Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Configuração de um Sensor de Imagens CMOS com Compressão de Imagens no Plano Focal para Operação em Modo de Vídeo

Autor:

Leandro d'Oliveira do Rêgo

Orientador:

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph.D.

Co-orientadora:

Fernanda Duarte Vilela Reis de Oliveira, Eng.

Examinador:

Prof. Antonio Petraglia, Ph.D.

Examinador:

Prof. Heraldo Luís Silveira de Almeida, D.Sc.

DEL

Abril de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica – Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro – RJ CEP 21949-900

DEDICATÓRIA

Este projeto é dedicado a todas as pessoas que contribuíram para que eu conseguisse me formar.

AGRADECIMENTOS

Primeiramente agradeço a Deus por estar sempre comigo, principalmente nos momentos mais difíceis, lembrando-me que não há mal que dure para sempre, e que tudo o que acontece tem sua razão de ser.

Muito obrigado aos meus pais, Marco e Isabel, e irmão, Thiago, por representarem meu "porto seguro", além de me ensinarem quão importante e poderoso é o amor de uma família. Muito obrigado por terem sempre cuidado de mim.

Obrigado também às minhas avós, Fany e Ruth, que contribuíram bastante tanto financeiramente, quanto emocionalmente com seu carinho e amor.

Agradeço meu avô Josmar, *in memoriam*, que sempre orou por mim e acreditou que eu seria uma grande pessoa, alguém de sucesso.

Muito obrigado aos meus orientadores do Projeto de Graduação, Fernanda D. V. R. de Oliveira e Prof. José Gabriel R. C. Gomes. Vocês me guiaram corretamente na execução desse trabalho, que não foi fácil. Muito obrigado por terem aguçado meu pensamento crítico e por massificarem na minha mente uma regra óbvia, mas com a qual só consegui me acostumar agora, "começar pelo mais simples". Levarei isso para a minha vida!

Agradeço imensamente aos maravilhosos colegas e amigos da faculdade: Gabriel Ab-Abib, Danilo Nóbrega, Felipe Clark, Felipe Luiz, Fernanda de Oliveira, Hugo Cuffa,Igor Aguiar, Isabela Apolinário, Kauli Gutierrez, Lívia de Almeida, Luiz Tavares, Nzagi Terra, Pedro Guimarães, Peterson Nogueira, Raphael Fernandes, Renato Sauer, Renato Tavares, Rodolpho Barbosa, Rodrigo Emanuel, Simão Coutinho, Zheng Ming. Vocês me ajudaram muito nos estudos, sanando minhas dúvidas e me auxiliando na preparação para as provas. Também agradeço pelas conversas descontraídas e "papos-cabeça" que tivemos nas salas de aula ou nos bancos do CT.

Obrigado ao colega de curso Felipe Ribeiro, por suas excelentes dicas e orientações no projeto, as quais me ajudaram a superar grandes dificuldades nessa empreitada.

Muito obrigado a minha namorada Suellen pela sua paciência e amor. Diante de minhas dificuldades com a faculdade, me retribuía com seu olhar de ternura e abraço carinhoso. Você sempre me consolou e revigorou minhas energias.

RESUMO

O projeto em questão visa o aumento da taxa de captura de quadros de uma câmera desenvolvida pelo PADS, Laboratório de Processamento Analógico e Digital de Sinais (UFRJ/COPPE/PEE e UFRJ/EPoli/DEL), com tecnologia CMOS (*complementary metal-oxide silicon*).

Para alcançar esse objetivo modificações foram realizadas no sistema da câmera, mas somente na parte de software, especificamente no bloco de processamento de imagens que é executado em computador, mantendo-se inalterado o hardware do equipamento. As modificações consistem na implementação em C/C++ do decodificador, que antes era implementado no MATLAB, e alteração na rotina de comunicação entre interface do usuário e decodificador. Essas alterações tornaram o decodificador muito mais rápido. E não há mais falhas de execução.

A câmera mencionada foi elaborada no Projeto de Graduação de Fernanda Duarte Vilela Reis de Oliveira, concluído em janeiro de 2012, e na Dissertação de Mestrado de Hugo de Lemos Haas, concluída em fevereiro de 2012. O aparelho apresentou avanços em comparação com os equipamentos convencionais, traduzidos por simplificação no hardware e aumento de velocidade do algoritmo de compressão das imagens capturadas.

Palavras-chave: câmera CMOS, processamento de imagens, decodificador, Armadillo, OpenCV.

ABSTRACT

This project aims at increasing the frame capture rate of a CMOS (complementary metal-oxide silicon) camera that was designed and implemented by PADS, the Analog and Digital Signal Processing Laboratory at the Electrical Engineering Program of COPPE/UFRJ and at the Electronics and Computer Engineering Department of EPoli/UFRJ.

To reach that goal, changes were applied to the software part of the camera system, particularly with respect to the image processing block that runs in a computer. The hardware part was left unchanged. The modifications consist of the implementation in C / C + + decoder, which was previously implemented in MATLAB, and change in routine communication between the user interface and decoder. These changes made the decoder much faster and eliminated previously observed execution errors.

This camera system was developed in the final undergraduate project of Ms. Fernanda D. V. R. Oliveira, which was concluded in January 2012, and in the M.S. thesis of Mr. Hugo L. Haas, which was concluded in February 2012. The device led to advances with respect to conventional equipment, in terms of hardware simplification and image compression algorithm speed-up.

Keywords: CMOS camera, image processing, decoder, Armadillo, OpenCV.

SIGLAS

CCD - Charge Coupled Device

CI - Circuito Integrado

CMOS - Complementary Metal Oxide Semiconductor

COPPE - Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

DEL - Departamento de Eletrônica

DPCM - Differential Pulse Code Modulation

EPOLI - Escola Politécnica

MATLAB - Matrix Laboratory

OPENCV - Open Source Computer Vision

PADS - Laboratório de Processamento Analógico e Digital de Sinais

PC - Personal Computer

PEE - Programa de Engenharia Elétrica

PIC - Peripheral Interface Controller

UFRJ - Universidade Federal do Rio de Janeiro

USB - Universal Serial Bus

VQ - Vector Quantization

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

Abordaremos, neste projeto, os sensores de imagem CMOS [1-6] *(complementary metal-oxide semiconductor)*. Um sensor CMOS pode ser compreendido como uma matriz bidimensional de foto-sensores, que convertem a luz em cargas elétricas. Atualmente existem duas tecnologias possíveis para aquisição de imagens: CMOS e CCD [2] (*charge coupled device*). Elas diferem basicamente na maneira de quantificar o total de energia armazenada em cada célula fotossensível da matriz. O uso da tecnologia CMOS apresenta algumas vantagens em relação ao CCD: menor custo, baixo consumo de energia e reuso de mesmo hardware para outras aplicações, como processamento de sinais.

1.2 – Delimitação

O presente projeto tem como origem uma câmera digital CMOS, projetada e testada no Laboratório de Processamento Analógico e Digital de Sinais (PADS) do Centro de Tecnologia da UFRJ. O projeto do circuito é detalhado no Projeto de Graduação de Fernanda Duarte Vilela Reis de Oliveira [1] e na Dissertação de Mestrado de Hugo de Lemos Haas [6]. A câmera projetada realiza a captura e a compressão de uma imagem utilizando quantização vetorial e DPCM implementados com hardware analógico. O circuito projetado trabalha somente com imagens em escala de cinza e a compressão, que é feita antes de convertermos o valor dos pixels para digital, possui perdas.

Na saída do chip teremos uma taxa de aproximadamente 1,0 bit por pixel. O chip CMOS é formado por uma matriz de 32x32 pixels ou 64 blocos de 4x4 pixels. A saída

do CI é de 1056 bits, porque temos 15 bits para cada bloco de 4x4 pixels e são acrescentados 12 bits a cada linha de oito blocos 4x4 pixels, que fará a correção da divergência do DPCM. Portanto, cada foto comprimida que sairá do chip terá 1056 bits. Um PIC será responsável por ler os bits de saída do CI e enviá-los a um computador, onde um decodificador programado no MATLAB irá transformar o conjunto de bits recebidos em uma imagem.

Esse trabalho tem a intenção elaborar esse mesmo decodificador em linguagem de programação C/C++, para que ocorra um ganho de velocidade no processamento das imagens, bem como redução na quantidade de falhas que ocorrem durante a captura voltada para o MATLAB. Essas falhas serão descritas no Capítulo 4.

1.3 – Justificativa

Como foi mencionado na Seção 1.2, os bits gerados pelo chip serão decodificados por um código em MATLAB. No entanto, o MATLAB é uma linguagem interpretada . Linguagens interpretadas, em geral, são mais lentas do que as linguagens compiladas, que é o caso de C e C++. Para realizar o ajuste de foco e abertura da lente de sensor de imagem é necessário termos um *Viewfinder*, que pode ser compreendido como o visor utilizado para o enquadramento das imagens feito pelas câmeras. Ele é necessário para vermos em tempo real o que o sensor está capturando e para que os ajustes, como foco e abertura da lente, possam ser feitos.

Para implementar um *Viewfinder* de maneira simples, foram utilizados dois programas: um programa em C, que se comunicava com o PIC através da USB, lia os bits enviados e escrevia em um arquivo, e um outro programa em MATLAB, responsável pela decodificação. O decodificador em MATLAB já havia sido programado antes do projeto do chip, e o programa de comunicação com o PIC foi feito de forma independente em C/C++, por isso a solução mais simples para o projeto do *Viewfinder* foi juntar os dois programas, sem fazer muitas alterações nos códigos. Os dois programas funcionavam simultaneamente, um lendo e o outro escrevendo em um mesmo arquivo. Para que o sistema de comunicação e a decodificação acontecessem de forma aproximadamente sequencial um segundo arquivo era utilizado como *flag*. Assim, era possível ajustar de forma muito precária o foco e a abertura da lente, pois a

resposta do sistema não era imediata, uma vez que a decodificação no MATLAB é lenta e a necessidade de escrever em dois arquivos também prejudicava o desempenho. Além disso, o acesso simultâneo dos dois programas em um mesmo arquivo provocava alguns erros de travamento.

Neste projeto final, a decodificação e apresentação das imagens serão feitos em C/C++, o que possibilitará um aumento valioso na velocidade de processamento das imagens. Adicionalmente, ocorrerá uma melhoria na precisão de ajuste focal da lente da câmera. Com a taxa de vídeo aumentada, ao alterarmos o foco da lente, instantaneamente perceberemos na tela a modificação realizada; o que não acontecia no projeto antigo, devido ao tempo de resposta maior.

1.4 – Objetivos

- Operar uma câmera CMOS, que realiza compressão de imagens no plano focal, em taxas de vídeo acima de 1,0 Hz;

- Descobrir qual é a maior taxa de vídeo alcançável;

- Reduzir de dois para um o número de programas que representam o atual sistema;

- Corrigir problemas de travamento durante a execução do software, provenientes do programa antigo.

1.5 – Metodologia

A metodologia a ser utilizada neste trabalho consiste no cumprimento das seguintes etapas:

- estudo das funções elementares do MATLAB, bem como treinamento básico nesse software;

- compreensão do código do antigo projeto escrito em MATLAB;

- treinamento no ambiente de desenvolvimento Visual Studio 2008;

- estudo e treinamento das bibliotecas Armadillo e OpenCV, utilizadas na criação do decodificador;
- desenvolvimento do decodificador;
- adaptação da interface de comunicação com o PIC, de forma que esta interface passe a admitir operação com o novo decodificador;
- testes do sistema para verificação da taxa de vídeo máxima alcançável.

1.6 – Descrição

O Capítulo 2 apresentará os estudos e treinamentos referentes ao MATLAB, Visual Studio 2008 e bibliotecas Armadillo e OpenCV.

O Capítulo 3 mostrará o passo-a-passo do desenvolvimento e otimização do sistema (rotina de instruções) construído neste projeto.

O Capítulo 4 fará a exposição dos resultados e testes feitos no sistema.

O Capítulo 5 apresentará a conclusão do projeto.

Capítulo 2

Fundamentos Teóricos

Para o desenvolvimento deste projeto foi necessário o estudo e a compreensão de alguns assuntos aqui abordados. A Seção 2.1 mostrará como ocorre o processo de codificação das imagens fotografadas. Na Seção 2.2, será visto uma abordagem simplificada do decodificador feito no MATLAB. A Seção 2.3 explicará como ocorre a comunicação PIC-PC. A Seção 2.4 mostrará o ambiente de desenvolvimento, no qual foi programado o presente sistema. A Seção 2.5 apresentará uma das principais bibliotecas utilizadas no desenvolvimento deste sistema, a Armadillo, e também suas principais funções e métodos. A Seção 2.6 apresentará a biblioteca OpenCV e suas principais funções e métodos usados na parte gráfica do código.

2.1 – Codificador do MATLAB

Informamos ao leitor que esta seção visa facilitar a compreensão do decodificador, que explicaremos na Seção 2.2. No entanto, salientamos que não é do escopo deste projeto a implementação de um codificador, por isso manteremos simples e resumida a explicação do mesmo.

O codificador em questão pode ser compreendido como uma sequência de etapas para compressão de informações, obtidas a partir de figuras capturadas pela câmera digital. Essa compressão é realizada seguindo a orientação do diagrama de blocos da Fig. 2.1.

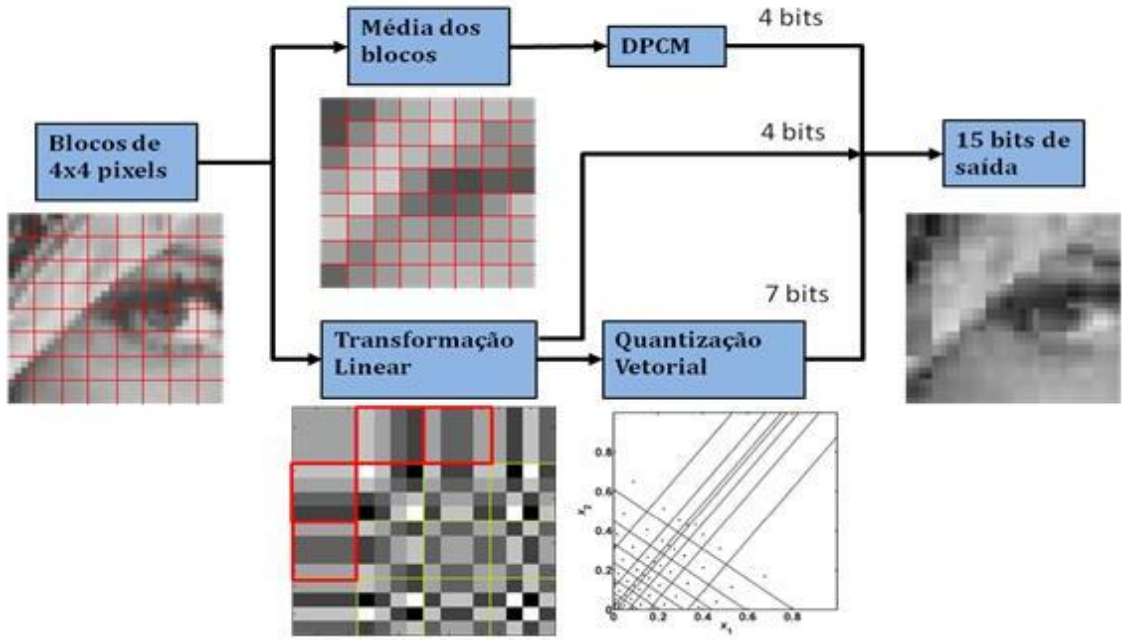


Figura 2.1: Representação do método de compressão do codificador.

Primeiramente, a imagem é dividida em blocos de 4x4 pixels, também chamados de amostras, que são enviados aos blocos "Transformação Linear" e "Média dos blocos". Em "Média dos blocos", são realizadas as médias de cada bloco de 4x4 pixels. As médias são importantes para que o DPCM possa realizar a diferença entre a média de um bloco e seu adjacente, resultando numa quantidade de informações reduzida, que agora poderão ser representadas com menos bits. O DPCM é um método de compressão que pode ser usado em imagens quando amostras adjacentes têm grande similaridade de informação, valor das médias próximas; justificando que é mais interessante transmitir a diferença entre as médias do que as médias em si. Ao final da compressão feita pelo DPCM, temos parte das informações das figuras armazenadas em quatro bits.

Paralelamente aos processos do parágrafo anterior, ocorrem a transformação linear e outro método de compressão, chamado quantização vetorial (VQ). A transformação linear consiste em uma rotação dos eixos coordenados de forma que, após esta rotação, alguns eixos concentrem a maior parte da energia do vetor original. Estes eixos são chamados de componentes principais e costumam ser mencionados em ordem decrescente de energia. No nosso caso, do total de 16 dimensões de cada bloco 4x4, apenas algumas dessas dimensões são consideradas: somente as quatro componentes de maior energia, marcadas em vermelho na Fig. 2.1, são utilizadas, pois

essas componentes possuem uma maior quantidade de informação da imagem. As 11 componentes restantes são descartadas. Após a transformação linear, serão calculados os módulos das quatro componentes que serão enviados para o estágio de VQ. Para representar os sinais de cada componente, são utilizados quatro bits.

