Universidade Federal
do Rio de Janeiro
Escola Politécnica

PROGRAMAÇÃO DE MICROCONTROLADORES E INTERFACE EM C++ PARA LEITURA DE DADOS EM SENSORES DE IMAGENS CMOS

Tiago Monnerat de Faria Lopes

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: José Gabriel R. C. Gomes
Co-orientadora: Fernanda D. V. R. Oliveira

Rio de Janeiro

Fevereiro de 2017

PROGRAMAÇÃO DE MICROCONTROLADORES E INTERFACE EM C++ PARA LEITURA DE DADOS EM SENSORES DE IMAGENS CMOS

Tiago Monnerat de Faria Lopes

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Tiago Monnerat de F. Lopes

Tiago Monnerat de Faria Lopes

Orientador:

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

Co-orientadora:

Fernanda Duarte Vilela Reis de Oliveira, M. Sc.

Examinador:

Prof. Antonio Petraglia, Ph. D.

Examinador:

Prof. Carlos Fernando Teodósio Soares, D. Sc.

Rio de Janeiro – RJ, Brasil

Fevereiro de 2017

Declaração de Autoria e de Direitos

Eu, *Tiago Monnerat de Faria Lopes* CPF *140.747.237-27*, autor da monografia *Programação de Microcontroladores e Interface em C++ para Leitura de Dados em Sensores de Imagens CMOS*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.

Tiago Monnerat de F. Lopes

Tiago Monnerat de Faria Lopes

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

AGRADECIMENTO

Agradeço à meus pais por todo o esforço e dedicação que garantiram minha educação e permanência dentro deste curso, não importasse por quais problemas passassem. Assim como agradeço todo o amor e carinho que me dedicaram desde que me entendo por gente.

Agradeço para todo o sempre ao meu amor, Carolina, que me deu forças, carinho e que desde o início confiou na minha capacidade de concluir este projeto, apesar de diversas vezes eu mesmo ter duvidado. Obrigado por seus ouvidos atentos, suas palavras que me encheram de energia e sua companhia constante, acalentadora e apaixonante. E nem louco eu deixo de agradecer a sua edição rigorosa e seus sorrisos lindos. Eu te amo.

Agradeço a minha vó e bisavó Helenas, cujo apoio eu nunca fiquei sem e por terem sempre me mantido no foco com constantes "Vai se formar quando?". Mesmo sendo uma pergunta que desperta em todos nós, graduandos, um sentimento agridoce.

Meu infinito agradecimento e admiração aos meus orientadores, Fernanda e Gabriel, por estarem sempre dispostos a me ajudar. Por seus conselhos e apoio, por sua amizade e paciência. E, obviamente, pela sua sublime orientação. Também deveria agradecer pelas histórias de livros e cachorros com você, Fernanda.

E meu mais profundo agradecimento a todos os meus amigos e companheiros por essa longa e labutosa jornada. Passamos diversos anos juntos aqui que mais pareceram décadas. Enfrentamos muitas dores de cabeça, gente maluca e toda essa nata que o bloco H parece atrair no quesito humano. Mas permanecemos juntos rindo enquanto o Titanic afundava, nos divertindo um pouco com a nossa própria desgraça. E houve coisas muito boas também. Muita rataria. Muitos dentes quebrados. Obrigado a todos vocês.

Obrigado a quem está lendo isso aqui também. Não sei exatamente quem é você, mas está tomando um pouco do seu tempo para ler isso tudo. Haja paciência!

RESUMO

Um imageador é um circuito integrado desenvolvido para a captura e processamento das imagens. A captura é feita utilizando uma matriz de fotodiodos, elementos fotossensíveis que transformam a luz incidente em corrente. O imageador utilizado nesse projeto é capaz de realizar a compressão das imagens capturadas através de hardware. Um processo rápido e eficaz, uma vez que a compressão é feita de forma paralela, antes dos valores dos pixels serem convertidos para digital.

Este projeto trata da revisão e melhoria de toda a parte de programação do sistema de testes desse imageador, projetado no laboratório PADS. O sistema consiste em um microcontrolador PIC responsável por enviar sinais de controle ao imageador, ler os bits de saída do imageador e transmitir o resultado para um computador que irá realizar a decodificação e apresentação da imagem. Por praticidade, o decodificador foi inicialmente programado em MATLAB, mas isso resulta em lentidão e erros durante os testes. O objeto deste trabalho é melhorar esses aspectos.

Abordamos a criação de uma interface amigável com o usuário, assim como garantimos uma melhora de velocidade e correções de erros de funcionamento.

A linguagem utilizada foi C++ com o intuito de uma melhor performance e demonstramos como substituimos um sistema que antes dependia tanto desta linguagem quanto da plataforma MATLAB para funcionar.

Palavras-Chave: microcontrolador, imageador, C++, interface.

ABSTRACT

An imager is an integrated circuit developed for capturing and processing of images. Image capture is accomplished using a photodiode matrix, photosensitive elements which transform incident light into electric current. The imager used by this project is capable of carring out the compression of captured images by hardware. Focal-plane image compression is a fast and effective process, since the compression is done in parallel, before conversion of pixel values to digital.

This project deals with the revision and improvement of all the programmating part of the testing system of this imager, which was designed at the PADS laboratory. The system consists of a PIC microcontroller in charge of sending control signals to the imager, reading the imager output bits and transmitting the results to the computer that performs image decoding and display. For convenience, the decoder was initially programmed in MATLAB, but this results in low frame rate and errors during the tests. The objective of this work is to improve the decoder with respect to communication errors and frame rate.

We focus on the creation of a user friendly interface and we also look for speed improvement and operation error correction.

The language used was C++ aiming at a better performance, and we demonstrate how the system replaces the previous system, which was based on C++ and MATLAB.

Key-words: microcontroller, imager, C++, interface.

SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

CMOS - Complementary Metal-Oxide-Semiconductor

PADS - Processamento Analógico e Digital de Sinais

VQ - Quantização Vetorial

DPCM - Modulação por Código de Pulsos Diferenciais

PCM - Modulação por Código de Pulsos

USB - Universal Serial Bus

PIC - Peripheral Interface Controller

OPENCV - Open Source Computer Vision

Sumário

1 Introdução **1**

1.1 - Tema 1

1.2 - Delimitação 1

1.3 - Justificativa 1

1.4 - Objetivos 2

1.5 - Metodologia 2

1.6 - Descrição 3

2 Estrutura Básica **4**

2.1 - Sensor de Imagem CMOS 4

2.2 - Microcontrolador 7

2.3 - Base do Processamento 9

3 Bibliotecas de Códigos **12**

3.1 - OpenCV 12

3.2 - Processamento de Arquivos 14

2.3 - Processamento de Dados 16

4 Melhorias Efetuadas **23**

4.1 - Foco no Usuário 23

4.2 - Lógica de Comandos 28

4.3 - Operações Implementadas 32

5 Resultados **40**

6 Conclusões **46**

Bibliografia **47**

Apêndice A - Códigos da Interface **48**

Lista de Figuras

2.1 – Diagrama Esquemático do Circuito de Leitura do Fotodiodo 5

2.2 – Diagrama de Blocos da Compressão 6

2.3 – Placa de Testes 8

2.4 – Interface Usada na Versão Anterior do Projeto 10

2.5 – Fluxograma do Decodificador Implementado em MATLAB 11

4.1 – Opções de Uso e Tempo de Integração 24

4.2 – Fluxograma da Decodificação do DPCM com Opções de Uso 26

4.3 – Fluxograma da Exibição ao Usuário com *Flips* Implementados 27

4.4 – Estrutura Antiga do Processamento 29

4.5 – Estrutura Melhorada 30

4.6 – Organização da *Thread* de Decodificação 31

4.7 – Comandos da Interface 33

4.8 – Lógica Implementada no Comando *Bloco Principal* 34

4.9 – Interface com *Debug* Acionado 36

4.10 – Interface com *Config* Acionado 37

5.1 – Captura pelo Comando *Bloco Principal*. 41

5.2 – Captura com *Correção DPCM* Acionado. 42

5.3 – Captura com *Impor Limites* Acionado 42

5.4 – Diferentes Efeitos de Filtros na Imagem Final. 43

5.5 – Imagem Lena Sem Filtros-Padrão. 43

5.6 – Imagem Lena Com Filtros-Padrão 44

5.7 – Comando Debug Acionado e em Execução 44

Lista de Tabelas

3.1 – Declarações OpenCV de Matrizes 13

3.2 – Funções OpenCV Utilizadas. 13

3.3 – Funções da Biblioteca de Processamento de Arquivos 15

3.4 – Função: processar dados lidos 16

3.5 – Funções: obtendo Phat 17

3.6 – Função: obtendo VQ 18

3.7 – Funções: obtendo a componente DPCM 18

3.8 – Função: organizando a exibição ao usuário 20

3.9 – Funções: comandos auxiliares 21

Capítulo 1

Introdução

1.1 – Tema

O tema do trabalho é a confecção de códigos em linguagem C++ para controlar o funcionamento de um sensor de imagem CMOS [1] e garantir que tanto o processo de controle e tomada de dados quanto o de reconstrução da imagem e exibição para o usuário sejam feitos sob uma única linguagem e coordenação.

1.2 – Delimitação

O objeto de estudo são os bits de saída de um sensor de imagem CMOS. Esse sensor de imagem é responsável por capturar e comprimir uma imagem. Os bits de saída desse chip são organizados de forma serial e podem ser utilizados por um decodificador para reconstruir a imagem comprimida. Um microcontrolador é utilizado para gerar os sinais de controle do chip e realizar a comunicação entre esse chip e um computador, ele é responsável por apresentar uma interface gráfica que define o modo de operação do chip ao usuário e por decodificar a imagem.

Este projeto é de interesse para as atividades executadas no Laboratório de Processamento Analógico e Digital de Sinais (PADS), da EPOLI/UFRJ (Departamento de Engenharia Eletrônica e de Computação) e da COPPE/UFRJ (Programa de Engenharia Elétrica). Especificamente, as atividades em questão são projetos e testes cujo objeto de estudo são os sensores de imagem CMOS.

1.3 – Justificativa

Um chip com tecnologia CMOS de 0.18 µm, que realiza a captura e compressão de imagens, também chamado de imageador, foi projetado e fabricado como parte de uma dissertação de Mestrado [2]. Para o controle e amostragem desse imageador, foram

projetados três códigos distintos que implementam diferentes etapas do processo de reconstrução da imagem captada pelo imageador: um para o controle do microcontrolador, com um código em linguagem C padrão gravado internamente; um para a interface com o usuário, usando linguagem C++; e um responsável pela decodificação e pós-processamento dos dados obtidos, implementado em linguagem MATLAB.

Apesar de funcional, essa estrutura já estabelecida possui problemas de execução que envolvem a utilização de arquivos, com acessos dessincronizados por mais de um programa concomitantemente, e apresenta baixa eficiência em relação ao processamento por utilizar diferentes códigos de maneira isolada.

É de interesse abordar novas ideias, como a capacidade de alterar o processo em tempo real ou a unificação de códigos, pelas quais esse sistema pode ser aperfeiçoado em termos de tempo de processamento e de novas funcionalidades. As melhorias neste estudo podem ser utilizadas em diversos outros projetos de imageadores cuja saída é um conjunto de bits que deve ser decodificado e apresentado em forma de imagem.

1.4 – Objetivos

O objetivo deste projeto é unificar as linguagens utilizadas no processo de reconstrução dos dados captados pelo imageador, utilizando apenas dois códigos em C ou C++: um para o microcontrolador e outro responsável tanto pela interface com o usuário quanto pela reconstrução e exibição da imagem.

Os efeitos dessa alteração vão desde a remoção dos erros de conflito que surgiam devido à utilização de uma interface isolada do decodificador em MATLAB, até um ganho considerável em velocidade de processamento.

O projeto visa alcançar um melhor desempenho para o sistema vigente e garantir que o novo sistema possuirá uma acessibilidade maior para que terceiros possam utilizá-lo. O projeto será detalhadamente documentado, possibilitando a sua modificação para a utilização em outros projetos que envolvam imageadores.

1.5 – Metodologia

Este projeto se baseia em códigos MATLAB já estabelecidos na versão vigente de funcionamento para o pós-processamento dos dados recebidos. A partir deles e com o

uso de uma biblioteca especializada para o trabalho em imagens, conhecida como OPENCV, um código em C++ foi elaborado para garantir os mesmos resultados, mas com um tempo de processamento reduzido.

Após garantir que os resultados obtidos com o decodificador em C++ são equivalentes aos resultados com o MATLAB, foram criadas novas funcionalidades e opções de uso, incluídas na interface à qual o usuário tem acesso, garantindo que este tenha o máximo possível de controle sobre as pequenas variações de operação.

Para obter uma comparação direta de operação entre o sistema antigo e o atualizado, foram realizados extensos testes em laboratório utilizando ambos os sistemas. A taxa de exibição de imagens, por exemplo, é uma medida interessante, pois permite uma comparação quantitativa entre os dois sistemas. O resultado dessa comparação será apresentado no Capítulo 5.

1.6 – Descrição

No Capítulo 2 são apresentadas a base teórica e prática que fundamentaram o surgimento do projeto apresentado. A parte física do projeto é descrita através do sensor de imagem CMOS e do microcontrolador utilizado. A parte de controle do sistema atual é resumida nesse capítulo, de forma a oferecer um parâmetro para comparação com a versão melhorada, posteriormente apresentada.

O Capítulo 3 apresenta as bibliotecas que foram utilizadas no código da interface e pós-processamento, explicando os objetivos de cada função.

São apresentadas no Capítulo 4 as melhorias que foram realizadas durante todo o decorrer da produção deste projeto. Será explicado como tentamos melhorar a experiência do usuário, como pensamos em nosso código de maneira lógica e também como desenvolvemos todas as operações que foram criadas para utilizar ao máximo o imageador.

Os resultados são apresentados no Capítulo 5: algumas imagens obtidas com o imageador e os códigos desenvolvidos neste projeto, o estado atual da interface e as melhorias que obtivemos em tempo de processamento graças às modificações implementadas.

Esse texto é finalizado apresentando todas as ideias que foram obtidas e as conclusões a que chegamos ao fim deste trabalho.

Capítulo 2

Estrutura Básica

Este capítulo trata da estrutura básica do projeto e se divide em três partes: uma descrição breve do sensor de imagem CMOS (Seção 2.1), uma descrição breve do microcontrolador utilizado (Seção 2.2) e alguns conceitos básicos sobre o processamento (Seção 2.3).

2.1 – Sensor de Imagem CMOS

Desde que a patente pela tecnologia CMOS foi estabelecida em 1963, diversas aplicações distintas foram desenvolvidas e lançadas no mercado em diferentes áreas da tecnologia [3]. Seu dimensionamento reduzido e baixo consumo de energia são vantagens que se alinham perfeitamente com a tendência de miniaturização dos produtos modernos. Porém, mais do que uma abordagem econômica, a utilização da tecnologia CMOS permitiu diversos avanços em desenvolvimento de hardware.

A implementação dessa tecnologia em câmeras digitais permitiu a introdução de circuitos de processamento dentro da matriz de pixels. Dessa forma, a imagem pode ser processada imediatamente após a sua captura [4]. A introdução de circuitos dentro da matriz de pixels pode ser ainda mais interessante, pois permite processamento em paralelo, podendo ser de grande vantagem para diminuir o tempo de processamento dos sistemas de visão. Outra vantagem dessa tecnologia é a flexibilidade para a leitura dos pixels, permitindo que qualquer pixel possa ser lido sem a necessidade de que outro seja lido em conjunto.

Foi com essas vantagens em mente que um projeto para um chip integrado com tecnologia CMOS de 0.35 µm [5], que captura e comprime as imagens diretamente através de hardware, foi desenvolvido no PADS. Utilizando a característica dos sensores CMOS de permitir a integração de circuitos dentro do mesmo chip da matriz de pixels, foram implementados nesse chip circuitos analógicos capazes de comprimir a imagem assim que ela é captada pelos sensores. Essa técnica permite que os dados lidos não necessitem passar por um processo de conversão analógico para digital antes da

compressão, o que diminui o tempo de processamento. A compressão utilizada é feita através de técnicas de transformada linear, quantização vetorial (VQ) e uma modulação por códigos de pulsos diferencias (DPCM).

Uma segunda versão deste projeto, que tem como objetivo melhorar a qualidade da imagem, foi fabricada com tecnologia CMOS de 0.18 µm [6]. Esse chip é utilizado como base para o estudo aqui proposto. Dentre as modificações implementadas no chip de 0.18 µm, podemos citar uma sensibilidade maior dos sensores à luz, um aumento na resolução da imagem de 32x32 pixels para 64x64 pixels e um aumento no número de bits utilizados durante a quantização vetorial, o que resulta em uma melhora da qualidade da imagem.

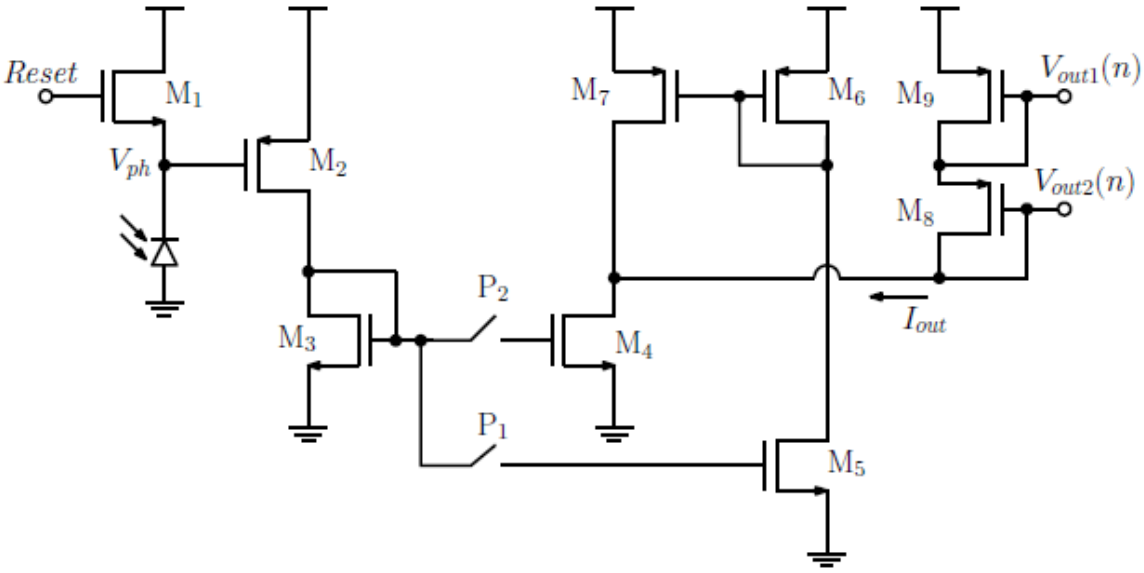

Figura 2.1 - Diagrama Esquemático do Circuito de Leitura do Fotodiodo.

O componente responsável por detectar a luz incidente é o fotodiodo. O circuito responsável por realizar a leitura desse componente, utilizado para cada pixel da matriz, é apresentado na Figura 2.1. A operação de leitura funciona com duas amostras, distintas no tempo, da corrente gerada pelo fotodiodo. Esses valores de corrente são diretamente relacionados à incidência de luz sobre o fotodiodo.

A corrente resultante após realizar a diferença entre essas duas medidas é a saída do circuito de leitura, I_{out}, que será copiada para o circuito que realiza a compressão da imagem. Essa leitura diferencial é importante para reduzirmos o ruído de padrão fixo (FPN).

A amostragem da corrente ocorre através de duas chaves analógicas que são regidas pelo microcontrolador, denominadas P1 e P2. Cada uma delas é acionada em um tempo preciso. O intervalo de tempo entre o desligamento da primeira chave e o desligamento da segunda chave é conhecido como tempo de integração. O conceito de tempo de integração será muito utilizado ao longo do texto.

O controle do tempo de integração diretamente através da interface foi uma das resoluções mais efetivas das melhorias implementadas, o que será abordado no Capítulo 4. Entretanto, aqui é importante apontar que cabe ao microcontrolador, responsável por gerar os sinais de controle do chip, realizar a definição do tempo de integração junto ao imageador.

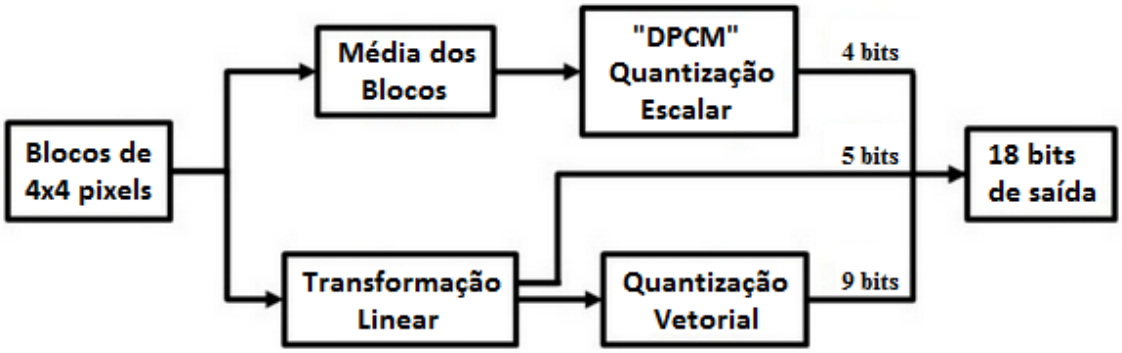

Figura 2.2 - Diagrama de Blocos da Compressão.

Para realizar a compressão, a imagem obtida é separada em blocos de 4×4 pixels, resultando em um arranjo com 16×16 blocos de 16 pixels. Cada um desses blocos passa por dois processos distintos, como demonstrado na Figura 2.2, que resultam em 18 bits por bloco.

Um desses processos consiste em obtermos a média dos blocos de 4×4 pixels. Ao invés de quantizarmos cada uma dessas médias, nós enviamos a diferença de uma dada média em relação àquela do bloco anterior. A média do bloco anterior é usada como estimativa para a média do bloco atual. Dessa forma, codificamos uma informação com variância reduzida, pois, em imagens naturais, existe alta correlação entre a média de dois blocos vizinhos. Este processo resultará num código binário que chamamos vetor com os bits do DPCM, mesmo nome do processo. O vetor com os bits do DPCM tem comprimento igual a quatro, ou seja, a diferença entre a média do bloco e a sua estimativa é quantizada utilizando quatro bits.

O outro procedimento consiste em passarmos cada bloco lido por uma transformação linear. Essa etapa representa a imagem utilizando uma base conhecida, com componentes de baixa e alta frequência. Quanto menor a frequência, maior é a importância daquela informação para a composição final da imagem. Somente os coeficientes das cinco componentes de maior energia são utilizados, todos os outros são descartados, sendo uma compressão com perdas. Como resultado dessa transformação linear, mantemos somente os coeficientes de maior importância. Separamos os bits de sinais dos coeficientes, representando cinco bits por bloco analisado, enquanto passamos adiante os módulos obtidos. Os módulos dos coeficientes são, então, mapeados através de uma quantização vetorial. Para implementar a quantização vetorial utilizando circuitos analógicos é realizada uma outra transformação linear, seguida de cinco quantizadores escalares, aplicados em paralelo a cada uma das cinco saídas escalares da transformação linear. O número de bits para cada coeficiente desta segunda transformação foi escolhido de forma a minimizar o erro médio quadrático, entre o vetor reconstruído e o vetor original, em um conjunto de treinamento, e de forma a minimizar ao mesmo tempo a taxa de bits. Após a quantização, o módulo dos coeficientes é representado utilizando nove bits. O número total de bits nesse procedimento são 14, sendo cinco bits de sinal e nove bits para os módulos.

Como podemos observar, cada um dos blocos da imagem pode ser representado por um vetor de 18 bits. O decodificador leva em conta quais desses bits são referentes ao VQ e quais são referentes ao DPCM.

2.2 – Microcontrolador

Um microcontrolador é um pequeno e simples computador implementado em um circuito integrado. Ele é capaz de realizar diversas tarefas segundo um código interno de funcionamento que pode ser programado de acordo com a necessidade de uso, possuindo capacidade de processamento e memória para realizar diversas funções.

É um dispositivo interessante para projetos automatizados, tanto por seu tamanho reduzido quanto por seu baixo custo. Apresenta a capacidade mista de trabalhar tanto com entradas analógicas quanto digitais porém, este projeto se limita a utilizar apenas suas capacidades com sinais digitais.

Foi utilizado um microcontrolador do tipo PIC 18F4550. A Figura 2.3 mostra o PIC instalado sobre a placa de testes, junto ao circuito integrado responsável pelo imageador. A função principal do microcontrolador desejada aqui é o controle de sinais de entrada e saída. Ele incorpora a função de intermediário entre o computador, que envia comandos desejados pelo usuário, e o imageador, que exige que certas entradas sejam controladas para que seu funcionamento ocorra de maneira adequada.

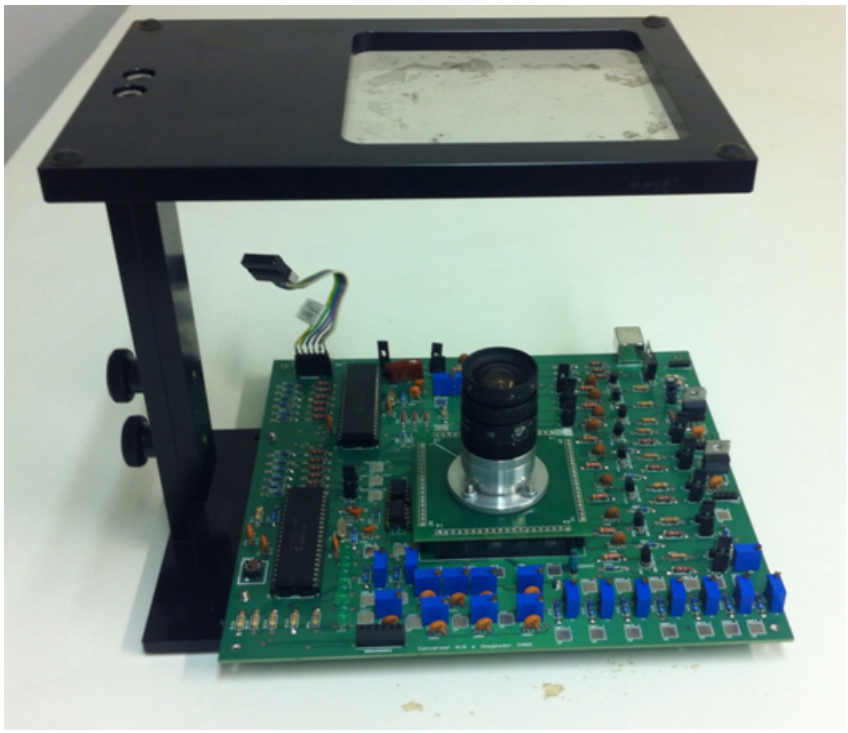

Figura 2.3 - Placa de Testes

Programamos o microcontrolador com rotinas que realizam tarefas distintas junto ao imageador. Um pacote de dados vindo do computador define qual delas será ativada. Resumidamente, a comunicação microcontrolador-computador consiste em comandos sendo recebidos pelo microcontrolador e em dados da imagem captada sendo enviados em pacotes de volta ao computador. Todo o processo ocorre através de comunicação via USB (*Universal Serial Bus*).

A parte do código que exige maior atenção se dedica aos comandos necessários para controlar o imageador. O programa interno do microcontrolador precisa sincronizar diversas entradas e saídas de forma que a captação feita pelos sensores de imagem seja concluída com êxito e que os resultados estejam disponíveis para a leitura.

Os sinais de controle do imageador incluem os responsáveis pelo fechamento das duas chaves de amostragem P1 e P2, já descritas anteriormente neste capítulo, e de um

transistor referente ao Reset, que inicia a operação do pixel, carregando a capacitância do fotodiodo que será descarregada pela luz. O sinal de Reset faz com que amostras passadas não afetem a captura de imagem atual.

Também faz parte dessa comunicação apontar ao imageador qual é a linha de blocos de pixels, dentro de um conjunto com 16 linhas de blocos, que a rotina deseja ler. Essa escolha ocorre através de uma única variável que mapeia a matriz de pixels como uma tabela, representando a posição dentro desta matriz.

O microcontrolador também lê os bits que compõem a imagem. Para tal, o microcontrolador deve enviar sinais para controlar um registrador de deslocamento interno do imageador e ler uma saída serial. Este controle é feito enviando um sinal de load, para que os bits sejam guardados no registrador e, em seguida, enviando um sinal de clock para que os bits sejam transmitidos um a um. A própria rotina de leitura se encarrega de organizar esses bits e transformá-los em bytes, estruturas fundamentais que compõem qualquer informação digital.

A programação interna do dispositivo é feita através de um código em C utilizando bibliotecas fornecidas pela empresa Microchip [7], produtora destes microcontroladores. As bibliotecas nos permitem criar as rotinas de operação em C e elas facilitam a comunicação via USB entre o código do microcontrolador e o computador. Esse código, então, é convertido para um arquivo hexadecimal que será gravado no microcontrolador através da placa de testes, havendo a necessidade de se utilizar um gravador em conjunto.

2.3 – Base do Processamento

Já apresentamos as duas primeiras etapas de obtenção da imagem: o imageador, que captura, comprime e transforma a imagem em bits; e o microcontrolador, que controla a operação do imageador, lê os dados de saída deste e os envia para o computador. Para finalizar a descrição da operação do sistema e apresentar por completo qual era o estado inicial em que se encontrava este projeto, devemos analisar uma última parte: a decodificação que acontece no computador e a interface com o usuário.

O processo tem início na interface microcontrolador-usuário. Um código em C++ gera uma interface que permite ao usuário definir quais atividades ele deseja executar, como podemos observar na Figura 2.4. Um comando é enviado, a partir da

escolha do usuário, para o microcontrolador, que gera os sinais de controle necessários à controlar o imageador. Após a amostragem e codificação da imagem, os dados obtidos saem do imageador, passam pelo microcontrolador e este os envia para o computador na forma de pacotes de 64 bits. Por causa desta limitação, diversos pedidos de leitura em sequência devem ser feitos pela interface e o microcontrolador envia as informações sem perdas ou sobreposições.

