JULIANO MOSCARDINI BERNARDES

COLORÍMETRO DIGITAL PORTÁTIL

LAVRAS – MG

2012

JULIANO MOSCARDINI BERNARDES

COLORÍMETRO DIGITAL PORTÁTIL

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. DSc. Wilian Soares Lacerda

LAVRAS – MG

2012

JULIANO MOSCARDINI BERNARDES

COLORÍMETRO DIGITAL PORTÁTIL

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 31 de Outubro de 2012.

Prof. MSc. Thomaz Chaves de Andrade Oliveira	DCC-UFLA
Prof. DSc. Danton Diego Ferreira	DEG-UFLA

Prof. DSc. Wilian Soares Lacerda

(Orientador)

LAVRAS – MG

2012

A minha família

AGRADECIMENTOS

Agradeço a Deus, meu orientador e minha família.

RESUMO

Este trabalho visa a construção de um dispositivo eletrônico capaz de identificar cores de forma automatizada. Para isto são usadas técnicas de eletrônica digital, eletrônica analógica e inteligencia computacional. A implementação do controle, e dos demais algoritmos é feita em software, e programada em um microcontrolador, que atua como elemento central do dispositivo. O uso de inteligencia computacional traz maior flexibilidade ao aparelho, dando a ele a habilidade de trabalhar com situações de maior imprecisão dos dados de entrada.

Palavras-Chave: Sistemas Embarcados, Hardware, Colorimetria, Algoritmos de Classificação

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Em determinadas situações a classificação de uma cor não é uma tarefa que pode ser atribuída exclusivamente ao olho humano, devido a características como a resposta espectral de cada indivíduo e também a fatores externos como a quantidade de luz incidente e propriedades da superfície observada. Isso abre espaço para que métodos automáticos possam oferecer uma maior precisão.

Aqui é proposto uma solução eletrônica para a classificação de cores, onde técnicas de inteligência computacional são aplicadas em conjunto com um Sistema Embarcado a fim de se obter melhores resultados.

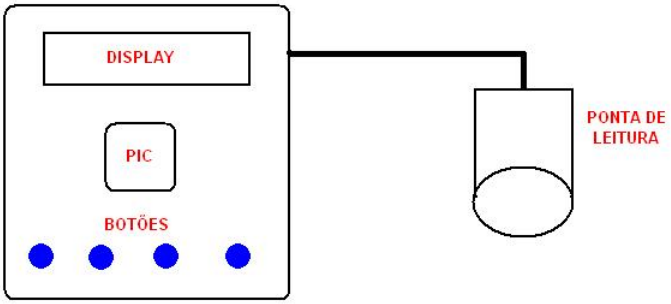

Figura 1.1: Diagrama resumido do protótipo.

A Figura 1.1 mostra o esboço do diagrama do protótipo desenvolvido ao fim da metodologia, e apresenta algumas das características mais importantes, como os quatro botões, o *display* de LCD, onde ambos são a interação com o usuário, o microcontrolador e a ponta de leitura.

Este trabalho tem base teórica nos materiais e métodos conhecidos e usados para a detecção de cores eletronicamente. Também abordando os fundamentos necessários para a construção de um equipamento eletrônico cujo objetivo é informar uma classificação de uma cor quanto a um padrão previamente apresentado, e as técnicas computacionais aplicadas para o desenvolvimento deste aparelho.

1.1 Justificativa

Existem muitos padrões de classificação de cores, sendo que alguns deles possuem um conjunto de dados amostrais muito grande, que pode dificultar a classificação somente pelo olho humano. Um exemplo é o padrão usado na disposição das cores na Carta de *Munsell*, que é considerado como um dos mais próximos do padrão perceptual humano (SOB, 2007) e sua disposição em um espaço de cores pode ser feita em termos de três componentes extraídas da resposta espectral da luz sobre uma superfície. Sabendo da existência de componentes eletrônicos que podem trabalhar como sensores da ação da luz, sendo estes conhecidos como foto-sensores, fica-se então um espaço para o emprego destes na mensuração da cor.

A construção de um aparelho classificador de cores no qual a resposta possa ser expressa em termos de um conjunto de amostras, se torna um trabalho interessante, já que esta abordagem até então é feita geralmente pelo olho humano.

A automatização do processo de classificação de cores facilitaria muito o trabalho de profissionais que dependem da leitura da cor de uma superfície em questão, e que em muitos casos não podem obter uma boa leitura devido a fatores externos, como a falta de luz, dificuldades para transporte de equipamentos, e outros. Os aparelhos utilizados para este fim de classificação de cores são conhecidos como colorímetros.

1.2 Objetivos

O objetivo principal deste trabalho é o desenvolvimento de um protótipo portátil para a classificação de cores quanto a um padrão já pré apresentado, fazendo dele um dispositivo capaz de aprender e reconhecer novas cores.

Também busca avaliar como o emprego de técnicas de Inteligência Computacional poderia ajudar a aumentar a sua precisão, facilitando assim o seu desenvolvimento.

Como em qualquer circuito eletrônico, é quase impossível se obter componentes eletrônicos precisamente iguais para o uso na fabricação, tornando necessária uma prévia calibração do equipamento recém produzido.

Oferecer um equipamento calibrado e de fácil utilização também são fatores a serem levados em conta na produção do protótipo. E buscar metodologias para permitir que seu desenvolvimento possa ser executado com recursos financeiros limitados, já que de nada adianta a produção de um equipamento de construção inviável, tanto financeiramente quanto em termos de tempo e mão de obra empregada.

1.3 Organização do Texto

O capítulo 2 mostra os dispositivos encontrados no mercado para fins próximos, e também faz um estudo avaliativo sobre os elementos a serem empregados na produção do protótipo. E apresenta algumas técnicas computacionais para o auxílio na implementação do sistema.

No capítulo 3 é apresentado como se pretende desenvolver o projeto, as ideias envolvidas na sua construção, e os resultados esperados.

Ao final, o capítulo 4 apresenta os resultados obtidos, e uma conclusão sobre o trabalho é feita no capítulo 5.

2 REFERENCIAL TEÓRICO

A classificação de uma cor nem sempre pode ser precisamente feita pelo olho humano, devido a diferentes disposições da iluminação do ambiente, e variações que possam ocorrer entre a percepção de cada indivíduo. Fica-se então em aberto um espaço a métodos alternativos que prometam, não só obter maior precisão, mas também uma padronização entre o que devemos realmente considerar como sendo de uma determinada cor.

A cor é definida por (MOTOKI *et al.*, 2007) como a sensação causada no olho humano pelo espectro óptico específico gerado na reflexão da luz em uma superfície. Logo sua classificação por um indivíduo se torna um parâmetro subjetivo. Fisicamente os comprimentos de onda que um determinado objeto é capaz de absorver, e refletir, podem ser medidos pela sua reflectância no espectro visível da luz (400 a 700 nm), e assim poder classificar sua cor quanto a um padrão. Cada cor possui um intervalo no espectro, como visto na tabela 2.1

Cor	Comprimento de Onda (nm)
vermelho	700 a 620
laranja	620 a 592
amarelo	592 a 578
verde	578 a 500
azul	500 a 450
violeta	450 a 400

Tabela 2.1: Algumas cores do espectro da luz visível (SANTOS, 2006)

Digitalmente, uma das formas mais populares de representação da cor é através do sistema RGB, proposto por Hermann von Helmholtz (1821-1894), onde cada cor é composta por três componentes, uma vermelha (R), uma verde (G) e outra azul (B) (MOTOKI *et al.*, 2007). Outra forma muito conhecida de representação é o padrão $L*a*b$, propostos pela Comissão Internacional de Iluminação (Commi-

sion Internationale L'Eclarirage (LECLAIRAGE, 1931)), onde o "L"é a componente de luminosidade e "a"e "b"são as de crominância[1], (HUNTERLAB, 2008). Existem também outros espaços para a representação das cores, como o HSL(Matiz, Saturação e Luminosidade), CMYK(Ciano, Magenta, Amarelo e Preto), HSV(Matiz, Saturação e Valor de brilho) e outros.

2.1 Colorímetros Digitais Comerciais

Colorímetros são aparelhos eletrônicos comercias cujo objetivo é mensurar a cor de um objeto ou substância. São muito usados na industria como forma de facilitar a obtenção de cores específicas de um produto. Em laboratórios de Química, por exemplo, são usados para a observação de fenômenos químicos que por ventura alteram a cor de uma substância. Inúmeras outras formas de utilização podem ser citadas, já que a coloração muitas vezes é muito significativa em vários contextos.

Existem vários fabricantes no mercado, onde o princípio de funcionamento destes aparelhos não varia muito, trata-se na maioria da casos de um sistema emissor de luz, e um de captação como células sensoras a luminosidade. Os feixes de luz são refletidos pelo objeto, incidindo sobre os sensores, permitindo a obtenção de dados, que são lidos, processados e apresentados em uma determinada escala ao usuário.

[1] Quantifica informações referentes a cor (MENDES *et al.*, 2000).

Figura 2.1: Colorímetros Digitais da empresa MINOLTA.

Em catálogos de fabricantes como a (HOLDINGS, 2011a) é possível observar características como fotocélulas de silício sendo usadas como sensores e lâmpadas de xenônio como fontes de iluminação. O tempo de leitura fica em torno de 1 segundo e são aparelhos econômicos, trabalhando geralmente alimentados por baterias. Apresentam interfaces com o usuário, como botões e displays de LCD, e muitos ainda oferecem uma gama de opções de configurações e possibilidades de visualização dos dados, como, por exemplo, a representação nos espaços de cores mais conhecidos. Alguns apresentam interfaces para comunicação com os computadores (HOLDINGS, 2011b).

Detalhes sobre a construção exata dos colorímetros comerciais são geralmente difíceis de se obter, já que as industrias tentam manter seus segredos de construção. Mas as abordagens para sua construção são semelhantes. (KARRAS; LADSON, 1984) usa um sistema onde há uma fonte luz que é emitida sobre o objeto em questão, e três filtros, um em cada fotodetector, recebendo a luminosidade refletida pelo objeto e assim obtendo três valores característicos da cor, esse formato de construção é conhecido como colorímetro de tri-estímulos. Outra abordagem seria o uso de mais de uma cor incidindo alternadamente sobre o objeto e recolhendo para cada alternativa o valor encontrado pelo fotodetector.

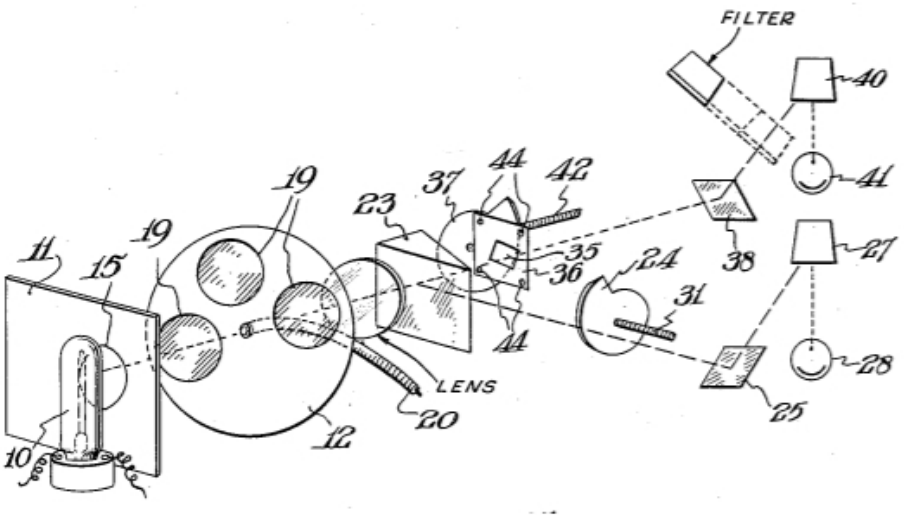

Figura 2.2: Diagrama da patente original do Colorímetro (SIMILARLY, 1956)

.

A figura 2.2 mostra o diagrama da patente original do colorímetro, onde é usada uma fonte de luz, indicada como 10, e um disco com 3 filtros (12), e esse disco é rotacionado para que as filtragens seja realizadas e assim o restante do aparelho tome conhecimento sobre as propriedades da luz que atravessam cada filtro. Assim ao fim do processo são conhecidos os três valores que representarão a cor em questão. Com o passar dos anos esses aparelhos apresentaram evoluções, uma das mais notáveis é a inserção de sistemas embarcados.

Em uma breve pesquisa realizada em *sites* de vendas durante o mês de outubro de 2011, observou-se que os colorímetros digitais variaram seus preço de mercado em uma faixa de 400 a 2000 dólares. O acréscimo de recursos e periféricos são os principais responsáveis pelo encarecimento dos aparelhos.

2.2 Fotodiodo e Fototransistor

O Fotodiodo é um dispositivo semicondutor de junção PN com a propriedade de controlar sua corrente reversa pela intensidade luz que nele incide. A quantidade que luz incidente transfere energia ao dispositivo que assim aumenta, quase que linearmente ao aumento da luz, a sua corrente reversa (LAZZARIN, 2008).

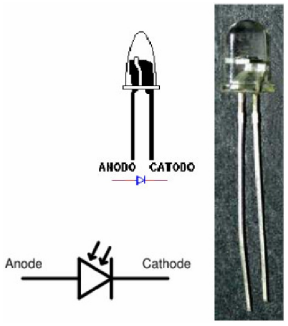

Figura 2.3: Simbologia e aspecto físico do fotodiodo. (LAZZARIN, 2008)

O germânio é mais apropriado para a construção destes dispositivos que o silício, já que o germânio cobre um espectro mais amplo de comprimentos de onda, mas em contrapartida tem uma corrente negra maior. A corrente negra é definida como a corrente reversa presente mesmo na ausência de luz (LAZZARIN, 2008).

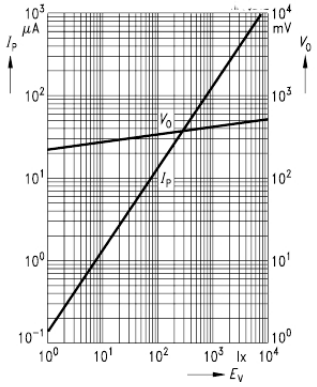

Figura 2.4: Corrente e Tensão em relação a irradiação da luz. (SEMICONDUCTORS, 2007)

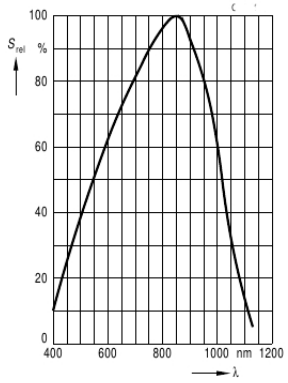

Figura 2.5: Sensibilidade espectral em relação ao comprimento de onda. (SEMICONDUCTORS, 2007)

Na Figura 2.4 é possível analisar o comportamento da tensão e da corrente que aparecem quando o fotodiodo é exposto a luz. Foi tomado como base o fotodiodo SFH 213 da empresa OSRAM. Já a Figura 2.5 mostra a sensibilidade a vários comprimentos de onda.

Existem dois princípios de operação para fotodiodos, quando ele trabalha como célula fotovoltaica, gerando tensão, ou como célula fotocondutiva, gerando corrente. A configuração fotovoltaica é o princípio das células solares utilizadas para a geração de energia solar. O modo fotocondutivo apresenta respostas mais

rápidas, porém é mais sensível a ruídos eletrônicos. A corrente gerada pela luz incidente não é suficiente para um controle direto, sendo então necessário um estágio de amplificação (LAZZARIN, 2008).

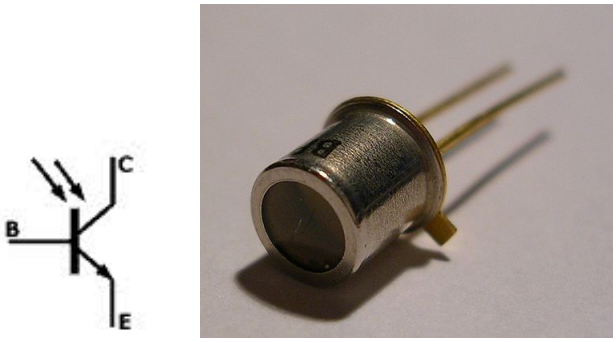

Figura 2.6: Simbologia do fototransistor e aspecto físico do fototransistor. (LAZZARIN, 2008)

Já o fototransistor opera sob um efeito conhecido como fotocondutividade. Como um transistor normal, ele é um dispositivo semicondutor de junção NPN, que detecta a incidência de luz e ainda fornece um ganho. Geralmente são usados apenas os terminais de coletor e emissor, já que o controle feito pela corrente de base nos transistores convencionais agora é feito pela luz (LAZZARIN, 2008).

Na maioria dos casos o fototransistor é usado como uma chave liga/desliga, onde a presença de luz faz o dispositivo entrar em condução e a ausência o deixa em estado de corte.

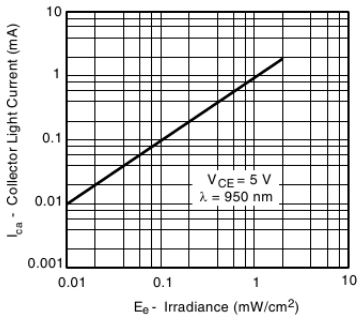

Figura 2.7: Corrente coletor versus a irradiação da luz. (VISHAY, 2008)

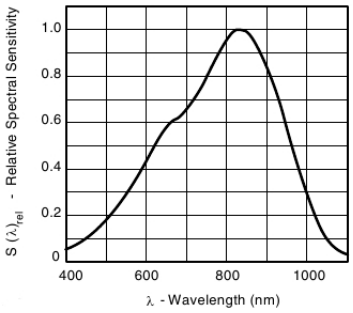

Figura 2.8: Sensibilidade espectral em relação ao comprimento de onda. (VISHAY, 2008)

Com base nas especificações fornecidas pela Vishay Semiconductors para o fototransistor BPW17N, é possível estimar pela Figura 2.7 a corrente de coletor em alguns níveis de irradiação da luz. A sensibilidade a certos comprimentos de ondas podem ser vistos na Figura 2.8.

2.3 LDR – *Light Dependent Resistor*: Resistor Variável de acordo com a incidência de Luz

O funcionamento desse componente se baseia no efeito fotoelétrico, que pode ser entendido como a propriedade de um metal ou semicondutor de transformar seus elétrons ligados aos átomos em elétrons livres capazes de conduzir corrente elétrica, e essa transformação acontece devido a exposição a uma luz numa certa faixa de frequência eletromagnética.

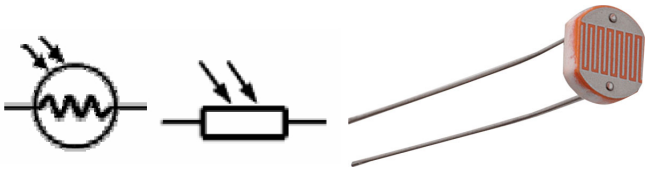

Figura 2.9: Simbologia e aspecto físico do LDR (LAZZARIN, 2008).

