UNICEUB – CENTRO UNIVERSITÁRIO DE BRASÍLIA
FAET – FACULDADE DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Transmissão alternativa de dados utilizando monitor LCD e WebCam

RENATA MONTEIRO FERREIRA DA COSTA

Professora Orientadora: Maria Marony Sousa F. Nascimento

Brasília
2007

RENATA MONTEIRO FERREIRA DA COSTA

Transmissão Alternativa de Dados Utilizando Monitor LCD e WebCam

Orientador: Prof. Maria Marony Sousa F. Nascimento.

Monografia apresentada ao Centro Universitário de Brasília, para obtenção do título de Bacharel em Engenharia da Computação.

Brasília-DF
Junho de 2007.

RESUMO

Neste trabalho foi verificada a viabilidade da realização de uma transmissão alternativa de dados, utilizando como emissor um monitor do tipo *Liquid Crystal Display* – LCD e como receptor uma WebCam interagindo com o software MATLAB. Este trabalho foi baseado na monografia e protótipo desenvolvido pelo Engenheiro João Paulo Barbosa, ex-aluno do Uniceub, que utilizou-se de um foto-sensor para a recepção dos dados. O presente trabalho sugere a possibilidade de utilização da WebCam como forma de capturar a recepção dos dados.

Palavras-chave: Transmissão de dados, monitor LCD, WebCam, RGB, HSV, USB, MATLAB e VB .NET.

ABSTRACT

This project objective is to verify the viability of realizing an alternative data transmission, using as a transmitter a LCD, Liquid Crystal Display, monitor, and as a receptor a WebCam that interacts with the MATLAB software. This project was based in the monograph and the prototype developed by the Engineer João Paulo Barbosa, a UniCEUB former student, who used a foto-sensor for the data reception. The present project suggests the possibility of using a WebCam as a way of capturing the data reception.

Key-Word: Data transmission, LCD monitors, WebCam, RGB, HSV, USB, MATLAB e VB .NET.

AGRADECIMENTOS

Agradeço a meus pais, a meu filho, ao Marcelo, aos amigos João Paulo, Ana Paula, Toni Gledson, Francis e Vanessa, além de todos os colegas e professores que me apoiaram.

SUMÁRIO

RESUMO III
ABSTRACT IV
AGRADECIMENTOS V
SUMÁRIO VI
LISTA FIGURAS VIII
LISTA DE SIGLAS X
1. INTRODUÇÃO 1
1.1. Contextualização do projeto 1
1.2. Motivação 2
1.3. Estrutura do trabalho 3
2. REVISÃO BIBLIOGRÁFICA 4
2.1. Sistemas de cores 4
2.1.1. Imagem digital 4
2.1.2. Relações entre pixels 6
2.1.2.1. Vizinhança 6
2.1.2.2. Conectividade 6
2.2. Cor 7
2.2.1. RGB 7
2.2.2. Escalas de cinza 9
2.2.3. Sistema YUV 11
2.2.4. Modelos HSV 13
2.3. API DirectX 16
2.4. Técnica de Double Buffering 17
2.5. Técnica de page-flipping 19
2.6. Caracteres ASCII 20
3. TRANSMISSOR 21
3.1. Monitor de vídeo LCD 21
3.2. Software transmissor 24
3.2.1. A Plataforma .NET 25
3.2.2. Pontos chaves do código 26
3.2.3. Freqüência de transmissão 29
4. RECEPTOR 30
4.1. MATLAB (MATrix LABoratory) 30
4.1.1. Toolbox de aquisição de imagens 31
4.2. Webcam 31
4.2.1. Histórico 32
4.2.2. Tecnologia 33
4.2.3. Modelo da WebCam utilizada no protótipo 34
4.3. Porta USB 35
4.3.1. Características Elétricas 37
4.3.2. Protocolo USB 38
4.3.3. Conexão de dispositivos USB 39
4.3.4. Tipos de Fluxo de Dados 40

4.4. Capturando imagens com a webcam 41
5. TESTES REALIZADOS 46
6. CONSIDERAÇÕES FINAIS 47
6.1. Dificuldades encontradas 47
6.2. Resultados obtidos 48
6.3. Conclusões 48
6.4. Sugestões para trabalhos futuros 49
7. REFERÊNCIA BIBLIOGRÁFICA 50
APÊNDICES 52
Apêndice 1 – Tabela de testes 53
Apêndice 2 – Código do programa transmissor 57
Apêndice 3 – Códigos dos scripts do receptor 65

LISTA FIGURAS

Figura 1.1: Representação esquemática do projeto 1

Figura 2.1 Exemplo de como uma imagem é afetada pela taxa de amostragem (fonte: Koenigkcan, Luciano Vieira) 5

Figura 2.1 Exemplo de como uma imagem é afetada pela utilização de diversos níveis de cores (Fonte: Koenigkcan, Luciano Vieira) 6

Figura 2.3 Representação esquemática das cores ativas no modelo RGB (Fonte: www.wikipedia.com) 8

Figura 2.4 Representação esquemática do modelo RGB (fonte: Koenigkcan, Luciano Vieira) 9

Figura 2.5: Representação de uma imagem colorida, em tons de cinza e preto-e-branco (fonte: www.inf.pucrs.br) 10

Figura 2.6: Representação esquemática do modelo YUV (Fonte: Koenigkcan, Luciano Vieira) 12

Figura 2.7: Representação esquemática do modelo de cores HSV (Fonte: www.wikipedia.com) 13

Figura 2.8: Representação esquemática em forma de cone do modelo de cores HSV (Fonte: GONZALES, Rafael C.; WOODS, Richards E.; EDDINS, Steven L) 14

Figura 2.9: Representação esquemática em forma de cone do modelo de cores HSV (Fonte: http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space) 14

Figura 2.10: Gama de cores representadas no eixo H (Fonte: Fonte: http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space) 15

Figura 2.11: Representação esquemática da transformação de HSV para H2SV (Fonte: http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space) 15

Figura 2.12: Desenho esquemático da técnica de Double Buffering (Fonte: http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html) 18

Figura 2.13: Desenho esquemático da técnica de Page Fliping (Fonte: http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html) 20

Figura 3.1: Representação esquemática sobre o posicionamento das moléculas de cristal líquido (Fonte: www.clubedohardware.com.br) 22

Figura 3.2: Representação esquemática da .NET FRAMEWORK (Fonte: hppt://msdn2.microsoft.com) 25

Figura 3.3: Fluxograma do código desenvolvido 26

Figura 4.1: Imagem da WebCam utilizada no protótipo 35

Figura 4.2: Topologia USB (Fonte: www.ufrj.br) e Exemplo prático de topologia (Fonte: www.rogercom.com) 36

Figura 4.3: Representação esquemática de um cabo de transmissão elétrica da USB (Fonte: www.ufrj.br) 37

Figura 4.4: Imagem de um cabo USB (Fonte: www.rogercom.com) 37

Figura 4.5: Estrutura de uma estrutura de array multidimensional (Fonte: Help do MATLAB) 42

Figura 4.6: Imagem capturada da WebCam e sua representação esquemática das zonas de transmissão 43

LISTA DE SIGLAS

APIs - Application Programming Interface - Interface de Programação de Aplicativos

ASCII - American Standard Code for Information Interchange

CCD - charge-coupled device

CMOS - complementary metal-oxide-semiconductor

CRT - Cathode Ray Tube – Tubo de Raios Catódicos

HSV – Hue Saturation Value – Matiz Saturação e Brilho

LCD - Liquid Crystal Display - Monitores de Cristal Líquido

MATLAB - MATrix LABoratory – Laboratório de Matrizes

NRZI - No Return to Zero Inverted – Inversão de não retorno ao zero

NTSC – National Television Standards Committee - Comitê de padrões nacional de televisões

PAL – Phase Alternating Line – Fase de Linhas alternadas

PNG - Portable Network Graphics – Rede de Gráficos Portáveis

RGB – Red Green Blue – Vermelho Verde Azul - Sistema de cores comumente utilizado.

SECAM – Séquentiel couleur à mémoire – Cores seqüenciais com memória

USB - Universal Serial Bus – Barramento serial universal

YCbCr – Sistema de cores onde o Y representa a luminosidade, o Cb representa um vetor de azul e o Cr um vetor de vermelho

YPbPr – Sistema de cores utilizados em vídeos analógicos e utiliza a mesma divisão do YCbCr

YUV – Modelo de cores onde o Y representa a luminosidade e o U e V representam a cromaticidade

1. INTRODUÇÃO

1.1. Contextualização do projeto

Este projeto apresenta um modelo de transmissão alternativa de dados, entre microcomputadores, utilizando, como emissor de dados, um monitor LCD, e para recepção de dados, uma câmera do tipo Webcam e o software MATLAB para o processamento da imagem.

Este modelo utiliza uma transmissão de dados do tipo texto, em que o monitor de cristal líquido - LCD emissor transmite uma variação de quatro cores (verde, azul, amarelo e vermelho), efetuada a captura destas cores pela webcam com o auxílio do software MATLAB é realizada uma conversão das cores em bits e posteriormente sua conversão em caracteres ASCII. Desta forma, ao final, a mensagem transmitida pelo emissor é demonstrada de forma clara.

Neste projeto não foi desenvolvido nenhum novo componente físico, foi utilizada uma WebCam comum, para o seu funcionamento. Quanto ao desenvolvimento, foi criado um software transmissor na linguagem VB .NET e scripts de captura e tratamento da imagem no MATLAB.

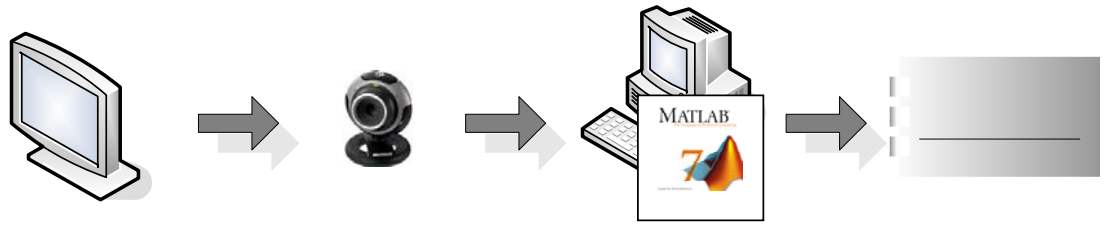

Figura 1.1: Representação esquemática do projeto

1.2. Motivação

O primeiro trabalho de transmissão alternativa de dados no Centro Universitário de Brasília foi desenvolvido pelo então aluno Thiago Mitsuka. No trabalho original eram utilizados dois níveis de cores, claro e escuro, possibilitando, assim, a transmissão de um único bit por vez, limitando a velocidade de transmissão à velocidade do monitor.

Posteriormente, o aluno João Paulo Barbosa desenvolveu um avanço significativo no projeto, adotando quatro níveis de cores (branco, cinza claro, cinza escuro e preto) dobrando a velocidade de transmissão para dois bits por variação de telas, além de implementar a verificação de erros utilizando a paridade combinada.

Ao acompanhar o desenvolvimento do projeto do então aluno João Paulo, foi fácil vislumbrar as diversas aplicações para uma forma de transferência de dados alternativa. Aplicações que vão desde a comunicação através de computadores sem nenhuma possibilidade de comunicação física para a transferência, até aplicações para as câmeras dos aparelhos celulares.

Inicialmente, a proposta da continuação do projeto do ex-aluno João Paulo era efetuar melhorias para a utilização de monitores LCD, além da utilização da porta USB do computador (ao invés da utilização da porta serial utilizada nos dois trabalhos anteriores). No entanto, no decorrer do desenvolvimento, surgiram algumas dificuldades, principalmente no que diz respeito à luminosidade dos monitores LCD, que utilizam padrões bem mais baixos que os monitores do tipo CRT. Para a resolução deste problema a alternativa seria retornar aos níveis de transmissão utilizados no primeiro trabalho.

Durante o decorrer dos estudos, foi encontrada a possibilidade de captura de imagens envolvendo a utilização de uma WebCam e o software MATLAB. Ao vislumbrar a possibilidade de captura de imagens diretamente através do software, rapidamente surgiu a idéia de utilizar a câmera como forma de recepção e tratar os dados através do MATLAB. Além disto, surgiu a idéia de utilizar escalas de cores para a transmissão. Durante pesquisas realizadas, encontrou-se diversos sistemas de cores que trabalham com uma separação entre matiz da cor, luminosidade e saturação,

resolvendo assim o problema de luminosidade do ambiente, já que um azul claro e um azul escuro continuam sendo, em sua essência azul, (representado pela componente matiz).

As técnicas de *Double Buffering* e *Page Flipping* (descritas posteriormente) utilizadas nos trabalhos anteriores foram aproveitadas neste projeto.

1.3. Estrutura do trabalho

Além deste capítulo introdutório, este trabalho está estruturado em seis capítulos distribuídos da seguinte forma:

No **Capítulo 2** é apresentada uma revisão bibliográfica onde são abordados os conceitos essenciais para o entendimento do projeto e seus softwares envolvidos.

No **Capítulo 3** é abordado, de forma específica, o módulo transmissor de dados, suas características e tecnologias utilizadas.

No **Capítulo 4** são detalhadas as informações sobre o módulo receptor, e como é realizada a decodificação dos bits para obter a mensagem original.

No **Capítulo 5** são apresentados os testes realizados com o protótipo.

No **Capítulo 6** são apresentadas as considerações finais e conclusões sobre o trabalho proposto.

2. REVISÃO BIBLIOGRÁFICA

2.1. Sistemas de cores

Segundo o modelo definido por Gonzales e Woods, (GONZALES, WOODS, 1993) uma imagem pode ser modelada em função da intensidade luminosa bidimensional em que o valor ou a amplitude nas coordenadas espaciais denota a intensidade da imagem naquelas coordenadas.

As imagens percebidas pela visão humana, por exemplo, é o resultado da reflexão da luz nos objetos e pode ser caracterizada por duas componentes:

- Quantidade de luz incidente nos objetos, chamada de iluminância;
- Percentual de luz refletida pelos objetos, chamada de reflectância.

2.1.1. Imagem digital

Partindo do conceito de imagem, tem-se como representação digital de uma imagem um conjunto de amostras igualmente espaçadas e organizadas na forma de uma matriz M x N

$$f(x,y) \simeq \begin{pmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{pmatrix} \quad (2.1)$$

A matriz representada pela expressão 2.1, é denominada imagem digital e cada um de seus elementos representa um pixel, que é o nome dado a cada elemento constituído da imagem. A razão do número de pixels que constituem uma imagem e a área representada na imagem define a amostra da imagem ou resolução espacial R da imagem digital. Essa razão é definida pela expressão 2.2:

$$R = \frac{\text{Número de Pixels}}{\text{Unidade de área}} \quad (2.2)$$

As informações de brilho da imagem, da mesma forma como as informações dos pixels, também são discretizadas para que sejam apresentadas em formato digital. Este

processo é também conhecido como quantização dos níveis de cinza e definem o número máximo de tons de cinza que a matriz poderá assumir.

A fidelidade com que a imagem será representada está intimamente ligada à resolução espacial (ou taxa de amostragem) e a quantização em níveis de cinza. À medida que estes valores são diminuídos, a imagem sofre uma degradação.

Por exemplo: uma imagem que possa ser representada por um gráfico contínuo de tons de luminosidade, será quantizada de forma a obtermos níveis padrões para estas amostragens. Se estes tons puderem assumir alguns poucos níveis, acarretará na transição abrupta entre as cores.

Portanto, as taxas de amostragem da imagem, aquém do necessário podem ocasionar a não representação de detalhes finos da imagem, além de tornar visível o efeito da discretização, dando aos objetos da cena o aspecto de serrilhamento em suas bordas. A Figura mostram imagens de um mesmo objeto, obtidas com a mesma câmera, e na mesma posição. No entanto, alteram-se a taxa de amostragem (respectivamente 70 pixels/cm, 35 pixels/cm e 13 pixels/cm e níveis de quantização de cinza).

Figura 2.1 Exemplo de como uma imagem é afetada pela taxa de amostragem (fonte: Koenigkcan, Luciano Vieira)

Outro problema comumente encontrado é a representação da imagem em níveis abaixo do necessário. Com isso ocorre a não representação de algumas variações de intensidades dos pixels, podendo vir a criar falsas bordas e, com isso, uma representação infiel da imagem. A Figura 2.1 demonstra uma mesma imagem representadas por 256, 16 e 4 níveis de cinza. Neste caso podemos observar que à medida que diminuem os níveis de cinza utilizados, mais evidente se tornam os efeitos que levam ao surgimento das falsas bordas em regiões homogêneas.