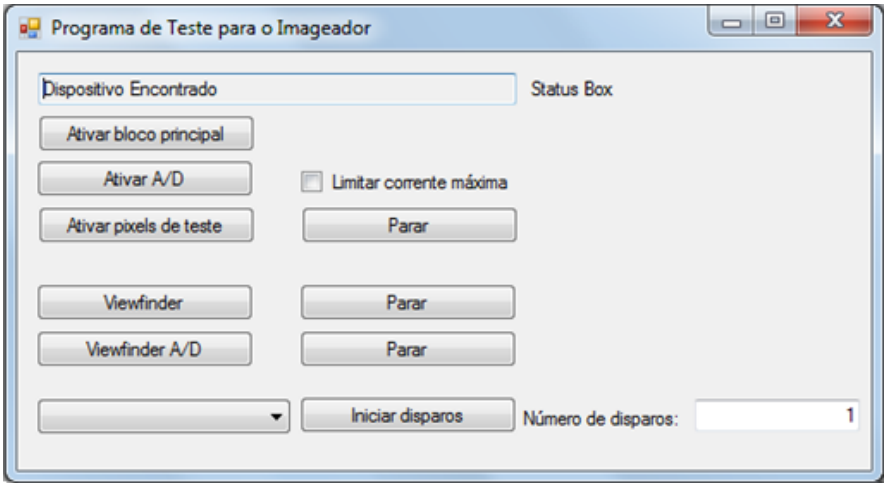

Figura 2.4 - Interface Usada na Versão Anterior do Projeto

Ao final desse processo de leitura, teremos os 4800 bits (18 bits de código básico × 64 blocos + 12 bits para correção de desvios do DPCM × 16 linhas) que compõem uma única imagem captada. O número 4800 tem uma parcela de 4608 bits necessários para representar as componentes VQ e DPCM da imagem, e outra de 192 bits de correção para o DPCM. Os efeitos desses bits de correção serão explicados no Capítulo 3. Esses pacotes de dados são quebrados bit a bit, alinhados pelo programa para organizar a decodificação, e salvos em um arquivo de texto.

Depois que os dados são salvos em um arquivo de texto, a interface indica, através de um semáforo (bit de controle), que a decodificação pode começar. Esse processo é realizado por um programa no MATLAB, que precisa ser acionado pelo usuário paralelamente à interface. O código de decodificação programado em linguagem MATLAB pela facilidade que esta linguagem fornece para o trabalho com grandes matrizes e operações matemáticas mais complexas.

Como pode ser visto no fluxograma da Figura 2.5, o código MATLAB lê os bits armazenados no arquivo de texto, transforma-os em um vetor, e dá início à decodificação da imagem. Esse vetor, então, é separado entre aquilo que representa o

VQ, o DPCM e os bits de correção. Cada um deles passará por seu próprio processo, como será melhor explicado no Capítulo 3, o que resulta em duas matrizes de 64×64 pixels. A imagem final é obtida através da soma entre as duas matrizes.

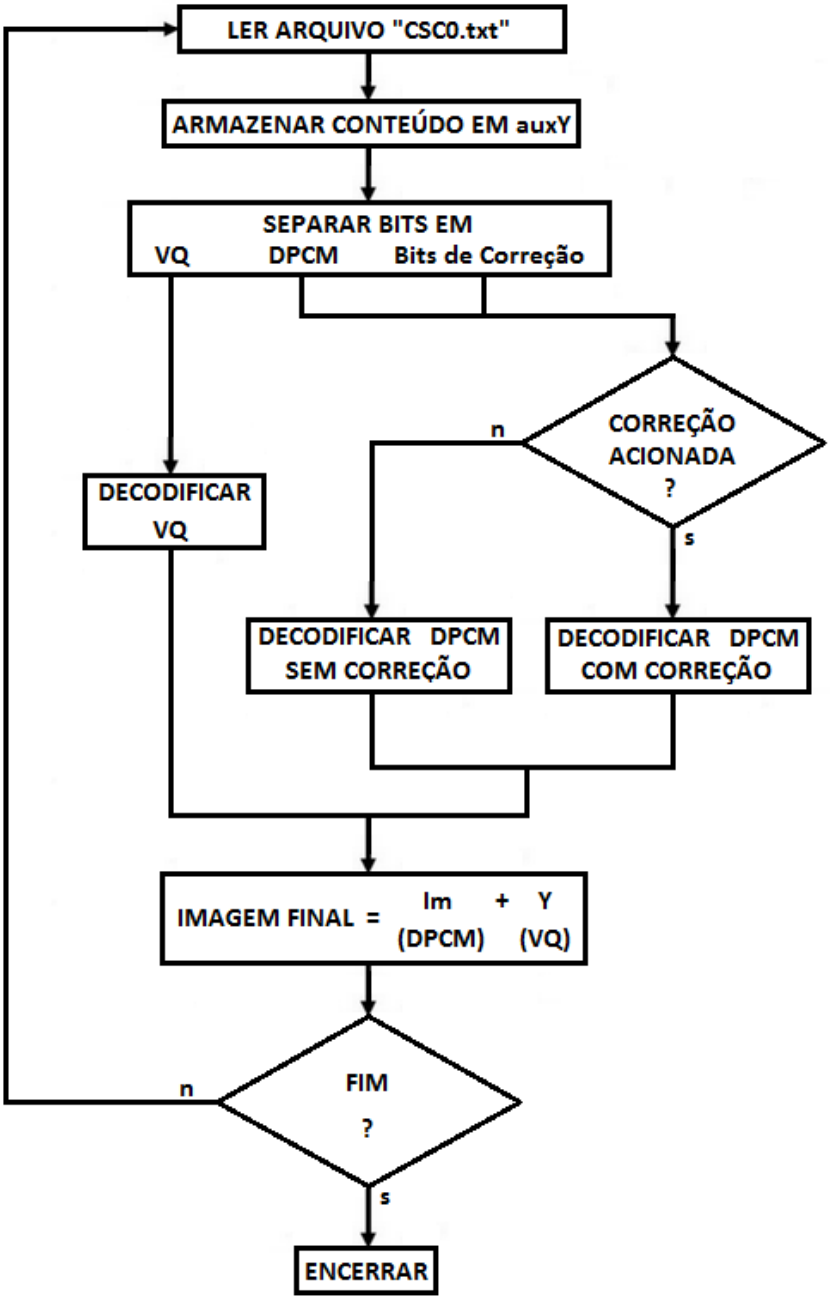

Figura 2.5 - Fluxograma do Decodificador Implementado em MATLAB

Capítulo 3

Bibliotecas de Códigos

Este capítulo trata das bibliotecas utilizadas no projeto e se divide em três partes: biblioteca OpenCV (Seção 3.1), biblioteca para processamento de arquivos (Seção 3.2) e biblioteca para processamento de dados (Seção 3.3).

3.1 – OpenCV

A biblioteca OpenCV é compartilhada de maneira gratuita, tanto para usos comerciais quanto acadêmicos, e projetada para diversas plataformas. O seu foco consiste em funções que realizam processamento em tempo real dentro do campo da visão computacional [8]. Em outras palavras, significa que ela trabalha adquirindo, analisando e/ou processando imagens digitais e vídeos.

Essa biblioteca foi escolhida por possibilitar o processamento das imagens obtidas pelo imageador de maneira análoga ao que era feito pelo código em MATLAB na versão anterior e por permitir a apresentação das imagens ao usuário. Isso ocorre porque a biblioteca OpenCV trata a imagem como uma estrutura denominada *Mat*, que representa a matriz de pixels e permite ao programador acessar cada elemento dela individualmente ou em grupos.

Outro motivo para essa escolha surgiu do fato do OpenCV apresentar algumas ferramentas muito úteis para o tratamento matemático de matrizes. Convoluções, multiplicações envolvendo matrizes e somatórios são alguns exemplos de ferramentas disponíveis nesta biblioteca e que são necessárias para a reconstrução da imagem. Para o nosso caso, é vantajoso possuir uma biblioteca que não só seja capaz de capturar e exibir imagens, mas que também forneça as ferramentas matemáticas para processa-lás, pois isso torna o código mais conciso e simples, uma vez que somente a biblioteca OpenCV será necessária.

Tabela 3.1 - Declarações OpenCV de Matrizes

```
cv::Mat1d matriz;
cv::Mat1d matriz = cv::Mat::ones(linhas, colunas, CV_64F);
cv::Mat1d matriz = cv::Mat::zeros(linhas, colunas, CV_64F);
cv::Mat1d matriz = cv::Mat_<double>(linhas, colunas) << /* dados */;

vector<cv::Mat1d> vetor_de_matrizes(comprimento_vetor);
```

Existem diversas formas de declarar matrizes dentro dessa biblioteca, mas utilizamos apenas aquelas que envolvem o armazenamento de números decimais em um arranjo específico de linhas e colunas. No caso, a variável *Mat1d* declara uma matriz composta por valores do tipo *double*, que são valores representados em ponto flutuante com 15 casas decimais após a vírgula. Como cada pixel dentro de nossas imagens é representado por um valor que varia entre o mínimo zero e o máximo um, não poderíamos trabalhar com uma variável sem partes fracionárias.

Além de simplesmente declararmos matrizes vazias, podemos também declará-las em tamanhos pré-determinados e com valores iniciais já preenchidos. Todas essas formas podem ser vistas na Tabela 3.1, onde apresentamos as funções utilizadas junto às nossas declarações durante a programação do novo sistema. Como um dos argumentos desses métodos, podemos ver o código CV_64F, responsável por determinar que os valores dos pixels de nossas matrizes são representados em ponto flutuante com 64 bits, garantindo uma precisão maior de representação.

Tabela 3.2 - Funções OpenCV Utilizadas

```
matriz(cv::Range(inicio, fim), cv::Range(inicio, fim));

cv::transpose(fonte, destino);
cv::flip(fonte, destino, direção);

cv::namedWindow(título, flags);
cv::imshow(título, matrix);
cv::filter2D(fonte, destino, posição_âncora, kernel);
```

Quanto às facilidades que a biblioteca propicia, temos algumas funções apresentadas na Tabela 3.2 que simplificam o trabalho envolvendo matrizes e sua apresentação como imagens.

A biblioteca OpenCV nos permite trabalhar apenas com partes específicas das matrizes através da função *cv::Range*. Quando utilizamos essa função para definir os argumentos de uma variável de matriz, podemos escolher quais linhas e colunas serão lidas. Essa flexibilidade nos permite utilizar apenas uma linha ou coluna de uma dada

matriz, ou mesmo uma sub-matriz, que exista interna à matriz fonte. O efeito dessa função não apenas se resume a destacar e utilizar parcialmente uma matriz, mas inclui alterar parcialmente uma matriz já declarada.

Realizar operações de transposição e inversão da ordem de linhas ou colunas de uma matriz também facilita muito a implementação do código. Essas funções da biblioteca OpenCV têm uma implementação simples que consome pouco tempo de máquina. Essas operações foram utilizadas em matrizes com mais de mil elementos sem representar um gargalo significativo no tempo de operação. A velocidade de processamento é importante, principalmente para os casos em que desejamos trabalhar com o imageador em tempo real, quando o uso do imageador envolve repetir o mesmo processo diversas vezes.

A apresentação dessas matrizes ao usuário também pode ser feita de maneira muito simples. Existe uma função, chamada *cv::namedWindow*, que irá criar uma janela cujo tamanho se ajusta à informação que desejamos exibir. O título da janela é definido por um dos argumentos da função. Em seguida, precisamos apenas declarar uma variável *Mat1d* que é associada à essa janela e é exibida. Para trabalhar dentro de uma iteração através da qual imagens são apresentadas por segundo, como desejado ao se trabalhar com o imageador, precisamos acrescentar uma pequena parada, de alguns microssegundos, no código para garantir que a imagem seja carregada adequadamente. Na versão de código feita neste projeto, utilizamos uma parada de quarenta microssegundos.

Uma função que demonstra bastante a versatibilidade da biblioteca OpenCV é a *cv::filter2*. Ela passa a imagem, armazenada em uma matriz, através de um filtro realizado de acordo com uma matriz kernel atribuída pelo usuário. Esse processo, realizado por uma convolução, consome baixo tempo de máquina e nos permite reduzir os efeitos de blocagem e ruído de nossa imagem final. A utilização de um filtro torna a imagem mais suave e evita grandes variações entre os valores de pixels vizinhos.

3.2 – Processamento de Arquivos

Algumas variáveis, essenciais para a decodificação da imagem, como o dicionário do VQ, são guardadas em arquivos de texto, que são lidos logo que a interface é iniciada. Dessa forma, o código se torna mais limpo, pois a declaração dessas variáveis é feita através de uma função responsável por ler os arquivos. Por este e outros motivos,

vimos a necessidade de criar uma biblioteca para o tratamento de arquivos de texto. A Tabela 3.3 mostra as declarações das funções implementadas nesta biblioteca.

O primeiro grupo de funções desta biblioteca contém aquelas responsáveis pela leitura de dados armazenados em um arquivo de texto e que são importantes para a composição de alguma matriz. A função *lerMatC* é responsável por ler o arquivo de texto *C.txt*, que contém os valores que compõem o dicionário do VQ e os armazena em uma variável de matriz do tipo *Mat1d*. O dicionário do VQ será explicado mais adiante.

Outra função dessa biblioteca, chamada *lerDados*, existe para ajudar em testes e *debug* dos códigos apresentados neste projeto. Essa função permite que a decodificação e pós-processamento da imagem, além de outras funções da interface gráfica, sejam testados sem a necessidade de conectar o computador que executa a interface gráfica à placa de testes. Para tal, essa função lê um arquivo de texto que armazena o conjunto de bits referentes à uma única imagem capturada pelo imageador. Isso permite que diversos testes sejam feitos e que a imagem resultante gerada pela versão em MATLAB seja comparada com a imagem gerada pela versão em C++, garantindo que as imagens decodificadas através do decodificador no MATLAB e do decodificador em C++ correspondam ao mesmo resultado final.

Tabela 3.3 - Funções da Biblioteca de Processamento de Arquivos

```
static cv::Mat1d lerMatC();
static cv::Mat1d lerDados();

static bool carregarInfo( nome_arquivo, numero_de_dados, endereço_destino);
static void sendMessageError( código_arquivo, código_erro);
```

Outra função importante é a *carregarInfo*. É através dela que certos arquivos de configuração são lidos. A leitura de arquivos de configuração permite ao usuário introduzir, mesmo com o programa já em funcionamento, variáveis na lógica de obtenção da imagem resultante. Existem dois arquivos de configuração no formato atual do projeto aqui apresentado: *Offset.txt* e *Tempo_de_Integracao.txt*. Eles são responsáveis por introduzir duas matrizes, de nomes iguais aos dos arquivos, que são utilizadas pelo sistema em quase todo processo implementado neste projeto. A participação de ambas na reconstrução da imagem será abordada mais à frente em detalhes.

Caso o processo de carregar os arquivos de configuração falhe, um erro é introduzido na lógica dos códigos. Esse erro resultaria numa falha e interrupção abrupta

do programa. Para que isto não aconteça, a função *sendMessageError* foi implementada. Ela analisa se os arquivos de configuração foram lidos corretamente e, de acordo com a necessidade, envia uma mensagem de erro ao usuário avisando que o processo não foi concluído. Esse erro não interrompe o processamento. Ele apenas avisa ao usuário que seu comando para usar esses arquivos não pôde ser realizado. Cabe ao usuário, após ser avisado pela mensagem de erro, checar a integridade desses arquivos.

3.3 – Processamento de Dados

Todas as rotinas de decodificação das imagens obtidas foram divididas e implementadas pela biblioteca Processamento de Dados, assim como algumas funções importantes para concluir esse trabalho de maneira organizada e eficiente. A decodificação foi dividida em etapas e cada uma delas foi transformada em uma função específica de estrutura simples. Essas funções transformam um certo dado ou matriz inicial e retornam uma ou mais matrizes em função de variáveis de controle.

Tabela 3.4 - Função: processar dados lidos

```
static void processData(aux_Y,                  //in: bits que compõem a imagem
                        aux_DC,                 //in: bits de correção ao DPCM
                        endereço_S1,            //out: bits do sinal de VQ
                        endereço_B1,            //out: bits do módulo de VQ
                        endereço_D1,            //out: bits do DPCM
                        endereço_vetor_DC1      //out: vetores de correção do DPCM
                       );
```

A imagem capturada pelo imageador e enviada ao computador é representada por um vetor de 4800 bits, conforme foi explicado na Seção 2.3. Organizados de maneira sequencial dentro deste vetor estão grupos de bits que contém informações sobre o DPCM e o VQ de cada bloco. Dentro deste vetor, os bits vêm separados por linhas de blocos da imagem. Para separar cada grupo de informações contidas dentro deste vetor, é usada a função *processData,* cuja declaração é mostrada na Tabela 3.4. Essa função recebe como entrada esse vetor de 4800 bits dividido em duas partes: uma parte representando a sequência de bits que compõem as partes de DPCM e VQ da imagem, e outra parte com os bits que servem para a correção de erros do DPCM.

De posse desses dados, a função sobrescreve três matrizes já declaradas e que foram passadas à ela através de seu endereço. A matriz S1 tem dimensão 5×256 e recebe os cinco bits de sinal das componentes da transformação linear para cada bloco,

enquanto os nove bits que codificam o vetor com cinco módulos (ou seja, o módulo de cada componente da transformação linear apresentada na Seção 2.1) são armazenados na matriz B1, que tem dimensão 9×256. A matriz D1 que tem dimensão 4×256, fica com os quatro bits do DPCM para cada bloco.

Por fim, essa função também passa três matrizes para um vetor chamado DC1. Cada uma delas representa os bits referentes a cada bloco de correção do DPCM, que serão utilizados mais a frente por outra etapa do processamento.

Para entender o processo de decodificação da imagem, começaremos analisando a parte em que trabalhamos com os bits de sinal e os bits do VQ. A imagem reconstruída a partir desses bits é armazenada em uma matriz chamada Y. Para obtê-la, primordialmente, precisaremos considerar algumas etapas intermediárias. A primeira dessas etapas será obter o vetor Phat (lê-se "P-chapéu", uma estimativa do vetor P, sendo que P corresponde aos cinco módulos originais dos resultados da transformação linear), que contém a reconstrução dos módulos dos cinco coeficientes da transformada linear. Isso ocorrerá na função denominada *getPhat*.

Tabela 3.5 - Funções: obtendo Phat

```
static cv::Mat1d cad2indexFernanda(dados      //in: dados que serão convertidos
                                  );
static cv::Mat1d getPhat(Cnovo,               //in: dicionário VQ
                         B1,                  //in: bits do módulo de VQ
                         S1                   //in: bits do sinal de VQ
                        );
```

Para obtermos Phat, precisamos de alguns recursos fixos já pré-definidos no código. Um deles é o dicionário que nos permite associar o valor obtido com os bits de módulo de VQ com um vetor de reconstrução para o Phat. Esse dicionário possui cinco linhas e 512 colunas, onde cada linha corresponde a uma dimensão do VQ. Essas cinco dimensões são referentes às cinco componentes de maior energia da transformada linear aplicada aos blocos com 4×4 pixels. Nesse dicionário 5×512, cada coluna representa uma aproximação de cinco valores que essas componentes podem assumir.

Outro recurso utilizado é a função *cad2indexFernanda*. Essa é uma função que já existia na versão anterior do projeto e que foi convertida de MATLAB para C++. Ela mapeia cada coluna da matriz B1 que armazena os bits do VQ em um índice para o dicionário. Especificamente, ele selecionará a coluna do dicionário que será utilizada.

Combinando essas ações anteriores e a coluna da matriz S1, reconstruimos os cinco coeficientes da transformada linear. Os valores dos coeficientes com sinal são salvos na matriz Phat com dimensão 5×256, que será então passada para a próxima etapa.

Tabela 3.6 - Função: obtendo VQ

```
cv::Mat1d Xhat = H_t*diag*Phat;

static cv::Mat1d getY(cv::Mat1d Xhat            //in: Phat ajustado
                      );
```

Para a reconstrução do VQ, multiplicamos Phat pela transposta da matriz H, que é a matriz utilizada durante a compressão para obter os coeficientes de maior importância para a imagem, e por uma matriz diagonal para compensar as diferenças entre as normas das linhas de H. Denominamos de Xhat o resultado desta etapa.

A função *getY*, apresentada na Tabela 3.6, encerra o processamento do VQ. Essa função transforma Xhat, que está na forma de matriz 16×256, em blocos de 4x4 pixels e organiza esses blocos, formando uma matriz com 64x64 pixels, chamada Y, que representa a textura da imagem capturada pelo imageador.

Tabela 3.7 - Funções: obtendo a componente DPCM

```
static int gray2Term(dados                      //in: dados que serão convertidos
                     );
static void processDPCM(D1,                     //in: bits do DPCM
                        vetor_DC1,              //in: vetores de correção do DPCM
                        C_dpcm,                 //in: dicionário DPCM
                        offset,                 //in: vetor com offsets
                        check,                  //in: controlador da correção
                        limit,                  //in: controlador do limite
                        endereço_MuHat,         //out: matriz com DPCM parcial
                        endereço_indexTerm      //out: vetor de índices
                       );
static cv::Mat1d getIm(MuHat
                       );
```

A obtenção da componente referente ao DPCM exige menos etapas, todas apresentadas na Tabela 3.7. Os bits que representam o DPCM, contidos na matriz D1, são organizados em vetores de 4 bits. O primeiro bit representa o sinal, enquanto que os outros três representam o módulo do erro de predição referente à média dos valores dos pixels do bloco atual. Esses três bits de módulo do DPCM são transformados em índices

para um dicionário de reconstrução, nesse caso o dicionário do DPCM, definido na variável *C_dpcm* logo que a interface é iniciada.

A função *gray2Term*, que também representa uma conversão direta de uma função em MATLAB do projeto anterior, converte esses três bits de módulo, que estão em código gray, para um número inteiro de um a oito. Os índices resultantes dessa função são armazenados dentro de um vetor chamado indexTerm.

Esses índices serão utilizados como um selecionador dentro do dicionário dedicado ao DPCM. Esse dicionário, muito mais simples que o dicionário 5×512 do VQ, apresenta apenas uma linha com oito colunas.

Para a reconstrução do DPCM, precisamos do valor médio do bloco anterior para encontrar o valor médio do bloco atual. A média do bloco atual é dada pela média do bloco anterior somada ao valor encontrado através do dicionário multiplicado pelo seu sinal (primeiro bit de D1). No caso do primeiro bloco de cada linha, utilizamos um valor constante (correspondente a 0.37, em uma escala de 0 a 1, e sujeito à erros de fabricação) para definir a estimativa desse bloco. O resultado da decodificação é guardado sequencialmente no vetor *MuHat*.

Devido à imprecisão ao fabricar os circuitos, podem surgir erros no DPCM que o levem a divergir do valor correto. Os bits de correção de DPCM são referentes a três blocos posicionados após o bloco 16 de cada linha e que possuem entrada igual a zero, e esses bits podem ser usados para, de maneira aproximada, reduzir essa possível divergência e manter sob controle os blocos do DPCM.

Todo esse processo é realizado pela função *processDPCM*, que possui algumas variantes de processamento que serão apresentadas mais à frente. Tais variantes permitem a configuração do processo de reconstrução do DPCM pelo usuário, possibilitando que ele tenha controle de algumas etapas simples dentro da reconstrução. Também existe uma segunda versão desta função, chamada de *processPCM*, que realiza exatamente a mesma operação, porém com um ponto distinto: o valor médio do bloco anterior não é usado para predição do bloco atual, acarretando que cada conjunto de três bits de módulo representa a média do bloco em si e não a diferença entre a média e uma estimativa diferente de zero, proveniente do bloco anterior.

Encerrando o processo do DPCM, temos a função *getIm*. Ela converte os resultados da função anterior, armazenados na matriz MuHat, para a matriz de 64x64 pixels que usamos para apresentar a imagem de DPCM decodificada, à qual nos referimos pelo símbolo Im.

Tabela 3.8 - Função: organizando a exibição ao usuário

```
static cv::Mat1d prepareFinalImage(Im,          //in: matriz DPCM
                                   Y,           //in: matriz VQ
                                   imagemFinal, //in: matriz DPCM + VQ
                                   imagemMedia  //in: média das imagens passadas
                                  );
```

A imagem final é encontrada através da soma entre a matriz de 64x64 encontrada com os bits do VQ, Y, e a matriz de mesmo tamanho encontrada com os bits do DPCM. O resultado dessa soma é guardado na variável imagemFinal. Com todas as componentes reconstruídas, cabe ao programa exibir os resultados ao usuário. A função responsável por essa operação já foi apresentada dentro da biblioteca OpenCV, mas ela tem a limitação de apresentar uma única matriz por vez através de uma janela previamente declarada. Como é interessante mostrar não só a matriz que representa a imagem como também as matrizes de VQ e DPCM, a função *prepareFinalImage* foi implementada. As quatro matrizes que devem ser apresentadas ao usuário são dadas como entrada para essa função: a matriz Im, que representa a média de cada bloco dentro da imagem; a matriz Y, que representa a textura da imagem obtida; a matrix imagemFinal, que representa a imagem, capturada e comprimida pelo imageador, reconstruída; e a matriz imagemMedia, que apresenta uma média entre todas as imagens obtidas em sequência, desde o ínicio da operação do decodificador. Com todas as matrizes disponíveis, nós as concatenamos lado a lado para formarem uma única matriz que contenha tudo aquilo que é necessário para o *display* (DPCM, VQ, imagem reconstruída atual e média temporal de imagens reconstruídas). Entre cada par de matrizes (DPCM e VQ, por exemplo), introduzimos colunas contendo somente pixels brancos por questões de organização, melhor visualização e estética.

Apesar desta biblioteca possuir em sua maioria funções dedicadas diretamente às rotinas de decodificação, há funções que implementam pequenas transformações de dados que são utilizadas diversas vezes dentro do programa como um todo, de forma geral. Essas funções podem ser agrupadas como comandos auxiliares que estão presentes para exercer uma tarefa repetidas vezes, ou como comandos que foram reunidos em uma função para simplificar o entendimento do programa como um todo.

Tabela 3.9 - Funções: comandos auxiliares

```
static void carregarByte(byte,                  //in: byte a ser escrito
                         destino,               //out: matriz a ser escrita
                         posição,               //in: posição aonde escrever byte
                         indice_do_byte         //in: indice do byte
                        );
static cv::Mat matlabReshape(matriz,            //in: matriz a ser modificada
                             linhas,            //in: linhas desejadas
                             colunas            //in: colunas desejadas
                            );
static System::String^ matToString(matriz,      //in: matriz a ser convertida
                                   precisão     //in: precisão de casas decimais
                                  );
static cv::Mat1i indexTermToMat(vetor_de_index //in: vetor a ser convertido
                                );
```

Podemos incluir a função *carregarByte* nas duas categorias mencionadas (uso repetido e parte de uma função maior). Ela é invocada diversas vezes dentro do *loop* de leitura implementado no código da interface, captando os dados que são recebidos via USB e armazenando-os de maneira ordenada dentro de uma matriz. Além de garantir que essas informações lidas sejam obtidas sem erros, preenche com zeros essa matriz quando um pacote é enviado incompleto. Essa situação ocorre sempre no último pacote enviado, que possui quatro bits a menos que um envio comum. Isso ocorre porque possuímos um número de bits a ser enviado que não é múltiplo do número de bits que enviamos em cada pacote.

Outra função que se fez necessária se chama *matlabReshape*. Essa função é necessária porque, apesar de existir uma função de *reshape* dentro da biblioteca OpenCV, a forma com que o MATLAB e a biblioteca OpenCV trabalham suas matrizes é diferente. Enquanto MATLAB indexa elementos de matrizes contando elementos da primeira coluna em primeiro lugar, a biblioteca OpenCV faz a contagem dos elementos da primeira linha em primeiro lugar.

O código de *matlabReshape*[1] foi encontrado para suprir essa necessidade, realizando todo o processo da mesma maneira que o MATLAB. Seu uso é simples e idêntico ao da versão original: uma matriz tem o número de linhas e colunas alterados para aqueles definidos pelo usuário como parâmetros de entrada da função, desde que o número de elementos dentro da matriz não seja alterado.

[1] A função *matlabReshape* foi obtida através do código compartilhado pelo usuário Tempux no fórum de programação StackOverflow, que pode ser acessado pelo link:
< *http://stackoverflow.com/questions/30847565/performane-issue-for-matlab-like-reshape-in-opencv*>

As duas últimas funções desta biblioteca têm utilidades similares. Enquanto *matToString* faz a conversão do conteúdo de uma matriz para uma variável do tipo *String*, e assim permite que ela seja exibida pela interface; a função *indexTermToMat* permite que um vetor contendo os índices do DPCM seja convertido para uma matriz. O funcionamento da primeira função complementa o funcionamento da segunda, uma vez que a matriz contendo os índices foi criada para que possa ser convertida em texto e ser exibida ao usuário. A exibição dessa variável é importante para os testes realizados, para que o usuário possa ver com detalhes o funcionamento do DPCM.

Capítulo 4

Melhorias Efetuadas

Este capítulo trata especificamente de cada uma das melhorias que foram realizadas na interface. Ele está dividido em três seções: melhorias focadas no usuário (Seção 4.1), a lógica de comandos utilizada (Seção 4.1) e operações implementadas (Seção 4.3).

4.1 – Foco no Usuário

Como o projeto se fundamenta em diversas partes trabalhando em conjunto, tínhamos um problema quando precisávamos atualizar qualquer parte da lógica de processamento. Precisávamos, por exemplo, modificar o código do microcontrolador caso desejássemos alterar o tempo de integração utilizado na captura de uma imagem inteira, ou de parte de uma imagem. Esse processo envolvia desligar toda a alimentação da placa de testes, modificar pinagens para acionar o modo de gravação de PIC, conectar um gravador externo e carregar o novo código. Também tínhamos de garantir que a decodificação em MATLAB e a interface estavam atualizadas de forma a permitir a nova abordagem ou alteração desejada. Uma bateria de testes em bancada exigia muito tempo e esforço repetitivo do usuário para obter resultados com as mais simples modificações.

Tal fato decorria das linhas de código estarem completamente fechadas à alterações em tempo real vindas do usuário, como variações de parâmetros ou mudanças de técnicas de amostragem. Diversas variáveis do programa (na leitura do sensor de imagem ou no decodificador), como o tempo de integração e o valor de predição para o DPCM do primeiro bloco de cada linha utilizados, precisam ser alteradas quando queremos estudar o funcionamento do imageador, porém não tínhamos esses recursos antes das melhorias aqui propostas.

Apesar de ser apenas um dos possíveis exemplos, ter encontrado um método para selecionar um tempo de integração através do usuário foi a primeira modificação que trouxe uma melhora significativa em termos de facilidade de uso do sistema. Essa

melhoria se tornou um modelo para todas as melhorias subsequentes, dentro da especificação de reduzir gastos do usuário para configurar o sistema. A cada função que era implementada, a mesma questão era posta em evidência: como assegurar que o usuário tenha a maior liberdade possível de configurar este processo sem precisar regravar dispositivos ou recompilar programas?

Sendo a primeira destas alterações, a nova técnica de alterar o tempo de integração exigiu que o controle desta variável saísse do código do microcontrolador e passasse para a interface com o usuário. Disponibilizamos um espaço na interface que permite introduzir múltiplos de dez dentro de uma gama de valores para o funcionamento do imageador. O intervalo de tempo de integração é de dez microssegundos até dez mil microssegundos. O usuário pode alterar o valor do tempo de integração mesmo durante a execução de algum comando. Isso permite observar na tela do computador os efeitos desta alteração em tempo real.