O LDR basicamente é um resistor que diminui o valor da sua resistência quando iluminado. A Equação (2.1) expressa a relação aproximada entre resistência e iluminação.

$$R = A * L^{-\alpha} \tag{2.1}$$

Onde o R é a resistência em Ohms, L a iluminação em Lux[2] e A e α são constantes (LAZZARIN, 2008).

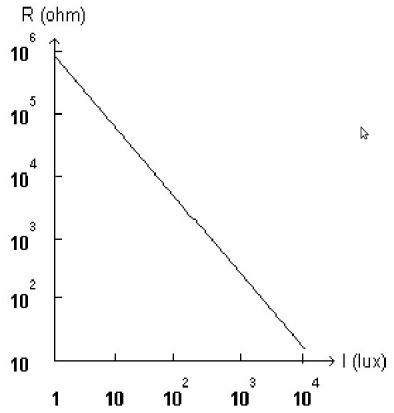

Figura 2.10: Resistência do LDR em função da iluminação (LAZZARIN, 2008).

A Figura 2.10 mostra o gráfico de resposta da resistência a quantidade de luz incidente no LDR. Já na Figura 2.11 encontramos a curva de sensibilidade do LDR para alguns comprimentos de ondas do espectro.

[2]É a unidade definida pelo Sistema Internacional de Medidas para medir Iluminamento. É equivalente a uma vela incidindo perpendicularmente em uma superfície de 1 metro quadrado.

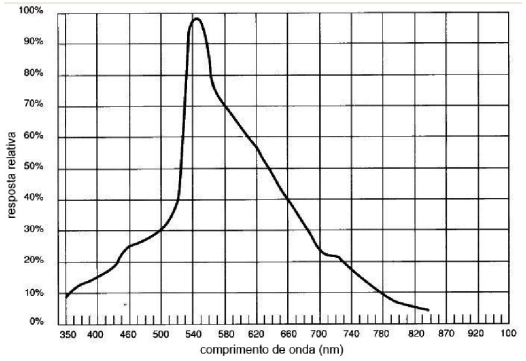

Figura 2.11: Sensibilidade de um LDR para vários comprimentos de onda da luz incidente (LAZZARIN, 2008).

O LDR pode ser um interessante fotosensor, já que trabalha na faixa do espectro visível (400 a 700 nm), mas apresenta desvantagens como um atraso de aproximadamente 100 ms para leitura. Geralmente é feito de sulfeto de cádmio (CdS), por ser sensível a luz no seu espectro visível, mas outros materiais podem ser usados como o Fosfeto de Cádmio (CdP), o Arseneto de Gálio (GaAs) e alguns outros.

2.4 Chip Sensor de RGB

Existem no mercado chips eletrônicos que quando expostos a luz conseguem reportar em três valores analógicos, representando o espaço RGB, os níveis de cada uma das cores primárias na composição da luz incidente. Um modelo conhecido é fabricado pela empresa Avago Technologies, o chip ADJD-E622-QR999 mostrado na Figura 2.12, é descrito como RGB *Color Sensor* (AVAGO, 2007).

Figura 2.12: Imagem de dois chips ADJD-E622-QR999, *Avago Technologies*.

O ADJD-E622-QR999 é um chip pequeno, 5x5x0.75 *mm* no encapsulamento QFN 5x5, e apresenta alto desempenho no tempo de resposta a leitura, e tudo isso em um circuito integrado (CI) fabricado com a tecnologia CMOS. Sua função é converter a luz para três tensões de saída, uma representando a banda vermelha, outra a verde e a última a azul. Para isso são usados três fotodiodos, cada um com um filtro respectivo a um dos três sinais de saída. Para se obter valores significativos de tensão são utilizados amplificadores de transimpedância, que convertem as baixas correntes que aparecem nos terminais dos fotodiodos, quando expostos a determinadas fontes de luz, em valores de tensão.

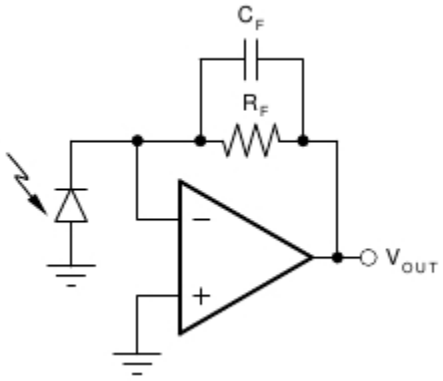

Figura 2.13: Um típico amplificador de transimpedância para monitoração de corrente em fotodiodos (WANG; EHRMAN, 1993).

A tensão de saída de cada canal do RGB aumenta linearmente com o aumento da intensidade da luz, podendo ser estimada pela Equação (2.2), que define a função ideal de tranferência de transimpedância.

$$V_{out} = -I_S * Z_F = -I_S * \frac{R_F}{1 + 2j\pi f R_F C_F} \tag{2.2}$$

Onde I_S representa a corrente que aparece nos terminais do fotodiodo quando exposto a luz. O chip trabalha alimentado por uma tensão em torno de 5v, podendo variar de 4.5 a 5.5v, e possui 16 pinos, na disposição mostrada na Figura 2.14, onde também podemos ver o diagrama de blocos do chip.

Pin Out for ADJD-E622-QR999

Pin	Pin Name	Normal Operation
Pin 1	VB_{OUT}	Analog output voltage for BLUE
Pin 2	VG_{OUT}	Analog output voltage for GREEN
Pin 3	VR_{OUT}	Analog output voltage for RED
Pin 4	VDD	5V DC Supply
Pin 5	GSGRN2	Gain Selection Green bit 2
Pin 6	GSGRN1	Gain Selection Green bit 1
Pin 7	GSRED2	Gain Selection Red bit 2
Pin 8	GSRED1	Gain Selection Red bit 1
Pin 9	GSRED0	Gain Selection Red bit 0
Pin 10	NC	No connection
Pin 11	NC	No connection
Pin 12	GSBLUE0	Gain Selection Blue bit 0
Pin 13	GSBLUE1	Gain Selection Blue bit 1
Pin 14	GSBLUE 2	Gain Selection Blue bit 2
Pin 15	GSGRN 0	Gain Selection Green bit 0
Pin 16	GND	Ground

Sensor IC Block Diagram

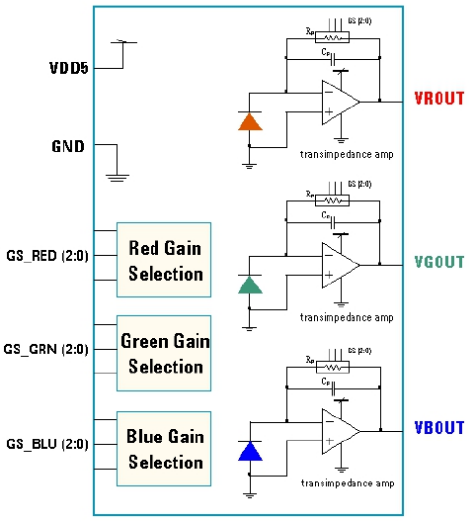

Figura 2.14: Descrição dos pinos do CI e diagrama de blocos(AVAGO, 2007).

Além dos pinos de alimentação e os três referentes as saídas analógicas usadas como resposta a luz incidente, o CI apresenta 3 bits de entrada para cada banda do RGB, tendo eles a função de ajustar o valor do resistor de resposta R_F, variando assim o valor do ganho de cada banda. O chip também apresenta uma abertura para a luz poder incidir sobre os fotodiodos.

A resposta espectral de cada banda quando o chip estiver com todos os bits de seleção de ganho em níveis lógicos 1 pode ser vista na Figura 2.15

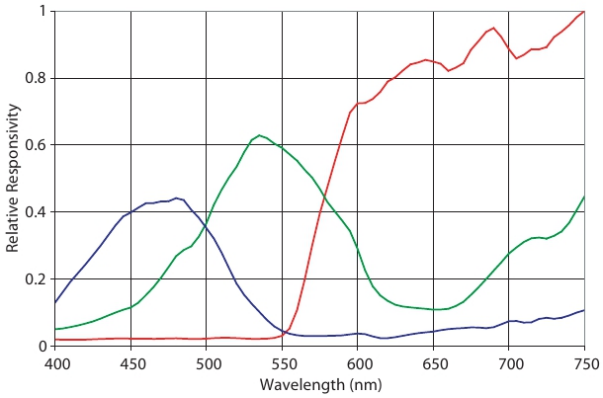

Figura 2.15: Resposta espectral (AVAGO, 2007).

E as tensões de saída para os níveis de irradiação de cada banda pode ser vista nos gráficos da Figura 2.16.

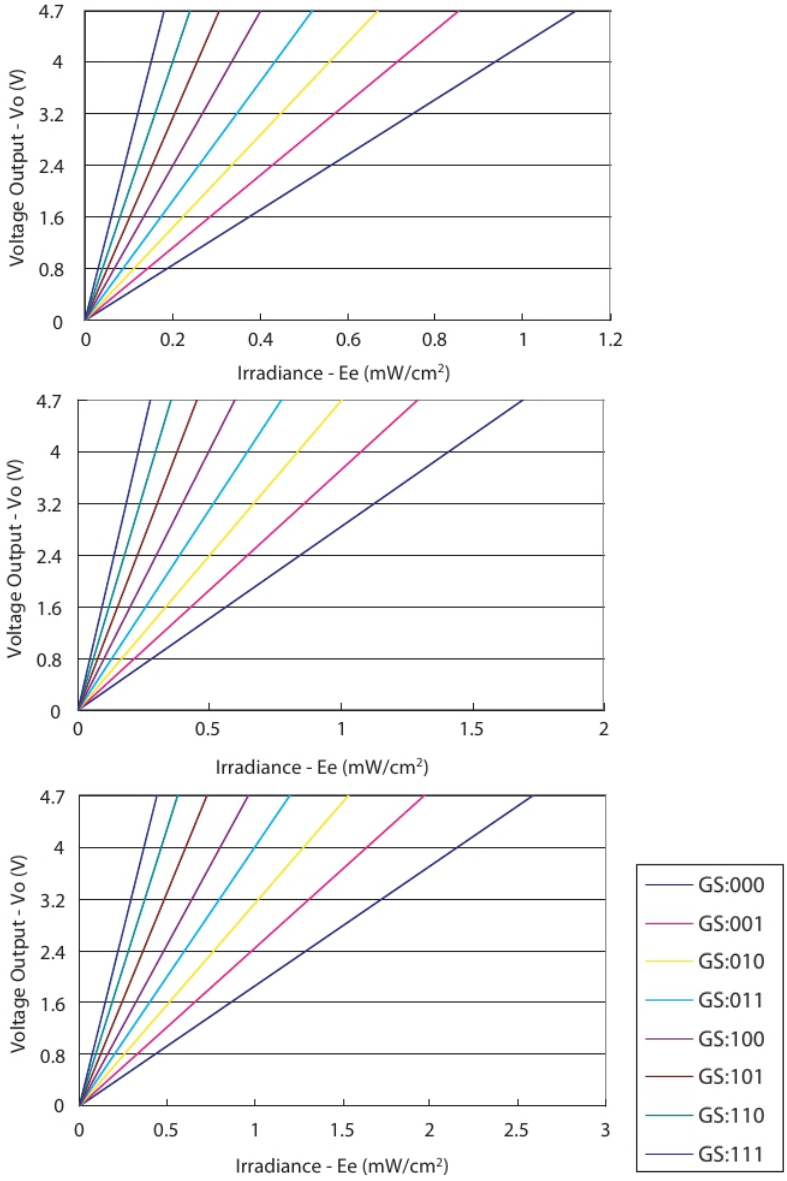

Figura 2.16: Tensão de saída versus a Irradiação da luz para a banda vermelha(645 nm), verde(542 nm) e azul(460 nm), respectivamente (AVAGO, 2007).

Existem também outros chips onde a conversão analógica/digital é feita internamente, e os valores são armazenados em registradores internos. Um exem-

plo é o ADJD-S313-QR999, também fabricado pela Avago. Neste chip toda a configuração dos parâmetros de leitura são armazenadas em registradores, e a comunicação com outros dispositivos é feita de forma serial, usando o protocolo I^2C. Esses chips possuem um valor de mercado inferior aos analógicos, tem uma precisão de 8 bits para cada banda do RGB e características de leitura semelhantes às do ADJD-E622-QR999.

O uso de um circuito industrializado pode melhorar a precisão da leitura e ainda diminuir o trabalho envolvido na construção de um aparelho para a medição de cores. Mas em alguns caso pode se tornar viável financeiramente a tentativa de montagem de circuitos eletrônicos para os mesmos fins.

2.5 Emissores de Luz

As cores nada mais são que a reflexão no espectro visível da luz incidente sobre uma superfície. Isso faz da fonte de iluminação um elemento importante quando se tem como objetivo mensurar as cores. Aqui será abordado como elemento emissor de luz o LED (*Light-emitting diode*), em especial os de alto brilho.

Um dos principais motivos de se utilizar LED's é a sua longa vida útil e eficiência luminosa. Segundo (OLIVEIRA *et al.*, 2007) um LED pode durar em torno 60 mil horas, enquanto uma lâmpada incandescente tem uma vida útil de aproximadamente 1000 horas e as fluorescentes duram em torno de 6000 horas. Outros pontos a serem destacados é seu tamanho, que é muito menor quando comparado com lâmpadas, por exemplo, e sua resistência a choques mecânicos (PINTO *et al.*, 2008).

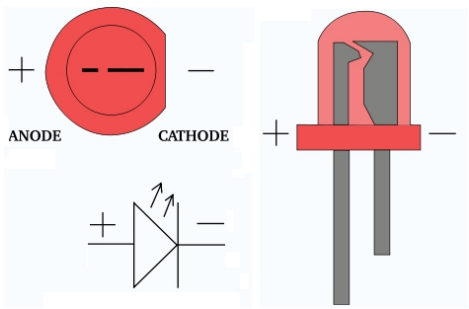

Figura 2.17: Símbolo e aspecto físico de um LED.

O diodo é um semicondutor de junção PN que só conduz corrente em um sentido, e quando polarizado diretamente os elétrons livres atravessam a junção, irradiando energia, que no caso do diodo é dissipada na forma de calor. Já no LED a maior parte da dissipação é sob forma de luminosidade. Geralmente diodos são feitos de cristais de silício, enquanto os LED's usam o gálio, o arsênio e o fósforo. O brilho de um LED depende proporcionalmente da corrente que nele flui (MARTINS, 2005).

Hoje é possível encontrar no mercado LED's com poder de iluminação parecido com os de muitos modelos de lâmpadas. Esse são conhecidos como LED's de alto brilho, sendo um dos mais difundidos o de luz branca.

2.6 Microcontroladores da Família PIC

Microcontroladores podem ser definidos como uma categoria especial de microprocessadores, com menos recursos computacionais e destinados a atividades de controle. São considerados computadores embutidos, e peças fundamentais em sistemas embarcados (WILMSHURST, 2006).

Uma das principais características que diferem o microcontrolador do microprocessador é que além do núcleo processador ele ainda agrega em um mesmo chip vários periféricos necessários para o seu funcionamento e interfaceamento

com o ambiente. A disposição deste periféricos é específica em cada modelo e varia de fabricante para fabricante.

Aqui será abordado os microcontroladores da família PIC(*Peripheral Interface Controller*), fabricados pela *Microchip Technology Inc.*. Existem vários dispositivos destes disponíveis hoje no mercado, sendo caracterizados pelas funcionalidades que agregam e pelas características internas de seus componentes. Algumas semelhanças que podem ser observadas são:

- Estrutura *Harvard*
- *Pipeline*
- Processador de 8 bits
- Arquitetura RISC
- Único acumulador (registrador de trabalho, W)
- *Reset*
- Vetor de interrupções

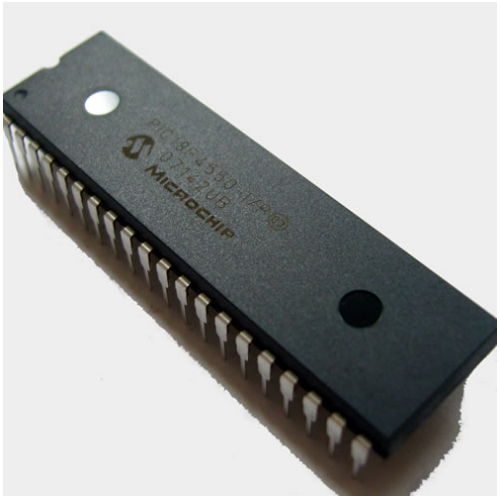

Figura 2.18: Imagem do PIC18F4550.

Em especial foi escolhido PIC18F4550, mostrado na Figura 2.18 , por oferecer um grande conjunto de interfaces, como a USB, os conversores analógico/digital (ADC) com grande número de entradas analógicas,13 no total, e 10 bit de resolução, suporte a *clock* externo para frequências de até 48MHz e grande número de pinos, o que facilita a programação e interligação com os componentes externos (PIC18F4550, 2009).

2.7 Protocolo de comunicação I^2C

O protocolo de comunicação I^2C (*Inter-Integrated Circuit*) foi criado pela Philips em meados de 1996 como um padrão da comunicação entre chips, com o objetivo de melhorar o desempenho do *hardware* e diminuir a complexidade dos circuitos.

A maioria dos chips atuais utiliza deste protocolo, já que várias vantagens podem ser observadas, como o controle via *software*, a escalabilidade, a facilidade da construção e diagnósticos de falhas, devido a sua simplicidade nas interconexões. Possui dois canais de comunicação o SDA, onde passam os dados de forma serial, e o SCL, que provê o sinal de *clock* para a sincronização dos bits seriais, sendo ambos canais bidirecionais. Desta forma o chip pode trabalhar como mestre e escravo.

O protocolo pode ser definido com base em dois sinais de controle, a condição de *Start* e a de *Stop*. Entre essas duas condições, os bits são transmitidos com base no sincronismo definido no sinal de *clock*. A cada *Byte* enviado, o dispositivo receptor envia um sinal de ACK(*Acknowledge*) para avisar do recebimento (FILHO, 2009).

