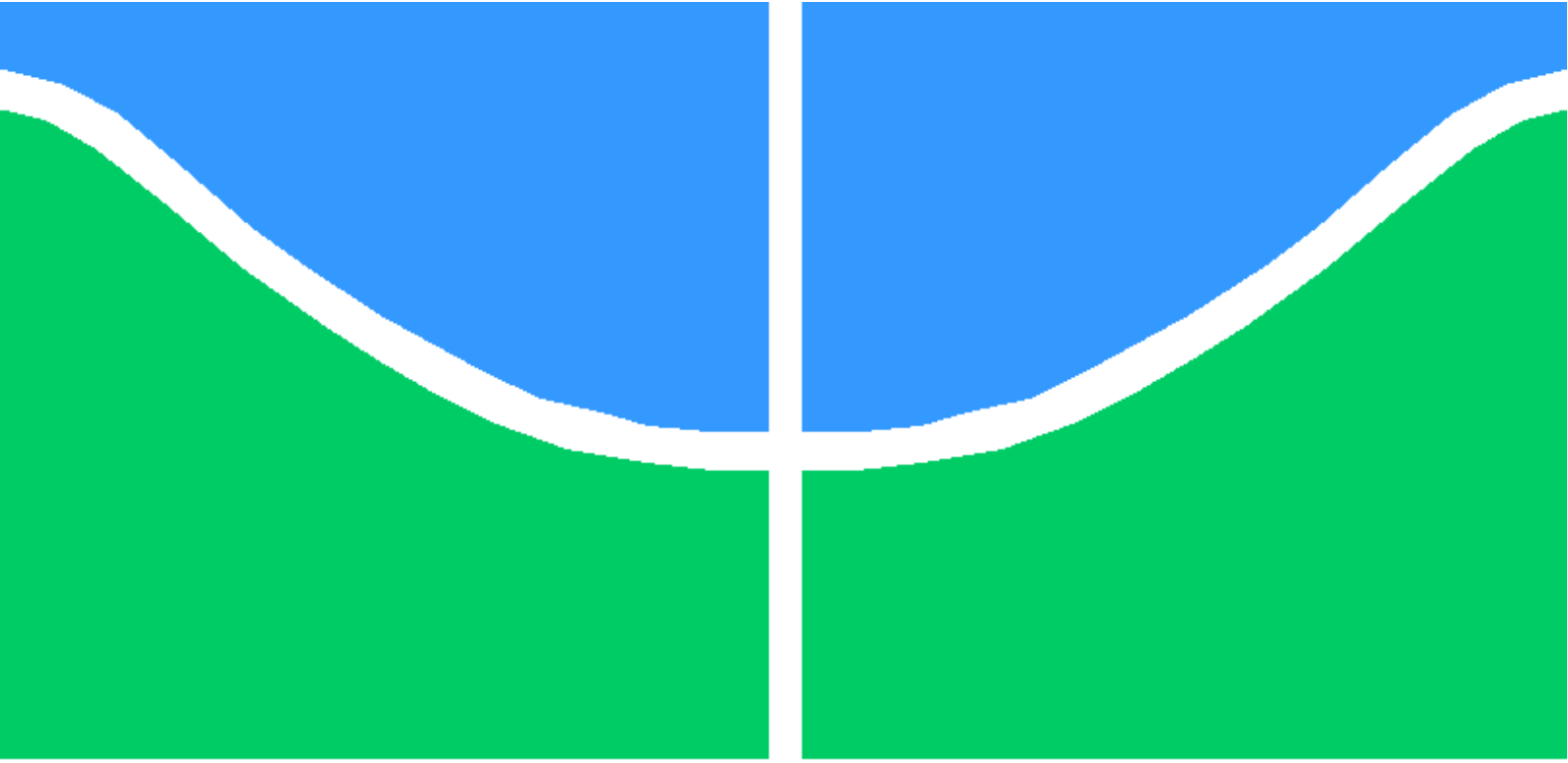


TRABALHO DE GRADUAÇÃO

MONITORAÇÃO NO PROCESSO DE SOLDAGEM GMAW POR MEIO DE VISÃO COMPUTACIONAL COM PROCESSAMENTO VIA FPGA

Rodrigo Ferreira Fernandes

Brasília, Dezembro de 2015

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

MONITORAÇÃO NO PROCESSO DE SOLDAGEM GMAW POR MEIO DE VISÃO COMPUTACIONAL COM PROCESSAMENTO VIA FPGA

Rodrigo Ferreira Fernandes

Relatório submetido ao Departamento de Engenharia Mecânica como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Guilherme Caribé, ENM/UnB _____
Orientador

Prof. Magno Batista Corrêa, ENM/UnB _____
Co-orientador

Dedicatória

Dedico este trabalho a...

Rodrigo Ferreira Fernandes

Agradecimentos

A inclusão desta seção de agradecimentos é opcional e fica à critério do(s) autor(es), que caso deseje(em) inclui-la deverá(ao) utilizar este espaço, seguindo está formatação.

Rodrigo Ferreira Fernandes

RESUMO

Continuar o desenvolvimento de um sensor de geometria da poça de fusão em um processo de soldagem GMAW por curto circuito por meio de algoritmos de processamento de imagem que possibilitem para cada quadro:

- Selecionar quadros propícios ao processamento;
- Determinar a largura e o eixo central do arame, bem como a distorção de perspectiva que ocorre no seu plano normal à direção do movimento de soldagem;
- Determinar a largura máxima da poça, assim como a distorção de perspectiva que ocorre no plano da poça de soldagem;

Os resultados do processamento acima serão organizados em vetores, os quais deverão ser filtrados por um Filtro de Kalman para geração de sinais propícios à utilização em uma malha de controle. Os algoritmos deverão ser validados comparando seus resultados com valores medidos à partir de testes controlados do processo.

ABSTRACT

Continue the development of a welding pool geometry sensor in a short circuit GMAW process by means of the development of an image processing algorithm that makes possible at each frame:

- Select frames appropriate to processing;
- Determine the wire's width and central axis, as well as perspective distortion in its plane normal to the welding's movement direction;
- Determine the pool's max width, as well as the distortion perspective in the pool's plane;

The results of the processing above will be organized in vectors, which should be filtered by a Kalman Filter for the generation of signals appropriate for utilization in a control system. The algorithms should be validated comparing its results with measured values from controlled process tests.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

A	Área	$[m^2]$

Símbolos Gregos

α	Difusividade térmica	$[m^2/s]$

Grupos Adimensionais

Nu	Número de Nusselt

Subscritos

amb	ambiente

Sobrescritos

$\cdot$	Variação temporal

Siglas

ABNT	Associação Brasileira de Normas Técnicas

Capítulo 1

Introdução

1.1 Contextualização

Qualidade de produtos deve ser um foco de atenção da indústria, principalmente em países emergentes, como o Brasil. Isso se deve ao aumento da exigência dos potenciais clientes por produtos e serviços de qualidade [7]. Essa exigência aumenta a competitividade do mercado [8] e obriga as indústrias a aperfeiçoar seus processos produtivos a fim de satisfazer os clientes e manter um bom nível de produtividade a baixo custo. O custo final dos produtos pode ser afetado pelo fator qualidade, aumentando em 80 a elasticidade do preço estimado [9]

Um dos fatores que influencia a qualidade de produtos é o processo de fabricação. A soldagem é um dos processos mais utilizados atualmente na indústria e o processo GMAW (*Gas Metal Arc Welding*) é um dos tipos de solda que apresentam mais vantagens: Alta taxa de deposição, facilidade para automação e variadas aplicações [10, 11].

1.2 Definição do problema

Esse processo requer mão de obra especializada cada vez mais escassa, grandes períodos de trabalho e alta regularidade. Além disso torna o ambiente altamente insalubre com radiação e gases tóxicos proveniente do arco, respingos de metal fundido e altas temperaturas. Este tipo de trabalho faz com que o soldador fique fatigado rapidamente, e isto é uma das principais causas da baixa produtividade e inconsistência em procedimentos com solda manual.

Além disso, regularidade é um aspecto fundamental em soldagem, pois melhora características estruturais e estéticas dos cordões de solda. Um soldador manual não consegue manter parâmetros como velocidade de avanço, altura do arco, ângulo de ataque e posicionamento da pistola por grandes períodos de tempo.

Esses fatores levaram a necessidade automatização para diminuir a exposição de trabalhadores e manter a regularidade do processo, aumentando a produtividade e a qualidade do produto final.

Infelizmente nem mesmo robôs e máquinas específicas para soldagem não são capazes produzir

cordões de solda perfeitos, inclusive podem não atender às especificações desejadas e cometer erros. Portanto, faz-se necessária a aplicação de controle do processo visando minimizar a quantidade de erros, evitar refugo e garantir as características desejadas aos produtos.

Existem vários métodos de controle para soldagem atualmente. A maioria deles consiste em monitorar parâmetros como tensão do arco e corrente, corrigir esses mesmos e outras variáveis como velocidade de alimentação do arame e velocidade de avanço. Esse tipo de controle é de malha aberta pois não monitora o cordão ou a poça de solda e apesar de melhorar a qualidade do processo não garante que os requisitos sejam cumpridos. Para garantir a qualidade da soldagem é necessário que seja feito um controle em malha fechada e isso só pode ser feito ao monitorar os estágios finais do processo.

1.3 Objetivos do projeto

A proposta deste trabalho é esse que o processamento seja feito através de uma FPGA em blocos sequenciais e paralelos. Cada bloco é responsável por uma etapa específica do processamento:

- Seleção de imagens propícias ao processamento;
- Pré-processamento das imagens;
- Obtenção de medidas do eletrodo;
- Obtenção de medidas da poça de soldagem

1.4 Apresentação do manuscrito

No capítulo 2 é feita uma revisão bibliográfica sobre o tema de estudo. Em seguida, o capítulo 3 descreve a metodologia empregada no desenvolvimento do projeto. Resultados experimentais são discutidos no capítulo 4, seguido das conclusões no capítulo 5. Os anexos contém material complementar.

Capítulo 2

Revisão Bibliográfica

2.1 Processo de Soldagem GMAW

O processo de soldagem GMAW do inglês *Gas Metal Arc Welding*, também conhecido como MIG/MAG (*Metal Inert Gas/Metal Active Gas*) é um dos processos de soldagem mais propícios para automação. Alguns aspectos básicos do processo GMAW concernentes ao problema aqui tratado serão detalhados e discutidos nos próximos subitens.

2.1.1 Fundamentos

O princípio da soldagem GMAW é a união de peças metálicas é devido ao aquecimento por um arco elétrico formado entre as mesmas e um eletrodo metálico nu e consumível, sendo todos os elementos envoltos por um gás ativo ou inerte.

O arame de soldar desempenha duas funções: por um lado é o eletrodo que conduz corrente, por outro, é também, em simultâneo, o material de adição a ser inserido na poça de soldagem. Este arame é introduzido mecanicamente através de um alimentador motorizado o que caracteriza o processo como semi-automático quando operado por humanos. [10]

A imagem 2.1 demonstra o processo de formação do cordão de solda e seus elementos principais.

Um gás de proteção que flui através do bocal da tocha protege o arco elétrico e o material em fusão, podendo o mesmo ser inerte (MIG) ou ativo (MAG). Os gases inertes, tais como o argônio e o hélio, não entram em reação com o material em fusão e apenas protegem os materiais fundidos de contaminantes. Por outro lado, os gases ativos, não só interferem no próprio arco elétrico, como também reagem com o material em fusão. Um exemplo de gás inativo é uma mistura de dióxido de carbono ou oxigênio com argônio. O componente ativo tem influência, por exemplo, sobre a penetração e/ou a temperatura do banho de fusão.

