0V e o endereço 2 se a tensão fosse de 5V. Este tipo de verificação encontra-se no código pois foi apenas criado um código que serviria para todos os arduinos. Foi implementado através de shunts. Finalmente é relevante referir que o bus i2c tem um limite de dispositivos que podem estar ligados de 127 pois são utilizados 7 bits para o endereço pelo arduino, por isso desta forma apenas é possível suportar 128 secretárias.

D. Calibração

De forma a calibrar os vários arduinos é necessário que os arduinos estejam sincronizados. Para tal vai existir um protocolo de comunicação entre os dois arduinos no inicialização do sistema.

Quando um dos arduinos se inicia este envia uma mensagem por I2C que significa que está pronto a iniciar a calibração , e consequentemente o controlo de iluminação. Quando um dos arduinos tal mensagem vai responder que também está pronto. Após ambos os arduinos saberem que o outro está pronto, vão começar a calibração sincronizadamente. Por ordem dos seus endereços cada um dos arduinos vai ligar o seu LED a uma determinada intensidade alternadamente. Em todos os luminaires vai ser lido o valor de luxs e vai ser calculada a influência que o o LED de i, tem sobre o sensor de luz j,K_{ij}. Após os cálculos das influência mútuas K_{ii} e das cruzadas $K_{ij},\ i \neq j$, vai ser calculado a influência da iluminação externa com todos os LEDs desligados. Estes valores vão ser necessários para o sistema de controlo e para os cálculos originários do consensus.

Após a obtenção destes valores consegue-se calcular os valores de luminosidade de correspondência aos estados de ocupação e desocupação de cada uma das mesas. Neste caso foi utilizado $\frac{1}{2}$ da iluminação máxima no caso de se encontrar desocupado e $\frac{2}{3}$ no caso de estar ocupado. De seguida vão ser enviados estes valores para o servidor para servirem de referência nos cálculos de variância de conforto. No total o sistema demora 2s a concluir a calibração.

E. Consensus

De forma a implementar controlo acoplado foi usado um algoritmo denominado *consensus* responsável pela resolução de um problema de otimização global. A solução para este problema vai evitar que os LEDs usem a sua potência máxima, o que leva a uma maior longevidade destes. Assim, a formulação do problema é dada pela expressão 2

$$d^* = \arg\min_d \{\frac{1}{2} d^T Q d + c^T d\}, \quad (2)$$

em que d é o vetor de valores de duty-cycle ideais para os LEDs, c o vetor de todos os custos de potência e Q a matriz com os custos relativo ao desgaste dos LEDs.

Os LEDs do sistema coordenam-se, trocando mensagens entre si e calculando parte da solução que leva à minimização da função de custo. O problema apresenta ainda duas restrições que têm de ser respeitadas (ver expressão 3),

$$\begin{cases} 0 \leq d_i \leq 100, \\ \sum_{j=1}^{N} k_{ij} d_j \geq L_i - o_i, \end{cases} \tag{3}$$

onde k_{ij} é a influência da luminância do LED j na sentida na mesa i, L_i a iluminância mínima desejada na mesa i e por fim o_i o efeito da iluminação externa na mesa i.

O objetivo principal é a determinação do valor óptimo de *duty-cycle* para cada LED, respeitando as restrições. Este algoritmo usa o método de multiplicadores de direcção alternada (ADMM) que divide este problema em pequenos passos, sendo cada um deles mais fácil resolvível. O ADMM promove a convergência levando a que todos os valores d_i devam ser idênticos através do método Lagrangiano referenciado em [3].

Denote-se d o vetor de valores, que contém o *duty-cycle* obtido para cada LED i, é dado pela expressão 4,

$$d = [d_1 \quad \dots \quad d_i \quad \dots \quad d_N]^T, \tag{4}$$

onde $d_i = 100$ significa que o LED está no seu valor máximo de iluminância enquanto que $d_i = 0$ significa que este está desligado.

Este algoritmo foi utilizado em dois casos. Foi usando durante a calibração, e posteriormente, caso fossem alterados as ocupações das secretárias. Após duas iterações consecutivas do consensus, sem que houvesse alterações nos valores recebidos e os calculados definiu-se uma condição de paragem, de modo a o consensus não ocorrer *ad aeternum*. Na secção III, irá ser discutido resultados usando o controlo acoplado implementado, e sem este.

F. Controlo

Visto que o nosso sistema tem um resposta rápida, optou-se pela utilização de um controlador PI, representado na figura 5. Este controlador permite-nos evitar perturbações e ruído durante o processo de operação do nosso sistema.

Apesar do controlador PID apresentar um controlo óptimo e um erro zero em estado estacionário (zero steady state error), neste caso não foi necessário garantir a eliminação de overshoot, através da componente de ganho derivativa, do sistema visto que os LEDS apresentam uma resposta rápida e flexível, e esta componente derivativa ser utilizada em sistemas com atrasos.

A função de controlo do sistema baseou-se na gestão de luminosidade do LED através de processos como *feedforward*, o *feedback* e *anti-windup*. Atribuindo uma certa referência a cada luminaire, são lidos os últimos valores de luxs para essa secretária e é aplicado um filtro passa baixo. Faz-se assim a média destes valores, obtendo resultados menos ruidosos. Primeiro é calculado o erro do sistema através da seguinte expressão 5.

$$e(t) = r(t) - y(t), \tag{5}$$

Sendo $r(t)$ a luminosidade de referência para o sistema e $y(t)$ a luminosidade medida. Este erro é então introduzido num controlador para o cálculo das componentes proporcionais e integrais através do esquema presente na figura 5 de modo a ser obtido o valor pwm correspondente. Caso o feedforward esteja ativo, o pwm correspondente ao lux atual irá também ser adicionado. Esta parte não está presente na figura 5 que apenas é demonstrado o loop de feedback.