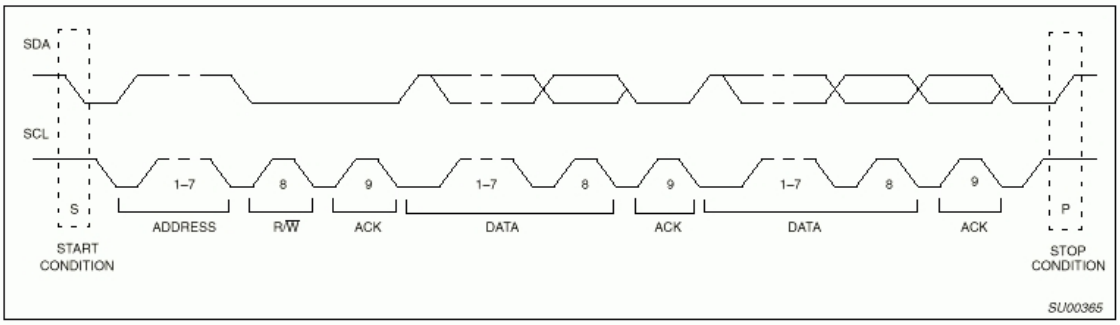

Figura 2.19: Formas de ondas de uma comunicação I^2C.

Com base na figura 2.19, vemos que após a condição de *Start*, é enviado um *Byte* contendo nos seus primeiros 7 bits o endereço do dispositivo escravo e no último bit um aviso se ocorrerá uma escrita ou uma leitura, então é enviado duas sequencias de de 8 bits, contendo os dados necessários para a realização da tarefa, e a comunicação é encerrada com a condição de *Stop*.

2.8 USB - *Universal Serial Bus*

O protocolo USB tem como principal função a comunicação de dados entre computadores e seus periféricos, e demais dispositivos eletrônicos que desejam se comunicar entre si. Emergiu das dificuldades encontradas para conectar dispositivos aos computadores, devido aos inúmeros tipos de conectores e padrões encontrados na época (ANDERSON, 1997).

Foi desenvolvida em meados de 1995 por um consórcio entre empresas como a *Microsoft*, Intel e outras. E um importante conceito usado era o de *Plug and Play*, onde após conectado o dispositivo já se encontraria pronto para uso, eliminando varias configurações necessárias anteriormente nas outras interfaces.

Os conectores do USB 2.0 possuem quatro canais. Dois deles são para alimentação, o V_{BUS} que é a tensão de referência, que geralmente é de 5 volts, e o GND que é o terra da alimentação. Os outros dois são os responsáveis pela a

transmissão dos dados o D+ e o D-. As taxas de transmissão variam de aplicação para aplicação, e podem chegar a velocidades de 500 Mega bits por segundo.

Nas conexões USB todos os nós estão ligados em um mesmo barramento. A comunicação é feita em forma de pacotes, onde todos eles iniciam com um campo de sincronização, que especifica a taxa de variação dos pulsos digitais. A transmissão dos bits é feita usando a codificação NRZI (*Non-Return to Zero Inverted*), que pode ser entendido da seguinte forma, se o bit a ser transmitido é 0, mantemos o nível da saída, se for 1, invertemos.

Cada pacote contem um PID(*id* do produtor), que identifica qual o seu tipo. Quando um dispositivo quer se comunicar com o outro ele envia um pacote contendo o endereço do *host* de destino, se alguém do barramento se identificar, será iniciada a transmissão do pacote com os dados. Logo em seguida o receptor envia um pacote de aviso de recebimento (UNIVERSAL..., 2000).

Todas as informações necessárias sobre o USB pode ser encontradas no link: `http://www.usb.org`[3].

2.9 Carta de Munsell

A Carta de Munsell nada mais é que um caderno com exemplos visuais de cores codificadas em três valores:

- matiz(*hue*) - comprimento de onda da luz
- croma(*chroma* ou *saturation*) - intensidade ou pureza da cor em relação ao cinza
- valor(*value*) - brilho ou tonalidade

É muito usada em Pedologia e Agronomia como referência para classificação de solos pela cor, já que a matéria orgânica e os óxidos de ferro são os principais agentes responsáveis pela cor dos solos (BOTELHO *et al.*, 2006).

[3]Página acessada em maio de 2012

O padrão de codificação das cores da carta é baseado no sistema Munsell de cores. Proposto por Albert H. Munsell em 1905, este sistema é considerado o melhor em termos de princípios perceptuais, também é conhecido como sistema HSV (SOB, 2007). Neste sistema as cores são representadas em amostras ordenadas e adjacentes com intervalos iguais de percepção visual. A representação ordenada das cores é feita por padrão com um cilindro tridimensional, como visto na Figura 2.20.

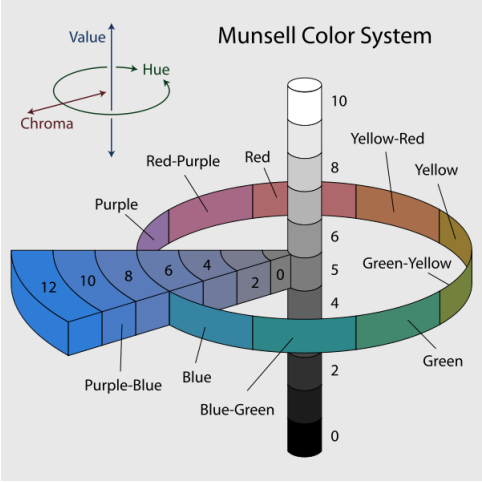

Figura 2.20: Representação cilíndrica do Sistema Munsell de cores.

O método de classificação de solos usando a Carta de Munsell é mundialmente usado, e a forma de atribuição de uma cor a um padrão descrito na Carta é feito visualmente, o que muitas vezes pode prejudicar a classificação devido a respostas diferentes ao espectro visível da luz em cada indivíduo, a quantidade de luz incidente no momento da comparação, e características da superfície (BOTELHO *et al.*, 2006). A Figura 2.21 mostra como algumas cores são dispostas na carta.

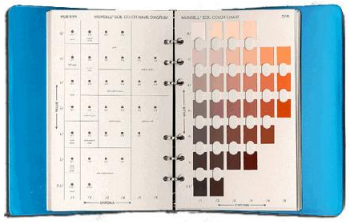

Figura 2.21: Imagem da Carta de Munsell.

2.10 Técnica do Vizinho-mais-Próximo

Trata-se de uma técnica de decisão não paramétrica, muito usada devido a sua facilidade de implementação. Onde dado um conjunto $D^n = \{x_1, ..., x_n\}$ de amostras e que $x' \in D^n$ seja a amostra mais próxima a um dado de teste x, com classificação desconhecida, logo pela técnica do vizinho-mais-próximo, x será classificado com o mesmo rótulo de x' (LACERDA; BRAGA, 2006).

Uma variação muito conhecida da técnica do vizinho-mais-próximo, é quando a classificação de um elemento desconhecido é determinada pela classificação dos k vizinhos mais próximos. Esta variação é conhecida como k-vizinhos-mais-próximos (kNN). O k é determinado antes da execução, geralmente como um valor ímpar, para evitar problemas com empates, assim os k elementos mais próximos ao dado desconhecido são buscados e a classificação mais presente entre eles é atribuída ao desconhecido.

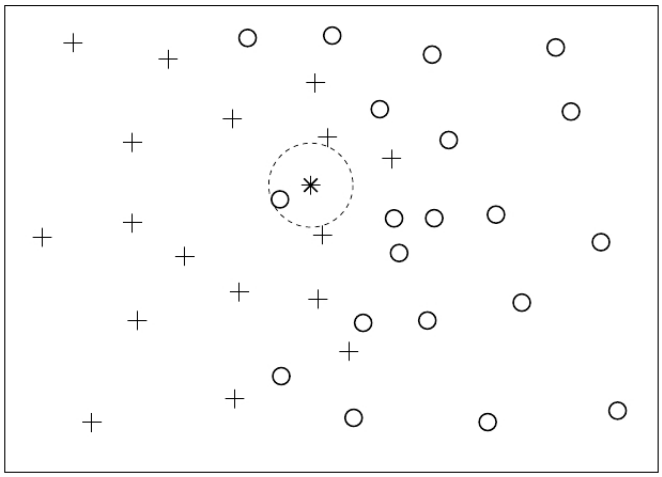

Figura 2.22: Técnica do vizinho-mais-próximo com (k = 1) (LACERDA; BRAGA, 2006).

A Figura 2.22 mostra como um dado desconhecido, quando posto no mesmo espaço que os dados de exemplo, pode ser classificado levando em conta a classificação mais frequente entre os *k* vizinhos mais próximos.

Apesar de sua simplicidade na definição, trata-se de uma técnica cara computacionalmente, devido aos cálculos necessários para determinação da distância para com os outros elementos, onde muitas vezes é necessário examinar a distância para cada um dos exemplos. Logo um conjunto muito grande de exemplos pode tornar a classificação inviável, então saber escolher um conjunto de elementos que possam trazer uma boa classificação e ao mesmo tempo evitar cálculos desnecessários passa a ser um problema para quem usa desta técnica.

O tamanho do conjunto de exemplos não é o único fator que interfere diretamente no desempenho desta técnica, mas também a métrica usada para o cálculo da distância e o número de dimensões dos dados a se trabalhar, podem influenciar diretamente no desempenho da técnica (LACERDA; BRAGA, 2006).

2.11 Redes Neurais Artificiais

As redes neurais artificiais (RNA) podem ser definidas como uma implementação do funcionamento de uma rede de neurônios biológicos em dispositivos eletrônicos. O centro do processamento das informações e aquisição do conhecimento em um ser humano, por exemplo, se encontra localizado no cérebro, que possui uma complexa rede de neurônios interconectados e trocando estímulos entre si. Esta conexão para troca de impulsos nervosos é conhecida como sinapse nervosa (KOVÁCS, 2002).

Cada neurônio é considerando como uma unidade básica de processamento, que pode ser representado por um modelo simplificado proposto por (MCCULLOCH; PITTS, 1943), que pode ser visto na figura 2.23. Nele há um conjunto de entradas $(x_1, x_2, x_3, ..., x_n)$ provenientes de sinapses, cada uma delas com um peso w. Primeiramente é computado o valor de *net*, onde $nte = \sum_{i=1}^{n}(x_i * w_i)$. O valor de *net* é comparado com o valor de μ, que é o valor de limiar, se *net* for maior ou igual, a saída binária do neurônio dispara 1, senão dispara 0. Esta comparação é conhecida como função de ativação, neste caso a função degrau, mas outros tipos de funções podem ser usadas, como a *linear*, a *sigmoidal* e outras (RAUBER, 2005).

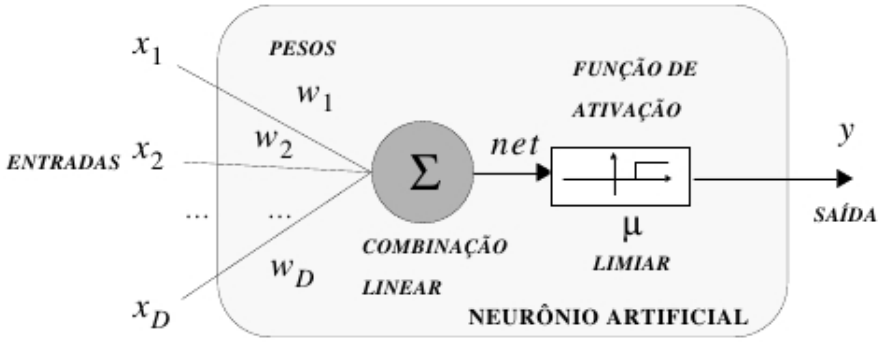

Figura 2.23: Modelo de um neurônio de McCulloch e Pitts (MCCULLOCH; PITTS, 1943).

O maior potencial do uso de redes neurais artificiais está no uso de um rede de neurônios interconectados. Um neurônio recebe estímulos em suas entradas, realiza os cálculos e envia sua saída para um próximo neurônio ou para a saída da rede, as redes onde a informação sempre passa adiante é conhecida como *feedforward*. Há várias topologias de redes, em algumas delas a informação não é propagada somente para frente, um exemplo são as com realimentação. Uma arquitetura muito utilizada é a *Multi Layer Perceptron*, que os neurônios estão organizados em camadas e cada camada envia informações para a posterior.

Uma das características mais importantes do uso de RNA é sua capacidade de aprendizado. Dado um conjunto de dados de treinamento para a rede, e as saídas esperadas para aquelas entradas, é possível, através de algoritmos de treinamento, definir os valores dos pesos de cada neurônio, e assim deixa- lá pronta para computar a saída de um conjunto de dados com resultado desconhecido, sendo eles da mesma família dos dados de treinamento.

Um dos algoritmos mais conhecidos para treinamento de RNA é o *Back-propagation*, que é usado para treinamento de redes multi-camadas, seu funcionamento pode ser explicado simplificadamente da seguinte maneira: Os dados de treinamento são apresentados a rede, então o erro é calculado, comparando a saída da rede com a resposta esperada, assim o algoritmo volta propagando o erro para as camadas anteriores, e assim ajustando os valores dos pesos de cada neurônio. E este procedimento é repetido até se obter saídas satisfatórias da rede.

Obtendo uma RNA treinada, a mesma se torna uma robusta ferramenta para se obter respostas satisfatórias para situações onde só as variáveis de entrada são conhecidas, mesmo estando elas com dados incompletos e ruídos.

2.12 Memórias EEPROM

Os sistemas de memória são os responsáveis pelo armazenamento dos dados, e estão presentes em inúmeras aplicações eletrônicas, sendo uma das partes mais importantes em sistemas computacionais.

A habilidade de reter informações é fundamental em várias tarefas, um exemplo é o processamento de dados de um computador digital, onde as instruções a serem executados e os dados que serão computados, ficam distribuídos em memórias conectadas ao processador (TOCCI; WIDMER; MOSS, 2003).

Existem várias classes de memórias, abrangendo inúmeras necessidades, algumas delas são:

- **RAM** (Random Access Memory): Onde independente da posição dos dados, serão acessados com a mesma latência;
- **SAM** (Senquential Access Memory): Os dados são acessados sequencialmente, um exemplo é a fita magnética.
- **RWM** (Read/Write Memory): As que podem ser lidas ou escritas.
- **ROM** (Read Only Memory): Uma classe de memórias não voláteis(não perdem seus dados com a falta de energia) onde a operação de leitura será a tarefa mais importante, e a operação de escrita é feita apenas uma vez. Existe também classes ROMs conhecidas como RMM(Read Mostly Memories) que podem ser escritas mais de uma vez, mas em todas a operação de escrita é uma tarefa mais trabalhosa que a de leitura.

2.13 Memórias Somente de Leitura

Existem dados que não são alterados frequentemente, ou que não podem ser perdidos na falta de energia, para este tipo de armazenamento, uma classe de memórias

é muito usada, as conhecidas como ROM. Nelas a escrita é geralmente feita apenas uma vez, mas também existem algumas variedades que podem ser reescritas inúmeras vezes. A operação de escrita nessa classe, é conhecida como programação ou "*queima*".

Exitem diversos tipos de ROMs, algumas delas são:

- **ROM**: Conhecidas apenas como ROMs, ou ROM de mascara, essas memórias são programadas de fábrica, seguindo uma especificação industrial ou do cliente. Permite ser programada apenas uma vez, sendo necessária uma substituição no caso de alguma alteração. Ainda são muito utilizadas quando necessitamos armazenar um grande volume de dados que não serão constantemente modificados;

- **PROM** (Programmable ROMs): Foram desenvolvidas para aplicações em menores escalas, onde o alto custo da fabricação de ROMs não seria viável. Elas podem ser programadas pelo usuário, e assim como as ROMs, uma vez programada, não poderá ser alterada;

- **EPROM** (Erasable PROM): Essa outra classe já permite que a programação feita pelo usuário possa ser apagada, para a realização de uma nova programação. O processo de gravação é geralmente demorado, podendo durar até alguns minutos. Para apagar uma célula EPROM, usamos a luz ultravioleta, que é aplicada sobre uma janela presente no encapsulamento do chip. Geralmente para apagar uma EPROM são gastos em torno de 15 minutos de exposição a luz ultravioleta. Uma grande desvantagem é que não é possível apagar apenas algumas células, ela é sempre apagada por completo;

- **EEPROM** (Electrically Erasable PROM): São ROMs que permitem ser apagadas eletricamente, facilitando o processo de gravação. E possuem a vantagem de poder apagar e reescrever bytes individuais. Uma desvantagem em relação as EPROMs, é que elas possuem uma densidade maior de área ocu-

pada em um circuito integrado, ficando assim na desvantagem em aplicações onde a densidade e o custo são críticos;

- **Flash**: Foi concebida pela necessidade de uma memória não-volátil de rápido acesso, e que pudesse se apagada eletricamente como a EEPROM, mas com um custo semelhante ao de uma EPROM. Possui esse nome devido a rapidez com a qual realiza as operações de escrita e de apagamento.

3 METODOLOGIA

Neste trabalho a metologia usada foi aplicada para concepção de um aparelho portátil capaz de identificar cores pré apresentadas ao dispositivo, e conseguir fazer aproximações de outras, realizando uma relação com as já conhecidas.

3.1 *Hardware*

Este aparelho trata-se de um Sistema Embarcado que coleta características sobre a resposta da reflexão da luz que ele incide sobre uma superfície, e a partir de exemplos previamente carregados em sua memória aplicar técnicas de inteligencia computacional para fazer a classificação, para obter assim a melhor aproximação desta cor em relação às pré conhecidas.

Para a implementação do protótipo foram utilizados os seguintes componentes de *Hardware*:

- Display de LCD com duas linhas de dezesseis colunas, para a visualização de informações referentes a leitura e funcionamento. Também há um potenciômetro para regulagem local do contraste do display;
- Porta USB para comunicação com um computador externo. Com objetivo de enviar e receber dados providos de um usuário;
- Quatro *Push-buttons*, para seleção de atividades e controle direto sobre as atividades do protótipo;
- Botão de *Reset* para reiniciar o dispositivo;
- Uma ponta de leitura composta por Led´s de iluminação, e um sensor para recolher os dados referentes ao reflexo emitido pela superfície iluminada;
- Um Microcontrolador PIC18F4550 como unidade de processamento central e de controle sobre os componentes já citados;

- Uma interface de gravação, para que o microcontrolador não necessite ser removido da placa para programação;
- Dois Led´s, um mostrando que a alimentação está conectada, e outro para aviso de execuções de leitura;
- Uma memória EEPROM de 512Kbits para armazenamento das informações necessárias para a execução dos algoritmos;
- Os demais componentes são utilizados para criar o ambiente eletrônico pedido por estes já citados.

O esquema do bloco principal do circuito proposto para o protótipo pode ser visto no anexo A.1.

Como sensor de luminosidade foi usado uma resistor do tipo LDR. O circuito responsável pela aquisição dos dados foi montado separadamente, havendo ligações externas para a conexão com o bloco principal do protótipo. Existem nele sete conexões básicas, duas para alimentação (VDD e GND), uma para obter a medida analógica referente a diferença de potencial nos terminais do LDR, e mais quatro para o acionamento dos LEDs de alto brilho. As cores utilizadas nos LEDs são: vermelho, verde, azul e amarelo. A alimentação do circuito é feita pela interface USB.