Por último, utilizaremos a técnica de VQ para representar os módulos das quatro componentes geradas pela transformação linear. Nesta etapa, já temos menos dimensões para comprimir, apenas quatro, que formam um vetor. O trabalho do quantizador é mapear um conjunto grande de números em um conjunto pequeno de números, conhecidos como índices, que servem para selecionar vetores de reconstrução a partir de um dicionário, que será definido a seguir. Mais uma vez, queremos representar uma quantidade grande de dados com poucos bits, no caso serão sete bits. O espaço de quatro dimensões será dividido em células, onde o valor mais representativo da célula é o centroide. O VQ irá mapear cada vetor que contém as quatro componentes em uma dessas células. O dicionário do VQ é o conjunto de todos os centroides.

Apesar da perda de dados causada pelo descarte das 11 componentes, ainda assim a imagem comprimida consegue representar bem a imagem real, como podemos observar na Fig. 2.1.

2.2 – Decodificador versão MATLAB

O antigo projeto tinha um conjunto de etapas que começavam na captura de uma imagem e compressão feita pelo chip. Em seguida, os dados gerados pela compressão eram enviados ao PIC, que por sua vez os entregava à interface do usuário, programada em C/C++ e responsável por realizar a comunicação entre o PIC e o PC através da porta USB de acordo com o que foi pedido pelo usuário. A interface armazenava os dados em um arquivo de texto, que era enviado ao decodificador para ser processado. A ilustração dessas etapas mencionadas pode ser vista na Fig. 2.2.

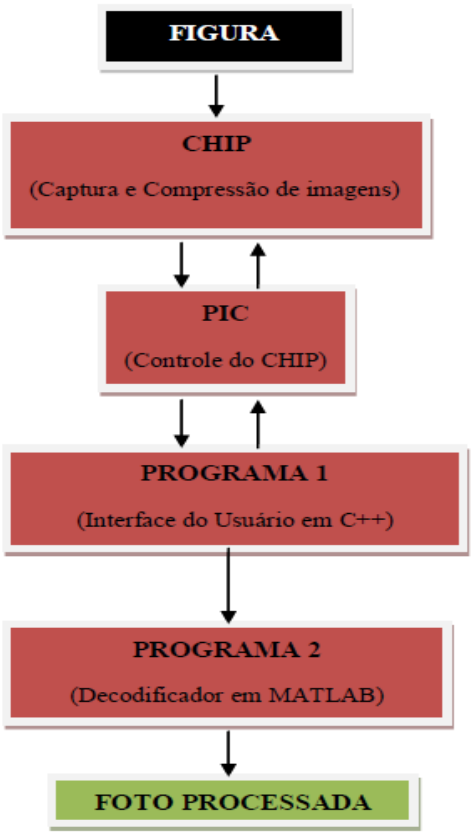


Figura 2.2: Etapas do antigo projeto para capturar uma imagem, processá-la e apresentá-la ao usuário.

A versão do decodificador [2] implementada através do MATLAB tem funcionamento representado pelo diagrama de blocos da Fig. 2.3. Um arquivo chamado EntradaBinaria.txt, que contém os dados mencionados no primeiro parágrafo desta seção, entra no decodificador, cuja estrutura pode ser encontrada no Apêndice A. No decodificador, a EntradaBinaria.txt será processada e resultará numa imagem na tela, apresentada na Fig. 2.4.

De forma resumida, podemos dizer que o código apresentado no Apêndice A executa as seguintes tarefas: simulação da codificação de uma figura, leitura da EntradaBinaria.txt e decodificação da figura. A terceira tarefa sempre será realizada. No entanto, o código irá realizar a primeira ou a segunda tarefa de acordo com o estado da variável "*EncoderTeorico*". Se essa variável for igual a 1, a compressão é feita através de um código que simula o que o chip faz, e se essa variável for igual a 0, os bits enviados pelo chip são utilizados para a decodificação.

A simulação da codificação de uma imagem é uma representação em software do método de compressão analógica realizado pelo chip CMOS, em hardware. Nesta parte, são usados os algoritmos de transformação linear, quantização vetorial e DPCM para que uma imagem de 32x32 pixels possa ser representada, após a compressão, por 1056 bits. Esse resultado da compressão será a entrada binária do decodificador.

Caso a variável "*EncoderTeorico*" seja igual a zero, é feita a leitura do arquivo de texto chamado de EntradaBinaria.txt. Esse arquivo é lido e seus valores são armazenados em quatro matrizes: a primeira contém os bits que representam os sinais das componentes após a transformação linear; a segunda contém todos os bits relativos ao VQ; a terceira são os bits do DPCM e a quarta os bits de correção do DPCM. Essas quatro matrizes serão encaminhadas para a etapa de decodificação.

Na decodificação, os valores de EntradaBinaria.txt, agora armazenados nas matrizes mencionadas, serão processados por dois decodificadores: um DPCM e um VQ. Através dos bits recebidos pelos decodificadores, serão gerados índices para a consulta de dois dicionários, um do DPCM e outro do VQ. Após esses processos, teremos a reconstrução de uma aproximação da imagem original, armazenada em uma matriz de dimensão 32x32. No final da decodificação, será apresentada na tela uma figura com quatro regiões, sendo: resultado do DPCM, resultado do VQ, imagem original e a média das imagens capturadas até o momento da impressão na tela, como pode ser visto na Figura 2.4.

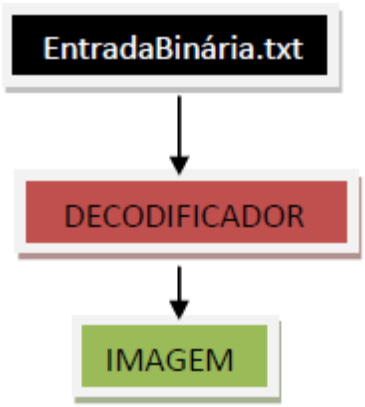


Figura 2.3: diagrama de blocos do antigo Decodificador.

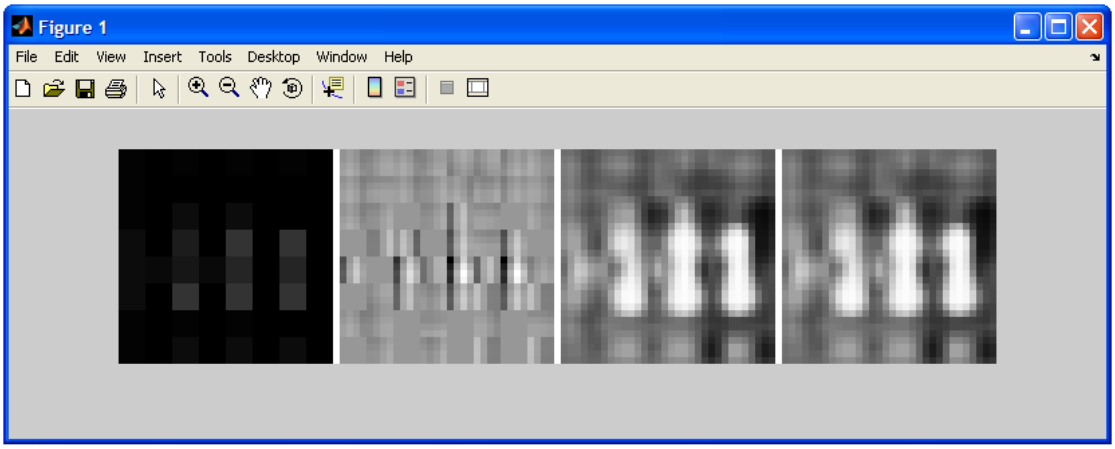


Figura 2.4: Foto gerada pelo antigo decodificador.

2.3 – Rotina de Tratamento da Entrada Binária

Como mencionado anteriormente, EntradaBinaria.txt é uma arquivo recebido pelo decodificador que representa uma foto capturada pela câmera.

No antigo sistema, cada entrada binária era criada através de uma rotina que armazenava seu conteúdo no arquivo de texto EntradaBinaria.txt. Esse arquivo era responsável por fazer a comunicação entre os módulos escritos em duas linguagens de programação diferentes: interface do usuário (C/C++) e decodificador (MATLAB). A interface do usuário recebia e tratava os bits enviados pela câmera, e esses eram enviados pelo arquivo de texto até o decodificador.

A rotina que trata os bits enviados pela câmera é um método “escrever” da classe “tratamento de arquivo”. Relembramos que toda a figura capturada pela nossa câmera digital é imediatamente dividida em blocos de 4x4 pixels. A matriz fabricada possui 32x32 pixels, logo, temos oito linhas que contêm oito blocos cada. Ou seja, para cada linha de blocos da figura, teremos 8x4x4 pixels = 128 pixels. Após passar pelo circuito de compressão, cada linha da figura será representada por 132 bits. Portanto, depois que todas as 8 linhas forem comprimidas teremos 8x132= 1056 bits.

No entanto, o PIC só consegue enviar para a interface do usuário dados do tipo char, e esses precisam ser modificados para bits, para serem lidos pelo MATLAB. Cada

char é formado por oito bits, então cada linha de blocos da figura, após compressão, será representada por 132/8 = 16,5 char. Para reescrever esse valor em bits, o método "escrever" executa um loop que realiza 17 iterações de conversão char para bit. As primeiras 16 iterações para os primeiros 16 char e mais uma iteração para o "meio char" restante. O loop precisará ser acionado oito vezes, já que temos oito linhas de blocos de 4x4 pixels. Após descobrirmos o valor de cada bit do char, esses bits serão guardados em um vetor que contém somente zeros e uns. A partir desse vetor, escrevemos a "EntradaBinaria.txt".

Para separar os bits de um char, utilizamos uma variável que serve como máscara, e que contém sete bits iguais a 0 e somente um bit igual a 1. Esse bit será deslocado a cada iteração, até que o 1 tenha passado por todos os oito bits da variável. Com essa máscara, fazemos operações lógicas do tipo AND com o char recebido. Desejamos descobrir o valor de um determinado bit do char recebido. O bit que é analisado a cada iteração é aquele que está na mesma posição do bit igual a 1 da máscara. Sabemos que o resultado da operação AND só será igual 1 se o bit analisado também tiver valor 1. Assim, dependendo desse resultado, preenchemos o vetor de bits com o qual escreveremos o arquivo enviado para o decodificador.

Informamos que no sistema atual não há mais necessidade de se escrever os 1056 bits num arquivo de texto, para que ocorra a comunicação entre interface e decodificador. Justificamos isso dizendo que esses módulos agora estão unidos num só bloco de processamento que será explicado no Capítulo 3. Entretanto, a conversão de char para bits ainda é necessária, e uma nova rotina para esse fim será apresentada na Seção 3.2.

O código do método escrever pode ser visualizado no Apêndice B.

2.4 – Visual Studio 2008

Para o desenvolvimento do decodificador em linguagem de programação C/C++ foi utilizado o software da Microsoft Visual Studio 2008. O VS2008 é uma IDE (*integrated development enviroment*) que permite o desenvolvimento nas mais

diferentes linguagens de programação: C, C++, Java, Python, Perl e outras. Ele também possibilita a implementação de um código que contenha essas linguagens mescladas: por exemplo, um programa que possua metade do seu código escrito em Perl e a outra metade escrito em Java.

Isso é possível ao efetuarmos a configuração /clr , apresentada detalhadamente na Seção 2.3.1, num projeto do VS2008. O significado de "clr" é *common language runtime*, que pode ser traduzido como rodando em linguagem comum. O que um projeto em VS2008 faz ao usar essa configuração é aceitar e compilar duas ou mais linguagens de programação num código, que após compilação se tornam uma linguagem intermediária da Microsoft chamada de MSIL (*Microsoft intermediate language*). Em seguida, a MSIL vira a linguagem de máquina comum, que é o resultado de qualquer compilação tradicional. Mais informações sobre o processo de compilação com a configuração de /clr são encontradas em [7] até [10].

Neste sistema foi necessário usar a configuração /clr, para possibilitar a união e compilação dos códigos do decodificador atual com os códigos de controle do PIC e interface do usuário num mesmo projeto VS2008. Sem esse tipo de configuração, teríamos erros durante o processo de compilação.

2.4.1 – Configuração /clr

Para facilitar a compreensão do leitor sobre a configuração /clr deste projeto, apresentaremos aqui nesta sub-seção como ela foi feita.

Usando o próprio projeto em questão, para acessarmos a configuração /clr, devemos:

1- Clicar com o botão direito do mouse no título do projeto e selecionarmos "Properties";
2- Na janela referente às propriedades do projeto, neste caso "MSHPUSB PnP Demo 2 Property Pages", apresentado em Fig. 2.5, deveremos clicar em "*Common Language Runtime Support*" e selecionar "*Common Language Runtime Support*(/clr)"

3- Clicar em “Aplicar”, e em seguida clicar em “Ok”.

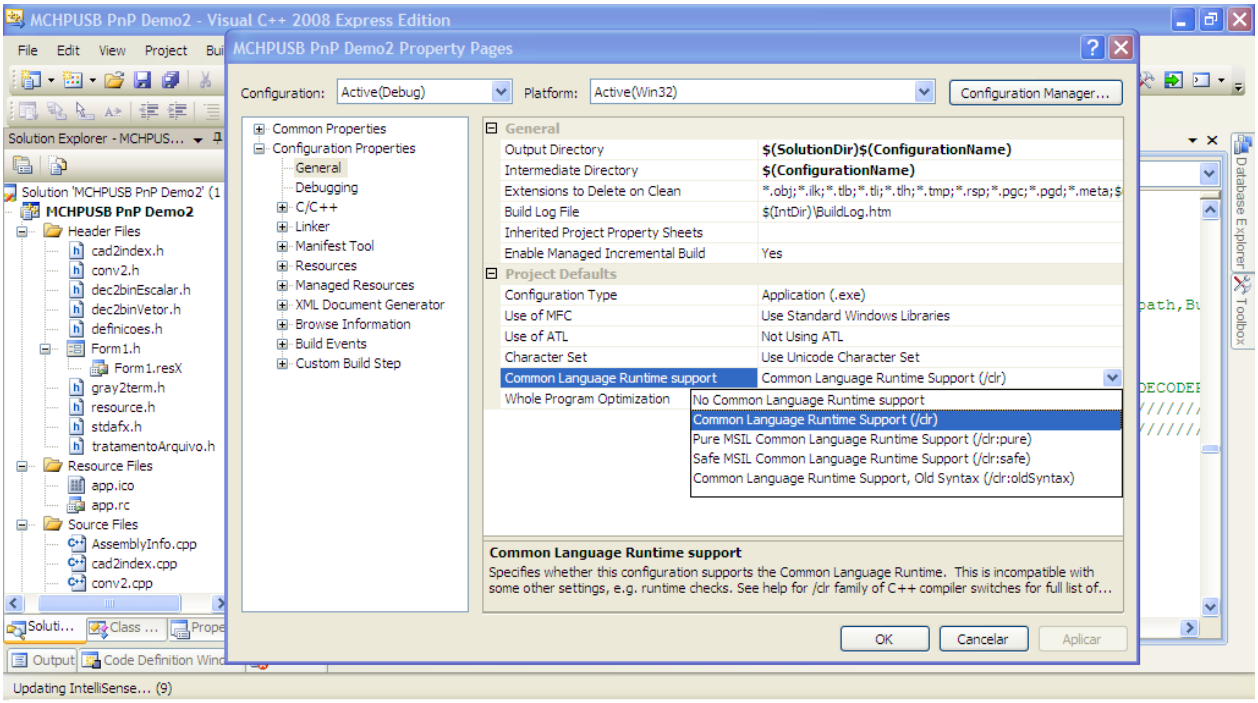


Fig.2.5: Projeto VS2008 sendo configurado para /clr.

2.5 – Armadillo

Armadillo é uma das principais bibliotecas para C++ usadas neste projeto. Ela contém um pacote de funções para facilitar o desenvolvimento de códigos que envolvem manipulação de matrizes. O Armadillo tenta, com significativo êxito, se parecer com um MATLAB para o C++. Mais detalhes em [11].