Além disto, o gás também tem influência nas perdas de elementos químicos, na temperatura da poça de fusão, na sensibilidade à fissuração e na porosidade, bem como na facilidade da execução da soldagem em diversas posições. Os gases nobres (processo MIG) são preferidos por razões

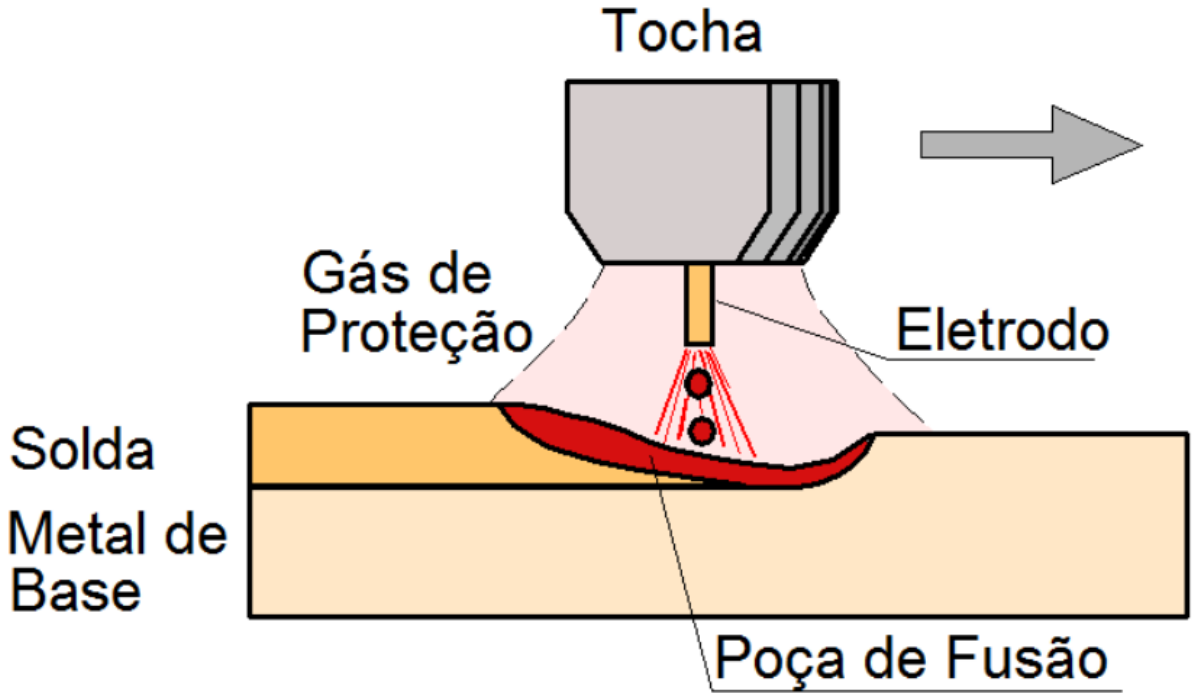


Figura 2.1: Elementos básicos da soldagem GMAW [1].

metalúrgicas, enquanto o CO2 puro, é preferido por razões econômicas. O processo MAG é utilizado somente na soldagem de materiais ferrosos, enquanto o processo MIG pode ser usado tanto na soldagem de materiais ferrosos quanto não ferrosos como Alumínio, Cobre, Magnésio, Níquel e suas ligas.

2.1.2 Equipamentos

O processo GMAW tem um conjunto específico de equipamentos. Por ser um processo semi-automático (ou completamente automático), possui mais componentes que métodos mais convencionais de soldagem como o eletrodo revestido e o oxiacetileno.

Os equipamentos necessários para esse tipo de solda são [10] [2]:

- Fonte de energia

  Este processo utiliza sempre corrente contínua. Dependendo do modo em que é feito o controle, pode ser necessário controlar a corrente ou a tensão fornecida pela fonte. É comum usar corrente constante ou tensão constante durante o processo;

- Alimentador de arame

  Existem diversos tipos de alimentadores com diferentes tipos de controle. O controle de alimentação do arame devem estar alinhados com o método de controle elétrico. Por exemplo: é possível usar alimentação constante de arame com tensão constante ou corrente constante e alimentação variável [10];

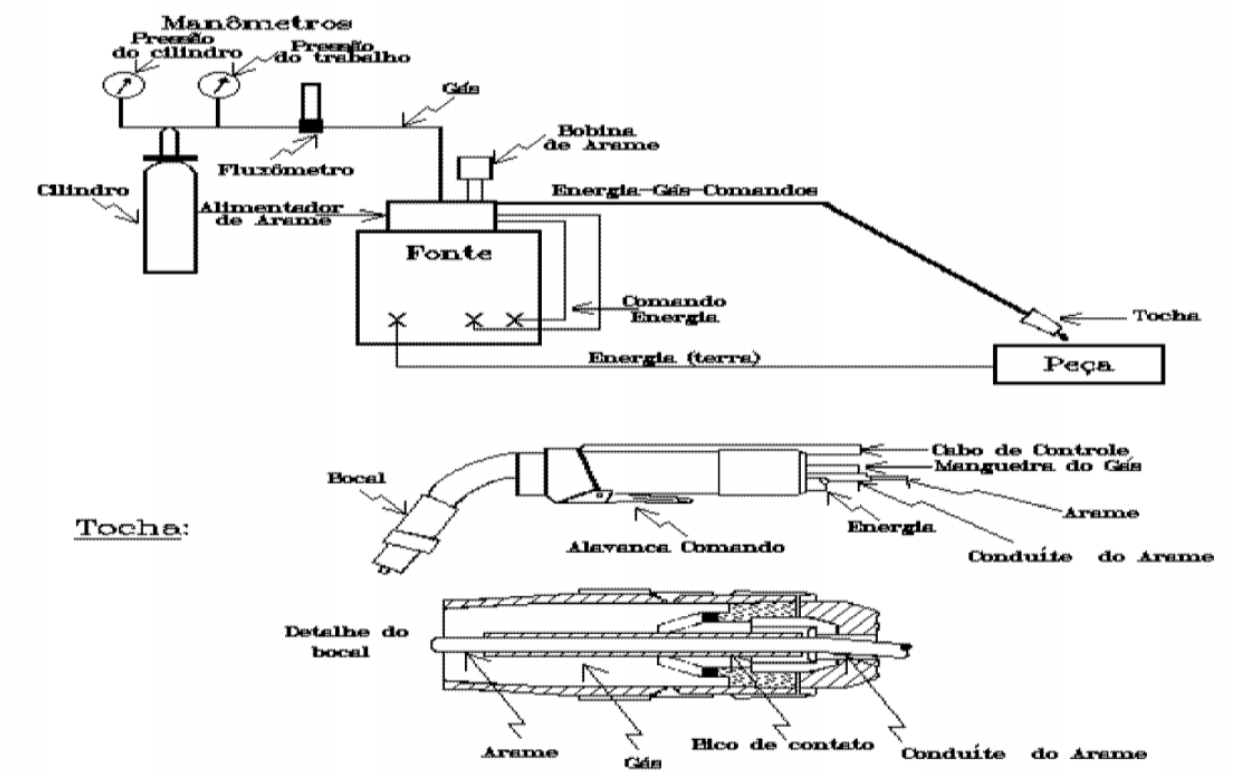


Figura 2.2: Esquemático dos equipamentos da soldagem GMAW [2].

- Fonte de gás protetor

  Normalmente um cilindro de gás (ou gases) e reguladores de pressão e/ou vazão

- Tocha de soldagem

  Existem diferentes tipos de tocha para proporcionar o desempenho máximo na soldagem para diferentes tipos de aplicações. Elas variam desde tochas para ciclos de trabalho pesados para atividades envolvendo altas correntes até tochas leves para baixas correntes e soldagem fora de posição. Em ambos os casos estão disponíveis tochas refrigeradas a água ou secas (refrigeradas pelo gás de proteção), e tochas com extremidades retas ou curvas.

  A tocha consiste em todo a aparato que é levado até o local de união das peças, seja pela mão de um operador ou uma máquina/robô. Ela contém os seguintes elementos básicos que podem ser vistos na figura 2.2 [2]:

  - Bocal: guia o gás para a junção, geralmente feito de cobre ou material cerâmico. Seu diâmetro deve ser compatível com a corrente de soldagem e o fluxo de gás;
  - Bico de contato: faz o contato elétrico com o eletrodo, feito de cobre;
  - Alavanca de comando ou Gatilho: Ativa a energização do circuito de soldagem, o alimentador de arame e o fluxo de gás.

2.1.3 Monitoramento dos parâmetros de soldagem

Diversas variáveis do processo GMAW podem ser ajustadas para uma boa soldagem do metal. Estas variáveis são a velocidade de alimentação do eletrodo, a distância do bocal à peça, o *stickout*, a inclinação de trabalho do eletrodo, e o fluxo de gás. Essas variáveis requerem um monitoramento constante por parte do operador, ou por um equipamento automático, que é o escopo deste trabalho. A velocidade de soldagem, a posição de soldagem e o diâmetro do eletrodo também influenciam consideravelmente na geometria do cordão de solda.

Quando a soldagem é feita por um soldador, a posição em relação à poça de soldagem e às peças sendo soldadas é constantemente verificada e corrigida. Dessa forma é possível manter uma qualidade boa da solda mesmo com as limitações mecânicas do ser humano. Este processo de ajuste que o soldador faz durante o processo de soldagem tem a característica de um sistema de controle em malha fechada.

Quando o processo é automatizado alguns parâmetros podem ser avaliados em tempo real, como corrente e tensão do arco para se inferir sobre variações na qualidade da solda. Outras abordagens como métodos baseados em inteligência artificial foram utilizadas para determinar uma metodologia para selecionar parâmetros de soldagem. Esses métodos apresentam a inconveniência de serem aplicáveis apenas em configurações específicas de soldagem e de funcionarem em malha aberta. Uma alternativa é utilizar um sistema de visão computacional para obter medidas geométricas durante a formação da solda. [12]

2.2 Processamento de imagens

As imagens capturadas em um processo de soldagem necessitam passar por um processamento para que os valores necessários sejam obtidos. Nesta seção será explicado como é feito esse processamento.

A área de processamento de imagens desperta interesse de estudiosos por ter diversas aplicações em duas principais categorias: (1) aprimoramento de informações pictóricas para interpretação humana; e (2) extração automática de dados relevantes à partir de uma cena [13]. A segunda categoria, que será utilizada neste trabalho, também pode ser designada como "análise de imagens", "visão por computador" ou "reconhecimento de padrões".

Diversas áreas se beneficiam de técnicas de processamento de imagens. Na medicina procedimentos computacionais melhoram o contraste, codificam níveis de intensidade para melhor interpretação e podem até mesmo calcular a quantidade de uma determinada célula em uma imagem. Geógrafos fazem uso de técnicas semelhantes e podem determinar áreas de vegetação, desmatamento, e poluição [14]. Existem muitos outros exemplos, que vão até o controle de qualidade e de processos.

2.2.1 Elementos de sistemas de processamento de imagens

Um sistema de processamento de imagens consiste alguns elementos básicos mostrados a seguir [14]:

1. Aquisição:

   São necessários: um dispositivo físico sensível a uma faixa de frequência no espectro eletromagnético (preferencialmente a luz visível) que produza um sinal elétrico proporcional ao nível de energia detectado; e um digitalizador que converte o sinal elétrico analógico em um sinal digital.

2. Armazenamento:

   Um desafio em processamento de imagens, o armazenamento de imagens digitais requer muita memória RAM ou *frame buffers (Armazenamento por curto tempo) e espaço em disco (Arquivamento).* ***Apesar de os computadores atuais não terem grandes problemas com armazenamento, alguns equipamentos podem não ter memória suficiente para a tarefa como, por exemplo, um microcontrolador.***

3. Processamento:

   É um procedimento algorítmico que normalmente é realizado via software. Em casos em que velocidade é um fator importante no processamento pode ser necessário o uso de hardware especializado. É a parte mais complexa do sistema, exige pesquisa e desenvolvimento.

4. Comunicação:

   A transmissão de imagens digitais necessita de uma alta largura de banda. Técnicas de compressão de imagens costumam ser utilizadas para reduzir este problema. Em aplicações em tempo real é necessária a sincronização dos dados.

5. Exibição:

   Por fim, o resultado do processamento é exposto ao ser humano através de um monitor. Pode ser desnecessária a exibição de todas as imagens de um processo que for automático.

O elemento mais importante neste trabalho, Processamento de Imagens,possui alguns passos fundamentais para se chegar ao objetivo proposto. São eles:

1. **Pré-processamento**: tem por objetivo melhorar a qualidade da imagem para que os estágios seguintes tenham mais garantia de sucesso. Geralmente envolve técnicas de realce de contrastes, remoção de ruído e isolamento de regiões.