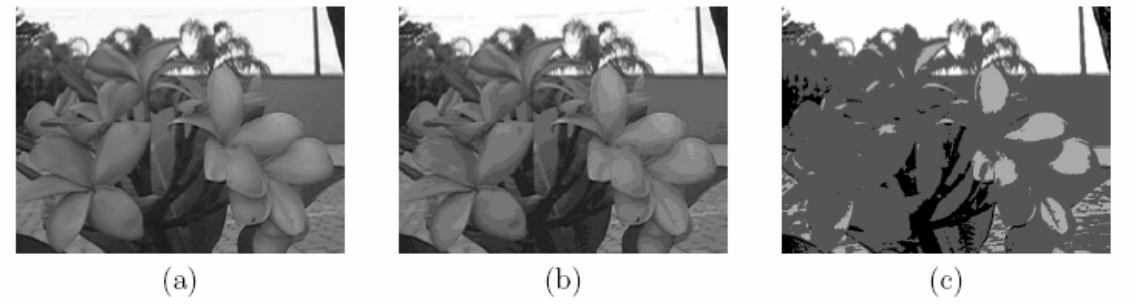

Figura 2.1 Exemplo de como uma imagem é afetada pela utilização de diversos níveis de cores (Fonte: Koenigkcan, Luciano Vieira)

2.1.2. Relações entre pixels

2.1.2.1. Vizinhança

Para a extração de informações em imagens digitais é necessário o estabelecimento do conceito vizinhança: um pixel possui 2 vizinhos horizontalmente e 2 verticalmente; caso o pixel esteja localizado na borda da imagem alguns de seus vizinhos estarão localizados fora da imagem. A vizinhança também pode ser definida como os pixels conectados diagonalmente com o pixel em questão, tendo assim 8 vizinhos (os 4 do conjunto vertical-horizontal e mais 4 do conjunto de diagonais).

2.1.2.2. Conectividade

Segundo Koenigkan, o conceito de conectividade tem sua importância devido à identificação de bordas e elementos de região em uma imagem.

A conexão se dá quando 2 pixels são de alguma forma vizinhos além de satisfazerem a alguns critérios de similaridade, entre estes podemos citar a utilização do mesmo nível de cinza.

2.2. Cor

É possível encontrar, em várias fontes e diversos sítios na internet, a utilização de três critérios para identificar as cores. São elas:

- Brilho: indica a noção de luminosidade, independente da cor apresentada;
- Matiz: é um atributo associado ao comprimento de onda predominante em uma mistura de ondas de luz representando a cor dominante percebida pelo observador;
- Saturação: é relativo ao grau de pureza da cor, ou seja, a quantidade de luz branca misturada a uma matiz. Quanto maior a saturação, menor é a quantidade de luz branca presente na imagem;

A Matiz e a Saturação, em conjunto, são conhecidas como cromaticidade. Para caracterizar uma cor, são necessárias a cromaticidade aliada ao brilho da imagem.

Para podermos utilizar uma cor em uma imagem digital, podemos utilizar os chamados modelos de cores, que têm por objetivo facilitar a especificação da informação de cor de maneira padronizada e consensual.

São descritas, a seguir, alguns padrões de representação de cores normalmente utilizados em imagens digitais.

2.2.1. RGB

Este é o modelo mais comumente usado em imagens digitais, onde as cores são representadas através da combinação das suas cores primárias: vermelho (R), verde (G) e azul (B). Neste modelo, as proporções que essas cores são aplicadas caracterizam a cor e assume-se que esses valores estejam entre 0 e 1. Atualmente, a maioria dos dispositivos de captura de imagens utilizam esta representação nativamente, o que torna esta representação extremante importante.

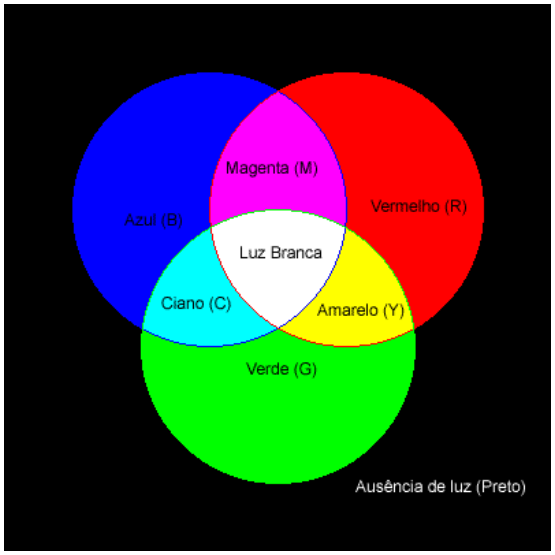

Figura 2.3 Representação esquemática das cores ativas no modelo RGB (Fonte: www.wikipedia.com)

No entanto, é importante ressaltar que este modelo não define exatamente o que é “vermelho”, “verde” e “azul”, de forma que os mesmos valores podem representar cores muito diferentes em diversos monitores.

Uma aplicação comum deste sistema de cor é em dispositivos que utilizem o fundo de tela preto, como em televisões e monitores CRT.

Neste sistema de cores, cada pixel pode ser representado com valores de vermelhos, verdes e azuis, e estes valores são convertidos nas cromaticidades e luminâncias a serem exibidos.

Para este sistema são utilizados 8 bits para cada cor, possibilitando 256 variações para cada uma. Desta forma é possível a representação de aproximadamente 16 milhões de cores.

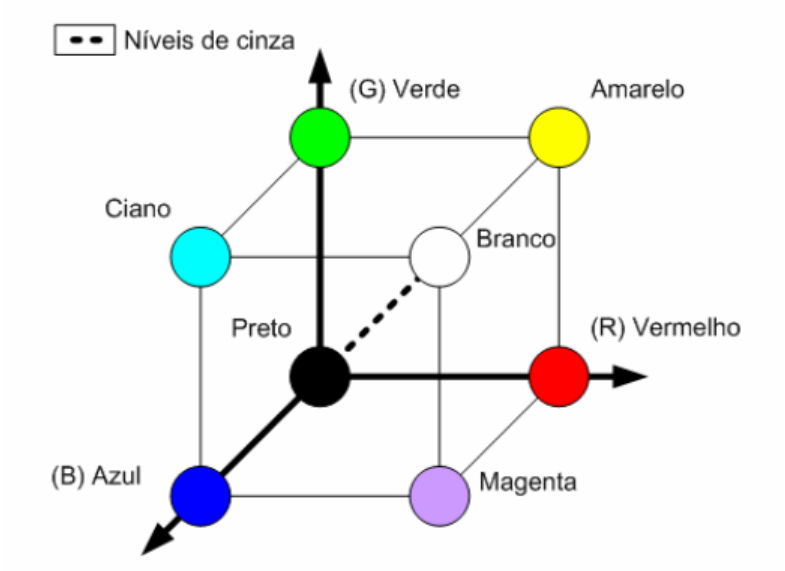

Figura 2.4 Representação esquemática do modelo RGB (fonte: Koenigkcan, Luciano Vieira)

Importante ressaltar outra característica do modelo RGB: a falta de linearidade e proporção. Se reproduzirmos no monitor a cor representada por (127, 127, 127) não teremos uma cor com intensidade e luminosidade correspondente à metade do valor representado por (255, 255, 255)

2.2.2. Escalas de cinza

Em sistemas computacionais, uma imagem em escala de cinza (*"grayscale"* ou *"greyscale"*) é representada por uma única amostra, variando do preto, na intensidade mais fraca ou escura, ao branco, de intensidade mais forte. As imagens em escalas de cinza diferem das imagens em preto-e-branco no sentido que existem várias escalas entre as duas cores preto e branco. No entanto, podemos observar, em alguns momentos, o termo preto-e-branco sendo utilizado para referenciar as escalas de cinza. Por exemplo: quando fala-se em fotografias preto-e-branco estão referindo-se na realidade a escalas de cinza. A Figura 2.2.2-1 demonstra uma mesma imagem de três formas: colorida, escala de cinza e preto-e-branco, respectivamente.

Figura 2.5: Representação de uma imagem colorida, em tons de cinza e preto-e-branco (fonte: www.inf.pucrs.br)

As imagens em escalas de cinza freqüentemente provém do resultado da medição da intensidade da luz em cada pixel em uma faixa do espectro eletromagnético, como por exemplo, a luz visível.

Imagens produzidas para visualização humana são geralmente armazenadas em 8 bits por pixel. Consequentemente é possível armazenar 255 tons de cinza. No entanto, esta distribuição não é linear.

A exatidão fornecida por esta escala não é suficiente para evitar totalmente os efeitos de serrilhamento das bordas. Por outro lado, são muito convenientes para a programação.

Para imagens onde a precisão destes níveis de cores é necessária, são utilizados sensores que armazenam as escalas de cinza de 10 a 12 bits por pixel, dificulta a programação. Para resolver este impasse, as cores podem ser armazenadas em 16 bits por amostra. Porém a maioria dos navegadores e programas em geral ainda trabalham com a codificação de 8 bits/pixel. Um exemplo de formato que já utiliza a escala em 16 bits é o *Portable Network Graphics* – PNG.

2.2.3. Sistema YUV

O modelo YUV define um espaço de cores em termos de luminância (iluminação ou ainda claridade) e cromatização, ou simplesmente a cor da imagem, onde Y representa a luminosidade; e U e V representam a cromaticidade.

Este modelo é utilizado como padrão para a transmissão em TV's européias além de PAL (*Phase Alternating Line*), NTSC (*National Television Standards Committee*) e SECAM (*"Séquentiel couleur à mémoire"* ou cores seqüenciais com memória). O sistema preto-e-branco e tons de cinza utilizam como única informação a luminância (Y). A essa informação foram adicionadas as informações de cores (U e V). Com a utilização do sistema de iluminação, e as informações de cores, é possível exibir uma imagem de ambas as formas, colorida ou simplesmente em preto-e-branco. A transformação de RGB para YUV ocorre da seguinte forma:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (2.2)$$

A maior vantagem é que a informação sobre a cor e sobre a luminosidade podem ser processadas separadamente. Desta forma, imagens exibidas somente com a componente Y são visualizadas como imagens na escala de cinza. Para o protótipo proposto, o grande ganho deste sistema de cores é relativo à separação da componente de luminância, isolando, desta forma, a questão de ajustes de luminosidades do local de transmissão. Evita-se assim, que cores possam ser trocadas devido à luminosidade do ambiente.

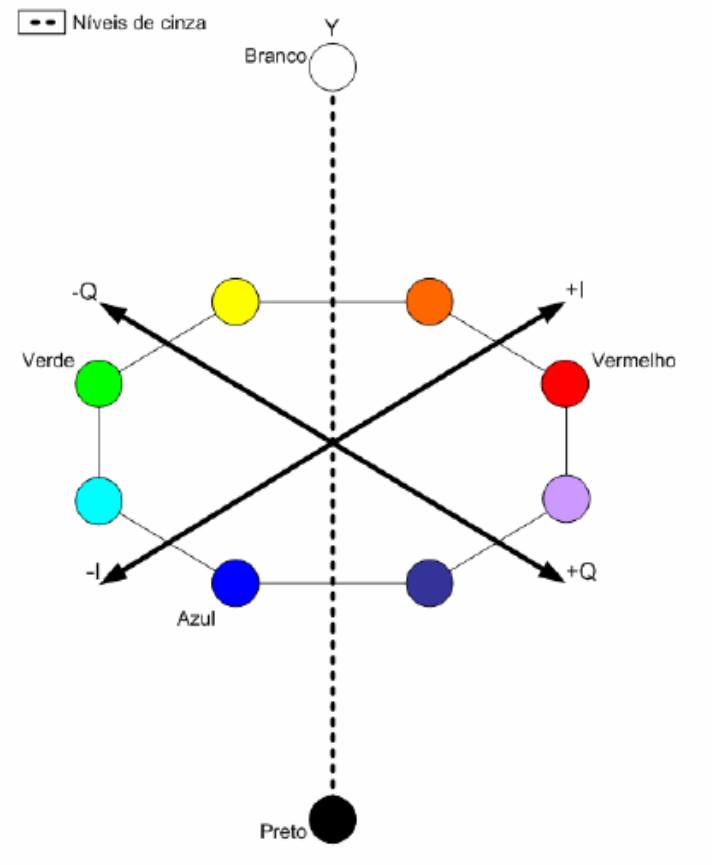

Figura 2.6: Representação esquemática do modelo YUV (Fonte: Koenigkcan, Luciano Vieira)

O modelo YUV representa mais fielmente a percepção de cores pelos olhos humanos do que o sistema de cores RGB, mas não é tão preciso quanto o modelo HSV e HSL.

Vale lembrar que muito comumente o termo YUV é utilizado erroneamente ao invés de Y'CbCr. De fato, em se tratando de vídeos digitais, normalmente onde fala-se YUV, na realidade trata-se de Y'CbCr (algumas vezes abreviado como YCC). Outro sistema semelhante é o YPbPr. Esses dois sistemas não serão abordados no escopo deste trabalho; no entanto, maiores informações podem ser obtidas em: http://en.wikipedia.org/wiki/YCbCr

2.2.4. Modelos HSV

O modelo HSV - Hue (H) Saturation (S) e Value (V), também conhecido como HSB (Hue, Saturation, Brightness), define o sistema de cores em três componentes:

- Hue – representa a cor ou matiz
- Saturation – representa a pureza da cor
- Value – Brilho

Este modelo foi criado em 1978 por Alvy Ray Smith. É representado por uma transformação não linear do sistema de cores RGB, e pode ser utilizada em progressões de cores.

É comumente utilizado em aplicações de computações gráficas e em vários contextos de aplicações onde são necessários elementos gráficos particulares.

A representação da matiz é geralmente representada por uma região circular e um triângulo utilizado para representar a saturação e brilho da imagem, conforme demonstrado na Figura 2.2.4-1 retirada do site Wikipedia.

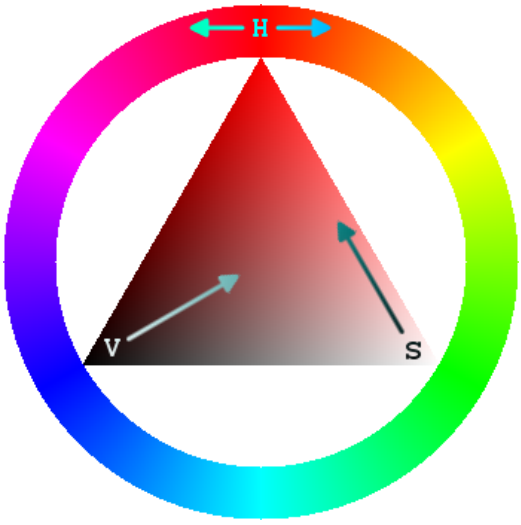

Figura 2.7: Representação esquemática do modelo de cores HSV (Fonte: www.wikipedia.com)

Tipicamente, utiliza-se também um eixo vertical para representar o brilho e o eixo horizontal para representar a saturação. A matiz é representada pelo ângulo do cone, conforme pode ser verificado nas Figuras 2.8 e 2.9.