Descreverei abaixo as funções e métodos que foram utilizados neste projeto e que são oriundos da Armadillo:

1. **zeros<type>(k,k):** retorna uma matriz com k x k zeros.
2. **ones<type>(k, k):** preenche com valor um todos os elementos da matriz do tipo type;

3. **trans(A) :** transpõe a matriz A. Um exemplo dessa função pode ser vista na Tab. 2.1;

Tabela 2.1 – Aplicação da função trans(A)

```
Mat<double> A; //cria matriz A de elementos tipo double
A << 3.0 << 0.0 <<endr //inicialização da matriz 2x2
  << 2.0 << 5.0 <<endr;
Mat<double> B; //cria matriz B de elementos tipo double
B = trans(A); //matriz sera 2x2
/*A matriz resultante sera:
  B = [3.0 2.0
       0.0 5.0]
*/
```

4. **conv_to<type>::from(X):** é similar a um type-casting; faz a conversão do tipo da matriz X para o tipo type desejado. Uma aplicação desta função pode ser vista na Tab. 2.2;

Tabela 2.2 – Uma aplicação da função conv_to<*type*>::from(X)

```
Mat<double> A;//cria matriz A de elementos tipo double
A << 1.0 << 2.0 <<endr; //inicialização da matriz A de
                        //tamanho 1x2
Mat<int> B; //cria matriz B de elementos inteiros
B = conv_to<int>::from(A); //converte elementos de A para
                           //inteiros e os copia na matriz B
/* A matriz resultante sera:
   B = [1 2]
*/
```

5. **A.n_elem:** devolve o número total de elementos de uma matriz A;

6. **reshape(A, n_linhas, n_colunas, dim=0):** modifica a dimensão da matriz A para n_linhas e n_colunas. A matriz resultante não precisa ter o mesmo total de elementos da matriz original. Se o tamanho da matriz desejada, isto é (n_linhas)x(n_colunas), for maior que a matriz original, então os elementos restantes da matriz desejada serão completados com zero. No entanto, se (n_linhas)x(n_colunas) for menor que a matriz original, então a matriz desejada

será apenas um sub-conjunto do total de elementos de A. A ordem de leitura dos elementos da matriz A é por linha(dim=0), ou seja, pegam-se os elementos da primeira linha, em seguida pegam-se os elementos da segunda linha e assim por diante. Um exemplo dessa aplicação pode ser visto na Tab. 2.3;

Tabela 2.3 – Aplicação da função reshape(A, n_linhas, n_colunas)

```
Mat<double> A; //cria matriz A de elementos tipo double
A << 5.0 << 1.0 <<endr //inicialização da matriz 2x2
  << 9.0 << 2.0 <<endr;
Mat<double> B; //cria matriz B de elementos tipo double
B = reshape(A, 3, 3, dim=0); //matriz sera 3x3
/*A matriz resultante sera:
  B = [5.0 1.0 9.0
       2.0 0.0 0.0
       0.0 0.0 0.0]
*/
```

7. **A(span(n_linha),span(n_coluna)):** retorna o elemento da matriz A indicado por (n_linha)x(n_coluna);

8. **flipud(A):** devolve a matriz A, mas com suas linhas revertidas. Se o número de linhas for 32, a k-ésima linha passará a ser a linha de número 33-k. Um exemplo dessa função pode ser visto na Tab. 2.4;

Tabela 2.4 – Uma aplicação da função flipud(A)

```
Mat<double> A; //cria matriz A de elementos tipo double
A << 1.0 << 1.0 <<endr //inicialização da matriz 2x2
  << 2.0 << 2.0 <<endr
  << 3.0 << 3.0 <<endr;
Mat<double> B;//cria matriz B de elementos tipo double
B = flipud(A);
/*A matriz resultante sera:
  B = [3.0 3.0
       2.0 2.0
       1.0 1.0]
*/
```

9. **join_rows(A, B):** duas matrizes A e B de mesmo número de linhas, mas que não necessariamente possuem o mesmo número de colunas, são unidas conectando suas linhas correspondentes. Um exemplo dessa função pode ser visto na Tab. 2.5;

Tabela 2.5 – Uma aplicação da função join_rows(A, B)

```
Mat<double> A;//cria matriz A de elementos tipo double
A << 1.0 << 1.0 <<endr //inicialização da matriz 2x2
  << 1.0 << 1.0 <<endr;
Mat< double > B;// cria matriz B de elementos tipo double
B << 2.0 <<endr //inicialização da matriz 2x1
  << 2.0 <<endr;
Mat<double> C;//cria matriz C de elementos tipo double
C = join_cols(A,B);//matriz sera 2x3
/*A matriz resultante sera:
  C = [1.0 1.0 2.0
       1.0 1.0 2.0]
*/
```

10. **A.fill(k):** preenche toda a matriz A com o valor k;

11. **A.n_rows:** devolve o número de linhas da matriz A;

12. **join_cols(A, B):** duas matrizes A e B de mesmo número de colunas, mas que não necessariamente possuem o mesmo número de linhas, são unidas conectando suas colunas correspondentes. Um exemplo dessa função pode ser visto na Tab. 2.6;

Tabela 2.6 – Uma aplicação da função join_cols(A, B)

```
Mat<double> A; //cria matriz A de elementos tipo double
A << 1.0 << 2.0 <<endr //inicialização da matriz 2x2
  << 3.0 << 4.0 <<endr;
Mat< double > B; // cria matriz B de elementos tipo double
B << 5.0 << 6.0 <<endr //inicialização da matriz 2x2
  << 7.0 << 8.0 <<endr;
Mat<double> C; //cria matriz C de elementos tipo double
C = join_cols(A,B); //matriz sera 4x2
/*A matriz resultante sera:
  C = [1.0 2.0
       3.0 4.0
       5.0 6.0
       7.0 8.0]
*/
```

13. **A.n_cols:** devolve o número de colunas da matriz A;

14. **min(A, dim=0) :** retorna um vetor com os menores valores de cada coluna da matriz A;

15. **max(A, dim=0) :** retorna um vetor com os maiores valores de cada coluna da matriz A;

16. **mean(A, dim=0):** retorna um vetor com os valores médios de cada coluna da matriz A;

17. **kron(A,B):** calcula o produto de Kronecker [15] entre as matrizes A e B. Na Matemática, esse produto é denotado pelo símbolo ⊗, onde A⊗B resulta numa matriz bloco. No contexto deste projeto, o que essa função faz é simplesmente ampliar uma imagem. Um exemplo dessa função pode ser visto na Tab. 2.7 ;

Tabela 2.7 – Uma aplicação da função Kron(A, B)

```
//Considerar a matriz A como uma imagem
//a matriz B como o elemento que ampliara a nossa foto
//e C sera a nossa matriz resultado
Mat<double> A; //cria matriz A de elementos tipo double
A << 3.0 << 4.0 <<endr //inicialização da matriz 2x2
  << 5.0 << 6.0 <<endr;
Mat< double > B; // cria matriz B de elementos tipo double
//Notar que a matriz B foi inicializada somente com
//elementos 1, pois só queremos expandir a matriz A e
//não mudar seus valores
B << 1.0 << 1.0 <<endr //inicialização da matriz 2x2
  << 1.0 << 1.0 <<endr;
Mat<double> C; //cria matriz C de elementos tipo double
C = kron(A,B); //matriz sera 4x4
//Observar que apos o Produto de Kronecker nossa foto foi
//ampliada para um tamanho final 4x4
/*A matriz resultante sera:
  C = [3.0 3.0 4.0 4.0
       3.0 3.0 4.0 4.0
       5.0 5.0 6.0 6.0
       5.0 5.0 6.0 6.0]
*/
```

18. A.save("A.txt", raw_ascii): salva o conteúdo da matriz A num arquivo txt.

Maiores esclarecimentos sobre a biblioteca Armadillo podem ser encontrados em [12].

Do conjunto de 18 funções e métodos, 6 são usados com bastante frequência para a construção do sistema: zeros() e ones(), para a inicialização de matrizes, join_cols() e join_rowns(), para a concatenação de matrizes ou vetores, .span(), para localização de elementos em matrizes ou vetores; e .save para armazenarmos em arquivos de texto dados das matrizes.

2.6 – Open Source Computer Vision

A biblioteca Open Source Computer Vision (OpenCV) [13] é uma biblioteca de uso gratuito que possui muitos recursos para se trabalhar com matrizes, imagens, áudio e vídeo. É possível o uso conjunto da OpenCV com as seguintes linguagens de

programação: C, C++, Python e Java. É oferecido o suporte para Windows, Mac OS, iOS e Android.

Neste projeto, a utilidade da OpenCV resume-se à impressão na tela de imagens já processadas pelo sistema. Para exemplificar o uso da OpenCV, utilizaremos como alvo a região do olho direito da Fig. 2.6, chamada Lena. Na Fig. 2.7, apresentamos o resultado da imagem gerada e decodificada pelo sistema atual. A construção da Fig. 2.7 será explicada em detalhes no Capítulo 3.

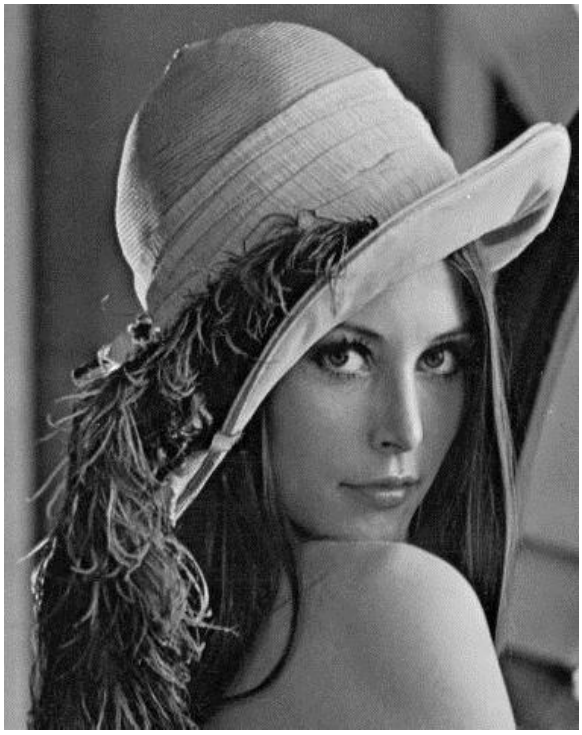


Figura 2.6: Foto da Lena

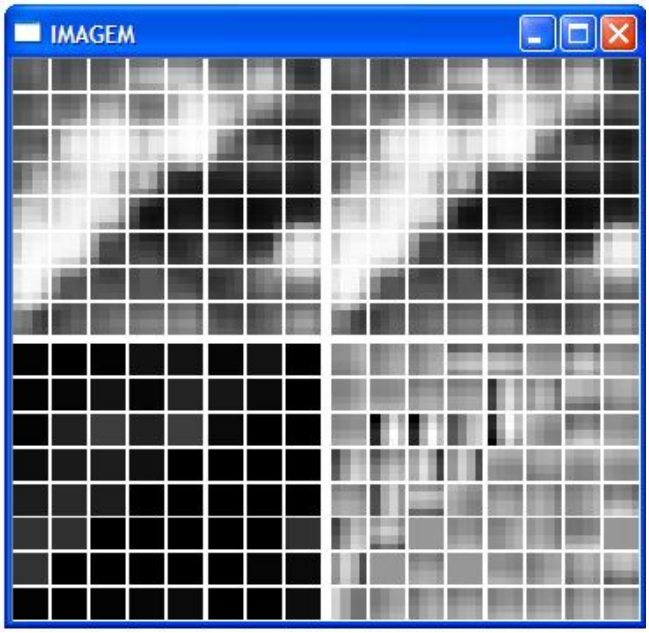


Figura 2.7: Foto gerada pelo novo decodificador.

Descreverei abaixo as funções e métodos que foram utilizados neste projeto e que são oriundos do OpenCV. As funções foram enumeradas na ordem em que aparecem no código do decodificador e todas são necessárias para a apresentação da Fig. 2.7:

1. **zeros(n_linhas, n_colunas, *type*)** : preenche com valor zero todos os elementos da matriz do tipo *type*;
2. **A.convertTo(A, *type*)** : é similar a um *type-casting*, faz a conversão do tipo da matriz A para o tipo *type* desejado;
3. **namedWindow("IMAGEM", flag=0)**: cria uma janela com o título IMAGEM, cujo tamanho é auto-ajustado(flag=0) para a figura exibida;
4. **imshow("IMAGEM", A)** : coloca a imagem proveniente da matriz A numa janela de título IMAGEM. A matriz A pode ser a representação de fotos coloridas ou em preto e branco. Os valores de A devem pertencer a faixa de valores entre 0 a 255. No entanto, caso estejam fora dessa faixa, imshow() fará a conversão correspondente;
5. **waitKey(k)** : fornece um tempo de k milissegundos, para que uma imagem fique sendo apresentada na tela. Deve ser usada conjuntamente com as funções "namedWindow()" e "imshow()";

Para tornar mais clara a utilização das funções da OpenCV, mostramos na Tab. 2.8 um trecho do código do decodificador referente à construção da Fig. 2.7.

Tabela 2.8 – Construção de uma foto do decodificador usando as funções do OpenCV

```
cv::Mat auxCV; //Declaração de um objeto do tipo Mat da
               //bilbioteca OpeCV
auxCV = cv::Mat::zeros(339, 339, CV_64F); //inicializando
                                          //matriz auxCV, tipo
                                          //CV_64F = double, de
                                          //tamanho 339x339 com
                                          //elementos zeros
auxCV.convertTo(auxCV, CV_32F); //elementos da matriz auxCV,
                                //tipo double, serao convertidos
                                //para tipo CV_32F = float
cv::namedWindow( “IMAGEM”, 0 ); //cria uma janela com titulo
                                //IMAGEM e com tamanho
                                //correspondente a dimensão da
                                //figura
imshow( “IMAGEM”, auxCV ); //recebe matriz auxCV e exibe seu
                           //conteudo na janela de titulo IMAGEM
cv::waitKey(20); //a imagem ficara na tela durante 20ms
```

Capítulo 3

Métodos

Neste capítulo, são apresentados o desenvolvimento do código do decodificador e algumas modificações feitas sobre a rotina de tratamento das informações oriundas da câmera digital. Na Seção 3.1, é explicado como ocorre o processamento de imagens, quais são seus processos internos, como esses processos se comunicam e quais são os seus resultados. Serão mostradas as funções auxiliares e suas respectivas entradas e saídas, que colaboram para o funcionamento do decodificador. Por fim, na Seção 3.2, será apresentada a rotina de programação que faz o processamento dos bits enviados pela câmera digital.

3.1 – Decodificador versão C/C++

O decodificador é um conjunto de processos, que ocorrem no novo sistema, para gerar uma imagem. Essa etapa faz parte de um grupo de procedimentos que tem início na aquisição de uma figura pelo chip CMOS, onde os dados são comprimidos e enviados para o PIC, que os entrega para o novo sistema. O diagrama de blocos que representa todo o grupo de procedimentos acima está presente na Fig.3.1.

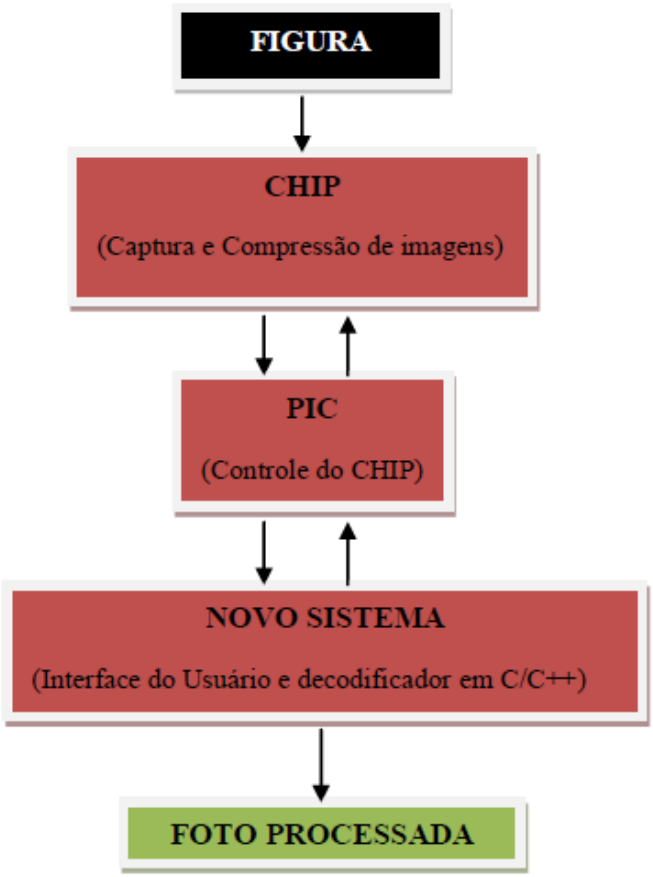


Figura 3.1: Etapas do novo projeto para capturar uma imagem, processá-la e apresentá-la ao usuário.

A Fig. 3.1 é semelhante ao grupo de procedimentos do antigo projeto, representado pela Fig. 2.1. Sua única diferença é bloco "Novo Sistema", que no projeto anterior era formado por dois programas: Interface do usuário (C++) e decodificador (MATLAB).

No interior de "Novo Sistema", estão localizados os blocos "Interface do Usuário" e "Decodificador", ilustrados pela Fig. 3.2.

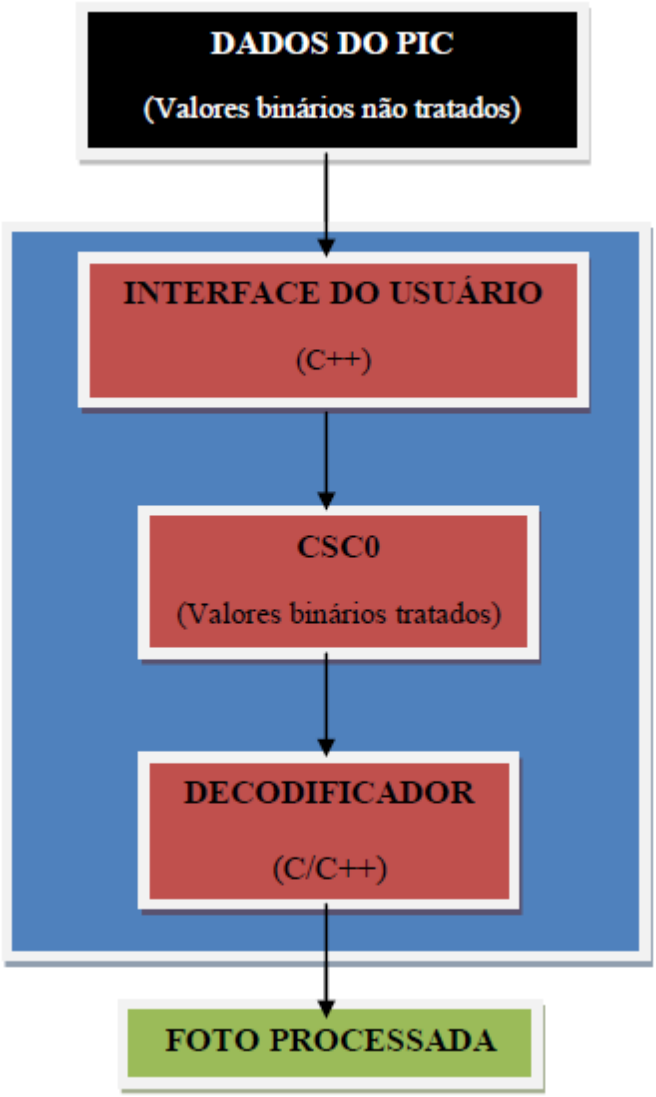


Figura 3.2: Diagrama de blocos do novo sistema.