Os LEDs e o LDR estão acomodados internamente em uma das extremidade de um cilindro, a outra extremidade fica aberta para contato com a superfície de análise. O princípio básico de funcionamento do sensor é o seguinte, a extremidade aberta entra em contato com a superfície de análise de forma que não haja outra forma de luz sobre ela, então cada LED é ligado individualmente, e a tensão produzida nos terminais do LDR para cada vezes que um dos quatro LEDs é ligado, será adquirida por uma das entradas analógicas do microcontrolador. No fim do processo o software embarcado terá em mãos o conjunto de tensões $V_{leitura}$,

onde $V_{leitura} = \{V_r, V_g, V_b, V_y\}$. O circuito responsável pela aquisição pode ser visto no anexo A.2.

Existe também uma interface USB para comunicação do dispositivo com um computador externo. Através desta interface os dados de leitura são enviados para um programa monitor, onde as informações serão visualizadas em um terminal. E também é através dele que ocorre a sincronização do conteúdo da EEPROM com arquivos texto presentes no computador. Sendo estes arquivos os responsáveis pelas informações que dão ao software a autonomia para a classificação do conjunto $V_{leitura}$.

As informações sobre a leitura também são disponíveis no display de LCD. Este também é o responsável pela parte mais importante da interação com o usuário, já que mostra o *menu* de opções e as informações referentes a escolha do usuário.

Um diagrama simplificado do protótipo pode ser visto na figura 3.1. Considerando que os LEDs e o LDR estão dentro de um "copinho", que possui a superfície interna totalmente preta, para evitar a reflexão, ou uma possível interferência na resposta espectral da superfície. Uma visualização da parte interior da ponta de leitura pode ser vista na figura 3.2.

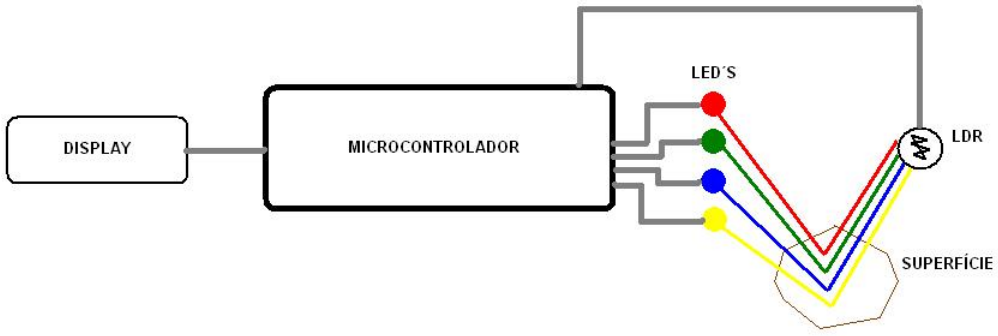

Figura 3.1: Diagrama em blocos do protótipo.

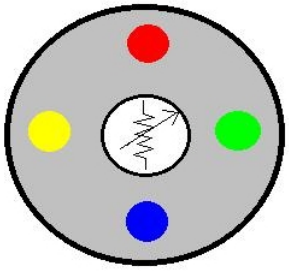

Figura 3.2: Visão inferior da ponta de leitura.

3.2 *Software*

Para implementação da camada de software foi escolhida a linguagem de programação C, devido as facilidades e abstrações providas pela mesma. Como compilador e ambiente de desenvolvimento foi utilizado o MikroC PRO versão 5.0.1, *software* produzido pela empresa *Microgenios* que é especialmente dedicado ao desenvolvimento para microcontroladores PIC.

Como algoritmo de classificação para as cores, foram implementadas duas abordagens, a Regra do Vizinho mais Próximo (VMP) e Redes Neurais Artificiais (RNA). Também foi necessária a programação de um monitor para a porta USB de um computador externo. Através desse *software* é possível enviar e receber dados entre o dispositivo e o computador, que também foi implementado usando a linguagem C.

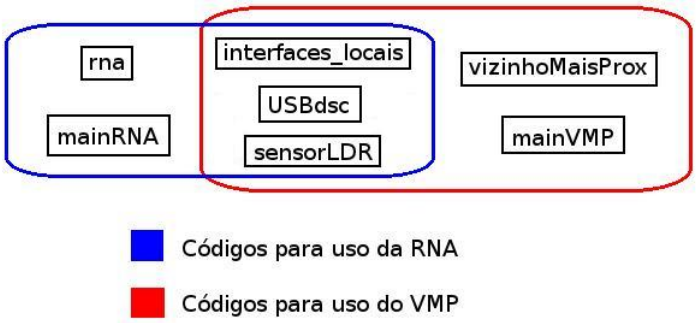

Figura 3.3: Diagrama dos códigos implementados para o dispositivo.

Um diagrama dos códigos implementados para dispositivo podem ser vistos na figura 3.3, nela há dois conjuntos, o em azul representam os arquivos *.c* utilizados para a implementação da classificação utilizando RNA, já os do vermelho, são os referentes a implementação utilizando a VMP. Eles foram implementados separadamente, pois a memória de programada do microcontrolador não foi suficiente para armazenar as duas implementações. Os códigos contidos na interseção desses conjuntos são as implementações que gerenciam as interfaces locais e a ponta de leitura.

O arquivo *USBdsc.c* é um descritor para a porta USB gerado pelo MikroC, nele estão contidas as informações que descrevem e dão as características necessárias a interface USB. É através deste código que podemos gerar as informações para identificação e para o tratamento externo do nosso dispositivo. Existe um opção no MikroC que permite a geração e edição dele sem contato direto com o código.

O *interfaces_locais.c* é o responsável pelo gerenciamento dos periféricos do protótipo. Nele existem funções que abstraem o controle da memória EEPROM, dos quatro *Push-buttons*, do envio e recebimento de dados pela porta USB e das abstrações para escrita de estruturas de dados diretamente na EEPROM. A tabela 3.1 mostra maiores detalhes sobre cada função implementada neste arquivo.

Função	Funcionalidade implementada
unsigned int leBotoes()	Retorna uma potência de 2 que representa quais botões estão sendo apertados no momento, por exemplo, se apertar o botão 1 ele retorna $1000_b = 8$.
void inicializa()	É responsável pela inicialização dos periféricos e configurações iniciais do microcontrolador.
void enviaUSB(**char** *str)	Recebe uma cadeia de caracteres, com no máximo 64, e os envia pela interface USB.
int leUSB(**char** palavra[])	Recebe um vetor de caracteres como parâmetro de referência. Faz a leitura do buffer da USB, colocando a *string* lida no parâmetro e retornando o seu tamanho.
void escreve_EEPROM(**unsigned int** end, **unsigned int** d)	Passado o endereço e um dado como parâmetro, este é escrito na EEPROM externa.
unsigned int le_EEPROM(**unsigned int** end)	Retorna da EEPROM o *byte* presente no endereço passado.
void EEPROM_StrRead(**unsigned int** a, **unsigned char** L, **unsigned char** *s)	Lê da EEPROM um conjunto de *L bytes* a partir do endereço *a*, e coloca no parâmetro de referência **s*.
void EEPROM_StrWrite(**unsigned int** a, **unsigned char** L, **unsigned char** *s)	Escreve na EEPROM um conjunto **s* de *L bytes* a partir do endereço *a*.
void inicializa_EEPROM()	Limpa os dados presentes na EEPROM externa.
int posicaoLivreEEPROM()	Retorna o próximo endereço livre na EEPROM.

Tabela 3.1: Funções do arquivo *interfaces_locais.c*

O arquivo *sensorLDR.c* é o responsável pelo gerenciamento da ponta de leitura. Ele implementa o controle sobre os LEDs, a leitura da tensão presente

nos terminais do LDR. E o envio pela USB dos dados coletados pelo sensor. A especificação das funções implementas está na tabela 3.2.

Função	Funcionalidade implementada
char *decodificaCor(**unsigned char** cor)	Recebe um valor de 0 a 3 e retorna a *string* que descreve a cor indexada. Por exemplo, se passado 3, ele retorna *"VERMELHO"*.
void ascendeLed(**unsigned char** L)	Ascende o LED indexado pelo parâmetro *L*. Por exemplo, se passado 1, ele ascende o LED azul.
void apagaLeds()	Apaga todos os LEDs.
unsigned int valor_LDR()	Retorna o valor da tensão nos terminais do LDR.
void calibrar_LDR()	Mostra no display de LCD, com um *delay* de 2 segundos, a diferença de potencial resultante no LDR, para cada LED.
void testa_LDR()	Mostra no display de LCD o valor atual da tensão nos terminais do LDR, com todos LEDs apagados, e com uma taxa de atualização de aproximadamente meio segundo.
void leituraLDR(**unsigned int** valores[])	Recebe um vetor de quatro inteiros por referência, e coloca nele o conjunto dos valores amostrados pela ponta de leitura.
void enviaLeituraPelaUSB()	Envia uma *string* pela USB que contem a especificação de quatro valores adquiridos pela ponta de leitura.
void amostragemPelaUSB()	Realiza 10 leituras com um intervalo de 2.5 segundos entre cada, e envia pela USB um relatório completo com cada leitura e a média geral.
void enviaMensagemUSB(**char** info[], **unsigned int** v[])	Recebe uma *string* que especifica a descrição do vetor de quatro valores, também passados por parâmetro. Apenas monta essa *string* e a envia pela porta USB.

Tabela 3.2: Funções do arquivo *sensorLDR.c*

3.3 Implementação da Regra do Vizinho mais Próximo (VMP)

Um dos métodos implementados para a classificação de novas amostras de cores, foi a Regra do Vizinho mais Próximo (VMP). A implementação aqui usada funciona da seguinte forma, é feita a leitura de uma nova cor, obtendo assim o conjunto $V_{leitura}$, que possui 4 valores, um para a diferença de potêncial nos terminais do LDR quando o LED vermelho está aceso, outro para o verde, azul e amarelo. A partir desses valores é feita uma busca sequencial entre todos os centroides previamente armazenados na EEPROM. A fórmula usada para escolher o melhor centroide é a seguinte: Dado o conjunto $V_{leitura} = \{V_r, V_g, V_b, V_y\}$, e um centroide C_i, onde $i \in \{1,2,3,4,...,N\}$, e que também é um conjunto de quatro valores. Busca-se o centroide que gere o menor valor d entre os N centroides.

$$d = |V_0 - C_{i0}| + |V_1 - C_{i1}| + |V_2 - C_{i2}| + |V_3 - C_{i3}| \tag{3.1}$$

A equação 3.1 também é conhecida como distância de *Manhattan*.

Encontrando o centroide que mais se aproxima ao conjunto $V_{leitura}$, a classificação realizada será expressa usando a descrição que acompanha os valores deste centroide. Essa descrição é uma *string* em código ASCII.

Uma das partes mais importantes para execução do algoritmo é a presença de um bom conjunto de busca na EEPROM. Este conjunto é obtido utilizando uma funcionalidade do dispositivo, que é a amostragem. Essa amostragem é feita da seguinte forma, é escolhida uma superfície de cor uniforme que seja maior que o diâmetro da ponta de leitura, então escolhemos a opção "Envia Amostragem" no menu principal do protótipo. Mantendo a ponta de leitura sobre a superfície, o dispositivo irá fazer 10 leitura com intervalos de 2.5 segundos, e ao final deste processo calculará a média para cada uma das bandas da cor amostrada. Um relatório completo de leituras amostrais pode ser acompanhado em um computador externo

pelo terminal de recepção de dados USB. Um exemplo de como seria esse relatório pode ser visto na figura 3.4, onde as dez primeiras linhas são as amostras de uma mesma cor, e a última é a média. Essa média será usada para definir os valores do centróide, para o nosso experimento usamos 30, onde cada um representa uma cor.

```
Amostra_0={Amarelo=714, Azul=883, Verde=753, Vermelho=206}
Amostra_1={Amarelo=709, Azul=882, Verde=754, Vermelho=206}
Amostra_2={Amarelo=710, Azul=883, Verde=753, Vermelho=206}
Amostra_3={Amarelo=710, Azul=883, Verde=754, Vermelho=206}
Amostra_4={Amarelo=710, Azul=883, Verde=754, Vermelho=206}
Amostra_5={Amarelo=710, Azul=883, Verde=754, Vermelho=206}
Amostra_6={Amarelo=710, Azul=883, Verde=754, Vermelho=206}
Amostra_7={Amarelo=710, Azul=883, Verde=755, Vermelho=206}
Amostra_8={Amarelo=710, Azul=884, Verde=755, Vermelho=206}
Amostra_9={Amarelo=710, Azul=884, Verde=755, Vermelho=207}
Amostra={Amarelo=710, Azul=883, Verde=754, Vermelho=206}
```

Figura 3.4: Exemplo de relatório enviado pela porta USB quando um cor é amostrada

Utilizando esta funcionalidade fazemos esse processo para um conjunto de cores de amostra, e ao fim, montamos um arquivo texto que será usado para especificar as cores presentes na EEPROM. Utilizamos a média como dado para este arquivo. Cada linha dele deve ter o seguinte formato:

NomeDaCor Valor_Amarelo Valor_Azul Valor_Verde Valor_Vermelho

Um exemplo deste arquivo de sincronização pode ser visto na figura 3.5.

```
Red   707   906   781   210
Green   804   924   739   390
Blue   856   835   817   452
Purple   749   864   822   262
Yellow   546   879   577   194
```

Figura 3.5: Formato de arquivo utilizado para sincronização com a EEPROM

Durante a sincronização do arquivo de centroides com a EEPROM, cada linha do arquivo é armazenada como mostrado na tabela 3.3. Neste exemplo foi considerando o armazenamento da primeira linha do arquivo da figura 3.5.

Red 707 906 781 210

Como o conversor analógico do PIC tem uma resolução de 10 *bits* e cada endereço da EEPROM armazena apenas um *byte*, 8 *bits*, Precisamos fazer algumas adaptações para realizar o armazenamento. Primeiramente considerando a descrição da cor, cada letra pode ser representada por um código ASCII, que por sua vez ocupa apenas um *byte*, então fazemos o seguinte, o tamanho da *string* é colocado na posição inicial, e em seguida cada letra em uma posição posterior. Quando terminamos de armazenar a descrição, armazenamos apenas os 8 *bits* mais significativos de cada um dos quatro números, e por último montamos um *Byte* composto pelo truncamento dos 2 *bits* menos significativos de cada um dos últimos 4 números armazenados. Por exemplo, os quatro números binários: $1011000011_b, 1110001010_b, 1100001101_b, 0011010010_b$, produziriam os cinco *Bytes*: $10110000_b, 11100010_b, 11000011_b, 00110100_b, 11100110_b$. E na próxima posição da EEPROM é colocado o valor zero, mostrando que é a próxima posição livre, e o fim do conjunto de dados armazenados.

	3	R	e	d	10110000	11100010	11000011	00110100	11100110	0	...
Byte	0	1	2	3	4	5	6	7	8	9	...

Tabela 3.3: Exemplo de armazenamento na EEPROM

As funções que implementam as funcionalidades da VMP estão presentes no arquivo *vizinhoMaisProx.c*, e uma descrição superficial de cada uma pode ser vista na tabela 3.4.

Havendo um conjunto de centroides alocados na EEPROM, apenas precisamos invocar a função destinada a fazer uma leitura, e utilizar os dados retornados

por ela para encontrar o centroide mais conveniente para a classificação. O pseudo código 1 mostra como o processo é feito.

Função	Funcionalidade implementada
void relatorioLDR(**unsigned int** v[], dadosCor dc)	Recebe os valores coletados, e a estrutura de dados composta que armazena as características do centroide escolhido. Utiliza esses dados para enviar pela USB um relatório da leitura e da classificação.
dadosCor resultadoCor(**unsigned int** dados[])	É a função que aplica o algoritmo da VMP, recebendo os valores da leitura, e retornando um centroide.
void classificarUsandoLDR()	Invoca a ponta de leitura para coletar os dados, e os repassa para a função *resultadoCor* para realizar a classificação.
void escreveCorEEPROM(**unsigned int** end, dadosCor c)	Recebe um centroide para ser armazenado na EEPROM no endereço *end*.
dadosCor leCorEEPROM(**unsigned int** ender)	Faz a leitura de um centroide armazenado na EEPROM no endereço *ender*.
void enviaCoresAmostradas()	Envia uma lista de todas as cores armazenas na EEPROM, através da USB.
void sincroArq_EEPROM()	Realiza o sincronismo do conteúdo da EEPROM com uma arquivo texto externo. Isso é feito com o auxilio de um *software* em um computador conectado ao dispositivo pela USB.

Tabela 3.4: Funções implementadas no arquivo *vizinhoMaisProx.c*.

```
menor = -1;
v = leituraLDR();
ret = ∅;
for cada centroide dc presente na EEPROM do
    d = |v[0] - dc.v[0]| + |v[1] - dc.v[1]| + |v[2] - dc.v[2]| + |v[3] -
    dc.v[3]|;
    if d < menor ou menor = -1 then
        menor = d;
        ret = dc;
    end
end
return ret;
```

Algorithm 1: Pseudo código da VMP implementada.

3.4 Implementação de uma Rede Neural Artificial (RNA)

Outra abordagem utilizada para a classificação das cores amostradas pela ponta de leitura, foi o uso de uma Rede Neural Artificial (RNA). Neste caso, devido a limitações do microcontrolador e maior simplicidade para a camada de *software*, o treinamento da rede foi realizado de forma *off-line*, coletando amostras da mesma maneira que na VMP, mas utilizando uma aplicação externa para modelagem e treinamento da RNA.

A ferramenta escolhida para facilitar o processo de escolha da melhor arquitetura e realização do treinamento foi o Neuroph[1] (*Java Neural Network Framework*). Esse aplicativo possui um interface muito intuitiva, e apenas conhecimentos básicos sobre redes neurais já é suficiente para utilizá-lo. Com ele é possível escolher e montar graficamente a arquitetura e importar os arquivos com os dados de treinamento. É distribuído juntamente com o seu código, que quando

[1] http://neuroph.sourceforge.net/, site acessado em julho de 2012.

acoplado com um projeto *Java*, acaba se tornado uma poderosa biblioteca de redes neurais.

Uma das partes fundamentais para o uso da RNA é ter um bom conjunto de treinamento. Nesse caso utilizamos novamente a função de amostragem do dispositivo, que devolve pela USB um conjunto de dados no formato já mostrado pela figura 3.4. Só que nesta abordagem todos os onze conjuntos de valores são usados, as dez amostras e a média.