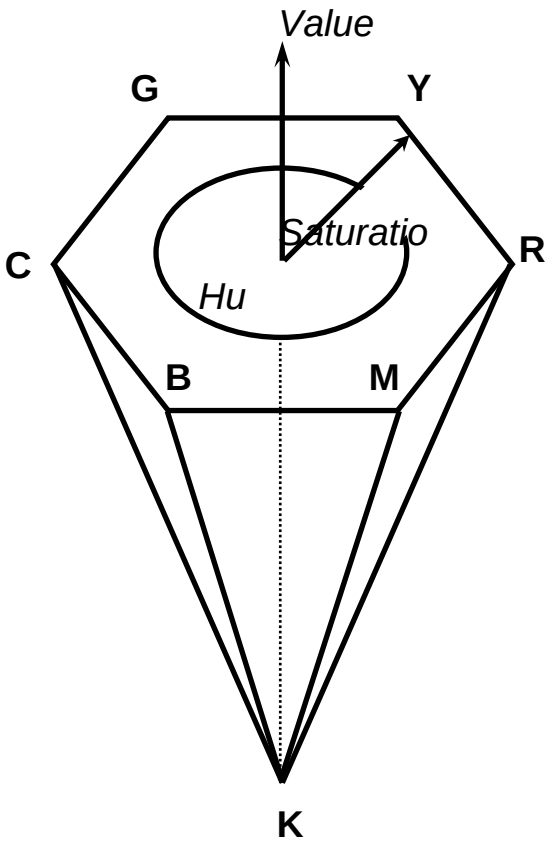

Figura 2.2.4: Representação esquemática em forma de cone do modelo de cores HSV (Fonte: GONZALES, Rafael C.; WOODS, Richards E.; EDDINS, Steven L)

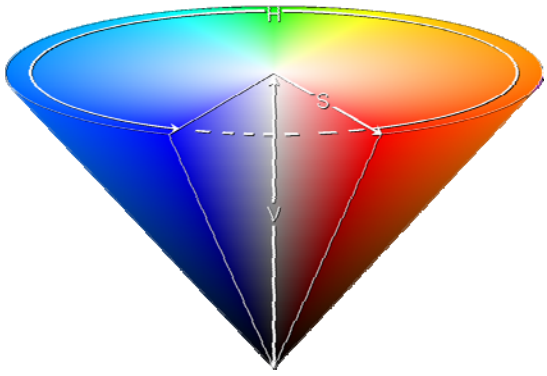

Figura 2.2.4: Representação esquemática em forma de cone do modelo de cores HSV (Fonte: http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space)

A maior dificuldade com relação ao espaço de cores HSV é relativa aos valores que podem ser assumidos pela matriz. Por exemplo: valores das cores vermelhas assumem valores próximos ao zero, isto pode ser um problema se o desejo é realizar

cálculos com as cores. A cor vermelha pode assumir valores entre 0 e 30 além de valores entre 330 a 360, como demonstrado na Figura 2.2.4 retirada do site da University of Southern California:

Figura 2.10: Gama de cores representadas no eixo H (Fonte: Fonte: http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space)

Para solucionar este tipo de problema, é comum a transformação da componente H em outras duas componentes, formando um eixo cartesiano. A cor é dada pelo ângulo formado pelas duas componentes. A esta transformação dá-se o nome de H2SV, conforme Figura 2.11:

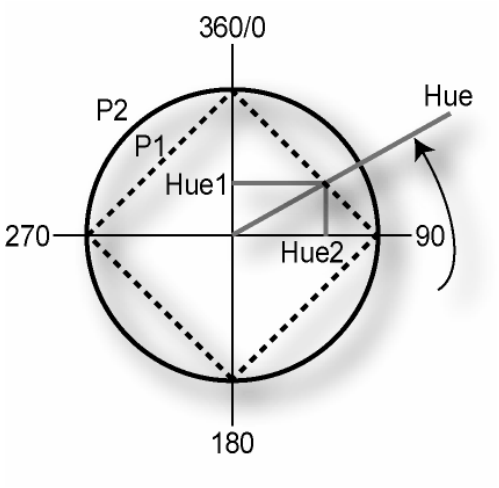

Figura 2.11: Representação esquemática da transformação de HSV para H2SV (Fonte: http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space)

No entanto, neste protótipo não foi utilizada esta transformação. Para solucionar a quebra de intervalo contínuo da cor vermelha, foi utilizada a representação dos dois intervalos que a cor pode assumir. Foi utilizado um intervalo longo de precisão que as cores podem assumir, permitindo uma grande margem de tolerância e evitando variações de hardware dos monitores.

A transformação de RGB para HSV ocorre da seguinte maneira:

De posse de uma cor definida no espaço de cores RGB com valores entre 0,0 e 1,0 onde os valores 0,0 e 1,0 representam, respectivamente, o valor mínimo e o valor máximo que pode ser assumido pelas componentes R, G e B. A transformação para o espaço de cores HSV ocorre conforme fórmula abaixo, onde MAX e MIN representam os valores máximo e mínimo respectivamente dos valores RGB

$$H = \begin{cases} 60 \times \frac{G-B}{MAX-MIN} + 0, & \text{if } MAX = R \\ & \text{and } G \geq B \\ 60 \times \frac{G-B}{MAX-MIN} + 360, & \text{if } MAX = R \\ & \text{and } G < B \\ 60 \times \frac{B-R}{MAX-MIN} + 120, & \text{if } MAX = G \\ 60 \times \frac{R-G}{MAX-MIN} + 240, & \text{if } MAX = B \end{cases}$$

$$S = \frac{MAX - MIN}{MAX} \qquad (2.3)$$

$$V = MAX$$

O resultado de H representa a cor variando de 0° a 360°, indicando o ângulo representado pela cor, a saturação S e o brilho V podem assumir valores entre 0 à 1.

Maiores informações sobre o sistema de cores HSV podem ser obtidas através dos sites: http://en.wikipedia.org/wiki/HSV_color_space e http://ilab.usc.edu/wiki/index.php/HSV_And_H2SV_Color_Space

2.3. API DirectX

Para facilitar o desenvolvimento de softwares são criados conjuntos de rotinas e padrões estabelecidos de forma a facilitar a utilização de suas funcionalidades por outros programas e aplicativos, sem exigir um conhecimento aprofundado sobre a implementação do software. Somente usam os seus serviços. Esses padrões são chamados de APIs (Application Programming Interface ou Interface de Programação de Aplicativos).

Microsoft DirectX é uma coleção de APIs que tratam de tarefas relacionadas a programação de jogos para o sistema operacional Microsoft Windows, ou seja, é quem padroniza a comunicação entre software e hardware. O DirectX foi inicialmente

distribuído pelos criadores de jogos junto com seus produtos, e, devido a sua grande e constante utilização, foi incluído no Windows.

Esta API possui funcionalidades de alto nível de forma a permitir o acesso ao hardware sem demandar um conhecimento aprofundado das particularidades de cada hardware. Isso só é possível devido a uma camada de abstração que transforma comandos genéricos, validos para todos os hardwares, em comandos específicos para acesso a cada dispositivo de vídeo específico. Desta forma, o DirectX possibilita a utilização dos recursos de hardware de forma a aumentar o desempenho das aplicações multimídias.

A API do DirectX possui os seguintes módulos:

- DirectDraw - acesso direto à placa de vídeo para gráficos 2D;
- Direct3D - acesso direto à memória de vídeo para gráficos 3D;
- DirectSound - acesso direto à placa de som;
- DirectPlay - acesso direto à rede;
- Direct Input - acesso direto à joysticks.

2.4. Técnica de Double Buffering

Ao se executar um programa, que exige uma alteração da tela a ser mostrada, a maneira normal de executar esta transmissão é apagar a tela inicial e ir desenhando na própria tela a nova imagem. No entanto, em casos onde se exige um alto desempenho de vídeo, podemos perceber uma “tremulação” da imagem. Este efeito é conhecido como efeito “flickering”. Os monitores de computadores redesenham constantemente a tela visível (em geral aproximadamente 60 Hz) e quando se deseja mostrar uma nova tela, ela deve ser desenhada ao mesmo tempo que o monitor atualiza a sua tela. Quando se utiliza a memória de vídeo diretamente, ocorre uma falta de sincronia entre o que está sendo desenhado e o que será desenhado no momento seguinte. É exatamente o mesmo efeito que acontece quando tentamos filmar uma televisão em uma câmera de vídeo. Ao ver o filme, percebemos uma tremulação na imagem da televisão filmada. É a falta de sincronia entre as imagens exibidas e as que estão sendo desenhadas no filme. E este é o

motivo principal para este protótipo utilizar monitores LCD, evitando assim problemas de sincronia entre o monitor emissor e as imagens capturadas pela WebCam.

A técnica de Double Buffering (ou Buffer ping-pong) foi criada para solucionar o problema de flickering. Esta técnica consiste em utilizar um buffer onde se desenha a próxima tela que será mostrada antes de enviá-la para o dispositivo de vídeo do computador. Com isto, a tela a ser desenhada é enviada “de uma só vez”.

Tem como principal vantagem a sua facilidade de implementação e serve como ponto de entrada para técnicas mais apuradas utilizadas na programação de jogos.

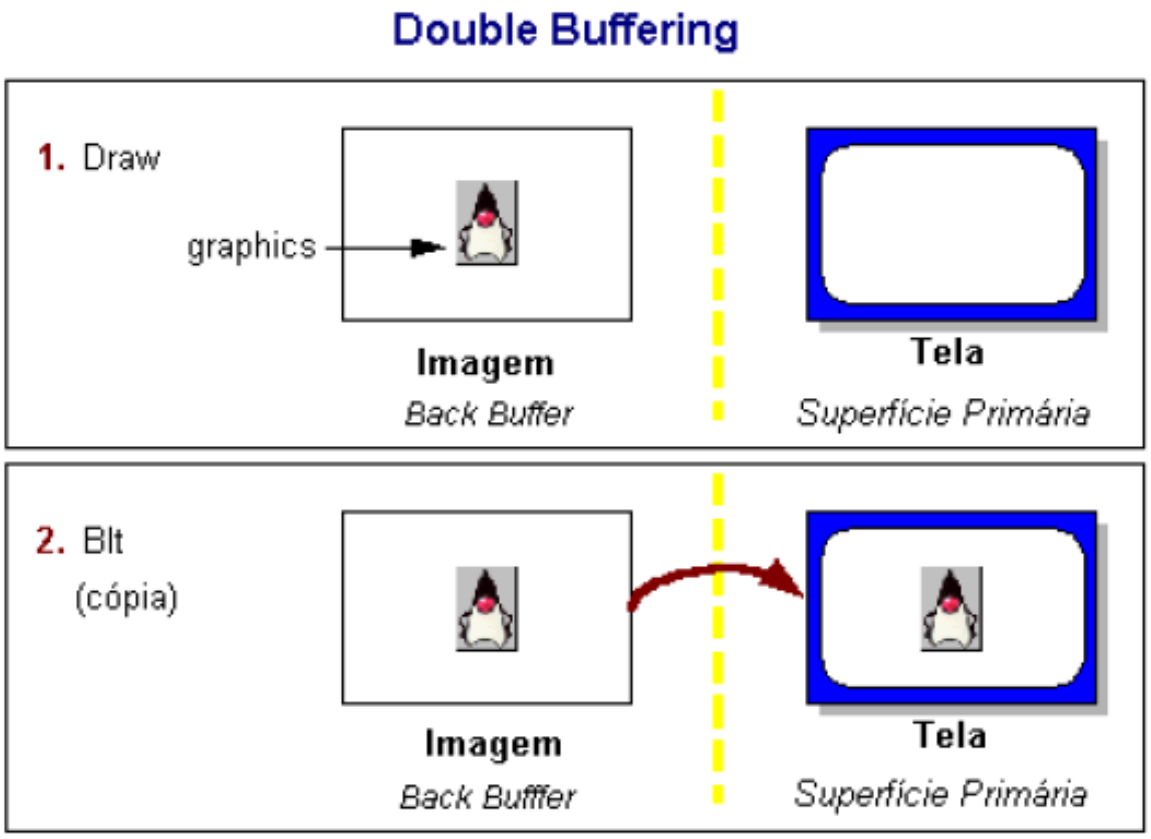

Figura 2.12: Desenho esquemático da técnica de Double Buffering (Fonte: http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html)

A superfície da tela é normalmente chamada de superfície primária e a superfície utilizada para desenhar a imagem que será mostrada na tela em seqüência à tela mostrada no momento é chamada de back buffer. De forma resumida, como representado na Figura 2.4, a técnica de Double Buffering permite a manipulação de

uma imagem no back buffer enquanto a superfície primária está sendo exibida [Barbosa, João Paulo].

Quando efetuamos a cópia da imagem desenhada no back buffer para a superfície primária, realizamos o que chamamos de blitting, ou simplesmente blit (representa a abreviação de "Block Image Transfer" na linguagem dos desenvolvedores de jogos).

2.5. Técnica de page-flipping

Uma outra forma de dinamizar a exibição de telas é através da técnica de page-flipping. Esta técnica pode ser utilizada em conjunto com a técnica de double-buffering. O page-flipping consiste basicamente em utilizar as funções do hardware da placa de vídeo, ao invés de mostrar na superfície primária o que foi desenhado no back buffer. Esta técnica trabalha com um conceito de ponteiro, onde a então superfície de back buffer torna-se a superfície primária e vice-versa, como pode ser melhor compreendido na Figura 2.13.

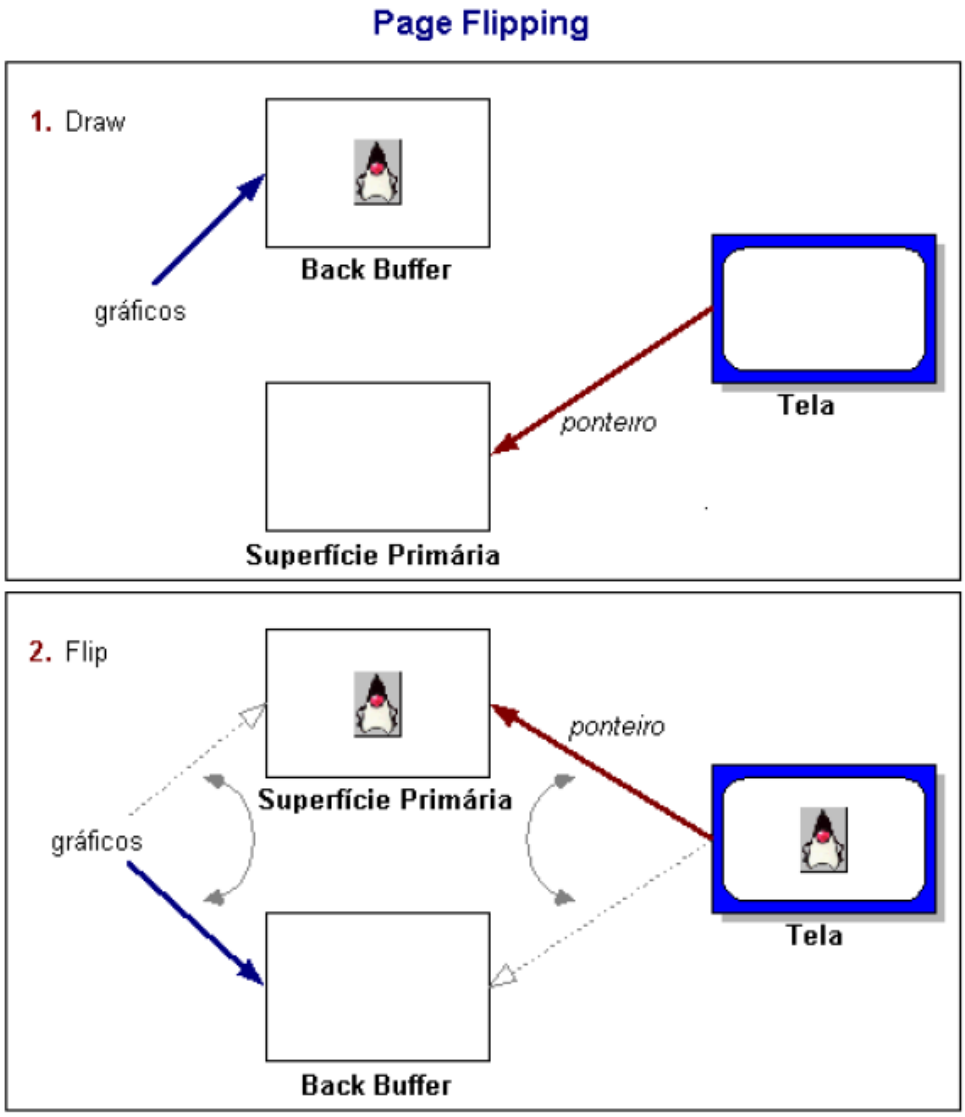

Figura 2.13: Desenho esquemático da técnica de Page Fliping (Fonte: http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html)

Neste protótipo, a técnica de page-flipping é utilizada, dando continuidade às técnicas utilizadas nas monografias anteriores.

2.6. Caracteres ASCII

Toda a transmissão envolvida neste protótipo é baseada no padrão de caracteres ASCII (American Standard Code for Information Interchange) . Este código é utilizado pelo computador para representar números, letras, pontuações e outros caracteres. É o

mais utilizado pela indústria pode ser formado por 7, 8 bits (adiciona-se um 0 na frente do padrão de 7 bits) ou ainda em algumas extensões por mais bits.

Como convenção este protótipo utiliza a representação de 8 bits, facilitando assim os padrões de transmissão. Em cada frame é transmitido um byte, portanto, um caractere.

3. TRANSMISSOR

3.1. Monitor de vídeo LCD

A utilização dos monitores LCD tem apresentado um crescimento acentuado, sendo utilizado em calculadoras, palmtops, aparelhos de telefonia móvel, monitores de notebooks e de computadores de mesa, entre tantos outros aparelhos. Este fato deve-se, principalmente, ao baixo consumo de energia, a baixa emissão de radiação e a luminosidade agradável, segundo o Clube do Hardware (site: www.clubedohardware.com.br).

A tecnologia LCD permite também que sejam exibidas imagens sem que haja a necessidade de utilização de um tubo de imagem, como ocorre com os monitores CRT (Catodic Ray Tube - Tubo de raios catódicos).