Na atual versão do decodificador, optou-se por escrever o código em C/C++ por serem essas as linguagens utilizadas na construção dos códigos anteriores: comunicação do PIC (linguagem C) e Interface Gráfica (linguagem C++).

O decodificador em C/C++ é representado pelo diagrama de blocos da Fig. 3.3.

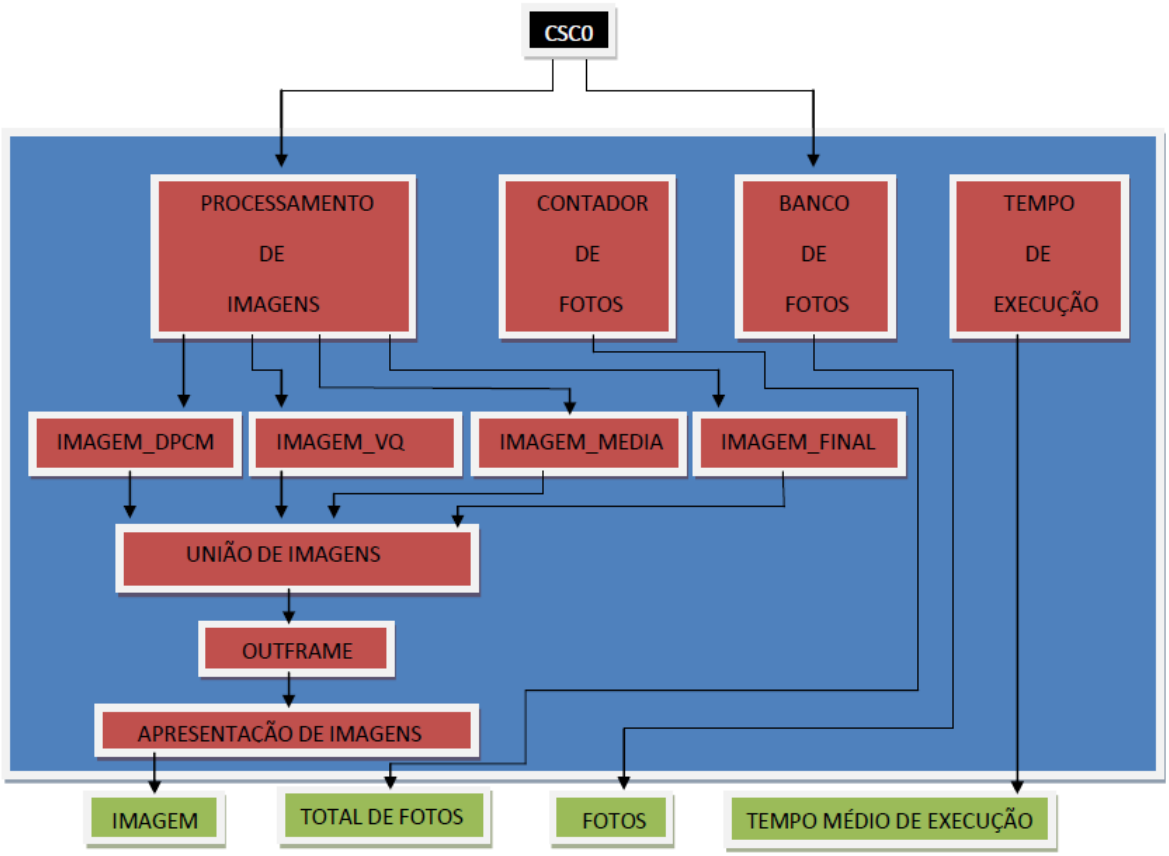


Figura 3.3: Diagrama de blocos do novo decodificador.

No primeiro parágrafo desta seção, mencionamos que dados são passados do PIC para o novo sistema. Esses dados são valores binários, zeros e uns, que após um determinado processamento resultarão numa variável chamada de CSC0. Esse processamento e a construção de CSC0 serão apresentados na Seção 3.2. Essa variável consiste em uma linha com 1056 bits [3]. Essa matriz 1 × 1056 é o resultado da compressão e digitalização de uma imagem capturada pela câmera.

Ao entrar no decodificador, CSC0 é encaminhado para dois processos: um é representado pelo bloco “Processamento de Imagens” e o outro é o bloco “Banco de Fotos”.

Primeiramente abordaremos o bloco “Banco de Fotos”. Ao receber CSC0, “Banco de Fotos” o armazena num arquivo chamado FOTOS.txt. Isso é feito, para que seja possível posterior processamento pelo usuário. Em FOTOS.txt, temos todas as figuras capturadas pela câmera. Esse arquivo é elaborado de forma tal que cada linha é uma matriz 1 x 1056, e os elementos(bits) de cada matriz são separados pelo caractere

espaço em branco, ou seja, “ ”. É pertinente dizer que esse arquivo não contém nenhum tipo de codificação, sendo possível verificar seu conteúdo através de qualquer programa Editor de Texto. Com o intuito de auxiliar na compreensão do leitor, exibiremos na Fig. 3.4 o conceito do arquivo “txt” mencionado neste parágrafo.

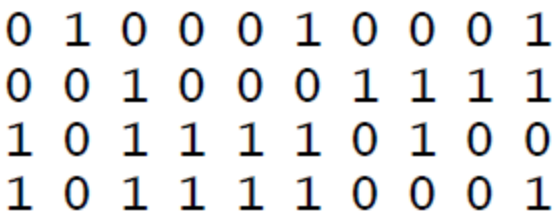


Figura 3.4: Representação do conceito do arquivo FOTOS.txt, onde cada linha contém os dados da imagem comprimida. (cada linha contém, na realidade, 1056 bits).

Através da Fig. 3.4, tentamos ilustrar para o leitor a visão que teria ao abrir o arquivo com as fotos.

“Processamento de Imagens” corresponde ao código do decodificador propriamente dito. De forma simplificada, o que ocorre dentro desse bloco está apresentado na Fig. 3.5 .

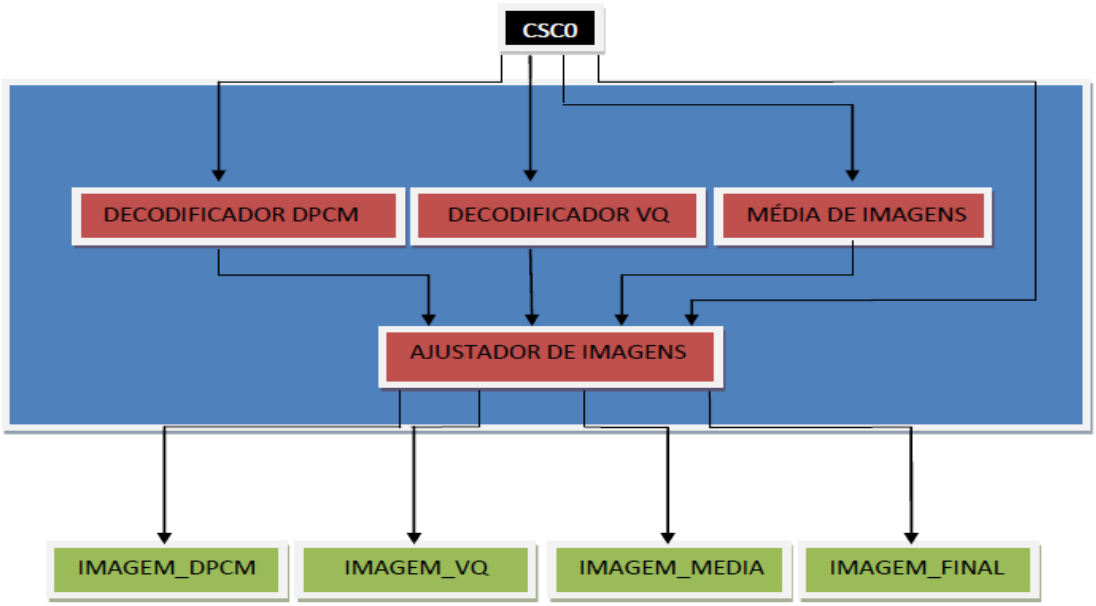


Figura 3.5: Procedimentos internos ao bloco “Processamento de Imagens”.

Podemos observar na Fig. 3.4 que a matriz de bits passa por quatro processos: Ajustador de Imagens, Decodificador DPCM, Decodificador VQ e Média de Imagens. O Ajustador de Imagens basicamente redimensiona a foto original para uma matriz de 160x160. Decodificador DPCM e Decodificador VQ já foram explicados na Seção 2.2 do Capítulo 2. Média de Imagens apenas acumula numa matriz todas as fotos que foram tiradas de uma figura e faz a média aritmética entre elas. Após esses processos obtemos quatro imagens, que nada mais são do que matrizes 160x160: "Imagem_Final", "Imagem_Media", "Imagem_DPCM" e "Imagem_VQ". Essas matrizes podem ser impressas na tela usando-se a função "imshow()" da biblioteca OpenCV.

Ao mostrarmos os resultados para o usuário, é conveniente colocarmos uma das imagens finais, que é instantânea, na parte superior esquerda e, à sua direita, o resultado médio. Abaixo destas imagens, são mostrados o resultado do DPCM à esquerda e o resultado do VQ à direita. A variável que contém as imagens dispostas desta maneira é chamada de "OutFrame" (Fig. 3.6) e ela é montada pela função "União de Imagens".

As imagens serão enviadas para o bloco "União de Imagens", para formar uma só figura, representada pela variável OutFrame. Como a compressão é feita por blocos de 4x4 pixels, é interessante analisar o efeito da compressão em cada um desses blocos. Assim, para facilitar essa visualização, acrescentamos às imagens uma grade, de forma que cada imagem é apresentada dividida em 64 blocos, isto é, 8x8 blocos de 4x4 pixels.

Apresentamos na Fig. 3.6 o conceito de OutFrame.

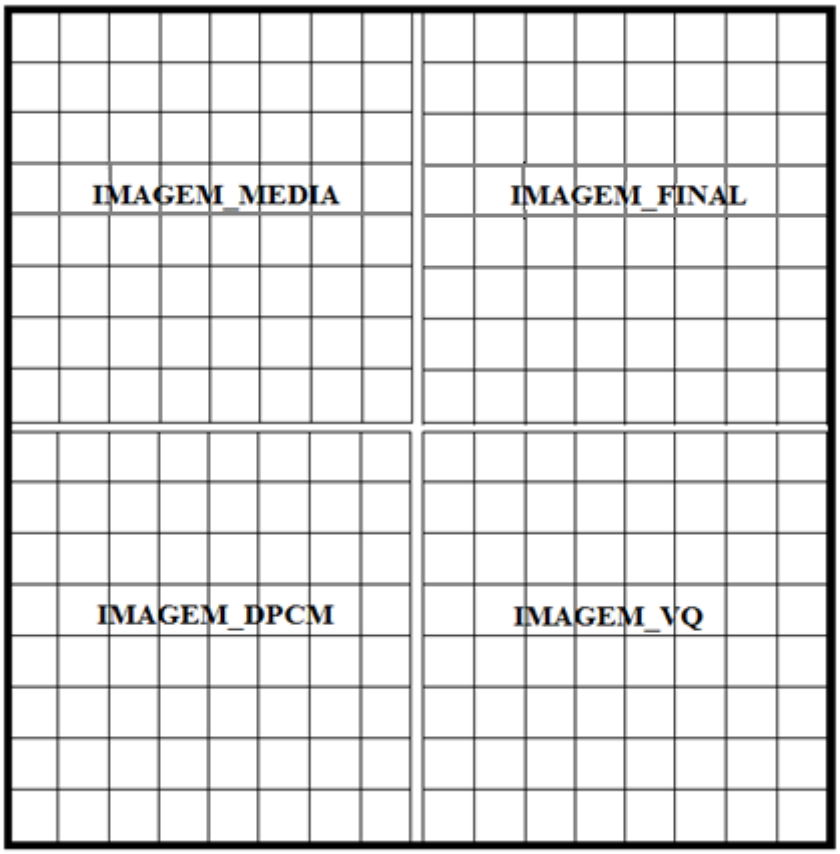


Figura 3.6: Conceito de posicionamento e apresentação das imagens do bloco "União de Imagens".

Antes de ser apresentada ao usuário a janela IMAGEM, contendo a figura final, OutFrame será tratada no bloco "Apresentação de Imagens", que é o processo construído com a ajuda da biblioteca OpenCV. Caracterizamos os processos internos do bloco "Apresentação de Imagens" na Fig. 3.7.

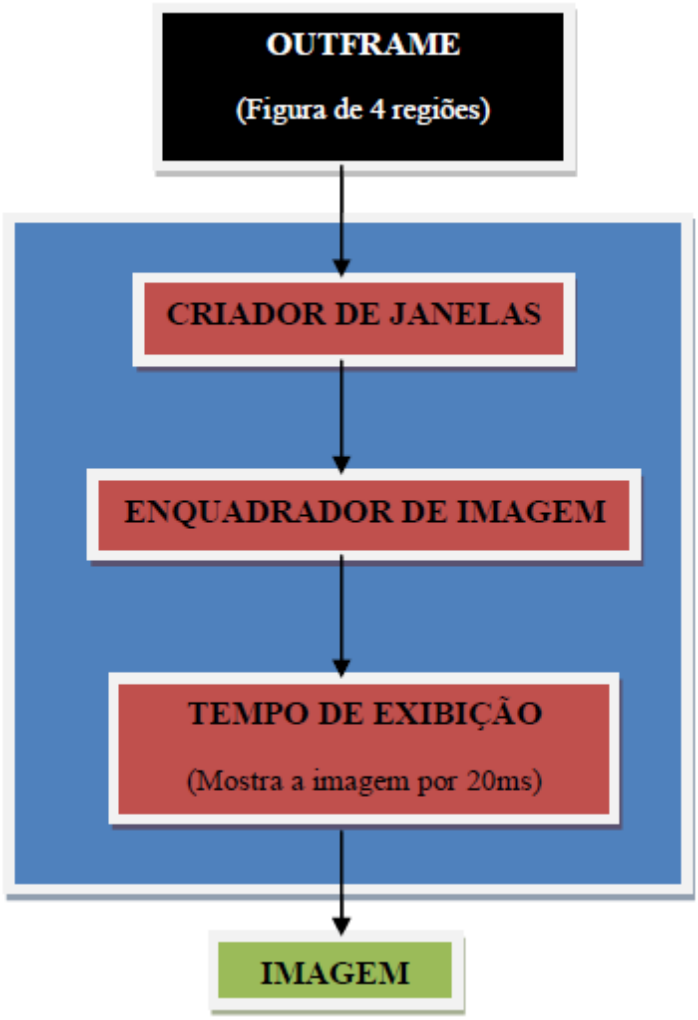


Figura 3.7: Procedimentos internos ao bloco "Apresentação de Imagens".

Na Fig. 3.7, OutFrame entra no bloco "Criador de Janelas", que gera uma janela para expor a foto. Após isso, o bloco "Enquadrador de Imagem" coloca a foto dentro da janela criada. Em seguida, o bloco "Tempo de Exibição" controlará a duração da apresentação da imagem. Por fim, a imagem será impressa na tela por tempo em milissegundos, definido em "Tempo de Exibição". Esses três blocos representam, respectivamente, as seguintes linhas de código da Tabela 2.8: `cv::namedWindow( "IMAGEM", 0 ), imshow( "IMAGEM", auxCV ) e cv::waitKey(20).`

Os blocos "Contador de Fotos" e "Tempo de Execução" não interagem diretamente com CSC0, no entanto entregam ao usuário dados relevantes sobre o funcionamento do sistema.

"Contador de Fotos" salva no arquivo "Total de Fotos.txt" o número de imagens CSC0 que o programa processou desde que o botão "*Viewfinder*" foi acionado até o acionamento do botão "Parar" na interface de comando do usuário. A interface gráfica e os botões citados podem ser vistos na Fig.3.8.

Figura 3.8: Interface do usuário.

Quando clicamos no botão "*Viewfinder*", o decodificador entra em ação, fazendo o processamento das figuras capturadas, conforme já esclarecemos nas Seções anteriores, e mostrando, em uma nova janela que foi aberta pelo programa, cada fotografia capturada pelo chip. Esse processamento pode ser interrompido em qualquer instante, bastando para isso clicar no botão "Parar" posicionado ao lado de "*Viewfinder*". No antigo projeto, ao acionarmos "*Viewfinder*", o decodificador desenvolvido em MATLAB era chamado para fazer o processamento. Portanto ficávamos com dois programas funcionando paralelamente, deixando o sistema antigo mais lento e posteriormente causando falhas de travamento nos programas. Essas falhas serão caracterizadas no Capítulo 4. No atual projeto, não foi necessário alterar os comandos da interface do usuário.