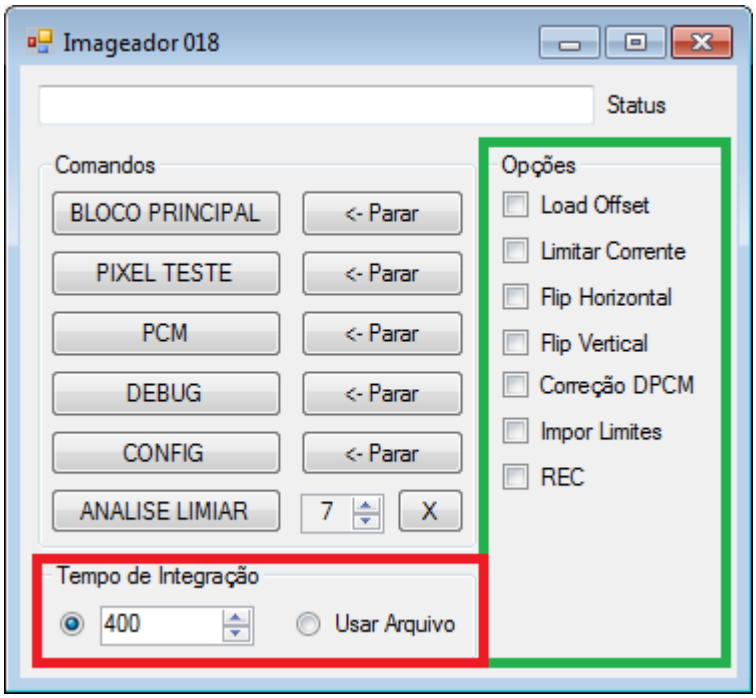

Figura 4.1 - Opções de Uso e Tempo de Integração

A interface enviará, via USB, ao microcontrolador três bytes para representar o valor presente da variável. Cada um deles representa, respectivamente, unidade, dezena e milhar do tempo de integração solicitado. Dentro do código do microcontrolador, esses valores são convertidos para o tempo que o dispositivo espera entre suas duas amostragens, P1 e P2, como já foi explicado na Seção 2.1.

Existem diferentes abordagens para esse processo. Podemos ver na Figura 4.1, na parte esquerda (marcada em vermelho), que o usuário pode escolher entre os dois métodos pressionando um seletor ao lado da caixa de entrada, onde introduzimos um valor numérico para o tempo de integração, ou um seletor ao lado da marcação *Usar Arquivo*. Pelo tipo de seletor, é impossível que o usuário escolha os dois métodos ao mesmo tempo.

Esse segundo método envolve usar um arquivo de texto denominado *Tempo_Integracao.txt*. Caso essa forma de definir o tempo de integração seja utilizada, a interface ignora o valor que o usuário introduz manualmente e envia os valores que o dito arquivo armazena. Essa segunda abordagem permite que linhas diferentes da imagem tenham valores de seus pixels capturados com tempos de integração diferentes. Essa abordagem surgiu porque detectamos comportamentos distintos entre cada linha de blocos da imagem, cujos motivos ainda estão sendo averiguados experimentalmente.

Muito similar ao que foi proposto acima, existe a opção de utilizar valores diferentes , em linhas de blocos diferentes, para os valores de predição para o DPCM do primeiro bloco da linha. Assim como no caso do tempo de integração, existe um arquivo chamado *Offset.txt* que armazena os valores de predição do primeiro bloco de cada linha. O usuário pode definir um valor diferente para cada linha de blocos da imagem. Como já dito, o processo de fabricação do chip cria componentes ligeiramente distintos em cada parte do sensor, o que resulta em respostas díspares para cada linha da imagem. Essa opção substitui o valor de predição fixo que é usado no circuito fabricado, mas pode ser desligada a qualquer instante mesmo com a decodificação da imagem em uso.

Um dos processos mais flexíveis do decodificador é o da reconstrução do DPCM. Não apenas o valor de predição do primeiro bloco de cada linha pode ser alterado, mas também outros detalhes sobre a forma como esse valor de predição é gerado. No capítulo anterior citamos a função *processDPCM*, que consta na biblioteca de Processamento de Dados. Dois argumentos desta função são controlados diretamente pelas opções de uso aqui apresentadas e podemos ver sua implementação no fluxograma da Figura 4.2.

O primeiro argumento controla o uso dos bits de correção do DPCM. Todo o processo de correção de erros do DPCM deve ser ativado através da opção *Correção DPCM*, mostrada na parte direta (marcada em verde) da Figura 4.1. Apesar de parecer estranho não se utilizar de um processo de correção diretamente ligado a decodificação

desde o princípio, é importante notar que isso se faz necessário para podermos comparar os resultados obtidos sem e com a correção de erros do DPCM. Assim como as demais opções a seguir, ela pode ser alterada em tempo real pelo usuário sem resultar em quaisquer erros de processamento.

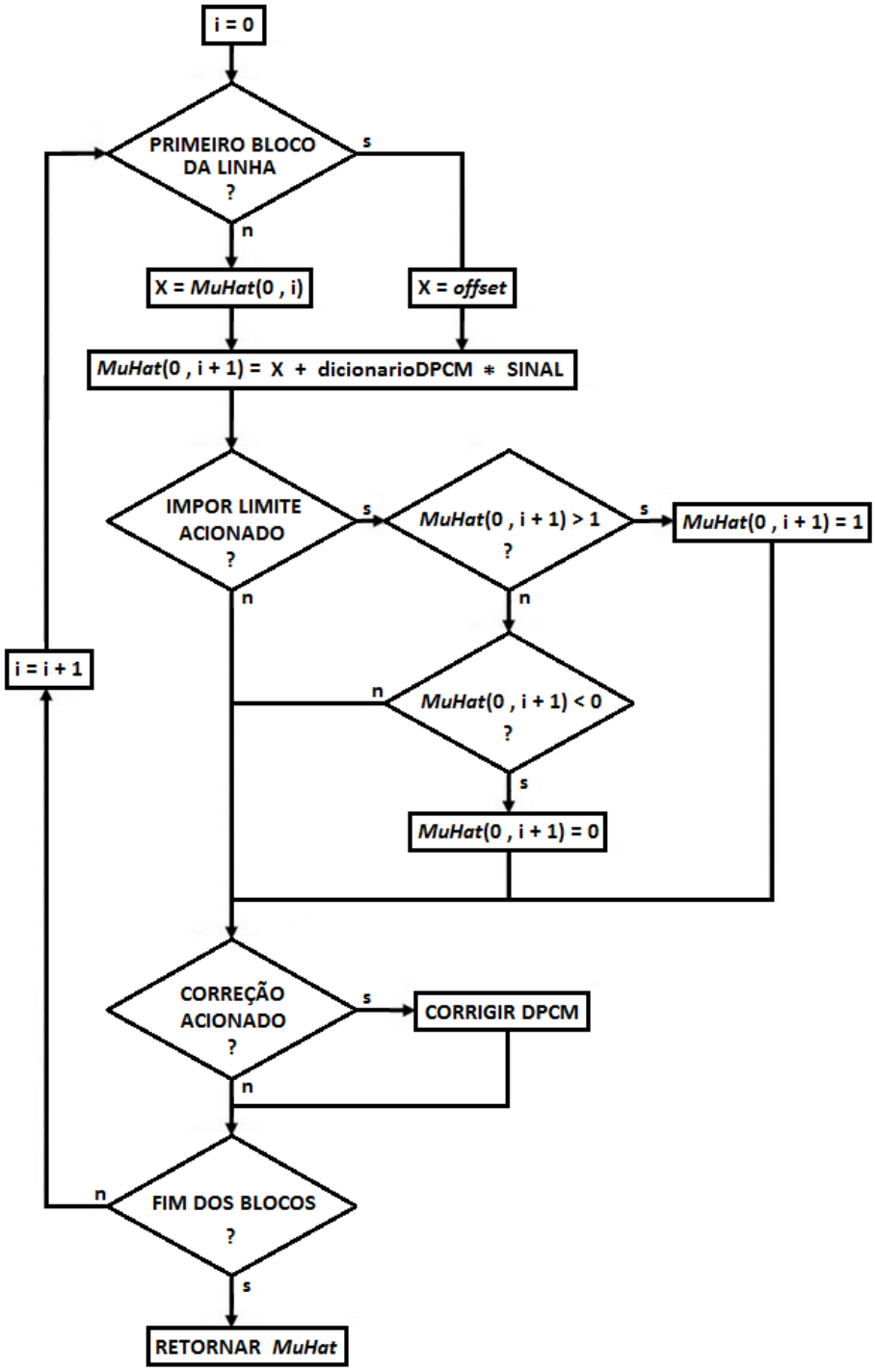

Figura 4.2 - Fluxograma da Decodificação do DPCM com Opções de Uso

O outro argumento é vinculado a opção de *Impor Limites*. Os resultados dos testes experimentais apresentaram valores maiores que um e menores que zero devido a erros do DPCM. Por esse motivo, a opção *Impor Limites* foi criada. Com a sua ativação, impedimos que os valores do DPCM sejam negativos ou que ultrapassem o valor unitário. É uma forma bruta de se tentar corrigir os problemas encontrados na prática. O fluxograma apresentado na Figura 4.2 mostra os passos necessários para encontrar a variável Muhat dentro da decodificação do DPCM, incluindo as opções de *Correção DPCM* e *Impor Limites*.

As demais funções operam em situações mais abrangentes do que a captação do DPCM. A opção de *Limitar Corrente*, por exemplo, ativa um circuito para limitar a corrente usada para a predição do nível médio do bloco seguinte. Ela pode, inclusive, ser limitada a zero. Com seu acionamento, enviamos um comando ao dispositivo que irá limitar o valor de predição do DPCM a um nível máximo de corrente. O valor dessa corrente depende de uma corrente de referência, que é definida por um potenciômetro na placa de testes do imageador. É uma opção importante para a execução de alguns testes de funcionamento. Fazendo a corrente de referência igual a um valor próximo de zero, o circuito que codifica o valor médio do bloco de pixels passa a trabalhar como um PCM (*pulse-code modulation*) de três bits.

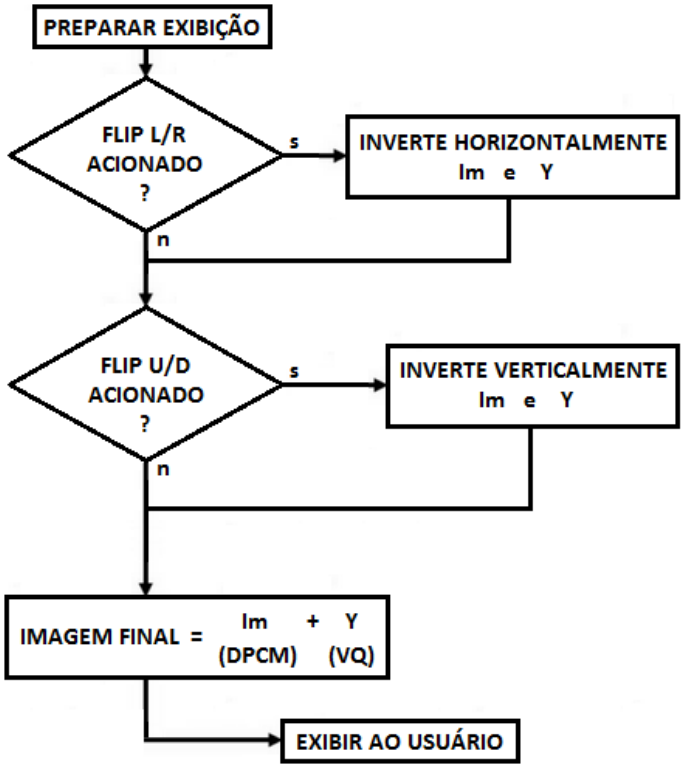

Figura 4.3 - Fluxograma da Exibição ao Usuário com *Flips* Implementados

As opções *Flip Horizontal* e *Flip Vertical*, implementadas seguindo a lógica apresentada na Figura 4.3, afetam a forma como nós vemos as imagens na tela do computador. Podemos alterar a posição do alvo na placa de testes e muitas vezes necessitamos rotacionar a placa de testes, para ajudar em testes e facilitar o ajuste de parâmetros. A nossa noção espacial em relação aos alvos e em relação à maneira como estes são projetados no plano focal do imageador através da lente pode ser perdida. As opções *Flip Horizontal* e *Flip Vertical* compensam rotações aplicadas pelo usuário ao movimentar o alvo, garantindo que ele tenha controle sobre seus referenciais espaciais e possa mover os alvos com maior facilidade.

Existindo a necessidade de estudar a imagem capturada mais a fundo e com mais recursos matemáticos, utilizamos a opção *REC*. Sua função é salvar os dados recebidos do imageador, sem alterações, em um arquivo de texto. Essa opção garante a portabilidade completa do projeto anterior para o atual, uma vez que podemos utilizar o MATLAB para analisar esses dados da mesma maneira como era feita antes. Para evitar a produção exagerada de arquivos de texto, entretanto, essa opção se limita a salvar apenas dez capturas feitas, que podem ser sobrescritas caso a opção seja novamente acionada. Isso se deve ao fato de que cada imagem é armazenada em um arquivo distinto.

4.2 – Lógica de Comandos

Uma tarefa importante deste projeto foi a de repensar completamente como as diferentes partes interagiam entre si e com os pedidos do usuário. Transformar os códigos em MATLAB do sistema antigo para o novo foi mais do que uma conversão para a linguagem C++. Ocorreu uma redução direta de possíveis congestionamentos de informação dentro da comunicação interna durante a decodificação. Em sua versão inicial, o microcontrolador enviava diversas leituras ao computador e estas eram armazenadas em arquivo de texto. Entretanto, o MATLAB utilizava apenas uma pequena fração delas, já que seu tempo de processamento era, em comparação ao do imageador, muito longo.

Como podemos ver na Figura 4.4, o isolamento entre os códigos de decodificação e a interface acabava por acarretar uma perda significativa de eficiência. Não havia como emparelhar as execuções das duas metades do projeto: cada uma trabalhava em

sua própria velocidade e a exibição ao usuário era limitada ao ritmo do processamento em MATLAB.

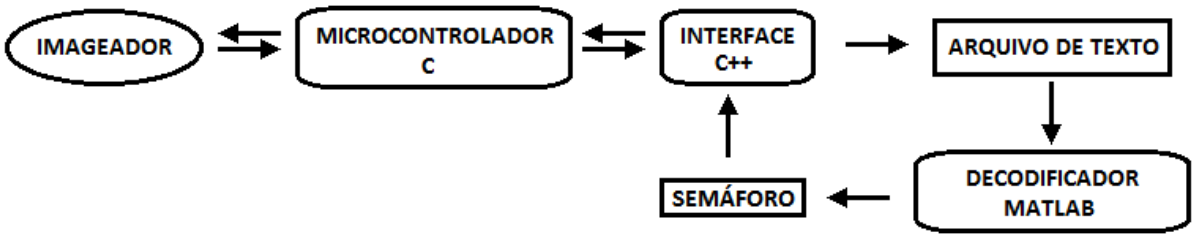

Figura 4.4 - Estrutura Antiga do Processamento

Além disso, existia um problema quanto à leitura e a escrita em arquivos de texto. Como não havia sincronização, caso a interface escrevesse no arquivo de texto no mesmo instante em que o código MATLAB estivesse fazendo a leitura do mesmo arquivo, ocorreria um erro e o programa seria interrompido de forma abrupta.

Esse problema ocorria com frequência durante tomadas de dados junto ao imageador. Isso se deve à quantidade enorme de capturas que eram necessárias em cada teste do projeto e pelo longo tempo em que ficavam funcionando. Mesmo uma pequena probabilidade acarreta vários erros quando damos um número elevado de oportunidades para tal evento.

A necessidade de um semáforo, como chamamos um bit de controle para o sistema, também foi um dos requisitos para que a versão antiga fosse funcional. Ele foi a forma encontrada para o código em MATLAB notificar a interface de que estava ou não utilizando o arquivo. Através deste semáforo, tínhamos conseguido reduzir as chances da interface gráfica acessar o arquivo de texto enquanto o MATLAB estava lendo. Entretanto, ele não foi capaz de resolver esse problema completamente. Isso se deve ao fato das leituras feitas pelo MATLAB não serem impedidas pelo semáforo. Apesar das leituras ocorrerem com uma frequência menor do que as escritas, existe a possibilidade do decodificador alterar o semáforo enquanto a interface já está requisitando o acesso ao arquivo, ocasionando num erro abrupto. Além disso, a utilização de um arquivo de texto como meio de comunicação entre interface e decodificador também pode causar lentidão ao sistema, pois as funções de tratamento de arquivos são lentas.

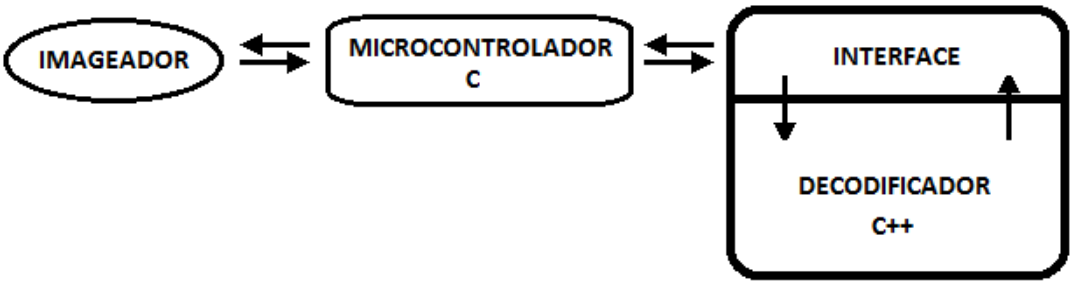

Figura 4.5 - Estrutura Melhorada

A solução encontrada já foi bastante discutida neste projeto: unificar as linguagens utilizadas e garantir que a interface e a decodificação ocorram dentro de um mesmo programa, como vislumbrado na Figura 4.5. Enquanto o uso da linguagem C++ proporciona um aumento da eficiência e da velocidade de processamento, a unificação do código da interface gráfica com o decodificador permite que eles se comuniquem diretamente e de maneira controlada. Eliminamos a necessidade de arquivos intermediários e flags de controle, assim como garantimos que todo o processo esteja em perfeita sintonia e sem desperdícios de capturas feitas pelo imageador.

A estrutura interna que unifica interface e decodificador facilita a execução de todas as demais melhorias apresentadas. Todas as operações projetadas, que serão apresentadas adiante, e que implementam variadas formas de se testar o imageador, têm como pré-requisito esse recurso da comunicação eficiente e rápida entre as duas partes agrupadas.

O comunicação entre elas funciona de maneira muito prática: duas *threads*, assíncronas entre si, são acionadas de maneira quase concomitante. *Thread* é a forma de um mesmo processo deixar de trabalhar sequencialmente e passar a ser capaz de realizar tarefas de maneira paralela. O funcionamento da *thread* é como a execução de um segundo programa, porém, de maneira mais simples e interligada ao processo que criou a *thread*.

Enquanto a primeira *thread* acionada implementa a interface e permite que os comandos selecionados pelo usuário sejam ativados instantaneamente, a outra fica encarregada de processar esses mesmos pedidos e executar o pós-processamento e decodificação da imagem, seguindo a lógica apresentada na Figura 4.6. Manter essas duas *threads* separadas garante que tanto a interface quanto o outro código possam funcionar ao mesmo tempo, sem que o processamento de uma interrompa ou atrase a outra.

A interface pode interferir no funcionamento da *thread* de decodificação. Essa dita interferência é feita de maneira direta com relação às variáveis de operação contudo, é na *thread* de decodificação e pós-processamento que é definido quando os valores e comandos recebidos serão utilizados. Isso impede que erros possam surgir por causa da natureza assíncrona da comunicação entre a interface e o decodificador.

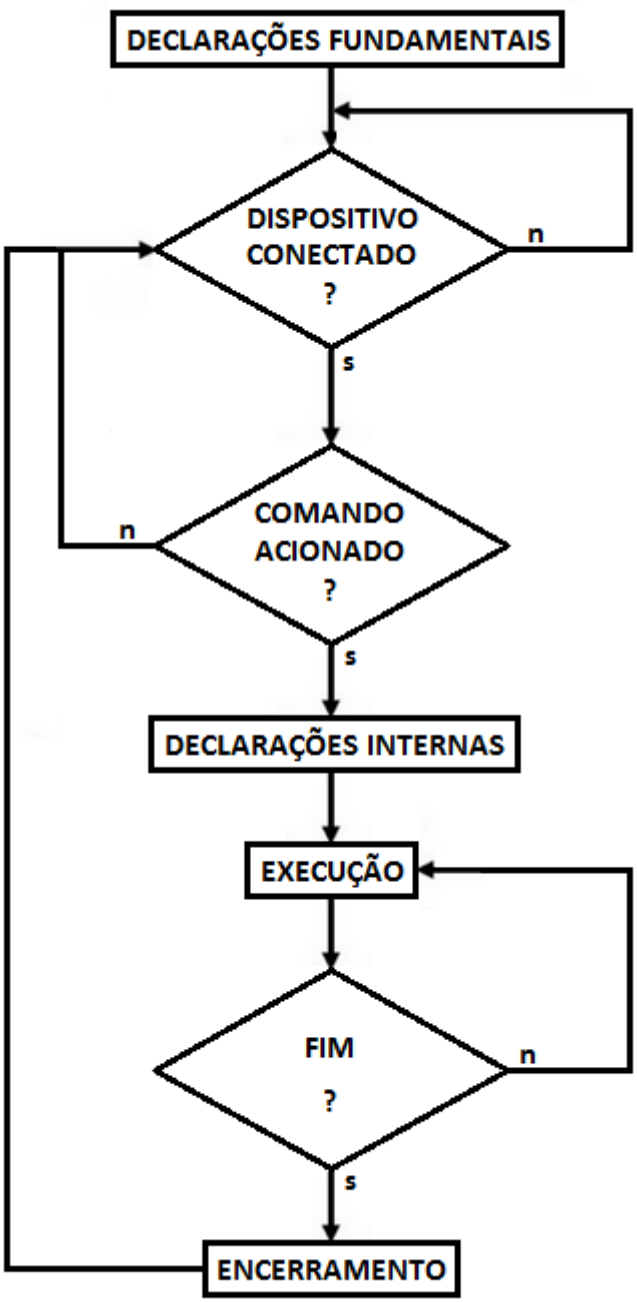

Figura 4.6 - Organização da *Thread* de Decodificação

Entretanto, o mesmo não ocorre para interferências feitas a partir da decodificação em direção à interface. O sistema impede que isso ocorra justamente para evitar a existência de erros por sincronismo. Para resolver essa limitação, uma variável do tipo *delegate* foi criada para intermediar essa comunicação. Ela tem o poder de invocar uma função interna que é capaz de executar diversas tarefas dentro da interface, principalmente no que tange alterar textos que são exibidos ao usuário com o conteúdo de matrizes. Dessa forma, a *thread* paralela é capaz de fazer um pedido à variável e essa, por sua vez, executa esse pedido quando a interface não estiver ocupada.

4.3 – Operações Implementadas

Com todas as inovações e melhorias postas em práticas, coube pensar em como melhorar os processos anteriores de análise do imageador, convertê-los para a nova linguagem e integrá-los à decodificação. Com as opções de uso que foram implementadas, também tivemos a chance de pensar em novas maneiras de se abordar antigos problemas com o imageador.

Desta sequência de pensamentos, surgiram os cinco comandos atuais implementados dentro do código da interface. Cada um deles executa uma rotina diferente que permite acessarmos o imageador e realizar diversos experimentos com ele. Da versão passada deste projeto, continuamos apenas com duas operações em comum: *Ativar Pixel Testes*, que foi mantido praticamente inalterado de sua versão original, graças à sua simplicidade, e *Ativar Bloco Principal*, que persistiu por ser a operação básica fundamental deste projeto. Todas as outras funções foram criadas para se encaixar nesta nova realidade em que não temos mais acesso às funcionalidades matemáticas do MATLAB para alterar, livremente, qualquer lógica desejada. Todas elas são apresentadas na Figura 4.7, que destaca onde se encontram na interface.

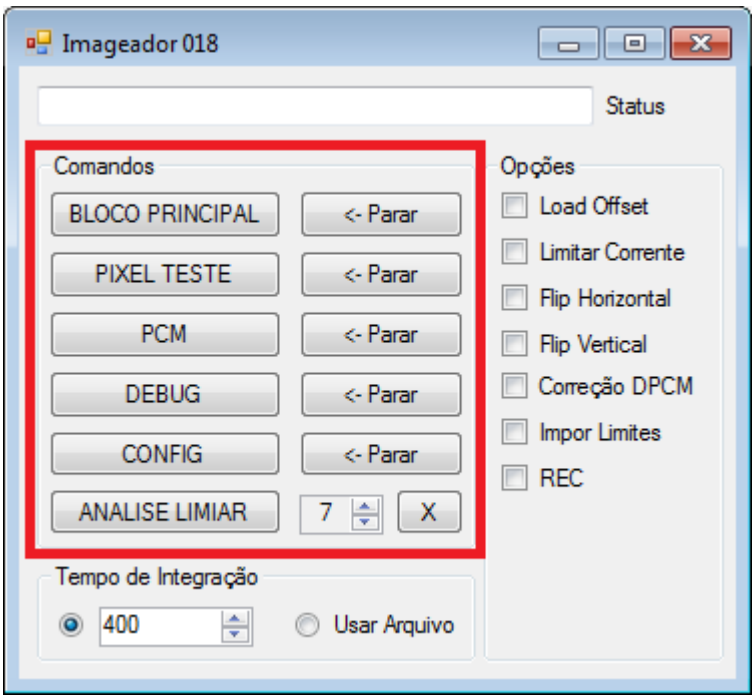

Figura 4.7 - Comandos da Interface

4.3.1 – *Bloco Principal*

O comando *Bloco Principal*, como já dito, é um dos comandos mais importantes implementados e é com base nele que outros comandos foram programados. Sendo ele a primeira operação programada para se testar os resultados do imageador, todos os que vieram a seguir tiveram início em adaptações e alterações dele. Este comando implementa a leitura da imagem vinda do microcontrolador, sua decodificação, pós-processamento e exibição ao usuário. Como sua estrutura é a mais fundamental deste projeto, usamos o seu tempo de execução como referencial para calcular o aumento de velocidade e eficiência que ocorreu entre o projeto anterior e esta versão apresentada.

O comando *Bloco Principal* implementa, e podemos visualisar isso na Figura 4.8, um *loop* através do qual enviamos ao microcontrolador diversos pedidos de leitura do imageador. Esse mesmo *loop* de leitura é repetido para a maioria dos comandos seguintes. Cada pedido é acompanhado pelo tempo de integração desejado e pela posição da linha de blocos da imagem que desejamos obter. Quando captamos todas as dezesseis linhas, armazenamos seus dados em uma única variável.

A partir dos dados lidos, realizamos a decodificação da imagem e o pós-processamento. Na decodificação, extraímos dos dados lidos as componentes de VQ e DPCM, assim como as informações necessárias para corrigir, caso desejado, os erros do

DPCM. Cada resultado é armazenado separadamente em duas matrizes 64×64, que correspondem às imagens de DPCM e de VQ. A imagem final é composta pela soma entre as duas matrizes e armazenada em uma matriz 64×64, a partir da qual será feito o *display* da imagem final.

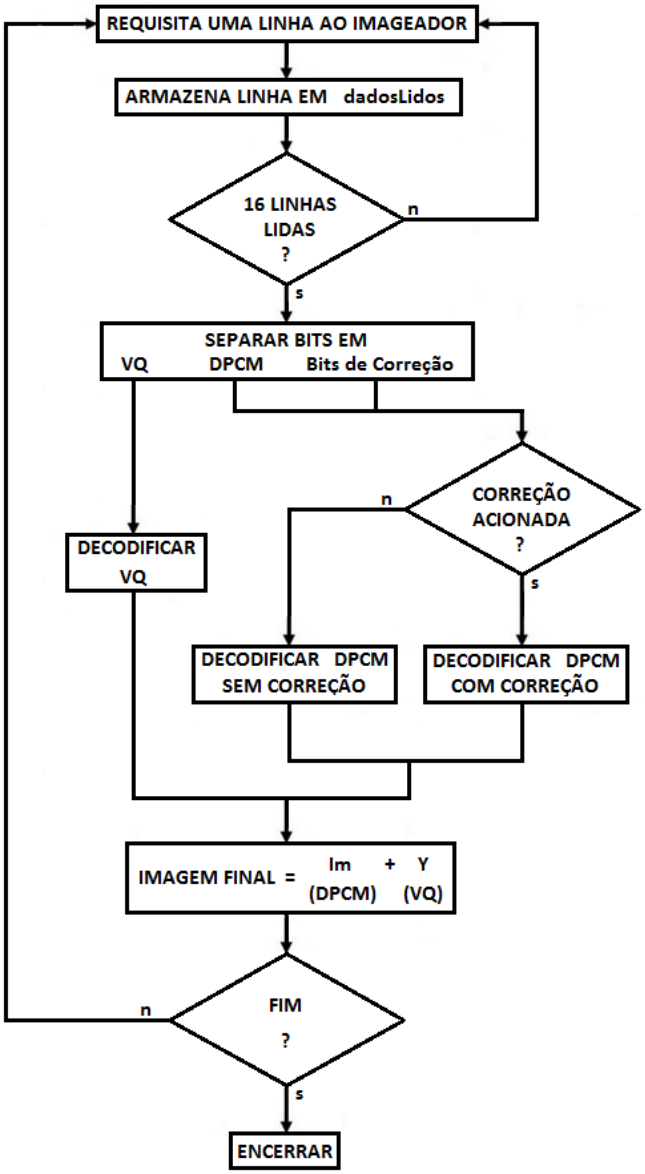

Figura 4.8 - Lógica Implementada no Comando *Bloco Principal*

4.3.2 – *Pixel Teste*

Dentro do imageador, existe um pixel separado da matriz principal e acessível para que possamos medir formas de onda em modo de tensão, que são importantes para

avaliar o seu funcionamento. Ele existe para estudarmos melhor o comportamento do pixel dentro de situações diversas. Por exemplo, podemos considerar diferentes aberturas da lente, tempos de integração ou diferentes comprimentos de onda da luz. Podemos estimar os efeitos que os erros ocorridos na fabricação do chip têm sobre o pixel de teste e sobre os demais sensores da matriz. A execução do *Pixel Teste* repete os sinais de controle do pixel em um *loop*, possibilitando a realização de medidas sobre esse pixel isolado, utilizando um osciloscópio.

Manter o pixel de teste trabalhando em *loop* é feito de forma muito simples. Não possuímos a necessidade de enviar as coordenadas do pixel que desejamos ler, então apenas enviamos repetidamente o tempo de integração desejado para a amostragem do pixel. A partir disso, basta acessarmos os pontos de interesse na placa de testes, utilizando um osciloscópio, e fazermos as medidas desejadas.

4.3.3 – *PCM*

Como descrito durante a explicação da decodificação do DPCM na Seção 3.3, existe uma segunda abordagem para a codificação da componente DC da imagem, em que não levamos em consideração o valor da diferença entre as médias dos blocos. Essa maneira de realizar a codificação e a decodificação é chamada de PCM. O comando *PCM* então realiza uma rotina de amostragens e tomadas de dados exatamente como o *Bloco Principal*. Entretanto, utilizamos aqui a função *processPCM*, implementada na biblioteca de Processamento de Dados. Para que esse comando funcione de forma correta, ele deve ser utilizado junto com o comando de limitar corrente e quando a corrente de limite é próxima de zero.