Segundo o artigo publicado no site http://www.infowester.com, redigido por Emerson Alecrim, como indica o nome, LCD é composto por um material denominado cristal líquido. Uma fina camada deste material é disposta entre duas camadas de vidro. Essas camadas de vidro possuem pequenos sulcos, isolados entre si, com um eletrodo ligado a um transistor. Cada um destes sulcos representa um dos pontos da imagem. Estas duas lâminas de vidro são dispostas entre duas camadas de um elemento polarizador. As moléculas de Cristal Líquido têm a capacidade de orientar a luz (de forma semelhante a um prisma). A luz emitente é composta, geralmente, por lâmpadas fluorescentes (usadas por gerarem pouco calor) ou Leds.

Quando os elementos elétricos existentes no monitor geram campos magnéticos, induzem os cristais a se posicionarem de modo a permitir a passagem somente das cores (ou luz) desejadas, ou para que bloqueiem a transmissão de luz.

Em telas monocromáticas, cada ponto da tela corresponde a um dos pontos da imagem e os cristais só podem assumir dois estados: um que permite a passagem da luz, de modo que fiquem transparente e outro que não possibilita a passagem da luz (opaco).

Para telas coloridas, diferentes níveis de tensão são aplicados às moléculas de cristal líquido, de forma a trabalhar sobre a luz branca emitida. Cada pixel da imagem é formado por um grupo de 3 pontos, um verde, um vermelho e outro azul. Como nos monitores CRT as cores são obtidas através de diferentes combinações de tonalidades dos três pontos.

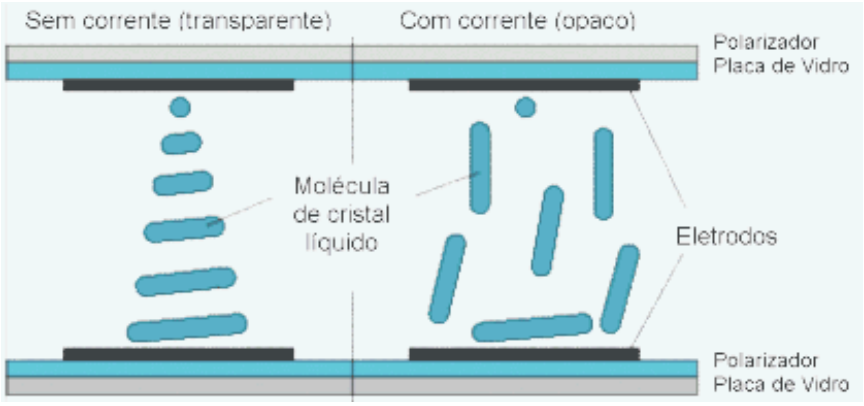

Figura 3.1: Representação esquemática sobre o posicionamento das moléculas de cristal líquido (Fonte: www.clubedohardware.com.br)

Em relação à resolução, os monitores LCD trabalham com taxas satisfatórias, mas há uma ressalva: é recomendável que o monitor trabalhe com a resolução que recebe de fábrica. Isso porque a exibição da imagem será prejudicada, caso uma taxa diferente seja usada. Por exemplo, pode acontecer de o monitor deixar uma borda preta em torno da imagem em resoluções menores que o padrão ou, ainda, o aparelho pode "esticar" a imagem, causando estranheza a quem vê. Além disso, tentar trabalhar com resoluções maiores é praticamente impossível.

Atualmente existem 2 principais tecnologias de LCD: os de matriz ativa e de matriz passiva. Os de matriz passiva possuem um ângulo de visão mais restrito e um tempo maior de delay. Por exemplo, um monitor comum de matriz passiva CRT (monitor tracidional) demora cerca de 15 a 20 milissegundos, enquanto um monitor LCD de matriz passiva, necessita cerca de 150 a 200 milissegundos. Por este motivo, em alguns notebooks existe a dificuldade de se enxergar o cursor do mouse. A própria imagem gerada por esse tipo de monitor LCD é de pior qualidade devido ao baixo nível

de contraste. No entanto, essa tecnologia não está mais sendo aplicada, e é encontrada apenas em aparelhos antigos.

Os monitores LCD de matriz ativa apresentam qualidade superior quanto à velocidade de atualização de imagem, se tornando próxima à velocidade de um monitor CRT, apresentando taxas variando de 40 a 50 milissegundos, (para se ter uma idéia, este velocidade já é suficiente para assistir a um DVD, mas ainda deixa a desejar quando se pretende utilizar em jogos de ação, onde exigem uma velocidade maior de transição de imagem). Além da velocidade, os monitores de matriz ativa são mais finos e leves, além de permitir um ângulo de visão e contrastes maiores.

Outro ponto interessante dos monitores LCD é o fato de serem digitais. Como todas as placas de vídeo atuais enviam sinais analógicos para o monitor (como o monitor CRT operava de maneira analógica era necessária esta conversão) é utilizado um novo circuito que é responsável por converter os sinais analógicos novamente em sinais digitais. A mudança efetuada pelas placas de vídeo são totalmente desnecessárias e acabam gerando uma significativa degradação da qualidade da imagem. Aumenta o número de circuitos usados no monitor, acarretando logicamente o aumento de custos. Segundo informações do site Clube do Hardware, estima-se que possa ser reduzido em aproximadamente 100 dolares o custo dos monitores com o uso de placas de vídeo que emitam o sinal digital diretamente.

Segundo o artigo publicado no site http://www.infowester.com, redigido por Emerson Alecrim, além da classificação de matriz ativa e passiva, temos alguns tipos de tecnologias LCD:

- **TN (Twisted Nematic):** encontrado nos monitores LCD de baixo custo. Nele, as moléculas de cristal líquido trabalham em ângulos de 90º. Monitores que usam TN podem ter a exibição da imagem prejudicada em animações muito rápidas;
- **STN (Super Twisted Nematic):** uma evolução do padrão TN, pode trabalhar com transição de imagens mais rápidas. Suas moléculas se movimentam mais rapidamente, possibilitando um ângulo de visão de aproximadamente 160º;
- **GH (Guest Host):** o GH é uma espécie de pigmento contido no cristal líquido que absorve luz. Esse processo ocorre de acordo com o nível do campo elétrico aplicado. Com isso, é possível trabalhar com várias cores;

- **Monitores TFT (Thin Film Transistor):** Essa tecnologia tem como principal característica a aplicação de transistores em cada pixel. Assim, cada unidade pode receber uma tensão diferente, permitindo, entre outras vantagens, a utilização de resoluções altas. Por outro lado, sua fabricação é tão complexa que não é raro encontrar monitores novos que contém pixels que não funcionam (os chamados "dead pixels"). Essa tecnologia é muito utilizada com cristal líquido, sendo comum o nome TFT-LCD para diferenciar esse equipamentos;
- **Telas de plasma:** A principal diferença deste tipo de tecnologia, é que cada pixel cria sua própria fonte de luz. A imagem da tela de plasma é muito nítida e não possui problemas de distorção nas extremidades da tela. Para gerar a luz em cada pixel, são usados eletrodos carregados entre painéis de cristal, que originam pequenas explosões de gás xenônio, que, por sua vez, reagem com luz ultravioleta, fazendo o fósforo de cada pixel brilhar (vermelho, verde ou azul).

Para o desenvolvimento do escopo deste trabalho independe o tipo de monitor LCD, já que a captura utilizada pela webcam não é afetada tão facilmente pela diferença de luminosidade, ao contrário dos fotodiodos que são extremamente sensíveis a qualquer tipo de interferência luminosa externa. Além disso, foi utilizada uma margem de segurança que diminui consideravelmente problemas com luminosidades externas e diferenças entre tecnologias de monitores LCD.

3.2. Software transmissor

O Software transmissor foi desenvolvido na linguagem de programação VB .Net. O projeto do protótipo transmissor é composto por três arquivos: um ativa o formulário mostrando as informações; um segundo atua como programa principal e finalmente o terceiro executa atividades auxiliares como definir a resolução de pixels de uma tela, captar a freqüência do monitor, entre outras.

3.2.1. A Plataforma .NET

Segundo a Wikipedia, a “Microsoft .NET é uma iniciativa da Microsoft em que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para .NET, pode ser executado em qualquer dispositivo ou plataforma que possua um framework: a "Plataforma .NET" (.NET Framework).”

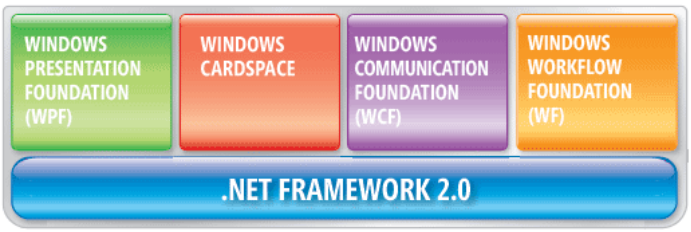

Figura 3.2: Representação esquemática da .NET FRAMEWORK (Fonte: hppt://msdn2.microsoft.com)

“...A plataforma .NET se baseia em um dos princípios utilizados na tecnologia Java (compiladores JIT). Os programas desenvolvidos para ela são duplo-compilados, ou seja são compilados duas vezes, uma na distribuição e outra na execução.

Um programa é escrito em qualquer das mais de vinte linguagens de programação disponível para a plataforma. O código fonte gerado pelo programador é então compilado pela linguagem escolhida, gerando um código intermediário em uma linguagem chamada MSIL (Microsoft Intermediate Language).”

3.2.2. Pontos chaves do código

O Fluxograma do projeto é representado pela figura 3.1.

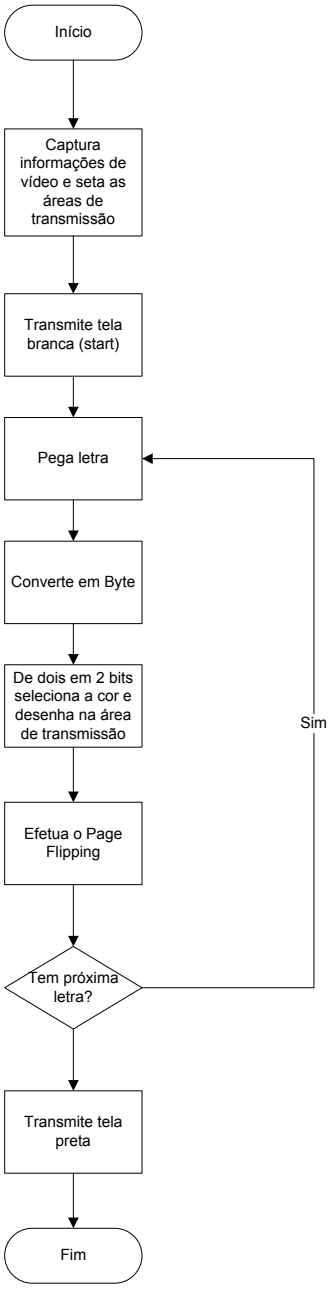

Figura 3.3: Fluxograma do código desenvolvido

O arquivo frmTransmissão.vb é responsável pela criação da janela que será responsável por mostrar as configurações de transmissão, como freqüência do monitor, o atraso desejado, além da freqüência de emissão. É neste momento também que é informado o tamanho da tela e a resolução e é inserida a mensagem a ser transmitida

O arquivo Transmissor.vb é o cerne do software. Na classe Transmissor, são executados os procedimentos de inicialização das superfícies primárias e secundárias, respectivamente:

```
Private g_pDDSFront As DirectDraw.Surface
Private g_pDDSBack As DirectDraw.Surface
```

A definição das cores a serem utilizadas para representar os pares de bits além das cores que representarão o início e o fim da transmissão são executadas por:

```
Private cor00 As Drawing.Color = Color.Blue
Private cor01 As Drawing.Color = Color.Yellow
Private cor10 As Drawing.Color = Color.Green
Private cor11 As Drawing.Color = Color.DarkRed
Private preto As Drawing.Color = Color.Black
Private branco As Drawing.Color = Color.White
```

É neste ponto também que são declarados os quadrantes ou zonas de transmissão. A tela do monitor foi dividida em quatro partes iguais e criadas da mesma forma que o quadrante superior direito abaixo:

```
q11 = New Rectangle(0, 0, resolucao.X / 2, resolucao.Y / 2)
```

Na função Transmitir, chama-se o objeto IniciarDirectDraw(), que é responsável por iniciar o DirectX.

Primeiramente inicia-se o dispositivo DirectDraw:

```
device = New DirectDraw.Device(DirectDraw.CreateFlags.Default)
```

É configurada a utilização do modo de tela cheia exclusiva, ou seja, ele suspende a execução de qualquer outro programa que esteja efetuando acesso à placa de vídeo, e fornece de modo exclusivo a utilização da placa de vídeo, garantindo, dessa forma, que não haverá interrupções da transmissão. Para isto, utiliza-se a linha de código:

```
device.SetCooperativeLevel(Me.janela,DirectDraw.CooperativeLevelFlags.
FullscreenExclusive)
```

Configura-se a área de transmissão:

```
device.SetDisplayMode(Me.resolucao.X,Me.resolucao.Y,Me.bitsPorPi
xel, 0, False)
```

É nessa mesma função IniciarDirectDraw() que são inicializadas as superfícies primárias e secundárias além de sincronizar o software com a freqüência vertical do monitor.

Logo após a execução do IniciarDirectDraw(), é mostrado uma seqüência contínua de telas brancas, que marcam o início da transmissão, pintando de branco todas as zonas de transmissão através da função DesenharRetangulo:

```
DesenharRetangulo(g_pDDSBack, q11, branco)
```

Esta função chama o método `superficie.Draw,` pertecente à API do DirectX

Após o desenho da tela principal, é executado o comando abaixo, que é responsável por realizar o flip entre a superfície primária e a superfície secundária:

```
g_pDDSFront.Flip(Nothing, DirectDraw.FlipFlags.Wait)
```

Esse é o momento de transmitir a mensagem. Essa transmissão é realizada letra a letra através do código abaixo:

```
 For Each letra As Char In texto
    For i As Integer = 1 To device.MonitorFrequency/frequencia
         DesenharByte(letra)
    Next
 Next
```

Para a transmissão de cada letra neste loop, é utilizada a função DesenharByte, que é responsável por converter cada letra em uma matriz de bits para que seja possível executar a transmissão. Essa conversão é realizada através do código abaixo:

```
Dim col() As Byte = {Convert.ToByte(letra)}
Dim matrizBits As New BitArray(col)
```

De posse da matriz de bits, é realizado o seguinte código:

```
desenharRetangulo(g_pDDSBack, q11, SelecionarCor(matrizBits(7), matrizBits(6)))
desenharRetangulo(g_pDDSBack, q12, SelecionarCor(matrizBits(5), matrizBits(4)))
desenharRetangulo(g_pDDSBack, q21, SelecionarCor(matrizBits(3), matrizBits(2)))
desenharRetangulo(g_pDDSBack, q22, SelecionarCor(matrizBits(1), matrizBits(0)))
```

Este código é responsável por desenhar cada uma das quatro zonas de transmissão utilizando para isso a função DesenharRetangulo. Ela utiliza as funções Draw do DirectX para desenhar na superfície. Na chamada da função DesenharRetangulo é iniciada a função SelecionarCor, que é reponsável por atribuir cada cor ao par de bits a ser transmitido. Para isto, ele varre a matriz de bits, buscando de dois em dois bits e verificando se o par é 00, 01, 10 ou 11 com isto, determina a cor correspondente a cada par.

Após a finalização das chamadas do DesenharRetangulo, é realizada a troca de supercície da primária para a secundária, realizando o page flipping.

Com a finalização da varredura de toda a matriz de bits a ser transmitida, o software desenha a tela totalmente preta, indicando o fim da transmissão.

3.2.3. Freqüência de transmissão

A transmissão a ser utilizada depende do modelo da câmera utilizada e da freqüência de atulização do monitor. Neste protótipo, foi utilizada uma WebCam com capacidade de aquisição de 30 frames por segundo, e o monitor utilizando uma freqüência de 60 Hz.

Portanto, para a correta captura de todos os frames emitidos, e utilizando a capacidade máxima da câmera, deve-se utilizar um atraso de 2, de modo que a

freqüência de transmissão seja a metade da freqüência de atualização suportada pelo monitor em questão.

Quando se deseja transmitir uma mensagem extensa, recomenda-se reduzir a freqüência de captura da câmera de 30 para 15 frames por segundo e utilizar um quarto da freqüência do monitor para transmissão. Este atraso é recomendado, devido a diferença entre a velocidade nominal da câmera e a velocidade real, no caso, a câmera utilizada apresenta uma freqüência nominal de 30 frames por segundo, mas durante a sua utilização, percebemos que sua captura gira em torno de 28 a 26 frames por segundo. Esta diferença pode ser atribuída à velocidade do software receptor, condições do ambiente e computadores utilizados.