Como arquitetura da RNA foi escolhida uma *Multi Layer Perceptron*, com função de ativação sigmoidal. Possui quatro entradas, que é tamanho do conjunto de valores retornado pela ponta de leitura. Na camada de saída há um neurônio para cada cor amostrada, no nosso caso foram utilizados trinta, pois trabalhamos com trinta cores de amostra. E a configuração das camadas intermediarias foi escolhida empiricamente, visando a que tivesse menos neurônios e ligações, já que isso impacta seriamente na ocupação do dispositivo de armazenamento. A melhor solução encontrada foi uma rede com 2 camadas intermediarias, onde cada uma possui quinze neurônios. Como algoritmo de treinamento usamos o *back-propagation* implementado no Neuroph, e o erro máximo de 0.001 foi a condição de parada. A topologia da rede pode ser vista na figura 3.6, onde a camada com apenas quatro neurônios representam as entradas.

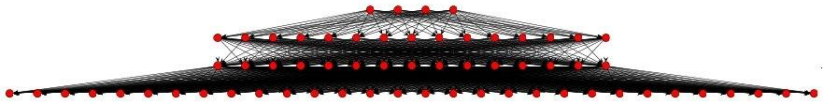

Figura 3.6: Representação da RNA como um grafo dirigido.

A parte mais crítica para essa implementação foi o armazenamento das informações necessárias para que o algoritmo da RNA pudesse trabalhar, nesse caso os pesos das ligações. Considerando a configuração utilizada, 4 entradas, duas camadas com 15 neurônios, e mais 30 na camada de saída, e como nesta configu-

ração todas as saídas de uma camada anterior são as entradas para cada neurônio da camada posterior, precisaremos então armazenar 735 pesos. Sendo estes pesos representados em ponto flutuante, que ocupa 4 *bytes* no microcontrolador, logo necessitamos de uma unidade de armazenamento que caiba 2940 *bytes*.

```
4
4
15
15
30

37.5483
−42.8590
48.7605
−118.7865
−124.6959
...
```

Figura 3.7: Formato do arquivo utilizado para sincronização dos pesos e da configuração da RNA com o conteúdo EEPROM.

São armazenados na EEPROM todos os pesos gerados durante o treinamento da RNA e os valores que descrevem a sua topologia, e estes dados também são sincronizados com um arquivo texto externo que contem essas informações. O formato do arquivo pode ser visto na figura 3.7, na primeira linha, colocamos quantas camadas terá a RNA, contando também a camada de entrada, na segunda linha colocamos quantas entradas tem a nossa rede, e nas próximas linhas a quantidade de neurônios em cada camada. Essas primeiras linhas descrevem a arquitetura de nossa RNA. Após a linha vazia colocaremos em cada uma um peso. Considerando um neurônio M_{ijk}, onde i descreve a posição da sua camada, j o seu posicionamento dentro da camada, e k o seu k-ésimo peso, sendo assim, estabelecemos uma ordem para que os pesos sejam colocados em cada linha do arquivo, sendo ela da seguinte forma:

M_{000}, M_{001}, M_{002}, ..., M_{010}, M_{011}, ..., M_{100}, M_{101}, M_{102}, ...

Os dados são armazenados de forma sequencial na EEPROM, na mesma ordem que aparecem no arquivo, assim o *software* pode ler a configuração da RNA e usar os pesos para calcular a saída. O algoritmo recebe os valores da leitura e coloca a RNA pra calcular, no fim do processo, apenas um neurônio é ativado, sendo ele o representante de uma cor, e esta é a classificação que a entrada recebeu.

Como o *chip* EEPROM que usamos possui um espaço de endereçamento de 16 *bits*, foi possível manter os dados da VMP e da RNA no mesmo chip. Isso foi feito seguindo a convenção que os 15 *bits* menos significativos contém o endereçamento, e o *bit* mais significativo avisa se estamos acessando espaço da VMP ou da RNA.

Cada neurônio precisa aplicar a função sigmoide, descrita pela equação 3.3, no valor de *net*, que é calculado pela equação 3.2, onde w_i é seu i-ésimo peso e x_i é sua i-ésima entrada.

$$net = \sum_{i=1}^{n} (x_i * w_i) \tag{3.2}$$

$$y = \frac{1}{1 + e^{-net}} \tag{3.3}$$

Para fazer os cálculos de cada neurônio usamos a função *calculaNeurônio*, descrita na figura 3.8. Essa função recebe como parâmetro o número de pesos, que pra nossa RNA também é o número de entradas, o endereço que se encontra o seu primeiro peso na EEPROM, e o vetor de dados de entrada. A cada nova leitura de um peso, ela incrementa o valor do endereço, e como este foi passado por referência nos parâmetros, no fim dos cálculos o endereço já estará atualizado com o valor utilizado pelo próximo neurônio. Por último ela retorna o valor computado para a saída.

```
double calculaNeuronio(int nPesos, int *endPesos, double entradas[])
{
    int i;
    double soma, valor;

    soma = 0.0;
    for(i = 0; i < nPesos; i++)
    {
        EEPROM_StrRead(*endPesos, sizeof(double), (unsigned char*)&valor);
        *endPesos +=  sizeof(double);
        soma += (double)(entradas[i]* valor);
    }
    return sigmoid(soma);
}
```

Figura 3.8: Função responsável pelo calculo da saída de uma neurônio.

Sabendo que a computação de uma camada depende somente dos dados das saídas da camada anterior, com exceção da primeira, que recebe diretamente os dados da entrada da rede, podemos realizar o calculo completo de uma RNA de forma sequencial, onde para cada camada, a partir da entrada, faremos a computação de cada um de seus neurônios. E ao fim teremos um conjunto de saídas para a última camada. Neste caso o interesse é buscar o neurônio que foi ativado. Será considerado ativado o que apresentar o maior valor na sua saída, este valor está entre 0 e 1. Conhecendo o neurônio ativado, buscamos a descrição da cor que ele representa, para assim realizar a classificação da entrada.

As as funcionalidades implementadas para a realização da classificação usado RNA estão no arquivo *rna.c*, e podem ser vistas na tabela 3.5.

Função	Funcionalidade implementada
void sincroArqPesos_EEPROM()	Essa função tem a tarefa de realizar a sincronização do arquivo texto externo que contém a topologia e os pesos da rede.
void mostraPesos()	Envia pela USB os pesos armazenados na EEPROM.
double calculaNeuronio(**int** nPesos, **int** *endPesos, **double** entradas[])	Recebe a quantidade de pesos, o endereço que eles estão na EEPROM, e o conjunto de entradas. Retorna o valor da computação feita por um neurônio da rede.
int saidaRNA(**double** entradas[])	Faz a computação da RNA sobre o conjunto de entradas, e retorna qual neurônio da saída foi ativado.
void classifica_RNA()	Ativa a ponta de leitura para coletar informações, e invoca a RNA para classificar esses dados. Mostra a classificação no LCD e envia um relatório pela USB.

Tabela 3.5: Funções implementadas no arquivo *rna.c*.

3.5 O monitor da interface USB

Para facilitar a comunicação com o dispositivo um *software* externo foi implementado, visando oferecer tarefas básicas e um maior conforto para o usuário. A aplicação basicamente é um gestor da interface USB, podendo identificar quando o dispositivo está conectado e enviar ou receber informações a ele. As tarefas básicas implementas são a capacidade de ficar no modo leitor, onde qualquer informação enviada pelo dispositivo possa ser mostrada, e as opções de sincronizar com a EEPROM o arquivo dos centroides, ou o de pesos.

As características predefinidas para o dispositivo, que o diferenciam dos demais conectados as portas USB do computador, são conhecidas como *id* do produto (PID) e *id* do vendedor (VID). Essas configurações são mantidas no arquivo

USBdsc.c, e são definidos para o nosso dispositivo um PID igual a $0x01_H$ e um VID de $0x1234_H$.

A partir disso, o primeiro passo do monitor é buscar se estas configurações correspondem a algum dispositivo conectado a uma das portas USB do computador. Caso ele encontre, é aberto um canal de comunicação entre o dispositivo e o computador. Assim todas as tarefas poderão ser agora executas, e ao fim do execução esse canal é fechado.

A aplicação é executada via linha de comando, quando ela é chamada um argumento deve ser passado, sendo ele os valores, 1, 2 ou 3. No 1 o modo leitor é ativado, para o 2 é iniciado o processo de sincronização do conteúdo da EEPROM com os centroides contidos em um arquivo texto, o 3 também faz a sincronização de um arquivo com a EEPROM, mas desta vez é o que contem os pesos e a topologia da RNA. Caso a opção 2 ou 3 seja escolhida, é necessário passar o caminho que leva a localização do arquivo de sincronização. Considerando um ambiente *unix*, os três modos de operação podem ser invocados como descrito na figura 3.9.

```
# modo leitor ativado
$ ./aplicativoGestorUSB 1

# sincronizacao dos centroides
$ ./aplicativoGestorUSB 2 /caminho/arquivoCentroides.txt

# sincronizacao das info. da RNA
$ ./aplicativoGestorUSB 3 /caminho/arquivoRNA.txt
```

Figura 3.9: Exemplos de como executar a aplicação gestora da comunicação USB.

No modo leitor a aplicação apenas fica esperando dados serem inseridos no *buffer* de leitura, e assim que ocorre essa ação, o conteúdo desse *buffer* é escrito na tela do computador.

Quando passado o 2 como argumento, inicialmente a aplicação pede que você escolha a opção *Sinc. Arq x Mem* no menu do dispositivo, assim a sincronização entre o arquivo e o conteúdo da EEPROM é iniciado automaticamente.

Já no modo 3, você deve colocar o dispositivo na opção *Sinc. Pes. x Mem*. Lembrando que o programa do computador atende ao dispositivo tanto quando ele trabalha somente com RNA ou VMP, já que as duas implementações não puderam ser colocadas juntas.

Para garantir uma segurança a mais na sincronização, toda vez que um dado é enviado do computador ao dispositivo, um código de segurança é transmitido junto, o computador só envia uma nova informação após o retorno deste código. Assim podemos garantir que a operação foi concluída com êxito.

4 RESULTADOS OBTIDOS

Ao fim da execução da metodologia foi concebido um protótipo como visto no anexo A.3, que provê o suporte para todas as funcionalidades implementadas. O dispositivo terá dois modos de operação para realizar a classificação, sendo um deles utilizando a regra do vizinho mais próximo e o outro usando redes neurais artificiais. Os dois modos não puderam ser implementados juntos em um mesmo software, pois os 32K*bytes* de memória de programa do microcontrolador não foram suficientes. Sendo assim necessário reprogramar quando quisermos utilizar a outra técnica de classificação.

Em termos de memória de programa utilizada no microcontrolador para a execução das técnicas, o uso de RNA foi mais crítico, ocupando 32673 dos 32768 *bytes* disponíveis, o que representa aproximadamente 99,71% dessa área, enquanto isso a regra do vizinho mais próximo (VMP) utiliza menos memória, 30309 *bytes*, o que é aproximadamente 92% do total. Em termos de memória principal, a VMP gasta um pouco a mais, cerca de 1219 dos 2048 *bytes* disponíveis, o que representa 60% do total, já RNA gastam um pouco a menos, 1212 *bytes*, o que mostra que as duas técnicas possuem um consumo do recurso de memória quase que equivalente. O armazenamento de vetores de caracteres é o principal responsável pelo grande consumo de memória, mas são peças fundamentais na execução, já que é através deles que o dispositivo pode se comunicar com o usuário.

O protótipo possui um consumo energético relativamente baixo, 130*mA*. Esse valor da uma certa garantia que a porta USB poderá alimentar o dispositivo sem problemas, já que pode fornecer até 500*mA*. Mas é esperado que o valor de consumo durante a execução normal seja menor, já que esses 130*mA* foram obtidos como um valor crítico, onde todos os componentes do circuito estarão trabalhando ao mesmo tempo, o que é um caso raro de se acontecer em ambas as técnicas. Pos exemplo, nesta medida de consumo os quatro LEDs foram ligados ao mesmo tempo, o que não acontece em nenhum momento durante uma execução usual, que

apenas acende um de cada vez. Os LEDs por serem de alto brilho consomem uma corrente considerável.

Já durante a classificação a VMP se mostrou mais rápida, levando aproximadamente 1,4 segundos para fazer a leitura, classificar os dados obtidos, mostrar a classificação no LCD e enviar um relatório pela USB, uma grande fração deste tempo é devido a *delays* necessários para a sincronização e para garantir que não haja conflitos entre as tarefas. Já o uso de RNA se mostrou um pouco mais demorado, gastando aproximadamente 2,1 segundos. A explicação para esse maior tempo vem da maior quantidade de dados a serem buscados na EEPROM e a necessidade de mais operações aritméticas de multiplicação, que demandam mais ciclos para a sua realização, e também um fator crítico no uso desta técnica no microcontrolador é que todos os dados são representados por números no formato de ponto flutuante, e estes gastam muito mais ciclos de operação para sua computação, quando comparados com operações somente com inteiros, que é o caso da VMP. Também os *delays* usados na leitura são comprometedores, pois a cada vez que um LED acende é esperado 280ms para que seja feita a leitura do valor do LDR, esse *delay* é necessário para a estabilização do valor da diferença de potencial nos terminais do LDR, como são quatro LEDs, logo mais de 1 segundo é gasto só com a aquisição dos dados. Como a memória EEPROM usada é serial, e realiza a comunicação pelo protocolo I^2C, o valor do *clock* é muito importante para que operações de leitura e escrita sejam realizadas rapidamente, aqui foi escolhido um valor de 1MHz, o máximo suportado pelo dispositivo usado. Uma comparação entre o desempenho das técnicas pode ser visto na tabela 4.1.

Técnica	**Tempo exec.**	**Memória de programa**	**RAM**
VMP	1,4s	30309B	1219B
RNA	2,1s	32673B	1212B

Tabela 4.1: Comparação entre as técnicas.

Como caso de teste para o dispositivo foi usado duas cartelas de cores, cada uma com 30. O padrão escolhido foi o das cores do HTML, já que existe um conjunto de cores já previamente nomeadas, onde não é necessário o uso da codificação geralmente utilizada.

As cores do HTML podem ser codificadas em três *bytes*, um para cada banda do RGB, e a notação usada para representação de cada *byte* é a hexadecimal. Mas também existe um conjunto de cores já previamente nomeadas, o que dispensa o uso da codificação. Um pequeno exemplo de cores que já possuem descrição textual no HTML pode ser vista na tabela 4.2. A descrição textual utilizada pelo HTML será o padrão de resposta para a classificação realizada pelos algoritmos.

Cor	**Código**
Black	#000000
White	#FFFFFF
Red	#FF0000
Green	#00FF00
Blue	#0000FF
Purple	#A020F0

Tabela 4.2: Exemplos de cores do HTML.

Como dados de treinamento foram usadas 30 cores, e mais 30 como casos de teste, isso para ambos algoritmos. Somente foram escolhidas cores que possuem descrição textual no HTML. Os dois métodos de classificação acertam todos os dados de treinamento, e aproximam bem para os dados de teste. O algoritmo que melhor se saiu na aproximação das cores de teste, com relação a algumas das cores de treinamento, foi a VMP, nela todas as cores de teste foram aproximadas as que visualmente eram as mais próximas. Já com o uso de RNA algumas cores foram mal aproximadas, mas mesmo assim a classificação era feita para cores que tinham uma relação visual. Um exemplo disso é a cor descrita como *Fuchsia*, que

visualmente se aproxima mais com a *Magenta*, sendo esta a classificação dada pela VMP, mas com o uso RNA, ela foi classificada como *Violet*. Outros fatores interferiram no trabalho de classificação, como a infidelidade da impressão das cores no papel, mostrando uma leve diferença entre o que é mostrado no arquivo virtual e que é colocado no papel. As tabelas com as 60 cores podem ser vistas no anexo A.5.

No geral, podemos dizer que os dois métodos possuem excelente poder de aproximação. O uso de RNA foi pior em tempo de execução e não teve uma classificação tão boa, mas talvez com uma busca maior de outras configurações e outros treinamentos isso possa ser melhorado. E também este caso de teste é pequeno, levando a crer que em outros contextos poderemos ter resultados diferentes para cada técnica.

Levando em conta os componentes utilizados na construção, e comparando com o preço de colorímetros comerciais, podemos afirmar que o protótipo possui um baixo custo. Já que pesquisando em alguns *sites* que vendem componentes eletrônicos, e baseando-se nos elementos com valores mais significativos, podemos estimar um preço total de R$50,00(Cinquenta Reais), onde alguns desses valores podem ser vistos na tabela 4.3. Esse preço total aproximado não considera a mão de obra empregada na construção.

Componente	**Preço (R$ - Reais)**
Display de LCD	10,00
PIC 18F4550	17,00
EEPROM 24C512	7,00

Tabela 4.3: Preços médio de alguns componentes usados no protótipo.

5 CONCLUSÃO

A mensuração da cor não é uma tarefa que em todos os momentos pode ser confiada somente ao olho humano, e que em várias situações sua má precisão pode afetar resultados úteis para uma tomada de decisão.

O uso de equipamentos eletrônicos melhora a precisão e provê maior confiança ao processo de classificação das cores. Mas nem sempre é encontrado um aparelho que realize a classificação da forma necessária pelo usuário.

Reconhecer cores é uma habilidade que o ser humano aprende geralmente nas fases iniciais da vida, através de informações obtidas de outras pessoas ou por aprendizado próprio. E algumas formas de deficiência também podem prejudicar esta habilidade.

A automatização deste processo por um aparelho eletrônico pode ser aplicada em várias situações, como aprendizado das cores por crianças, percepção de diferenças entre tonalidades, classificações quanto a um modelo e outras.

Neste trabalho mostramos que o uso de técnicas como a regra do vizinho mais próximo e redes neurais artificiais, podem ser ferramentas valiosas para a realização da classificação de cores. E mesmo com um protótipo simples, conseguimos bons resultados. Apesar de utilizarmos poucos dados de treinamento, foi possível perceber que cores consideradas idênticas aos nossos olhos, são facilmente distinguidas pelo aparelho. E para as cores das quais ele não conhece, realiza uma boa aproximação com relação as conhecidas.

Foi possível concluir que um componente simples como o LDR já é capaz de realizar várias medidas de precisão da resposta da luz. E que também uma construção inversa aos colorímetros convencionais, onde existe apenas uma fonte luminosa e vários sensores, também é viável. E mesmo com o atraso da estabilização do valor de tensão do LDR, e sua pequena variação de leitura para leitura, as técnicas ainda sim foram capazes de acertar a classificação.

É difícil afirmar qual das técnicas foi a melhor, já que o número de casos de treinamento foi pequeno, e talvez com outros conjuntos o comportamento seja diferente, principalmente quando o número de instâncias aumenta muito. E também adotando outras abordagens durante o treinamento, escolhendo outras arquiteturas, e trabalhando com outros fatores que interferem diretamente no comportamento das RNA, talvez seja possível obter uma rede com características melhores que a encontrada.

5.1 Trabalhos futuros

Talvez o uso de outros sensores possa melhorar os resultados, como por exemplo, o chip sensor visto no referencial teórico, ou outros foto sensores, como fotodiodos ou fototransistores. O LDR sofre muitas interferências de fatores externos, o que provoca muitos ruídos nas leituras, em outros contextos esses ruídos podem se tornar críticos, tornando o seu uso inviável.