4.3.4 – *Debug*

Eliminar o MATLAB do projeto trouxe benefícios com relação à comunicação entre partes distintas, mas também resultou na perda de algumas funcionalidades importantes. Uma delas era a de conseguir visualizar uma matriz interna ao processo durante a execução do mesmo. Isso permitia alterarmos parâmetros da placa de testes ou mesmo mudarmos o tempo de integração e verificar em tempo real os efeitos numéricos desta alteração. Para não perdermos uma função tão importante, o comando *Debug* foi implementado.

O comando *Debug* expande a interface, como podemos ver na Figura 4.9, e apresenta um espaço onde algumas variáveis que fazem parte do processo de decodificação do DPCM podem ser selecionadas e exibidas. Como o processo se tornou muito rápido depois de todas as melhorias, essa função também implementa uma pequena pausa após cada captura. Esse tempo extra foi necessário para que o usuário tivesse tempo para ler os valores exibidos na tela do computador.

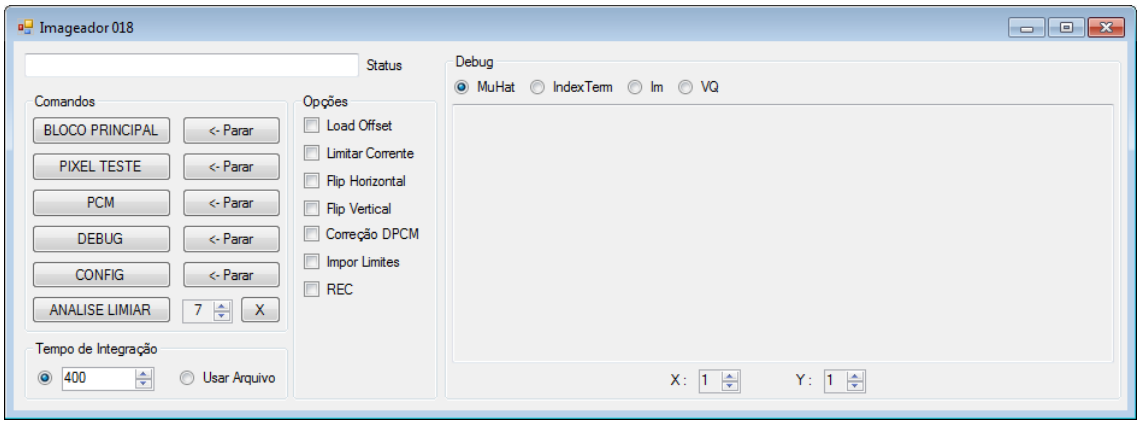

Figura 4.9 - Interface com Debug Acionado

Como algumas matrizes são muito grandes, como é o caso da imagem armazenada na variável *Im*, foi necessário implementar coordenadas cartesianas para podermos avaliar todo o seu conteúdo. Limitamos a exibição de valores a apenas uma sub-matriz ou bloco com 16×16 itens dentro da janela de *debug*. Caso uma matriz possua mais itens do que isso, as coordenadas X e Y servirão para selecionarmos qual é a sub-matriz ou bloco que desejamos visualizar. A variável *Im*, por exemplo, que possui 64×64 valores armazenados, necessitará que as coordenadas cartesianas variem de um a quatro.

4.3.5 – *Config*

Outro comando que altera o formato da interface é o *Config*, que representa mais uma customização dos processos utilizados do que uma rotina própria, diferente dos comandos explicados anteriormente. Ele serve para termos acesso à configurações mais específicas, que possuem características mais estéticas do que de intervenção direta no programa.

Como podemos observar na Figura 4.10, todos os itens que surgem com o comando *Config* podem ser desligados pelo usuário, a qualquer momento durante a operação da interface. Através dele, podemos alterar o brilho das componentes do VQ e

do DPCM, com o objetivo de melhorar a visualização. Essas alterações, assim como as outras aqui apresentadas, têm efeito direto sobre a exibição ao usuário e também alteram as matrizes que guardam os resultados de VQ e DPCM, assim como a imagem reconstruída. Portanto, se os valores originais enviados pelo imageador forem necessários é preciso salvá-los antes destas alterações.

Dentro desta nova tela também podemos selecionar opções de filtros diferentes que foram implementados com o objetivo de melhorar a qualidade da imagem reconstruída. Podemos dilatar a imagem ou causar um *blur*, que é um efeito de borramento sobre a imagem final, ou mesmo passar os valores de todos os pixels desta imagem por uma função tangente hiperbólica (com ganho e nível arbitrários), o que faz com que valores um pouco afastados do nível médio da imagem tendam a ficar ainda mais afastados da média na imagem resultante (e mais próximos dos extremos 0 ou 1), o que, por sua vez, aumenta o contraste da imagem.

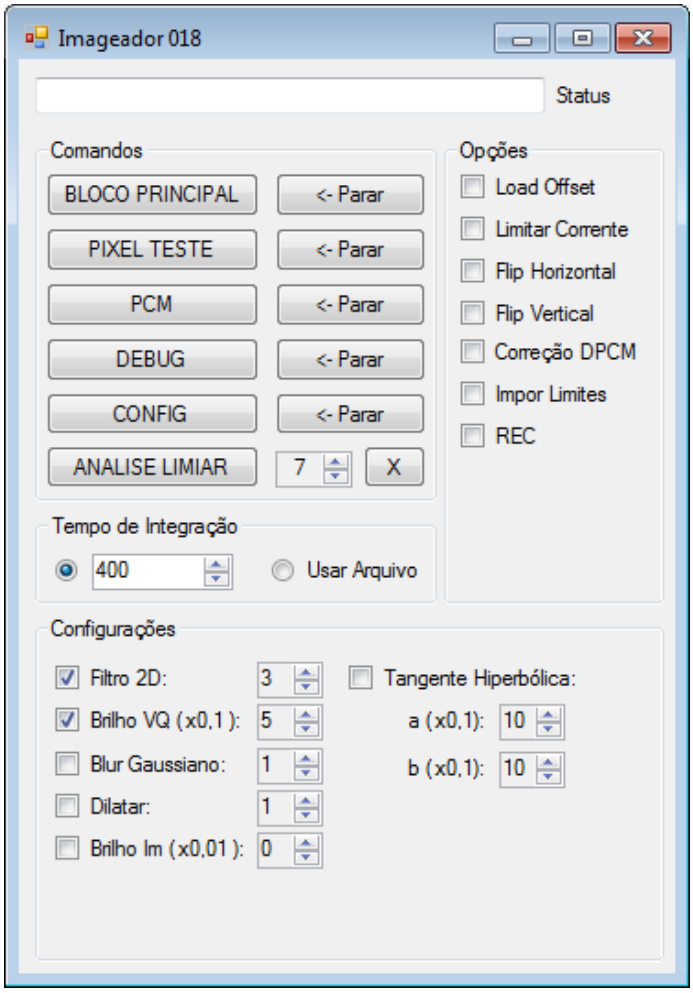

Figura 4.10 - Interface com Config Acionado

As opções de brilho no VQ e o filtro 2-D são consideradas *default*, e já surgem acionadas quando o programa é iniciado. A opção *Brilho VQ* ajuda na visualização das componentes decodificadas a partir dos bits do VQ, que representam os detalhes da imagem. A opção *Filtro 2-D* diminui o ruído da imagem e melhora a aparência da imagem final.

4.3.6 – *Análise Limiar*

O último comando a ser implementado foi *Análise Limiar*. Com o objetivo de identificar e modelar os erros do DPCM, foram feitos diversos testes. Esse comando ajudou na automatização de uma bateria de testes que consumiam muito tempo do usuário. Dentre os circuitos que compõem o DPCM, podemos testar o funcionamento do quantizador escalar determinando quando os índices do DPCM correspondem à entradas que passam de uma célula do quantizador escalar para a célula seguinte, ou seja, quando um limiar é ultrapassado. O limiar que define a divisão entre duas células do quantizador escalar é ultrapassado quando o índice em questão passa de um determinado valor (seis, por exemplo, sendo que esse valor é escolhido pelo usuário através da interface) para o valor seguinte (sete, neste exemplo).

Ao todo, existem dezesseis células no quantizador escalar, podendo ser positivas ou negativas e numeradas, em módulo, de um até oito. Os níveis de reconstrução obtidos a partir dos índices das células são determinados pelo dicionário do DPCM e são organizados em uma matriz com 16×16 valores. Cada um deles representará a média dos valores dos pixels de um bloco de 4x4 pixels dentro da matriz do imageador.

Os valores dos índices do DPCM dependem diretamente dos valores dos limiares, que são definidos através de uma tensão de referência, e dos valores dos pixels de cada bloco e do bloco anterior, que dependem da quantidade de luz incidente em cada pixel e do tempo de integração. O teste de calibração dos limiares de quantização escalar do DPCM é realizado utilizando uma imagem branca com incidência de luz constante. Ao limitarmos o valor previsto do bloco em zero, como quando trabalhamos em modo PCM, fazemos com que o índice do DPCM da coluna dois em diante não dependa do índice ou do valor de predição gerados pelo bloco anterior. Esse teste de calibração consiste em mantermos uma tensão fixa de referência enquanto o comando busca, para cada limiar e para cada voltagem de polarização associada à corrente de referência, qual

é o tempo de integração para o qual o índice gerado pelo quantizador muda de uma célula anterior ao limiar para uma célula posterior ao limiar. Como existe uma certa instabilidade nos valores dos índices, utilizamos uma aproximação para que o programa detecte o limiar. Quando, em um conjunto de imagens, contendo as 15 imagens capturadas mais recentemente, existe pelo menos 80% de incidência de valores acima do limiar, consideramos que o tempo de integração utilizado faz com que a média dos pixels do bloco ultrapasse o limiar em questão.

Para capturar e analisar esses dados de calibração, estamos com diversos testes em andamento. Procuramos uma relação linear entre tempo de integração e tensão de referência e, através dessa relação experimental, buscamos corrigir erros experimentais, preferencialmente, presentes na primeira coluna do DPCM e, se possível, presentes também nas demais colunas do DPCM. Além disso, podemos averiguar qual é a tensão de polarização (associada à corrente de referência para a definição dos limiares) na placa de testes que nos traz o melhor resultado em termos de qualidade da imagem reconstruída após o procedimento de calibração.

Capítulo 5

Resultados

Nem todas as melhorias propostas e aplicadas por este projeto podem ser quantificadas para que possamos medir, experimentalmente, os resultados obtidos. O foco dos aprimoramentos sempre foi o de melhorar a experiência que o usuário vivencia enquanto utiliza o programa durante os testes do imageador. Enquanto a velocidade de processamento é uma característica quantificável dos nossos resultados, a praticidade com que ele se encaixou às necessidades de bancada não é uma característica que possa ser medida, mas pode ser apontada como uma consequência subjetiva notável do acréscimo de qualidade ao programa.

Com a unificação de códigos, não observamos mais aqueles erros de operação que resultavam da comunicação entre a interface gráfica e o decodificador. Esse problema estava associado ao uso de um arquivo de texto que servia de intermediário para a passagem dos bits, como explicado no Capítulo 4. Com o novo sistema, a operação do imageador pode se estender por várias horas de uso sem que o processo seja interrompido indesejadamente por um erro. A unificação da interface gráfica e do decodificador em um mesmo código também permitiu que todas as capturas feitas pelo imageador sejam utilizadas. Isso permite que nenhuma leitura seja descartada, algo que ocorria constantemente no sistema anterior, enquanto o MATLAB realizava a decodificação e a interface estava lendo bits que não eram utilizados.

O controle do tempo de integração pelo usuário diretamente através da interface eliminou a necessidade de regravar o microcontrolador e recompilar a própria interface constantemente. Essa independência da regravação do microcontrolador leva à uma prática de trabalho mais constante, além de permitir a realização de testes experimentais em menor tempo e com menor esforço físico.

O avanço quantificável obtido foi no tempo total de processamento. Para medir esse valor selecionamos a função "Bloco Principal" e medimos a diferença entre o instante de tempo de entrada na iteração deste comando e o instante de tempo em que o

comando termina. Quando dividimos o número total de imagens capturadas e exibidas pelo tempo total de processamento medido, encontramos a velocidade de processamento ou, como também chamamos, a taxa de exibição.

Durante os testes iniciais com o novo código, antes que a decodificação da imagem fosse dividida em funções dentro da biblioteca Processamento de Dados, a taxa de exibição era igual a seis quadros por segundo. No sistema anterior, com o código em MATLAB, a taxa de exibição era de aproximadamente um quadro a cada segundo. A baixa taxa do sistema antigo dificultava os testes, pois o ajuste do foco, abertura ou posicionamento do alvo precisavam ser feitos de forma muito lenta. Essas dificuldades já não existem no novo sistema, pois sua taxa permite que esses ajustes sejam feitos e visualizados com atraso menor. Quando a lógica apresentada no Capítulo 4 foi completamente implementada e a decodificação começou a ser implementada por funções, essa taxa de exibição subiu para nove quadros por segundo. Trata-se de um resultado bastante estável, uma vez que nenhuma das variáveis de processamento ou opções de uso reduziu esse valor substancialmente.

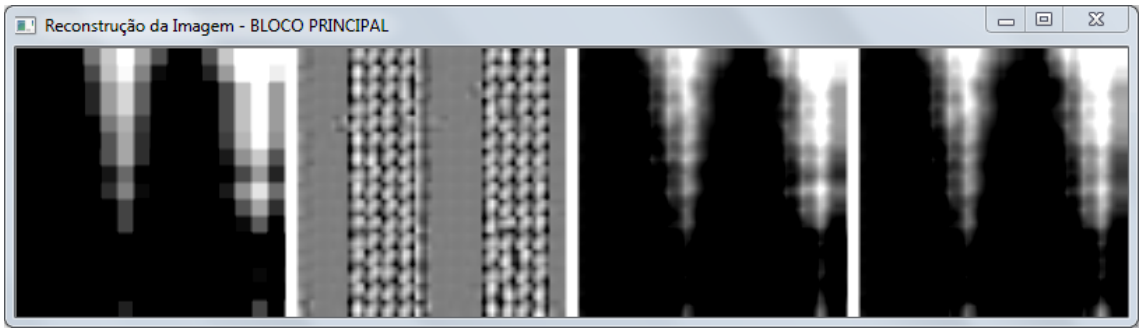

Figura 5.1 - Captura pelo Comando *Bloco Principal*

A implementação das opções de uso também surtiu em bons resultados para os testes feitos em bancada. As imagens podem ser refinadas com diversos métodos que permitem ao usuário utilizá-los ou não. Usamos como exemplo uma captura de um alvo chamado de "imagem com padrão de zebra vertical" ou, simplesmente, zebra vertical. Damos esse nome à uma imagem com uma sequência de colunas brancas seguidas de colunas pretas de mesmo espaçamento entre si. Utilizamos nessa captura um tempo de integração de 890 µs.

Como podemos ver na Figura 5.1, o DPCM (primeira sub-imagem da esquerda para a direita) desta captura ainda não está com a mesma qualidade que o VQ (segunda sub-imagem da esquerda para a direita) e, portanto, os testes atuais se focam justamente

em aperfeiçoarmos essa parte. Entretanto, podemos utilizar alguns recursos para melhorar um pouco a captura e a decodificação dos dados referentes ao DPCM.

Os bits de correção do DPCM podem ser acionados pela opção *Correção DPCM* na interface gráfica e melhorar um pouco a qualidade desta componente da imagem. O resultado dessa função pode ser visto na Figura 5.2. Como pode ser observado, a qualidade do DPCM melhora ligeiramente nesse método.

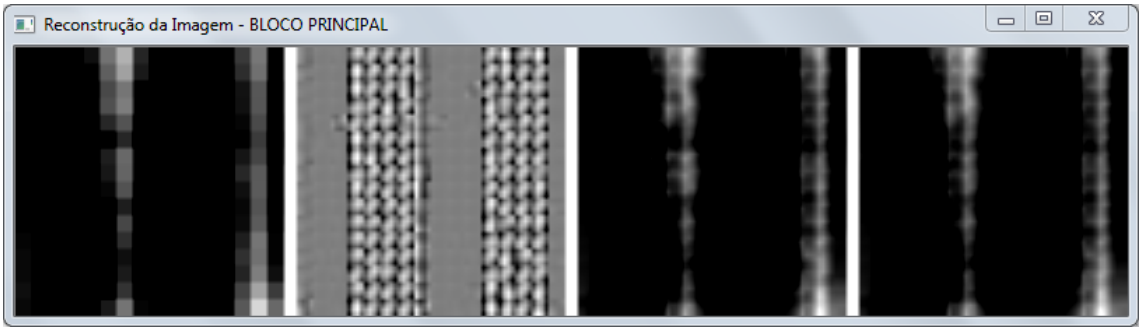

Figura 5.2 - Captura com *Correção DPCM* Acionado

Apesar do resultado ainda sofrer com uma baixa qualidade do DPCM, temos uma visão melhor das colunas brancas que compõem o alvo e que antes eram suprimidas por uma vasta mancha escura. Antes da correção de DPCM mostrada na Figura 5.2, sempre se observavam, associadas à captura de zebras verticais, colunas escuras mais largas do que as linhas brancas.

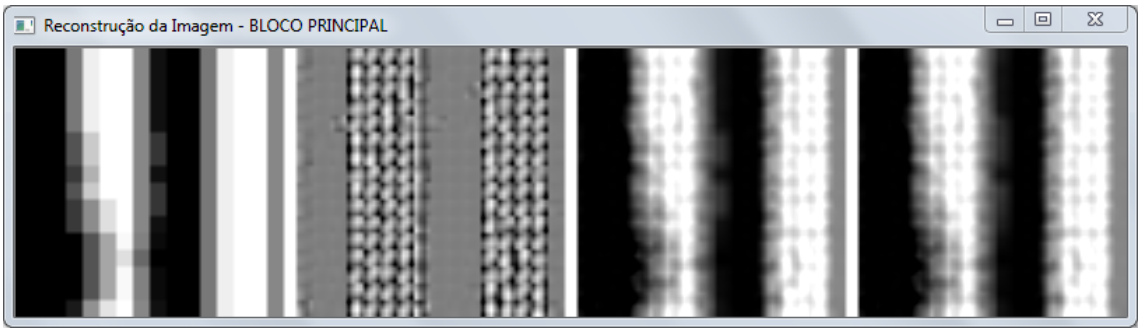

Figura 5.3 - Captura com *Impor Limites* Acionado

Uma segunda abordagem seria não acionarmos a correção do DPCM, mas sim impedirmos que os valores do DPCM estourem os limites que temos para uma imagem. Acionamos, então, a função *Impor Limites*, que implementa essa operação. Como podemos ver na Figura 5.3, os resultados são muito satisfatórios. Conseguimos observar

com facilidade a média dos blocos que o DPCM representa e conseguimos distinguir o objeto fotografado com qualidade na imagem final obtida.

As opções de filtros implementados na função *Config* modificam, obviamente, os resultados experimentais obtidos. Esses novos parâmetros podem ser observados em qualquer um dos comandos que a interface nos traz, o que dá meios para retocar os resultados experimentais.

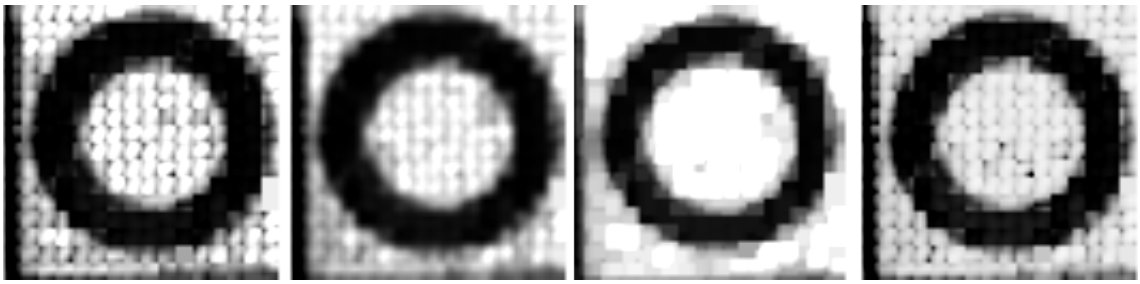

Figura 5.4 - Diferentes Efeitos de Filtros na Imagem Final

Na Figura 5.4 temos um exemplo prático de resultados retocados através de filtros 2-D. Capturamos um alvo contendo um círculo, levando em conta três possíveis filtros 2-D aplicados após a decodificação da imagem através do comando *PCM*. Temos aqui, respectivamente da esquerda para a direita, uma imagem exibida sem usar filtros, uma imagem com o *blur* gaussiano, uma imagem com a função *Dilate* e uma imagem na qual cada pixel é processado por uma função tangente hiperbólica com ganho e nível médio arbitrários. Cada uma dessas variantes modifica um pouco a imagem, permitindo controle sobre definição, contraste e efeitos de esfumaçamento.

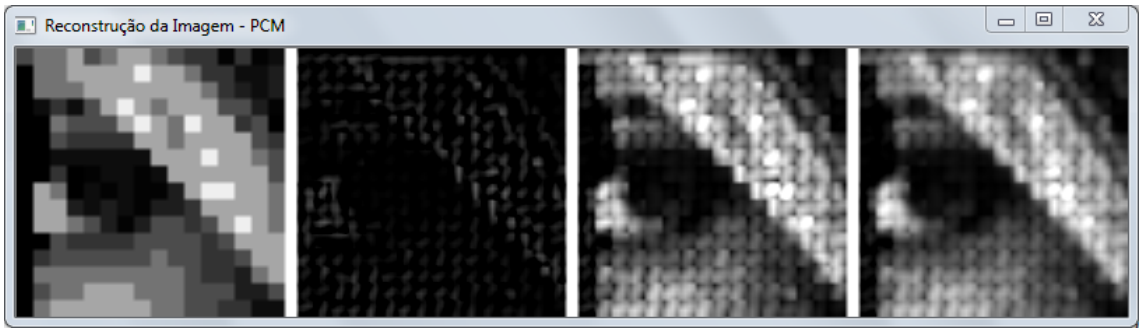

Figura 5.5 - Imagem Lena Sem Filtros-Padrão

Também podemos analisar uma imagem mais complexa e verificar outros efeitos que os parâmetros da interface têm sobre ela. Na Figura 5.5 temos a captura de uma imagem conhecida como "Lena", sem utilizar as opções de brilho no VQ e nem o *filtro2D* pertencentes à função *Config*. Essa captura foi feita através do comando *PCM*,

que até o momento apresenta os melhores resultados práticos. Percebemos que há uma dificuldade em se observar a resposta do VQ, enquanto diversos efeitos de blocagem ocorrem na imagem final (terceira sub-imagem, da esquerda para direita).

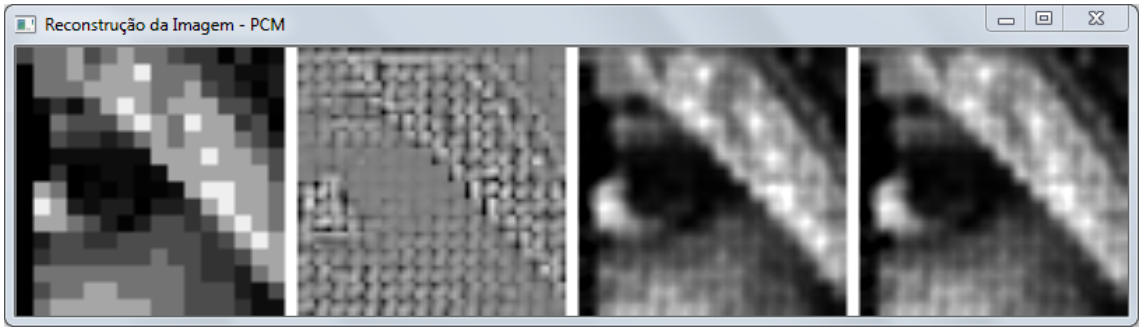

Figura 5.6 - Imagem Lena Com Filtros-Padrão

Os problemas de blocagem são reduzidos quando acionamos a opção padrão do filtro 2-D. A outra opção padrão, brilho do VQ, permite uma melhor visualização do resultado da decodificação dos bits do VQ. Os resultados na Figura 5.6 representam essa melhoria bastante visível de uma situação para outra quando estas opções-padrão estão acionadas. Como não causam uma mudança brusca nos resultados experimentais e melhoram nossa visibilidade, essas duas opções são sempre acionadas, cabendo ao usuário desligá-las caso deseje o contrário.

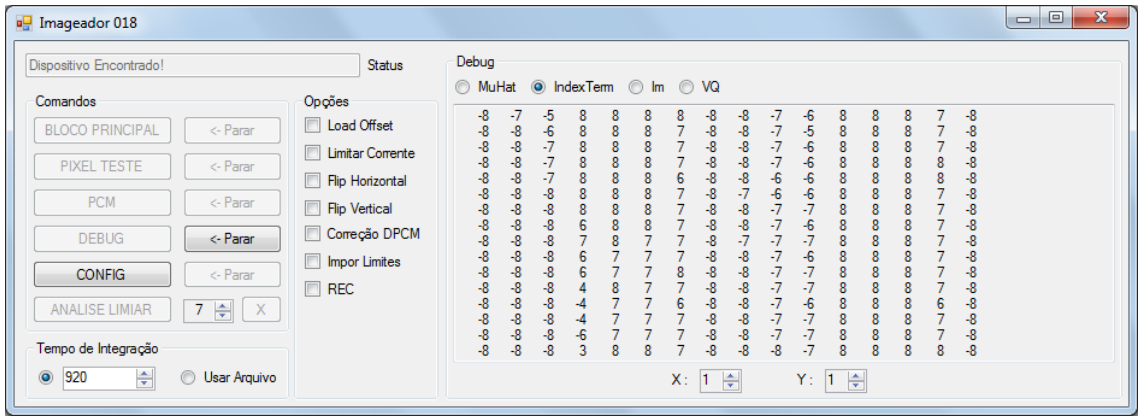

Figura 5.7 - Comando *Debug* Acionado e em Execução

A implementação do comando *Debug* foi um grande êxito para os testes em bancada e melhorou a facilidade de uso da interface atual, com respeito à versão anterior da interface. Se esta função não tivesse sido efetiva, não poderíamos substituir a versão anterior nos testes experimentais. A Figura 5.7 apresenta a interface funcionando

após o acionamento do comando *Debug*, e apresentando ao usuário a matriz de índices do DPCM.

Por fim, obtivemos um resultado menos prático para se apresentar, mas que se mostrou uma das principais melhorias deste projeto. Com a unificação de códigos e a organização do decodificador de uma maneira ordenada e comentada em blocos, tornou-se muito mais rápido e prático implementar novas melhorias para o desenvolvimento do projeto. Ideias que precisavam analisar variáveis simultâneas puderam ser implementadas com facilidade dentro deste novo código.

Capítulo 6

Conclusões

Nesse projeto foi apresentado um novo sistema para a realização de testes com um imageador CMOS. Esse sistema, que em sua versão anterior necessitava de uma interface gráfica em C++ e um decodificador em MATLAB, foi implementado utilizando somente a linguagem C++ e com uma nova lógica de funcionamento focada em eficiência. Diversas melhorias puderam ser observadas, como o aumento da taxa de exibição e a possibilidade de escolha do tempo de integração do pixel a partir da interface gráfica.

Com a finalização deste trabalho, os resultados se mostraram muito agradáveis e superaram as expectativas iniciais quanto ao aumento da taxa de exibição. A interface que foi desenvolvida neste projeto, que inicialmente era vista como uma alternativa para a interface do projeto anterior, se provou eficaz para realizar as mesmas tarefas em um tempo de execução reduzido. Também permite que novas funções sejam implementadas com maior facilidade e que novos testes sejam executados com maior controle e menos trabalho.

O completo cancelamento dos erros de execução, solucionados graças à nova estrutura lógica, mostrou-se duradouro e está vinculado à simplicidade dos novos programas. Indo além da correção de erros, a unificação dos códigos levou à comunicação direta entre todas as etapas de projeto.

A organização do código e sua estrutura em nível de comandos também propiciou um ambiente em que novas ideias podem facilmente ser implementadas e aproveitadas em outros projetos. Espera-se que as melhorias apresentadas aqui sejam apenas um ponto de partida para uma plataforma amigável e adequada para os testes em imageadores realizados no laboratório PADS.

Como diversos testes surgem a cada semana, esperamos que este projeto seja melhorado de diversas novas formas e que versões mais atualizadas e com novos recursos possam surgir num futuro próximo.

Bibliografia

[1]OHTA, J. *Smart CMOS Image Sensors and Applications*, CRC Press.

[2]OLIVEIRA, F. *Imageador CMOS Utilizando Tecnologia de 0.18 um para Captura e Compressão de Imagens no Plano Focal*, Dissertação de Mestrado, COPPE/UFRJ, dezembro de 2013.

[3]RAZAVI, B., *Fundamentals of Microeletronics*, Editora Wiley, 2006.

[4]YADID-PECHT, O.; ETIENNE-CUMMINGS, R. *CMOS Imagers: From Phototransduction to Image Processing*, Kluwer Academic Publishers, 2004

[5]OLIVEIRA, F.; HASS, H.; GOMES, J.; et al. “CMOS Imager With Focal-Plane Analog Image Compression Combining DPCM and VQ”, em *Circuits and Systems I: Regular Papers, IEEE Transactions on, v. 60, n. 5*, pp. 1331-1334, maio de 2013.

[6]OLIVEIRA, F.; LOPES, T.; GOMES, J.; BARÚQUI, F.; PETRAGLIA, A. "Focal-plane image encoder with cascode current mirrors and increased vector quantization bit rate", em *Anais do 29th Symposium on Integrated Circuits and Systems Design (SBCCI 2016)*, Belo Horizonte, Brasil, agosto de 2016.

[7]MICROCHIP TECHNOLOGY INC. *Microchip Libraries for Applications.* Disponível em: <http://www.microchip.com/mplab/microchip-libraries-for-applications>. Acesso em: 25 de outubro de 2016.

[8]OPENCV. *Open Source Computer Vision*. Disponível em: <www.opencv.org>. Acesso em: 12 de novembro de 2016.