2. **Segmentação**: é a divisão da imagem pré-processada em partes ou objetos constituintes. Por exemplo, caracteres em uma imagem de texto. É uma das tarefas mais difíceis de se implementar e geralmente é o que define se o processamento vai ter sucesso ou não.

3. **Representação e descrição**: nesse estágio, os dados obtidos pela segmentação em forma de pixels passam a ser representados de forma fronteiras e/ou regiões completas. A partir dessa representação, há a descrição de características quantitativas ou qualitativas, por exemplo, uma concavidade ou um buraco.

4. **Reconhecimento e interpretação**: finalmente, os objetos descritos anteriormente devem ser reconhecidos como algum padrão ou um valor para então se fazer a interpretação de um conjunto de objetos reconhecidos e atribuir um significado a esse conjunto. Por exemplo, uma imagem da palavra "sim" deve conter os objetos reconhecidos como "s","i", "m" e ser interpretada com a palavra "sim".

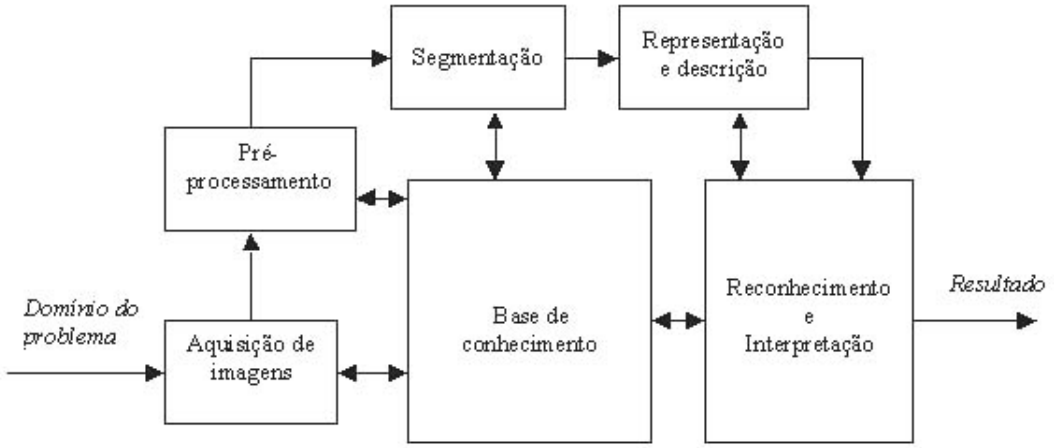


Figura 2.3: Passos Fundamentais em processamento de imagens digitais

Todos esses passos são possíveis quando se tem uma **Base de conhecimento** prévia que fica codificada no sistema de processamento de imagens. Com esses conhecimentos sobre o problema em questão e os métodos disponíveis pode-se criar uma solução viável. Por exemplo, quando é conhecida a região da imagem que tem informações de interesses é mais fácil segmentar essa imagem.

2.2.2 Realce de imagens

O objetivo das técnicas de realce é obter uma imagem mais apropriada para uma aplicação específica por meio de técnicas de processamento. As técnicas escolhidas, assim como seus parâmetros e a ordem em que são aplicadas dependem completamente da aplicação. Existem basicamente dois métodos de realce: **Métodos no domínio espacial** e **Métodos no domínio da frequência**.

Os métodos no domínio espacial operam diretamente sobre o agregado de pixels que compõem uma imagem. Funções de processamento de imagens no domínio espacial podem ser expressas como:

$$g(x,y) = T[f(x,y)] \tag{2.1}$$

em que $f(x,y)$ é a imagem de entrada, $g(x,y)$ é a imagem processada e T é um operador sobre f, definido sobre alguma vizinhança de (x,y). T pode operar sobre um conjunto de imagens de entrada, como será explicado mais adiante.

A estratégia é definir uma subimagem em torno de um pixel (x,y) (quadrado, retângulo ou mesmo aproximação de um círculo) e aplicar a operação sobre essa subimagem para cada pixel da imagem e obter g em cada posição correspondente. A vizinhança mais simples que pode ser definida tem tamanho 1×1, e T é uma *transformação de níveis de cinza* ou *processamento ponto-a-ponto* da forma:

$$s = T(r) \tag{2.2}$$

em que r e s representam os níveis de cinza de $f(x,y)$ e $g(x,y)$, respectivamente. Este tipo de função permite operações como mudança de contraste, binarização de imagem ou limiarização. Outra variedade de funções pode ser aplicada sobre a imagem ao se trabalhar vizinhanças maiores. Costuma-se utilizar uma janela ou *máscara* em forma de matriz 3×3 para se fazer operações. Essa técnica costuma ser chamada de *filtragem*.

Os métodos no domínio da frequência se baseiam no teorema da convolução com um operador linear invariante com a posição $h(x,y)$ na forma:

$$g(x,y) = h(x,y) * f(x,y) \tag{2.3}$$

ou, a partir do teorema da convolução, no domínio da frequência:

$$G(u,v) = H(u,v)F(u,v) \tag{2.4}$$

em que G, H e F são transformadas de Fourier de g, h e f, respectivamente. Similarmente aos sistemas lineares, a transformada $H(u,v)$ é chamada de *função de transferência óptica*. A equação 2.3 é um processo espacial análogo ao uso de máscaras e $h(x,y)$ costuma ser chamada de *máscara de convolução espacial*.

A seguir são detalhados os métodos de processamento utilizados neste trabalho.

2.2.2.1 Limiarização

Esta é um método espacial que consiste em separar uma imagem em diferentes níveis de cinza. O comum é ter apenas um limiar e dois níveis, o que constitui em uma binarização da imagem. A binarização pode ser feita de duas formas: transformar todos os pixels abaixo de um limiar em zero (preto) ou transformar todos os acima do limiar no nível máximo (branco).

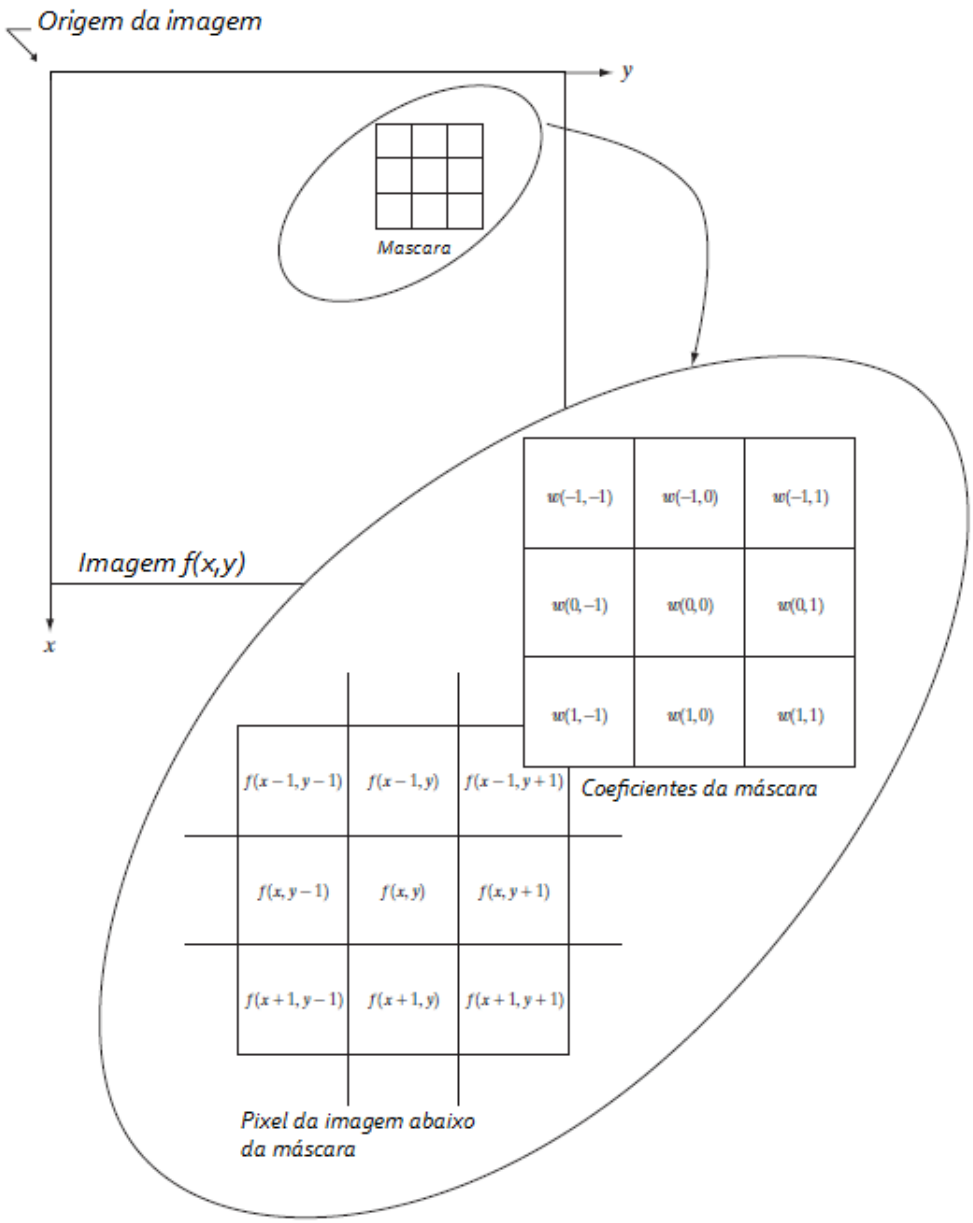


Figura 2.4: Máscara e vizinhança de pixel

O mais comum é a limiarização que transforma todos os pixels abaixo do limiar em preto e todos acima em branco, ou seja uma imagem binária.

A limiarização por ser feita com um limiar pré definido ou de forma automática. Um exemplo de limiarização automática é a de equilíbrio de histograma, onde um algoritmo é utilizado para separar equilibradamente os dois grupos de pixels mais claros e mais escuros.

2.2.3 Distorção de perspectiva

Imagens de objetos tridimensionais capturados por uma câmera podem ter suas dimensões distorcidas pois a imagem é uma projeção bidimensional de parte do objeto. Essa distorção pode tornar impossível a tarefa de medir corretamente dimensões do objeto em questão.

A projeção que ocorre no caso de captura de imagem por uma câmera é do tipo cônica, e não cilíndrica, isso implica que objetos de dimensões iguais no mundo real que estejam a diferentes distâncias da câmera terão suas projeções com diferentes tamanhos na imagem. Além disso câmeras podem ficar em ângulos oblíquos em relação aos planos de interesse dos objetos e essa projeção oblíqua pode resultar em diferentes medidas em diferentes eixos.

Para eliminar o problema de distorção existem dois métodos comumente utilizados: visão estereoscópica e transformação de projeção. O método de visão estereoscópica, apesar de permitir gerar um objeto tridimensional virtual, foi descartado neste trabalho pois seria tecnicamente inviável. Em ambientes com um razoável controle, como manter a câmera fixa em relação ao objeto, a transformação de projeção é suficiente para se obter os dados necessários.

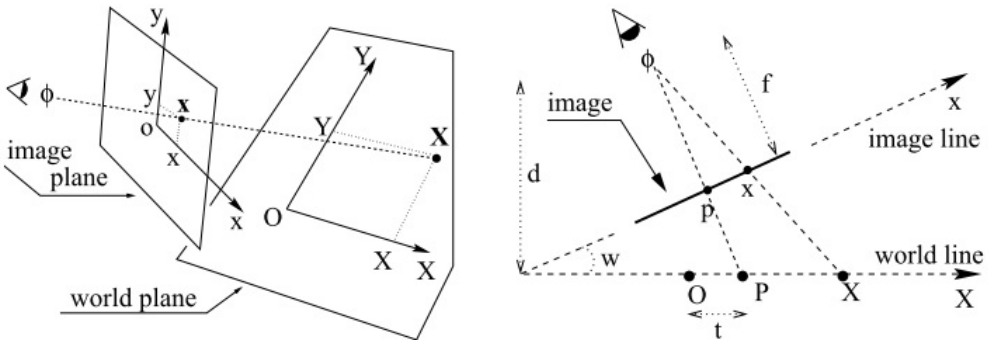


Figura 2.5: [3] (a) Modelo de câmera plana. (b) Modelo de câmera unidimensional.