4. RECEPTOR

4.1. MATLAB (MATrix LABoratory)

O MATLAB é um software voltado para cálculos numéricos, podendo ser utilizado de diversas maneiras. Mais comumente é utilizado para as análises numéricas, cálculos diversos envolvendo matrizes, sinais e gráficos. O principal motivo para sua larga utilização é a facilidade de utilização, provendo ambiente interativo e de fácil manipulação para problemas e expressões matemáticas. É adequado aos que desejam implementar e testar soluções com facilidade e precisão, sem a necessidade de conhecimentos específicos de uma linguagem de programação mais complexa.

O elemento básico de manipulação é a matriz numérica retangular, podendo conter elementos complexos. Vale lembrar que um número escalar é uma matriz de dimensão 1 x 1 e que um vetor é uma matriz que possui somente uma linha ou uma coluna. Esta forma de atuação permite a resolução de diversos problemas numéricos rapidamente, e comparativamente mais rápido que o tempo de programação para resolução dos mesmos problemas em linguagens estruturadas. Seus comandos podem ser escritos de maneira muito semelhante ao como escrevemos as expressões algébricas.

Atualmente o MATLAB dispõe de uma vasta coleção de bibliotecas denominadas toolboxes, bastante abrangentes, de funções matemáticas, geração de

gráficos e manipulação de dados que auxiliam o trabalho do programador. O Software permite ainda ao usuário escrever sua biblioteca personalizada, podendo enriquecer assim a linguagem e incorporar novas funções necessárias ao seu objetivo final. Uma destas bibliotecas é justamente a *Image Acquirement* (Aquisição de Imagens), que dispõe de formas de aquisição de vídeos e imagens.

A maneira mais simples de se fazer um programa em MATLAB é criar um arquivo texto com a lista de comandos desejados. Para isto, basta salvá-lo com a extensão ".m". Os comandos são os mesmos utilizados no Command Window, utilizando a mesma sintaxe. Um programa escrito assim é chamado script e toda vez que for executado, efetua os comandos como se fossem entrados sequencialmente via teclado. Este recurso facilita a criação de programas ou funções que serão executadas diversas vezes.

4.1.1. Toolbox de aquisição de imagens

A toolbox de Aquisição de Imagens é uma coleção de funções que expande a capacidade do MATLAB. Ela provê uma gama de possibilidades de operações incluindo a aquisição de imagens através de vários tipos de operações, incluindo a aquisição de imagens através de muitos tipos de dispositivos, desde câmeras profissionais de vídeo até Webcams baseadas em tecnologia USB (dispositivo este escolhido para o desenvolvimento deste projeto). Permite a configuração de funções que são acionadas mediante a ocorrência de algum evento externo, de forma a trazer os dados da imagem para serem trabalhados no ambiente do MATLAB.

Também é possível estender a utilização das toolboxes escrevendo scripts, usando uma combinação de toolboxes como a toolbox de processamento de imagens e transformação de dados utilizadas neste protótipo.

4.2. Webcam

Webcam é uma câmera de vídeo de baixo custo que capta imagens em tempo real, transferindo-as de modo quase instantâneo para a área de trabalho do computador

ou para uma página de Internet. É muito utilizada em videoconferências. Geralmente possui baixa qualidade de imagem e em alguns modelos apresenta microfone para captura dos sons do ambiente.

Adicionalmente ao uso em videoconferências, foi vislumbrado o desejo dos "internautas" em ver imagens de câmeras de outras pessoas pelo mundo. A denominação Webcam surgiu em referência à aplicação da tecnologia geralmente utilizada: "web" refere-se à Internet, e podemos encontrar outras denominações descrevendo o que pode ser visualizado com a câmera, como nestcam ou streetcam.

Hoje existem milhares de câmeras podendo ser encontradas em casas, escritórios, e outras construções, promovendo imagens panorâmicas das cidades, provendo segurança em locais de grande circulação. São utilizadas para monitorar tráfego, tempo, e até mesmo as atividades vulcânicas." Tradução livre do site wikipedia.com

4.2.1. Histórico

Em 1991, surgiu a primeira Webcam no Departamento de Ciências da Universidade de Cambidge nos Estados Unidos. A webcam, como é conhecida hoje, foi finalmente concluída em agosto de 2001.

Como muitas das novas tecnologias, rapidamente surgiu uma utilidade comercial e com ela um avanço tecnológico substancial. Impulsionado inicialmente pela indústria pornográfica, que requeria a disponibilização de imagens ao vivo, rapidamente desenvolveu plugins para a utilização em navegadores da Internet.

Após este desenvolvimento impulsionado pela indústria pornográfica, e com a sua disponibilização a baixo custo para usuários gerais, rapidamente surgiram novas aplicações como videoconferências, utilização de vídeos em programas de bate-papos pela Internet, até programas de seguranças que transmitem as imagens captadas em tempo-real, entre outras.

Sua última aplicação foi nos vídeo-games Play Station 2 e no Xbox 360 que utilizam o "Eye Toy", que permite ao jogador interagir com o jogo através de movimentos e detecção de cores.

4.2.2. Tecnologia

Webcams são compostas basicamente por uma lente e um sensor de imagem. Geralmente as câmeras possuem lentes de plástico móveis que servem para ajustes no foco da câmera. Os sensores podem ser basicamente CMOS (Complementary metal–oxide–semiconductor) ou CCD (charge-coupled device). De acordo com a tecnologia utilizada, não são necessariamente dispositivos de baixo custo. Geralmente trabalham com imagens em VGA e captação em torno de 25 quadros (“frames”) por segundo.

Possuem dispositivos eletrônicos que lêem as imagens captadas e as transmitem aos computadores interligados. Algumas câmeras (como as dos aparelhos de telefones celulares) usam sensores CMOS com suportes eletrônicos construídos em um mesmo chip, de forma a diminuir espaço e reduzir custos de fabricação.

4.2.2.1. Sensores CMOS

Os sensores CMOS ou complementary metal-oxide-semiconductor, têm sua principal vantagem centrada no baixíssimo consumo de energia, embora não sejam capazes de operar tão velozmente quanto circuitos integrados de outras tecnologias. Por causa disso, são largamente utilizados em calculadoras, relógios digitais, e outros dispositivos alimentados por pequenas baterias.

4.2.2.2. CCD

O sensor CCD - charge-coupled device, ou dispositivo de carga acoplado, ou ainda dispositivo de captura de cor, é um sensor utilizado para gravação de imagens. É composto por um circuito integrado, contendo uma série de capacitores. Sob o controle de um circuito externo, cada capacitor pode transferir sua carga elétrica para um outro capacitor vizinho.

Este sensor surgiu em 1969 criado por Willard Boyle e por George E. Smith em laboratórios de AT&T Bell. O laboratório estava trabalhando em um vídeo-fone e em dispositivos de memórias não voláteis (Bubble memory). Unindo as duas pesquisas em andamento, os pesquisadores desenvolveram o que chamaram de “Charge Bubble

Devices”. A essência deste invento era a capacidade de transferir a carga ao longo da superfície de um semicondutor, objetivando o armazenamento de dados. No entanto, logo as pesquisas evoluíram e demonstraram que o dispositivo poderia armazenar cargas através da entrada de foto-sensores e imagens digitais poderiam ser criadas. A partir deste momento ocorreu uma corrida para o desenvolvimento desta tecnologia.

Uma câmera CCD funciona da seguinte maneira: “Uma câmara forma a imagem através de uma lente convergente, isto é, uma lente que direciona os raios de luz em direção aos outros. Estes raios se encontram em uma superfície, chamada focal, onde está o chip (circuito integrado). Cada parte da superfície focal recebe a luz de uma parte da imagem. Neste chip, um CCD, cada fóton contém uma quantidade de energia suficiente para deslocar um elétron para um canal estreito no semicondutor. O CCD tem colunas destes canais foto-sensíveis, de modo que o padrão da luz que atinge o chip forma um padrão de cargas nestes canais. Para obter a imagem de vídeo, a câmara usa técnicas eletrônicas para transferir as cargas entre as colunas, e, finalmente, a câmara lê a carga elétrica ponto a ponto, coluna a coluna, até que o padrão de carga, representando o padrão de luz, seja completo. Uma câmara CCD tem um sinal de saída linear sobre uma faixa de intensidade muito maior do que a câmara de tubo, Tem menor distorção geométrica, e sua eficiência quântica pode chegar a 80%.” Texto retirado do site do Instituto de física da UFRGS - www.if.ufrgs.br. Informações adicionais também podem ser encontradas no site de Astronomia amadora da França no endereço: http://www.astrosurf.com/re/abc_camaras_ccd_pre.pdf

4.2.3. Modelo da WebCam utilizada no protótipo

Para o desenvolvimento deste protótipo, foi utilizada nos testes uma WebCam fabricada pela A4tech, modelo PK-636MA. Ela possui um sensor do tipo CMOS, com resolução de 640 por 480 (suportando a capacidade de 350 k pixels). A captura das imagens é feita de forma dinâmica e possui resolução de 1.3 mega pixels. A conexão da câmera com o computador se dá através da porta USB, que tem o seu detalhamento descrito a seguir.

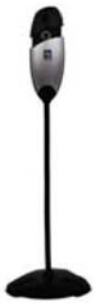

Figura 4.2.3-1: Imagem da WebCam utilizada no protótipo

Outra característica importante desta câmera é o seu ajuste de foco automático e o seu auto-ajuste de branco. Quanto ao foco ser ajustado automaticamente é uma grande vantagem para o protótipo. No entanto, o ajuste automático de branco é prejudicial, já que afeta a essência das cores a serem capturadas. Neste protótipo esta característica foi desabilitada nas configurações da câmera.

4.3. Porta USB

A porta USB (Universal Serial Bus – barramento serial universal) é um tipo de conexão plug-and-play que permite a conexão de diversos periféricos. Seu desenvolvimento iniciou-se em 1995 com uma parceria entre as empresas Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC e Philips.

Uma das primeiras versões, a 1.0, utilizava velocidade de 1.5 Mbps. Logo em seguida foi lançada a versão 1.1 que utilizava velocidades entre 1,5 a 12 Mbps. E finalmente no final de 2000 foi concebida a versão 2.0, além de manter a compatibilidade com as versões anteriores, foram implementadas melhorias que abrangem desde a topologia até a velocidade, podendo atingir 480 Mbps, equivalente a cerca de 60MBps.

A porta USB é um barramento que suporta a conexão de vários equipamentos na mesma porta, dividindo a largura de banda de transmissão. Outra característica importante deste barramento é permitir que equipamentos sejam conectados, configurados, desconfigurados e desconectados sem que o computador necessite ser desligado.

Um sistema USB pode ser descrito por três áreas de atuação:

- Interconexão
- Dispositivos
- Host

Com relação à interconexão, podemos dizer que é a forma como os dispositivos são conectados e se comunicam. Pode ser subdividido em topologia de barramento (modelo de comunicação entre USB e o Host), relação inter-camadas (as tarefas da USB executadas por cada camada do sistema), modelos de Fluxos de dados (como os dados se movimentam) e USB Schedule (forma como a USB opera nas interconexões compartilhadas, criada para suportar as conexões isossíncronas – que tem seu sincronismo determinado por intervalos regulares de tempo).

A porta USB é disposta seguindo a topologia de estrela para suportar as diversas conexões simultâneas, onde o ponto central é um hub conector. Esta topologia é conhecida também como *tiered-star* (estrela disposta em camadas/níveis). Na figura 4.3-1 é demonstrada a topologia utilizada pela USB.

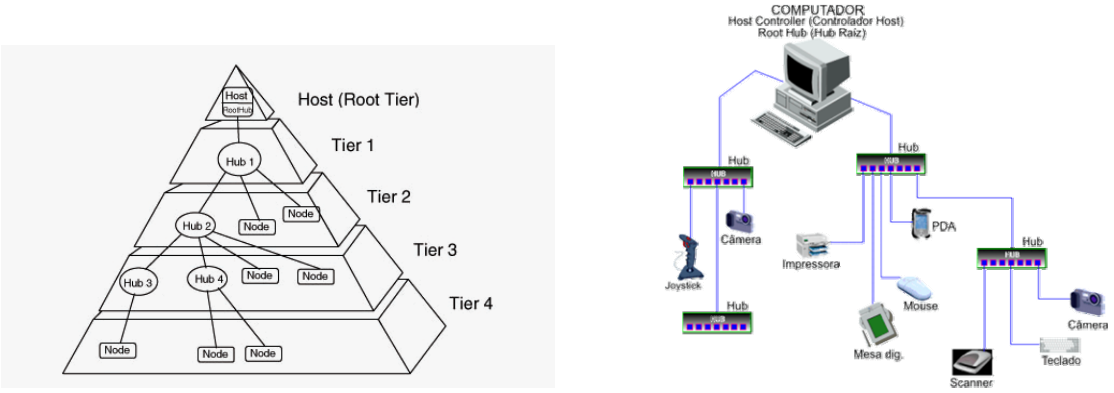

Figura 4.3-1: Topologia USB (Fonte: www.ufrj.br) e Exemplo prático de topologia (Fonte: www.rogercom.com)

Em um sistema USB existe um único host. Este host é responsável por detectar a inclusão e remoção de dispositivos, gerenciar o fluxo de controle de dados entre os dispositivos conectados, fornecer alimentação (tensão e corrente) aos dispositivos conectados e monitorar os sinais do bus USB.

A interface USB para o sistema refere-se ao controlador. Este controlador, pode ser implementado por uma combinação entre software e hardware, e, através de um

encadeamento de hubs, pode prover diversos pontos de conexão (limitado a 127 dispositivos em uma única porta).

Um dispositivo USB pode ter um Hub que provê pontos adicionais no sistema e pode ter funções que aumentam a capacidade do sistema.

Os dispositivos apresentam um padrão de interface baseado na compreensão do protocolo, nas respostas padrões de operação (como configuração e reinício), além do padrão de capacidade das informações descritivas.

4.3.1. Características Elétricas

As transferências de sinais elétricos são realizadas através de dois condutores com quatro fios, dois deles para a transmissão de dados e outros dois para a alimentação elétrica do dispositivo conectado, conforme figuras 4.3.1 e 4.3.1:

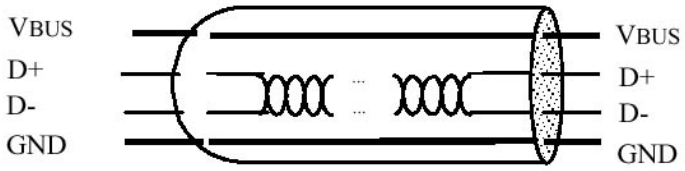

Figura 4.3: Representação esquemática de um cabo de transmissão elétrica da USB (Fonte: www.ufrj.br)

Figura 4.4: Imagem de um cabo USB (Fonte: www.rogercom.com)

O cabo Vbus (5V) é o fio positivo de fornecimento de energia. O GND (0V) é o pino negativo de energia do Bus. A porta USB pode fornecer no máximo 5 Volts de tensão e 500mA de corrente elétrica, para cada porta do Root Hub do host.

Os outros dois fios D+(dado+) e D-(dado-) são usados para transferência de dados entre o Host, hub e dispositivos. Todos os sinais de dados trafegam apenas por esses dois fios usando a codificação NRZI (No Return to Zero Inverted). Ou seja, o bit 1 é codificado através de uma transição da maior voltagem para a menor, ou também o inverso, da menor para a maior. Já o bit 0 é codificado sem haver transição. Durante o intervalo de um bit, a voltagem é constante.

Para a realização da comunicação, existem duas taxas de transferências que podem ser utilizadas, uma a 12 Mbps e outra, mais lenta, a 1,5 Mbps. Para transmissões realizadas em taxas de transferências mais baixas, exigem menores cuidados com relação a proteções eletromagnéticas. No entanto, a porta suporta ambas as taxas em um mesmo dispositivo devido ao controle automático de mudanças entre as transferências.

4.3.2. Protocolo USB

O USB é um barramento receptor, que tem todas as suas transferências de dados iniciadas pelo controlador do host. As transações efetuadas pelo barramento envolvem até três pacotes.

Cada transação tem início quando o controlador do host envia um pacote USB chamado "pacote de sinal" (token packet) descrevendo o tipo e a direção da transação, o endereço do dispositivo e o número do ponto final (endpoint). O número de endpoint é um valor de 4 bits entre 0H e FH, inclusive, associado a um ponto final de um periférico USB. O dispositivo USB é capaz de verificar que o pacote é para si, através da decodificação do endereço apropriado.

Em uma transação, dados são transferidos do host para o dispositivo ou vice-versa. A direção da transferência é especificada neste pacote inicial. O emissor envia um pacote de dados ou indica que não há mais dados para serem transferidos. O destinatário, em geral, responde com um "pacote aperto de mão" (handshake packet), indicando o sucesso da transferência.