Apêndice A

Códigos da Interface

Define.h

```
#ifndef DEFINE_H

	#define DEFINE_H

	#define ATIVAR_BLOCO_PRINCIPAL				0x31
	#define ATIVAR_AD					0x32
	#define ATIVAR_PIXELS_TESTE				0x33
	#define CHIP_OFF					0x34
	#define TESTE_FPN					0x35

	#define N_BLOCOS					16*16
	#define N_LINHAS_BLOCOS					16
	#define N_COMPONENTES					5

	#define WINDOW_WIDTH_DEBUG				941
	#define WINDOW_WIDTH_NORMAL				374
	#define WINDOW_HEIGHT_NORMAL				344
	#define WINDOW_HEIGHT_CONFIG				535

	#define OFFSET_FILE					0
	#define TINT_FILE					1
	#define FILE_NOT_FOUND					404
	#define FILE_NOT_OK					405

	#define TINT_VALUE					10

	#define MAT_TEXT					0
	#define MAT_16x16					1
	#define MAT_64x64					2
	#define BACK_TO_NORMAL					3
	#define REC_OFF						4


#endif
```

MyForm.cpp

```
#include "MyForm.h"

using namespace ProjetoFinal;

[STAThreadAttribute]
int main(array<System::String^>^ args)
{
	// Enabling Windows XP visual effects before any controls are created
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	// Create the main window and run it
	Application::Run(gcnew MyForm());
	return 0;
}
```

ProcessamentoDeDados.h

```
#ifndef PROCESSAMENTODEDADOS_H
#define PROCESSAMENTODEDADOS_H

#include "Define.h"

#include <iomanip>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace System::Windows::Forms;

class ProcessamentoDeDados
{
	public:

		static void processData(cv::Mat aux_Y, cv::Mat aux_DC, cv::Mat& S1, cv::Mat& B1, cv::Mat& D1,
					vector<cv::Mat>& DC1);

		static cv::Mat1d cad2indexFernanda(cv::Mat mat);
		static cv::Mat1d getPhat(cv::Mat1d Cnovo, cv::Mat B1, cv::Mat S1);
		static cv::Mat1d getY(cv::Mat1d Xhat);

		static int gray2Term(cv::Mat mat);
		static void processDPCM(cv::Mat D1, vector<cv::Mat> DC1, cv::Mat C_dpcm, cv::Mat offset, bool check,
					bool limit, cv::Mat1d& MuHat, int(&indexTerm)[N_BLOCOS]);
		static void processPCM(cv::Mat D1, vector<cv::Mat> DC1, cv::Mat C_dpcm, cv::Mat offset, bool check,
					cv::Mat1d& MuHat, int(&indexTerm)[N_BLOCOS]);
		static cv::Mat1d getIm(cv::Mat1d MuHat);

		static void carregarByte(unsigned char byte, cv::Mat mat, int position, int numByte);
		static cv::Mat1i indexTermToMat(int indexTerm[]);
		static System::String^ matToString(cv::Mat mat, int precision);
		static cv::Mat matlabReshape(const cv::Mat mat, int row, int col);

		static cv::Mat1d prepareFinalImage(cv::Mat1d Im, cv::Mat1d Y, cv::Mat1d imagemFinal,
						cv::Mat1d imagemMedia);

};

#endif
```

ProcessamentoDeArquivos.h

```
#ifndef PROCESSAMENTODEARQUIVOS_H
#define PROCESSAMENTODEARQUIVOS_H

#include "Define.h"

#include <fstream>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace std;
using namespace System::Windows::Forms;

class ProcessamentoDeArquivos
{
public:

	static void imprimirMat(cv::Mat mat, string nomeArq);

	static cv::Mat1d lerMatC();
	static cv::Mat1d lerDados();

	static bool carregarInfo(string nomeArq, int numDados, cv::Mat1d& result);
	static void sendMessageError(unsigned fileName, unsigned typeError);


};

#endif
```

ProcessamentoDeArquivos.cpp

```
#include "processamentoDeArquivos.h"

//-----------------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Imprime a matriz seleciona para um arquivo de texto.
/// </summary>
/// <param name="matrix"> Matriz a ser impressa </param>
/// <param name="nomeArq"> Nome do arquivo .txt </param>
void ProcessamentoDeArquivos::imprimirMat(cv::Mat matrix, string nomeArq)
{
	fstream matFile;
	matFile.open(nomeArq, fstream::out);

	for (int i = 0; i < matrix.rows; i++){
		for (int j = 0; j < matrix.cols; j++){
			if (matrix.at<double>(i, j) >= 0){
				matFile << " ";
				matFile << matrix.at<double>(i, j);
				matFile << " ";
			}
			else{
				matFile << matrix.at<double>(i, j);
				matFile << " ";
			}
		}
		matFile << "\n";
	}

	matFile.close();
}
//-----------------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Lê o arquivo referente ao dicionário DPCM (C.txt) e carrega seus valores.
/// </summary>
/// <return> Matriz com valores lidos </return>
cv::Mat1d ProcessamentoDeArquivos::lerMatC(){

	cv::Mat1d result(5, 512, CV_64F);
	string line;
	vector<string> tokens;
	fstream matFile;
	matFile.open("C.txt", fstream::in);
	size_t pos = 0;
	string delimiter = ",";
	int index;

	while (getline(matFile, line))
	{
		while ((pos = line.find(delimiter)) != std::string::npos)
		{
			tokens.push_back(line.substr(0, pos));
			line.erase(0, pos + delimiter.length());
		}
		tokens.push_back(line);
	}

	index = 0;

	for (int i = 0; i < result.rows; i++){
		for (int j = 0; j < result.cols; j++){
			result.at<double>(i, j) = atof(tokens.at(index).c_str());
			index++;
		}
	}

	matFile.close();

	return result;
}

/// <summary>
/// Lê o arquivo referente a uma tomada de dados feitos pelo imageador e carrega seus valores por questões de teste.
/// </summary>
/// <return> Matriz com valores lidos </return>
cv::Mat1d ProcessamentoDeArquivos::lerDados(){

	cv::Mat1d result(1, 4800, CV_64F);
	string line;
	vector<string> tokens;
	fstream matFile;
	matFile.open("A.txt", fstream::in);
	size_t pos = 0;
	string delimiter = ",";
	int index;

	while (getline(matFile, line))
	{
		while ((pos = line.find(delimiter)) != std::string::npos)
		{
			tokens.push_back(line.substr(0, pos));
			line.erase(0, pos + delimiter.length());
		}
		tokens.push_back(line);
```

```
	}

	index = 0;

	for (int i = 0; i < result.rows; i++){
		for (int j = 0; j < result.cols; j++){
			result.at<double>(i, j) = atof(tokens.at(index).c_str());
			index++;
		}
	}

	matFile.close();

	return result;
}
//-----------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Carrega as informações armazenadas dentro de um determinado arquivo de textos.
/// </summary>
/// <param name="nomeArq"> Nome do arquivo a ser lido </param>
/// <param name="numDados"> Número de valores a ser lido </param>
/// <param name="result"> Endereço para a matriz que armazenará os valores </param>
/// <return> Retorna o sucesso ou fracasso da operação </return>
bool ProcessamentoDeArquivos::carregarInfo(string nomeArq, int numDados, cv::Mat1d& result){

	result = cv::Mat::zeros(1, numDados, CV_64F);
	string line, val1;
	int i = 0;
	bool sucess = true;
	fstream matFile;

	matFile.open(nomeArq, fstream::in);
	while (getline(matFile, line) && i < numDados)
	{
		for (unsigned k = 0; k < strlen(line.c_str()); k++){
			if (!(line[k] >= '0' && line[k] <= '9' || line[k] == ' ' || line[k] == '.'))
				sucess = false;
		}
		result.at<double>(0, i) = atof(line.c_str());
		i++;
	}

	matFile.close();

	return sucess;
}

/// <summary>
/// Envia uma mensagem de erro ao usuário.
/// </summary>
/// <param name="fileName"> Código do arquivo com problemas </param>
/// <param name="typeError"> Código do Erro </param>
void ProcessamentoDeArquivos::sendMessageError(unsigned fileName, unsigned typeError){

	if (fileName == OFFSET_FILE){
		if (typeError == FILE_NOT_FOUND)
			MessageBox::Show(L"Arquivo \"Offset.txt\" não foi encontrado.", L"AVISO: Falta Arquivos de
				Configuração!", MessageBoxButtons::OK, MessageBoxIcon::Warning);
		if (typeError == FILE_NOT_OK)
			MessageBox::Show(L"Arquivo \"Offset.txt\" foi encontrado mas apresenta problemas.", L"AVISO:
				 Erro em Arquivos de Configuração!", MessageBoxButtons::OK,
				 MessageBoxIcon::Warning);
	}
	if (fileName == TINT_FILE){
		if (typeError == FILE_NOT_FOUND)
			MessageBox::Show(L"Arquivo \"Tempo_Integracao.txt\" não foi encontrado.", L"AVISO: Falta
				Arquivos de Configuração!", MessageBoxButtons::OK, MessageBoxIcon::Warning);
		if (typeError == FILE_NOT_OK)
			MessageBox::Show(L"Arquivo \"Tempo_Integracao.txt\" foi encontrado mas apresenta problemas.",
				L"AVISO: Erro em Arquivos de Configuração!", MessageBoxButtons::OK,
				MessageBoxIcon::Warning);
	}

}
//-----------------------------------------------------------------------------------------------------------------------------------//
```

ProcessamentoDeDados.cpp

```
#include "processamentoDeDados.h"

//-----------------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Separa os vetores de bytes recebidos em variáveis menores, organizadas em função de uso e utilidade.
/// </summary>
/// <param name="aux_Y"> Bits que compõem a imagem </param>
/// <param name="aux_DC"> Bits de correção do DPCM </param>
/// <param name="S1"> Endereço da variável para os bits do sinal de VQ </param>
/// <param name="B1"> Endereço da variável para os bits do módulo de VQ </param>
/// <param name="D1"> Endereço da variável para os bits do DPCM </param>
/// <param name="DC1"> Endereço do vetor de matrizes que representam a correção do DPCM </param>
void ProcessamentoDeDados::processData(cv::Mat aux_Y, cv::Mat aux_DC, cv::Mat& S1, cv::Mat& B1, cv::Mat& D1,
                                       vector<cv::Mat>& DC1){

        DC1.at(0) = cv::Mat::zeros(4, 1, CV_64F);
        DC1.at(1) = cv::Mat::zeros(4, 1, CV_64F);
        DC1.at(2) = cv::Mat::zeros(4, 1, CV_64F);
        cv::Mat aux;
        int index = 0;
        for (int i = 0; i < 16; i++){
                for (int j = 0; j < 16; j++){

                        index = 18 * j;
                        if (i == 0 && j == 0){
                                cv::transpose(aux_Y(cv::Range(0, 1), cv::Range(0, 5)), S1);
                                cv::flip(S1, S1, 0);

                                cv::transpose(aux_Y(cv::Range(0, 1), cv::Range(5, 14)), B1);
                                cv::flip(B1, B1, 0);

                                cv::transpose(aux_Y(cv::Range(0, 1), cv::Range(14, 18)), D1);
                                cv::flip(D1, D1, 0);
                        }
                        else {
                                cv::transpose(aux_Y(cv::Range(i, i + 1), cv::Range(index, index + 5)), aux);
                                cv::flip(aux, aux, 0);
                                cv::hconcat(S1, aux, S1);

                                cv::transpose(aux_Y(cv::Range(i, i + 1), cv::Range(index + 5, index + 14)), aux);
                                cv::flip(aux, aux, 0);
                                cv::hconcat(B1, aux, B1);

                                cv::transpose(aux_Y(cv::Range(i, i + 1), cv::Range(index + 14, index + 18)), aux);
                                cv::flip(aux, aux, 0);
                                cv::hconcat(D1, aux, D1);
                        }
                }

                cv::transpose(aux_DC(cv::Range(i, i + 1), cv::Range(0, 4)), aux);
                cv::flip(aux, aux, 0);
                cv::hconcat(DC1.at(2), aux, DC1.at(2));

                cv::transpose(aux_DC(cv::Range(i, i + 1), cv::Range(4, 8)), aux);
                cv::flip(aux, aux, 0);
                cv::hconcat(DC1.at(1), aux, DC1.at(1));

                cv::transpose(aux_DC(cv::Range(i, i + 1), cv::Range(8, 12)), aux);
                cv::flip(aux, aux, 0);
                cv::hconcat(DC1.at(0), aux, DC1.at(0));

        }

        DC1.at(2) = DC1.at(2)(cv::Range(0, 4), cv::Range(1, 17));
        DC1.at(1) = DC1.at(1)(cv::Range(0, 4), cv::Range(1, 17));
        DC1.at(0) = DC1.at(0)(cv::Range(0, 4), cv::Range(1, 17));

}
//-----------------------------------------------------------------------------------------------------------------------------------------//

//-----------------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Converte os valores de uma matriz recebida para uma matriz de índices pré-determinados.
/// </summary>
/// <param name="B"> Matriz contendo os dados a serem convertidos </param>
/// <return> Matriz convertida</return>
cv::Mat1d ProcessamentoDeDados::cad2indexFernanda(cv::Mat B){

        cv::Mat1d word;
        cv::Mat1d I = cv::Mat::zeros(3, B.cols, CV_64F);
        cv::Mat1d result = cv::Mat::zeros(9, B.cols, CV_64F);
        cv::Mat1d factor = (cv::Mat_<double>(1, 9) << 256, 128, 64, 32, 16, 8, 4, 2, 1);
        int control;

        for (int k = 0; k < B.cols; k++){

                word = B(cv::Range(0, 3), cv::Range(k, k + 1));
                control = (int)(4 * word.at<double>(0, 0) + 2 * word.at<double>(1, 0) + 1 * word.at<double>(2, 0));
                switch (control)
                {
                case 0:
                        I.at<double>(0, k) = 2;
                        break;
```

```
        case 1:
                I.at<double>(0, k) = 3;
                break;
        case 2:
                I.at<double>(0, k) = 1;
                break;
        case 3:
                I.at<double>(0, k) = 0;
                break;
        case 4:
                I.at<double>(0, k) = 5;
                break;
        case 5:
                I.at<double>(0, k) = 4;
                break;
        case 6:
                I.at<double>(0, k) = 6;
                break;
        case 7:
                I.at<double>(0, k) = 7;
                break;
        default:
                break;
        }

        word = B(cv::Range(3, 5), cv::Range(k, k + 1));
        control = (int)(2 * word.at<double>(0, 0) + 1 * word.at<double>(1, 0));
        switch (control)
        {
        case 0:
                I.at<double>(1, k) = 1;
                break;
        case 1:
                I.at<double>(1, k) = 0;
                break;
        case 2:
                I.at<double>(1, k) = 2;
                break;
        case 3:
                I.at<double>(1, k) = 3;
                break;
        default:
                break;
        }

        word = B(cv::Range(5, 7), cv::Range(k, k + 1));
        control = (int)(2 * word.at<double>(0, 0) + 1 * word.at<double>(1, 0));
        switch (control)
        {
        case 0:
                I.at<double>(2, k) = 1;
                break;
        case 1:
                I.at<double>(2, k) = 0;
                break;
        case 2:
                I.at<double>(2, k) = 2;
                break;
        case 3:
                I.at<double>(2, k) = 3;
                break;
        default:
                break;
        }

}

for (int x = 0; x < B.cols; x++){
        switch ((int)I.at<double>(0, x))
        {
        case 0:
                result.at<double>(0, x) = 0;
                result.at<double>(1, x) = 0;
                result.at<double>(2, x) = 0;
                break;
        case 1:
                result.at<double>(0, x) = 1;
                result.at<double>(1, x) = 0;
                result.at<double>(2, x) = 0;
                break;
        case 2:
                result.at<double>(0, x) = 0;
                result.at<double>(1, x) = 1;
                result.at<double>(2, x) = 0;
                break;
        case 3:
                result.at<double>(0, x) = 1;
                result.at<double>(1, x) = 1;
                result.at<double>(2, x) = 0;
                break;
        case 4:
                result.at<double>(0, x) = 0;
                result.at<double>(1, x) = 0;
                result.at<double>(2, x) = 1;
                break;
```

```
                case 5:
                        result.at<double>(0, x) = 1;
                        result.at<double>(1, x) = 0;
                        result.at<double>(2, x) = 1;
                        break;
                case 6:
                        result.at<double>(0, x) = 0;
                        result.at<double>(1, x) = 1;
                        result.at<double>(2, x) = 1;
                        break;
                case 7:
                        result.at<double>(0, x) = 1;
                        result.at<double>(1, x) = 1;
                        result.at<double>(2, x) = 1;
                        break;
                default:
                        break;
                }

                switch ((int)I.at<double>(1, x))
                {
                case 0:
                        result.at<double>(3, x) = 0;
                        result.at<double>(4, x) = 0;
                        break;
                case 1:
                        result.at<double>(3, x) = 1;
                        result.at<double>(4, x) = 0;
                        break;
                case 2:
                        result.at<double>(3, x) = 0;
                        result.at<double>(4, x) = 1;
                        break;
                case 3:
                        result.at<double>(3, x) = 1;
                        result.at<double>(4, x) = 1;
                        break;
                default:
                        break;
                }

                switch ((int)I.at<double>(2, x))
                {
                case 0:
                        result.at<double>(5, x) = 0;
                        result.at<double>(6, x) = 0;
                        break;
                case 1:
                        result.at<double>(5, x) = 1;
                        result.at<double>(6, x) = 0;
                        break;
                case 2:
                        result.at<double>(5, x) = 0;
                        result.at<double>(6, x) = 1;
                        break;
                case 3:
                        result.at<double>(5, x) = 1;
                        result.at<double>(6, x) = 1;
                        break;
                default:
                        break;
                }

                result.at<double>(7, x) = B.at<double>(7, x);
                result.at<double>(8, x) = B.at<double>(8, x);

        }

        result = factor*result;
        result += 1;

        return result;

}

/// <summary>
/// Obtém a variável Phat a partir dos dados captados pelo imageador e o dicionário VQ.
/// </summary>
/// <param name="Cnovo"> Dicionário VQ </param>
/// <param name="B1"> Bits do módulo de VQ </param>
/// <param name="S1"> Bits do sinal de VQ </param>
/// <return> Variável Phat </return>
cv::Mat1d ProcessamentoDeDados::getPhat(cv::Mat1d Cnovo, cv::Mat B1, cv::Mat S1){

        cv::Mat temp;
        cv::Mat1d Phat = cv::Mat::zeros(5, 256, CV_64F);
        cv::Mat1d ivq = ProcessamentoDeDados::cad2indexFernanda(B1);

        cv::Mat S = cv::Mat::ones(S1.rows, S1.cols, CV_64F) - S1; //VQ Signal

        for (int i = 0; i < Phat.cols; i++){
                Cnovo.col((int)ivq.at<double>(0, i) - 1).copyTo(Phat.col(i));
        }

        for (int i = 0; i < Phat.rows; i++){
                temp = Phat.row(i).mul(((S.row(i) * 2) - 1));
                temp.copyTo(Phat.row(i));
        }
```

```
	temp = Phat(cv::Range(0, 2), cv::Range::all()) * 8;
	temp.copyTo(Phat(cv::Range(0, 2), cv::Range::all()));

	temp = Phat(cv::Range(2, 4), cv::Range::all()) * 2;
	temp.copyTo(Phat(cv::Range(2, 4), cv::Range::all()));

	temp = Phat(cv::Range(4, 5), cv::Range::all()) * 5;
	temp.copyTo(Phat(cv::Range(4, 5), cv::Range::all()));

	return Phat;
}

/// <summary>
/// Obtém a componente VQ a partir de uma variável intermediaria.
/// </summary>
/// <param name="Xhat"> Variável Phat ajustada </param>
/// <return> Componente VQ </return>
cv::Mat1d ProcessamentoDeDados::getY(cv::Mat1d Xhat){

	cv::Mat1d Y = cv::Mat::zeros(64, 64, CV_64F);
	cv::Mat1d bloco = cv::Mat::zeros(4, 4, CV_64F);
	int k, j;
	for (int i = 0; i < N_BLOCOS; i++){
		k = i / N_LINHAS_BLOCOS;
		j = i - k*N_LINHAS_BLOCOS;
		for (int c = 0; c < 4; c++){
			Xhat(cv::Range(4 * c, 4 * c + 4), cv::Range(i, i + 1)).copyTo(bloco.col(c));
		}
		cv::transpose(bloco, bloco);
		bloco.copyTo(Y(cv::Range(4 * k, 4 * k + 4), cv::Range(4 * j, 4 * j + 4)));
	}
	return Y;
}
//-----------------------------------------------------------------------------------------------------------------------------------//

//-----------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Converte uma matriz com valores em código gray para valores dentro de uma escala numérica.
/// </summary>
/// <param name="D"> Matriz com valores em código gray </param>
/// <return> Valor inteiro equivalente </return>
int ProcessamentoDeDados::gray2Term(cv::Mat D){

	int result_index;

	if (D.at<double>(0, 0) == 1){
		if (D.at<double>(0, 1) == 1){
			if (D.at<double>(0, 2) == 1)
				result_index = 8;
			else
				result_index = 7;
		}
		else{
			if (D.at<double>(0, 2) == 1)
				result_index = 5;
			else
				result_index = 6;
		}
	}
	else{
		if (D.at<double>(0, 1) == 1){
			if (D.at<double>(0, 2) == 1)
				result_index = 1;
			else
				result_index = 2;
		}
		else{
			if (D.at<double>(0, 2) == 1)
				result_index = 4;
			else
				result_index = 3;
		}
	}

	return result_index;
}

/// <summary>
/// Processa os valores referentes a componente DPCM em função de certos parâmetros.
/// </summary>
/// <param name="D1"> Bits do DPCM </param>
/// <param name="DC1"> Vetores de correção do DPCM </param>
/// <param name="C_dpcm"> Dicionário DPCM </param>
/// <param name="offset"> Vetor com Offsets utilizados </param>
/// <param name="check"> Controlador da correção do DPCM </param>
/// <param name="limit"> Controlador de limitador </param>
/// <param name="MuHat"> Matriz com DPCM parcial </param>
/// <param name="indexTerm"> Vetor de Índices </param>
void ProcessamentoDeDados::processDPCM(cv::Mat D1, vector<cv::Mat> DC1, cv::Mat C_dpcm, cv::Mat offset, bool check, bool limit,
	cv::Mat1d& MuHat, int(&indexTerm)[N_BLOCOS]){

	cv::Mat temp;
	int nOffset = 0;
	int l = 0;
	double error = 0;
	for (int i = 0; i < N_BLOCOS; i++){
		cv::transpose(D1.col(i), temp);
		temp(cv::Range(0, 1), cv::Range(1, 4)).copyTo(temp);
		indexTerm[i] = ProcessamentoDeDados::gray2Term(temp);
```

```
		if (i % N_LINHAS_BLOCOS == 0){
			MuHat.at<double>(0, i + 1) = offset.at<double>(0, nOffset) +
						C_dpcm.at<double>(indexTerm[i] - 1, 0)*
						(D1.at<double>(0, i) * 2 - 1);
			nOffset++;
		}
		else{
			MuHat.at<double>(0, i + 1) = MuHat.at<double>(0, i) +
						C_dpcm.at<double>(indexTerm[i] - 1, 0)*
						(D1.at<double>(0, i) * 2 - 1);

			if (limit){
				if (MuHat.at<double>(0, i + 1) > 1)
					MuHat.at<double>(0, i + 1) = 1;

				if (MuHat.at<double>(0, i + 1) < 0)
					MuHat.at<double>(0, i + 1) = 0;
			}
		}

		//Correção no DPCM
		if (((i + 1) % N_LINHAS_BLOCOS == 0) && (check)){

			cv::transpose(DC1.at(0)(cv::Range(1, 4), cv::Range(l, l + 1)), temp);
			error = MuHat.at<double>(0, i) +
				C_dpcm.at<double>(ProcessamentoDeDados::gray2Term(temp) - 1, 0)*
				(DC1.at(0).at<double>(0, l) * 2 - 1);
			cv::transpose(DC1.at(1)(cv::Range(1, 4), cv::Range(l, l + 1)), temp);
			error = error + C_dpcm.at<double>(ProcessamentoDeDados::gray2Term(temp) - 1, 0)*
					(DC1.at(1).at<double>(0, l) * 2 - 1);
			cv::transpose(DC1.at(2)(cv::Range(1, 4), cv::Range(l, l + 1)), temp);
			error = error + C_dpcm.at<double>(ProcessamentoDeDados::gray2Term(temp) - 1, 0)*
				    (DC1.at(2).at<double>(0, l) * 2 - 1);

			for (int k = 0; k < N_LINHAS_BLOCOS; k++){
				//MuHat(i-L+k+1)=MuHat(i-L+k+1)-(Error/(L+4))*k;
				MuHat.at<double>(0, i - N_LINHAS_BLOCOS + k + 2) =
					MuHat.at<double>(0, i - N_LINHAS_BLOCOS + k + 2) -
					(error / (N_LINHAS_BLOCOS + 0))*(k + 1);
			}

			l++;
		}

	}
}

/// <summary>
/// Processa os valores referentes a componente PCM em função de certos parâmetros.
/// </summary>
/// <param name="D1"> Bits do DPCM </param>
/// <param name="DC1"> Vetores de correção do DPCM </param>
/// <param name="C_dpcm"> Dicionário DPCM </param>
/// <param name="offset"> Vetor com Offsets utilizados </param>
/// <param name="check"> Controlador da correção do DPCM </param>
/// <param name="limit"> Controlador de limitador </param>
/// <param name="MuHat"> Matriz com DPCM parcial </param>
/// <param name="indexTerm"> Vetor de Índices </param>
void ProcessamentoDeDados::processPCM(cv::Mat D1, vector<cv::Mat> DC1, cv::Mat C_dpcm, cv::Mat offset, bool check,
				cv::Mat1d& MuHat, int(&indexTerm)[N_BLOCOS]){

	cv::Mat temp;
	//ProcessamentoDeDados x;
	int nOffset = 0;
	int l = 0;
	double error = 0;
	for (int i = 0; i < N_BLOCOS; i++){
		cv::transpose(D1.col(i), temp);
		temp(cv::Range(0, 1), cv::Range(1, 4)).copyTo(temp);
		indexTerm[i] = ProcessamentoDeDados::gray2Term(temp);

		if (i % N_LINHAS_BLOCOS == 0){
			MuHat.at<double>(0, i + 1) = offset.at<double>(0, nOffset) +
						C_dpcm.at<double>(indexTerm[i] - 1, 0)*(D1.at<double>(0, i) * 2 - 1);
			nOffset++;
		}
		else{
			MuHat.at<double>(0, i + 1) = C_dpcm.at<double>(indexTerm[i] - 1, 0)*(D1.at<double>(0, i) * 2 - 1);

			if (MuHat.at<double>(0, i + 1) > 1)
				MuHat.at<double>(0, i + 1) = 1;

			if (MuHat.at<double>(0, i + 1) < 0)
				MuHat.at<double>(0, i + 1) = 0;

		}

		//Correção no DPCM
		if (((i + 1) % N_LINHAS_BLOCOS == 0) && (check)){

			cv::transpose(DC1.at(0)(cv::Range(1, 4), cv::Range(l, l + 1)), temp);
			error = MuHat.at<double>(0, i) + C_dpcm.at<double>(ProcessamentoDeDados::gray2Term(temp) - 1, 0)*
					(DC1.at(0).at<double>(0, l) * 2 - 1);
			cv::transpose(DC1.at(1)(cv::Range(1, 4), cv::Range(l, l + 1)), temp);
			error = error + C_dpcm.at<double>(ProcessamentoDeDados::gray2Term(temp) - 1, 0)*
					(DC1.at(1).at<double>(0, l) * 2 - 1);
			cv::transpose(DC1.at(2)(cv::Range(1, 4), cv::Range(l, l + 1)), temp);
```

```
                        error = error + C_dpcm.at<double>(ProcessamentoDeDados::gray2Term(temp) - 1, 0)*
                                                (DC1.at(2).at<double>(0, l) * 2 - 1);

                        for (int k = 0; k < N_LINHAS_BLOCOS; k++){
                                //MuHat(i-L+k+1)=MuHat(i-L+k+1)-(Error/(L+4))*k;
                                MuHat.at<double>(0, i - N_LINHAS_BLOCOS + k + 1) =
                                        MuHat.at<double>(0, i - N_LINHAS_BLOCOS + k + 1) -
                                        (error / (N_LINHAS_BLOCOS + 4))*(k + 1);
                        }

                        l++;
                }

        }
}

/// <summary>
/// Processa a matriz parcial do DPCM e obtém a componente em seu estado final.
/// </summary>
/// <param name="MuHat"> Componente parcial do DPCM </param>
/// <return> Componente DPCM </return>
cv::Mat1d ProcessamentoDeDados::getIm(cv::Mat1d MuHat){

        cv::Mat temp;
        cv::Mat1d Im = cv::Mat::zeros(64, 64, CV_64F);
        int indexMuHat = 1;
        for (int i = 0; i < 64; i += 4){
                for (int j = 0; j < 64; j += 4){
                        temp = cv::Mat::ones(4, 4, CV_64F) * MuHat.at<double>(0, indexMuHat);
                        temp.copyTo(Im(cv::Range(i, i + 4), cv::Range(j, j + 4)));
                        indexMuHat++;
                }
        }

        return Im;
}
//-------------------------------------------------------------------------------------------------------------------------------------//

//-------------------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Intercepta e armazena os bytes lidos do microcontrolador em uma matriz.
/// </summary>
/// <param name="byte"> Byte a ser escrito </param>
/// <param name="mat"> Matriz sendo escrita </param>
/// <param name="position"> Posição aonde escrever o byte </param>
/// <param name="numByte"> Índice do byte </param>
void ProcessamentoDeDados::carregarByte(unsigned char byte, cv::Mat mat,int position, int numByte){

        int mascara = 0x01;
        int max = 8;
        int i = 0;

        if (numByte == 37)
                max = 4;

        while (i < max)
        {

                if ((byte & mascara) == mascara)
                        mat.at<double>(position + i) = 0;
                else
                        mat.at<double>(position + i) = 1;

                mascara *= 2;

                i++;

        }


}

/// <summary>
/// Converte uma array contendo os índices da componente DPCM para uma matriz de mesmo conteúdo.
/// </summary>
/// <param name="indexTerm"> Array contendo os índices </param>
/// <return> Matriz de índices </return>
cv::Mat1i ProcessamentoDeDados::indexTermToMat(int indexTerm[]){

        cv::Mat1i result;
        cv::Mat1i temp = cv::Mat::zeros(1, 16 * 16, CV_64F);

        for (int i = 0; i < temp.cols; i++)
                temp.at<int>(0, i) = indexTerm[i];

        result = matlabReshape(temp, 16, 16);
        cv::transpose(result, result);

        return result;
}
```

```
/// <summary>
/// Converte uma matriz para o formato String.
/// </summary>
/// <param name="mat"> Matriz a ser convertida </param>
/// <param name="precision"> Precisão de casas decimais </param>
/// <return> String equivalente </return>
System::String^ ProcessamentoDeDados::matToString(cv::Mat mat, int precision)
{
	stringstream result;

	result << fixed << setprecision(precision);
	for (int i = 0; i < mat.rows; i++){
		for (int j = 0; j < mat.cols; j++)
			result << right << setw(8) << mat.at<double>(i, j);
		if (i != (mat.rows - 1))
			result << "\r\n";
	}

	return gcnew System::String(result.str().c_str());
}

/// <summary>
/// Reestrutura uma matriz para uma nova composição com o mesmo número de elementos.
/// </summary>
/// <param name="m"> Matriz a ser reformulada </param>
/// <param name="new_row"> Novo número de linhas </param>
/// <param name="new_col"> Novo número de colunas </param>
/// <return> Matriz reformulada </return>
cv::Mat ProcessamentoDeDados::matlabReshape(const cv::Mat m, int new_row, int new_col){

	int new_ch = 1;
	int old_row, old_col, old_ch;
	old_row = m.size().height;
	old_col = m.size().width;
	old_ch = m.channels();

	cv::Mat m1(1, new_row*new_col*new_ch, m.depth());

	vector <cv::Mat> p(old_ch);
	split(m, p);
	for (unsigned i = 0; i < p.size(); ++i){
		cv::Mat t(p[i].size().height, p[i].size().width, m1.type());
		t = p[i].t();
		cv::Mat aux = m1.colRange(i*old_row*old_col, (i + 1)*old_row*old_col).rowRange(0, 1);
		t.reshape(0, 1).copyTo(aux);
	}

	vector <cv::Mat> r(new_ch);
	for (unsigned i = 0; i < r.size(); ++i){
		cv::Mat aux = m1.colRange(i*new_row*new_col, (i + 1)*new_row*new_col).rowRange(0, 1);
		r[i] = aux.reshape(0, new_col);
		r[i] = r[i].t();
	}

	cv::Mat result;
	merge(r, result);
	return result;

}
//-----------------------------------------------------------------------------------------------------------------------------//

//-----------------------------------------------------------------------------------------------------------------------------//
/// <summary>
/// Processa os valores referentes a componente DPCM em função de certos parâmetros.
/// </summary>
/// <param name="D1"> Bits do DPCM </param>
/// <param name="DC1"> Vetores de correção do DPCM </param>
/// <param name="C_dpcm"> Dicionário DPCM </param>
/// <param name="offset"> Vetor com Offsets utilizados </param>
/// <return> Matriz à ser exibida</return>
cv::Mat1d ProcessamentoDeDados::prepareFinalImage(cv::Mat1d Im, cv::Mat1d Y, cv::Mat1d imagemFinal, cv::Mat1d imagemMedia){

	cv::Mat1d imageTotal;
	cv::Mat space = cv::Mat::ones(64, 3, CV_64F);
	cv::hconcat(Im, space, imageTotal);
	cv::hconcat(imageTotal, Y, imageTotal);
	cv::hconcat(imageTotal, space, imageTotal);
	cv::hconcat(imageTotal, imagemFinal, imageTotal);
	cv::hconcat(imageTotal, space, imageTotal);
	cv::hconcat(imageTotal, imagemMedia, imageTotal);

	cv::Size size(0, 0);
	cv::resize(imageTotal, imageTotal, size, 3, 3, 1);

	return imageTotal;
}
//-----------------------------------------------------------------------------------------------------------------------------//
```