Assim, caso a luminosidade medida é mais baixa que a desejada, o erro será positivo, e os coeficientes do controlador levam a um valor positivo de pwm, ou seja, a que o LED se torne mais brilhante. Para conseguir reduzir este erro, de forma a alcançar o menor erro possível entre os valores de referência e medidos, o controlador acaba por aumentar a luminosidade medida. Também foi implementado anti-windup que permite a componente integradora seja de certa forma mais estável e não integre para infinito quando o sistema apresentar valores de luminosidade acima dos que o sistema consegue controlar, ou seja quando a caixa é aberta. Esta *feature* faz então com que o sistema volte mais rapidamente à referência como será posteriormente demonstrado na análise de resultados.Para que o sistema funcionasse à velocidade mais elevada possível desde o momento em que é lido o valor de luminosidade presente no LDR até ser enviado o valor de tensão a aplicar ao LED, apenas estão incluídas operações relativas ao cálculo do valor de tensão a aplicar, desta forma a que o valor calculado seja o mais actual possível, se existir muito tempo entre a leitura e a actuação, as condições do sistema podem ter mudado e assim a resposta calculada não é a mais adequada.

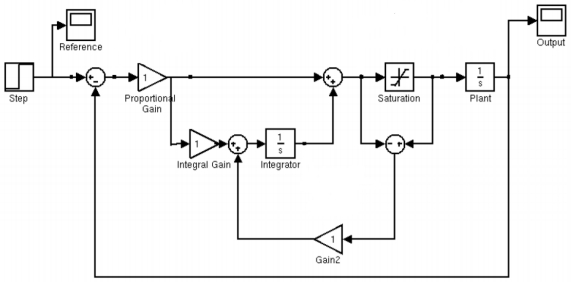


Fig. 5: Diagrama de blocos do controlador do sistema.

No diagrama 5 está explicito o esquema correspondente à implementação digital no slide 16 do cap 9 de [4]. Para encontrar todos os ganhos e constantes referidas neste slide foi usado o método de resposta ao escalão de Ziegler Nichols[7]. A resposta ao escalão do sistema obtido está representada em 6.

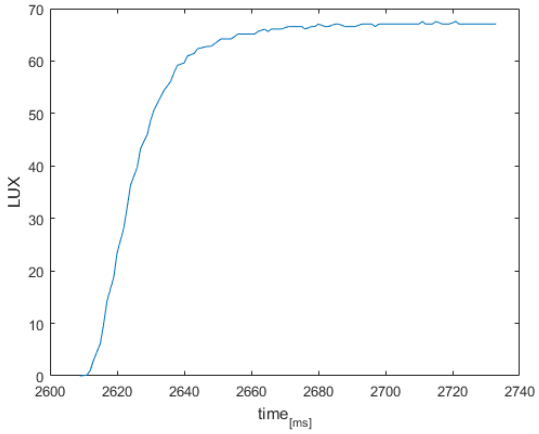


Fig. 6: Resposta do sistema ao escalão

É importante notar que tem todo o sistema de controlo foram usados luxs em vez de tensão devido á relação entre pwm e lux ser aproximadamente linear (figura 8) face à relação entre pwm e tensão que é logaritmíca (figura 7).

G. Servidor

O servidor é um programa que corre num raspberry pi 3 cujo propósito é mostrar aos clientes que se conectam a ele informações provenientes do sistema(arduinos+raspberry) e adicionalmente receber comandos dos tais clientes de modo a fazer alterações no sistema. Este servidor é um programa c++ que corre assincronamente no raspberry e que se conecta aos clientes via TCP. Este está também

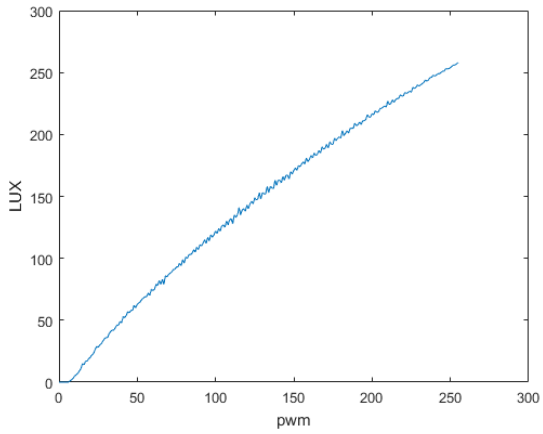


Fig. 7: Relação entre *tensão* e *pwm*

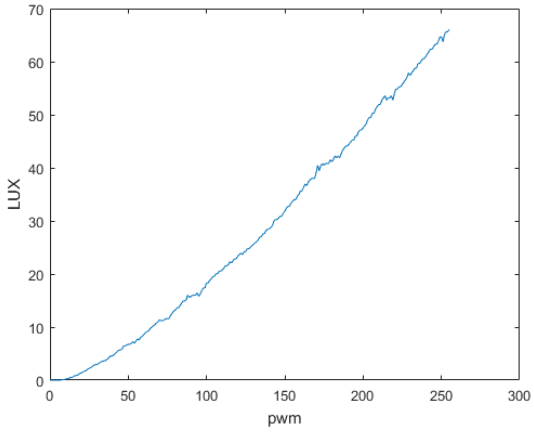


Fig. 8: Relação entre *lux* e *pwm*

ligado a um dos luminaires via serial(UART) e ao bus de I2C comum a todos os luminaires. Este tem nele guardadas informações acerca do sistema, que permitem informar o cliente sobre os seguintes aspetos:

- Valores de duty-cycle para todos os luminaire recebidos nos últimos 60 segundos, e o instante temporal correspondente;
- Valores de iluminação (lux) para todos os luminaire recebidos nos últimos 60 segundos, e o instante temporal correspondente;
- Energia consumida por cada luminaire;
- Erro de conforto de cada luminaire;
- Variância de conforto para cada luminaire;
- Instante de tempo em que o sistema se iniciou;
- Valor de iluminação externa de cada luminaire;
- Valor de referência de iluminação de cada luminaire;
- Valor de limite mínimo de iluminação de cada luminaire;
- Valor de ocupação de cada secretária;

1) Comunicações com os luminaires: O servidor comunica com os luminaires com o propósito de monitorizar o estado do conjunto de luminaires ou para efetuar alterações neste.

Através da ligação Serial feita do servidor para um dos arduinos (constituinte de um luminaire) é possível para enviar comandos. Através de um i2c sniffer o servidor é obtém a resposta dada pelos luminaires.