O modelo para transferências de dados entre a fonte ou um destino no host e um ponto final de um dispositivo é conhecido como pipe (tubo ou canal). Existem dois tipos de pipe: correntes (stream) e mensagens. A diferença entre os dois consiste basicamente na falta de uma estrutura USB definida nas correntes, enquanto as mensagens possuem tal estrutura.

Os tubos possuem associações com as bandas de transmissão de dados, tipos de serviços de transferência e características de endpoint, como direção e tamanho de buffers. A maioria dos tubos passam a existir quando um dispositivo é configurado e, sempre que este dispositivo estiver conectado, uma mensagem é enviada para que possa fornecer acesso às configurações, informações de controle e status do dispositivo.

A listagem das transações permite o controle do fluxo para várias correntes de tubos (stream pipes) o que permite a construção de listas flexíveis. Múltiplas correntes de pipes podem ser "servidas" em diferentes intervalos e com pacotes de diferentes tamanhos.

4.3.3. Conexão de dispositivos USB

Todos os dispositivos são conectados à USB através de portas ou hubs. Quando um dispositivo se conecta através de um hub, os indicadores de estados existentes nos hubs alertam para a conexão ou remoção de um dispositivo em suas portas. No caso de uma conexão, o host ativa a porta e endereça o periférico através do control pipe indicando o endereço padrão (default).

O host atribui um único endereço USB para o dispositivo e, após isto, determina se a nova ligação é um hub ou uma função. Este processo é estabilizado utilizando o endereço USB atribuído e o número para o ponto final zero (zero endpoint).

Uma função é um dispositivo que é capaz de transmitir ou receber dados ou informações de controle do barramento, estando conectado, por um cabo, a uma porta ou hub. Além disso, um pacote físico pode conter funções múltiplas e possuir hosts embutidos. Estes são conhecidos como dispositivos compostos e aparecem para o host como um hub com um ou mais dispositivos não-removíveis.

Cada função contém informações de configurações que descrevem suas capacitações e recursos exigidos. Antes de poder utilizar uma função, o host deve configurá-la. Esta configuração inclui a alocação da banda de transmissão e a seleção das opções específicas para a configuração da função. Exemplos de uma função incluem: dispositivos de entrada, como um teclado; de saída como uma impressora; de telefonia; entre diversos outros.

Se uma função for conectada, então notificações de conexão serão manipuladas pelo software do host apropriado para esta função.

4.3.4. Tipos de Fluxo de Dados

O barramento suporta dados funcionais e substituição de controles entre o host e periféricos, assim como pipes uni ou bidirecionais. As transferências de dados realizam-se entre o software do host e um endpoint particular em um dispositivo USB. Geralmente, o movimento de dados através de um tubo é independente do fluxo de dados em outro tubo.

A arquitetura USB compreende quatro tipos básicos de transferências de dados:

- Transferência de Controle: usada para configurar um dispositivo no instante de sua conexão e pode ser usada para outros propósitos específicos, incluindo controle de outros pipes no dispositivo.
- Transferência de Volume de Dados: gerada e consumida em grandes quantidades e simultaneamente. Possui uma ampla e dinâmica latitude em transmissões de reserva.
- Transferências Interruptas de Dados: usada para caracteres ou coordenadas com percepções humanas ou características de respostas regenerativas.
- Transferência Isossíncrona de Dados: ocupa uma quantidade pré-negociável da banda de transmissão do barramento, com a distribuição de pulsos. Chamada também de transferência de correntes em tempo real (streaming real-time transfers).

A WebCam utilizada neste protótipo faz uso da tecnologia USB para a conexão com o host. Utiliza basicamente as tecnologias de transferência Isossíncrona, devido à necessidade de sincronia da velocidade e sincronia com o host.

4.4. Capturando imagens com a webcam

Para a captura de dados foi utilizado o MATLAB versão 7.0 (R14) com a toolbox de aquisição de imagens.

O primeiro passo para iniciar a WebCam utilizando o MATLAB, foi criar uma variável denominada vid, que inicia a WebCam com o código abaixo, onde o parâmetro winvideo corresponde ao dispositivo de vídeo, no caso a WebCam. O número que vem em seguida corresponte à quantidade de frames que serão capturados por instância. No caso, capturaremos um frame por vez, já que será necessário tratar cada tela capturada:

```
vid = videoinput('winvideo',1)
```

Após este comando é necessário iniciar a pré-visualização da imagem, para que possa ser efetuado o correto posicionamento da WebCam de forma a visualizar todo o monitor emissor. Para isto, a linha de código correspondente é:

```
preview (vid)
```

Para contornar problemas de identificação inicial das cores branco e preta, que marcam respectivamente o início e o fim da transmissão. E como não é viável a transformação de todos os frames capturados para HSV durante a capura das telas, em função da velocidade de transmissão, é necessário capturar inicialmente as variáveis de início de fim de transmissão. Esta captura é realizada através dos comandos:

```
bmedia = mean(mean(mean(getsnapshot(vid))))-3  %captura uma tela
%branca transmitida e subtrae-se 3 para permitir uma margem de erro.

fmedia = mean(mean(mean(getsnapshot(vid))))+ 3 %captura uma tela
%preta transmitida e adiciona-se 3 para permitir uma margem de erro.
```

Após isto, estamos aptos a realmente iniciar a transmissão.

O script “geral.m” é responsável por realizar todo o tratamento necessário para a transmissão. Segue-se o detalhamento dos pontos chaves do código:

Para a captura das telas apresentadas, é necessária a utilização do comando abaixo, que é responsável por armazenar em uma variável denominada “imagem” a tela capturada pela webcam no momento da execução deste script:

```
imagem = getsnapshot(vid);
```

Após a obtenção da imagem, ela é armazenada em uma estrutura de array multidimensional, que são basicamente, uma extensão dos arrays tradicionais. Na figura 4.4-1 é representada de forma simplificada essa estrutura:

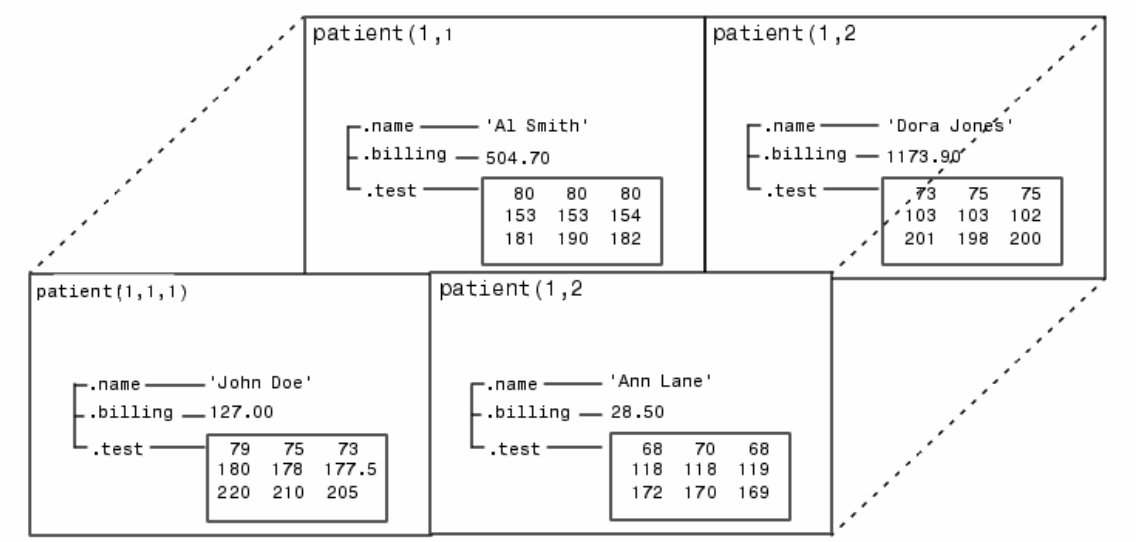

Figura 4.5: Estrutura de uma estrutura de array multidimensional (Fonte: Help do MATLAB)

Conforme exemplificado nesta imagem, temos vários pacientes e cada um destes pacientes tem as suas propriedades, como o seu nome, o valor pago e uma matriz teste. Para o acesso a cada uma destas propriedades, basta referenciar o número do paciente seguido de um ponto e a propriedade desejada. Por exemplo, deseja-se saber o nome do paciente(1,1,1) para isto, basta digitar o comando:

```
Paciente(1,1,1).name
```

Além de armazenar os dados de acordo com as propriedades, é possível realizar cálculos diretamente com as propriedades. Por exemplo, se desejarmos somar a quantia paga por todos os pacientes basta utilizar o código abaixo:

```
Total=sum([pacient.billing])
```

Para o armazenamento das imagens obtidas, a estrutura acima exemplificada é utilizada da seguinte maneira:

```
e(nframe).frame=[imagem];
```

“nframe” representa o número de ordem da imagem capturada. A propriedade “.frame” armazena a “imagem” capturada nas linhas anteriores.

Após este armazenamento, é calculada a média geral da imagem capturada através da codificação representada abaixo, onde é realizada a média de cada uma colunas das 3 dimensões no espaço de cor RGB. Posteriormente a média do resultado das médias das colunas, gera as médias das dimensões e finalmente a média total dos três espaços:

```
mediageral=mean(mean(mean(imagem)))
```

Com isto, é realizada a verificação se a média é correspondente à cor branca capturada antes da transmissão iniciar e serve para marcar o início da transmissão. Caso seja detectada uma transmissão cuja média seja inferior à tela branca capturada inicialmente, será iniciado o armazenamento das telas. Este armazenamento ocorre até que a média geral seja correspondente ou inferior à tela de transmissão totalmente preta também capturada antes da transmissão, fato este que marca o fim da transmissão.

Somente após o fim da captura, é iniciado o tratamento da imagem para a decodificação da transmissão. Optou-se por efetuar estes cálculos ao final, devido a velocidade de transmissão.

Para cada frame capturado, é executada inicialmente uma transformação do espaço de cores RGB para o espaço HSV, através do comando abaixo, HSV é a variável onde será armazenado o resultado da transformação realizada pelo comando “rgb2hsv” da imagem armazenada na estrutura; i representa o número da tela capturada; e a propriedade “.frame” representa o a imagem em si que foi armazenada:

```
hsv=rgb2hsv(e(i).frame)
```

A imagem HSV é dividida nas quatro zonas de captura através dos seguintes comandos, conforme pode ser observado na figura 4.6:

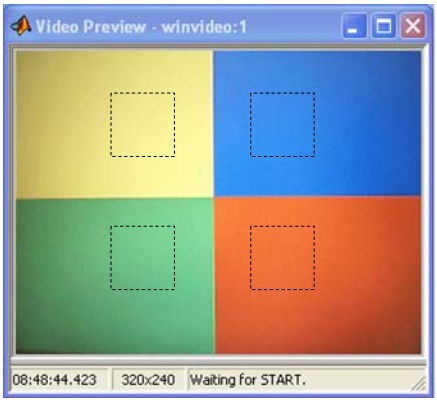

Figura 4.6 – Imagem capturada da WebCam e sua representação esquemática das zonas de transmissão.

```
pedaco1=hsv([4:100],[4:100],1)
pedaco2=hsv([4:100],[204:300],1)
pedaco3=hsv([20:230],[4:100],1)
pedaco4=hsv([110:230],[249:300],1)
```

Pedaco1, pedaco2, pedaco3 e pedaco4 são as zonas de transmissão; e hsv([a:b],[a:b],1) representa a imagem no espaço hsv; [a:b] representa o intervalo da imagem que será utilizado representando os eixos X e Y; e o 1 representa a primeira dimensão da imagem, a dimensão correspondente à componente matiz (cor) da imagem.

Para certificar-se que a imagem capturada foi de alta qualidade e que possibilite a verificação da cor é realizado um cálculo de desvio padrão, através do código:

```
std2 = std(std(double(pedaco1(:,:,1) )))
```

Este cálculo é efetuado através da função disponibilizada pelo MATLAB std. No entanto, um ponto importante para este calculo é a transformação da zona de transmissão para o tipo “double” realizada de acordo com a função: “`double(pedaco1(:,:,1))`” , caso contrário, a operação não poderá ser realizada. Caso o desvio padrão seja superior a 10, a imagem é considerada corrompida e é informado que a imagem foi capturada com muito ruído.

De posse da zona de transmissão válida, é calculado o valor médio da cor da região através do código a seguir:

```
cor1=mean(mean(pedaco1))*360
```

Acima, vemos o cálculo da média da matriz bidimensional pedaco1. O valor obtido é multiplicado por 360, pois durante a conversão do espaço RGB para HSV, a matiz da cor é representada por um valor entre 0 e 1. A multiplicação é realizada para termos o ângulo que representa a cor para a atribuição do par de bits transmitido.

Após a obtenção do valor médio da cor, é invocada a função que será responsável por atribuir a cada cor o seu respectivo par de bits:

```
bit1=atribuiBit(cor1)
```

Para a execução da função atribuiBit, é passado como parâmetro o valor médio da cor obtida e tem como resposta o par de bits correspondente ou um erro devido a valores vazios ou fora das margens estabelecidas para cada cor.

De posse dos valores dos bits, é necessário realizar o encadeamento dos bits de acordo com sua posição na tela, e o seu correto encadeamento. Para isto, foi utilizado o seguinte cálculo:

```
a1=bitshift(cor1,6)
b1=bitshift(cor2,4)
c1=bitshift(cor3,2)
d1=cor4
```

A função bitshift (x,y) efetua o deslocamento do valor x em y posições para a esquerda. Esse deslocamento corresponde a realizar uma multiplicação de 2^k. Para realizar o encadeamento dos bits deslocados, são utilizadas as seguintes linhas de código, onde bitor realiza uma operação de OR entre os bits:

```
e=bitor(a1,b1)
f=bitor(e,c1)
g=bitor(f,d1)
```

Finalmente, com o bit completo, a conversão do número obtido em uma letra é realizada da seguinte forma:

```
letra(i)=char(g)
```

A frase obtida, armazenada na variável letra, é finalmente a mensagem que foi transmitida.

5. TESTES REALIZADOS

Para a comprovação da viabilidade do protótipo, foram realizadas diversas transmissões de forma a se medir o desempenho e retidão da transmissão.

Os testes foram realizados em diversos ambientes, com diferentes luminosidades. Foi atribuída uma escala de luminosidade do ambiente, variando de 0 a 3, onde o 0 corresponde a um ambiente muito escuro, o 1 a um ambiente com baixa iluminação, 2 ambiente claro e 3 ambiente extremamente iluminado (ao ar livre).

A tabela completa apresentada no apêndice mostra os testes realizados, com o texto utilizado para transmissão, o que foi recebido e a quantidade de letras transmitidas erroneamente. Através dela, é possível estimar que cerca de 70% das transmissões efetuadas foram realizadas com sucesso.

É possível observar a transmissão mais efetiva em ambiente de luminosidade média, (ambientes 1 e 2). Além disto, podemos observar que quando se deseja transmitir frases longas a ocorrência de erros é muito maior proporcionalmente. Esta ocorrência pode ser atribuída a perda de sincronismo entre a transmissão do monitor e as imagens capturadas pela WebCam. Uma possível solução seria a inclusão de frames de controle em determinados blocos de bytes ou a utilização de uma câmera mais precisa.

Observamos também que, apesar da utilização do modelo HSV, o protótipo se mostrou sensível às condições do ambiente, ainda que, consideravelmente reduzida em relação ao projeto que o antecedeu.

Outra solução possível para mitigar a ocorrência dos erros é a implementação de técnicas de bi-paridade e correções de erros, que poderão reduzir sensivelmente a quantidade de bits interpretados de maneira errônea.

luminosidade	tentativas	acertos	%
0	20	13	65
1	21	19	90,47619
2	46	36	78,26087
3	60	35	58,33333
total	147	103	70,06803

6. CONSIDERAÇÕES FINAIS

6.1. Dificuldades encontradas

Como comentado anteriormente, o projeto inicialmente se propunha a realizar a transmissão através de uma simples melhoria/evolução dos protótipos anteriormente desenvolvidos. No entanto, ao montar o mesmo dispositivo desenvolvido anteriormente, surgiram várias dificuldades relativas à diferença de luminosidade de monitores CRT para monitores LCD. Esta dificuldade forçava o projeto a retornar ao estágio inicial com a utilização de somente dois níveis de transmissão.

Durante conversas com colegas, professores e pesquisas, surgiu a possibilidade de utilizar como receptor a WebCam. No entanto ainda restava a dúvida de como poderia ser feita a captura. Inicialmente, sugeriu-se o desenvolvimento de um software para realizar essa captura. Porém, trabalhar com os drivers das WebCams, a princípio, “soou” como algo muito distante com relação ao prazo para a conclusão do projeto. Com um pouco mais de pesquisa sobre o assunto, foi encontrada uma forma para a utilização do MATLAB. Esta forma funcionou corretamente e, a partir daí, surgiram as primeiras idéias palpáveis com relação à utilização da WebCam.