MyForm.h

```
#pragma once

#include <windows.h>
#include <Dbt.h>
#include <string>
#include <math.h>
#include <chrono>

#include "Define.h"
#include "mpusbapi.h"
#include "processamentoDeDados.h"
#include "processamentoDeArquivos.h"

#include <opencv2/imgproc.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>


//LEMBRETE: ao alterar firmware, atualizar o valor abaixo!
// "Vid_xxxx&Pid_xxxx" aonde xxxx é um número hexadecimal de 16-bit.
#define DeviceVID_PID "vid_04d8&pid_000c"

using namespace std;

namespace ProjetoFinal {

	using namespace System;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::Collections;
	using namespace System::ComponentModel;
	using namespace System::Windows::Forms;
	//-----------------------------------------------//
	using namespace System::Threading;
	using namespace System::Runtime::InteropServices;

	#pragma region Declaração de Fábrica - Comunicação USB

	#ifdef UNICODE
	#define Seeifdef	Unicode
	#else
	#define Seeifdef	Ansi
	#endif

	//See the mpusbapi.dll source code (_mpusbapi.cpp) for API related documentation for these functions.
	//The source code is in the MCHPFSUSB vX.X distributions.
	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBGetDLLVersion", CallingConvention=CallingConvention::Cdecl)]
	extern "C" DWORD MPUSBGetDLLVersion(void);
	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBGetDeviceCount", CallingConvention=CallingConvention::Cdecl)]
	extern "C" DWORD MPUSBGetDeviceCount(PCHAR pVID_PID);
	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBOpen", CallingConvention=CallingConvention::Cdecl)]
	extern "C" HANDLE MPUSBOpen(DWORD instance,	//  Input
				PCHAR pVID_PID,		// Input
				PCHAR pEP,		// Input
				DWORD dwDir,		// Input
				DWORD dwReserved);	// Input

	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBClose", CallingConvention=CallingConvention::Cdecl)]
	extern "C" BOOL MPUSBClose(HANDLE handle);	//Input
	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBRead", CallingConvention=CallingConvention::Cdecl)]
	extern "C" DWORD MPUSBRead(HANDLE handle,	// Input
				PVOID pData,		// Output
				DWORD dwLen,		// Input
				PDWORD pLength,		// Output
				DWORD dwMilliseconds); // Input

	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBWrite", CallingConvention=CallingConvention::Cdecl)]
	extern "C" DWORD MPUSBWrite(HANDLE handle,	// Input
				PVOID pData,		// Output
				DWORD dwLen,		// Input
				PDWORD pLength,		// Output
				DWORD dwMilliseconds); // Input
	[DllImport("MPUSBAPI.dll" , EntryPoint="_MPUSBReadInt", CallingConvention=CallingConvention::Cdecl)]
	extern "C" DWORD MPUSBReadInt(HANDLE handle,	// Input
				PVOID pData,		// Output
				DWORD dwLen,		// Input
				PDWORD pLength,		// Output
				DWORD dwMilliseconds);	// Input

	//Need this function for receiving all of the WM_DEVICECHANGE messages.  See MSDN documentation for
	//description of what this function does/how to use it. Note: name is remapped "RegisterDeviceNotificationUM" to
	//avoid possible build error conflicts.
	[DllImport("user32.dll" , CharSet = CharSet::Seeifdef, EntryPoint="RegisterDeviceNotification",
	CallingConvention=CallingConvention::Winapi)]
	extern "C" HDEVNOTIFY WINAPI RegisterDeviceNotificationUM(	HANDLE hRecipient,
									LPVOID NotificationFilter,
									DWORD Flags);
	#pragma endregion
```

```
//-------------Variáveis Globais----------------//
HANDLE  EP1INHandle = INVALID_HANDLE_VALUE;
HANDLE  EP1OUTHandle = INVALID_HANDLE_VALUE;

boolean AttachedState = false;                //Variável de controle para encaixe do cabo USB.

boolean sairAnaliseLimiar = false;            //Variável de controle para cancelar função: ANALISE LINEAR.

boolean PCMControl = false;                   //Variável de controle para acesso a função: PCM.
boolean debugControl = false;                 //Variável de controle para acesso a função: DEBUG.
boolean limiarControl = false;                //Variável de controle para acesso a função: ANALISE LINEAR.
boolean pxlTesteControl = false;              //Variável de controle para acesso a função: PIXEL TESTE.
boolean blcPrincipalControl = false;          //Variável de controle para acesso a função: BLOCO PRINCIPAL.
//----------------------------------------------//

/// <summary>
/// Janela principal do programa.
/// </summary>
public ref class MyForm : public System::Windows::Forms::Form {

public:
        MyForm(void)
        {
                InitializeComponent();

                //Globally Unique Identifier (GUID).
                GUID InterfaceClassGuid = {0xa5dcbf10, 0x6530, 0x11d2, 0x90, 0x1F, 0x00, 0xC0, 0x4F, 0xB9,
                                0x51, 0xED}; //Valores de GUID para dispositivos periféricos USB.

                //Configurando para notificações do tipo WM_DEVICECHANGE:
                DEV_BROADCAST_DEVICEINTERFACE MyDeviceBroadcastHeader;
                MyDeviceBroadcastHeader.dbcc_devicetype = DBT_DEVTYP_DEVICEINTERFACE;
                MyDeviceBroadcastHeader.dbcc_size = sizeof(DEV_BROADCAST_DEVICEINTERFACE);
                MyDeviceBroadcastHeader.dbcc_reserved = 0;
                MyDeviceBroadcastHeader.dbcc_classguid = InterfaceClassGuid;
                RegisterDeviceNotificationUM((HANDLE)this->Handle, &MyDeviceBroadcastHeader,
                DEVICE_NOTIFY_WINDOW_HANDLE);

                //Checando se o dispositivo já se encontra conectado antes mesmo do programa ser ativado:
                if(MPUSBGetDeviceCount(DeviceVID_PID)) // Conectado.
                {
                        EP1OUTHandle = MPUSBOpen(0, DeviceVID_PID, "\\MCHP_EP1", MP_WRITE, 0);
                        EP1INHandle = MPUSBOpen(0, DeviceVID_PID, "\\MCHP_EP1", MP_READ, 0);

                        StatusBox_txtbx->Text = "Dispositivo Encontrado!";
                        AttachedState = TRUE;

                        checkREC->Enabled = true;
                        checkLimit->Enabled = true;
                        checkFlipLR->Enabled = true;
                        checkFlipUD->Enabled = true;
                        checkOffset->Enabled = true;
                        checkCorrecao->Enabled = true;

                        radioTempoBox->Enabled = true;
                        radioTempoLoad->Enabled = true;

                        buttonPCM->Enabled = true;
                        buttonDebug->Enabled = true;
                        buttonConfig->Enabled = true;
                        buttonLimiar->Enabled = true;
                        buttonPxlTeste->Enabled = true;
                        buttonBlcPrincipal->Enabled = true;

                }
                else    //Desconetado ou mal configurado.
                {
                        StatusBox_txtbx->Text = "Dispositivo Não Encontrado: Cheque a Comunicação";
                        AttachedState = FALSE;

                        checkREC->Enabled = false;
                        checkFlipLR->Enabled = false;
                        checkFlipUD->Enabled = false;
                        checkLimit->Enabled = false;
                        checkOffset->Enabled = false;
                        checkCorrecao->Enabled = false;
                        checkLimitDPCM->Enabled = false;

                        radioTempoBox->Enabled = false;
                        radioTempoLoad->Enabled = false;

                        buttonPCM->Enabled = false;
                        buttonDebug->Enabled = false;
                        buttonConfig->Enabled = false;
                        buttonLimiar->Enabled = false;
                        buttonPararPCM->Enabled = false;
                        buttonPxlTeste->Enabled = false;
                        buttonPararDebug->Enabled = false;
                        buttonPararConfig->Enabled = false;
                        buttonBlcPrincipal->Enabled = false;
                        buttonPararPxlTeste->Enabled = false;
                        buttonPararBlcPrincipal->Enabled = false;

                }

                ReadWriteThread->RunWorkerAsync();     //Acionando a thread responsável pelo processo
                                                       de executar os comandos.

        }
```

```
protected:
        /// <summary>
        /// Libera recursos sendo utilizados ao encerrar o programa.
        /// </summary>
        ~MyForm()
        {
                //Garantindo que toda HANDLE seja finalizada.
                if (EP1OUTHandle != INVALID_HANDLE_VALUE)
                        MPUSBClose (EP1OUTHandle);
                if (EP1INHandle != INVALID_HANDLE_VALUE)
                        MPUSBClose (EP1INHandle);

                if (components)
                {
                        delete components;
                }
        }

public:
        delegate void parleyDelegate(String^ text, int function);

        #pragma region Iniciando Componentes
private:
        System::Windows::Forms::Label^  label1;
        System::Windows::Forms::Label^  label3;
        System::Windows::Forms::Label^  label2;
        System::Windows::Forms::Label^  label4;
        System::Windows::Forms::Label^  label5;

        System::Windows::Forms::GroupBox^  groupBox1;
        System::Windows::Forms::GroupBox^  groupBox2;
        System::Windows::Forms::GroupBox^  groupBox3;
        System::Windows::Forms::GroupBox^  groupBox4;
        System::Windows::Forms::GroupBox^  groupBox5;

        System::Windows::Forms::CheckBox^  checkREC;
        System::Windows::Forms::CheckBox^  checkTanh;
        System::Windows::Forms::CheckBox^  checkLimit;
        System::Windows::Forms::CheckBox^  checkFlipLR;
        System::Windows::Forms::CheckBox^  checkFlipUD;
        System::Windows::Forms::CheckBox^  checkOffset;
        System::Windows::Forms::CheckBox^  checkDilate;
        System::Windows::Forms::CheckBox^  checkCorrecao;
        System::Windows::Forms::CheckBox^  checkBrilhoVQ;
        System::Windows::Forms::CheckBox^  checkFiltro2D;
        System::Windows::Forms::CheckBox^  checkBrilhoIm;
        System::Windows::Forms::CheckBox^  checkGaussiana;
        System::Windows::Forms::CheckBox^  checkLimitDPCM;

        System::Windows::Forms::Button^  buttonPxlTeste;

        System::ComponentModel::IContainer^  components;

        System::Windows::Forms::TextBox^  textBoxDebug;
        System::Windows::Forms::TextBox^  StatusBox_txtbx;

        System::Windows::Forms::RadioButton^  radioDebugIm;
        System::Windows::Forms::RadioButton^  radioDebugVq;
        System::Windows::Forms::RadioButton^  radioTempoBox;
        System::Windows::Forms::RadioButton^  radioTempoLoad;
        System::Windows::Forms::RadioButton^  radioDebugMuHat;
        System::Windows::Forms::RadioButton^  radioDebugIndex;

        System::ComponentModel::BackgroundWorker^  ReadWriteThread;

        System::Windows::Forms::Button^  buttonPCM;
        System::Windows::Forms::Button^  buttonDebug;
        System::Windows::Forms::Button^  buttonConfig;
        System::Windows::Forms::Button^  buttonLimiar;
        System::Windows::Forms::Button^  buttonPararPCM;
        System::Windows::Forms::Button^  buttonPararDebug;
        System::Windows::Forms::Button^  buttonPararConfig;
        System::Windows::Forms::Button^  buttonBlcPrincipal;
        System::Windows::Forms::Button^  buttonPararAnalise;
        System::Windows::Forms::Button^  buttonPararPxlTeste;
        System::Windows::Forms::Button^  buttonPararBlcPrincipal;

        System::Windows::Forms::NumericUpDown^  numericY;
        System::Windows::Forms::NumericUpDown^  numericX;
        System::Windows::Forms::NumericUpDown^  timeInput;
        System::Windows::Forms::NumericUpDown^  numericAlfa;
        System::Windows::Forms::NumericUpDown^  numericKernel;
        System::Windows::Forms::NumericUpDown^  numericDilate;
        System::Windows::Forms::NumericUpDown^  numericVQBright;
        System::Windows::Forms::NumericUpDown^  numericImBright;
        System::Windows::Forms::NumericUpDown^  numericBlurGauss;
        System::Windows::Forms::NumericUpDown^  numericCoeficienteB;
        System::Windows::Forms::NumericUpDown^  numericCoeficienteA;
        #pragma endregion
```

```
#pragma region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
void InitializeComponent(void)
{
        this->StatusBox_txtbx = (gcnew System::Windows::Forms::TextBox());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->ReadWriteThread = (gcnew System::ComponentModel::BackgroundWorker());
        this->buttonBlcPrincipal = (gcnew System::Windows::Forms::Button());
        this->buttonPararBlcPrincipal = (gcnew System::Windows::Forms::Button());
        this->timeInput = (gcnew System::Windows::Forms::NumericUpDown());
        this->checkLimit = (gcnew System::Windows::Forms::CheckBox());
        this->checkFlipLR = (gcnew System::Windows::Forms::CheckBox());
        this->buttonPxlTeste = (gcnew System::Windows::Forms::Button());
        this->buttonPararPxlTeste = (gcnew System::Windows::Forms::Button());
        this->checkFlipUD = (gcnew System::Windows::Forms::CheckBox());
        this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
        this->radioTempoLoad = (gcnew System::Windows::Forms::RadioButton());
        this->radioTempoBox = (gcnew System::Windows::Forms::RadioButton());
        this->checkOffset = (gcnew System::Windows::Forms::CheckBox());
        this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
        this->checkREC = (gcnew System::Windows::Forms::CheckBox());
        this->checkLimitDPCM = (gcnew System::Windows::Forms::CheckBox());
        this->checkCorrecao = (gcnew System::Windows::Forms::CheckBox());
        this->buttonPCM = (gcnew System::Windows::Forms::Button());
        this->buttonDebug = (gcnew System::Windows::Forms::Button());
        this->buttonPararPCM = (gcnew System::Windows::Forms::Button());
        this->buttonPararDebug = (gcnew System::Windows::Forms::Button());
        this->textBoxDebug = (gcnew System::Windows::Forms::TextBox());
        this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
        this->label3 = (gcnew System::Windows::Forms::Label());
        this->label2 = (gcnew System::Windows::Forms::Label());
        this->numericY = (gcnew System::Windows::Forms::NumericUpDown());
        this->numericX = (gcnew System::Windows::Forms::NumericUpDown());
        this->radioDebugVq = (gcnew System::Windows::Forms::RadioButton());
        this->radioDebugIm = (gcnew System::Windows::Forms::RadioButton());
        this->radioDebugIndex = (gcnew System::Windows::Forms::RadioButton());
        this->radioDebugMuHat = (gcnew System::Windows::Forms::RadioButton());
        this->buttonConfig = (gcnew System::Windows::Forms::Button());
        this->buttonPararConfig = (gcnew System::Windows::Forms::Button());
        this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());
        this->buttonPararAnalise = (gcnew System::Windows::Forms::Button());
        this->numericAlfa = (gcnew System::Windows::Forms::NumericUpDown());
        this->buttonLimiar = (gcnew System::Windows::Forms::Button());
        this->groupBox5 = (gcnew System::Windows::Forms::GroupBox());
        this->label5 = (gcnew System::Windows::Forms::Label());
        this->numericCoeficienteB = (gcnew System::Windows::Forms::NumericUpDown());
        this->label4 = (gcnew System::Windows::Forms::Label());
        this->checkTanh = (gcnew System::Windows::Forms::CheckBox());
        this->numericCoeficienteA = (gcnew System::Windows::Forms::NumericUpDown());
        this->checkBrilhoIm = (gcnew System::Windows::Forms::CheckBox());
        this->numericImBright = (gcnew System::Windows::Forms::NumericUpDown());
        this->checkDilate = (gcnew System::Windows::Forms::CheckBox());
        this->checkGaussiana = (gcnew System::Windows::Forms::CheckBox());
        this->checkBrilhoVQ = (gcnew System::Windows::Forms::CheckBox());
        this->numericDilate = (gcnew System::Windows::Forms::NumericUpDown());
        this->checkFiltro2D = (gcnew System::Windows::Forms::CheckBox());
        this->numericBlurGauss = (gcnew System::Windows::Forms::NumericUpDown());
        this->numericVQBright = (gcnew System::Windows::Forms::NumericUpDown());
        this->numericKernel = (gcnew System::Windows::Forms::NumericUpDown());
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->timeInput))->BeginInit();
        this->groupBox1->SuspendLayout();
        this->groupBox2->SuspendLayout();
        this->groupBox3->SuspendLayout();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericY))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericX))->BeginInit();
        this->groupBox4->SuspendLayout();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericAlfa))->BeginInit();
        this->groupBox5->SuspendLayout();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>
                                        (this->numericCoeficienteB))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>
                                        (this->numericCoeficienteA))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>
                                        (this->numericImBright))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericDilate))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>
                                        (this->numericBlurGauss))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>
                                        (this->numericVQBright))->BeginInit();
        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericKernel))->BeginInit();
        this->SuspendLayout();
        //
        // StatusBox_txtbx
        //
        this->StatusBox_txtbx->Enabled = false;
        this->StatusBox_txtbx->Location = System::Drawing::Point(9, 10);
        this->StatusBox_txtbx->Name = L"StatusBox_txtbx";
        this->StatusBox_txtbx->Size = System::Drawing::Size(278, 20);
        this->StatusBox_txtbx->TabIndex = 0;
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Location = System::Drawing::Point(290, 13);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(37, 13);
```

```
this->label1->TabIndex = 1;
this->label1->Text = L"Status";
//
// ReadWriteThread
//
this->ReadWriteThread->DoWork += gcnew System::ComponentModel::DoWorkEventHandler(this,
                      &MyForm::ReadWriteThread_DoWork);
//
// buttonBlcPrincipal
//
this->buttonBlcPrincipal->Location = System::Drawing::Point(6, 19);
this->buttonBlcPrincipal->Name = L"buttonBlcPrincipal";
this->buttonBlcPrincipal->Size = System::Drawing::Size(116, 24);
this->buttonBlcPrincipal->TabIndex = 4;
this->buttonBlcPrincipal->Text = L"BLOCO PRINCIPAL";
this->buttonBlcPrincipal->UseVisualStyleBackColor = true;
this->buttonBlcPrincipal->Click += gcnew System::EventHandler(this, &MyForm::buttonTest_Click);
//
// buttonPararBlcPrincipal
//
this->buttonPararBlcPrincipal->Enabled = false;
this->buttonPararBlcPrincipal->Location = System::Drawing::Point(131, 19);
this->buttonPararBlcPrincipal->Name = L"buttonPararBlcPrincipal";
this->buttonPararBlcPrincipal->Size = System::Drawing::Size(82, 24);
this->buttonPararBlcPrincipal->TabIndex = 5;
this->buttonPararBlcPrincipal->Text = L"<- Parar";
this->buttonPararBlcPrincipal->UseVisualStyleBackColor = true;
this->buttonPararBlcPrincipal->Click += gcnew System::EventHandler(this,MyForm::buttonPararTest_Click);
//
// timeInput
//
this->timeInput->Increment = System::Decimal(gcnew cli::array< System::Int32 >(4) { 10, 0, 0, 0 });
this->timeInput->Location = System::Drawing::Point(31, 22);
this->timeInput->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 10000, 0, 0, 0 });
this->timeInput->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 10, 0, 0, 0 });
this->timeInput->Name = L"timeInput";
this->timeInput->Size = System::Drawing::Size(77, 20);
this->timeInput->TabIndex = 6;
this->timeInput->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 400, 0, 0, 0 });
//
// checkLimit
//
this->checkLimit->AutoSize = true;
this->checkLimit->Location = System::Drawing::Point(9, 42);
this->checkLimit->Name = L"checkLimit";
this->checkLimit->Size = System::Drawing::Size(99, 17);
this->checkLimit->TabIndex = 9;
this->checkLimit->Text = L"Limitar Corrente";
this->checkLimit->UseVisualStyleBackColor = true;
//
// checkFlipLR
//
this->checkFlipLR->AutoSize = true;
this->checkFlipLR->Location = System::Drawing::Point(9, 65);
this->checkFlipLR->Name = L"checkFlipLR";
this->checkFlipLR->Size = System::Drawing::Size(92, 17);
this->checkFlipLR->TabIndex = 10;
this->checkFlipLR->Text = L"Flip Horizontal";
this->checkFlipLR->UseVisualStyleBackColor = true;
//
// buttonPxlTeste
//
this->buttonPxlTeste->Location = System::Drawing::Point(6, 49);
this->buttonPxlTeste->Name = L"buttonPxlTeste";
this->buttonPxlTeste->Size = System::Drawing::Size(116, 24);
this->buttonPxlTeste->TabIndex = 11;
this->buttonPxlTeste->Text = L"PIXEL TESTE";
this->buttonPxlTeste->UseVisualStyleBackColor = true;
this->buttonPxlTeste->Click += gcnew System::EventHandler(this, &MyForm::buttonPxlTeste_Click);
//
// buttonPararPxlTeste
//
this->buttonPararPxlTeste->Enabled = false;
this->buttonPararPxlTeste->Location = System::Drawing::Point(131, 49);
this->buttonPararPxlTeste->Name = L"buttonPararPxlTeste";
this->buttonPararPxlTeste->Size = System::Drawing::Size(82, 24);
this->buttonPararPxlTeste->TabIndex = 12;
this->buttonPararPxlTeste->Text = L"<- Parar";
this->buttonPararPxlTeste->UseVisualStyleBackColor = true;
this->buttonPararPxlTeste->Click += gcnew System::EventHandler(this,&MyForm::buttonPararPxlTeste_Click);
//
// checkFlipUD
//
this->checkFlipUD->AutoSize = true;
this->checkFlipUD->Location = System::Drawing::Point(9, 88);
this->checkFlipUD->Name = L"checkFlipUD";
this->checkFlipUD->Size = System::Drawing::Size(80, 17);
this->checkFlipUD->TabIndex = 13;
this->checkFlipUD->Text = L"Flip Vertical";
this->checkFlipUD->UseVisualStyleBackColor = true;
//
// groupBox1
//
this->groupBox1->Controls->Add(this->radioTempoLoad);
this->groupBox1->Controls->Add(this->radioTempoBox);
this->groupBox1->Controls->Add(this->timeInput);
this->groupBox1->Location = System::Drawing::Point(9, 248);
this->groupBox1->Name = L"groupBox1";
this->groupBox1->Size = System::Drawing::Size(219, 50);
```

```
this->groupBox1->TabIndex = 14;
this->groupBox1->TabStop = false;
this->groupBox1->Text = L"Tempo de Integração";
//
// radioTempoLoad
//
this->radioTempoLoad->AutoSize = true;
this->radioTempoLoad->Location = System::Drawing::Point(129, 22);
this->radioTempoLoad->Name = L"radioTempoLoad";
this->radioTempoLoad->Size = System::Drawing::Size(86, 17);
this->radioTempoLoad->TabIndex = 18;
this->radioTempoLoad->Text = L"Usar Arquivo";
this->radioTempoLoad->UseVisualStyleBackColor = true;
//
// radioTempoBox
//
this->radioTempoBox->AutoSize = true;
this->radioTempoBox->Checked = true;
this->radioTempoBox->Location = System::Drawing::Point(11, 24);
this->radioTempoBox->Name = L"radioTempoBox";
this->radioTempoBox->Size = System::Drawing::Size(14, 13);
this->radioTempoBox->TabIndex = 17;
this->radioTempoBox->TabStop = true;
this->radioTempoBox->UseVisualStyleBackColor = true;
//
// checkOffset
//
this->checkOffset->AutoSize = true;
this->checkOffset->Location = System::Drawing::Point(9, 19);
this->checkOffset->Name = L"checkOffset";
this->checkOffset->Size = System::Drawing::Size(81, 17);
this->checkOffset->TabIndex = 15;
this->checkOffset->Text = L"Load Offset";
this->checkOffset->UseVisualStyleBackColor = true;
//
// groupBox2
//
this->groupBox2->Controls->Add(this->checkREC);
this->groupBox2->Controls->Add(this->checkLimitDPCM);
this->groupBox2->Controls->Add(this->checkOffset);
this->groupBox2->Controls->Add(this->checkCorrecao);
this->groupBox2->Controls->Add(this->checkFlipUD);
this->groupBox2->Controls->Add(this->checkFlipLR);
this->groupBox2->Controls->Add(this->checkLimit);
this->groupBox2->Location = System::Drawing::Point(232, 43);
this->groupBox2->Name = L"groupBox2";
this->groupBox2->Size = System::Drawing::Size(120, 255);
this->groupBox2->TabIndex = 16;
this->groupBox2->TabStop = false;
this->groupBox2->Text = L"Opções";
//
// checkREC
//
this->checkREC->AutoSize = true;
this->checkREC->Location = System::Drawing::Point(9, 155);
this->checkREC->Name = L"checkREC";
this->checkREC->Size = System::Drawing::Size(48, 17);
this->checkREC->TabIndex = 16;
this->checkREC->Text = L"REC";
this->checkREC->UseVisualStyleBackColor = true;
//
// checkLimitDPCM
//
this->checkLimitDPCM->AutoSize = true;
this->checkLimitDPCM->Location = System::Drawing::Point(9, 132);
this->checkLimitDPCM->Name = L"checkLimitDPCM";
this->checkLimitDPCM->Size = System::Drawing::Size(87, 17);
this->checkLimitDPCM->TabIndex = 16;
this->checkLimitDPCM->Text = L"Impor Limites";
this->checkLimitDPCM->UseVisualStyleBackColor = true;
//
// checkCorrecao
//
this->checkCorrecao->AutoSize = true;
this->checkCorrecao->Location = System::Drawing::Point(9, 109);
this->checkCorrecao->Name = L"checkCorrecao";
this->checkCorrecao->Size = System::Drawing::Size(103, 17);
this->checkCorrecao->TabIndex = 13;
this->checkCorrecao->Text = L"Correção DPCM";
this->checkCorrecao->UseVisualStyleBackColor = true;
//
// buttonPCM
//
this->buttonPCM->Location = System::Drawing::Point(6, 79);
this->buttonPCM->Name = L"buttonPCM";
this->buttonPCM->Size = System::Drawing::Size(116, 24);
this->buttonPCM->TabIndex = 17;
this->buttonPCM->Text = L"PCM";
this->buttonPCM->UseVisualStyleBackColor = true;
this->buttonPCM->Click += gcnew System::EventHandler(this, &MyForm::buttonPCM_Click);
//
// buttonDebug
//
this->buttonDebug->Location = System::Drawing::Point(6, 109);
this->buttonDebug->Name = L"buttonDebug";
this->buttonDebug->Size = System::Drawing::Size(116, 24);
this->buttonDebug->TabIndex = 18;
this->buttonDebug->Text = L"DEBUG";
this->buttonDebug->UseVisualStyleBackColor = true;
```

```
this->buttonDebug->Click += gcnew System::EventHandler(this, &MyForm::buttonDebug_Click);
//
// buttonPararPCM
//
this->buttonPararPCM->Enabled = false;
this->buttonPararPCM->Location = System::Drawing::Point(131, 79);
this->buttonPararPCM->Name = L"buttonPararPCM";
this->buttonPararPCM->Size = System::Drawing::Size(82, 24);
this->buttonPararPCM->TabIndex = 19;
this->buttonPararPCM->Text = L"<- Parar";
this->buttonPararPCM->UseVisualStyleBackColor = true;
this->buttonPararPCM->Click += gcnew System::EventHandler(this, &MyForm::buttonPararPCM_Click);
//
// buttonPararDebug
//
this->buttonPararDebug->Enabled = false;
this->buttonPararDebug->Location = System::Drawing::Point(131, 109);
this->buttonPararDebug->Name = L"buttonPararDebug";
this->buttonPararDebug->Size = System::Drawing::Size(82, 24);
this->buttonPararDebug->TabIndex = 20;
this->buttonPararDebug->Text = L"<- Parar";
this->buttonPararDebug->UseVisualStyleBackColor = true;
this->buttonPararDebug->Click += gcnew System::EventHandler(this, &MyForm::buttonPararDebug_Click);
//
// textBoxDebug
//
this->textBoxDebug->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 8.25F,
                                System::Drawing::FontStyle::Regular,
                                System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
this->textBoxDebug->Location = System::Drawing::Point(6, 42);
this->textBoxDebug->Multiline = true;
this->textBoxDebug->Name = L"textBoxDebug";
this->textBoxDebug->ReadOnly = true;
this->textBoxDebug->Size = System::Drawing::Size(550, 215);
this->textBoxDebug->TabIndex = 21;
this->textBoxDebug->WordWrap = false;
//
// groupBox3
//
this->groupBox3->Controls->Add(this->label3);
this->groupBox3->Controls->Add(this->label2);
this->groupBox3->Controls->Add(this->numericY);
this->groupBox3->Controls->Add(this->numericX);
this->groupBox3->Controls->Add(this->radioDebugVq);
this->groupBox3->Controls->Add(this->radioDebugIm);
this->groupBox3->Controls->Add(this->radioDebugIndex);
this->groupBox3->Controls->Add(this->radioDebugMuHat);
this->groupBox3->Controls->Add(this->textBoxDebug);
this->groupBox3->Location = System::Drawing::Point(358, 10);
this->groupBox3->Name = L"groupBox3";
this->groupBox3->Size = System::Drawing::Size(562, 288);
this->groupBox3->TabIndex = 22;
this->groupBox3->TabStop = false;
this->groupBox3->Text = L"Debug";
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(289, 264);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(20, 13);
this->label3->TabIndex = 28;
this->label3->Text = L"Y :";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(185, 264);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(20, 13);
this->label2->TabIndex = 28;
this->label2->Text = L"X :";
//
// numericY
//
this->numericY->Location = System::Drawing::Point(315, 262);
this->numericY->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 4, 0, 0, 0 });
this->numericY->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericY->Name = L"numericY";
this->numericY->ReadOnly = true;
this->numericY->Size = System::Drawing::Size(35, 20);
this->numericY->TabIndex = 27;
this->numericY->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
//
// numericX
//
this->numericX->Location = System::Drawing::Point(211, 262);
this->numericX->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 4, 0, 0, 0 });
this->numericX->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericX->Name = L"numericX";
this->numericX->ReadOnly = true;
this->numericX->Size = System::Drawing::Size(35, 20);
this->numericX->TabIndex = 27;
this->numericX->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
//
// radioDebugVq
//
this->radioDebugVq->AutoSize = true;
```

```
this->radioDebugVq->Location = System::Drawing::Point(194, 19);
this->radioDebugVq->Name = L"radioDebugVq";
this->radioDebugVq->Size = System::Drawing::Size(40, 17);
this->radioDebugVq->TabIndex = 24;
this->radioDebugVq->Text = L"VQ";
this->radioDebugVq->UseVisualStyleBackColor = true;
//
// radioDebugIm
//
this->radioDebugIm->AutoSize = true;
this->radioDebugIm->Location = System::Drawing::Point(152, 19);
this->radioDebugIm->Name = L"radioDebugIm";
this->radioDebugIm->Size = System::Drawing::Size(36, 17);
this->radioDebugIm->TabIndex = 24;
this->radioDebugIm->Text = L"Im";