O bloco “Tempo de Execução” informa, através do arquivo “Tempo Médio de Execução.txt”, o tempo médio gasto para se processar todas as amostras, desde que o botão “ViewFinder” foi acionado até o acionamento do botão “Parar” na interface de comando do usuário. Este tempo é importante para descobrirmos a velocidade de processamento do novo decodificador e sabermos quantas fotos por minuto são tiradas pelo chip.

O código do decodificador atual pode ser visto no Apêndice C.

A seguir, serão descritas as funções auxiliares que colaboram para o funcionamento do código principal do decodificador.

A função dec2binEscalar recebe dois parâmetros de entrada: “decimal” (entre 0 e 9) e “n” (2 ou 3). Quando “n” é dois, o valor retornado é um binário de dois dígitos, e quando é três, o valor retornado é um binário de três dígitos. O valor binário retornado é armazenado numa matriz. Um exemplo dessa função pode ser visto na Tab. 3.1.

Tabela 3.1– Aplicação da função dec2binEscalar (a,n)

```
double a; //declara um variavel do tipo double
int n; //declara um variavel do tipo int
a = 3.0;
n = 2;
Mat<double> X; //declara uma matriz X do tipo double
X = dec2binEscalar(a,2); //X recebe o valor de 'a' convertido
//em binário de 2 digitos
/*A matriz resultante sera:
  X = [1 1]
*/
```

Essa função precisou ser criada para uso dentro da função auxiliar dec2binVetor, que recebe um vetor “V” contendo valores decimais e retorna valores binários de dois ou três dígitos, semelhante ao que é feito na função dec2binEscalar.

Esses valores binários serão armazenados em uma matriz, um dígito por coluna. Uma aplicação dessa função pode ser vista na Tab. 3.2.

Tabela 3.2– Aplicação da função dec2binVetor (V,n)

```
int n; //declara um variavel do tipo int
n = 3;
Mat<double>X; //declara uma matriz X do tipo double
Row<double> V; //declara um vetor V do tipo double
V << 5.0 << endr; //inicializa V como o valor 5;
X = dec2binVetor(V,3); //X recebe o valor de 'V' convertido
                       //em binário de 3 digitos
/*A matriz resultante sera:
  X = [1 0 1]
*/
```

Os códigos de dec2binEscalar e dec2binVetor estão nas Seções C.1 e C.2.

A função dec2binVetor é usada para a implementação da função cad2index, que recebe como entrada uma matriz e retorna um vetor-linha com 64 posições. No sistema de quantização vetorial empregado no imageador, os dados de entrada são vetores reais com quatro componentes. Quando o imageador atribui um índice discreto (de 1 a 128) a um vetor de entrada, este índice discreto é representado através de uma notação binária específica ao trabalho de pesquisa que está sendo conduzido, que é chamada de notação *Cadence*, em referência às simulações dos diagramas esquemáticos dos circuitos que implementam a quantização vetorial. A função cad2index é necessária para realizar o mapeamento inverso, ou seja, da notação binária *Cadence* para a representação discreta original (de 1 a 128). O índice encontrado através do mapeamento inverso mencionado indica a linha do dicionário do VQ que deve ser utilizada na decodificação. Esse dicionário possui 128 linhas e quatro colunas, ou seja, 128 possíveis vetores para a reconstrução do vetor das quatro componentes de maior energia do bloco de pixels. Como é conveniente que 64 blocos sejam decodificados de uma só vez, porque é o número total de blocos que há em uma imagem com 32 × 32 pixels, então a função cad2index admite como entrada uma matriz binária com sete linhas e 64 colunas. O código dessa função é dado na Seção C.3.

A função gray2term recebe uma entrada binária em código Gray e retorna um valor binário do tipo termômetro, em que o número de elementos iguais a 1 é equivalente a um inteiro decimal. A função gray2term tem utilidade semelhante à de cad2index e uma aplicação da mesma pode ser vista na Tabela 3.3. O código de gray2term pode ser visto na Seção C.4. Utilizando essa função, encontramos o índice do

dicionário do DPCM. Acessando a coluna do dicionário referente a esse índice, encontramos o valor que deve ser utilizado para reconstruir o erro do DPCM e, a partir desse erro, o valor médio do bloco atual.

Tabela 3.3– Aplicação da função gray2term (B)

```
Mat<double> B; //cria matriz B de elementos double
B << 0.0 << 0.0 << 0.0 <<endr; //inicialização da matriz 1x3
Mat<double> R; //cria matriz R de elementos double
R = gray2term(B);
/*a matriz R sera
  R = [3]
*/
```

A função conv2 recebe duas matrizes e retorna a convolução bidimensional entre elas. Ela é necessária para o sistema proposto, mas não existe na biblioteca Armadillo. Para implementarmos conv2 partimos do seguinte produto de elementos de matrizes: matriResultante[m,n] = matrizA[i,j] x matrizB[m-i,n-j]. Como índices de linhas, temos m e i, como índices de colunas, temos n e j. m representa o número de linhas de matrizA mais o número de linhas de matrizB menos um, e n é o número de colunas de matrizA mais o número de colunas de matrizB menos um. Nós calculamos esse produto através de quatro laços *for*. O laço *for* mais externo percorre todas as linhas da matriz resultante. O segundo percorre todas as colunas da matriz resultante. O terceiro percorre todas as linhas do produto de elementos das matrizes A e B. E o laço mais interno percorre todas as colunas dos produtos de elementos das matrizes A e B. O produto somente será calculado quando estivermos dentro dos limites de linhas e colunas das matrizes A, B e resultante. Uma aplicação dessa função pode ser vista na Tabela 3.4. O código da função conv2 é dado na Seção C.5. A convolução é utilizada para filtrar a imagem final, resultante da decodificação, com um filtro passa-baixas. Essa técnica torna a imagem mais agradável subjetivamente.

Tabela 3.4– Aplicação da função conv2 (H1,H2)

```
Mat<double> H1; //cria matriz H1 de elementos double
H1 << 1.0 << 2.0 << endr //inicialização da matriz 2x2
   << 4.0 << 5.0 << endr;
Mat<double> H2; //cria matriz H2 de elementos double
H2 << 3.0  << 12.0 << endr //inicialização da matriz 2x2
   << 33.0 << 1.0  << endr;
Mat<double> R; //cria matriz R de elementos double
R = conv2(H1,H2);
/*a matriz R sera
  R = [3    18    24
       45   130   62
       132  169    5]
*/
```

3.2 – Construção de CSC0

O sistema novo não faz mais uso do MATLAB, e fica responsável tanto pela interface do usuário, quanto pelo processamento de imagens (decodificador), o que elimina a necessidade de ler e escrever em um arquivo para realizar a comunicação entre esses módulos, atividade que consome tempo considerável de execução.

Para construir a variável CSC0 de 1056 bits, foi implementada uma rotina muito parecida com o método “escrever” do antigo sistema. Para facilitar a explicação, apresentaremos o código da rotinha de construção de CSC0 na Tabela 3.5.

Tabela 3.5– Construção da variável binária CSC0

```
//Escreve o conteudo do buffer em um arquivo texto
for(j=0;j<17;j++)
{
//Converte Buffer[j] para uma string que contem cada bit separado por
//‘, ‘. Em seguida, escreve no arquivo.
umat CSC0_aux=zeros<umat>(1,4);
int mascara = 0x01;
int i = 0;
int max = 8;
//Desse forma, o bit menos significativo será o 1o impresso
//Como 132 bits sao lidos, o ultimo byte nao sera completamente
//preenchido.
//Para esse byte so os 4 primeiros bits que importam.
if(j == 16){
max = 4;
}
else{
CSC0_aux = join_rows(CSC0_aux,CSC0_aux);
}
while(i < max)
{
if((Buffer[j] & mascara) == mascara){
CSC0_aux(i) = 1;
}
else{
CSC0_aux(i) = 0;
}
mascara *= 2;
i++;
}//END***while(i < max)
if(CSC0.n_elem == 1){
CSC0 = zeros<umat>(1,8);
CSC0 = CSC0_aux;
}
else{
CSC0 = join_rows(CSC0,CSC0_aux);
}
}//END***for(j=0;j<17;j++)
```

Semelhante à explicação da rotina de tratamento de entrada binária da Seção 2.3, aqui também temos um loop que fará 17 iterações, sendo que cada iteração é a conversão de um char em oito bits, a menos da última iteração, que fará a conversão de um char em quatro bits. Na primeira iteração, a variável CSC0, que é uma matriz do Armadillo, não terá nenhum elemento, e por isso será inicializada com oito valores zeros e depois receberá o conteúdo do primeiro char armazenado na variável CSC0_aux, que é o trecho de código representado pelas linhas: “CSC0 = zeros<umat>(1,8)” e “CSC0 = CSC0_aux”. A partir da segunda iteração até a décima sexta, teremos apenas uma concatenação dos bits, a cada oito, em CSC0. Na última

iteração, uniremos os últimos quatro bits à CSC0, totalizando 1056 valores binários armazenados em CSC0.

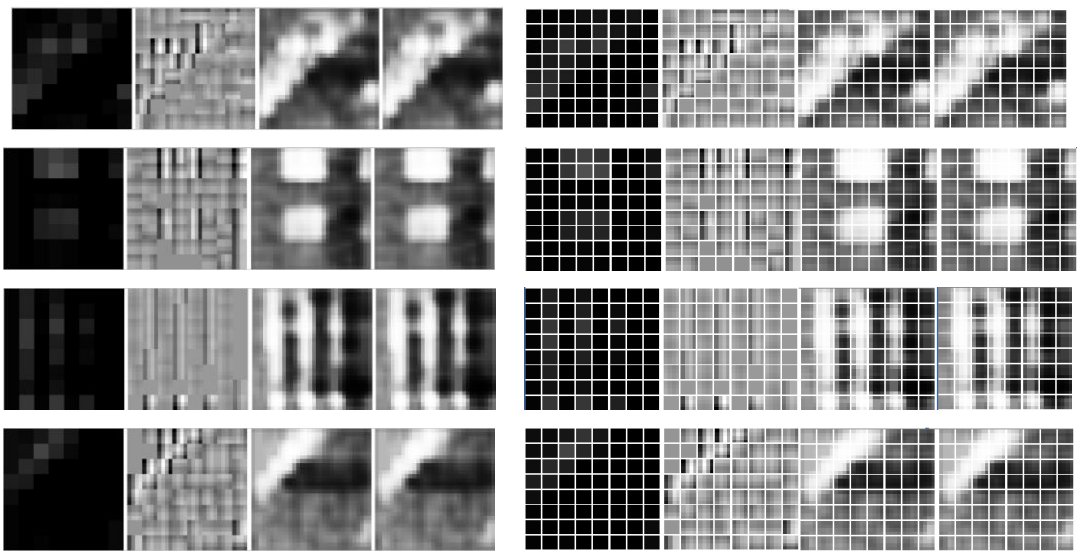
Capítulo 4

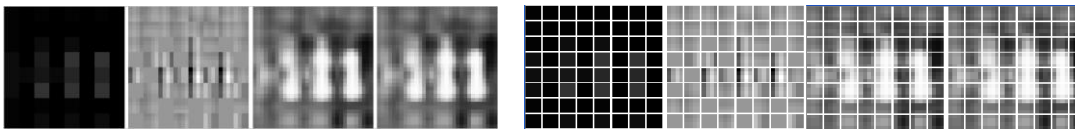
Resultados

Serão apresentados aqui os resultados das comparações realizadas entre os dois sistemas projetados. Na Seção 4.1, é verificado que as fotos obtidas através do sistema novo são iguais às que eram obtidas com o sistema antigo. Na Seção 4.2, são indicadas as diferenças nas velocidades de execução de cada sistema. Na Seção 4.3, veremos problemas que apareciam no sistema antigo durante a sua execução e não mais foram vistos no sistema novo.

4.1 – Comparação de Fotos Obtidas pelos Sistemas Antigo e Atual

A Fig. 4.1 apresenta algumas fotos processadas e apresentadas na tela do PC a partir do sistema antigo, à esquerda. Na parte da direita são mostradas as mesmas fotos, mas processadas e apresentadas pelo sistema atual.





(a) **(b)**

Figura 4.1: (a) imagens do decodificador antigo, (b) imagens geradas pelo decodificador novo.

Da esquerda para direita, Fig. 4.1 (a) e (b), temos os resultados da decodificação DPCM, VQ, média aritmética de 458 imagens e imagem final. Foram comparadas as matrizes que originaram os resultados das figuras acima no MATLAB, fazendo-se a diferença entre elas: matriz_DPCM(a) - matriz_DPCM(b), matriz_VQ(a) - matriz_VQ(b), matriz_MÉDIA(a) - matriz_MÉDIA(b) e matriz_FINAL(a) - matriz_FINAL(b). Foi observado um erro de $1{,}0 \times 10^{-4}$ nas diferenças entre as matrizes VQ, média e final; e um erro de $1{,}0 \times 10^{-16}$ na diferença das matriz DPCM. Visualmente esses erros não são perceptíveis e a única diferença que podemos observar é a existência da grade branca construída pelo código em C++, Fig. 4.1 (b), que divide as imagens em blocos de 4x4 pixels e tem a função de auxiliar o usuário na percepção do efeito da compressão em cada bloco.

4.2 – Comparação de Taxas de Captura Obtidas pelos Sistemas

Para o cálculo da taxa de captura alcançada pelo sistema novo, foi criada uma rotina de instruções , que consiste do código de marcação de tempo reproduzido na Tab. 4.1.

Tabela 4.1 – Código C++ para avaliação da taxa de captura de quadros.

```
//include para aferir tempo de execução
#include <time.h>
//include para manipular matrizes
#include "armadillo"
//variáveis para aferir tempo de execução
mat TE; //matriz para salvar o tempo de execução de uma foto
mat TE_total= zeros<mat>(1,1); //tempo de várias fotos
clock_t start, finish; //contadores de tempo
double iteracao=0; //inicializa contador de fotos
 //Início do teste de tempo de execução
start = clock(); //início da contagem do tempo

//Fim do Teste de Tempo de Execução
finish = clock(); //fim da contagem de tempo
TE = ((double)(finish - start))/(CLOCKS_PER_SEC ); //cálculo do tempo
                                                   //de execução de
                                                   //uma foto
TE_total = TE_total + TE; //cálculo do tempo de fotos feitas até o
                          //momento
iteracao++; //contagem de fotos
iterator = iteracao; //variável que guarda o número de fotos feitas
TE_total = TE_total/iteracao; //cálculo do tempo médio de cada foto
TE_total.save("Tempo Médio de Execução.txt", raw_ascii); //salva o
                                                         //Tempo de
                                                         //Execução
                                                         //Médio em
                                                         //segundos
//FIM
```

Através desse código, é possível salvar em arquivos de texto o número de fotos ("Número de Fotos.txt") e o tempo de execução médio ("Tempo Médio de Execução.txt"). Considerando um PC Intel Core 2 Duo com 2 GB RAM, foi obtido um tempo médio, após 458 fotos feitas, de 0,19 s, que corresponde a uma taxa de captura de 5,1 Hz, ou seja 5 fotos/s.

Para o cálculo do tempo no sistema antigo, foram utilizadas duas funções nativas do MATLAB: as funções "tic" (tempo inicial) e "toc" (tempo final). A função "tic" foi colocada na linha anterior ao recebimento de EntradaBinaria.txt e a função "toc" foi colocada na linha seguinte ao comando de impressão na tela, imshow(), da foto processada. Foram feitas 458 fotos, no mesmo PC com processador Core 2 Duo, com tempo total de 441,83 s, que corresponde a um tempo médio de 0,96 s e uma taxa de captura de 1,05 Hz, ou seja, 1 foto/s.

Para tentar melhorar a taxa de captura, o sistema atual foi executado em um PC com processador Intel Core i5 e com 6 GB de RAM, e alcançou uma taxa de captura em torno de 10 Hz. O sistema antigo não foi testado nesse PC.

4.3 – Ausência de Erros de Travamento no Sistema Atual

O sistema antigo apresentava duas falhas. Ambas causam término do programa antes do acionamento do botão "Parar", em circunstâncias que serão apresentadas nas Seções 4.3.1 e 4.3.2. No sistema atual, não foram observados esses problemas ou quaisquer outras questões de mau funcionamento. Uma vez que juntamos o decodificador e o programa da interface do usuário em um mesmo código, não temos mais dois programas acessando um mesmo arquivo (EntradaBinaria.txt). Como os problemas de travamento eram provenientes desse fato, o novo sistema soluciona esses problemas.

A utilização de outros programas simultaneamente à execução do sistema antigo aumenta a probabilidade dos dois tipos de falha.

4.3.1 – Falhas

Apresentaremos nesta seção as duas falhas observadas no sistema antigo durante a sua execução e explicaremos seus motivos, além de apresentarmos as mensagens de erros correspondentes produzidas.

Na primeira falha, os programas MATLAB e C++ tentam abrir o arquivo EntradaBinaria.txt simultaneamente: o MATLAB para ler e o C++ para escrever. Esse problema está associado a uma mensagem de erro gerada no MATLAB, que transcreveremos na página seguinte:

“ ??? Error using ==> load

File C:\Users\Gabriel\Desktop\Imageador\Códigos\CSC0.txt is currently in use by another process or thread you should be able to load once the other process or thread has released the file.

Error in using ==> Viewfinder 6 at 214

aux_X = load(CSC0.txt);”

Na segunda falha, o arquivo “CSC0.txt”, sem todos os 1056 bits, é lido pelo MATLAB. Aqui o C++ ainda não terminou de escrever o “CSC0.txt”. O MATLAB abriu o arquivo logo após o C++ ter escrito alguns bits e fechado o arquivo e antes que o C++ pudesse ter aberto novamente o CSC0 para escrever os bits restantes. A mensagem de erro gerada no MATLAB será transcrita abaixo:

“ ??? Error using ==> reshape

To RESHAPE the number of elements must not change.