O uso de um conversor analógico/digital com uma resolução maior que a do microcontrolador, que é de 10 *bits*, pode ajudar a conseguir dados mais precisos, pois torna o sensor mais sensível a pequenas alterações.

Como melhorias para o protótipo podemos colocar uma bateria como fonte de energia, eliminado a necessidade de conexão com o computador. Seria bom utilizar um microcontrolador que consiga armazenar as duas técnicas ao mesmo tempo, eliminando o inconveniente de reprogramar sempre que necessária uma troca entre os dois métodos. E também a implementação de outros algoritmos de classificação, ou até mesmo melhorias nos que já estão implementados poderiam aumentar a eficiência do classificador.

5.2 Considerações finais

O protótipo mostrou ser flexível a novos contextos, por oferecer uma boa quantidade de recursos para novas amostragens e inserção de dados. E não é muito difícil encontrar aplicações onde suas funcionalidades possam ser aplicadas, como por exemplo, na detecção de plantas doentes utilizando a coloração da sua folha, automatização da escolha de caminhos por um robô, utilizando várias linhas coloridas, uma para cada caminho, e outras atividades onde a classificação automática de cores é uma das suas necessidades.

Considerando outras técnicas de classificação de cores, como o processamento de imagens, onde o uso de uma câmera é necessário para obtenção dos dados de entrada dos algoritmos, a abordagem implementada apresenta algumas vantagens, já que a construção de uma circuito completo de um sensor ou o uso de chip específico, ainda são soluções com melhor custo benefício, e que dão resultados tão bons quanto estas técnicas.

O projeto se mostrou viável financeiramente e com complexidade relativamente baixa, fazendo dele uma boa escolha quando a classificação de uma cor é necessária. E comprova que essa ideia de usar componentes simples para confecção de um sensor, aliada com o uso de algoritmos já conhecidos de classificação, são suficientes para a implementação de um dispositivo eletrônico capaz de reconhecer cores.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDERSON, D. *USB system architecture*. [S.l.]: Addison-Wesley, 1997.

AVAGO, T. L. *ADJD-E622-QR999, RGB Color Sensor in QFN Package*. United States: [s.n.], 2007.

BOTELHO, M.; DALMOLIN, R.; PEDRON, F. de A.; AZEVEDO, A. de; RODRIGUES, R.; MIGUEL, P. Medida da cor em solos do rio grande do sul com a carta de munsell e por colorimetria. *Ciência Rural*, v. 36, n. 4, 2006.

FILHO, R. P. D. A. *Protocolo de Comunicação I2C*. 2009. http://www2.eletronica.org/artigos/eletronica-digital/protocolo-de-comunicacao-i2c. Acessado em maio de 2012.

HOLDINGS, I. K. M. *Catálogos de Produtos*. Osaka, Japan: [s.n.], 2011. http://sensing.konicaminolta.us/technologies/colorimeters/. Acessado em outubro de 2011.

HOLDINGS, I. K. M. *CR-410 PRODUCT SPECS*. Osaka, Japan: [s.n.], 2011.

HUNTERLAB. *CIE L*a*b* Color Scale: applications note*. 2008. http://www.hunterlab.com/ColorEducation/ColorTheory. Vol. 8, No. 7, acessado em setembro de 2011.

KARRAS, P.; LADSON, J. *Colorimeter instrument with fiber optic ring illuminator*. [S.l.]: Google Patents, ago. 7 1984. US Patent 4,464,054.

KOVÁCS, Z. *Redes neurais artificiais: fundamentos e aplicações*. [S.l.]: Editora Livraria da Fisica, 2002.

LACERDA, W.; BRAGA, A. de P. *Projeto e implementação de circuitos classificadores digitais com controle da generalização baseado na regra do vizinho-mais-próximo modificada*. Tese (Doutorado) — PhD thesis, Escola de Engenharia da Universidade Federal de Minas Gerais, Belo Horizonte, 2006.

LAZZARIN, L. Fotodiodos e fototransistores. *MATERIAIS ELÉTRICOS: COMPÊNDIO DE TRABALHOS VOLUME 7*, p. 747, 2008.

LECLAIRAGE, C. I. D. *Proceedings of the eight session*. 1931. Vienna, Austria.

MARTINS, E. Comparação entre os desempenhos de faróis automotivos convencionais e aqueles que empregam diodos emissores de luz. *São Paulo*, 2005.

MCCULLOCH, W.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, Springer, v. 5, n. 4, p. 115–133, 1943.

MENDES, L.; IANO, Y.; SABLÓN, V.; PIETRO, R. D. Subsistema de compressmo e codificaçmo do sinal de vídeo dos padrões hdtv (parte ii). *Revista Científica*, v. 1516, p. 2338, 2000.

MOTOKI, A.; ZUCCO, L.; SICHEL, S.; AIRES, J.; PETRAKIS, G. Desenvolvimento da técnica para especificação digital de cores ea nova nomenclatura para classificação de rochas ornamentais com base nas cores medidas. *Geosciences= Geociências*, v. 25, n. 4, p. 403–415, 2007.

OLIVEIRA, A.; MARCHESAN, T.; PRADO, R.; CAMPOS, A. Distributed emergency lighting system leds driven by two integrated flyback converters. In: IEEE. *Industry Applications Conference, 2007. 42nd IAS Annual Meeting. Conference Record of the 2007 IEEE*. [S.l.], 2007. p. 1141–1146.

PIC18F4550, M. T. I. *Microcontroller Datasheet*. 2009.

PINTO, R.; COSETIN, M.; MARCHESAN, T.; CAMPOS, A.; PRADO, R. do. Lâmpada compacta empregando leds de alto-brilho. In: *XVII Congresso Brasileiro de Automática–CBA, Juiz de Fora-MG*. [S.l.: s.n.], 2008.

RAUBER, T. Redes neurais artificiais. *Universidade Federal do Espírito Santo*, 2005.

SANTOS, S. F. Síntese de pigmentos cerâmicos e desenvolvimento de cores em porcelanas feldspáticas. COPPE UFRJ, DISSERTAÇÃO PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA METALÚRGICA E DE MATERIAIS, Rio de Janeiro, RJ, Brasil, p. 3–14, 2006.

SEMICONDUCTORS, O. O. *Silizium-PIN-Fotodiode mit sehr kurzer Schaltzeit Silicon PIN Photodiode with Very Short Switching Time*. 2007. Disponível em: <www.osram-os.com>.

SIMILARLY, Y. *Colorimeter*. [S.l.]: Google Patents, dez. 18 1956. US Patent 2,774,276.

SOB, O. Estudo in vitro da estabilidade de cor e opacidade de cinco sistemas cerâmicos sob influência do envelhecimento artificial acelerado. 2007.

TOCCI, R.; WIDMER, N.; MOSS, G. *Sistemas digitais: princípios e aplicações*. [S.l.]: Prentice Hall, 2003.

UNIVERSAL Serial Bus Specification. 2000. http://www.usb.org/developers/docs/. Acessado em maio de 2012.

VISHAY, S. *BPW17N, Silicon NPN Phototransistor, RoHS Compliant*. 2008. Document Number: 81516, Rev. 1.7. Disponível em: <www.vishay.com>.

WANG, T.; EHRMAN, B. Compensate transimpedance amplifiers intuitively. *Application Notes, Burr-Brown*, 1993.

WILMSHURST, T. *Designing embedded systems with PIC microcontrollers: principles and applications*. [S.l.]: Newnes, 2006.

A ANEXOS

A.1 ANEXO 1

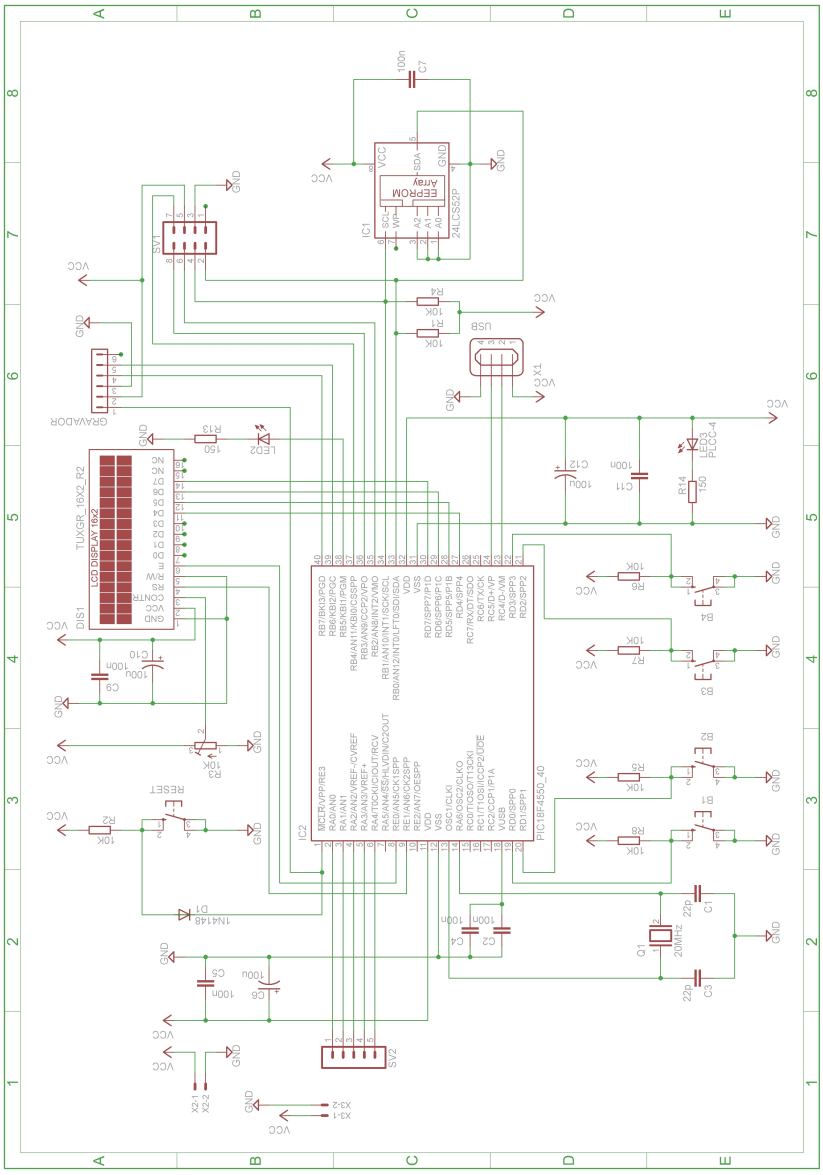

Figura A.1: Esquema eletrônico para montagem do bloco principal do protótipo.

A.2 ANEXO 2

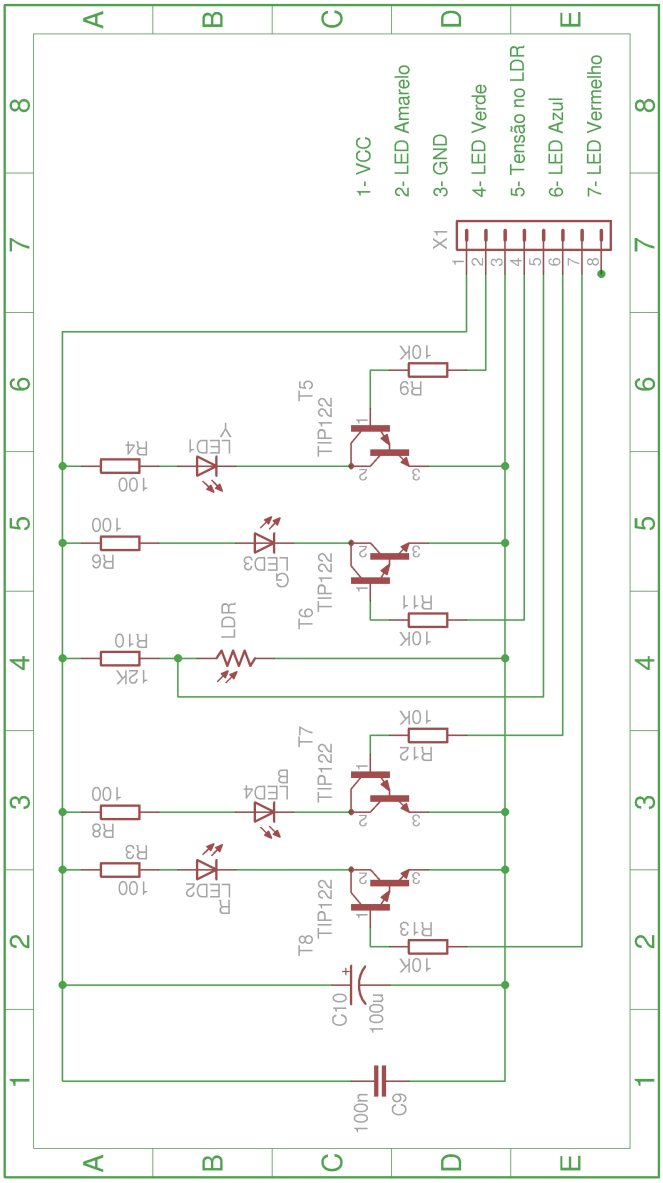

Figura A.2: Esquema eletrônico da ponta de leitura.

A.3 ANEXO 3

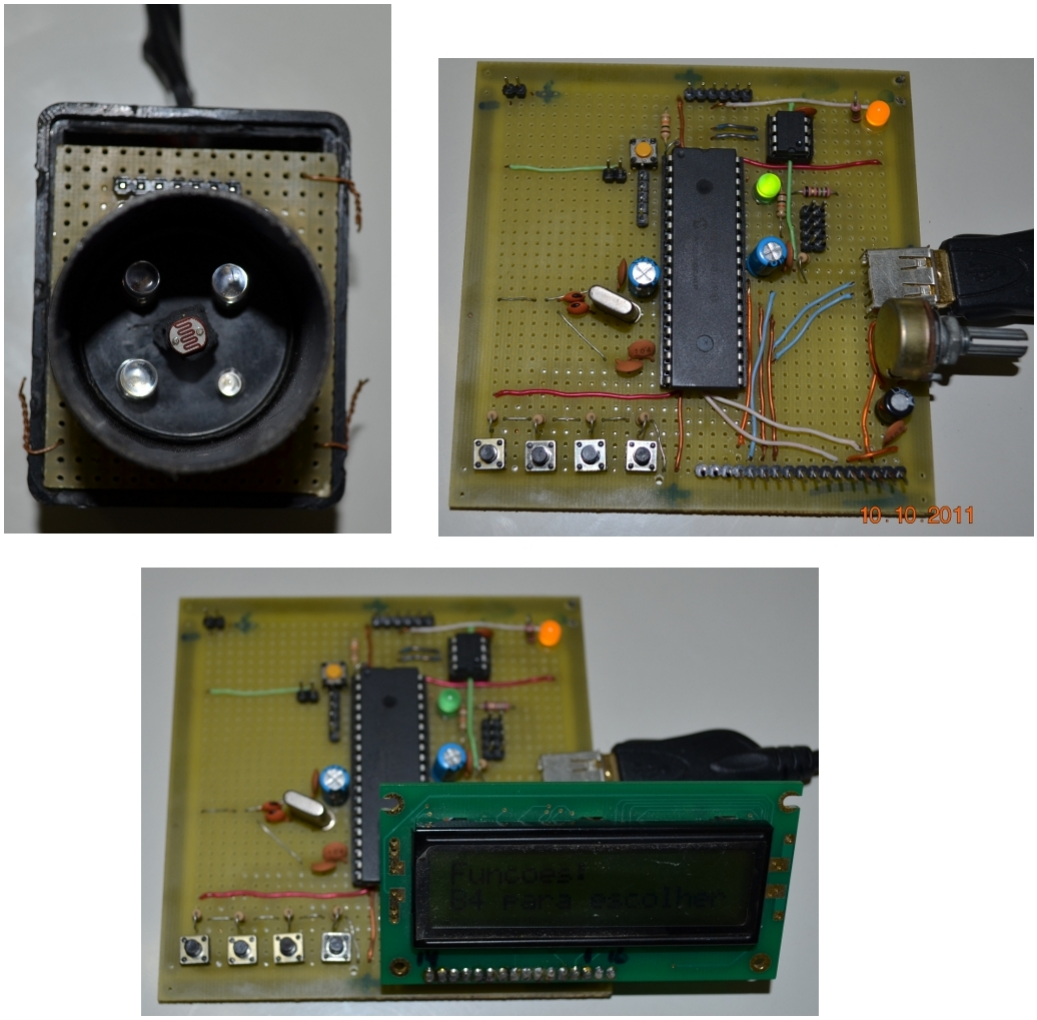

Figura A.3: Fotos do protótipo.

A.4 ANEXO 4

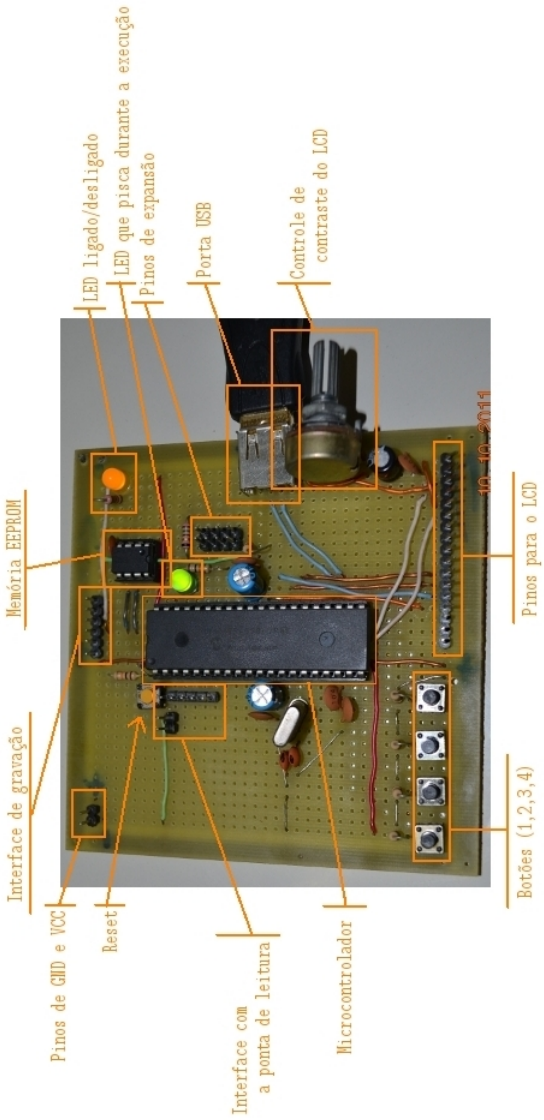

Figura A.4: Foto comentada do protótipo.