this->radioDebugIm->UseVisualStyleBackColor = true;
//
// radioDebugIndex
//
this->radioDebugIndex->AutoSize = true;
this->radioDebugIndex->Location = System::Drawing::Point(71, 19);
this->radioDebugIndex->Name = L"radioDebugIndex";
this->radioDebugIndex->Size = System::Drawing::Size(75, 17);
this->radioDebugIndex->TabIndex = 23;
this->radioDebugIndex->Text = L"IndexTerm";
this->radioDebugIndex->UseVisualStyleBackColor = true;
//
// radioDebugMuHat
//
this->radioDebugMuHat->AutoSize = true;
this->radioDebugMuHat->Checked = true;
this->radioDebugMuHat->Location = System::Drawing::Point(8, 19);
this->radioDebugMuHat->Name = L"radioDebugMuHat";
this->radioDebugMuHat->Size = System::Drawing::Size(57, 17);
this->radioDebugMuHat->TabIndex = 22;
this->radioDebugMuHat->TabStop = true;
this->radioDebugMuHat->Text = L"MuHat";
this->radioDebugMuHat->UseVisualStyleBackColor = true;
//
// buttonConfig
//
this->buttonConfig->Location = System::Drawing::Point(6, 139);
this->buttonConfig->Name = L"buttonConfig";
this->buttonConfig->Size = System::Drawing::Size(116, 23);
this->buttonConfig->TabIndex = 16;
this->buttonConfig->Text = L"CONFIG";
this->buttonConfig->UseVisualStyleBackColor = true;
this->buttonConfig->Click += gcnew System::EventHandler(this, &MyForm::buttonConfig_Click);
//
// buttonPararConfig
//
this->buttonPararConfig->Enabled = false;
this->buttonPararConfig->Location = System::Drawing::Point(131, 139);
this->buttonPararConfig->Name = L"buttonPararConfig";
this->buttonPararConfig->Size = System::Drawing::Size(82, 23);
this->buttonPararConfig->TabIndex = 16;
this->buttonPararConfig->Text = L"<- Parar";
this->buttonPararConfig->UseVisualStyleBackColor = true;
this->buttonPararConfig->Click += gcnew System::EventHandler(this, &MyForm::buttonPararConfig_Click);
//
// groupBox4
//
this->groupBox4->Controls->Add(this->buttonPararAnalise);
this->groupBox4->Controls->Add(this->numericAlfa);
this->groupBox4->Controls->Add(this->buttonLimiar);
this->groupBox4->Controls->Add(this->buttonBlcPrincipal);
this->groupBox4->Controls->Add(this->buttonPararConfig);
this->groupBox4->Controls->Add(this->buttonPararBlcPrincipal);
this->groupBox4->Controls->Add(this->buttonConfig);
this->groupBox4->Controls->Add(this->buttonPararDebug);
this->groupBox4->Controls->Add(this->buttonPxlTeste);
this->groupBox4->Controls->Add(this->buttonPararPCM);
this->groupBox4->Controls->Add(this->buttonPCM);
this->groupBox4->Controls->Add(this->buttonDebug);
this->groupBox4->Controls->Add(this->buttonPararPxlTeste);
this->groupBox4->Location = System::Drawing::Point(9, 43);
this->groupBox4->Name = L"groupBox4";
this->groupBox4->Size = System::Drawing::Size(219, 199);
this->groupBox4->TabIndex = 23;
this->groupBox4->TabStop = false;
this->groupBox4->Text = L"Comandos";
//
// buttonPararAnalise
//
this->buttonPararAnalise->Enabled = false;
this->buttonPararAnalise->Location = System::Drawing::Point(179, 168);
this->buttonPararAnalise->Name = L"buttonPararAnalise";
this->buttonPararAnalise->Size = System::Drawing::Size(34, 23);
this->buttonPararAnalise->TabIndex = 17;
this->buttonPararAnalise->Text = L"X";
this->buttonPararAnalise->UseVisualStyleBackColor = true;
this->buttonPararAnalise->Click += gcnew System::EventHandler(this, &MyForm::buttonPararAnalise_Click);
//
// numericAlfa
//
this->numericAlfa->Location = System::Drawing::Point(131, 171);
this->numericAlfa->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 7, 0, 0, 0 });
this->numericAlfa->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericAlfa->Name = L"numericAlfa";
```

```
this->numericAlfa->ReadOnly = true;
this->numericAlfa->Size = System::Drawing::Size(42, 20);
this->numericAlfa->TabIndex = 3;
this->numericAlfa->TextAlign = System::Windows::Forms::HorizontalAlignment::Center;
this->numericAlfa->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 7, 0, 0, 0 });
//
// buttonLimiar
//
this->buttonLimiar->Location = System::Drawing::Point(6, 168);
this->buttonLimiar->Name = L"buttonLimiar";
this->buttonLimiar->Size = System::Drawing::Size(116, 23);
this->buttonLimiar->TabIndex = 21;
this->buttonLimiar->Text = L"ANALISE LIMIAR";
this->buttonLimiar->UseVisualStyleBackColor = true;
this->buttonLimiar->Click += gcnew System::EventHandler(this, &MyForm::buttonLimiar_Click);
//
// groupBox5
//
this->groupBox5->Controls->Add(this->label5);
this->groupBox5->Controls->Add(this->numericCoeficienteB);
this->groupBox5->Controls->Add(this->label4);
this->groupBox5->Controls->Add(this->checkTanh);
this->groupBox5->Controls->Add(this->numericCoeficienteA);
this->groupBox5->Controls->Add(this->checkBrilhoIm);
this->groupBox5->Controls->Add(this->numericImBright);
this->groupBox5->Controls->Add(this->checkDilate);
this->groupBox5->Controls->Add(this->checkGaussiana);
this->groupBox5->Controls->Add(this->checkBrilhoVQ);
this->groupBox5->Controls->Add(this->numericDilate);
this->groupBox5->Controls->Add(this->checkFiltro2D);
this->groupBox5->Controls->Add(this->numericBlurGauss);
this->groupBox5->Controls->Add(this->numericVQBright);
this->groupBox5->Controls->Add(this->numericKernel);
this->groupBox5->Location = System::Drawing::Point(9, 304);
this->groupBox5->Name = L"groupBox5";
this->groupBox5->Size = System::Drawing::Size(343, 189);
this->groupBox5->TabIndex = 24;
this->groupBox5->TabStop = false;
this->groupBox5->Text = L"Configurações";
//
// label5
//
this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(201, 76);
this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(48, 13);
this->label5->TabIndex = 9;
this->label5->Text = L"b ( x0,1):";
//
// numericCoeficienteB
//
this->numericCoeficienteB->Location = System::Drawing::Point(253, 74);
this->numericCoeficienteB->Minimum = System::Decimal(
                        gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericCoeficienteB->Name = L"numericCoeficienteB";
this->numericCoeficienteB->ReadOnly = true;
this->numericCoeficienteB->Size = System::Drawing::Size(36, 20);
this->numericCoeficienteB->TabIndex = 8;
this->numericCoeficienteB->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 10, 0, 0, 0 });
//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(201, 50);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(48, 13);
this->label4->TabIndex = 7;
this->label4->Text = L"a ( x0,1):";
//
// checkTanh
//
this->checkTanh->AutoSize = true;
this->checkTanh->Location = System::Drawing::Point(171, 26);
this->checkTanh->Name = L"checkTanh";
this->checkTanh->Size = System::Drawing::Size(131, 17);
this->checkTanh->TabIndex = 6;
this->checkTanh->Text = L"Tangente Hiperbólica:";
this->checkTanh->UseVisualStyleBackColor = true;
//
// numericCoeficienteA
//
this->numericCoeficienteA->Location = System::Drawing::Point(253, 48);
this->numericCoeficienteA->Minimum = System::Decimal(
                        gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericCoeficienteA->Name = L"numericCoeficienteA";
this->numericCoeficienteA->ReadOnly = true;
this->numericCoeficienteA->Size = System::Drawing::Size(36, 20);
this->numericCoeficienteA->TabIndex = 5;
this->numericCoeficienteA->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 10, 0, 0, 0 });
//
// checkBrilhoIm
//
this->checkBrilhoIm->AutoSize = true;
this->checkBrilhoIm->Location = System::Drawing::Point(11, 119);
this->checkBrilhoIm->Name = L"checkBrilhoIm";
this->checkBrilhoIm->Size = System::Drawing::Size(110, 17);
this->checkBrilhoIm->TabIndex = 4;
this->checkBrilhoIm->Text = L"Brilho Im ( x0,01 ):";
this->checkBrilhoIm->UseVisualStyleBackColor = true;
```

```
//
// numericImBright
//
this->numericImBright->Location = System::Drawing::Point(121, 118);
this->numericImBright->Minimum = System::Decimal(gcnew cli::array< System::Int32 > (4)
                                                 { 100, 0, 0, System::Int32::MinValue });
this->numericImBright->Name = L"numericImBright";
this->numericImBright->ReadOnly = true;
this->numericImBright->Size = System::Drawing::Size(36, 20);
this->numericImBright->TabIndex = 3;
//
// checkDilate
//
this->checkDilate->AutoSize = true;
this->checkDilate->Location = System::Drawing::Point(11, 96);
this->checkDilate->Name = L"checkDilate";
this->checkDilate->Size = System::Drawing::Size(59, 17);
this->checkDilate->TabIndex = 2;
this->checkDilate->Text = L"Dilatar:";
this->checkDilate->UseVisualStyleBackColor = true;
//
// checkGaussiana
//
this->checkGaussiana->AutoSize = true;
this->checkGaussiana->Location = System::Drawing::Point(11, 73);
this->checkGaussiana->Name = L"checkGaussiana";
this->checkGaussiana->Size = System::Drawing::Size(100, 17);
this->checkGaussiana->TabIndex = 2;
this->checkGaussiana->Text = L"Blur Gaussiano:";
this->checkGaussiana->UseVisualStyleBackColor = true;
//
// checkBrilhoVQ
//
this->checkBrilhoVQ->AutoSize = true;
this->checkBrilhoVQ->Checked = true;
this->checkBrilhoVQ->CheckState = System::Windows::Forms::CheckState::Checked;
this->checkBrilhoVQ->Location = System::Drawing::Point(11, 49);
this->checkBrilhoVQ->Name = L"checkBrilhoVQ";
this->checkBrilhoVQ->Size = System::Drawing::Size(108, 17);
this->checkBrilhoVQ->TabIndex = 2;
this->checkBrilhoVQ->Text = L"Brilho VQ ( x0,1 ):";
this->checkBrilhoVQ->UseVisualStyleBackColor = true;
//
// numericDilate
//
this->numericDilate->Location = System::Drawing::Point(121, 95);
this->numericDilate->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 99, 0, 0, 0 });
this->numericDilate->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericDilate->Name = L"numericDilate";
this->numericDilate->ReadOnly = true;
this->numericDilate->Size = System::Drawing::Size(36, 20);
this->numericDilate->TabIndex = 1;
this->numericDilate->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
//
// checkFiltro2D
//
this->checkFiltro2D->AutoSize = true;
this->checkFiltro2D->Checked = true;
this->checkFiltro2D->CheckState = System::Windows::Forms::CheckState::Checked;
this->checkFiltro2D->Location = System::Drawing::Point(11, 26);
this->checkFiltro2D->Name = L"checkFiltro2D";
this->checkFiltro2D->Size = System::Drawing::Size(68, 17);
this->checkFiltro2D->TabIndex = 2;
this->checkFiltro2D->Text = L"Filtro 2D:";
this->checkFiltro2D->UseVisualStyleBackColor = true;
//
// numericBlurGauss
//
this->numericBlurGauss->Increment = System::Decimal(gcnew cli::array< System::Int32 >(4) { 2, 0, 0, 0 });
this->numericBlurGauss->Location = System::Drawing::Point(121, 72);
this->numericBlurGauss->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 99, 0, 0, 0 });
this->numericBlurGauss->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });
this->numericBlurGauss->Name = L"numericBlurGauss";
this->numericBlurGauss->ReadOnly = true;
this->numericBlurGauss->Size = System::Drawing::Size(36, 20);
this->numericBlurGauss->TabIndex = 1;
this->numericBlurGauss->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

//
// numericVQBright
//
this->numericVQBright->Location = System::Drawing::Point(121, 48);
this->numericVQBright->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 10, 0, 0, 0 });
this->numericVQBright->Name = L"numericVQBright";
this->numericVQBright->ReadOnly = true;
this->numericVQBright->Size = System::Drawing::Size(36, 20);
this->numericVQBright->TabIndex = 1;
this->numericVQBright->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 5, 0, 0, 0 });
//
// numericKernel
//
this->numericKernel->Increment = System::Decimal(gcnew cli::array< System::Int32 >(4) { 2, 0, 0, 0 });
this->numericKernel->Location = System::Drawing::Point(121, 25);
this->numericKernel->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 11, 0, 0, 0 });
this->numericKernel->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 3, 0, 0, 0 });
this->numericKernel->Name = L"numericKernel";
this->numericKernel->ReadOnly = true;
this->numericKernel->Size = System::Drawing::Size(36, 20);
this->numericKernel->TabIndex = 1;
```

```
            this->numericKernel->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 3, 0, 0, 0 });
            //
            // MyForm
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(358, 303);
            this->Controls->Add(this->groupBox5);
            this->Controls->Add(this->groupBox4);
            this->Controls->Add(this->groupBox3);
            this->Controls->Add(this->groupBox2);
            this->Controls->Add(this->groupBox1);
            this->Controls->Add(this->label1);
            this->Controls->Add(this->StatusBox_txtbx);
            this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedSingle;
            this->Name = L"MyForm";
            this->Text = L"Imageador 018";
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->timeInput))->EndInit();
            this->groupBox1->ResumeLayout(false);
            this->groupBox1->PerformLayout();
            this->groupBox2->ResumeLayout(false);
            this->groupBox2->PerformLayout();
            this->groupBox3->ResumeLayout(false);
            this->groupBox3->PerformLayout();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericY))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericX))->EndInit();
            this->groupBox4->ResumeLayout(false);
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericAlfa))->EndInit();
            this->groupBox5->ResumeLayout(false);
            this->groupBox5->PerformLayout();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^> (this->numericCoeficienteB))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^> (this->numericCoeficienteA))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericImBright))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericDilate))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^> (this->numericBlurGauss))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericVQBright))->EndInit();
            (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->numericKernel))->EndInit();
            this->ResumeLayout(false);
            this->PerformLayout();

        }
        #pragma endregion

        #pragma region Botões da Interface
private:
        System::Void buttonTest_Click(System::Object^  sender, System::EventArgs^  e) {

                buttonPCM->Enabled = false;
                buttonDebug->Enabled = false;
                buttonLimiar->Enabled = false;
                buttonPxlTeste->Enabled = false;
                buttonBlcPrincipal->Enabled = false;
                buttonPararBlcPrincipal->Enabled = true;

                blcPrincipalControl = true;

        }

        System::Void buttonPararTest_Click(System::Object^  sender, System::EventArgs^  e) {

                buttonPCM->Enabled = true;
                buttonDebug->Enabled = true;
                buttonLimiar->Enabled = true;
                buttonPxlTeste->Enabled = true;
                buttonBlcPrincipal->Enabled = true;
                buttonPararBlcPrincipal->Enabled = false;

                blcPrincipalControl = false;

        }

        System::Void buttonPxlTeste_Click(System::Object^  sender, System::EventArgs^  e) {

                buttonPCM->Enabled = false;
                buttonDebug->Enabled = false;
                buttonLimiar->Enabled = false;
                buttonPxlTeste->Enabled = false;
                buttonPararPxlTeste->Enabled = true;
                buttonBlcPrincipal->Enabled = false;

                pxlTesteControl = true;

        }

        System::Void buttonPararPxlTeste_Click(System::Object^  sender, System::EventArgs^  e) {

                buttonPCM->Enabled = true;
                buttonDebug->Enabled = true;
                buttonLimiar->Enabled = true;
                buttonPxlTeste->Enabled = true;
                buttonBlcPrincipal->Enabled = true;
                buttonPararPxlTeste->Enabled = false;

                pxlTesteControl = false;

        }
```

```
System::Void buttonPCM_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonPCM->Enabled = false;
        buttonDebug->Enabled = false;
        buttonLimiar->Enabled = false;
        buttonPararPCM->Enabled = true;
        buttonPxlTeste->Enabled = false;
        buttonBlcPrincipal->Enabled = false;

        PCMControl = true;

}

System::Void buttonPararPCM_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonPCM->Enabled = true;
        buttonDebug->Enabled = true;
        buttonLimiar->Enabled = true;
        buttonPxlTeste->Enabled = true;
        buttonPararPCM->Enabled = false;
        buttonBlcPrincipal->Enabled = true;

        PCMControl = false;
}

System::Void buttonDebug_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonPCM->Enabled = false;
        buttonDebug->Enabled = false;
        buttonLimiar->Enabled = false;
        buttonPxlTeste->Enabled = false;
        buttonPararDebug->Enabled = true;
        buttonBlcPrincipal->Enabled = false;

        radioDebugIm->Enabled = true;
        radioDebugVq->Enabled = true;
        radioDebugIndex->Enabled = true;
        radioDebugMuHat->Enabled = true;

        debugControl = true;

        this->Width = WINDOW_WIDTH_DEBUG;

}

System::Void buttonPararDebug_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonPCM->Enabled = true;
        buttonDebug->Enabled = true;
        buttonLimiar->Enabled = true;
        buttonPxlTeste->Enabled = true;
        buttonPararDebug->Enabled = false;
        buttonBlcPrincipal->Enabled = true;

        radioDebugIm->Enabled = false;
        radioDebugVq->Enabled = false;
        radioDebugIndex->Enabled = false;
        radioDebugMuHat->Enabled = false;

        debugControl = false;

        this->Width = WINDOW_WIDTH_NORMAL;

}

System::Void buttonConfig_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonConfig->Enabled = false;
        buttonPararConfig->Enabled = true;

        this->Height = WINDOW_HEIGHT_CONFIG;

}

System::Void buttonPararConfig_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonConfig->Enabled = true;
        buttonPararConfig->Enabled = false;

        this->Height = WINDOW_HEIGHT_NORMAL;
}

System::Void buttonLimiar_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonPCM->Enabled = false;
        buttonDebug->Enabled = false;
        buttonConfig->Enabled = false;
        buttonLimiar->Enabled = false;
        buttonPxlTeste->Enabled = false;
        buttonBlcPrincipal->Enabled = false;
        buttonPararAnalise->Enabled = true;

        radioDebugIm->Enabled = false;
        radioDebugVq->Enabled = false;
        radioDebugIndex->Enabled = false;
        radioDebugMuHat->Enabled = false;

        limiarControl = true;
```

```
        this->Width = WINDOW_WIDTH_DEBUG;

}

System::Void buttonPararAnalise_Click(System::Object^  sender, System::EventArgs^  e) {

        buttonPCM->Enabled = true;
        buttonDebug->Enabled = true;
        buttonConfig->Enabled = true;
        buttonLimiar->Enabled = true;
        buttonPxlTeste->Enabled = true;
        buttonBlcPrincipal->Enabled = true;
        buttonPararAnalise->Enabled = false;

        radioDebugIm->Enabled = true;
        radioDebugVq->Enabled = true;
        radioDebugIndex->Enabled = true;
        radioDebugMuHat->Enabled = true;

        limiarControl = false;
        sairAnaliseLimiar = true;

        this->Width = WINDOW_WIDTH_NORMAL;

}

#pragma endregion

void parleyCommunication(String^ text, int function) {

        if (this->textBoxDebug->InvokeRequired)
        {
                parleyDelegate^ d = gcnew parleyDelegate(this, &MyForm::parleyCommunication);
                this->Invoke(d, gcnew array<Object^> {text,function});
        }
        else
        {
                if (function == MAT_TEXT)
                        this->textBoxDebug->Text = text;
                if (function == MAT_16x16){
                        this->numericX->Value = 1;
                        this->numericY->Value = 1;
                        this->numericX->Maximum = 1;
                        this->numericY->Maximum = 1;
                }
                if (function == MAT_64x64){
                        this->numericX->Value = 1;
                        this->numericY->Value = 1;
                        this->numericX->Maximum = 4;
                        this->numericY->Maximum = 4;
                }
                if (function == BACK_TO_NORMAL){
                        buttonPCM->Enabled = true;
                        buttonDebug->Enabled = true;
                        buttonConfig->Enabled = true;
                        buttonLimiar->Enabled = true;
                        buttonPxlTeste->Enabled = true;
                        buttonBlcPrincipal->Enabled = true;
                        buttonPararAnalise->Enabled = false;
                }
                if (function == REC_OFF){
                        checkREC->Checked = false;
                }
        }
}

System::Void ReadWriteThread_DoWork(System::Object^  sender, System::ComponentModel::DoWorkEventArgs^  e) {

        unsigned char Buffer [64];
        DWORD ActualLength;
        cv::Mat1d dadosLidos;
        int position,contador;

        cv::Mat C_dpcm = (cv::Mat_<double>(8, 1) << 0.0063, 0.0250, 0.0563, 0.1000,
                                        0.1500, 0.2250, 0.3250, 0.4688);

        cv::Mat1d offsetDefault = cv::Mat::ones(1, 16, CV_64F) * 0.4688;

        cv::Mat1d Cnovo = ProcessamentoDeArquivos::lerMatC();

        cv::Mat H = (cv::Mat_<double>(5, 16) <<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,
                                               4,  2, -2, -4,  2,  1, -1, -2, -2, -1,  1,  2, -4, -2,  2,  4);

        cv::Mat H_t;
        cv::transpose(H, H_t);

        cv::Mat diag = (cv::Mat_<double>(5, 5) <<
                                    0.025,       0,             0,             0,             0,
                                        0,   0.025,             0,             0,             0,
                                        0,       0,        0.0625,             0,             0,
                                        0,       0,             0,        0.0625,             0,
                                        0,       0,             0,             0,        0.0100);
```

```
while(true)
{
	if(AttachedState == TRUE)	//Não tentar acionar a comunicação USB sem que o dispositivo esteja
	  adequadamente conectado e configurado.
	{
		#pragma region Comando do BLOCO PRINCIPAL
		if (blcPrincipalControl)
		{
			contador = 0;

			cv::namedWindow("Reconstrução da Imagem - BLOCO PRINCIPAL", cv::WINDOW_AUTOSIZE);

			cv::Mat1d offsetLoad;
			if (std::ifstream("Offset.txt")){
				if (!ProcessamentoDeArquivos::carregarInfo("Offset.txt", 16, offsetLoad)){
					ProcessamentoDeArquivos::sendMessageError(OFFSET_FILE,
                                                                FILE_NOT_OK);
				}
			}
			else{
				offsetLoad = offsetDefault;
				ProcessamentoDeArquivos::sendMessageError(OFFSET_FILE, FILE_NOT_FOUND);
			}

			cv::Mat1d tempoIntegracao;
			if (std::ifstream("Tempo_Integracao.txt")){
				if (!ProcessamentoDeArquivos::carregarInfo("Tempo_Integracao.txt", 16,
                                                                tempoIntegracao)){
					ProcessamentoDeArquivos::sendMessageError(TINT_FILE, FILE_NOT_OK);
				}
			}
			else{
				tempoIntegracao = cv::Mat::ones(1, 16, CV_64F) * 400;
				ProcessamentoDeArquivos::sendMessageError(TINT_FILE, FILE_NOT_FOUND);
			}

			cv::Mat1d MuHat = cv::Mat::zeros(1, N_BLOCOS + 1, CV_64F);

			cv::Mat sumImagens = cv::Mat::zeros(64, 64, CV_64F);

			vector<cv::Mat> DC1(3);

			bool recIsActive = false;

			while (blcPrincipalControl)
			{
				//Zerar algumas variáveis que são reutilizadas dentro de um mesmo ciclo
				dadosLidos = cv::Mat::zeros(1, 4800, CV_64F);
				position = 0;
				for (int i = 0; i < 16; i++)
				{
					//Os buffers 2 até 5 indicam qual linha está sendo lida. O buffer 6
					  garante que a corrente será ou não limitada.
					if (i % 2 == 0)
						Buffer[5] = '0';
					else
						Buffer[5] = '1';

					if (i % 4 <= 1)
						Buffer[4] = '0';
					else
						Buffer[4] = '1';

					if (i % 8 <= 3)
						Buffer[3] = '0';
					else
						Buffer[3] = '1';

					if (i < 8)
						Buffer[2] = '0';
					else
						Buffer[2] = '1';

					if (checkLimit->Checked)
						Buffer[6] = '1';
					else
						Buffer[6] = '0';

					//Envio do tempo de integração(T_int) desejado.
					if (radioTempoBox->Checked){
						// T_int definido via interface direta.
						Buffer[7] = (byte)(((int)timeInput->Value) / 100);
						// T_int += Buffer[7] * 100us;
						Buffer[8] = (byte)(((int)timeInput->Value) / 10 % 10);
						// T_int += Buffer[8] * 10us;
					}
					if (radioTempoLoad->Checked){
						// T_int definido via arquivo de texto.
						Buffer[7] = (byte)(((int)tempoIntegracao.at<double>
                                                    (0, i)) / 100);
						// T_int += Buffer[7] * 100us;
						Buffer[8] = (byte)(((int)tempoIntegracao.at<double>
                                                    (0, i)) / 10 % 10);
						// T_int += Buffer[8] * 10us;
					}

					Buffer[0] = ATIVAR_BLOCO_PRINCIPAL;
```

```
        MPUSBWrite(EP1OUTHandle, Buffer, 9, &ActualLength, INFINITE);
        //Envia o comando definido via USB para o dispositivo.
        MPUSBRead(EP1INHandle, Buffer, 45, &ActualLength, INFINITE);
        //Recebe a resposta do firmware em relação ao comando enviado.

        for(int j = 0; j < 38; j++)
        {
                //Acumula cada um dos bytes lidos em uma
                  variável(dadosLidos) que representa a imagem obtida.
                ProcessamentoDeDados::carregarByte(Buffer[j + 2],
                                                   dadosLidos, position, j);
                if (j != 37)
                        position += 8;
                else
                        position += 4;
        }

}
//<------0,04s
cv::Mat aux_Y;
if (dadosLidos.rows*dadosLidos.cols != 4800){
        aux_Y = cv::Mat::ones(300, 16, CV_64F);
}
else{
        aux_Y = ProcessamentoDeDados::matlabReshape(dadosLidos, 300, 16);
}

cv::Mat aux_DC;
cv::transpose(aux_Y(cv::Range(0, 12), cv::Range(0, 16)), aux_DC);

cv::transpose(aux_Y(cv::Range(12, 300), cv::Range(0, 16)), aux_Y);
cv::flip(aux_Y, aux_Y, 1);

cv::Mat S1, B1, D1;
ProcessamentoDeDados::processData(aux_Y, aux_DC, S1, B1, D1, DC1);

cv::Mat1d Phat = ProcessamentoDeDados::getPhat(Cnovo, B1, S1);

cv::Mat1d Xhat = H_t*diag*Phat;

cv::Mat1d Y = ProcessamentoDeDados::getY(Xhat);

cv::Mat1d offset;
if (checkOffset->Checked)
        offset = offsetLoad;
else
        offset = offsetDefault;

int indexTerm[N_BLOCOS];
ProcessamentoDeDados::processDPCM(D1, DC1, C_dpcm, offset,
                                  checkCorrecao->Checked,
                                  checkLimitDPCM->Checked, MuHat, indexTerm);

cv::Mat1d Im = ProcessamentoDeDados::getIm(MuHat);
//<---- 0,12s

if (checkFlipLR->Checked){
        cv::flip(Im, Im, 1);
        cv::flip(Y, Y, 1);
}
if (checkFlipUD->Checked){
        cv::flip(Im, Im, 0);
        cv::flip(Y, Y, 0);
}

cv::Mat1d imagemFinal = Y + Im ;

if (checkFiltro2D->Checked){
        unsigned kernel_size = (int)numericKernel->Value;
        cv::filter2D(imagemFinal, imagemFinal, -1,
                (cv::Mat::ones(kernel_size, kernel_size, CV_64F) /
                (kernel_size*kernel_size)));
}
if (checkGaussiana->Checked){
        cv::Mat kernel_gauss = (cv::Mat_<double>(3, 3) <<
                                        1, 2, 1,
                                        2, 4, 2,
                                        1, 2, 1 );
        kernel_gauss = kernel_gauss / 16;
        cv::filter2D(imagemFinal, imagemFinal, -1, kernel_gauss);
}
if (checkDilate->Checked)
        cv::dilate(imagemFinal, imagemFinal, (cv::Mat::ones(3, 3, CV_64F) /
                                              (int)numericDilate->Value));

if (checkBrilhoVQ->Checked)
        Y = Y + (0.1 * (int)numericVQBright->Value);

if (checkTanh->Checked){
        for (int i = 0; i < imagemFinal.rows; i++){
                for (int j = 0; j < imagemFinal.cols; j++)
                        imagemFinal.at<double>(i, j) =
                            (0.1*(float)numericCoeficienteA->Value) *
                           tanh((0.1*(float)numericCoeficienteB->Value)*
                           imagemFinal.at<double>(i, j));
        }

}
if (checkBrilhoIm->Checked)
```

```
                        imagemFinal = imagemFinal + (0.01 * (int)numericImBright->Value);

                if (checkREC->Checked && !recIsActive){
                        contador = 0;
                        sumImagens = cv::Mat::zeros(64, 64, CV_64F);
                        recIsActive = true;
                }

                contador++;

                if (recIsActive){
                        string nomeArquivo = "captura(" +
                                                std::to_string(contador) + ").txt";
                        ProcessamentoDeArquivos::imprimirMat(dadosLidos, nomeArquivo);
                }
                if (recIsActive && (contador == 10)){
                        recIsActive = false;
                        parleyCommunication(" ", REC_OFF);
                }

                sumImagens = sumImagens + imagemFinal;
                cv::Mat1d imagemMedia = sumImagens / contador;

                cv::Mat1d imagemTotal = ProcessamentoDeDados::prepareFinalImage(Im, Y,
                                                                imagemFinal, imagemMedia);

                //<---- 0,12s no modo default
                cv::imshow("Reconstrução da Imagem - BLOCO PRINCIPAL", imagemTotal);
                cvWaitKey(40);
                //<---- 0,16s
        }

        Buffer[0] = CHIP_OFF;

        MPUSBWrite(EP1OUTHandle, Buffer, 6, &ActualLength, INFINITE);
        //Send the command now over USB
        MPUSBRead(EP1INHandle, Buffer, 1, &ActualLength, INFINITE);
        //Receive the answer from the device firmware through USB

} // fim do IF referente ao programa estar com o comando BLOCO PRINCIPAL ativado
#pragma endregion

#pragma region Comando do PIXEL TESTE
if (pxlTesteControl)
{
        while (pxlTesteControl)
        {
                Buffer[0] = ATIVAR_PIXELS_TESTE;