Os comandos enviados para os luminaires têm como objetivo mudar a ocupação das secretárias, reiniciar o sistema ou pedir informações. Todos estes comandos são mostrados na tabela V com o seu respetivo significado. É importante de notar que apenas os tipo expressos na tabela VIII requerem pedidos ao sistema, sendo que medidas como o duty-cycle e o luxs na secretária estão constantemente a ser enviados para o raspberry via I2C de modo a poder, calcular as grandezas de energia, potência, variância conforto e erro de conforto que serão faladas na em **??**, pois estas são diretamente derivadas destes outputs. Como referido anteriormente, para receber informações provenientes dos luminaires o raspberry está ligado ao bus i2c e, através de um i2csniffer (ver subsecção II-G4), este consegue descodificar a informação enviada no protocolo i2c entre arduinos. Desta forma consegue receber informações de todos os luminaires no sistema.

As mensagens enviadas entre o cliente e o servidor são depois trocadas consoante o protocolo especificado na seççao II-H. As classes mais importantes utilizadas foram a boost/asio Chrono thread time

2) Funcionamento do servidor: O servidor foi escrito em c++ e no seu core está o uso da biblioteca boost [5], sendo que são usadas funções assíncronas de read write e timers provenientes da parte asio desta biblioteca, de modo a permitir ao servidor, conseguir lidar com vários clientes ao mesmo tempo. Além disso o servidor tem duas threads para assegurar o paralelismo de duas tarefas. Uma thread trata das comunicações com os clientes e por serial para um dos luminaires(*io thread*) e a outa trata de ler os dados provenientes do busi2c através do i2c sniffer(i2c thread). Além disso existe uma classe onde são guardados e atualizados.

3) Thread io: Esta thread é a responsável por servir os clientes, ou seja processa a informação que estes enviam, e altera o sistema, o responde-lhes com a informação requerido.

Inicialmente esta thread, está à espera de que um cliente se conecte, e para está em modo *accept*. Para estes se ligarem ao servidor este devem estar num IP e porto bem-conhecidos que neste caso são 10.0.0.1:10000, que corresponde ao ip atribuído ao raspberry, e o porto 10000 neste utilizado pelo programa.

A comunicação com os cliente é feita através de TCP (Transmission Control Protocol) que é usado para transmissão de packets na internet. Este protocolo é connection-oriented e garante que os packets chegam ao seu destino (ao contrário do UDP que não tem essa garantia e é connectionless) o que é ideal para o este sistema.

Assim o servidor faz *listen* no seu *welcome socket* por pedidos de conexões por parte de clientes. Quando há um pedido de conexão, o servidor cria um socket para este cliente, de modo a poder continuar a ficar a espera de pedidos no *welcome socket*.

Ao processo processo inicial de estabelecimento de conexão, dá-se o nome de *three-way handshake* 9 onde são trocados pacotes de modo assegurar que a ligação fica bem estabelecida.

É também aberta uma porta série a 115200 Baud de modo a poder enviar dados para os luminaires.

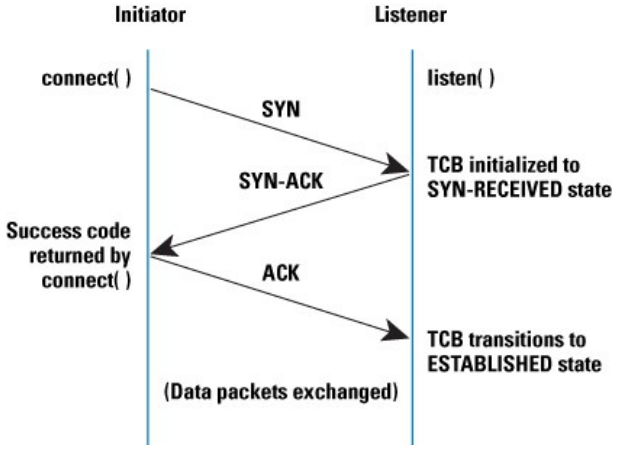


Fig. 9: Handshake no início de uma ligação TCP

Após a ligação do cliente, a thread entra no loop de comunicação básico, em queo servidor está sempre a escutar se o cliente manda alguma mensagem, e se esta for enviada, o servidor responde (write)

e depois volta para o estado de escutar(read). Todas estas operações são feitas assincronamente através das funçãoes *async_write()* e *async_read_some()* providenciadas pela *boost::asio* de modo a que a thread não fique parada em ações que demorem tempo (o que aconteceria se o servidor implementado fosse síncrono), permitindo assim responder a vários clientes ao mesmo tempo. Desta forma, várias tarefas podem correr enquanto uma outra antiga está também a ser executada, sendo que quando esta é concluída é chamado o *callback* correspondente.

Assim neste loop de comunicação. O servidor está a espera de informação proveniente do cliente. Quando a recebe, esta passa por uma função de validação onde é detetado a que ação corresponde o comando pedido. Este poderá estar a pedir informação ao sistema, pedir para fazer reset deste ou ainda para mudar a ocupação de uma das secretárias. Todos estes comandos estão descritos na tabela VI. Assim dependendo do comando recebido o servidor irá reagir de forma diferente.

Caso o comando seja de mudar a ocupação ou reiniciar o sistema, o servidor vai simplesmente reencaminhar esta mesnsagem para o sistema de luminaires via a Serial. Se o cliente pedir informação e esta já estiver guardada no servidor, este lê-a e envia-a para o cliente. No caso em que esta não esteja presente, o servidor faz um pedido aos luminaires, que após processarem este pedido envia a informação requerida de volta para o servidor via i2c, através da outra thread a correr paralelo, onde este a processa, guarda-a e envia-a para o cliente.

É importante notar que o cliente está, a cada 10 segundos, a enviar mensagens chamadas de *heartbeat* que servem para demonstrar ao servidor que está ainda ativo. Análogamente o servido deve responder a esta resposta, devido a manter a ligação ativa *Keep Alive*. Isto é necessário devido a um deadline_timer presente num cliente que desliga automaticamente o cliente caso não exista troca de mensagens com o servidor nos últimos 60 segundos.

Um caso de pedido informação que falta especificar, é o pedido de *streaming* de certo parâmetro. Com este pedido o cliente requer que o server lhe esteja a mandar continuamente informação relativa ao duty-cycle ou à iluminação presente em cada uma das secretárias. Neste caso foi implementado um *timer* que caso a funcionalidade de *streaming* esteja ativo, a cada 1 segundos envia a informação pedida pelo cliente. É importante referir que é possível o stream de vários parâmetros ao mesmo tempo para o mesmo cliente e mesmo até para clientes diferentes.