Error in using ==> Viewfinder 6 at 226

aux_Y=reshape(aux_X' ,132,8);”

4.4 – Método de Leitura *Off-Line* das Fotos Realizadas

Após o botão “Parar” ter sido acionado na interface do usuário, o programa salvará num arquivo chamado “fotos.txt” que contém todas as imagens capturadas pela câmera. Essas amostras podem ser usadas para algum experimento posterior, a critério do usuário.

Para ler, no MATLAB, imagens previamente capturadas pela câmera, sugerimos uma rotina semelhante ao pseudo-código da Tabela 4.2.

Tabela 4.2 – Leitura de dados off-line no MATLAB

```
DadosBinariosCapturados = load('fotos.txt');
TotalFotos = size(DadosBinariosCapturados,1);
for ContadorFoto = 1:TotalFotos,
      % FotoAtual = load('CSC0.txt');
      FotoAtual = DadosBinariosCapturados(ContadorFoto,:);
      X = Decodificacao(FotoAtual);
      MostraImagem(X);
end;
```

Capítulo 5

Conclusão

O presente trabalho conseguiu alcançar as seguintes metas propostas:

- Operar uma câmera CMOS, que realiza compressão de imagens no plano focal, em taxas de vídeo acima de 1 Hz.
- Descobrir qual é a maior taxa de vídeo alcançável.
- Reduzir de dois para um o número de programas que representam o atual sistema.
- Corrigir problemas de travamento durante a execução do software, provenientes do programa antigo.

Ao ser executado em um *notebook* DELL com processador Intel Core 2 Duo e 2GB de memória RAM, o sistema novo atinge uma taxa de captura de quadros em torno de 5 Hz, enquanto que o sistema antigo alcança aproximadamente 1 Hz no mesmo computador. Como já explicado nos Capítulos anteriores, este sistema funciona com um único programa escrito em C/C++. Com relação aos problemas de travamento, eles não ocorrem mais nesta versão.

Com este projeto foi possível aprender e adquirir novos conhecimentos, que destacamos abaixo:

- MATLAB: Manipulação básica de matrizes através de declaração, operações aritméticas básicas(soma, subtração, divisão e multiplicação), apresentação de imagens na tela, e elaboração de estruturas de repetição com laços *for* e *if*.

- Interface de desenvolvimento integrada(VC++): Compreensão de como criar soluções usando essa IDE, trabalhar com mais de uma linguagem de programação como C e C++ e compilá-las na mesma solução, instalação de bibliotecas não disponíveis no VC++ (Armadillo e OpenCV).

- Biblioteca de manipulação de matrizes (Armadillo): Trabalhar com matrizes realizando a sua transposição, inversão, calculando determinante e outros casos vistos no Capítulo 2.

- Biblioteca gráfica (OpenCV): Fazer a impressão na tela de figuras geradas por matrizes.

- Gerenciamento de Projeto: Construção de lista de metas a serem alcançadas dentro dos prazos estipulados, observando a prioridade de cada meta. Entendimento que, em geral, ocorrem imprevistos nos desenvolvimentos dos projetos e devemos manter alguma flexibilidade de tempo para atingir os objetivos desejados.

- Abordagem de problemas: qualquer projeto pode ser considerado como a resolução de algum problema e devemos começar a solucionar esse a partir das hipóteses mais simples percebidas pela equipe técnica envolvida. Lembramos que problemas complexos podem ser divididos em problemas mais simples, técnica popularmente conhecida como “Dividir para conquistar”.

Este projeto pode ser usado por qualquer pessoa, mesmo aquelas que não possuem conhecimento na área de processamento de sinais. No entanto, seria proveitoso ao usuário que este tivesse noções das técnicas de compressão de imagens mencionadas no Capítulo 2, Quantização Vetorial e DPCM. Isso facilitaria o entendimento de como são geradas as imagens.

O novo sistema, além dos objetivos alcançados, tem sua importância fundamentada na simplificação de uso do software, pois é mais fácil trabalhar com um programa do que com dois. Outra nova característica importante é a possibilidade de realizar um ajuste focal no equipamento com maior precisão, porque, à medida que alteramos o foco da câmera, seus resultados aparecem na tela instantaneamente.

Como sugestões de trabalhos futuros, além das implementações mencionadas no parágrafo anterior, podemos recomendar a operação do sistema em taxas de vídeo mais elevadas, por exemplo acima de 100 Hz. Mencionamos isso, pois, devido ao tempo de resposta do chip CMOS, aproximadamente 8 ms, imaginamos que a possibilidade de apresentar até 125 quadros/s é factível.

Seria interessante também, que as modificações feitas na rotina chamada pelo botão “Viewfinder” da interface do usuário, fossem implementadas nos outros botões da interface, como os botões: “Ativar bloco principal”, “Ativar pixel independente”, “Ativar blocos de teste” e “Ativar pixels de teste”. Dessa maneira, teríamos otimizado todas as rotinas chamadas pela interface. Se as imagens geradas pelo próprio sistema fossem embutidas na interface, isso tornaria mais agradável a interação do usuário com o programa, porque ele poderia operar comandos e observar resultados numa mesma janela.

Bibliografia

[1] F. D. V. R. de Oliveira, “Circuito Integrado para Compressão de Imagens no Plano Focal utilizando Quantização Vetorial e DPCM”, Projeto Final de Graduação, DEL/EPOLI/UFRJ, janeiro de 2012.

[2] F. D. V. R. Oliveira, H. L. Haas, J. G. R. C. Gomes e A. Petraglia. CMOS Imager with Focal-Plane Analog Image Compression Combining DPCM and VQ. IEEE Trans. Circuits and Systems Part I: Regular Papers, volume PP, no. 99, pp. 1-14, publicado online em 11 de março de 2013. DOI: 11.1109/TCSI.2012.2226505.

[3] F. D. V. R. Oliveira, H. L. Haas, J. G. R. C. Gomes, and A. Petraglia, “A CMOS imaging system featuring focal-plane image compression based on DPCM and VQ,” in Proc. IEEE Latin Amer. Symp. Circuits Syst., Playa del Carmen, Mexico, 2012, pp. 1–4.

[4] F. D. V. R. Oliveira, H. L. Haas, J. G. R. C. Gomes, and A. Petraglia, "Current-mode analog integrated circuit for focal-plane image compression," SBCCI, 2012, pp. 1-6.

[5] F.D.V.R.Oliveira, H. L.Haas, J. G. R. C. Gomes, andA.Petraglia, “A circuit for focal-plane image compression using vector quantization,” in Proc. IEEE Int. Symp. Signals, Circuits, Systems, Iasi, Romania, Jun. 2011, pp. 181–184.

[6] Hugo L. Haas, “Projeto de Circuitos para Compressão de Imagens no Plano Focal de Câmeras CMOS”, Dissertação de Mestrado, DEL/EPOLI/UFRJ, janeiro de 2012.

[7] http://msdn.microsoft.com/en-us/library/ms173265(v=vs.90).aspx (último acesso em 07/02/2013)

[8] http://msdn.microsoft.com/pt-br/library/8bs2ecf4(v=vs.90).aspx (último acesso em 15/10/2012)

[9] http://msdn.microsoft.com/pt-br/library/c5tkafs1(v=vs.90).aspx (último acesso em 15/10/2012)

[10] http://msdn.microsoft.com/pt-br/library/ht8ecch6(v=vs.90).aspx(último acesso em 15/10/2012)

[11] http://arma.sourceforge.net/docs.html#syntax (último acesso em 13/01/2013)

[12] http://arma.sourceforge.net/docs.html (último acesso em 13/01/2013)

[13] http://opencv.org/ (último acesso em 07/02/2013)

[14] http://en.wikipedia.org/wiki/Kronecker_product (último acesso em 07/02/2013)

Apêndice A

Codificador e Decodificador (MATLAB)

A seguir, é fornecido código-fonte para codificação e decodificação de imagens 32 × 32. A partir do comando “if leitura_sensor”, seis linhas após o comentário “Imager” na página seguinte, o código tem relevância particular para este projeto.

```
close all;
clear all;
EncoderTeorico=1;

 % Constantes %

H = [  2     1    -1    -2     2     1    -1    -2     2     1    -1    -2     2     1    -1    -2;
       2     2     2     2     1     1     1     1    -1    -1    -1    -1    -2    -2    -2    -2;
       1    -1    -1     1     1    -1    -1     1     1    -1    -1     1     1    -1    -1     1;
       1     1     1     1    -1    -1    -1    -1    -1    -1    -1    -1     1     1     1     1];
w = [ -0.41998907763678  -0.13141444169639  -0.71812032458305   0.53911280693391 ;  ...
       0.23365707958990  -0.42692361948093   0.48628829725329   0.72571639380775 ;  ...
       0.79742076822793   0.41786811040985  -0.36661165734511   0.23473869174029 ;  ...
      -0.36487485830363   0.79110853234270   0.33678299255441   0.35719860548241 ];
W = (round (W*2))/2; T = [0 0.05 0.1 0.2 0.3 0.4 0.6 -0.15 0 0.1 0.05 0];
T0 = [0.0125    0.0375    0.0750    0.1250    0.1875    0.2750    0.4000]';
C0 = [0.0063    0.0250    0.0563    0.1000    0.1500    0.2250    0.3250    0.4688]';
CDPCM=C0;
F = imread('lena_olho_original.bmp');
F = imresize(F,[32 32]);
G=rgb2gray(F); clear F; % [P01a] Imagens Teóricas ou Imagens Experimentais


if EncoderTeorico,


%%%%%%%%%%
%        %
% Encoder %
%        %
%%%%%%%%%%

i=1;
X=[];
while (i<size(G,1));
k=1; while (k<size(G,2));
X=[X [G(i,k:k+3) G(i+1,k:k+3) G(i+2,k:k+3) G(i+3,k:k+3)]'];
k=k+4;
end;
i=i+4;
end;
X=double(X)/255; % [P02a] Faixa Dinâmica de Entrada = [0,1] ou [a,b]
H(1:2,:)=H(1:2,:)/4/1.5811; %
H(3:4,:)=H(3:4,:)/4;        % Power Adjustment
P=H*X;                      %
P(1:2,:)=P(1:2,:)*1.5811/2; %
S=zeros(4,size(P,2));
for i=1:size(P,2), S(1,i)=(P(1,i)>=0);
S(2,i)=(P(2,i)>=0); S(3,i)=(P(3,i)>=0);
S(4,i)=(P(4,i)>=0);
end;
F = fliplr(W)'*abs(P);
Th = zeros(12,size(F,2));
for i=1:size(F,2), Th(1,i)=(F(1,i)-T(1))>=0;
Th(2,i)=(F(1,i)-T(2))>=0; Th(3,i)=(F(1,i)-T(3))>=0; Th(4,i)=(F(1,i)-T(4))>=0;
Th(5,i)=(F(1,i)-T(5))>=0; Th(6,i)=(F(1,i)-T(6))>=0; Th(7,i)=(F(1,i)-T(7))>=0;
Th(8,i)=(F(2,i)-T(8))>=0; Th(9,i)=(F(2,i)-T(9))>=0; Th(10,i)=(F(2,i)-T(10))>=0;
Th(11,i)=(F(3,i)-T(11))>=0;
Th(12,i)=(F(4,i)-T(12))>=0;
```

```
end;
B = fliplr(dec2bin(sum(Th(1:7,:),1),3)=='1')';
B = [B; fliplr(dec2bin(sum(Th(8:10,:),1),2)=='1')'];
B = [B; Th(11,:); Th(12,:)];
IVQ = fliplr(2.^((1:7)-1))*B+1;
BM = mat2cad(B')'; % Código Cadence (fix transposes) % [P03a] Códigos Hugo-Cadence

% DPCM%

Mu=zeros(1,size(X,2));
Mu=mean(X,1);
MuHat=zeros(1,size(X,2)+1);
D=zeros(4,size(X,2));
index = zeros (1,size(X,2));
for i=1:size(X,2);
    if mod(i-1,size(G,2)/4)==0, Dif=Mu(i);
else Dif=Mu(i)-MuHat(i);
end;
    D(1,i)=(Dif>=0);
Dif=abs(Dif);
index(i) = sum(Dif>T0)+1;
D(2:4,i)=Term2Gray(index(i));
    if mod(i-1,size(G,2)/4)==0, MuHat(i+1)= C0(index(i))*(D(1,i)*2-1);
else MuHat(i+1)= MuHat(i)+C0(index(i))*(D(1,i)*2-1);
end;
end;
B1=BM;%
D1=D; % save MatrizesTemp.mat D1 S1 B1; % [P04a] Dados Codificados para o Decoder
S1=S; %

else

%%%%%%%%%%
%        %
% Imager %
%        %
%%%%%%%%%%


s=dir('..\TXT\*.mat');
s(1).name='110919Lena10B.mat';
contador_imagem=1;
load(strcat('..\TXT\',s(contador_imagem).name));
contagem_rodada=150;
aux_X=BKP_TXT(contagem_rodada).TXT;
B1=[];
S1=[];
D1=[];
DC1=zeros(4,8,3);
leitura_sensor=0;
if leitura_sensor, A=0;
save Flag.txt A -ascii;
pause(.2);
aux_X=load('CSC0.txt');                                          % O pseoudo código da
A=1; save Flag.txt A -ascii;                                     % Tab. 4.2 foi baseado
BKP_TXT(contagem_rodada).TXT=aux_X;                              % trecho de cinco li-
else aux_X=BKP_TXT(contagem_rodada).TXT;                         % nhas de código.
if length(aux_X)<1056, aux_X=[aux_X zeros(1,1056-length(aux_X))]; %
end;
end;
aux_Y=reshape(aux_X',132,8);
aux_Y=(flipud(aux_Y))';
for i=1:8,
    for j=1:8, aux_o = 15*(j-1)+1;
S1 = [S1 flipud(aux_Y(i,aux_o:aux_o+3)')];
D1 = [D1 flipud(aux_Y(i,aux_o+4:aux_o+7)')];
B1 = [B1 flipud(aux_Y(i,aux_o+8:aux_o+14)')];
end;
aux_o = 15*(j-1)+1;
DC1(:,i,3) = flipud(aux_Y(i,aux_o:aux_o+3)');
DC1(:,i,2) = flipud(aux_Y(i,aux_o+4:aux_o+7)');
DC1(:,i,1) = flipud(aux_Y(i,aux_o+8:aux_o+11)');
end;
S1=1-S1;
end;

%%%%%%%%%%
%        %
% Decoder %
%        %
%%%%%%%%%%

ivq=cad2index(B1); %
Sb=S1;             % [P05a] Sinais Hugo-Cadence são invertidos
l=1;               %
fim=1
pause;
```

```
CDPCM=C0; % [PO2b] Faixa Dinâmica DPCM = Original ou Aumentada?
L=8; % size(G,2)/4;
for i=1:64,
    index(i)=Gray2Term(D1(2:4,i)');
    if mod(i-1,L)==0
        MuHat(i+1)=CDPCM(index(i))*(D1(1,i)*2-1);
    else
        MuHat(i+1)=MuHat(i)+CDPCM(index(i))*(D1(1,i)*2-1);
    end;
end; % reshape(MuHat(2:65),8,8)'
Im=zeros(32,32); % Im=zeros(size(G)); % reconstructed DPCM image
for i=1:4:size(Im,1)
    for k=1:4:size(Im,2)
        Im(i:i+3,k:k+3) = MuHat(((i-1)*size(Im,2)/4+(k-1))/4+2);
    end;
end;

%VQ%

load CodebookPrincipal.mat -ascii; % [PO6a] Alterações do Dicionário
C=CodebookPrincipal;               %
Phatb=zeros(4,64);
for i=1:size(ivq,2), PhatO(:,i)=C(:,ivq(i));
end;
for i=1:size(PhatO,1), Phatb(i,:)= PhatO(i,:).*(Sb(i,:)*2-1);
end;
Phatb(1:2,:)=Phatb(1:2,:)/1.5811*2; % Reverse Power Adjustment
Xhatb=H'*Phatb; Yb=zeros(size(Im));
for i=1:64;
    k=floor((i-1)/L);
    j=i-k*L-1;
    Yb((k*4)+1:(k*4)+4,(j*4)+1:(j*4)+4)= [Xhatb(1:4,i) Xhatb(5:8,i) Xhatb(9:12,i)
Xhatb(13:16,i)]';
end;
if not(EncoderTeorico), Im=flipud(Im);
Yb=flipud(Yb);
end;
ImagemFinalb=Im+Yb; %
ab=min(min(Yb));    % [PO7a] Sem Filtragem
bb=max(max(Yb));    %
f=[1 1 1;1 1 1;1 1 1]/9;
ImagemFinalb=conv2(ImagemFinalb,f);
ImagemFinalb=ImagemFinalb(2:end-1,2:end-1);
m1=min(min(ImagemFinalb));
m2=max(max(ImagemFinalb));
ImagemFinalb=(ImagemFinalb-m1)/(m2-m1);
ImagemFinalb=0.5+0.5*tanh(10*(ImagemFinalb-mub));
figure(4); hist(reshape(ImagemFinalb,1024,1),50);
imagem_media_VQ = kron((Yb-ab)/(bb-ab),ones(5,5)); % [PO8a] Faixa Dinâmica de Saída
imagem_media = kron(ImagemFinalb,ones(5,5)); % [PO9a] Formato de Display
if not(isempty(G)), imagem_original = kron(double(G)/255,ones(5,5));
else imagem_original = zeros(167,167);
end;
separa = imagem_media_VQ;
separa2 = imagem_media;
separa3 = imagem_original;
separa4 = kron(Im,ones(5,5));
SP = zeros(size(separa,1)+7,size(separa,2)+7,3);
SP(:,:,2) = ones(size(separa,1)+7,size(separa,2)+7);
SP(:,:,1) = ones(size(separa,1)+7,size(separa,2)+7);
SP(:,:,3) = ones(size(separa,1)+7,size(separa,2)+7);
SP2 = SP;
SP3 = SP;
SP4 = SP;
for i=1:8,
offi = (i-1)*(4*5+1)+1; offi1 = (i-1)*(4*5)+1;
for j=1:8,
offj = (j-1)*(4*5+1)+1; offj1 = (j-1)*(4*5)+1;
SP(offi:offi+4*5-1,offj:offj+4*5-1,1) = separa(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP(offi:offi+4*5-1,offj:offj+4*5-1,2) = separa(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP(offi:offi+4*5-1,offj:offj+4*5-1,3) = separa(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP2(offi:offi+4*5-1,offj:offj+4*5-1,1) = separa2(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP2(offi:offi+4*5-1,offj:offj+4*5-1,2) = separa2(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP2(offi:offi+4*5-1,offj:offj+4*5-1,3) = separa2(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP3(offi:offi+4*5-1,offj:offj+4*5-1,1) = separa3(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP3(offi:offi+4*5-1,offj:offj+4*5-1,2) = separa3(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP3(offi:offi+4*5-1,offj:offj+4*5-1,3) = separa3(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP4(offi:offi+4*5-1,offj:offj+4*5-1,1) = separa4(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP4(offi:offi+4*5-1,offj:offj+4*5-1,2) = separa4(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
SP4(offi:offi+4*5-1,offj:offj+4*5-1,3) = separa4(offi1:offi1+4*5-1,offj1:offj1+4*5-1);
end;
end;
figure(1);
imshow([ones(5,349,3);ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2
ones(size(SP,1),5,3) ; ones(5,349,3) ; ones(size(SP,1),5,3) SP4 ones(size(SP,1),5,3) SP
ones(size(SP,1),5,3) ; ones(5,349,3)]); % figure(2); imshow(SP); figure(3); imshow(SP3);
```

```
if not(isempty(G)),
PSNR=10*log10(((1-1/256)^2)/mean(mean((ImagemFinalb-double(G)/255).^2)))%[P10A]
                                                                        %Cálculos
                                                                        %Corretos de
                                                                        %PSNR
end;
OutFrame = [ones(5,349,3) ; ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2
ones(size(SP,1),5,3) ; ones(5,349,3) ; ones(size(SP,1),5,3) SP4 ones(size(SP,1),5,3) SP
ones(size(SP,1),5,3) ; ones(5,349,3)];
OutFrame = OutFrame(6:344,6:344,:);
OutFrame = rgb2gray(uint8(255*OutFrame));
close all; imshow(OutFrame);
```