                // T_int definido via interface direta.
                Buffer[7] = (byte)(((int)timeInput->Value) / 100);
                // T_int += Buffer[7] * 100us;
                Buffer[8] = (byte)(((int)timeInput->Value) / 10 % 10);
                // T_int += Buffer[8] * 10us;


                MPUSBWrite(EP1OUTHandle, Buffer, 9, &ActualLength, INFINITE);
                //Send the command now over USB
                MPUSBRead(EP1INHandle, Buffer, 4, &ActualLength, INFINITE);
                //Receive the answer from the device firmware through USB
        }



        Buffer[0] = CHIP_OFF;

        MPUSBWrite(EP1OUTHandle, Buffer, 6, &ActualLength, INFINITE);
        //Send the command now over USB
        MPUSBRead(EP1INHandle, Buffer, 1, &ActualLength, INFINITE);
        //Receive the answer from the device firmware through USB
} // fim do IF referente ao programa estar com o comando PIXEL TESTE ativado
#pragma endregion

#pragma region Comando do PCM
if (PCMControl)
{
        contador = 0;

        cv::namedWindow("Reconstrução da Imagem - PCM", cv::WINDOW_AUTOSIZE);

        cv::Mat1d offsetLoad;
        if (std::ifstream("Offset.txt")){
                if (!ProcessamentoDeArquivos::carregarInfo("Offset.txt", 16, offsetLoad)){
                        ProcessamentoDeArquivos::sendMessageError(OFFSET_FILE,
                                                                FILE_NOT_OK);
                }
        }
        else{
                offsetLoad = offsetDefault;
                ProcessamentoDeArquivos::sendMessageError(OFFSET_FILE, FILE_NOT_FOUND);
        }

        cv::Mat1d tempoIntegracao;
        if (std::ifstream("Tempo_Integracao.txt")){
                if (!ProcessamentoDeArquivos::carregarInfo("Tempo_Integracao.txt", 16,
                                                        tempoIntegracao)){
                        ProcessamentoDeArquivos::sendMessageError(TINT_FILE, FILE_NOT_OK);
                }
        }
```

```
else{
	tempoIntegracao = cv::Mat::ones(1, 16, CV_64F) * 400;
	ProcessamentoDeArquivos::sendMessageError(TINT_FILE, FILE_NOT_FOUND);
}

cv::Mat1d MuHat = cv::Mat::zeros(1, N_BLOCOS + 1, CV_64F);

cv::Mat sumImagens = cv::Mat::zeros(64, 64, CV_64F);

vector<cv::Mat> DC1(3);

while (PCMControl)
{
	//Limpar algumas variáveis que são reutilizadas dentro de um mesmo ciclo
	dadosLidos = cv::Mat::zeros(1, 4800, CV_64F);
	position = 0;

	for (int i = 0; i < 16; i++)
	{
		//Os buffers 2,3, 4 e 5 definirao qual linha esta sendo lida
		if (i % 2 == 0)
			Buffer[5] = '0';
		else
			Buffer[5] = '1';

		if (i % 4 <= 1)
			Buffer[4] = '0';
		else
			Buffer[4] = '1';

		if (i % 8 <= 3)
			Buffer[3] = '0';
		else
			Buffer[3] = '1';

		if (i < 8)
			Buffer[2] = '0';
		else
			Buffer[2] = '1';

		if (checkLimit->Checked)
			Buffer[6] = '1';
		else
			Buffer[6] = '0';

		//Envio do tempo de integração(T_int) desejado.
		if (radioTempoBox->Checked){
			// T_int definido via interface direta.
			Buffer[7] = (byte)(((int)timeInput->Value) / 100);
			// T_int += Buffer[7] * 100us;
			Buffer[8] = (byte)(((int)timeInput->Value) / 10 % 10);
			// T_int += Buffer[8] * 10us;
		}
		if (radioTempoLoad->Checked){
			// T_int definido via arquivo de texto.
			Buffer[7] = (byte)(((int)tempoIntegracao.at<double>
								(0, i)) / 100);
			// T_int += Buffer[7] * 100us;
			Buffer[8] = (byte)(((int)tempoIntegracao.at<double>
								(0, i)) / 10 % 10);

			// T_int += Buffer[8] * 10us;
		}

		Buffer[0] = ATIVAR_BLOCO_PRINCIPAL;

		MPUSBWrite(EP1OUTHandle, Buffer, 9, &ActualLength, INFINITE);
		//Send the command now over USB
		MPUSBRead(EP1INHandle, Buffer, 45, &ActualLength, INFINITE);
		//Receive the answer from the device firmware through USB

		for (int j = 0; j < 38; j++)
		{
			//Acumula cada um dos bytes lidos em uma
			  variável(dadosLidos) que representa a imagem obtida.
			ProcessamentoDeDados::carregarByte(Buffer[j + 2],
					dadosLidos, position, j);
			if (j != 37)
				position += 8;
			else
				position += 4;
		}

	}

	cv::Mat aux_Y;
	if (dadosLidos.rows*dadosLidos.cols != 4800){
		aux_Y = cv::Mat::ones(300, 16, CV_64F);
	}
	else{
		aux_Y = ProcessamentoDeDados::matlabReshape(dadosLidos, 300, 16);
	}

	cv::Mat aux_DC;
	cv::transpose(aux_Y(cv::Range(0, 12), cv::Range(0, 16)), aux_DC);

	cv::transpose(aux_Y(cv::Range(12, 300), cv::Range(0, 16)), aux_Y);
	cv::flip(aux_Y, aux_Y, 1);
```

```
                cv::Mat S1, B1, D1;
                ProcessamentoDeDados::processData(aux_Y, aux_DC, S1, B1, D1, DC1);

                cv::Mat1d Phat = ProcessamentoDeDados::getPhat(Cnovo, B1, S1);

                cv::Mat1d Xhat = H_t*diag*Phat;

                cv::Mat1d Y = ProcessamentoDeDados::getY(Xhat);

                cv::Mat1d offset;
                if (checkOffset->Checked)
                        offset = offsetLoad;
                else
                        offset = offsetDefault;

                int indexTerm[N_BLOCOS];
                ProcessamentoDeDados::processPCM(D1, DC1, C_dpcm, offset,
                                                 checkCorrecao->Checked,
                                                 MuHat, indexTerm);

                cv::Mat1d Im = ProcessamentoDeDados::getIm(MuHat);


                if (checkFlipLR->Checked){
                        cv::flip(Im, Im, 1);
                        cv::flip(Y, Y, 1);
                }
                if (checkFlipUD->Checked){
                        cv::flip(Im, Im, 0);
                        cv::flip(Y, Y, 0);
                }

                Im = 2 * Im;
                cv::Mat1d imagemFinal = Y + Im;

                if (checkFiltro2D->Checked){
                        unsigned kernel_size = (int)numericKernel->Value;
                        cv::filter2D(imagemFinal, imagemFinal, -1,
                                 (cv::Mat::ones(kernel_size, kernel_size, CV_64F) /
                                 (kernel_size*kernel_size)));
                }
                if (checkGaussiana->Checked){
                        cv::Mat kernel_gauss = (cv::Mat_<double>(3, 3) << 1, 2, 1,
                                2, 4, 2,
                                1, 2, 1);
                        kernel_gauss = kernel_gauss / 16;
                        cv::filter2D(imagemFinal, imagemFinal, -1, kernel_gauss);
                }
                if (checkDilate->Checked)
                        cv::dilate(imagemFinal, imagemFinal, (cv::Mat::ones(3, 3, CV_64F) /
                                                              (int)numericDilate->Value));

                if (checkBrilhoVQ->Checked)
                        Y = Y + (0.1 * (int)numericVQBright->Value);

                if (checkTanh->Checked){
                        for (int i = 0; i < imagemFinal.rows; i++){
                                for (int j = 0; j < imagemFinal.cols; j++)
                                        imagemFinal.at<double>(i, j) =
                                             (0.1*(float)numericCoeficienteA->Value) *
                                              tanh((0.1*(float)numericCoeficienteB->Value)*
                                              imagemFinal.at<double>(i, j));
                        }

                }
                if (checkBrilhoIm->Checked)
                        imagemFinal = imagemFinal + (0.01 * (int)numericImBright->Value);

                contador++;

                sumImagens = sumImagens + imagemFinal;
                cv::Mat1d imagemMedia = sumImagens / contador;

                cv::Mat1d imagemTotal = ProcessamentoDeDados::prepareFinalImage(Im, Y,
                                                          imagemFinal, imagemMedia);

                cv::imshow("Reconstrução da Imagem - PCM", imagemTotal);
                cvWaitKey(40);
        }

        Buffer[0] = CHIP_OFF;

        MPUSBWrite(EP1OUTHandle, Buffer, 6, &ActualLength, INFINITE);
        //Send the command now over USB
        MPUSBRead(EP1INHandle, Buffer, 1, &ActualLength, INFINITE);
        //Receive the answer from the device firmware through USB

} // fim do IF referente ao programa estar com o comando PCM ativado
#pragma endregion

#pragma region Comando do DEBUG
if (debugControl)
{

        cv::Mat1d offsetLoad;
        if (std::ifstream("Offset.txt")){
                if (!ProcessamentoDeArquivos::carregarInfo("Offset.txt", 16, offsetLoad)){
                        ProcessamentoDeArquivos::sendMessageError(OFFSET_FILE,
                                                                  FILE_NOT_OK);
```

```
        }
}
else{
        offsetLoad = offsetDefault;
        ProcessamentoDeArquivos::sendMessageError(OFFSET_FILE, FILE_NOT_FOUND);
}

cv::Mat1d tempoIntegracao;
if (std::ifstream("Tempo_Integracao.txt")){
        if (!ProcessamentoDeArquivos::carregarInfo("Tempo_Integracao.txt", 16,
                                                   tempoIntegracao)){
                ProcessamentoDeArquivos::sendMessageError(TINT_FILE, FILE_NOT_OK);
        }
}
else{
        tempoIntegracao = cv::Mat::ones(1, 16, CV_64F) * 400;
        ProcessamentoDeArquivos::sendMessageError(TINT_FILE, FILE_NOT_FOUND);
}

cv::Mat1d MuHat = cv::Mat::zeros(1, N_BLOCOS + 1, CV_64F);

cv::namedWindow("Reconstrução da Imagem - DEBUG", cv::WINDOW_AUTOSIZE);

cv::Mat sumImagens = cv::Mat::zeros(64, 64, CV_64F);

vector<cv::Mat> DC1(3);

cv::Mat1d temp;

System::String^ texto;

unsigned selectionValue = 0;

while (debugControl)
{

        //Limpar algumas variáveis que são reutilizadas dentro de um mesmo ciclo
        dadosLidos = cv::Mat::zeros(1, 4800, CV_64F);
        position = 0;

        for (int i = 0; i < 16; i++)
        {
                //Os buffers 2,3, 4 e 5 definirao qual linha esta sendo lida
                if (i % 2 == 0)
                        Buffer[5] = '0';
                else
                        Buffer[5] = '1';

                if (i % 4 <= 1)
                        Buffer[4] = '0';
                else
                        Buffer[4] = '1';

                if (i % 8 <= 3)
                        Buffer[3] = '0';
                else
                        Buffer[3] = '1';

                if (i < 8)
                        Buffer[2] = '0';
                else
                        Buffer[2] = '1';

                if (checkLimit->Checked)
                        Buffer[6] = '1';
                else
                        Buffer[6] = '0';

                //Envio do tempo de integração(T_int) desejado.
                if (radioTempoBox->Checked){
                        // T_int definido via interface direta.
                        Buffer[7] = (byte)(((int)timeInput->Value) / 100);
                        // T_int += Buffer[7] * 100us;
                        Buffer[8] = (byte)(((int)timeInput->Value) / 10 % 10);
                        // T_int += Buffer[8] * 10us;
                }
                if (radioTempoLoad->Checked){
                        // T_int definido via arquivo de texto.
                        Buffer[7] = (byte)(((int)tempoIntegracao.at<double>
                                                        (0, i)) / 100);
                        // T_int += Buffer[7] * 100us;
                        Buffer[8] = (byte)(((int)tempoIntegracao.at<double>
                                                        (0, i)) / 10 % 10);

                        // T_int += Buffer[8] * 10us;
                }

                Buffer[0] = ATIVAR_BLOCO_PRINCIPAL;

                MPUSBWrite(EP1OUTHandle, Buffer, 9, &ActualLength, INFINITE);
                //Send the command now over USB
                MPUSBRead(EP1INHandle, Buffer, 45, &ActualLength, INFINITE);
                //Receive the answer from the device firmware through USB

                for (int j = 0; j < 38; j++)
                {
                        //Carrega cada um dos bytes lidos para o montante deles já
                          salvo
```

```
                ProcessamentoDeDados::carregarByte(Buffer[j + 2],
                                        dadosLidos, position, j);
                if (j != 37)
                        position += 8;
                else
                        position += 4;
        }

}

cv::Mat aux_Y;
if (dadosLidos.rows*dadosLidos.cols != 4800){
        aux_Y = cv::Mat::ones(300, 16, CV_64F);
}
else{
        aux_Y = ProcessamentoDeDados::matlabReshape(dadosLidos, 300, 16);
}

cv::Mat aux_DC;
cv::transpose(aux_Y(cv::Range(0, 12), cv::Range(0, 16)), aux_DC);

cv::transpose(aux_Y(cv::Range(12, 300), cv::Range(0, 16)), aux_Y);
cv::flip(aux_Y, aux_Y, 1);

cv::Mat S1, B1, D1;
ProcessamentoDeDados::processData(aux_Y, aux_DC, S1, B1, D1, DC1);

cv::Mat1d Phat = ProcessamentoDeDados::getPhat(Cnovo, B1, S1);

cv::Mat1d Xhat = H_t*diag*Phat;

cv::Mat1d Y = ProcessamentoDeDados::getY(Xhat);

cv::Mat1d offset;
if (checkOffset->Checked)
        offset = offsetLoad;
else
        offset = offsetDefault;

int indexTerm[N_BLOCOS];
ProcessamentoDeDados::processDPCM(D1, DC1, C_dpcm, offset,
                        checkCorrecao->Checked,checkLimitDPCM->Checked,
                        MuHat, indexTerm);

cv::Mat1d Im = ProcessamentoDeDados::getIm(MuHat);

if (checkFlipLR->Checked){
        cv::flip(Im, Im, 1);
        cv::flip(Y, Y, 1);
}
if (checkFlipUD->Checked){
        cv::flip(Im, Im, 0);
        cv::flip(Y, Y, 0);
}

cv::Mat1d imagemFinal = Y + Im;

if (checkFiltro2D->Checked){
        unsigned kernel_size = (int)numericKernel->Value;
        cv::filter2D(imagemFinal, imagemFinal, -1,
                (cv::Mat::ones(kernel_size, kernel_size, CV_64F) /
                (kernel_size*kernel_size)));
}
if (checkGaussiana->Checked){
        cv::Mat kernel_gauss = (cv::Mat_<double>(3, 3) << 1, 2, 1,
                2, 4, 2,
                1, 2, 1);
        kernel_gauss = kernel_gauss / 16;
        cv::filter2D(imagemFinal, imagemFinal, -1, kernel_gauss);
}
if (checkDilate->Checked)
        cv::dilate(imagemFinal, imagemFinal, (cv::Mat::ones(3, 3, CV_64F) /
                                                (int)numericDilate->Value));

if (checkBrilhoVQ->Checked)
        Y = Y + (0.1 * (int)numericVQBright->Value);

if (checkTanh->Checked){
        for (int i = 0; i < imagemFinal.rows; i++){
                for (int j = 0; j < imagemFinal.cols; j++)
                        imagemFinal.at<double>(i, j) =
                          (0.1*(float)numericCoeficienteA->Value) *
                          tanh((0.1*(float)numericCoeficienteB->Value)*
                          imagemFinal.at<double>(i, j));
        }

}

if (checkBrilhoIm->Checked)
        imagemFinal = imagemFinal + (0.01 * (int)numericImBright->Value);

contador++;

sumImagens = sumImagens + imagemFinal;
cv::Mat1d imagemMedia = sumImagens / contador;

cv::Mat1d imagemTotal = ProcessamentoDeDados::prepareFinalImage(Im, Y,
                                                imagemFinal, imagemMedia);
```

```
                cv::imshow("Reconstrução da Imagem - DEBUG", imagemTotal);
                cvWaitKey(40);

                if (radioDebugMuHat->Checked){
                        if (selectionValue != 1)
                        {
                                selectionValue = 1;
                                parleyCommunication(" ", MAT_16x16);
                        }
                        temp = ProcessamentoDeDados::matlabReshape(MuHat(cv::Range::all(),
                                                cv::Range(1, N_BLOCOS + 1)), 16, 16);
                        texto = ProcessamentoDeDados::matToString(temp, 3);
                }
                if (radioDebugIndex->Checked){
                        if (selectionValue != 2)
                        {
                                selectionValue = 2;
                                parleyCommunication(" ", MAT_16x16);
                        }
                        for (int i = 0; i < 256; i++)
                                indexTerm[i] = indexTerm[i] *
                                                (int)(D1.at<double>(0, i) * 2 - 1);
                        temp = ProcessamentoDeDados::indexTermToMat(indexTerm);
                        texto = ProcessamentoDeDados::matToString(temp, 0);
                }
                if (radioDebugIm->Checked){
                        if (selectionValue != 3)
                        {
                                selectionValue = 3;
                                parleyCommunication(" ", MAT_64x64);
                        }
                        int x, y;
                        y = 16 * ((int)numericY->Value - 1);
                        x = 16 * ((int)numericX->Value - 1);
                        texto = ProcessamentoDeDados::matToString(Im(cv::Range(y, y + 16),
                                                        cv::Range(x, x + 16)), 3);

                }
                if (radioDebugVq->Checked){
                        if (selectionValue != 4)
                        {
                                selectionValue = 4;
                                parleyCommunication(" ", MAT_64x64);
                        }
                        int x, y;
                        y = 16 * ((int)numericY->Value - 1);
                        x = 16 * ((int)numericX->Value - 1);
                        texto = ProcessamentoDeDados::matToString(Y(cv::Range(y, y + 16),
                                                        cv::Range(x, x + 16)), 3);

                }

                parleyCommunication(texto, MAT_TEXT);

                Sleep(250);
        }

} // fim do IF referente ao programa estar com o comando DEBUG ativado
#pragma endregion

#pragma region Comando do ANALISE LIMIAR
if (limiarControl)
{
        contador = 0;

        cv::Mat1d MuHat = cv::Mat::zeros(1, N_BLOCOS + 1, CV_64F);

        vector<cv::Mat> DC1(3);

        cv::Mat1d result = cv::Mat::zeros(16, 16, CV_64F);
        cv::Mat1d ocorrencia = cv::Mat::zeros(16, 16, CV_64F);
        cv::Mat1d mask = cv::Mat::zeros(16, 16, CV_64F);

        sairAnaliseLimiar = false;
        int Tint = TINT_VALUE;
        double limiar = (double)numericAlfa->Value;
        double maxVal;
        double minVal;
        System::String^ texto;

        boolean terminouAnaliseLimiar = false;

        parleyCommunication(" ", MAT_16x16);

        texto = ProcessamentoDeDados::matToString(result, 0);
        parleyCommunication(texto, MAT_TEXT);

        while (!sairAnaliseLimiar && !terminouAnaliseLimiar)
        {
                //Limpar algumas variáveis que são reutilizadas dentro de um mesmo ciclo
                dadosLidos = cv::Mat::zeros(1, 4800, CV_64F);
                position = 0;

                for (int i = 0; i < 16; i++)
                {
                        //Os buffers 2,3, 4 e 5 definirao qual linha esta sendo lida
                        if (i % 2 == 0)
                                Buffer[5] = '0';
                        else
```

```
                Buffer[5] = '1';

        if (i % 4 <= 1)
                Buffer[4] = '0';
        else
                Buffer[4] = '1';

        if (i % 8 <= 3)
                Buffer[3] = '0';
        else
                Buffer[3] = '1';

        if (i < 8)
                Buffer[2] = '0';
        else
                Buffer[2] = '1';

        Buffer[6] = '1';


        Buffer[7] = (byte)(Tint / 100);
        Buffer[8] = (byte)(Tint / 10 % 10);

        Buffer[0] = ATIVAR_BLOCO_PRINCIPAL;

        MPUSBWrite(EP1OUTHandle, Buffer, 9, &ActualLength, INFINITE);
        //Send the command now over USB
        MPUSBRead(EP1INHandle, Buffer, 45, &ActualLength, INFINITE);
        //Receive the answer from the device firmware through USB

        for (int j = 0; j < 38; j++)
        {
                //Carrega cada um dos bytes lidos para o montante deles já
                  salvo
                ProcessamentoDeDados::carregarByte(Buffer[j + 2],
                                                   dadosLidos, position, j);
                if (j != 37)
                        position += 8;
                else
                        position += 4;
        }

}

cv::Mat aux_Y;
if (dadosLidos.rows*dadosLidos.cols != 4800){
        aux_Y = cv::Mat::ones(300, 16, CV_64F);
}
else{
        aux_Y = ProcessamentoDeDados::matlabReshape(dadosLidos, 300, 16);
}

cv::Mat aux_DC;
cv::transpose(aux_Y(cv::Range(0, 12), cv::Range(0, 16)), aux_DC);

cv::transpose(aux_Y(cv::Range(12, 300), cv::Range(0, 16)), aux_Y);
cv::flip(aux_Y, aux_Y, 1);

cv::Mat S1, B1, D1;
ProcessamentoDeDados::processData(aux_Y, aux_DC, S1, B1, D1, DC1);

cv::Mat1d Phat = ProcessamentoDeDados::getPhat(Cnovo, B1, S1);

cv::Mat1d Xhat = H_t*diag*Phat;

cv::Mat1d Y = ProcessamentoDeDados::getY(Xhat);

int indexTerm[N_BLOCOS];
ProcessamentoDeDados::processDPCM(D1, DC1, C_dpcm,
                        cv::Mat::zeros(1, 16, CV_64F), false, false,
                        MuHat, indexTerm);

for (int i = 0; i < 256; i++)
        indexTerm[i] = indexTerm[i] * (int)(D1.at<double>(0, i) * 2 - 1);
cv::Mat1d term = ProcessamentoDeDados::indexTermToMat(indexTerm);

contador++;

for (int i = 0; i < term.cols; i++){
        for (int j = 0; j < term.rows; j++){
                if (term.at<double>(i, j) > limiar)
                //Analisa a linha (LINHA + 1) e a coluna (COLUNA + 1)
                        ocorrencia.at<double>(i, j) =
                                        ocorrencia.at<double>(i, j) + 1;
        }
}

cv::minMaxLoc(ocorrencia, 0, &maxVal, 0, 0);

if (contador == 2 && maxVal == 0){
        Tint += 10;
        contador = 0;
        ocorrencia = cv::Mat::zeros(16, 16, CV_64F);
}

if (contador == 5 && maxVal < 3){
        Tint += 10;
        contador = 0;
        ocorrencia = cv::Mat::zeros(16, 16, CV_64F);
```

```
                                        }

                                        if (contador == 15){
                                                for (int i = 0; i < ocorrencia.cols; i++){
                                                        for (int j = 0; j < ocorrencia.rows; j++){
                                                                if (((ocorrencia.at<double>(i, j)/contador) > 0.8)
                                                                        && mask.at<double>(i, j) == 0.0){
                                                                        //Analisa a linha (LINHA + 1) e
                                                                          a coluna (COLUNA + 1)
                                                                        result.at<double>(i, j) = Tint;
                                                                        mask.at<double>(i, j) = 1.0;
                                                                }
                                                        }
                                                }

                                                cv::minMaxLoc(mask, &minVal, 0, 0, 0);
                                                if (minVal == 1.0){
                                                        texto = ProcessamentoDeDados::matToString(result, 0);
                                                        parleyCommunication(texto, MAT_TEXT);
                                                        terminouAnaliseLimiar = true;
                                                }
                                                else{
                                                        Tint += 10;
                                                        texto = ProcessamentoDeDados::matToString(result, 0);
                                                        parleyCommunication(texto, MAT_TEXT);
                                                        contador = 0;
                                                        ocorrencia = cv::Mat::zeros(16, 16, CV_64F);
                                                }
                                        }

                                        if (Tint > 2300){
                                                terminouAnaliseLimiar = true;
                                        }

                                }

                                if (terminouAnaliseLimiar){
                                        ProcessamentoDeArquivos::imprimirMat(result, "resultado_analise.txt");
                                        MessageBox::Show(L"Análise foi concluída com sucesso. Verifique o arquivo
                                                        de texto.", L"Concluído!", MessageBoxButtons::OK,
                                                        MessageBoxIcon::Information);
                                }

                                Buffer[0] = CHIP_OFF;

                                MPUSBWrite(EP1OUTHandle, Buffer, 6, &ActualLength, INFINITE);
                                //Send the command now over USB
                                MPUSBRead(EP1INHandle, Buffer, 1, &ActualLength, INFINITE);
                                //Receive the answer from the device firmware through USB

                                parleyCommunication(" ", BACK_TO_NORMAL);

                                limiarControl = false;

                        } // fim do IF referente ao programa estar com o comando PIXEL TESTE ativado
                        #pragma endregion

                } // fim do IF referente ao programa estar com o cabo USB acionado

                Sleep(8);       //Um pequeno delay para a rotina não ser executada rápida demais,
                                  gerando alto consumo e baixo retorno.
        }
}


protected:
        virtual void WndProc( Message% m ) override{
                //Essa função é chamada quando uma mensagem do Windows é recebida pelo programa.

                // Garantir que somente as mensagens do tipo WM_DEVICECHANGE, que desejamos, sejam avaliadas.
                if(m.Msg == WM_DEVICECHANGE)
                {
                        if(((int)m.WParam == DBT_DEVICEARRIVAL) ||
                          ((int)m.WParam == DBT_DEVICEREMOVEPENDING) ||
                          ((int)m.WParam == DBT_DEVICEREMOVECOMPLETE) ||
                          ((int)m.WParam == DBT_CONFIGCHANGED) )
                        {
                                if(MPUSBGetDeviceCount(DeviceVID_PID))
                                //Checar se o dispositivo USB que usamos está conectado.
                                {
                                        if(AttachedState == FALSE) // Se ele estava desconectado, agora não está mais.
                                        {
                                                EP1OUTHandle = MPUSBOpen(0, DeviceVID_PID, "\\MCHP_EP1", MP_WRITE, 0);
                                                EP1INHandle = MPUSBOpen(0, DeviceVID_PID, "\\MCHP_EP1", MP_READ, 0);

                                                StatusBox_txtbx->Text = "Dispositivo Encontrado!";
                                                AttachedState = TRUE;
                                                checkREC->Enabled = true;
                                                checkFlipLR->Enabled = true;
                                                checkFlipUD->Enabled = true;
                                                checkLimit->Enabled = true;
                                                checkOffset->Enabled = true;
                                                checkCorrecao->Enabled = true;
                                                checkLimitDPCM->Enabled = true;
                                                radioTempoBox->Enabled = true;
                                                radioTempoLoad->Enabled = true;
                                                buttonPxlTeste->Enabled = true;
                                                buttonBlcPrincipal->Enabled = true;
                                                buttonPCM->Enabled = true;
```

```
                                        buttonDebug->Enabled = true;
                                        buttonConfig->Enabled = true;
                                        buttonLimiar->Enabled = true;

                                }
                        }
                        else    // Dispositivo não conectado, não encontrado ou que perdemos a
                                  comunicação com ele.
                        {
                                if(MPUSBClose(EP1OUTHandle))
                                        EP1OUTHandle = INVALID_HANDLE_VALUE;
                                if(MPUSBClose(EP1INHandle))
                                        EP1INHandle = INVALID_HANDLE_VALUE;

                                StatusBox_txtbx->Text = "Dispositivo Não Encontrado: Cheque a Comunicação";
                                AttachedState = FALSE;
                                checkREC->Enabled = false;
                                checkFlipLR->Enabled = false;
                                checkFlipUD->Enabled = false;
                                checkLimit->Enabled = false;
                                checkOffset->Enabled = false;
                                checkCorrecao->Enabled = false;
                                checkLimitDPCM->Enabled = false;
                                radioTempoBox->Enabled = false;
                                radioTempoLoad->Enabled = false;
                                buttonPararPxlTeste->Enabled = false;
                                buttonPararBlcPrincipal->Enabled = false;
                                buttonPararPCM->Enabled = false;
                                buttonPararDebug->Enabled = false;
                                buttonPxlTeste->Enabled = false;
                                buttonBlcPrincipal->Enabled = false;
                                buttonPCM->Enabled = false;
                                buttonDebug->Enabled = false;
                                buttonConfig->Enabled = false;
                                buttonLimiar->Enabled = false;

                        }
                }
        }

        Form::WndProc( m );
    }

};
}
```