No desenvolvimento dos trabalhos, as principais dificuldades encontradas concentraram-se inicialmente nos estudos sobre os espaços de cores, já que a idéia era encontrar alguma forma de separar a matiz de cores da luminosidade e saturação. O primeiro espaço pesquisado neste sentido foi o YIQ, que foi descartado em função de misturar as componentes de matiz e saturação, representada pela combinação de dois eixos que representam a cromaticidade como um todo.

Posteriormente, a dificuldade encontrada foi relativa à sincronização entre a WebCam e o programa emissor, mas, para tanto, surgiu a opção de utilizar uma cor específica para marcar o início da transmissão e outra para a finalização da mesma.

6.2. Resultados obtidos

Os resultados esperados foram obtidos. No entanto, muitas melhorias ainda devem ser implementadas para que a transmissão possa ocorrer de maneira segura e constante. Durante os testes foram identificadas instabilidades, tanto da câmera quanto do MATLAB, que podem ser sanadas através da utilização de uma câmera com íris fixa, ou que tenha uma taxa de transmissão mais elevada, além da criação de um programa específico para captura e transformação dos dados. Recomenda-se a evolução para o uso de J2ME de forma a possibilitar a utilização em aparelhos celulares com câmera acoplada.

6.3. Conclusões

Neste protótipo foi utilizada programação em VB .NET e MATLAB e uma WebCam simples. Com estes instrumentos, foi possível realizar uma transmissão de dados, com cerca de 70% de transmissões bem sucedidas. Foi possível detectar que transmissões efetuadas em ambientes muito claros possuem um indicador muito aquém, se comparada em ambientes mais escuros (onde foi possível chegar a um índice de 85 a 90%).

Para chegar a este resultado, foi necessário uma pesquisa bastante extensa sobre diversos sistemas de cores, além do estudo da ToolBox de Aquisição e Processamento de imagens do MATLAB. Também exigiu estudos sobre programação na linguagem VB .NET e MATLAB, as quais não detinha nenhum conhecimento.

O projeto mostrou-se possível como comprovado pelo protótipo implementado, apesar das limitações da versão proposta neste projeto. Diversas melhorias podem ser implementadas, tornando a transmissão mais segura e precisa além de possibilitar a sua utilização em outras aplicações. Ainda assim, o trabalho atendeu as expectativas gerando resultados esperados.

6.4. Sugestões para trabalhos futuros

Como o trabalho seguiu uma linha um pouco diferente da seguida pelos projetos anteriores, e com o uso de novas tecnologias, existe uma gama enorme de possibilidades de evolução do projeto, que envolve desde a implementação da paridade de bits até o desenvolvimento utilizando J2ME para a utilização das câmeras existentes nos aparelhos telefônicos móveis.

Além dessas sugestões, existe a possibilidade de implementar uma transmissão estegnografada, com a utilização por exemplo de apresentações PowerPoint para transmitir alguma mensagem oculta.

Outra sugestão é aumentar a velocidade de transmissão, utilizando para isto um número maior de zonas de transmissão, e/ou utilizar uma câmera de maior qualidade para que a transmissão possa ocorrer de forma mais veloz.

7. REFERÊNCIA BIBLIOGRÁFICA

MITSUKA, Tiago Almeida. Transmissão Alternativa de Dados, Centro Universitário de Brasília, 2004.

BARBOSA, João Paulo. Transmissão Multinível e Detecção e Correção de Erros no Projeto de Transmissão Alternativa de Dados, Centro Universitário de Brasília, 2005

GONZALES, R.C. and WOODS, R.E., Digital Image Processing, Editora Pearson Prentice Hall, 1993.

GONZALES, Rafael C.; WOODS, Richards E.; EDDINS, Steven L. Digital Image Processing Using MATLAB, Editora Pearson Prentice Hall, 2004.

KOENIGKAN, Luciano Vieira, Método de análise do contorno de aglomerados de gotas de chuva artificial em imagem digital, 2005

CRUVINEL, P. E.; CESTANA, S; JORGE, L. A. C., Métodos e Aplicações do Processamento de Imagens Digitais, Embrapa, 1996

CHAPMAN, Stephen J. Programação em MATLAB para engenheiros, Thomson Learning 2003.

Sites da internet:

http://ilab.usc.edu [Último acesso em 14/06/2007]

http://www.liv.ic.unicamp.br/~bergo/mc102/slide-t21.pdf [Último acesso em 14/06/2007]

www.wikipedia.org [Último acesso em 22/05/2007]

www.clubedohardware.com.br [Último acesso em 18/05/2007]

www.mathworks.com [Último acesso em 22/05/2007]

http://www.xuti.net/index.php?option=com_content&task=view&id=51&Itemid=32 [Último acesso em 18/05/2007]

www.clubedohardware.com.br [Último acesso em 18/05/2007]

www.codigolivre.com.br [Último acesso em 18/05/2007]

http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html [Último acesso em 18/05/2007]

www.microsoft.com [Último acesso em 18/05/2007]

http://msdn.microsoft.com/directx [Último acesso em 18/05/2007]

http://www.rogercom.com/PortaUSB/MotorPasso.htm [Último acesso em 18/05/2007]

http://www.infowester.com [Último acesso em 14/06/2007]

http://ilab.usc.edu/wiki/index.php/Main_Page [Último acesso em 19/05/2007]

http://www.pads.ufrj.br/ [Último acesso em 19/05/2007]

www.usb.org [Último acesso em 19/05/2007]

http://www.rogercom.com [Último acesso em 19/05/2007]

www.hp.com [Último acesso em 20/05/2007]

http://www.a4tech.com/ [Último acesso em 20/05/2007]

http://msdnwiki.microsoft.com/ [Último acesso em 26/05/07]

http://www2.ufpa.br [Último acesso em 27/05/2007]

http://www.inf.pucrs.br [Último acesso em 13/06/2007]

http://www.dcmm.puc-rio.br/cursos/ipdi [Último acesso em 13/06/2007]

http://www.astrosurf.com/re/abc_camaras_ccd_pre.pdf [Último acesso em 14/06/2007]

APÊNDICES

Apêndice 1 – Tabela de testes

número	transmitido	recebido	letras erradas	luminosidade
1	ZReNaTa	ZReNAda	2	2
2	ZReNaTa	ZReNaTa	0	2
3	ZReNaTa	ZReNaTa	0	2
4	ZReNaTa	jRfNBda	4	2
5	ZReNaTa	ZReNaTa	0	2
6	ZReNaTa	ZReNaTa	0	2
7	ZReNaTa	ZReNaTa	0	1
8	ZReNaTa	ZZeNaTa	1	0
9	ZReNaTa	ZRenaTa	1	3
10	ZReNaTa	ZReNaTa	0	2
11	Gabriela Martins	Gabriela Martins	0	2
12	Gabriela Martins	Gabriela Martins	0	2
13	Gabriela Martins	Gabriela Martins	0	2
14	Gabriela Martins	Gabriela Martins	0	2
15	Gabriela Martins	Gabriela Martins	0	2
16	Gabriela Martins	Gabrin`b Abpxncs	9	2
17	Gabriela Martins	GAbrrela MArtyns	4	2
18	Gabriela Martins	Gabriela Martins	0	2
19	Gabriela Martins	Gabriela Martins	0	2
20	Gabriela Martins	GAbrela Martins	2	2
21	Ceub	Ceub	0	2
22	Ceub	Ceub	0	2
23	Ceub	CEub	1	2
24	Ceub	Ceub	0	2
25	Ceub	Ceub	0	2
26	Ceub	CEub	1	2
27	Ceub	Ceub	0	2
28	Ceub	CEub	1	2
29	Ceub	CEur	2	2
30	Ceub	Ceub	0	2
31	Engenharia da Computação	Engenharia d Computação	1	3
32	Engenharia da Computação	ENgenharia da Computação	1	3
33	Engenharia da Computação	Engenharia da Computação	0	3
34	Engenharia da Computação	Engenharia dCmtçã	5	3
35	Engenharia da Computação	Engenharia da Computação	0	3
36	Engenharia da Computação	ENgenharra ¤a COmputqçãï	6	3
37	Engenharia da Computação	Engenharia da Computação	0	3
38	Engenharia da Computação	ENgenharia da Computação	1	3
39	Engenharia da Computação	Engenharia da Computação	0	3
40	Engenharia da Computação	Engenharia da Computação	0	3
41	Centro Universitário de Brasília	Centro Universitário de Brasílaa	1	1
42	Centro Universitário de Brasília	Centro Universitárik de Brcsílaa	3	1
43	Centro Universitário de Brasília	Centro Universitário de Brasília	0	2
44	Curso de Engenharia da Computação	Curso de Engenharia da Computação	0	1
45	Curso de Engenharia da Computação	Curso de Engenharia da Computação	0	2
46	Minha orientadora se chama Marony	MMnda ¯qrintqdorr ³u £`ama MAr□ny	13	1
47	Minha orientadora se chama Marony	Minha orientadora se chama Marony	0	1

48	Minha orientadora se chama Marony	MInha orientadora se chama Marony	1	1
49	Minha orientadora se chama Marony	Minha orientadora se chama Marony	0	1
50	EnGeNhArIa	EnGeNhArIa	0	3
51	EnGeNhArIa	EnGeNhArIa	0	3
52	EnGeNhArIa	EnGeNhArIa	0	3
53	EnGeNhArIa	EnGeN`ArAa	3	3
54	EnGeNhArIa	EnGeNhArIa	0	3
55	EnGeNhArIa	EnGeNhArIa	0	3
56	EnGeNhArIa	ENGENHArAA	4	3
57	EnGeNhArIa	EnGeNhArIa	0	3
58	EnGeNhArIa	□NoGnLbBrAa	9	3
59	EnGeNhArIa	EnGeNhArIa	0	3
60	Luiz Eduardo	Luiz Eduardo	0	3
61	Luiz Eduardo	Luir Eduaplo	3	
62	Luiz Eduardo	Luiz Eduardo	0	3
63	Luiz Eduardo	Luiz Eduardo	0	3
64	Luiz Eduardo	Luiz Eduardo	0	3
65	Luiz Eduardo	Luiz Eduardo	0	3
66	Luiz Eduardo	Luyz0EDuqrto	4	3
67	Luiz Eduardo	Luiz Eduardo	0	3
68	Luiz Eduardo	Luiz Eduardo	0	3
69	Luiz Eduardo	Luiz Eduardo	0	3
70	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	2
71	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
72	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
73	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
74	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
75	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	Teste de transmissão do projeto final da Renata Molteas` Fjrribr`(b` Ccxra	14	3
76	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	Teste de transmissão do projeto final da Renata Monteiro Ferreira da Ccpta	2	3
77	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
78	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
79	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
80	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa	0	3
81	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
82	ABCDEFGHIJKLMNOPQRSTUVXZ	◆BCHFFGHIJKLMNO`QRSTUVXX	5	3
83	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
84	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
85	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
86	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
87	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
88	ABCDEFGHIJKLMNOPQRSTUVXZ	◆BCHFFGHJJKLNNO@QRSTUVXX	7	3
89	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
90	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	3
91	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3

92	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
93	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
94	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
95	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
96	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
97	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
98	abcdefghijklmnopqrstuvxz	abcdefghijjlmn`prsptvxxz	4	3
99	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
100	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	3
101	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
102	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
103	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
104	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
105	ABCDEFGHIJKLMNOPQRSTUVXZ	BCHFFGHJJKLNNO@RRSTUVXX	8	2
106	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNIPQRRTVXXZ	4	2
107	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
108	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
109	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
110	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	2
111	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
112	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
113	abcdefghijklmnopqrstuvxz	ab`dfghhjkllno`prsxvvxx	10	2
114	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
115	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
116	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
117	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
118	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
119	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxx	1	2
120	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	2
121	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
122	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
123	ABCDEFGHIJKLMNOPQRSTUVXZ	BC@DFGHHJKLNNO@RRSXVVXX_	11	1
124	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
125	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
126	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
127	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
128	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
129	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
130	ABCDEFGHIJKLMNOPQRSTUVXZ	ABCDEFGHIJKLMNOPQRSTUVXZ	0	1
131	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
132	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
133	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
134	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
135	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
136	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
137	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
138	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
139	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
140	abcdefghijklmnopqrstuvxz	abcdefghijklmnopqrstuvxz	0	1
141	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão dk proje|` jnbb` h` Znbhr`"Oolvibw`FRrvirr $a Ostq ±uu ´prtq da tpqns}issóï altUr^ativa de dados, utilizando um monitor LCD e uma WebCam		1

142	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de trajsmassã` l` psojd|`(jnb``(b` jnbdr`Ootuir□ □Frrvirr ¤a □Ostq quu trqtq da trQns]issão alternativa de dados, utilizando um monitor LCD e uma WebCam		1
143	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do rojeto inal da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	2	2
144	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do projetb ffbb` hb Rnbhr`"Oolvibs`(Fbrvibr`$b □Ostq ±uuprtq ¤a ´pqns}issóï altur^atYvQ de dados, utilizando um monitor LCD e uma WebCam		2
145	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão dg proje|` jnbb` `` Znbhr`"Oolvib{`□Fbrvirr $a □Ostq ±uu ´prtq ¤a tps}issóï altUr^ativa de dados, utilizando um monitor LCD e uma WebCam		2
146	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do projeto final da Renata Monteiri Ferriarb hb Ccpp` yu`(prhr`(b`$prnc}mssóï !ltur~btyzr ¤e ¤ados< utylizqndo u] monitor LCD e uma gebCam		
147	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uia WeaCam	1	2
148	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dadoq(uteljrbll` }b oonh|cp @H@*``*}a`&ebCAm		2
149	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	0	2
150	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da transmissão alternativa de dados, utilizando um monitor LCD e uma WebCam	teste de transmissão do projeto final da Renata Monteiro Ferreira da Costa que trata da trbasmcssã` `ltbrbhzjr`(f`(bdnc| &tzlfzrndo µ} onjt□r0□CD e u}a gebCAm		1