2.2.3.1 Coordenadas Homogêneas

Matematicamente o modelo de câmera e de projeção pode ser definido por uma matriz de transformação H. Pontos do plano real, Π são representados por vetores em letra maiúscula, $\mathbf{X}$, e as imagens correspondentes, no plano π são representadas por vetores de letra minúscula, $\mathbf{x}$. A projeção de perspectiva dos pontos correspondentes é dada por [15] :

$$\mathbf{X} = T\mathbf{x} \tag{2.5}$$

Onde T é uma matriz 3×3, "=" representa igualdade em escala. Os vetores de pontos são dados por: $\mathbf{X} = (X_1, X_2, X_3)^T$ e $\mathbf{x} = (x_1, x_2, x_3)^T$

O ponto da imagem, uma projeção, e representado por três coordenadas cartesianas, $\mathbf{x} =$

$(x_1, x_2, x_3)^T$. Essas coordenadas são chamadas de homogêneas. Apenas a direção do vetor é importante visto que, independente da distância da câmera, qualquer ponto real em determinada direção aparecerá em um único ponto na projeção. Portanto todos os pontos da forma $\lambda\mathbf{x} = (\lambda x_1, \lambda x_2, \lambda x_3)$ são equivalentes.

Para representar os pontos em um plano cartesiano convencional da forma (x, y), deve-se construir um plano especial π_e), perpendicular ao eixo x_3 a uma distância unitária na direção de x_3. A intersecção do vetor $\mathbf{x}$ com o plano π_e é o ponto $x_e = (x, y, 1)$.

É de interesse que a posição desse plano não afete a posição das coordenadas cartesianas (x, y), portanto define-se essas coordenadas da seguinte forma:

$$x_e = (\frac{x_1}{x_3}, \frac{x_2}{x_3}, 1)^T = (x, y, 1)^T$$

O modelo da câmera é completamente especificado pela matriz T, que pode ser calculada com a posição relativa dos dois planos e o ponto focal da câmera. Porém, essa matriz também pode ser calculada diretamente por correspondência entre pontos na imagem e pontos no mundo real. Esse cálculo é descrito na seção 2.2.3.2.

2.2.3.2 Cálculo da matriz de transformação

A equação 2.2.3.1 pode ser melhor visualizada a seguir:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \tag{2.6}$$

A escala λ da matriz não afeta a equação, portanto apenas os oito graus de liberdade correspondentes à razão dos elementos da matriz são significantes.

À partir da equação 2.2.3.1, cada correspondência entre pontos reais e pontos da imagem gera duas equações de coordenada cartesianas $\mathbf{H}$. Para n correspondências obtém-se um sistema com $2n$ equações com 8 variáveis. Se $n = 4$, obtém-se a solução exata [15]. Se $n > 4$, a matriz é super-determinada e estima-se $\mathbf{H}$ por minimização [3].

A representação em coordenadas cartesianas demonstra a natureza não linear da transformação:

$$x = \frac{x_1}{x_3} = \frac{t_{11}X + t_{12}Y + t_{13}}{t_{31}X + t_{32}Y + t_{33}} \tag{2.7}$$

$$y = \frac{x_2}{x_3} = \frac{t_{21}X + t_{22}Y + t_{23}}{t_{31}X + t_{32}Y + t_{33}} \tag{2.8}$$

Para definir os elementos da matriz de transformação deve-se ter quatro correspondências de pontos entre plano real e projeção. Com a escala de $\mathbf{T}$ arbitrária, e $t_{33} = 1$, temos os pontos representados por $(\lambda_i x_i, \lambda_i y_i, lambda_i)^T = T(X_i, Y_i, 1)^T$. O sistema de equações lineares resultantes é:

$$\begin{bmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -y_1X_1 & -y_1Y_1 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -y_2X_2 & -y_2Y_2 \\ X_3 & Y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -x_3Y_3 \\ 0 & 0 & 0 & X_3 & Y_3 & 1 & -y_3X_3 & -y_3Y_3 \\ X_4 & Y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -x_4Y_4 \\ 0 & 0 & 0 & X_4 & Y_4 & 1 & -y_4X_4 & -y_4Y_4 \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{12} \\ t_{13} \\ t_{21} \\ t_{22} \\ t_{23} \\ t_{31} \\ t_{32} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix} \tag{2.9}$$

Esse sistema linear garante a solução para a matriz **T**, desde que nenhum grupo de três pontos sejam colineares.

Finalmente, para determinar a matriz de transformação deve-se resolver a equação 2.9 para t_{11} a t_{32}. O sistema é do formato $A \cdot T = B$, e pode ser calculado com a regra da matriz inversa. A solução segue-se na equação 2.11 abaixo:

$$T = A^{-1} \cdot B \tag{2.10}$$

$$\begin{bmatrix} t_{11} \\ t_{12} \\ t_{13} \\ t_{21} \\ t_{22} \\ t_{23} \\ t_{31} \\ t_{32} \end{bmatrix} = \begin{bmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -y_1X_1 & -y_1Y_1 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -y_2X_2 & -y_2Y_2 \\ X_3 & Y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -x_3Y_3 \\ 0 & 0 & 0 & X_3 & Y_3 & 1 & -y_3X_3 & -y_3Y_3 \\ X_4 & Y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -x_4Y_4 \\ 0 & 0 & 0 & X_4 & Y_4 & 1 & -y_4X_4 & -y_4Y_4 \end{bmatrix}^{-1} \cdot \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix} \tag{2.11}$$

2.2.3.3 Minimização de erros da transformação

Para que se minimize as margens de erro da matriz de transformação **T**, são necessários mais de quatro correspondências de pontos. De modo geral, como demonstrado em [3], quanto mais pontos utilizados na determinação da matriz de transformação, menor é a incerteza. Para n pontos, sistema de equações é semelhante à 2.9:

$$\begin{bmatrix} X_1 & Y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -y_1X_1 & -y_1Y_1 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -y_2X_2 & -y_2Y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & 1 & 0 & 0 & 0 & -x_nX_n & -x_nY_n \\ 0 & 0 & 0 & X_n & Y_n & 1 & -y_nX_n & -y_nY_n \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{12} \\ t_{13} \\ t_{21} \\ t_{22} \\ t_{23} \\ t_{31} \\ t_{32} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_n \\ y_n \end{bmatrix} \tag{2.12}$$

2.2.4 Detecção de cantos

Muitas tarefas em processamento de imagens requerem a identificação de elementos em imagens, como figuras geométricas, linhas e pontos. Existem diversos métodos e algorítimos para detecção de cantos, mas apenas o método de Harris [16] será descrito nessa seção pois apresenta alta taxa de acerto e fácil implementação de paralelismo, que pode acelerar o processo em um FPGA.

2.2.4.1 Detector de Moravec

O algorítimo parte do princípio de diferenciação de intensidade em porções de imagens usado no detector de imagens de Moravec [17]. O detector de Moravec funciona considerando uma janela na imagem, e determina a mudança de intensidade da imagem que resulta do deslocamento dessa janela em várias direções. São feitas as seguintes considerações:

1. Se a imagem na janela é plana, (aproximadamente constante em intensidade), então todas os deslocamentos resultam em uma mudança pequena;
2. Se a janela de move ao longo de uma borda, então a mudança de intensidade é pequena, mas se o deslocamento for perpendicular à borda, a mudança é grande;
3. Se a janela contém um canto ou ponto isolado, então todas os deslocamentos resultam em uma grande mudança de intensidade. Portanto, um canto pode ser detectado quando a menor mudança produzida por qualquer deslocamento for grande.

Foi feita a descrição matemática das considerações acima. Denotando as intensidades como I, a mudança E produzida por um deslocamento (x, y) é dado por:

$$E_{x,y} = \sum_{u}^{v} w_{u,v} |I_{x+u,y+v} - I_{u,v}|^2 \tag{2.13}$$

onde w especifica a janela da imagem: é uma unidade dentro de uma região retangular específica, e zero fora dela. Os deslocamentos (x, y), são do tipo $(1, 0), (1, 1), (0, 1), (-1, 1)$. Portando o detector de Moravec é basicamente: procurar por um máximo local em $min(E)$ acima de uma limite estabelecido.

2.2.4.2 Detector de auto-correlação

A performance do detector de Moravec contém uma série de erros conforme demonstrado em [16], que então os listou e fez as devidas correções:

1. **A resposta é anisotrópica porque apenas um grupo de deslocamentos discretos é considerado** - todos os possíveis deslocamentos podem ser cobertos fazendo uma expansão analítica em torno do centro de deslocamento.

$$E_{x,y} = \sum_{u}^{v} w_{u,v}[I_{x+u,y+v} - I_{u,v}]^2 = \sum_{u}^{v} w_{u,v}[xX + yY + O(x^2, y^2)]^2 \tag{2.14}$$

   onde os primeiros gradientes são aproximados por

$$X = I \otimes (-1, 0, 1) \approx \delta I/\delta x$$
$$Y = I \otimes (-1, 0, 1)^T \approx \delta I/\delta y$$

   Então para pequenas mudanças, E pode ser escrito

$$E(x, y) = Ax^2 + 2Cxy + By^2$$

   onde

$$A = X^2 \otimes w$$
$$B = Y^2 \otimes w$$
$$C = (XY) \otimes w$$

2. **A resposta é ruidosa porque a janela é binária e retangular** - usar uma janela circular suave, por exemplo uma Gaussiana:

$$w_{u,v} = exp - (u^2 + v^2)/2\sigma^2$$

3. **O operador responde muito cedo a bordas porque o apenas o mínimo de E é considerado** - reformular a medida de canto para usar a variação de E com a direção do deslocamento. A mudança, E, para um pequeno deslocamento (x, y) pode ser escrita como:

$$E(x, y) = (x, y)M(x, y)^T$$

   onde a matriz M, 2×2 simétrica é:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{2.15}$$

Os autovalores α e β de M correspondem à principais curvaturas da função de autocorrelação local. São três os casos em relação aos autovalores:

1. Se ambos são pequenos, a janela corresponde a uma imagem plana;
2. Se um é pequeno e o outro é grande, a janela corresponde a uma borda;
3. Se ambos são grandes, a janela corresponde a um canto.

Com essas três considerações, se faz necessário uma medida também uma medida de qualidade de cantos. Calcula-se então uma medida R, função de $/alpha$ e $/beta$. Para evitar a decomposição dos autovalores de M, usa-se o traço e a determinante da matriz, sendo que:

$$Tr(M) = \alpha + \beta = A + B$$
$$Det(M) = \alpha\beta = AB - C^2$$

Finalmente, Harris usou a seguinte formulação:

$$R = Det(M) - kTr(M)^2 \tag{2.16}$$

Com essa formulação, pode-se definir o tipo da região localizada em (u, v):

- Canto, se R é positivo;
- Borda, se R é negativo;
- Região plana, se R é pequeno.

Como R pode assumir uma grande gama de valores para diferente regiões da imagens utiliza-se limiares para definição dessas regiões. Isto gera a identificação de cinco regiões diferentes: fundo, duas classes de canto e duas de bordas. Quando o objetivo é apenas encontrar cantos, somente dois limiares são necessários.

2.2.5 Perfis Verticais e Horizontais

Uma das ferramentas para se encontrar elementos em imagens é perfil vertical e horizontal da imagem, ou perfis X e Y [18]. Esses perfis consistem basicamente na soma dos valores de pixels de linhas (perfil horizontal) ou colunas (perfil vertical) de uma imagem.

Esses perfis normalmente são armazenados em vetores para comparação, visualização ou extração de informações. Por exemplo: o valor máximo de um perfil horizontal pode identificar a linha onde se encontra a maior largura de um objeto na imagem.

Com os vetores de perfis também pode-se utilizar suas derivadas. Como esses vetores são discretos, as derivadas são apenas o resultado da subtração de um elemento pelo elemento consecutivo.

Desta forma, podem ser criados pares de vetores com estes valores. As quantidades de elementos desses vetores correspondem a quantidade de linhas e colunas da imagem.