Apêndice B

Rotina de escrita de bits em arquivo de texto do sistema antigo

O código seguinte recebe dados do chip e faz uma conversão de char para bits e os reescreve em uma arquivo de texto.

```
void tratamentoArquivo::escrever(string nomeArq, char conteudo, int numByte)
{
	string binario;
	int mascara = 0x01;
	int i = 0;
	int max = 16;
	binario.resize(16);
	//Abrindo o arquivo e posicionando o cursor na última posicao.
	arquivo.open(nomeArq.c_str());
	arquivo.seekg (0, ios::end);
	//Desse forma, o bit menos significativo será o 1o impresso
	//Como 132 bits sao lidos, o ultimo byte nao sera completamente preenchido.
	//Para esse byte so os 4 primeiros bits que importam.
	if(numByte == 16)
		max = 8;

	while(i < max)
	{
		if((conteudo & mascara) == mascara)
			binario.at(i) = '1';
		else
			binario.at(i) = '0';
		mascara *= 2;
		i++;
		binario.at(i) = ',';
		i++;
	}

	//O conteudo passado pelo pic deve ser convertido para binario e dividido
	//para que fique no formato que o MATLAB lê

	{
		binario[i] = teste%2;
		teste = teste/2;
		i++;
		binario[i] = “;”;
		i++;
	}
	binario[i] = conteudo%2;
	i++;
	binario[i] = “;”;
	i++;
	binario[i] = conteudo/2;
	i++;
	binario[i] = “;”;
	i++;
	while(i<16)
	{
		binario[i] = “0”;
		i++;
		binario[i] = “;”;
		i++;
```

```
        }*/
        arquivo.write(binario.c_str(), strlen(binario.c_str()));
        arquivo.close();
}
```

Apêndice C

Novo decodificador

Este apêndice contém ainda mais cinco seções nas quais serão dados os detalhes de sub-funções utilizadas no código principal que está a seguir.

```
 //*****************************BEGIN***DECODER*****************************
////////////////////////////////VARIÁVEIS//BEGIN////////////////////////////////

 mat H, ivq, Sb, MuHat, Im, C,Phatb, Phat0, Xhatb, Yb;
 mat ImagemFinalb, f;
 mat W,C_aux;
 mat T;
 mat TE;//matriz para salvar o tempo de execução
 mat iterator;
 mat T0;
 mat C0;
 mat CDPCM;
 cube F;
 mat aux_X;
 mat A;
 mat aux_Y;
 cube DC1 = zeros<cube>(4,8,3);
 mat tudo1 = ones<mat>(4,64);
 int aux_o, i, j, L;
 double l, index, ab, bb;
 mat S1;
 mat D1;
 mat B1;
 double iteracao=0;
 clock_t start, finish;//variáveis para aferir tempo de execução

////////////////////////////////VARIÁVEIS//END/////////////////////////////////////

//Codebook Principal representado pela variável C
//tem dimensão 4x128
//Colunas 1 até 7

C_aux<<0<<0<<0<<0.0301<<0.0054<<0.0040<<0.0080
     <<0<<0<<0<<0.0068<<0.0072<<0.0107<<0.0063
     <<0<<0<<0<<0.3982<<0.0040<<0.0170<<0.0126
     <<0<<0<<0<<0.0102<<0.0066<<0.0072<<0.0479
 // Colunas 8 até 14
<<0.0043<<0.0228<<0.0152<<0.0233<<0.0180<<0<<0
<<0.0094<<0.0065<<0.0077<<0.0037<<0.0075<<0<<0
<<0.0341<<0.0091<<0.0293<<0.0225<<0.0708<<0<<0
<<0.0366<<0.0070<<0.0064<<0.0373<<0.0203<<0<<0
 // Colunas 15 até 21
<<0<<0<<0.2863<<0.2224<<0.3363<<0.2991<<0.0863
<<0<<0<<0.1191<<0.1585<<0.0626<<0.0788<<0.2350
<<0<<0<<0.0582<<0.1576<<0.1179<<0.3079<<0.0209
<<0<<0<<0.0231<<0.0216<<0.0767<<0.0588<<0.0730
  //Colunas 22 até 28
<<0.0503<<0.0803<<0.0436<<0.1641<<0.1254<<0.1622<<0.1126
<<0.2654<<0.1756<<0.2193<<0.1916<<0.2214<<0.1387<<0.1894
<<0.0939<<0.0398<<0.1595<<0.0342<<0.1345<<0.0723<<0.2053
<<0.0487<<0.2124<<0.1645<<0.0479<<0.0345<<0.1694<<0.1228
  //Colunas 29 até 35
<<0.0230<<0.0120<<0.0226<<0.0114<<0.1691<<0.1426<<0.1711
<<0.2798<<0.3101<<0.2039<<0.2620<<0.0182<<0.0468<<0.0127
<<0.0143<<0.0571<<0.0192<<0.1280<<0.0329<<0.0940<<0.0719
<<0.1029<<0.0551<<0.2743<<0.2140<<0.0080<<0.0042<<0.0283
  //Colunas 36 até 42
<<0.1496<<0.0272<<0.0175<<0.0249<<0.0161<<0.0922<<0.0709
<<0.0244<<0.0981<<0.1113<<0.0653<<0.0830<<0.0592<<0.0751
<<0.1777<<0.0110<<0.0445<<0.0198<<0.0814<<0.0223<<0.0730
<<0.0223<<0.0379<<0.0243<<0.1245<<0.0930<<0.0195<<0.0156
```

```
 // Colunas 43 até 49
<<0.0902<<0.0704<<0<<0<<0.0015<<0<<0.4509
<<0.0364<<0.0578<<0<<0<<0.0242<<0<<0.3044
<<0.0408<<0.1190<<0<<0<<0.0028<<0<<0.0708
<<0.0795<<0.0563<<0<<0<<0.3280<<0<<0.0449
  //Colunas 50 até 56
<<0.3517<<0.5419<<0.5534<<0.2010<<0.1454<<0.1960<<0.1331
<<0.3383<<0.1759<<0.1915<<0.4654<<0.5020<<0.3608<<0.4588
<<0.2518<<0.1846<<0.6505<<0.0334<<0.1598<<0.0836<<0.2782
<<0.0448<<0.1975<<0.1047<<0.1143<<0.0826<<0.3804<<0.3161
  //Colunas 57 até 63
<<0.2851<<0.2100<<0.2735<<0.2052<<0.0767<<0.0570<<0.0396
<<0.4111<<0.4702<<0.3236<<0.3969<<0.5508<<0.5659<<0.4168
<<0.0448<<0.2118<<0.1168<<0.2994<<0.0197<<0.1039<<0.0209
<<0.0839<<0.0583<<0.2814<<0.2386<<0.1584<<0.0864<<0.4740
  //Colunas 64 até 70
<<0.0138<<0.1325<<0<<0.1295<<0.1022<<0.0156<<0.0101
<<0.4423<<0.0035<<0<<0.0044<<0.0079<<0.0452<<0.0524
<<0.3312<<0.0488<<0<<0.0681<<0.1724<<0.0078<<0.0290
<<0.4772<<0.0037<<0<<0.0077<<0.0087<<0.0226<<0.0154
  //Colunas 71 até 77
<<0.0153<<0.0087<<0.0541<<0.0404<<0.0529<<0.0478<<0
<<0.0258<<0.0383<<0.0238<<0.0321<<0.0136<<0.0224<<0
<<0.0154<<0.0535<<0.0149<<0.0474<<0.0289<<0.0918<<0
<<0.0790<<0.0592<<0.0126<<0.0106<<0.0530<<0.0319<<0
  //Colunas 78 até 84
<<0<<0<<0<<0.3930<<0.2960<<0.4375<<0.3634
<<0<<0<<0<<0.1915<<0.2499<<0.1104<<0.1656
<<0<<0<<0<<0.0758<<0.2046<<0.1329<<0.3366
<<0<<0<<0<<0.0345<<0.0295<<0.1097<<0.0853
  //Colunas 85 até 91
<<0.1369<<0.0905<<0.1330<<0.0741<<0.2203<<0.1652<<0.2168
<<0.3381<<0.3784<<0.2603<<0.3253<<0.2886<<0.3330<<0.2160
<<0.0273<<0.1391<<0.0572<<0.2134<<0.0431<<0.1652<<0.0917
<<0.0944<<0.0693<<0.2674<<0.2154<<0.0699<<0.0527<<0.2356
  //Colunas 92 até 98
<<0.1537<<0.0461<<0.0301<<0.0342<<0.0301<<0.2172<<0.1731
<<0.2793<<0.3995<<0.4365<<0.3056<<0.3485<<0.0625<<0.0983
<<0.2585<<0.0181<<0.0839<<0.0240<<0.1432<<0.0459<<0.1223
<<0.1949<<0.1338<<0.0658<<0.3640<<0.2402<<0.0146<<0.0103
  //Colunas 99 até 105
<<0.2325<<0.2143<<0.0467<<0.0281<<0.0422<<0.0268<<0.1231
<<0.0321<<0.0510<<0.1578<<0.1802<<0.1122<<0.1440<<0.1130
<<0.0866<<0.2183<<0.0151<<0.0635<<0.0279<<0.1136<<0.0286
<<0.0580<<0.0337<<0.0557<<0.0351<<0.1667<<0.1267<<0.0312
  //Colunas 106 até 112
<<0.0964<<0.1207<<0.0902<<0.0076<<0.0041<<0.0088<<0
<<0.1334<<0.0762<<0.1096<<0.2045<<0.2436<<0.1359<<0
<<0.1008<<0.0552<<0.1541<<0.0083<<0.0308<<0.0119<<0
<<0.0230<<0.1233<<0.0879<<0.0846<<0.0321<<0.2259<<0
  //Colunas 113 até 119
<<0.5310<<0.3351<<0.7310<<0.4871<<0.2827<<0<<0.3150
<<0.5101<<0.7016<<0.2389<<0.4530<<0.6315<<0<<0.4200
<<0.0755<<0.5434<<0.2242<<0.5089<<0.0242<<0<<0
<<0.0843<<0.1333<<0.3460<<0.3720<<0.1561<<0<<0.4980
  //Colunas 120 até 128
<<0<<0.3603<<0<<0.3261<<0.2249<<0.0976<<0.1737<<0.0696<<0<<endr
<<0<<0.5686<<0<<0.5462<<0.5940<<0.7556<<0.7417<<0.6695<<0<<endr
<<0<<0.0928<<0<<0.2107<<0.3808<<0.0185<<0.1475<<0.0353<<0<<endr
<<0<<0.1664<<0<<0.2745<<0.3396<<0.2506<<0.0126<<0.7376<<0<<endr;
//Fim do CodeBook Principal

H<<0.3162<<0.1581<<-0.1581<<-0.3162<<0.3162<<0.1581<<-0.1581<<-0.3162<<0.3162<<0.1581
<<-0.1581<<-0.3162<<0.3162<<0.1581<<-0.1581<<-0.3162<<endr
<<0.3162<< 0.3162<< 0.3162<< 0.3162<< 0.1581<< 0.1581<< 0.1581<< 0.1581<<-0.1581<<-0.1581
<<-0.1581<<-0.1581<<-0.3162<<-0.3162<<-0.3162<<-0.3162<<endr
<<0.2500<<-0.2500<<-0.2500<< 0.2500<< 0.2500<<-0.2500<<-0.2500<< 0.2500<<0.2500<<-0.2500
<<-0.2500<< 0.2500<< 0.2500<<-0.2500<<-0.2500<< 0.2500<<endr
<<0.2500<< 0.2500<< 0.2500<< 0.2500<<-0.2500<<-0.2500<<-0.2500<<-0.2500<<-0.2500<<-0.2500
<<-0.2500<<-0.2500<< 0.2500<< 0.2500<< 0.2500<< 0.2500<<endr;

 ////////////////////////////////////CONSTANTES//BEGIN//////////////////////////////

W <<-0.41998907763678<<-0.13141444169639<<-0.71812032458305<<0.53911280693391<<endr
  <<0.23365707958990<<-0.42692361948093<<0.48628829725329<<0.72571639380775<<endr
  <<0.79742076822793<<0.41786811040985<<-0.36661165734511<<0.23473869174029<<endr
  <<-0.36487485830363<<0.79110853234270<<0.33678299255441<<0.35719860548241<<endr;
T <<0<<0.05<<0.1<<0.2<<0.3<<0.4<<0.6<<-0.15<<0<<0.1<<0.05<<0<<endr;
T0 <<0.0125<<0.0375<<0.0750<<0.1250<<0.1875<<0.2750<<0.4000<<endr;
C0 <<0.0063<<0.0250<<0.0563<<0.1000<<0.1500<<0.2250<<0.3250<<0.4688<<endr;
CDPCM = C0;
T0 = trans(T0);
C0 = trans(C0);

////////////////////////////////////CONSTANTES//END//////////////////////////////////
```

```
////////////////////////////////////IMAGER//BEGIN/////////////////////////////////////

aux_X = conv_to<mat>::from(CSC0); //conversão de “umat” para “mat”
//Cria banco de fotos
//Cada linha da matriz “fotos” representa uma foto
if (fotos.n_elem == 1){
  fotos = zeros<umat>(1,1056);
  fotos = CSC0;
}
else{
   fotos = join_cols(fotos,CSC0);
}
aux_Y = reshape(trans(aux_X), 132, 8);
aux_Y = trans(flipud(aux_Y));
for (i=0; i < 8; i++) {
   for (j=0; j < 8; j++) {
     aux_o = 15*(j);
     S1 = join_rows(S1,flipud(trans(aux_Y(i,span(aux_o,aux_o+3)))));
     D1 = join_rows(D1,flipud(trans(aux_Y(i,span(aux_o+4,aux_o+7)))));
     B1 = join_rows(B1,flipud(trans(aux_Y(i,span(aux_o+8,aux_o+14)))));
};
j=7;
aux_o = 15*(j);
DC1(span(), span(i), span(2)) = flipud(trans(aux_Y(i,span(aux_o,aux_o+3))));
DC1(span(), span(i), span(1)) = flipud(trans(aux_Y(i,span(aux_o+4,aux_o+7))));
DC1(span(), span(i), span(0)) = flipud(trans(aux_Y(i,span(aux_o+8,aux_o+11))));
};
S1 = tudo1 - S1;//testar a matriz

////////////////////////////////////IMAGER//END//////////////////////////////////

////////////////////////////////////DECODER//BEGIN/////////////////////////////////////

ivq = cad2index(B1);
Sb = S1;
l = 1;
CDPCM =2*C0; //linha que altera o contraste da imagem final
L = 8;
MuHat = zeros<mat>(1,65);
for(int i=0; i<=63; i++){
index = gray2term(trans(D1(span(1,3),span(i))));
int index_int = int (index);
if ((i % L) == 0){
 MuHat(i+1) = CDPCM(index_int-1)*(D1(0,i)*2 - 1);
}
else
 {
  MuHat(i+1) = MuHat(i) + CDPCM(index_int-1)*(D1(0,i)*2 - 1);
 }
}
Im = zeros<mat>(32,32);
for(int i=0; i<=28; i=i+4){
        for(int k=0; k<=28; k=k+4){
          mat Im_aux = zeros<mat>(4,4);
          int u_int = ( (i*8+(k))/4 + 2 ) - 1 ;
          Im_aux.fill( MuHat( u_int) );
          Im(span(i,i+3),span(k,k+3)) = Im_aux;
        }
}

/////////VQ///////

Phatb = zeros<mat>(4,64);
Phat0 = zeros<mat>(4,64);
for(unsigned int i=0; i<ivq.n_cols; i++){
        int u_int = int(ivq(i) - 1);//conversão de double para int
        Phat0(span() , span(i)) = C_aux( span() , span(u_int) );
}
for(unsigned int i=0; i<Phat0.n_rows; i++){
  Phatb(span(i), span()) = Phat0(span(i), span())%(Sb(span(i),span())*2-1); // o símbolo
                                                                            // '%' para
                                                                           // objetos
                                                                           // classe
                                                                           // 'mat', faz
                                                                           // o mesmo
                                                                           // que '.*'
                                                                           // no Matlab
}

Phatb(span(0,1), span()) = Phatb(span(0,1), span())/1.5811*2;
Xhatb = trans(H)*Phatb;
Yb = zeros<mat>(32,32);
for(int i=0; i<=63; i++){
  int k,L;
  mat aux1, aux2;
  L=8;
  double u = (i)/L;
```

```
  k = int(floor(u));
  j=i-k*L;
  aux1 = join_rows( Xhatb(span(0,3), span(i)),Xhatb(span(4,7), span(i)) );
  aux2 = join_rows( Xhatb(span(8,11), span(i)),Xhatb(span(12,15), span(i)) );
  Yb(span( k*4, k*4 + 3),span(j*4, j*4 + 3)) = trans(join_rows(aux1,aux2));
}
Im=flipud(Im);
Yb=flipud(Yb);
ImagemFinalb = Yb + Im;
ab=min(min(Yb));
mat abMat,bbMat;
abMat = ab;
bb=max(max(Yb));
bbMat = bb;
f = ones<mat>(3,3);
f = f/9;
mat ImagemFinalb_conv2 = zeros<mat>(34,34);
ImagemFinalb_conv2=conv2(ImagemFinalb,f);
ImagemFinalb=ImagemFinalb_conv2(span(1,32),span(1,32));
double mub;
mub = mean(mean(ImagemFinalb));
mat mubMat;
mubMat=mub;
ImagemFinalb = 0.5 + 0.5*tanh(10*(ImagemFinalb-mub));
mat imagem_media_VQ, imagem_media, aux;
aux = ones<mat>(5,5);
imagem_media_VQ = kron((Yb-ab)/(bb-ab),aux);
imagem_media = kron(ImagemFinalb,aux);
matriz_media = matriz_media + imagem_media;
mat imagem_original, separa, separa2, separa3, separa4;
separa = imagem_media_VQ;
separa2 = imagem_media;
separa3 = matriz_media/(iteracao+1);
separa4 = kron(Im,aux);
cube SP, SP2, SP3, SP4;
SP = zeros<cube>(separa.n_rows+7, separa.n_cols+7, 3);
SP(span(), span(), span(1)) = ones<mat>(separa.n_rows+7, separa.n_cols+7);
SP(span(), span(), span(0)) = ones<mat>(separa.n_rows+7, separa.n_cols+7);
SP(span(), span(), span(2)) = ones<mat>(separa.n_rows+7, separa.n_cols+7);
mat SP_slice0, SP_slice1, SP_slice2;
SP_slice0 = SP(span(), span(), span(0));
SP_slice1 = SP(span(), span(), span(1));
SP_slice2 = SP(span(), span(), span(2));
SP2 = SP;//Igual ao Matlab
SP3 = SP;//Igual ao Matlab
SP4 = SP;//Igual ao Matlab
for(int i=0 ; i<=7; i++){
  int offi = (i)*(4*5+1);
  int offi1 = (i)*(4*5);
  for(int j=0 ; j<=7; j++){
    int offj = (j)*(4*5+1);
    int offj1 = (j)*(4*5);
    SP(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(0)) =
separa(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(1)) =
separa(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(2)) =
separa(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP2(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(0)) =
separa2(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP2(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(1)) =
separa2(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP2(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(2)) =
separa2(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP3(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(0)) =
separa3(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP3(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(1)) =
separa3(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP3(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(2)) =
separa3(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP4(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(0)) =
separa4(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP4(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(1)) =
separa4(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));

    SP4(span(offi, offi + 4*5 - 2), span(offj, offj + 4*5 - 2), span(2)) =
separa4(span(offi1, offi1 + 4*5 -2), span(offj1, offj1 + 4*5 - 2));
```

```
  }
}
SP_slice0 = SP(span(), span(), span(0));
SP_slice1 = SP(span(), span(), span(1));
SP_slice2 = SP(span(), span(), span(2));

////Construindo Figura de 4 regiões////
mat M1,M2,M3,M4;
M1 = ones<mat>(5,349);
M2 = ones<mat>(167,5);// ones(size(SP,1),5,3)
M3 = SP3(span(),span(),span(0));// SP3
M4 = join_rows(M2, M3);// ones(size(SP,1),5,3) SP3
mat M5,M6;
M5 = SP2(span(),span(),span(0));//SP2
M6 = join_rows(M2, M5); //ones(size(SP,1),5,3) SP2
mat M7;
M7 = join_rows(M4, M6); // ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2
mat M8;
//ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2 ones(size(SP,1),5,3)
M8 = join_rows(M7, M2);
mat M9, M10;
M9 = SP4(span(),span(),span(0));// SP4
M10 = join_rows(M2, M9);//ones(size(SP,1),5,3) SP4
mat M11, M12;
M11 = SP(span(),span(),span(0)); // SP
M12 = join_rows(M2, M11); // ones(size(SP,1),5,3) SP
mat M13;
M13 = join_rows(M10, M12);
mat M14;
// ones(size(SP,1),5,3) SP4 ones(size(SP,1),5,3) SP ones(size(SP,1),5,3)
M14 = join_rows(M13, M2);
mat M15;
//ones(5,349,3) ; ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2 ones(size(SP,1),5,3)
;
M15 = join_cols(M1, M8);
mat M16;
//ones(5,349,3) ; ones(size(SP,1),5,3) SP4 ones(size(SP,1),5,3) SP ones(size(SP,1),5,3)
;
M16 = join_cols(M1, M14);
mat M17;
//[ones(5,349,3) ; ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2
ones(size(SP,1),5,3) ; ones(5,349,3) ; ones(size(SP,1),5,3) SP4 ones(size(SP,1),5,3) SP
ones(size(SP,1),5,3) ;
M17 = join_cols(M15, M16);
mat M18;
//[ones(5,349,3) ; ones(size(SP,1),5,3) SP3 ones(size(SP,1),5,3) SP2
ones(size(SP,1),5,3) ; ones(5,349,3) ; ones(size(SP,1),5,3) SP4 ones(size(SP,1),5,3) SP
ones(size(SP,1),5,3) ; ones(5,349,3)]
M18 = join_cols(M17, M1);
mat OutFrame = M18(span(5,343),span(5,343));

//Fim da Figura de 4 regiões
cv::Mat auxCV;
auxCV = cv::Mat::zeros(339, 339, CV_64F);
for(int nLinhas=0; nLinhas<339; nLinhas++ ){
  for(int nColunas=0; nColunas<339; nColunas++){
    auxCV.at<double>(nLinhas,nColunas) = (OutFrame(nLinhas,nColunas));
  }
}
auxCV.convertTo(auxCV, CV_32F);
cv::namedWindow( “IMAGEM”, 0 );
imshow( “IMAGEM”, auxCV );

//Fim do Teste de Tempo de Execução

finish = clock();
TE = ((double)(finish - start))/(CLOCKS_PER_SEC );
TE_total = TE_total + TE;
cv::waitKey(20);
iteracao++;//isso é salvo na matriz iterator

////////////////////////////////DECODER//END////////////////////////////////

//*******************************END***DECODER*****************************************
```