De modo a poder fazer gráficos dos dados de iluminação (lux) e duty-cycle recebidos, é necessário guardar esse dado e o instante de tempo a que corresponde num ficheiro. Para isso, para cada luminaire presente no sistema foram criados os ficheiros lums<endereço>.csv e duts<endereço>.csv. Esta funcionalidade apenas funciona corretamente quando apenas um cliente está ligado pois, os ficheiro são abertos, quando o cliente se conecta, e fechados quando o cliente se desconecta.

4) Thread i2c: A função desta thread é descodificar a informação que passa no bus I2C guardá-la no servidor e calcular grandezas derivadas das informações referidas.

Tudo isto é possível através de um *i2c sniffer*. Isto é um software que entende capta a atividade no bus i2c, analisando o estado do bus i2c conseguindo assim descodificar a informação transmitida pelo bus e convertê-la num formato que o servidor compreenda.

Optámos pela implementação deste *i2c sniffer* devido ao raspberry não funcionar corretamente em modo multi-master (modo em que os luminaires estão ligados). Opcionalmente poder-se-ia ter posto o raspberry a funcionar em modo de slave. Optou-se pelo *i2c sniffer* devido à fácil implementação através de código disponibilizado pela biblioteca pigpio [6]. O que um i2csniffer faz é captar a atividade no bus i2c, analisando o estado do bus i2c conseguindo assim descodificar a informação transmitida pelo bus e utilzá-la no servidor.

Como referido anteriormente, ambos os luminaires estão constantemente a mandar informações sobre o seu duty-cycle e iluminação. Isto é feito de modo a poder calcular com fidelidade vários parâmetros do sistema sendo a energia6 gasta por cada um luminaires desde que o sistema foi ligado

$$E = \sum_{i=2}^{N} d_i - 1(t_{i-1} - t_i) \tag{6}$$

A potência instântânera 7 de cada lumiraire

$$P_i = 1 * d_i(\%) \tag{7}$$

O erro de conforto 8 que permite verificar se a iluminação do sistema está acima dos níveis requeridos,

$$C_{error} = \frac{1}{N} \sum_{i=1}^{N} max(l_{ref}(t_i) - l_{meas}(t_i), 0) \tag{8}$$

e a variância de conforto 9 que verifica se a iluminação se mantém aproximadamente constante, ou seja os luminaires não piscam (*flicker*)

$$V_{flicker} = \frac{1}{N} \frac{\sum_{i=3}^{N} |l_{meas}(t_i) - 2l_{meas}(t_{i-1}) + l_{meas}(t_{i-2})|}{T_s^2} \tag{9}$$

Todos estes dados são guardados no servidor.

É importante notar que de modo a fazer a comunicação entre ambas as threads com segurança, foi necessário o uso de *mutexes* e *conditional variables*, o que estes fazem, é impedir que uma variável que esteja a ser lida, não possa estar ao mesmo tempo a ler escrita, e vice-versa.

Todo o funcionamento das threads está exibido no fluxograma da figura 10.

5) Como correr o servidor: Para iniciar o servidor, inicialmente necessitam de ser instaladas as bibliotecas [5] e [6]. Depois é correr na terminal os seguintes comandos:

$$sudo\ pigpiod\ -s\ 2$$

que ativa o os serviços de gpio do pi,

$$pigs\ no$$

que recebe um *handle* de notificação, que será o 0

$$pigs\ nb\ 0\ 0xC$$

que define em que pinos do raspberry as notificações de SCL/SDA proveem. 0xC significa que provem dos pinos 3 e 2

Após correr estes comando o programa está pronto para ser iniciado através de:

$$./server\ <Serial><flagSerial></dev/pigpio0$$

onde serial corresponde à porta que está aberta, por exemplo ACMO ou USB0. A *flagSerial* desativa as comunicações serial caso o seu valor seja $-d$ de modo a poder correr o servidor mesmo sem um luminaires. </dev/pigpio0 corresponde ao *pipe* de notificação que o i2csniffer estará a ler.

H. Protocolos

Nesta secção são descritos todos os protocolos11 utilizados.

O protocolo de comunicação existente entre o servidor e seus clientes encontra-se na tabela tabela I. Todos os comandos provenientes do clientes são compostos por 3 partes separadas por um espaço como dito em 10:

$$c_1\ c_2\ c_3 \tag{10}$$

Posteriormente, para comunicarmos do servidor para os luminaires através de uma ligação serial, foi necessário que o luminaire e a que