2.3 Sistemas reconfiguráveis

Os custos para se testar circuitos na indústria de eletrônicos são muito elevados, pois é necessário fazer todo o *setup* do processo de produção para um novo circuito. Por esse motivo foi desenvolvida

a FPGA, ou *Field Programable Gate Array*, que é um circuito reconfigurável que pode rapidamente assumir as características de um circuito digital, com todas as suas ligações e portas lógicas.

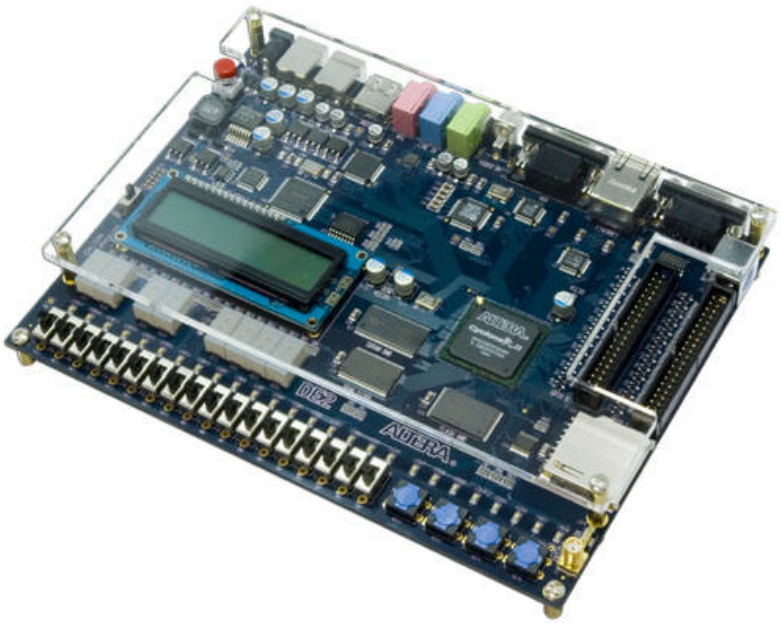


Figura 2.6: Placa Altera DE2, com FPGA Cyclone II [4]

2.3.1 História

Os primeiro dispositivo programáveis foi o *Programmable Read Only Memory*(PROM) , uma memória não volátil que podia ser do tipo fabricado em massa ou programável em campo pelo usuário. Por sua vez o tipo o PROM reconfigurável em campo tinha duas subdivisões: EPROM (*Erasable* PROM) e EEPROM (*Electronic Erasable* PROM), que pode ser reprogramado múltiplas vezes. Em seguida foram criados os *Programmable Logic Devices* (PLDs), em que o tipo mais comum implementa um conjunto fixo de portas OR precedido por uma matriz de portas AND programáveis. [19]

Em 1985, a empresa Altera criou o primeiro dispositivo reprogramável da indústria, o chip EP300, Os usuários poderiam apagar as células EPROM do EP300 ao emitir luz ultravioleta através de uma janela de quartzo acima do circuito. A opção meramente conveniente à época se mostraria um grande elemento na indústria. [5]

As FPGAs foram desenvolvidas como uma evolução dos PLDs, mais especificamente dos PLAs *Programmable Logic Arrays* (PLAs), como descrito em [20]. Em 1985 a empresa Xilinx criou a primeira FPGA comercial [21], com portas e interconexões reprogramáveis. Contudo, foi durante a década de 1990 que esses dispositivos ficaram famosos e assumiram diversas aplicações.

Atualmente os sistemas reconfiguráveis continuam em alta, principalmente agora que as tecnologias de fabricação estão próximas do limite de miniatuzação. A evolução do poder de proces-

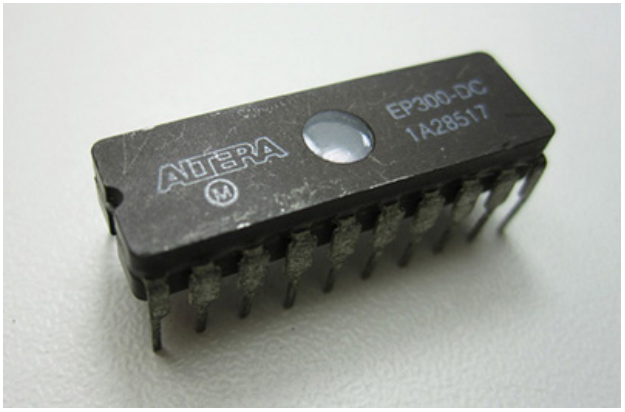


Figura 2.7: EP300 da Altera [5]

samento de arquiteturas convencionais (Arquitetura de Von Neumann), baseadas em um *hardware* genérico que recebe fluxos de instruções, demonstra desaceleramento [22]. Com o esperado limite da Lei de Moore [23] , muitos projetistas recorrem ao uso de *hardware* específico para solucionar diferentes problemas.

As FPGAs atuais são produzidas com uma série de facilidades que incluem: memória, entradas e saídas de alta velocidade e blocos lógicos. Elas podem tanto ser utilizadas para desenvolvimento e teste de circuits digitais (motivo pelo qual foram desenvolvidas) como podem ser implementadas no produto final. As aplicações vão desde processamento de imagens a mineração de *bitcoins*

2.3.2 Aspectos Gerais

De acordo com a patente de *Re-programmable PLA* [20] os *Programmable Gate Array* (PLAs) reconfiguráveis são descritos abaixo:

> Em geral, um PLA é um circuito lógico que recebe uma pluralidade de sinais digitais de entrada e gera uma pluralidade de sinais digitais de saída onde cada um dos sinais de saída é uma combinação de soma-de-produto programável dos sinais de entrada. Em PLAs convencionais, um circuito é fornecido para gerar uma pluralidade de termos que são o E lógico de sinais de entrada selecionados; e um outro circuito é fornecido para gerar os sinais de saída ao operar seletivamente esses termos com OU ou E lógicos. Um PLA típico pode ter um total de x sinais de entrada, gerar um total de y termos "E" à partir dos sinais de entrada, e gerar um total de 2 sinais de saída ao operar seletivamente os termos y com um OU lógico.

Outras tecnologias de semicondutores reprogramáveis também foram desenvolvidas, como os *Application Specific Integrated Circuits* (ASICs), onde o dispositivo é desenvolvido apenas para um tipo de aplicação. Ainda há uma subdivisão dentro das FPGAs:

- *One-Time Programmable* (OTP), que funciona como um ASIC depois de programado;
- SRAM-*based*, que pode ser reconfigurado múltiplas vezes e é o tipo mais utilizado.

Os elementos básicos de um FPGA são [6]:

- Combinação lógica
- Interconexão
- Blocos de entrada e saída

A arquitetura geral com os elementos básicos pode ser visto na figura 2.8. A lógica combinacional é dividida em blocos pequenos, chamados de CLBs (*Combinational Logic Blocks*) ou LE (*Logic Elements*). O CLB ou LE realiza operações simples com uso de portas lógicas e flip-flops.

As interconexões são feitas entre elementos lógicos e/ou entre portas de entrada e saída por meio de conexões programáveis. A interconexão pode ser logicamente organizada em canais ou outras unidades. Sinais importantes como um clock podem ser ligados a vários (até mesmo a todos) elementos.

Os pinos de entrada e saída são representados por blocos de entrada e saída, IOB (*I/O Blocks*). Esses blocos podem ser configurados para receber ou fornecer sinais ao meio exterior.

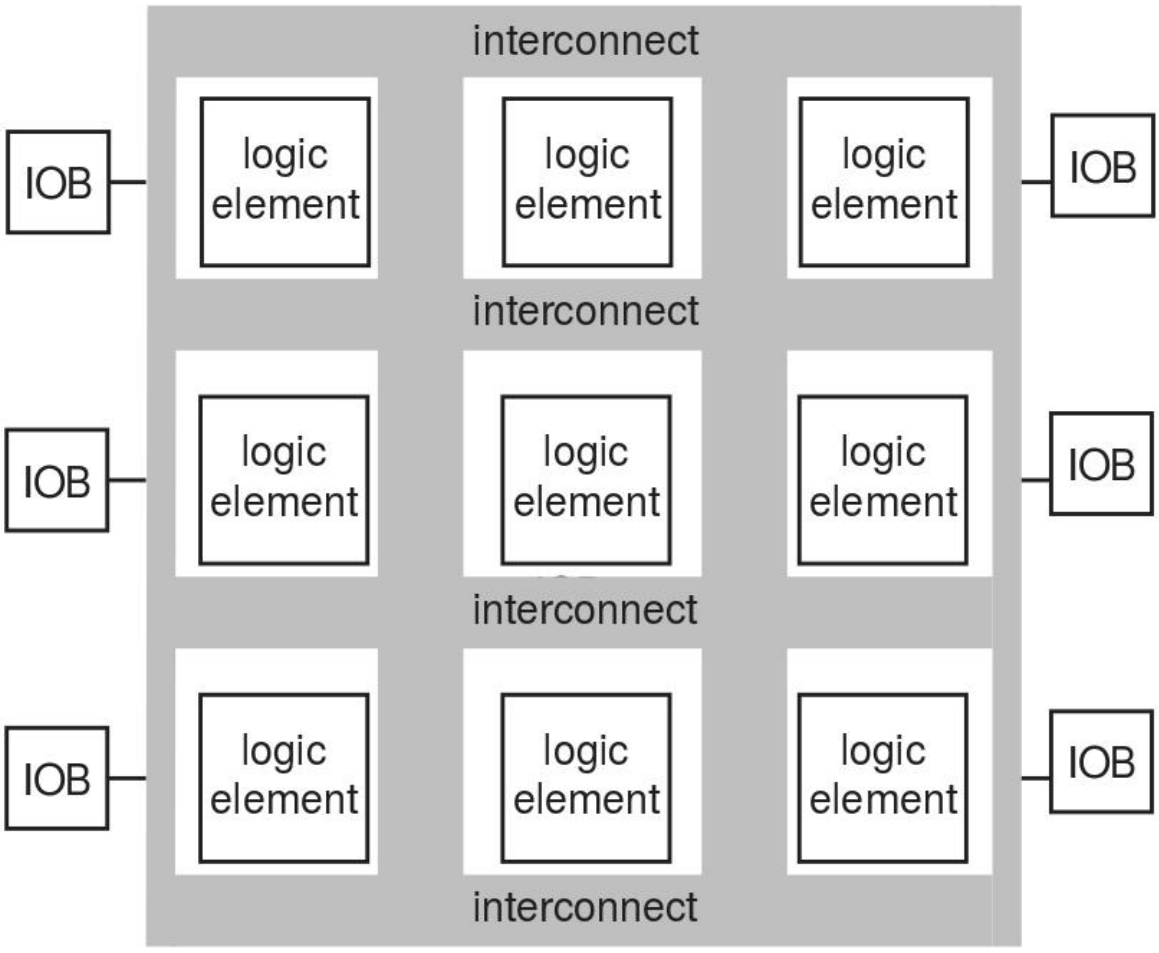


Figura 2.8: Arquitetura geral de uma FPGA [6]

LER DESSES TEXTOS E EXPLICAR MAIS

https://en.wikipedia.org/wiki/Field-programmable_gate_array#cite_note-book-16
file:///home/rodrigo/Downloads/US4508977.pdf
https://www.google.com/patents/US8112466?dq=field+programmable+gate+array+patent&hl=pt-BR&sa=X
https://www.google.com/patents/US4870302?dq=freeman+1989+configurable&hl=pt-BR&sa=X&ved=0CBsQ6
https://www.google.com.br/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8&client=ubuntu#q=

2.4 Linguagens de Descrição de Hardware e VHDL

Uma linguagem de descrição de hardware, ou HDL (*Hardware Description Language*) é semelhante a uma linguagem de programação em termos de sintaxe e estrutura, porém como o próprio nome diz ela serve mais para descrever do que para programar. Esse tipo de linguagem tem por objetivo facilitar o projeto de circuitos eletrônicos, com uma descrição formal e precisa. Uma HDL padronizada permite a intercambialidade de informações entre diferentes fabricantes e projetistas além da automatização de simulação e síntese de circuitos. Existem variadas HDLs, dentre as quais foi escolhida a VHDL para este trabalho por motivos a serem expostos adiante.