Apêndice 2 – Código do programa transmissor

O programa principal do transmissor do protótipo é executado pelo código à seguir:

```
Imports Microsoft.DirectX

Public Class Transmissor

    'Declara variáveis da classe
    Private janela As Form
    Private resolucao As Point
    Private bitsPorPixel As Integer
    Private frequencia As Integer

    'Declara dispositivo
    Private device As DirectDraw.Device

    'Declara superficies
    Private g_pDDSFront As DirectDraw.Surface 'superficie primária, a que está sendo mostrada
    Private g_pDDSBack As DirectDraw.Surface 'superfície secundária, próxima a ser mostrada

    'Declara quadrantes
    Private q11, q12, q21, q22 As Rectangle 'quadrantes de desenho para transmissao paralela

    'Declara cores
    Private cor00 As Drawing.Color = Color.Blue

    Private cor01 As Drawing.Color = Color.Yellow
    Private cor10 As Drawing.Color = Color.LimeGreen
    Private cor11 As Drawing.Color = Color.Red
    'cores para marco de início e fim da transmissao (sincronia com webcam)
    Private preto As Drawing.Color = Color.Black
    'fim da transmissao
```

```
        Private branco As Drawing.Color = Color.White
        'início da transmissao


        'Construtor da classe
        Public Sub New(ByVal _janela As Form, ByVal _resolucao As
Drawing.Point, ByVal _bitsPorPixel As Integer, ByVal _frequencia As
Integer)
            Me.janela = _janela
            Me.resolucao = _resolucao
            Me.bitsPorPixel = _bitsPorPixel
            Me.frequencia = _frequencia

            q11 = New Rectangle(0, 0, resolucao.X / 2, resolucao.Y /
2) 'quadrante superior esquedo
            q12 = New Rectangle(resolucao.X / 2, 0, resolucao.X / 2,
resolucao.Y / 2) 'quadrante superior direito
            q21 = New Rectangle(0, resolucao.Y / 2, resolucao.X / 2,
resolucao.Y / 2) 'quadrante inferior esquerdo
            q22 = New Rectangle(resolucao.X / 2, resolucao.Y / 2,
resolucao.X / 2, resolucao.Y / 2) 'quadrante inferior direito

        End Sub

        Public Sub Transmitir(ByVal texto As String)
            Try
                'Inicializa o DirectDraw
                IniciarDirectDraw()

                Dim y As Integer = 0
                For y = 0 To 120
                    'antes de comecar a transmitir, mostra tela
branca 5 vezes (start da webcam)
                    DesenharRetangulo(g_pDDSBack, q11, branco)
                    DesenharRetangulo(g_pDDSBack, q12, branco)
                    DesenharRetangulo(g_pDDSBack, q21, branco)
                    DesenharRetangulo(g_pDDSBack, q22, branco)

                    g_pDDSFront.Flip(Nothing,
DirectDraw.FlipFlags.Wait)
                Next
```

```
                'Envia cada letra
                For Each letra As Char In texto
                    For i As Integer = 1 To device.MonitorFrequency
/ frequencia
                        'como a webcam é mais lenta, é preciso
ajustar a frequencia para garantir a captura
                        DesenharByte(letra)
                    Next
                Next

            Catch ex As Exception  'tratamento de erros
                Throw ex
            Finally

                Dim z As Integer = 0
                For z = 0 To 200
                    'ao final, mostra tela preta para indicar o fim
da transmissao
                    'assim garanto a sincronia com a webcam
                    DesenharRetangulo(g_pDDSBack, q11, preto)
                    DesenharRetangulo(g_pDDSBack, q12, preto)
                    DesenharRetangulo(g_pDDSBack, q21, preto)
                    DesenharRetangulo(g_pDDSBack, q22, preto)

                    g_pDDSFront.Flip(Nothing,
DirectDraw.FlipFlags.Wait)
                Next

                'Libera o DirectDraw
                LiberarDirectDraw()
            End Try

        End Sub

        ''' <summary>
        '''   Inicializa   o   dispositivo,   superfícies   e   demais
componentes do DirectDraw
        ''' </summary>
        ''' <remarks></remarks>
        Private Sub IniciarDirectDraw()
```

```
            'Inicializa o dispositivo padrão
            device                          =                          New
DirectDraw.Device(DirectDraw.CreateFlags.Default)
            device.SetCooperativeLevel(Me.janela,
DirectDraw.CooperativeLevelFlags.FullscreenExclusive)
            device.SetDisplayMode(Me.resolucao.X,     Me.resolucao.Y,
Me.bitsPorPixel, 0, False)

            'Inicializa superfície primária
            Dim ddsd As New DirectDraw.SurfaceDescription()
            ddsd.BackBufferCount = 1
            ddsd.SurfaceCaps.PrimarySurface = True
            ddsd.SurfaceCaps.Flip = True
            ddsd.SurfaceCaps.Complex = True
            g_pDDSFront = New DirectDraw.Surface(ddsd, device)

            'Inicializa superficie secundária
            Dim ddscaps As New DirectDraw.SurfaceCaps()
            ddscaps.BackBuffer = True
            g_pDDSBack = g_pDDSFront.GetAttachedSurface(ddscaps)

            'Sincroniza a aplicação com a frequência do monitor

device.WaitForVerticalBlank(DirectDraw.WaitVbFlags.BlockEnd)

        End Sub

        ''' <summary>
        '''  Desenha um retângulo em uma determinada superfície,
preenchido com um determinada cor.
        ''' </summary>
        '''  <param name="superficie">Superfícia a ser desenhado o
retângulo.</param>
        '''  <param name="retangulo">Retângulo inicializado com sua
localização e dimensões.</param>
        '''    <param    name="cor">Cor    de    preenchimento    do
retângulo</param>
        ''' <remarks></remarks>
```

```
        Private Sub DesenharRetangulo(ByVal superficie As
DirectDraw.Surface, ByVal retangulo As Drawing.Rectangle, ByVal cor As
Drawing.Color)
            Dim blt As New DirectDraw.DrawEffects
            blt.FillColor = cor.ToArgb

            superficie.Draw(retangulo,       Nothing,       Nothing,
DirectDraw.DrawFlags.ColorFix, blt)
        End Sub

        ''' <summary>
        ''' Recebe uma letra (byte) e realiza a transmissão
        ''' </summary>
        '''     <param     name="letra">Letra     (byte)     a     ser
transmitido.</param>
        ''' <remarks></remarks>
        Private Sub DesenharByte(ByVal letra As Char)


            'Dim j As Integer = 0
            'j = Convert.ToSByte(letra)
            Dim col() As Byte = {Asc(letra)}
            Dim matrizBits As New BitArray(col)
            'chama desenhar retangulo parametros a superficie q vai
desenhar,
            'em q quadrante e qual a cor, para cor chamo o metodo
com a posicao da matriz)
            DesenharRetangulo(g_pDDSBack,                        q11,
SelecionarCor(matrizBits(7), matrizBits(6)))
            DesenharRetangulo(g_pDDSBack,                        q12,
SelecionarCor(matrizBits(5), matrizBits(4)))
            DesenharRetangulo(g_pDDSBack,                        q21,
SelecionarCor(matrizBits(3), matrizBits(2)))
            DesenharRetangulo(g_pDDSBack,                        q22,
SelecionarCor(matrizBits(1), matrizBits(0)))


            g_pDDSFront.Flip(Nothing,      DirectDraw.FlipFlags.Wait)
'troca a superficie primaria com a secundária

        End Sub
```

```
        ''' <summary>
        '''  Função auxiliar para selecionar a cor correta a ser
usada.
        ''' </summary>
        ''' <param name="bit1">Valor do primeiro bit.</param>
        ''' <param name="bit2">Valor de segundo bit.</param>
        ''' <returns></returns>
        ''' <remarks></remarks>
        Private Function SelecionarCor(ByVal bit1 As Boolean, ByVal
bit2 As Boolean) As Drawing.Color
            If bit1 Then
                If bit2 Then
                    Return cor11
                Else
                    Return cor10
                End If
            Else
                If bit2 Then
                    Return cor01
                Else
                    Return cor00
                End If
            End If
        End Function

        Private Sub LiberarDirectDraw()
            g_pDDSBack.Dispose()
            g_pDDSFront.Dispose()
            device.Dispose()

            Me.janela.WindowState = FormWindowState.Normal
        End Sub

    End Class
```

A tela do programa é executada pelo código:

```
Public Class frmTransmissao

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles MyBase.Load
        txbFrequencia.Text = Auxiliar.FrequenciaMonitor()
        txbWidth.Text = Auxiliar.Resolucao.X
        txbHeigth.Text = Auxiliar.Resolucao.Y
        txbBPP.Text = Auxiliar.BitsPorPixel
        CalculaFrequenciaFinal()
    End Sub

    Private Sub Form1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles MyBase.Click

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
        Dim transmissor As New Transmissor(Me, New
Point(txbWidth.Text, txbHeigth.Text), txbBPP.Text,
txbFrequenciaFinal.Text)
        transmissor.Transmitir(txbTexto.Text)
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button3.Click
        Me.Close()
    End Sub

    Private Sub nudAtrazo_ValueChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
nudAtrazo.ValueChanged
        If txbFrequencia.Text.Length > 0 Then
            CalculaFrequenciaFinal()
        End If
    End Sub

    Private Sub CalculaFrequenciaFinal()
```

```
            txbFrequenciaFinal.Text                                    =
Integer.Parse(txbFrequencia.Text) / Integer.Parse(nudAtrazo.Value)
        End Sub
    End Class
```

O Auxiliar.vb utiliza os seguintes códigos:

```
    Imports Microsoft.DirectX

    Public Class Auxiliar

        Private         Shared         device         As         New
DirectDraw.Device(DirectDraw.CreateFlags.Default)

        Public Shared Function FrequenciaMonitor() As Integer
            Return device.MonitorFrequency
        End Function

        Public Shared Function Resolucao() As Point
            Return        New        Point(device.DisplayMode.Width,
device.DisplayMode.Height)
        End Function

        Public Shared Function BitsPorPixel() As Integer
            Return          (device.DisplayMode.LinearSize          /
device.DisplayMode.Width) * 8
        End Function

    End Class
```

Apêndice 3 – Códigos dos scripts do receptor

O cerne do projeto concentra-se na função abaixo, denominada geral5.m:

```
function geral5=parameterfun(vid, bmedia, fmedia)
clear e* media l*;
letra='???';

nframe=1
imagem = getsnapshot(vid);
e(nframe).frame=[imagem]; %estrutura de armazenamento, utilizando um
array multidimensional
media(nframe).media=mean(mean(mean(imagem))); %verifica se a média da
cor transmitida esta fora
                                    %da área de alcance da aplicacao
ou
                                    %seja, se estiver transmitindo
preto
                                    %ou banco ele para


        while media(nframe).media >= bmedia
            imagem = getsnapshot(vid);
            e(nframe).frame=[imagem];
            media(nframe).media=mean(mean(mean(imagem)));
        end

        while media(nframe).media > fmedia %valor do preto
            nframe=nframe+1;
            imagem = getsnapshot(vid);
            e(nframe).frame=[imagem];
            media(nframe).media=mean(mean(mean(imagem)));
            if media(nframe).media < 35 | media(nframe).media > 220
                display ('FIM DA TRANSMISSÃO')
                break
            end
        end %end do while



for i=1:nframe-1    %este loop trata cada frame adquirido exceto o
último, já que
                    %será a captura do ultimo frame ou de um frame
invalido

    hsv=rgb2hsv(e(i).frame);            %transforma em hsv

    %%%%%%%%%%%%%%%%pedaço 1 %%%%%%%%%%%%%%
    pedaco1=hsv([23:64],[40:103],1);      %pega um pedaço da imagem
    std1 = std(std(double(pedaco1)));     %calcula o desvio padrao do
desvio
                                        %padrao da transformacao da
matriz de
                                        %uint8 em double, nao é
possível fazer
```

```
                                        %o calculo com uint8 como o
original
    if std1>0.5
        display ('imagem com muito ruido -- std1')
        break
    end
    cor1=mean(mean(pedaco1))*360;     %Calcula o valor médio da cor no
pedaço
    bit1=atribuiBit(cor1);            %Chama a funçao para atribuir o
valor
                                     %de acordo com o tom de cinza
captado


    %%%%%%%%%%%%%%%%pedaço 2 %%%%%%%%%%%%%%
    pedaco2=hsv([18:63],[235:285],1);     %pega um pedaço da imagem
    std2 = std(std(double(pedaco2)));     %calcula o desvio padrao do
desvio
                                        %padrao da transformacao da
matriz de
                                        %uint8 em double, nao é
possível fazer
                                        %o calculo com uint8 como o
original
    if std2>0.5
        display ('imagem com muito ruido -- std2')
        break
    end
    cor2=mean(mean(pedaco2))*360;     %Calcula o valor médio da cor no
pedaço
    bit2=atribuiBit(cor2);             %Chama a funçao para atribuir o
valor



    %%%%%%%%%%%%%%%%pedaço 3 %%%%%%%%%%%%%%
    pedaco3=hsv([158:202],[43:108],1);     %pega um pedaço da imagem
    std3 = std(std(double(pedaco3)));     %calcula o desvio padrao do
desvio
                                        %padrao da transformacao da
matriz de
                                        %uint8 em double, nao é
possível fazer
                                        %o calculo com uint8 como o
original
    if std3>0.5
        display ('imagem com muito ruido -- std3')
        break
    end
    cor3=mean(mean(pedaco3))*360;     %Calcula o valor médio da cor no
pedaço
    bit3=atribuiBit(cor3)    ;         %Chama a funçao para atribuir o
valor



    %%%%%%%%%%%%%%%%pedaço 4 %%%%%%%%%%%%%%
    pedaco4=hsv([162:196],[235:273],1); %pega um segundo pedaço da
imagem
    std4 = std(std(double(pedaco4(:,:,1) )));
    if std4>0.5
```

```
        display ('imagem com muito ruido -- std4')
        break
    end
    cor4=mean(mean(pedaco4(:,:,1)))*360;
    bit4=atribuiBit(cor4);             %Chama a funçao para atribuir o
valor



     media(i).valor=[cor1, cor2, cor3, cor4];
     media(i).cor=[bit1, bit2, bit3, bit4];
     media(i).desvio = [std1, std2, std3, std4];
     %%%%%%%%%%%%%%desloca os bits %%%%%%%%%%%%

    bit1=uint8(bit1);
    bit2=uint8(bit2);
    bit3=uint8(bit3);
    bit4=uint8(bit4);

    a1=bitshift(bit1,6);
    b1=bitshift(bit2,4);
    c1=bitshift(bit3,2);
    d1=bit4;

    k=bitor(a1,b1);
    f=bitor(k,c1);
    g=bitor(f,d1);


    %%%%%%%%%%%%%%%%%%atribui o valor para char %%%%%%%%%%

    media(i).letra=char(g);
    media(i).bit=g;
    letra(i)=char(g);
    display(i);
end
display('fim');
geral5=letra

end
```

A função atribuibit() é executada pelos seguintes códigos:

```
function atrbit1 = parameterfun(valor)

if valor <= 0
          atrbit1 = 'vazio'                  %caso tenha valor nulo
          display ('valor nulo')

elseif valor > 0 & valor <= 40 | valor > 349 & valor <= 360
           atrbit1 = 3;                       %para cor vermelha
atribui par de bits --> 11

elseif valor > 40 & valor <= 60
           atrbit1=1;                         %para cor amarela
atribui par de bits --> 01
```

```
elseif valor > 60 & valor <= 180
            atrbit1=2;                          %para cor verde
atribui par de bits --> 10

elseif valor > 180 & valor <= 349
            atrbit1=0;                          %para cor azul atribui
par de bits -->00
else
                atrbit1='fora'
                display ('valor fora das magens estabelecidas')


end
```

A seguir, o código gerado através do GUIDE do MATLAB, que representa a interface gráfica do receptor:

```
function varargout = tela(varargin)
% TESTETELA1 M-file for testetela1.fig
%      TESTETELA1, by itself, creates a new TESTETELA1 or raises the
existing
%      singleton*.
%
%      H = TESTETELA1 returns the handle to a new TESTETELA1 or the
handle to
%      the existing singleton*.
%
%      TESTETELA1('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in TESTETELA1.M with the given input
arguments.
%
%      TESTETELA1('Property','Value',...) creates a new TESTETELA1 or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before testetela1_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to testetela1_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help testetela1

% Last Modified by GUIDE v2.5 03-Jun-2007 14:55:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn', @testetela1_OpeningFcn, ...
                   'gui_OutputFcn',  @testetela1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin(1));
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before testetela1 is made visible.
function testetela1_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to testetela1 (see VARARGIN)

% Choose default command line output for testetela1
handles.output = hObject;
%handles = guihandles(vid)
handles.video = videoinput('winvideo', 1);
preview (handles.video)

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes testetela1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = testetela1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;




% --- Executes on button press in pushbutton2.
function varargout = pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%varargout=PVW_toolbox2();
%varargin{1}=vid;
```

```
% --- Executes on button press in CapturarBranco.
function CapturarBranco_Callback(hObject, eventdata, handles)
% hObject    handle to CapturarBranco (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
bmedia=media(handles.video)-3;
%bmedia=num2str(bbmedia);
set(handles.branco,'String',bmedia);


function branco_Callback(hObject, eventdata, handles)
% hObject    handle to branco (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of branco as text
%        str2double(get(hObject,'String')) returns contents of branco
as a double


% --- Executes during object creation, after setting all properties.
function branco_CreateFcn(hObject, eventdata, handles)
% hObject    handle to branco (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end



function Preto_Callback(hObject, eventdata, handles)
% hObject    handle to Preto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Preto as text
%        str2double(get(hObject,'String')) returns contents of Preto
as a double


% --- Executes during object creation, after setting all properties.
function Preto_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Preto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end


% --- Executes on button press in CapturarPreto.
function CapturarPreto_Callback(hObject, eventdata, handles)
% hObject    handle to CapturarPreto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fmedia=media(handles.video)+3;
%fmedia=num2str(ffmedia);
set(handles.preto,'String',fmedia);

% --- Executes on button press in transmite.
function transmite_Callback(hObject, eventdata, handles)
% hObject    handle to transmite (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
branco=get(handles.branco,'String')
branco=str2num(branco)
display(branco)

preto=get(handles.preto,'String')
preto=str2num(preto)
display(preto)

lfrase=geral5(handles.video,branco,preto);
set(handles.lfrase,'String',lfrase);

function lfrase_Callback(hObject, eventdata, handles)
% hObject    handle to lfrase (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lfrase as text
%        str2double(get(hObject,'String')) returns contents of lfrase
as a double


% --- Executes during object creation, after setting all properties.
function lfrase_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lfrase (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end
```

```
function preto_Callback(hObject, eventdata, handles)
% hObject    handle to preto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of preto as text
%        str2double(get(hObject,'String')) returns contents of preto
as a double


% --- Executes during object creation, after setting all properties.
function preto_CreateFcn(hObject, eventdata, handles)
% hObject    handle to preto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor')
);
end
```

Na tela é chamada uma função que calcula o valor médio do frame capturado, para confirmar as condições de ambiente:

```
function media = parameterfun(vid)
media = mean(mean(mean(getsnapshot(vid))))

end
```