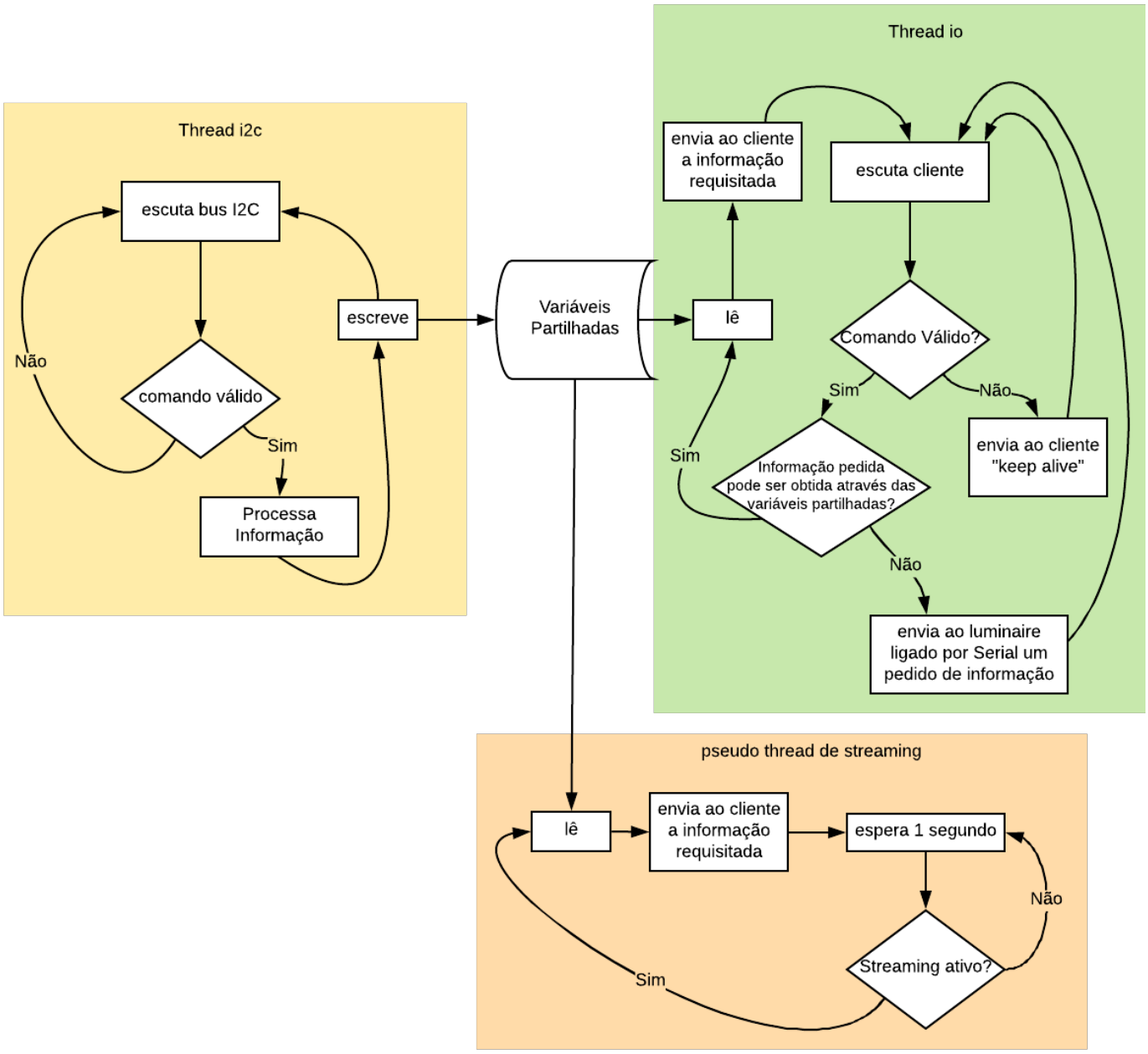


Fig. 10: Esquema de funcionamento do servidor.

Comando	c_1	c_2	c_3	Resposta
obter valor	g	tabelaVII	endereço ou 'T'	c_2 c_3 <val>
definir ocupação	s	endereço	valor de ocupaçao	ack
reset do sistema	r			ack
último minuto	b	'd'ou 'l'	endereço	b c_2 c_3 <val1>, <val2>...
começar stream	c	'd' ou 'l'	endereço	c c_2 c_3 <val> <time>
parar stream	d	'd' ou 'l'	endereço	

TABLE I: Comunicações entre cliente e servidor

caractér	Significado
L	limite inferior de lux em cada secretaria
o	ocupação da secretária
O	lux proveniente de fontes externas
r	luxs de referência

TABLE II: Tipos de dados requisitados por serial

servidor se encontra conectado interprete a mensagem proveniente do servidor e a envie para o luminaire destino. O comando enviado para o luminaire será da forma $c_1c_2c_3$, desta vez sem serem separados por um espaço, que correspondem às primeiras 3 linhas da tabela VI sendo que caso c_1 seja 'g', c_2 irá corresponder aos caracteres expresso na tabela VIII pois o servidor apenas necessita de atualizar estes tipos. Visto que os outros estão a ser constantemente atualizados.

Foi criado um protocolo para as mensagens enviadas por I2C, tendo em mente a maximização do ritmo de transmissão. Sendo que a mensagem com maior dimensão terá 6 bytes. As mensagens por I2C vão ser separadas em 3 tipos diferentes de mensagens, sendo que cada um dos tipos vai ter o seu primeiro byte como um byte identificador. (Ver tabela III)

Assim, a mensagens relativas a consensus serão formatadas segundo a tabela IV, as provenientes do servidor consoante a tabela V e as que serão enviadas para o servidor consoante a tabela VI.

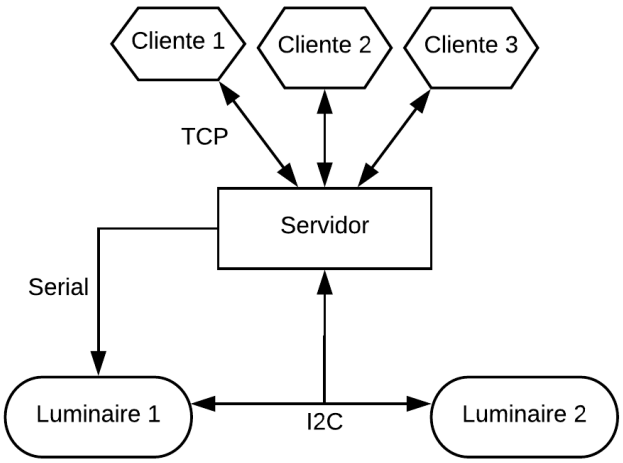


Fig. 11: Diagrama de protocolos.

1º caracter	Relativo a que tipo de comunicação
'%'	Consensus
'#'	Mensagem originária do servidor do outro arduino
'$'	Mensagem para o servidor

TABLE III: Comunicações no i2c para fazer o consensus

Vejamos o seguinte loop do comando "g L 1":

1) Cliente envia para o servidor "g L 1";
2) Servidor envia para luminaire (via Serial) "gLc".
3) Luminaire envia para luminaire destino "gL".
4) Luminaire destino envia para servidor via i2c "$gL<endereco><valor>".

É de notar que de modo a ser mais facilmente testado sem o servidor, ou seja inserido comando diretamente no serial monitor do arduino, converteu-se de índices 0,1,2... para b,c,d...

III. Análise de Resultados

Várias experiências foram feitas, de modo a verificar todos o efeitos com e sem a utilização de alguns métodos.