2.4.1 História do VHDL

O surgimento do VHDL se deve à necessidade do *Defense Advanced Research Projects Agency* (DARPA) em desenvolver uma ferramenta de projeto e documentação para o projeto VHSIC (*Very High Speed Integrated Circuit*). O Departamento de Defesa definiu em 1983 os requisitos para uma linguagem de descrição padrão para circuitos e contratou as empresas IBM, Texas e Intermetrics para a tarefa. [24]

A linguagem foi padronizada pelo IEEE (*Institute of Electrical and Electronic Engineers*) com base na versão 7,2 e com o auxílio da empresa CLSI, contratada pela Força Aérea dos Estados Unidos. [24] Em 1987, surgiu o padrão IEEE 1076-1987, denominado VHSIC *Hardware Description Language* ou VHDL. A padronização da linguagem contou com a participação de profissionais de diversas áreas: design de sistemas de computadores, aeroespacial, comunicações, desenvolvedores de CAD, desenvolvimento de circuitos integrados etc. [25]

Em 1993, o padrão 1076 recebeu uma revisão para facilitar ainda mais o trabalho de projetistas. A principal mudança era o melhor gerenciamento hierárquico de arquivos [26]. No mesmo ano surgiu o padrão 1164, que definia o pacote "std_logic_1164" que trouxe mais versatilidade ao modelamento por representar mais condições reais como alta impedância e nós não inicializados. O padrão 1164 é muito utilizado até os dias de hoje, principalmente depois de suas revisão em 2008 .

Em 1997, o surgiu mais um padrão, o 1076.3 [27], que propôs novos tipos de dados como inteiros e reais, e novas funções como soma e multiplicação. Esses tipos são abstratos em código e um mero

conjunto de bits no circuito real, porém a linguagem permite as operações apenas com variáveis de mesmo tipo. Essa padronização caracteriza o VHDL como uma linguagem fortemente tipada.

2.4.2 Aspectos gerais da VHDL

A linguagem VHDL, pode ser entendida tanto por máquinas quanto por humanos como na própria definição do IEEE [27]. Tem por objetivo organizar e documentar os projetos para diminuir o tempo de trabalho dos projetistas além de promover a rápida sintetização e de simulação de circuitos.

Uma das suas principais características é a alta hierarquização. É possível criar projetos com múltiplos níveis de hierarquia utilizando entidades (descrições de circuitos) dentro de entidades. Essa estrutura facilita o desenvolvimento partindo de níveis mais altos de abstração para níveis menores, com o método chamado *top down* (muito comum nas linguagens de programação mais recentes). Também é usada a descrição do comportamento esperado do circuito.

Em um circuito sintetizado, a maioria dos comandos é executado concorrentemente, independente do nível hierárquico dos componentes e da ordem em que são declarados.

Apesar disso é possível criar operações sequenciais tais como linguagens de programação em regiões específicas do código, nos subprogramas e processos. Essas estruturas são muito úteis para operações específicas como operações aritméticas mais complexas.

Algumas das características principais da linguagem são:

- *Case-insensitive*, ou seja não identifica diferença entre caixa alta ou baixa em seu código;
- Fortemente tipada;
- Escalonável;
- Hierárquica

2.4.3 Entidades

É uma abstração primária do VHDL. As entidades podem ser instanciadas dentro de outras entidades, formando assim a hierarquia do projeto. Cada instância é vista pela entidade superior como uma caixa preta da qual se conhece apenas as entradas e saídas. A ferramenta de sintetização cria as instâncias à partir do nível mais alto da hierarquia, a Entidade de Projeto (ou *Top Level Entity*).

Cada entidade ser declarada com a palavra reservada "ENTITY" seguida do nome que a identifica. Em seguida podem ser opcionalmente declaradas as cláusulas PORT" ou "GENERIC". A cláusula "PORT" define os tipos de portas de entrada e saída da entidade. A cláusula "GENERIC" permite passar informações estáticas para uma entidade. Essas cláusulas são opcionais e podem ser omitidas, por exemplo em um *testbench* usado para simulações.

O comportamento de uma entidade é definido pela arquitetura e precisa ser declarado com a palavra "ARCHITECTURE" seguida do nome dado a arquitetura e a identificação da entidade

a qual ela está definindo. Em seguida, podem ser declarados sinais, constantes, subprogramas ou identificação de outras entidades que serão instanciadas dentro desta. Finalmente, a lógica é descrita entre as palavras "BEGIN" e "END" da arquitetura, com todos os comandos, as operações e instanciações.

O exemplo abaixo mostra uma entidade que utiliza a instância de outras em seu código:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all;

ENTITY main is
    port ( CLOCK : in std_logic;
    start_stop  : in std_logic ;
            up_down       : in std_logic ;
    D1A, D1B, D1C, D1D, D1E, D1F, D1G, DP1 : out std_logic;
  );



END main ;

ARCHITECTURE ESTRUTURA of main is

SIGNAL BCD : std_logic_vector(3 downto 0);
SIGNAL binario4bits : integer(11 downto 0);
COMPONENT CONTADOR
PORT (
  CLK   : in std_logic;
  start_stop    : in std_logic ;
    up_down     : in std_logic ;
  binario4bits  : out integer(3 downto 0)
  );
END component;

COMPONENT BIN_7SEG
PORT (
  BIN : in std_logic_vector(3 downto 0);
  SEG_A, SEG_B, SEG_C, SEG_D, SEG_E, SEG_F, SEG_G, DP : out std_logic
  );
END component;

BEGIN
conta1 : CONTADOR port map (CLOCK, start_stop, up_down, binario12bits);
```

```
conv1 : BIN_7SEG port map (binario4bits, D1A, D1B, D1C, D1D, D1E, D1F, D1G, DP1);
END ESTRUTURA;
```

2.4.3.1 Entidade de Projeto

A entidade de projeto ou *top level entity* é o nível mais alto na hierarquia de um projeto VHDL. Por convenção, o documento dessa entidade geralmente leva o nome do projeto e é por este documento que a ferramenta de sintetização começa a leitura.

Assim como outras entidades, a entidade de projeto tem entradas e saídas de diferentes tipos, porém estas não são utilizadas por outras entidades pois esta não está contida em nenhuma outra. As portas declaradas para essa entidade são diretamente relacionadas com o ambiente externo ao dispositivo. Por exemplo: são as portas desta entidade que o sintetizador usa para mapear os pinos de uma placa FPGA. Todas as outras portas de outras entidades são sinais internos à placa neste caso.

A arquitetura da entidade de projeto define o comportamento geral de todo o circuito, com as principais entidades que serão instanciadas e o relacionamento entre as mesmas. Em grandes projetos é comum não haver operações nesta entidade, somente ligações entre outras entidades e entre estas com o meio exterior.

2.4.4 Bibliotecas e Pacotes

O VHDL faz uso de bibliotecas para reaproveitar códigos e padronizar os projetos. Existem bibliotecas padrão como a IEEE, que possui diversos pacotes, como o 1076.3 e o 1164, que geralmente podem ser usados em um mesmo projeto. O projetista também pode definir bibliotecas próprias para fins de organização ou uso em outros projetos.

A maioria das bibliotecas precisam ser declaradas no início do código com a palavra chave "LIBRARY" para se ter acesso a suas definições e pacotes. Há duas bibliotecas que estão sempre disponíveis automaticamente sem necessidade de declaração nos documentos: a biblioteca padrão "std" e a biblioteca "work".

Os pacotes são definições que podem ser utilizadas em diferentes modelos definidos em diferentes arquivos e estão sempre contidos em alguma biblioteca. É um recurso que permite reutilizar código para poupar tempo e principalmente para manter a padronização de projetos. Em uma linguagem fortemente tipada, como é o caso do VHDL, é de suma importância a utilização de pacotes em detrimento de definições locais.

Existem diversos pacotes padrões, muitos dos quais distribuídos pelo IEEE, mas também há os pacotes definidos pelo próprio projetista como citado anteriormente.

Cada pacote utilizado precisa ser declarado no início do documento para se ter acesso aos elementos definidos no mesmo ao longo do código. É necessário declarar de antemão a biblioteca em que o pacote está contido, caso não esteja nas bibliotecas "std" ou "work". O exemplo abaixo

permite acesso aos tipos e operações do pacote 1164 da IEEE:

```
LIBRARY ieee;
USE ieee.std_logic_1164;
```

Pacotes definidos pelo usuário precisam ser claramente identificados pela palavra chave "package" e ficam disponíveis para seus documentos dentro da biblioteca "work", a não ser que sejam explicitamente declaradas dentro de outra biblioteca definida pelo usuário. O exemplo abaixo mostra a declaração e utilização de um pacote:

```
PACKAGE meu_pacote IS
  TYPE \textbf{meu_inteiro} IS range -127 to 127;
END meu_pacote;

USE work.meu_pacote.all;