A.5 ANEXO 5

Cartelas de cores

Figura A.5: Cartela 1.

Figura A.6: Cartela 2.

A.6 ANEXO 6

A.6.1 Códigos do microcontrolador

Listing A.1: vmp.c

```
/*
 * vmp.c : Main para o uso da regra do vizinho mais proximo
 */


#include "sensorLDR.h"
#include "interfaces_locais.h"
#include "vizinhoMaisProx.h"
#include "rna.h"

#define NF 8


/* funcionalidades */
char funcoes[NF][17] = {
"B4 para escolher",
"Dados LDR + LEDs",
"Classif. por VMP",
"Valor no LDR    ",
"Envia leitura   ",
"Envia Amostragem",
"Envia Cores     ",
"Sinc. Arq x Mem "};


unsigned int escolheFuncao()
{
    int f = 0;
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Funcoes:");
    while(1)
    {
        Lcd_Out(2,1,funcoes[f]);
        delay_ms(120);
```

```
        if( leBotoes() == 8 )
        {
            PORTB.RB5 = ~PORTB.RB5;
            f = ( (f + 1) % NF );
            f = (f == 0) ? 1 : f;
        }
        else if (( leBotoes() == 4 ) && ( f != 0))
        {
            return f;
        }
    }
    return 0;
}

void main(void)
{

    inicializa();

    while(1)
    {
        switch (escolheFuncao())
        {
                case 1:
                    calibrar_LDR();
                    break;

                case 2:
                    classificarUsandoLDR();
                    break;

                case 3:
                    testa_LDR();
                    break;

                case 4:
                    enviaLeituraPelaUSB();
                    break;

                case 5:
```

```
                    amostragemPelaUSB();
                    break;

                case 6:
                    enviaCoresAmostradas();
                    break;

                case 7:
                    sincroArq_EEPROM();
                    break;

        }

    }
}
```

Listing A.2: LDR_RNA.c

```
/*
 * LDR_RNA.c: main para o uso de RNA
 */


#include "sensorLDR.h"
#include "interfaces_locais.h"
#include "rna.h"

#define NF 4


/* funcionalidades disponiveis */
char funcoes[NF][17] = {
"B4 para escolher",
"Classif. c/ RNA ",
"Sinc. Arq. Pesos",
"Mostra Pesos    "};


unsigned int escolheFuncao()
{
    int f = 0;
```

```
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Funcoes:");
    while(1)
    {
        Lcd_Out(2,1,funcoes[f]);
        delay_ms(120);
        if( leBotoes() == 8 )
        {
            PORTB.RB5 = ~PORTB.RB5;
            f = ( (f + 1) % NF );
            f = (f == 0) ? 1 : f;
        }
        else if (( leBotoes() == 4 ) && ( f != 0))
        {
            return f;
        }
    }
    return 0;
}

void main(void)
{

    inicializa();

    while(1)
    {
        switch (escolheFuncao())
        {
                case 1:
                    classifica_RNA();
                    break;

                case 2:
                    sincroArqPesos_EEPROM();
                    break;

                case 3:
                    mostraPesos();
                    break;
```

```
        }

    }
}
```

Listing A.3: interfaces_locais.h

```
/*
 * interfaces_locais.h
 */


#ifndef INTERFACES_LOCAIS_H
#define INTERFACES_LOCAIS_H



#define POS_LIVRE_B1 0xA2
#define POS_LIVRE_B2 0xA3
#define N_CORES_EEPROM 0xA4
#define DELAY_EEPROM    2
#define DISP 0xA0




unsigned int leBotoes();

void inicializa();

void mostraLCD( int linha, int coluna, char *str );

void enviaUSB( char *str );

void interrupt();

int leUSB( char palavra[] );


void escreve_EEPROM(unsigned int endereco, unsigned int dado);
```

```
unsigned int le_EEPROM(unsigned int endereco);

void EEPROM_StrWrite(unsigned int a, unsigned char l, unsigned char *s);

void EEPROM_StrRead(unsigned int a, unsigned char l, unsigned char *s);

void inicializa_EEPROM();

int posicaoLivreEEPROM();

void setPosicaoLivreEEPROM( int pl );

void addNumeroCores_EEPROM( int i );

int numeroCores_EEPROM();

#endif
```

Listing A.4: interfaces_locais.c

```
/*
 * interfaces_locais.c
 */


#include "interfaces_locais.h"


/* LCD module connections */

sbit LCD_RS at RE1_bit;
sbit LCD_EN at RE0_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;


sbit LCD_RS_Direction at TRISE1_bit;
sbit LCD_EN_Direction at TRISE0_bit;
sbit LCD_D4_Direction at TRISD4_bit;
```

```
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
/* End LCD module connections*/



// Buffers should be in USB RAM, please consult datasheet
unsigned char readbuff[64] absolute 0x500;
unsigned char writebuff[64] absolute 0x540;



/* funcao que retorna a potencia de 2
 * que representa quais botoes estao sendo apertados
 */
unsigned int leBotoes()
{
    int i;
    unsigned int b = 0x00;
    TRISD = 0x0F;
    for ( i = 0; i < 4; i++ )
    {
        if ( Button(&PORTD, i, 0, 0 ) )
        {
            delay_ms(8);
            if ( Button(&PORTD, i, 0, 0) )
            {
                b = b | (0x01 << i);
            }
        }
    }
    return (b & 0x0F);
}



/* inicializacao do microcontrolador */
void inicializa()
{
    /* inicia LCD */
    ADCON1 = 0x0F;     /* desabilita o conversor A/D*/
```

```
    LATD = 0;
    LATE = 0;
    TRISB = 0x00;
    PORTB.RB5 = 0;

    Lcd_Init();
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Cmd(_LCD_CURSOR_OFF);
    delay_ms(200);
    /* ********************************** */

    /* inicia USB */
    HID_Enable(&readbuff,&writebuff);        // Enable HID communication

    I2C1_Init(1000000);
}

void mostraLCD( int linha, int coluna, char *str )
{
    ADCON1 = 0x0F;     /* desabilita o conversor A/D*/
    LATD = 0;
    LATE = 0;
    Lcd_Out(linha,coluna,str);
    delay_ms(80);
}


void interrupt()
{
    USB_Interrupt_Proc(); // USB servicing is done inside the interrupt
}


void enviaUSB( char *str )
{
     strcpy(writebuff,str);
     while(!HID_Write(&writebuff,64));
}

int leUSB( char palavra[] )
```

```
{
    int ret;
    ret = HID_Read();
    strcpy(palavra,readbuff);
    return ret;
}


unsigned int le_EEPROM( unsigned int endereco)
{
    unsigned int dado;
    unsigned char *end;
    end = &endereco;
    I2C1_Start();               // issue I2C start signal
    I2C1_Wr(DISP);              // send byte via I2C  (device address + W)
    I2C1_Wr(end[0]);                 // send first byte (address of EEPROM location)
    I2C1_Wr(end[1]);                 // send last byte (address of EEPROM location)
    I2C1_Repeated_Start();      // issue I2C signal repeated start
    I2C1_Wr(DISP+0x01);            // send byte (device address + R)
    dado = I2C1_Rd(0);        // Read the data (NO acknowledge)
    I2C1_Stop();                // issue I2C stop signal
    //delay_ms(DELAY_EEPROM);
    return dado;
}

void escreve_EEPROM( unsigned int endereco, unsigned int dado)
{
  unsigned char *end;
  end = &endereco;
  I2C1_Start();               // issue I2C start signal
  I2C1_Wr(DISP);              // send byte via I2C  (device address + W)
  I2C1_Wr(end[0]);                 // send first byte (address of EEPROM location)
  I2C1_Wr(end[1]);                 // send last byte (address of EEPROM location)
  I2C1_Wr(dado);              // send data (data to be written)
  I2C1_Stop();                // issue I2C stop signal
  delay_ms(DELAY_EEPROM);
}
```

```
/*
 * a : address
 * l : length
 * s : pointer to data
 */
void EEPROM_StrRead(unsigned int a, unsigned char l, unsigned char *s)
{
    unsigned char i ;
    for(i = 0 ; i < l ; i++)
    {
       *s = le_EEPROM(a + i);
       s++ ;
    }
}

void EEPROM_StrWrite(unsigned int a,
                                  unsigned char l, unsigned char *s)
{
    unsigned char i ;
    for(i = 0 ; i < l ; i++)
    {
        escreve_EEPROM(a+i, *s);
        s++ ;
    }
}



void inicializa_EEPROM()
{
     EEPROM_Write(POS_LIVRE_B1,0);
     EEPROM_Write(POS_LIVRE_B2,0);
     EEPROM_Write(N_CORES_EEPROM,0);
}

int posicaoLivreEEPROM()
{
     int pl;
     pl = EEPROM_Read(POS_LIVRE_B1);
     pl = pl << 8;
```

```
    pl = pl | EEPROM_Read(POS_LIVRE_B2);
    return pl;
}


void setPosicaoLivreEEPROM( int pos_livre )
{
    EEPROM_Write(POS_LIVRE_B1, pos_livre >> 8);
    EEPROM_Write(POS_LIVRE_B2, pos_livre & 0xFF);
}

int numeroCores_EEPROM()
{
   return EEPROM_Read(N_CORES_EEPROM);
}


void addNumeroCores_EEPROM( int n )
{
  EEPROM_Write(N_CORES_EEPROM, (numeroCores_EEPROM() + n));
}
```

Listing A.5: rna.h

```
/*
 * rna.h
 */


#ifndef RNA_H
#define RNA_H




#include "sensorLDR.h"
#include "interfaces_locais.h"
#include "vizinhoMaisProx.h"
```

```
#define END_N_CAMADAS 0x8000;
#define END_N_NEURO_CAMAD 0x8001;



void sincroArqPesos_EEPROM();

void mostraPesos();

double calculaNeuronio(int nPesos, int *endPesos, double entradas[]);

int saidaRNA( double entradas[] );

void copiaVetor( double vet1[], double vet2[], int tam );

void classifica_RNA();

double sigmoid( double x );

dadosCor indentificaCor( int cor );

#endif
```

Listing A.6: rna.c

```
/*
 * rna.c
 */


#include "rna.h"

void sincroArqPesos_EEPROM()
{
      char palavra[16];
      int estado, t_dado, sucesso, n_camad, aux, fim;
      unsigned int end_pesos;
      float peso;

      inicializa_EEPROM();
      Lcd_Cmd(_LCD_CLEAR);
```

```
Lcd_Out(1,1,"Carregando EEPROM");

estado = 1;
sucesso = 0;
t_dado = 0;
fim = 0;


Lcd_Out(2,1,"Armazenando.     ");

while(( leBotoes() != 1 ) && (fim == 0))
{
      if ( leUSB(palavra) > 0 )
      {
        PORTB.RB5 = ~PORTB.RB5;
        if ( (estado == 1) && (strcmp(palavra,"201") == 0))
        {
           estado = 2;
           enviaUSB(palavra);
        }
        else if ( estado == 2)
        {
           n_camad = atoi(palavra);
           escreve_EEPROM(0x8000, n_camad);
           enviaUSB("202");
           estado = 3;
        }
        else if ( estado == 3 )
        {
           if (n_camad > 0)
           {
                aux = atoi(palavra);
                escreve_EEPROM(0x8001 + t_dado, aux);
                t_dado++;
                n_camad--;
                enviaUSB("203");
                if( n_camad == 0 )
                {
                     estado = 4;
                     end_pesos = 0x8001 + t_dado;
```

```
                    }
                }
            }
            else if ( estado == 4 )
            {
                if (strcmp(palavra,"205") == 0)
                {
                    estado = 1;
                    enviaUSB(palavra);
                    sucesso = 1;
                    fim = 1;
                }
                else
                {
                    enviaUSB("204");
                    peso = atof(palavra);
                    EEPROM_StrWrite( end_pesos, sizeof(double),
                                                    (unsigned char*)&peso );
                    end_pesos += sizeof(double);
                }
            }
            delay_ms(10);
          }
          delay_ms(15);
      }
      if( sucesso == 0 )
      {
         inicializa_EEPROM();
         Lcd_Out(2,1,"Falha na grav.  ");
      }
      else Lcd_Out(2,1,"Sucesso na grav.");
      delay_ms(1500);
}

void mostraPesos()
{
    char texto[16];
    unsigned int nc, i, j, np, ep;
    double r;
```

```
    nc = le_EEPROM(0x8000);

    ep = nc + 0x8001;
    np = 0;
    Lcd_Cmd(_LCD_CLEAR);
    for(i = ep-1; i > 0x8001; i--)
    {
        np += (le_EEPROM(i)*le_EEPROM(i-1));
    }

    ep = nc + 0x8001;
    for(i = 0, j = 0; j < np; i+=sizeof(double), j++)
    {
        EEPROM_StrRead(ep+i, sizeof(double), (unsigned char*)&r );
        sprintf(texto,"%f", r );
        Lcd_Out(1,1,texto);
        strcat(texto,"\n");
        enviaUSB(texto);
        delay_ms(1000);
    }
}


void classifica_RNA()
{
    unsigned int v[4] = {0,0,0,0};
    double ent[4];
    int pos, i;
    dadosCor dc;

    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"COR: ");
    Lcd_Out(2,1,"?");
    while( leBotoes() != 1 )
    {
        if ( leBotoes() == 2 )
        {
            leituraLDR( v );
            for(i = 0; i < 4; i++)
            {
```

```
                ent[i] = v[i]/1023.0f;
            }

            Lcd_Out(1,1,"COR: ");
            Lcd_Out(2,1,"Buscando... ");
            pos = saidaRNA(ent);
            Lcd_Out(2,1,"             ");
            dc = indentificaCor(29-pos);
            Lcd_Out(2,1,dc.nomeCor);
            relatorioLDR( v, dc );
            delay_ms(3000);
        }
    }
    Lcd_Cmd(_LCD_CLEAR);
}




double sigmoid( double x )
{
    double a;
    a = ( x < 0.0) ? ((-1) * x) : x;

    a = exp((-1.0)*a);
    a = (double)(1.0 + a);
    a = (double)(1.0 / a);

    return (x < 0.0) ? (1-a) : a;
}


int saidaRNA( double entradas[] )
{
    int nc, i, j, nnca, endPesos, nnc, maior;
    double resultAnterior[40], resultado[40], max;
    char texto[64];

    max = 0.0;
```

```
    maior = -1;
    nc = le_EEPROM(0x8000);
    EndPesos =  nc + 0x8001;
    for(i = 0; i < nc; i++)
    {
        if( i == 0 )
        {
            copiaVetor( resultado, entradas, le_EEPROM(0x8001));
        }
        else
        {
            nnca =  le_EEPROM(0x8001 + i - 1);
            nnc =  le_EEPROM(0x8001 + i);
            copiaVetor( resultAnterior, resultado, nnca);
            for(j = 0; j < nnc; j++)
            {
                resultado[j] = calculaNeuronio(nnca, &EndPesos, resultAnterior);
            }
        }
        PORTB.RB5 = ~PORTB.RB5;
    }
    j =  le_EEPROM(0x8001 + nc - 1);
    for(i = 0; i < j; i++)
    {
        if( resultado[i] > max)
        {
            max =  resultado[i];
            maior = i;
        }
        sprintf(texto,"%d -> %.3f\n",i,resultado[i]);
        enviaUSB(texto);
    }
    return maior;
}

double calculaNeuronio(int nPesos, int *endPesos, double entradas[])
{
    int i;
    double soma, valor;
```

```
    soma = 0.0;
    for(i = 0; i < nPesos; i++)
    {
         EEPROM_StrRead(*endPesos, sizeof(double), (unsigned char*)&valor);
         *endPesos +=  sizeof(double);
         soma += (double)(entradas[i]* valor);
    }
    return sigmoid(soma);
}


void copiaVetor( double vet1[], double vet2[], int tam )
{
    int i;
    for(i = 0; i < tam; i++)
    {
        vet1[i] = vet2[i];
    }
}


dadosCor indentificaCor( int cor )
{
      unsigned int pos, tmp, j;
      dadosCor dc;

      j = 0;
      pos = 0;
      do
      {
        PORTB.RB5 = ~PORTB.RB5;

        tmp = le_EEPROM(pos);
        delay_ms(10);
        if( tmp != 0 )
        {
             dc = leCorEEPROM( pos );
             dc.nomeCor[tmp] = '\0';
             if(j == cor)
             {
```

```
                return dc;
            }
            j++;
        }
        pos = pos + tmp + 6;
    } while (tmp != 0);
    return dc;
}
```

Listing A.7: sensorLDR.h

```
/*
 * sensorLDR.h
 */

#ifndef SENSOR_LDR_H
#define SENSOR_LDR_H

#include "interfaces_locais.h"

#define AMARELO  0
#define AZUL     1
#define VERDE    2
#define VERMELHO 3

#define modulo(v)   (v < 0 ) ? v * (-1) : v



char *decodificaCor( unsigned char cor );

void ascendeLed( unsigned char led );

void apagaLeds();

unsigned int valor_LDR(); /* captura dado */

void calibrar_LDR();

void testa_LDR();
```

```
void leituraLDR( unsigned int valores[] );

void enviaLeituraPelaUSB();

void amostragemPelaUSB();

void enviaMensagemUSB( char info[], unsigned int v[] );

void testaLeds();



#endif
```