A. Redução do ruído

Primeiramente foi verificado a eficácia dos métodos utilizados para reduzir o ruído da medição efetuada pelo ADC do pino analógico 1 (ver Figura 2) onde entra uma tensão relacionada com a iluminação por áreas. Na figura 12

Ao comparar 12.1 e 12.2 podemos observar os efeitos positivos das médias, estas impedem que o valor oscile demasiado. A média implementada foi dos últimos 7 valores, dando igual peso a todos os valores o que gera um sinal ainda um pouco ruidoso. De notar que os gráficos em 12 estão muito ampliados e por isso o aparente ruído em 1 mesmo apesar das média não é notável para o olho humano.

Uma maneira de fazer o sinal ainda mais suave, seria incluir mais valores na média e implementar uma média ponderada. Por exemplo usando os últimos 20 valores e associando pesos maiores a valores obtidos mais recentemente, não houve no entanto necessidade do implementar, pois o sistema já funciona corretamente.

Para verificar o efeito do condensador são analisados 12.1 e 12.3 nestes apesar de não ser bem visível a inserção de um condensador no circuito, diminui a amplitude do ruído tal como esperado.

B. Feedback vs Feedback+Feedforward

Na imagem 13 pode-se verificar a reação de um luminaire perante as diversas alterações que possam ocorrer durante o seu funcionamento. Inicialmente a secretária encontra se livre, começando aos 6

Byte	Significado
0	'%' - caracter de start
1	Parte inteira do duty-cycle do outro arduino
2	Parte inteira do duty-cycle do outro arduino
3	Parte inteira do duty-cycle do próprio arduino
4	Parte decimal do duty-cycle do próprio arduino
5	'0' - carácter de end

TABLE IV: Comunicações no i2c para fazer o consensus

Byte	Significado
0	'#' - caracter de start para o servidor
1	Operação a efectuar no arduino g-get s-set r-reset
2	Se B_1 ='g': caractér na tabelaVIII. Se B_1 ='s': 1-ocupado, 0-não ocupado
3	0x00 - caractér de terminação valor a ser enviado

TABLE V: Formatação de comandos transmitidos em i2c provenientes do servidor

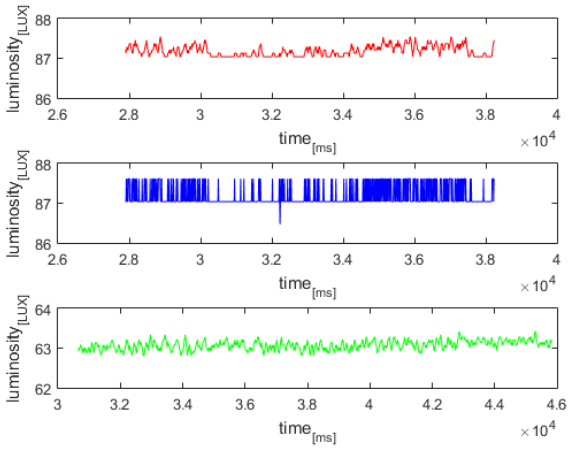


Fig. 12: 1 - Circuito com condensador e médias; 2 - Circuito com condensador e sem médias; 3 - Circuito sem condensador e com médias

em que o sistema passa a ter $\frac{1}{3}$ da luminância total que as condições permitem (estado desocupado). No próximo acontecimento, aos 9s fica ocupado tendo como referencia $\frac{2}{3}$ da luminância total , de seguida passa a desocupado , depois a tampa da caixa foi aberta deixando a iluminação externa entrar, sendo posteriormente fechada.

Através da vizualização do duty-cycle ao longo do tempo consegue-se compreender que a solução com feedback+feedforward é bastante mais rápida, que a apenas com feedback onde existe um período mais demorado de transição entre estados.

Isto pode ser melhor verificado na imagem **??** onde se pode notar as principais diferenças devido à influência da presença do feedfoward. O problema da solução de feedback+feedforward é o facto de por vezes a transição ser brusca demais originando overshoot.

C. Efeito do anti-windup

Na imagem 14 foi posto o sistema a trabalhar, após o equilíbrio a tampa da caixa foi aberta, expondo o LDR a perturbação externa, lendo um valor superior à referência, após algum tempo foi fechada a tampa, deixando o sistema sem perturbações externas, esperando até que o equilíbrio fosse restabelecido. Consegue-se notar que quando o sistema com antiwindup está implementado, após ter sido sofrido uma perturbação externa o sistema consegue voltar a valor de referencia rapidamente, enquanto que quando não está implementado, mesmo

Byte	Significado
0	'$' - caracter de start
1	Tipo de data que está a ser enviada (Ver tabela VIII)
2	Endereço de onde provém a informação
3	Parte inteira do valor a ser enviado
4	Parte decimal do valor a ser enviado
5	0x00 - caractér de end

TABLE VI: Comandos provenientes do cliente

caractér	Significado
l	luxs
d	duty-cycle
L	limite inferior de lux em cada secretaria
o	ocupação da secretária
O	lux proveniente de fontes externas
r	luxs de referência
p	potência instântanea
e	energia consumida
v	variância
c	conforto

TABLE VII: Comunicações no i2c relevantes para o servidor

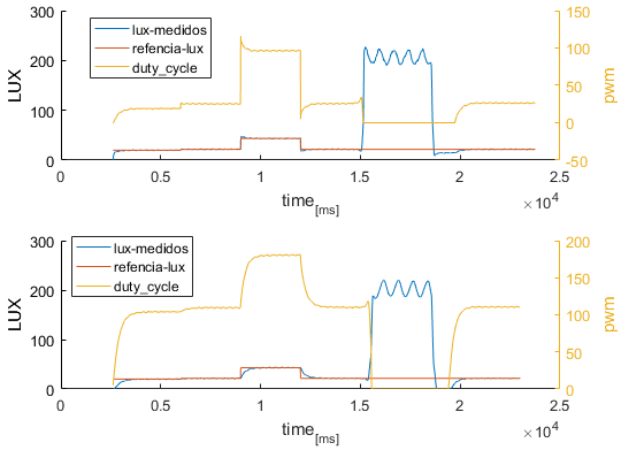
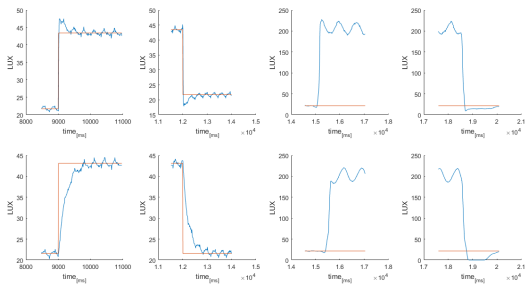


Fig. 13: Valores de luxs/duty-cicle do sistema perante perturbações, 1- com feefoward, 2- sem feedfoward ;



transições em detalhe de 13, da esquerda para a direita, estado -> ocupado, estado -> desocupado, abrir a tampa, fechar a tampa; de cima para baixo ,com feed foward, sem feedfoward

estando o luminaire exposto durante menos tempo à perturbação, o sistema vai demorar muito mais tempo a retomar a estabilidade.