ENTITY minha_entidade IS
PORT(
    entrada : in \textbf{meu_inteiro};
    saida: out \textbf{meu_inteiro} );
END minha_entidade;
```

2.4.5 Sintetização

Por ser uma linguagem muito abrangente e complexa, algumas das funcionalidades possíveis na linguagem podem não ser possíveis na implementação física, como leitura de arquivos. Portanto é importante considerar o circuito real desejado. A ferramenta de síntese e simulação também deve ser considerada, pois a mesma pode aceitar diversos comandos irrealizável no mundo real.

A sintetização ocorre em várias etapas para levar a descrição do nível de abstração da linguagem até o circuito real. Os fabricantes de dispositivos reprogramáveis (FPGAs) costumam fornecer ferramentas que fazem todo o processo automaticamente. Esses níveis geralmente são:

1. Análise

   Verificação de sintaxe, tipos e hierarquia do código, semelhante a uma primeira passagem de um compilador. A ferramenta de síntese pode também gerar uma descrição VHDL mais simplificada nessa etapa através de iterações, traduzindo macros para estruturas mais simples até obter uma descrição sintetizável.

2. Geração de circuito RTL (*Register Transfer Level*)

   É gerado um circuito de primitivas genéricas no sintetizador, como contadores, somadores e portas lógicas

3. Portas

As primitivas genéricas são substituídas por componentes reais do dispositivo empregado, como flip-flops, carrys e latches.

4. *Place and Route*

   Com a descrição de componentes e ligações entre os mesmos, o sintetizador faz o posicionamento das primitivas em um dispositivo, utilizando os componentes e ligações disponíveis.

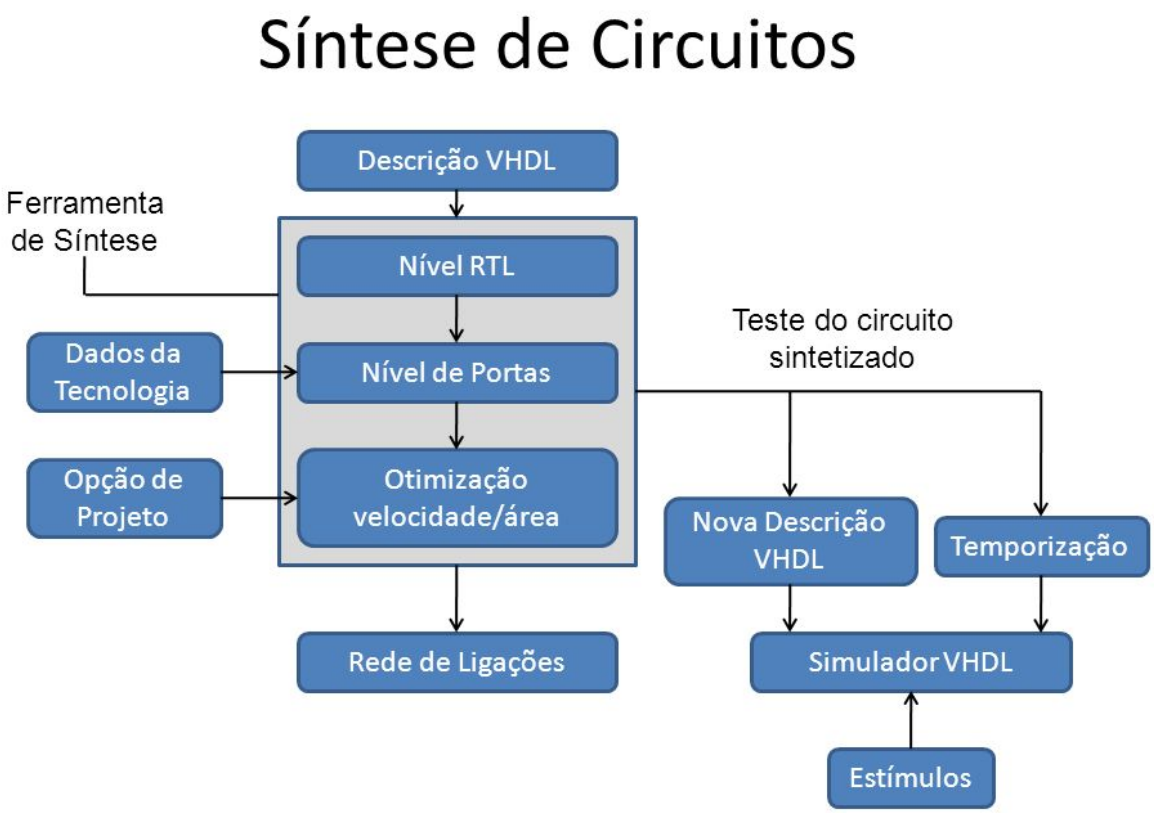


Figura 2.9: Síntese de uma descrição VHDL

2.4.6 Simulação

Uma das ferramentas mais úteis para o desenvolvimento de qualquer sistema é o simulador que dá ao projetista a possibilidade de simular o projeto em suas várias etapas. Com as linguagens de descrição de hardware não é diferente. Geralmente os sintetizadores de VHDL ou Verilog também são vendidos ou distribuídos com algum simulador.

A princípio pode-se pensar que um simulador rodando em uma arquitetura de Von Neumann não será útil pois não tem a capacidade de processar na mesma velocidade que um *hardware* de aplicação específica. Porém, o processo de compilação e escrita em uma placa FPGA tem um alto custo de tempo e o simulador normalmente é capaz de dar resultados satisfatórios apenas com

uma sintetização a nível de RTL. Dessa forma é possível testar e corrigir rapidamente alterações no projeto.

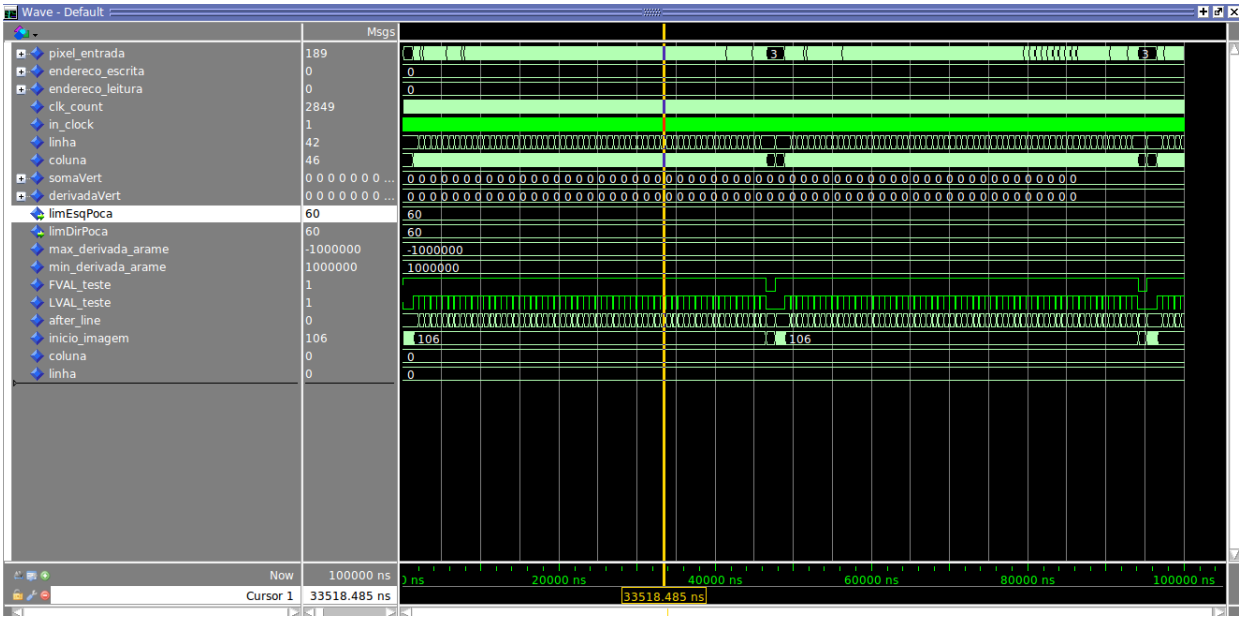


Figura 2.10: Resultado de uma simulação em VHDL

A seguir serão descritos alguns aspectos de simuladores de forma sucinta.

2.4.6.1 Funcionamento

O simulador considera um circuito como uma coleção de sinais e processos. Sinais podem mudar de valor com o tempo sob o impacto de processos. Uma mudança de sinal é chamada de transação. [28]

O que um simulador deve fazer em um determinado momento é indicado por uma lista de ações classificadas por tempo. Esta é a **lista de eventos**. Evento é a designação para uma mudança de sinal ou ativação de um processo em um momento específico. Por exemplo, se um processo que está ativo no momento $t = t_0$ tem o comando $a <=' 1' \ after \ 10ns$, então a transação $a <=' 1'$ é colocada na lista de eventos no momento $t = t_0 + 10ns$. [28]

Uma transação nunca ocorre imediatamente, mesmo quando se o código não especificar nenhum atraso($after$). Portanto, uma transação é colocada na lista de eventos no momento $t = t_0 + \Delta$, em que Δ é zero (ou infinitesimalmente pequeno). Isso permite que processos que ocorrem simultaneamente sejam ordenados no tempo. Isto é possível porque $0 < \Delta < 2\Delta$ para o simulador. O conceito de infinitesimalmente pequeno é chamado de **atraso delta**. [28]

A simulação começa com a criação da lista de eventos. Todos os processos da descrição VHDL são colocados na posição correta da lista. Durante a simulação a lista de eventos é processada em ordem temporal. Novos eventos que resultantes de transações são colocados na posição correta da lista. A simulação acaba quando a lista de eventos está vazia, quando é forçada a terminar(restrição de tempo) ou encontra um erro. [28]

Este processo é chamado de simulação orientada a eventos (*event-driven*) e é utilizada por praticamente todos os simuladores digitais. [28]

2.4.6.2 Estímulos de entrada

Alem da descrição de *hardware*, um simulador precisa dos sinais de entrada que influenciarão os processos.

Alguns softwares tem funções que ajudam o projetista a criar estímulos para o simulador, como um arquivo de formas de onda (*waveform*). Esses arquivos são facilmente editáveis com funções simples para vetores de ondas como: clock, contadores, valores aleatórios, textos ou edição manual feita visualmente. Estes arquivos são bastante úteis para o teste de blocos mais simples.

Porém, simulações mais detalhadas e extensas é mais comum o uso de *testbenches*. Os *testbenches* são arquivos criados em VHDL e aproveitam a própria estrutura e os elementos da linguagem para fornecer os valores desejados ao simulador.

O mais comum é se construir um *testbench* com uma entidade de projeto sem entradas ou saídas e apenas um componente que é a entidade de projeto a ser testada. Este componente costuma ser chamado de *Design Under Test* (DUT), ou *Design Under Verification* (DUV). Dessa forma é possível criar valores para todos os sinais de entrada do DUT dentro da arquitetura do *testbench*.

Para alterar os valores de entrada com o tempo, podem ser utilizados os comandos "after" e "wait". Estes comandos, em conjunto com a estrutura do VHDL, os procedimentos e processos, permitem criar sinais síncronos ou assíncronos para fornecer dados de entrada a um *testbench*. Pode-se criar um clock, por exemplo.

2.4.6.3 Apresentação de resultados

Os simuladores costumam apresentar os resultados da simulação em uma janela com as formas de onda dos sinais de saída do DUT. Para facilitar a leitura, os simuladores apresentam os valores sobre as ondas, ou ao lado destas. Se um vetor é representado, podem ser visualizadas as palavras binárias ou alguma decodificação destas, como valores decimais, hexadecimais, ou mesmo caracteres.

Em geral, pode-se remover ou adicionar outros sinais internos ao DUT, como saídas e entradas de entidades instanciadas no DUT ou mesmo sinais internos a estas instâncias. Esta funcionalidade é extremamente poderosa pois permite testar o funcionamento de todas as partes do projeto, independente do nível de hierarquia.

A figura 2.10 demonstra a variação de diversos sinais ao longo do tempo em uma simulação. Pode-se ver na coluna mais à esquerda a lista de sinais escolhidos para se visualizar os resultados. A coluna à direita mostra as formas de onda desses sinais ao final da simulação. A coluna do meio mostra os valor assumido por cada sinal no momento marcado pela linha amarela na coluna à direita.

2.5 Aquisição de imagens

As imagens necessárias para o processamento devem ser obtidas adequadamente à velocidade do processo em questão. Apesar do processo de soldagem não ser muito rápido se comparado à velocidade de circuitos digitais, a obtenção das informações úteis não é trivial. As melhores imagens para identificação de parâmetros só são visíveis por curtos períodos de tempo durante o momento de curto-circuito da solda, não mais que alguns milissegundos.

Figura 2.11: Câmera industrial

Câmeras comuns não são capazes de registrar estas imagens, pois não conseguem transmitir os dados de forma suficientemente rápida. O curto-circuito começa e acaba antes que a câmera transmita um *frame* inteiro. O mais comum nesse caso é uma imagem com brilho excessivo provindo do arco de solda que dura muito mais tempo que o curto-circuito.

Algumas câmeras especiais, desenvolvidas para uso industrial ou pesquisa científica, são capazes de registrar *frames* em períodos extremamente reduzidos de tempo. Estas possuem alta taxa de captura de imagens e consequentemente uma alta taxa de transmissão de dados. Este é o caso da câmera DALSA DS-21-0001M150 (figura 2.11), utilizada neste trabalho, que possui uma taxa máxima de amostragem de 150 fps com resolução de 1 *megapixel*(1024 x 1024 pixels). Com cada pixel representado em 8 níveis de cinza, a taxa de transmissão desta câmera é de 1200 Mb/s, ou aproximadamente 1,2 Gb/s. [29]