Listing A.8: sensorLDR.c

```
/*
 * sensorLDR.c
 */


#include "sensorLDR.h"


char *decodificaCor( unsigned char cor )
{
    switch(cor)
    {
        case AMARELO  : return "Amarelo";
        case VERMELHO : return "Vermelho";
        case VERDE    : return "Verde";
        case AZUL     : return "Azul";
        default       : return "ERRO";
    }
}


void ascendeLed( unsigned char led )
{
    TRISA = 0x01;
    if (( led >= 0 ) && ( led  < 5 ))
```

```
    {
        PORTA = ( 0b00000010 << led );
    }
}

void apagaLeds()
{
    TRISA = 0x01;
    PORTA = 0b00000000;
}




void testaLeds()
{
    int i = 0;
    do
    {
        if( leBotoes() == 2 )
        {
            apagaLeds();
            i = (i + 1) %  4;
            PORTB.RB5 = ~PORTB.RB5;
        }
        ascendeLed( i );
        delay_ms(100);
    } while ( leBotoes() != 1 );
    apagaLeds();
}



unsigned int valor_LDR() /* captura dado */
{
    TRISA = 0x01;
    ADC_Init();
    delay_ms(30);
    return ADC_Read(0); /* retorna a diferenca de potencial no LDR */
```

```
}



void calibrar_LDR()
{
    int i;
    char texto[16], valor[16];
    TRISA = 0x01;


    for ( i = 0; i < 4; i++ )
    {
        strcpy(texto,"LED ");
        strcat(texto, decodificaCor(i) );
        ascendeLed(i);
        Lcd_Cmd(_LCD_CLEAR);
        Lcd_Out(2,1,texto);
        delay_ms(1000);
        sprintf(valor,"%d",valor_LDR());
        apagaLeds();
        strcpy(texto,"LDR: ");
        strcat(texto, valor );
        Lcd_Out(1,1,texto);
        delay_ms(1000);

    }
    delay_ms(1000);
    Lcd_Cmd(_LCD_CLEAR);
}


void leituraLDR( unsigned int valores[] )
{
    int i;
    TRISA = 0x01;

    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Fazendo Leitura");
    Lcd_Out(2,1,"Processando...");
```

```
    for ( i = 0; i < 4; i++ )
    {
        ascendeLed(i);
        delay_ms(280);
        valores[i] = valor_LDR();
        apagaLeds();
    }
    Lcd_Cmd(_LCD_CLEAR);
}



void testa_LDR()
{
    char texto[16], valor[16];
    TRISA = 0x01;
    apagaLeds();
    Lcd_Cmd(_LCD_CLEAR);
    do
    {
        strcpy(texto,"LDR: ");
        sprintf(valor,"%d   ",valor_LDR());
        strcat(texto, valor );
        Lcd_Out(1,1,texto);
    } while ( leBotoes() != 1 );
    delay_ms(500);
    Lcd_Cmd(_LCD_CLEAR);
}

void enviaLeituraPelaUSB()
{
    unsigned int v[4] = {0,0,0,0};

    PORTB.RB5 = ~PORTB.RB5;
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Lendo...");
    leituraLDR( v );
    enviaMensagemUSB("\nLeitura",v);
    PORTB.RB5 = ~PORTB.RB5;
    Lcd_Cmd(_LCD_CLEAR);
```

```
}

int media(int dados[], int nd)
{
    int i, soma = 0;
    for(i=0; i < nd; i++)
    {
        soma += dados[i];
    }
    return soma/nd;
}


void enviaMensagemUSB( char info[], unsigned int v[] )
{
    int i;
    char texto[64], valor[16];
    sprintf(texto,"\n%s={",info);
    for ( i = 0; i < 4; i++ )
    {
        strcat(texto,decodificaCor(i));
        strcat(texto,"=");
        if ( i == 3 ) sprintf(valor,"%d",v[i]);
        else          sprintf(valor,"%d, ",v[i]);
        strcat(texto,valor);
    }
    strcat(texto,"}");
    enviaUSB(texto);
}


void amostragemPelaUSB()
{
    char texto[64];
    int i,j;
    unsigned int dados[5][10];
    unsigned int v[4] = {0,0,0,0};

    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Amostrando ");
```

```
    for(i=0; i < 10; i++)
    {
        PORTB.RB5 = ~PORTB.RB5;
        leituraLDR( v );
        PORTB.RB5 = ~PORTB.RB5;
        Lcd_Out(1,1,"Amostrando ");
        for (j=0; j < 4; j++ )  dados[j][i] = v[j];
        sprintf(texto,"Amostra_%d",i);
        enviaMensagemUSB(texto,v);
        delay_ms(2500);
    }

    for ( i = 0; i < 4; i++ )
    {
        v[i] = media(dados[i],10);
    }
    enviaMensagemUSB("Amostra",v);
    Lcd_Cmd(_LCD_CLEAR);
}
```

Listing A.9: vizinhoMaisProx.h

```
/*
 * vizinhoMaisProx.h
 */


#ifndef VIZINHO_MAIS_PROXIMO_H
#define VIZINHO_MAIS_PROXIMO_H


#include "sensorLDR.h"
#include "interfaces_locais.h"




typedef struct centroide
{
        char nomeCor[12];
```

```
        unsigned int banda[4];
}dadosCor;



void relatorioLDR( unsigned int dadosLidos[], dadosCor dc );


dadosCor resultadoCor(unsigned int dados[]);

void classificarUsandoLDR();

void escreveCorEEPROM( unsigned int ender, dadosCor cor );

dadosCor leCorEEPROM( unsigned int ender );

void carregaEEPROM();

void insereCorEEPROM(dadosCor cor);

void enviaCoresAmostradas();

void sincroArq_EEPROM();


#endif
```

Listing A.10: vizinhoMaisProx.c

```
/*
 * vizinhoMaisProx.c
 */

#include "vizinhoMaisProx.h"


unsigned int escreveCorEEPROM( unsigned int ender, dadosCor cor )
{
    int i;
    unsigned int dado, dadoT, filtro7a0, filtra1e0, t;
    filtro7a0 = 0 | 0xFF;
```

```
    filtra1e0 = 0 | 0x03;
    t = strlen(cor.nomeCor);

    escreve_EEPROM(ender++, t);

    for(i = 0; i < t; i++)
    {
        dado = cor.nomeCor[i];
        escreve_EEPROM(ender++, dado);
    }

    for(i = 0; i < 4; i++)
    {
        dado = (cor.banda[i] >> 2) & filtro7a0;
        escreve_EEPROM(ender++, dado);
    }
    dado = 0;
    for(i = 0; i < 4; i++)
    {

        dadoT = cor.banda[i] & filtra1e0;
        dado = dado << ( (i>0)*2 );
        dado = dado | dadoT;
    }
    escreve_EEPROM(ender++, dado);
    delay_ms(DELAY_EEPROM);
    escreve_EEPROM(ender, 0);
    return ender;
}


dadosCor leCorEEPROM( unsigned int ender )
{
        dadosCor dc;
        unsigned int dado, filtra1e0, t;
        int i;
        filtra1e0 = 0x03;
        t = le_EEPROM(ender++);
        for(i = 0; i < t; i++)
        {
```

```
            dado = le_EEPROM(ender++);
            dc.nomeCor[i] = dado;
        }
        for(i = 0; i < 4; i++)
        {
            dc.banda[i] = le_EEPROM(ender++);
        }
        dado = le_EEPROM(ender);

        for(i = 3; i >= 0; i--)
        {
            dc.banda[i] =  (dc.banda[i] << 2) | (filtra1e0 & dado);
            dado  = dado >> 2;
        }
        return dc;
}




void insereCorEEPROM(dadosCor cor)
{
     int pos_livre;

     pos_livre = posicaoLivreEEPROM();

     pos_livre = escreveCorEEPROM(pos_livre , cor );

     setPosicaoLivreEEPROM( pos_livre );

     addNumeroCores_EEPROM( 1 );
}


void enviaCoresAmostradas()
{
      unsigned int pos, i, tmp;
      char texto[64], valor[16];
      dadosCor dc;
      pos = 0;
```

```
        Lcd_Cmd(_LCD_CLEAR);
        Lcd_Out(1,1,"Enviando Cores");
        enviaUSB("\n\nCores presentes na EEPROM, para uso da VMP:\n\n");
        do
        {
          PORTB.RB5 = ~PORTB.RB5;

          tmp = le_EEPROM(pos);
          delay_ms(50);
          if( tmp != 0 )
          {
                dc = leCorEEPROM( pos );
                dc.nomeCor[tmp] = '\0';
                strcpy(texto,dc.nomeCor);
                strcat(texto," ={");
                for ( i = 0; i < 4; i++ )
                {
                     strcat(texto,decodificaCor(i));
                     strcat(texto,"=");
                     if ( i == 3 ) sprintf(valor,"%d",dc.banda[i]);
                     else          sprintf(valor,"%d, ",dc.banda[i]);
                     strcat(texto,valor);
                }
                strcat(texto,"};\n");
                enviaUSB(texto);
          }
          pos = pos + tmp + 6;
        } while (tmp != 0);
        enviaUSB("\n-------------------------------------------\n\n");
}



dadosCor resultadoCor(unsigned int dados[])
{
      int i, pos, tmp, flag, valor, aux, aux2;
      dadosCor dc, ret;
      flag = 0;
      pos = 0;
```

```
    do
     {
       PORTB.RB5 = ~PORTB.RB5;
       tmp = le_EEPROM(pos);
       if( tmp != 0 )
       {
            aux = 0;
            dc = leCorEEPROM(pos );
            dc.nomeCor[tmp] = '\0';
            for ( i = 0; i < 4; i++ )
            {
                aux2 = dc.banda[i] - dados[i];
                aux2 = (aux2 < 0) ? ((-1)*aux2) : aux2;
                aux = aux + aux2 + aux2;
            }
            if ( (aux < valor) || (flag == 0) )
            {
               valor = aux;
               flag = 1;
               ret = dc;
            }
       }
       pos = pos + tmp + 6;
     } while (tmp != 0);
    return ret;
}



void classificarUsandoLDR()
{
    unsigned int v[4] = {0,0,0,0};
    dadosCor dc;

    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"COR: ");
    Lcd_Out(2,1,"?");
    while( leBotoes() != 1 )
    {
        if ( leBotoes() == 2 )
```

```
        {
            leituraLDR( v );
            Lcd_Out(1,1,"COR: ");
            Lcd_Out(2,1,"Buscando... ");
            dc = resultadoCor(v);
            Lcd_Out(2,1,"             ");
            Lcd_Out(2,1,dc.nomeCor);
            relatorioLDR(v,dc);
            delay_ms(3000);
        }
    }
    Lcd_Cmd(_LCD_CLEAR);
}

void relatorioLDR( unsigned int dadosLidos[], dadosCor dc )
{
    char texto[64], valor[16];
    int i;
    enviaUSB("\n\n\nDados Lidos pelo sensor LDR= ");
    strcpy(texto,"{");
    for ( i = 0; i < 4; i++ )
    {
        strcat(texto,decodificaCor(i));
        strcat(texto,"= ");
        if ( i == 3 ) sprintf(valor,"%d",dadosLidos[i]);
        else          sprintf(valor,"%d, ",dadosLidos[i]);
        strcat(texto,valor);
    }
    strcat(texto,"}\n");
    enviaUSB(texto);

    strcpy(texto,"Cor Aproximada: ");
    strcat(texto,dc.nomeCor);
    enviaUSB(texto);

    enviaUSB("\nInformacoes sobre cor aproximada= ");
    strcpy(texto,"{");
    for ( i = 0; i < 4; i++ )
    {
        strcat(texto,decodificaCor(i));
```

```
        strcat(texto,"= ");
        if ( i == 3 ) sprintf(valor,"%d",dc.banda[i]);
        else          sprintf(valor,"%d, ",dc.banda[i]);
        strcat(texto,valor);
    }
    strcat(texto,"}\n\n");
    enviaUSB(texto);
}


void sincroArq_EEPROM()
{
    char palavra[16];
    int estado, t_dado, sucesso;
    dadosCor dc;

    inicializa_EEPROM();
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Carreg. EEPROM");
    Lcd_Out(2,1,"Sinc.. com arq..");

    estado = 1;
    sucesso = 0;
    t_dado = 0;

    while(( leBotoes() != 1 ) && (sucesso == 0))
    {
        if ( leUSB(palavra) > 0 )
        {
          if ( (estado == 1) && (strcmp(palavra,"101") == 0))
          {
             estado = 2;
             enviaUSB(palavra);
          }
          else if ( (strcmp(palavra,"103") == 0) && (estado == 2))
          {
             estado = 1;
             enviaUSB(palavra);
             sucesso = 1;
          }
```

```
            else if ( estado == 2)
            {
               enviaUSB("102");
               if (t_dado == 0)
               {
                    strcpy(dc.nomeCor,palavra);
                    t_dado++;
               }
               else if ( t_dado > 0 )
               {
                    dc.banda[t_dado - 1] = atoi(palavra);
                    if ( t_dado == 4 )
                    {
                       t_dado = 0;
                       insereCorEEPROM( dc );
                       PORTB.RB5 = ~PORTB.RB5;
                       delay_ms(30);
                    }
                    else t_dado++;
               }
            }
            delay_ms(15);
          }
          delay_ms(15);
      }
      if( sucesso == 0 )
      {
         inicializa_EEPROM();
      }
}
```

A.6.2 Código para interface USB do computador

Listing A.11: pc_usb.c

```
/*

Implementa a interface de comunicao USB

*/
```

```
#include <stdio.h>
#include <wchar.h>
#include <string.h>
#include <stdlib.h>
#include "hidapi.h"

// Headers needed for sleeping.
#ifdef _WIN32
   #include <windows.h>
#else
   #include <unistd.h>
#endif

#define MAX_BUF 65


void copiaString(unsigned char buf[], char *str)
{
   unsigned int i;
   memset(buf,0,MAX_BUF);
   buf[0] = 0x0;
   for(i = 1; i < (strlen(str)+1); i++)
   {
      buf[i] = str[i-1];
   }
}

int comparaString(unsigned char buf[], char *str)
{
   int i, t = strlen(str);
   if ( t <= 0 ) return 0;
   for(i = 0; i < t; i++)
   {
      if ( buf[i] != str[i] )
      {
         return 0;
      }
   }
   return 1;
}
```

```
int envioComResposta( hid_device *handle,
            unsigned char buf[], char *envia, char *recebe )
{
    int res;
   copiaString(buf, envia);
   if (hid_write(handle, buf, MAX_BUF) < 0)
      printf("Falha na escrita de (%s)\n",envia);

   res = hid_read(handle, buf, sizeof(buf));
   if (( res > 0) && (comparaString(buf,recebe) == 1 ))
   {
      return 1;
   }
   return 0;
}


void sincronizaComArquivo( hid_device *handle, unsigned char buf[], char *arquivo )
{
   char dadoLido[5][16];
   FILE *arq;
   int j;

   printf("\n\nSincronizar EEPROM com arquivo %s\n",arquivo);
   printf("Coloque o dispositivo na opcao: Sinc. Arq x Mem\n");

   if (( arq = fopen (arquivo, "r")) == NULL )
   {
        printf("Desculpe, o arquivo nao pode ser aberto.");
        exit(1);
    }

   buf[0] = 0x0;
   int conf = 1;
   conf = envioComResposta( handle, buf, "101", "101" );
   while( ( !feof(arq) ) && ( conf == 1 ) )
   {
      fscanf(arq,"%s %s %s %s %s\n",dadoLido[0],
```

```
                    dadoLido[1], dadoLido[2], dadoLido[3], dadoLido[4]);
        for (j = 0; j < 5; j++)
        {
            conf = envioComResposta( handle, buf, dadoLido[j], "102");
        }
        printf(".");
    }
    printf("\n");
    if ( conf != 1 )
    {
        printf("Falha na comunicacao 1 ou 2!\n");
    }
    conf = envioComResposta( handle, buf, "103", "103");
    if ( conf != 1 )
    {
        printf("Falha na comunicacao 3!\n");
    }
    else printf("\n Fim da Sincronizacao.\n");
}

void sincronizaArquivoPesos( hid_device *handle, char *arquivo )
{
    char dadoLido[5][16];
    unsigned char buf[MAX_BUF];
    FILE *arq;
    int j;

    printf("\n\nSincronizar EEPROM com arquivo que contem os pesos da RNA\n",arquivo);
    printf("Coloque o dispositivo na opcao: Sinc. Pes. x Mem\n");

    if (( arq = fopen (arquivo, "r")) == NULL )
    {
        printf("Desculpe, o arquivo nao pode ser aberto.");
        exit(1);
     }

    buf[0] = 0x0;
    int conf = 1;
    int n_camadas;
```

```
conf = envioComResposta( handle, buf, "201", "201" );
if ( conf == 1 )
{
   fscanf(arq,"%s\n",dadoLido[0]);
   conf = envioComResposta( handle, buf, dadoLido[0], "202" );
   n_camadas = atoi(dadoLido[0]);
   for (j = 0; ((j < n_camadas) && (conf == 1)); j++)
   {
      fscanf(arq,"%s\n",dadoLido[0]);
      conf = envioComResposta( handle, buf, dadoLido[0], "203" );
      printf("%s, ",dadoLido[0]);
   }
   printf("\n");
}

if ( conf != 1 )
{
   printf("Falha na comunicacao!\n");
   exit(1);
}

while( ( !feof(arq) ) && ( conf == 1 ) )
{
   fscanf(arq,"%s\n",dadoLido[0]);
   conf = envioComResposta( handle, buf, dadoLido[0], "204" );
   printf("%s, ",dadoLido[0]);
}
printf("\n");
if ( conf != 1 )
{
   printf("Falha na comunicacao 1 ou 2!\n");
   exit(1);
}
conf = envioComResposta( handle, buf, "205", "205");
if ( conf != 1 )
{
   printf("Falha na comunicacao 3!\n");
   exit(1);
}
else printf("\n Fim da Sincronizacao.\n");
```

```
}


void leitor( hid_device *handle )
{
   unsigned char buf[MAX_BUF];
   int res = 0;
   printf("Iniciando monitoremento dos dados recebidos pela porta USB:\n\n");
   do
   {
      memset(buf,0x00,sizeof(buf));
      res = hid_read(handle, buf, sizeof(buf));
      if (res < 0)
         printf("Unable to read()\n");
      else if ( res > 0 )
      {
         printf("%s", buf);
      }
      #ifdef WIN32
      Sleep(100);
      #else
      usleep(500*1000);
      #endif
   } while (res >= 0);
}




int main(int argc, char* argv[])
{
   unsigned char buf[MAX_BUF];
   hid_device *handle;
   struct hid_device_info *devs, *cur_dev;
   int op, presente = 0;


#ifdef WIN32
```

```
    UNREFERENCED_PARAMETER(argc);
    UNREFERENCED_PARAMETER(argv);
#endif


    if ( argc < 2 )
    {
        printf("\nFaltou paramentros:\n");
        printf("%s opcao arquivo(somente se necessario)\n",argv[0]);
        printf("opcoes = \n 1 - Leitor da USB \n \
                  2 - Sincronizar arquivo com EEPROM\n \
                  3 - Sincronizar arquivo de pesos com a EEPROM}");
        return 1;
    }
    char arq[30];
    op = atoi(argv[1]);
    if( op > 1)
    {
        strcpy(arq,argv[2]);
    }

    devs = hid_enumerate(0x0, 0x0);
    cur_dev = devs;
    while (cur_dev)
    {
        if (( cur_dev->vendor_id == 0x1234 ) && (cur_dev->product_id == 0x01))
        {
            presente = 1;
        }
        cur_dev = cur_dev->next;
    }
    hid_free_enumeration(devs);

    if ( presente == 0 )
    {
        printf("\nO dispositivo buscado nao esta conectado em uma porta USB.\n");
        return 1;
    }
```

```
    // Open the device using the VID, PID,
    // and optionally the Serial number.
    ////handle = hid_open(0x4d8, 0x3f, L"12345");
    handle = hid_open(0x1234, 0x01, NULL);
    if (!handle)
    {
        printf("unable to open device\n");
        return 1;
    }

    if( op == 2 ) sincronizaComArquivo( handle, buf, arq );
    else if ( op == 1 ) leitor( handle );
    else if ( op == 3 ) sincronizaArquivoPesos( handle, arq );

    hid_close(handle);

    /* Free static HIDAPI objects. */
    hid_exit();

    return 0;
}
```