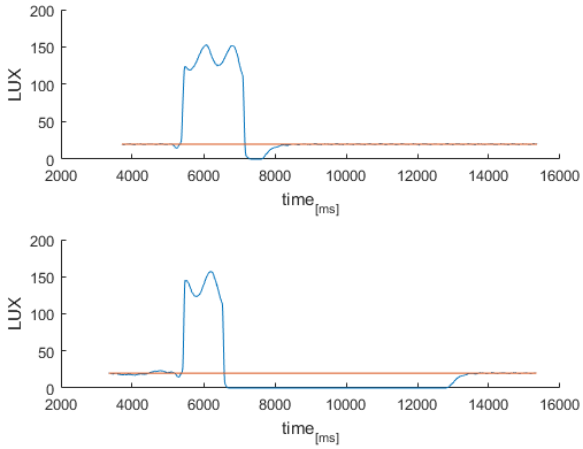


Fig. 14: Efeito da implementação do anti-wind-up; de cima para baixo:

D. Luminaires acoplados vs desacoplados

A partir das figuras 15 e 16 consegue-se notar a importância da implementação do controlo distribuído, pois quando este se encontra desativado os valores da luminância sofrem grandes alterações ao longo do tempo, existindo flickering que vai causar que o conforto do utilizador seja muito inferior.

É possível também observar que eventualmente o sistema desacoplado 16 vai atingir o mesmo equilibrio que o sistema acoplado, no entanto o tempo que demora a atingir esse estado é elevado.

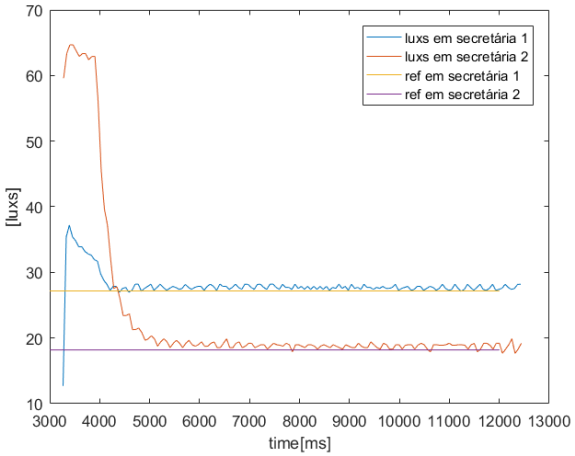


Fig. 15: Sistema Acoplado

E. Potências iguais vs diferentes

Da figura 17 entende-se que caso os dois LEDs tenham a mesma potências (1W) pode-se notar que os duty-cycles dos dois luminaires serão próximo, pois cada LED terá mais influência no LDR do seu luminaire e por isso faz sentido distribuir o duty-cycle pelos dois LEDs de modo a gastar menos energia possível. Caso uma dos LEDs tenha uma potência mais elevada(20W) pode-se verificar que o sistema quase não acende esse LED, de modo a poupar energia. Pode-se então confirmar que quando o custo de um dos LEDs é substancialmente superior ao outro nota-se que o duty-cycle do LED que tem o menor custo vai ter um duty-cicle bastante superior.

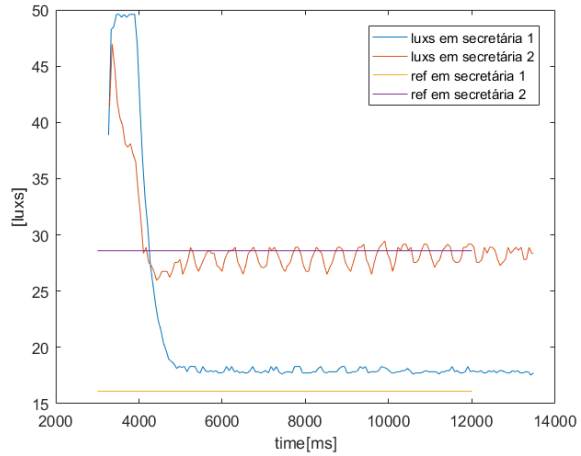


Fig. 16: Sistema Desacoplado

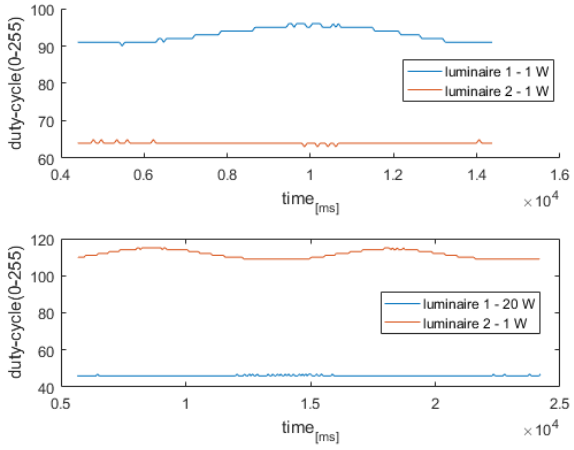


Fig. 17: 1- Ambos LEDs com potência 1W; 2 - LED 1 com potência 20W e LED 2 com 1W

F. Atrasos do sistema e limitações temporais

Um dos atrasos mais importantes do sistema é o que se dá início com a leitura da luminosidade e acaba com o envio da tensão a aplicar ao LED é de cerca de 640 microsegundos.

Para o controlo do sistema foi utilizado o relógio do arduino com precisão de 4 microsegundos que consegue contar até aproximadamente 70 minutos recomeçando a contagem nesta ocasião. Devido a no nosso caso não ter o sistema a correr durante tanto tempo, não foi necessário tomar em consideração este fator no entanto caso fosse implenentado num sistema que fosse suposto estar ligado durante dias inteiros seria necessário estar atento a esse aspecto de modo a prevenir erros.