C.1. Código dec2binEscalar

Função que recebe um decimal e retorna o valor binário correspondente armazenado. É usado como função auxiliar da função dec2binVetor.

```
#include “stdafx.h”
#include <iostream>
#include “armadillo”
using namespace arma;
using namespace std;

mat dec2binEscalar(double dec, int n) {
	mat M;
	int decInt= int(dec);
	if ((n<=2)&&(dec<=3)){
		switch(decInt){
			case 0:
				M <<0<<0<<endr;
				break;
			case 1:
				M <<0<<1<<endr;
				break;

			case 2:
				M <<1<<0<<endr;
				break;
			case 3:
				M <<1<<1<<endr;
				break;
			default:
				cout<<“DEFAULT 1: Numero fora do intervalo [0,7]!”<<endl;
		}
	}
	else {

		switch(decInt){
			case 0:
				M <<0<<0<<0<<endr;
				break;
			case 1:
				M <<0<<0<<1<<endr;
				break;

			case 2:
				M <<0<<1<<0<<endr;
				break;
			case 3:
				M <<0<<1<<1<<endr;
				break;
			case 4:
				M <<1<<0<<0<<endr;
				break;
			case 5:
				M <<1<<0<<1<<endr;
				break;

			case 6:
				M <<1<<1<<0<<endr;
				break;
			case 7:
				M <<1<<1<<1<<endr;
				break;
			default:
				cout<<“DEFAULT 2: Numero fora do intervalo [0,7]!”<<endl;
		}

	}
	return M;
}//End dec2binEscalar
```

C.2. Código dec2binVetor

Função que recebe um decimal e retorna o valor binário correspondente armazenado numa matriz. É usado como função auxiliar da função cad2index.

```
#include “stdafx.h”
#include <iostream>
#include “armadillo”
#include “dec2binEscalar.h”

using namespace arma;
using namespace std;

mat dec2binVetor(Row<double> V, int n){

	mat M;
	for (unsigned int i = 0; i < V.n_cols; i++){

		if(M.is_empty())
			M = dec2binEscalar(V(i), n);
		else
			M = join_cols(M,dec2binEscalar(V(i), n));

	}


return M;
}//End dec2binVetor
```

C.3. Código cad2index

A função cad2index é importante para realizar o mapeamento da notação binária *Cadence* para a representação discreta original (de 1 a 128).

```
#include “stdafx.h”
#include “armadillo”
#include <iostream>
#include “cad2index.h”
#include “dec2binEscalar.h”
#include “dec2binVetor.h”
using namespace arma;
using namespace std;
mat cad2index(const mat B){
        mat I = zeros<mat>(2, B.n_cols);
        mat matConstante;
        mat B1;
        mat ivq;
        for(unsigned int k=0; k < B.n_cols; k++){
                mat    word1 = B(span(0,2), span(k));
                mat L1;
                L1 << 4<< 2<< 1<<endr;
                int operacao1 = int (as_scalar(L1 * word1));//converte o resultado da
multiplicaçao de matrizes num escalar

                switch (operacao1){
                case 0:
                        I(0,k)=2;
                        break;
                case 1:
                        I(0,k)=3;
                        break;
                case 2:
                                I(0,k)=1;
                        break;
                case 3:
                                I(0,k)=0;
                        break;
                case 4:
                                I(0,k)=5;
                        break;
                case 5:
                                I(0,k)=4;
                        break;
                case 6:
                                I(0,k)=6;
                        break;
                case 7:
                                 I(0,k)=7;
                        break;

                }//switch (operacao1)

                mat word2 = B(span(3,4),span(k));
                mat L2;
                L2<<2<<1<<endr;
                int operacao2 =int (as_scalar(L2 * word2));//converte o resultado da
multiplcaçao de matrizes num escalar
                switch (operacao2){
                        case 0:
                                I(1,k)=1;
                                break;
                        case 1:
                                I(1,k)=0;
                                break;
                        case 2:
                                        I(1,k)=2;
                                break;
                        case 3:
                                        I(1,k)=3;
                                break;
                }//switch (operacao2)
```

```
        B1 = trans(fliplr(dec2binVetor(I(span(0),span()),3)));
        B1 = join_cols(B1,trans(fliplr(dec2binVetor(I(span(1),span()),2))));
        B1 = join_cols(B1, B(span(5),span()));//análogo à sintaxe do matlab [B1; B(6,:)]
        B1 = join_cols(B1, B(span(6),span()));//análogo à sintaxe do matlab [B1; B(7,:)]
        }//for(int k=0; k <= B.n_cols; k++)
        matConstante<<64<<32<<16<<8<<4<<2<<1<<endr;//representa fliplr(2.^((1:7)-1))
        ivq = (matConstante*B1);
        ivq++;
return ivq;
}//mat cad2index(const mat B)
```

C.4. Código gray2term

A função gray2term recebe uma entrada binária em código Gray e retorna um valor binário. Toda vez que a função encontra o valor 1 na entrada binária, ela indica que achou um inteiro decimal.

```
 #include “stdafx.h”
#include “gray2term.h”
double gray2term(rowvec gray){
        mat  aux;
        double e, i,index;
        aux<<0<<1<<1<<endr
           <<0<<1<<0<<endr
           <<0<<0<<0<<endr
           <<0<<0<<1<<endr
           <<1<<0<<1<<endr
           <<1<<0<<0<<endr
           <<1<<1<<0<<endr
           <<1<<1<<1<<endr;

        e=1;
        i=0;
                while (e!=0){
          i=i+1;
         e = sum(abs(gray-aux(span(i-1),span()))) ;
         }
        index=i;
        return index;
}
```

C.5. Código conv2

Função conv2 realiza a convolução bidimensional entre duas matrizes.

```
#include “stdafx.h”
#include “conv2.h”
mat conv2(mat M1, mat M2){
int row, col;
row = (M1.n_rows + M2.n_rows - 1);//numero de linhas de Mresp
col = (M1.n_cols + M2.n_cols - 1);//numero de colunas de Mresp
mat Mresp = zeros<mat>(row, col);
//calculo de convolução 2D usando quatro laços for aninhados
for(unsigned int MrespRow=0; MrespRow<Mresp.n_rows; MrespRow++){ //laço da linha da
                                                                 //matriz resposta
                                                                 //“Mresp”
  for(unsigned int MrespCol=0; MrespCol<Mresp.n_cols; MrespCol++){ //laço da coluna da
                                                                   //matriz resposta
                                                                   //“Mresp”
    mat somaParcial = zeros<mat>(1,1);
    for(unsigned int krow=0; krow<Mresp.n_rows; krow++){ //laço das linhas das matrizes
                                                         //de entrada “M1” e “M2”
       for(unsigned int kcol=0; kcol<Mresp.n_cols; kcol++){ //laço das colunas das
                                                            //matrizes de entrada
                                                            //“M1” e “M2”
         if((krow>=0) && (krow<M1.n_rows) &&
           (kcol>=0) && (kcol<M1.n_cols) &&
           (MrespRow-krow>=0) && (MrespRow-krow<M2.n_rows) &&
           (MrespCol-kcol>=0) && (MrespCol-kcol<M2.n_cols)){

                 somaParcial(span(0),span(0)) =
                 somaParcial(span(0),span(0)) + M1(span(krow),span(kcol))
                 *M2(span(MrespRow-krow), span(MrespCol-kcol));

          }//if

        }//kcol
      }//krow
       Mresp(span(MrespRow), span(MrespCol)) = somaParcial(span(0),span(0)) ;
    }//MrepsCol
  }//MrespRow
  return Mresp;
}
```

Apêndice D

Rotina de construção da variável binária CSC0

Esta rotina fará o tratamento dos valores binários oriundos do chip, que chegaram à interface do usuário na forma de char e precisarão se convertidos em bits, para que possam ser lidos pelo decodificador

```
for(j=0;j<17;j++)
{
//Converte Buffer[j] para uma string que contem cada bit separado por ‘ ‘. Em seguida,
escreve em CSC0.
umat CSC0_aux=zeros<umat>(1,4);
int mascara = 0x01;
int i = 0;
int max = 8;
//Desse forma, o bit menos significativo será o 1o impresso
//Como 132 bits sao lidos, o ultimo byte nao sera completamente preenchido.
//Para esse byte so os 4 primeiros bits que importam.
if(j == 16){
  max = 4;
}
else{
   CSC0_aux = join_rows(CSC0_aux,CSC0_aux);
}
while(i < max){
   if((Buffer[j] & mascara) == mascara){
     CSC0_aux(i) = 1;
   }
else{
   CSC0_aux(i) = 0;
}
mascara *= 2;
i++;
}//END***while(i < max)
if(CSC0.n_elem == 1){
  CSC0 = zeros<umat>(1,8);
  CSC0 = CSC0_aux;
}
else{
   CSC0 = join_rows(CSC0,CSC0_aux);
}
}//END***for(j=0;j<17;j++)
```