Tempos de comunicação entre o cliente e o servidor não foram testados, no entanto este não são relevantes para o correto funcionamento do sistema, e estão bastante dependentes da localização geográfica entre o cliente e o servidor. Caso ambos o cliente e servidor estejam na mesma rede local este atraso é da ordem das centenas de milissegundo e é quase impercetível.

Outros tempos pouco relevantes para o bom funcionamento do sistema, são os tempos desde que o cliente define a ocupação de uma secretária e a luminária desta é alterada, pois desde que o tempo de resposta seja inferior ao segundo não influencia muito o erro de conforto total. Verificou-se que este rondava os 200ms.

G. Erro em regime estacionário

É possível analisar erro em regime estacionário ou (*steady state error*) através dos parâmetros de erro de conforto 8 e variância de conforto (flicker) 9 e também através da figura 15. Como se pode observar, ao atingir o regime estacionário, o nível de iluminação sempre próximo da referência e geralmente também acima desta gerando um erro de conforto que será aproximadamente 0 lux.

Em relação ao flicker, através do gráfico obtido pode-se observar que existe flicker, devido ao valor de iluminação estar sempre a flutuar. No entanto esta flutuação não é vísivel para o olho humano e por isso pode ser desprezável. Através do servidor verificou-se que a variância de conforto corresponde a aproximadamente 7000 lux/s^2 o que parece muito no entanto o que faz com que este valor seja elevado é o facto de na forma da variância, existe um termo a dividir que corresponde a T_s^2, pois T_s corresponde ao periodo do sistema (*sampling interval*) que é de cerca de 10 milisegundos. Concluímos portantos que nao se deve confiar muito nesta medida.

H. Limitações de memória do servidor

Visto que o servidor está a correr num raspberry pi que apenas tem 1GB de RAM, certas medidas foram tomadas de modo a que a memória dinâmica ocupada (RAM) não fosse elevada. Assim poucos valores foram guardados durante o programa (Ver lista em II-G). Destas informações guardadas as que poderiam escalão mais tempo e podem ocupar mais memória são as listas que guardam o duty-cycle, os lux e o instante de tempo em que cada um foi medido. Primeiramente estas listas foram limitadas apenas a valores que foram guardados nos últimos 60 segundos o que por si limita a memória utilizada. Para limitar ainda mais a memória, não foram guardados todos os valores de duty-cycle e lux medidos, mas apenas 1 em cada 5 ou seja apesar do o servidor calcular energia, e erro e variância de conforto com todos os valores de pwm e lux que recebe, apenas guardas estes mesmo valore de lux e pwm a cada 5 receções. Assim cada uma destas listas terá no máximo à volta de 1000 elementos. Assim, assumindo que todas as variáveis na lista referida em II-G ocupam 8 Bytes (o que não é verdades pois por exemplo floats ocupam apena 4 Bytes). Calcula-se que no máximo o sistema ocupe sempre menos de 5KB em memória.

Além disso são guardados ficheiros relativos a estes mesmos lux e pwm e o respetivo instante de tempo no raspberry. Para limitar a dimensão destes ficheiros o servidor apenas começa a escrever quando existe um servidor conectado. Verificou se que em 10 segundos, o ficheiro atinje 3KB, podemos extrapolar que se o programa tiver a correr durante 1hora é irá ocupar 1MB. Visto que a memória do cartão de memória utilizado é de 10GB e tem no mínimo 1GB livre, estes limites não serão atinjidos a menos que o sistema esteja a correr durante muito tempo.

IV. Conclusões

Neste projeto desenvolveu-se um programa de controlo de iluminação distribuído para utilização em gabinetes de trabalho, com o principal objetivo de minimização de consumo de energia e aumento do conforto do utilizador. Para conseguir esta minimização é detetada a iluminação atual, e é alterado a quantidade de iluminação proveninete de LEDs de forma a manter a iluminação sempre acima ou igual a níveis mínimos recomendados. No programa foram utilizados dois dispositivos contorladores/processadores principais sendo eles os arduinos e um raspeberry Pi. Toda a parte relacionada com controlo de iluminação foi tratada pelo controlador(arduino), enquanto que a monitorização foi feita através de um server desenvolvido no raspberry.

Utilizou-se vários protocolos de comunicação para alcançar o objectivo do trabalho, como o Serial e I2C, e TCP, de modo a conectar todas as partes individuais do sistema.

Foi usado o algoritmo *consensus* para o controlo do valor de iluminação, que mostrou ser um algoritmo com uma rápida convergência e robustez, no entanto, no caso do sistema apresentar mais do que dois arduinos é necessário fazer alguns melhoramentos.

Através desta estratégia consegue-se assim atingir um sistema de iluminação eficiente, que se consegue adaptar as necessidades atuais da sala em que está implementado.

V. Contribuição

Tema	João Ramiro	José Pedro Boavida Miragaia	Diogo Góis
Parte 1			
Calibração	X	X	X
Controlo	X	X	X
Leitura Serial		X	
Transformação LUXS	X		X
Parte 2			
Consensus		X	X
Protocolos	X	X	
Server	X		X
Relatório			
Introdução	X	X	X
Visão Geral	X	X	
Luminaire		X	X
Esquema i2c	X		
Calibração		X	
Consensus		X	X
Controlo		X	X
Servidor	X		
Protocolos	X	X	
Análise de Resultados	X	X	
Conclusão	X		X
Contribuição	X	X	X

TABLE VIII: Tipos de dados requisitados por serial

References

[1] Alexandre Bernardino, *Project SCDTR*. `SCDTR1718-Project.pdf`
[2] LIDA OPTICAL&ELECTRONIC COM, *LDR Datasheet*. $\texttt{LDR}_d atasheet.pdf$
[3] Alexandre Bernardino, *The consensus algorithm*. `consensus.pdf`
[4] Alexandre Bernardino, *Distributed Real-Time Control Systems*
[5] *Boost Libraries* `http://www.boost.org/`
[6] *pigpio Libraries* `http://abyz.me.uk/rpi/pigpio/`
[7] *Ziegler Nichols* `https://en.wikipedia.org/wiki/Ziegler%E2%80%93Nichols_method`