2.5.1 Camera Link

Atualmente existem protocolos rápidos o suficiente para transmissão de dados à partir da ordem de 1 Gb/s, como a fibra ótica e os cabos ethernet Cat 6 e Cat 7. Mas a câmera em questão utiliza um outro protocolo padronizado especificamente para aplicação com câmeras, o *Camera Link*. Este protocolo permite transmissão de até 6,8 Gb/s em uma configuração máxima de três *Channel-links*

e transmite até 80 bits por ciclo. Esta característica é muito interessante para uso com FPGAs pois tem um grau de paralelismo naturalmente superior à outros protocolos.

O Camera Link usa de um a três *Channel-links*. Com apenas um é classificado como Configuração Base (*Base Configuration*) e pode transmitir até 2,04 Gb/s. As outras configurações são a Média (*Medium Configuration*) e a Completa (*Full Configuration*) e transmitem 4,08 e 6,8 Gb/s respectivamente.

O *Channel-link* é uma interface de comunicação com um par emissão e recepção. Ele recebe 28 bits e os transmite de forma serial em quatro canais de 7 bits e um canal de clock, todos em LVDS(*Low Voltage Differential Signaling*). [30]

2.5.1.1 Especificações

A configuração base transmite 24 bits de dados do pixel mais 3 bits de validação e um bit reserva a cada ciclo de clock. Os bits de validação são *Data Valid*, *Line Valid* e *Frame Valid*. As configurações média e completa utilizam doi e três *Channel-links* respectivamente, duplicando ou triplicando a quantidade de bits transmitios simultaneamente. Elas podem omitir os bits de validação redundantes e transmitir até 80 bits de dados por ciclo [30]. Esta característica da interface *Camera Link* a torna ideal para processamento com alto nível de paralelismo, como em FPGAs.

A transmissão serial é feita com 7 bits em cada canal a cada ciclo de clock. O início de cada palavra fica no meio do valor alto do clock e o clock tem o período de valor alto maior que o de valor baixo na proporção 4:3. Esta configuração pode ser melhor observada na imagem 2.12. [31]

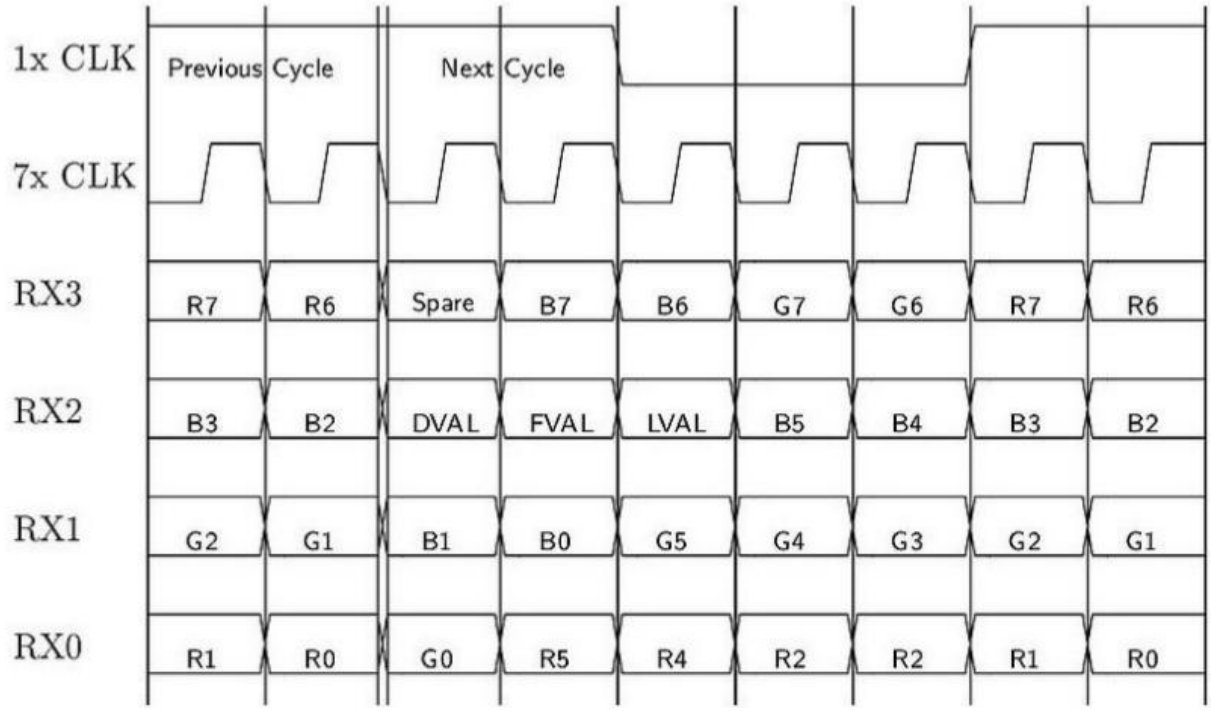


Figura 2.12: Codificação de dados do *Camera Link*

Pode-se notar na imagem 2.12 que os dados de cada cor são distribuídos de forma complicada entre os quatro canais da interface.

Os bits de validação dependem basicamente da câmera utilizada mas seguem sempre um mesmo padrão [30]:

- FVAL (*Frame Valid*) tem o valor ALTO para linhas válidas e sem deslocamento entre a borda de subida de FVAL e o começo da primeira linha válida.
- LVAL (*Line Valid*) tem o valor ALTO para pixels válidos e sem deslocamento entre o começo de LVAL e o primeiro pixel válido.
- DVAL (*Data Valid*) tem o valor ALTO para dados válidos.

Estes bits podem ser melhor visualizados no manual do fabricante da câmera utilizada nos experimentos, conforme a imagem 2.13 abaixo [32]:

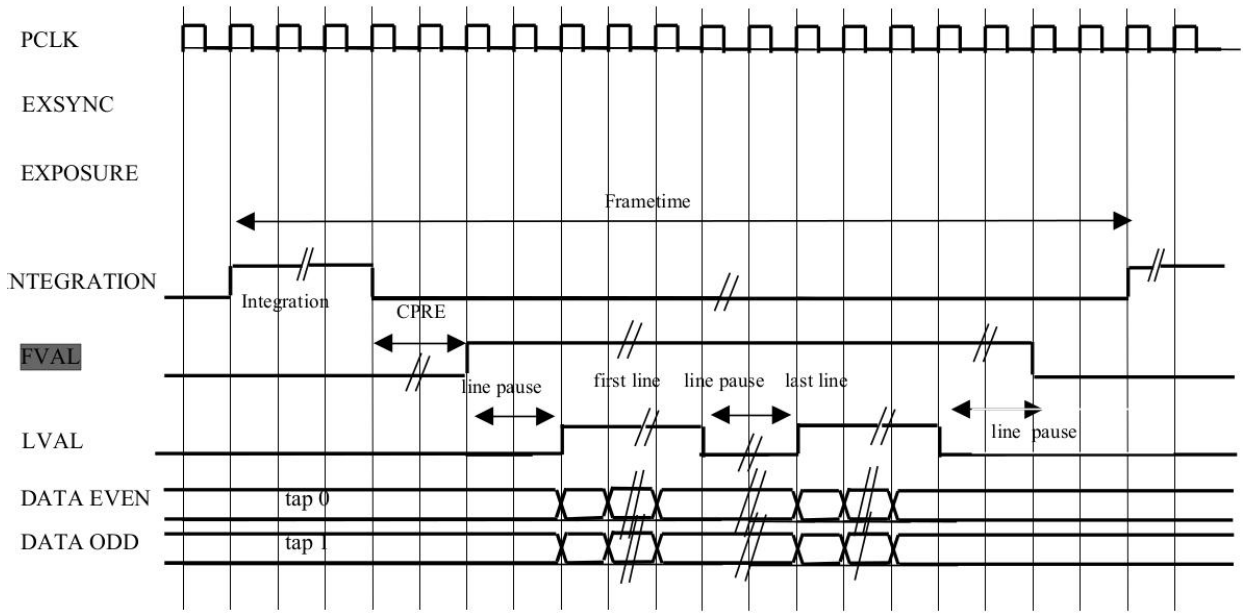


Figura 2.13: Esquema de validação de dados da câmera DALSA DS-21-0001M150

A imagem 2.13 demonstra com clareza os momentos de ativação e desativação dos bits de validação, além de outras características interessantes dessa transmissão de dados. O valor *INTEGRATION* por exemplo se refere ao tempo de exposição da câmera, CPRE (*Clocks after exposure of the sensor*) que é um tempo mínimo entre a exposição e o início da transmissão de dados e *line pause* é o tempo entre uma linha e outra.

2.5.2 Configuração da câmera

A câmera é extremamente versátil e permite uma grande gama de configurações que podem inclusive alterar a duração de alguns tempos demonstrados na imagem 2.13. Algumas opções incluem [32]:

- Tempo de exposição - é a duração de tempo que a câmera recebe luz do ambiente externo a cada imagem. Normalmente, quanto maior for o tempo de exposição, maior o brilho das imagens obtidas;
- *Frame Time* - define o tempo de duração de cada frame, ou seu inverso: a taxa de aquisição da câmera, também chamada de FPS (*Frames Per Second*).
- Dimensionamento de janela - é uma das funções mais interessantes dessa câmera, pois permite usar apenas setores de interesse dentro da janela da câmera. Isto permite a aquisição de imagens menores a uma taxa mais elevada de FPS. Por exemplo, pode ser transmitido um vídeo de 1024 x 1024 pixels (1 Mp) a 150 FPS ou um de 512 x 1024 pixels (0,5) Mp a 300 FPS.

Todos esses parâmetros podem ser definidos através de comunicação serial por um software fornecido pelo fabricante. As configurações ficam armazenadas na EEPROM (Electrically-Erasable Programmable Read-Only Memory) da câmera e continuam em vigor mesmo após a mesma ser desligada. Isso significa que a câmera pode ser configurada com o software e utilizada para aquisição com outra interface mesmo após ser desligada.

2.6 Regressão Linear

Regressão é uma ferramenta comum para diversos problemas em engenharia e estatística em geral. Uma regressão consiste basicamente em uma função matemática que descreve a relação entre duas variáveis sendo o comportamento de uma dependente da outra [33].

Em uma regressão, uma das variáveis, normalmente chamada de x, é tida como independente ou variável controlada. Esta variável pode ser medida com bom nível de exatidão. A outra variável, normalmente tida como y é aleatória. O que se deseja obter é a função matemática $\mu(x)$ de dependência que y tem em relação a x [33].

Uma regressão linear é um caso específico, o mais simples dentre as regressões, e a função que se procura é uma linha reta da forma:

$$\mu(x) = b + ax \tag{2.17}$$

O método mais utilizado para se encontrar a reta que melhor descreve uma sequencia com pontos do tipo $(x_1, y_1),\cdots,(x_n, y_n)$ é o método dos Mínimos Quadrados Ordinários, ou MQO. O princípio é o seguinte: "A reta deve ser ajustada através dos pontos de modo que a soma dos quadrados das distâncias entre os pontos e a reta seja mínima, onde a distância é medida na vertical(enxo y)."[33]

Além deste princípio é necessária mais uma condição para a solução matemática única deste método: "Os valores $x_1,\cdots,x_n$ na amostra $(x_1, y_1),\cdots,(x_n, y_n)$ não podem ser todos iguais."[33]

À partir de uma amostra $(x_1, y_1),\cdots,(x_n, y_n)$, descreve-se a seguinte linha, chamada **linha de regressão exemplo**:

$$y = b + ax \tag{2.18}$$

A distância vertical entre um ponto (x_i, y_i) e a reta em 2.18 é dada por:

$$|y_i - (b + ax_i)| \tag{2.19}$$

Portanto a soma dos quadrados dessas distâncias é

$$q = \sum_{i=1}^{n} (y_i - b - ax + i)^2 \tag{2.20}$$

Para o MQO, deve-se determinar a e b de modo a minimizar o valor de q na equação 2.20. Portanto:

$$\frac{\partial q}{\partial a} = 0 \tag{2.21}$$

$$\frac{\partial q}{\partial b} = 0 \tag{2.22}$$

A linha exemplo pode ser reescrita da forma:

$$y - \bar{y} = a(x - \bar{x}) \tag{2.23}$$

Onde $\bar{x}$ e $\bar{y}$ são os valores médios de x e y na amostra. O coeficiente a na equação 2.23 é chamado de corficiente de regressão e é dado por:

$$a = \frac{s_{xy}}{x_x^2} \tag{2.24}$$

A covariância de amostra s_{xy} é:

$$s_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \frac{1}{n-1} \left[\sum_{i=1}^{n} x_i y_i - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right) \left(\sum_{i=1}^{n} y_i \right) \right] \tag{2.25}$$

E a variância dos valores x, s_x^2 é dada por:

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right] \tag{2.26}$$

De 2.23 pode-se observar que a reta passa pelo ponto $\bar{x}, \bar{y}$. Como x é uma variável ordinária, será necessária também a variância dos valores y:

$$s_y^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} y_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} y_i\right)^2\right] \tag{2.27}$$

Ao se diferenciar a equação 2.20 e se aplicar 2.22 resulta em:

$$\frac{\partial q}{\partial a} = -2\sum(y_i - b - ax_i) = 0 \tag{2.28}$$

$$\frac{\partial q}{\partial b} = -2\sum x_i(y_i - b - ax_i) = 0 \tag{2.29}$$

Onde se soma de i de 1 a n. Divide-se por 2, escreve-se cada lado das duas somas como três somas e leva-se as somas contendo x_i e $x_i y_i$ para o lado direito. Então obtém-se as equações normais:

$$bn + a\sum x_i = \sum y_i \tag{2.30}$$

$$b\sum x_i + a\sum x_i^2 = \sum x_i y_i \tag{2.31}$$

Este é um sistema linear de duas equações e duas variáveis e pode ser resolvido com a equação 2.32 abaixo:

$$\begin{vmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{vmatrix} = n\sum x_i^2 - \left(\sum x_i\right) = n(n-1)s_x^2 = n\sum(x_i - \bar{x})^2 \tag{2.32}$$

Este sistema tem solução única desde que sejam atendidas as condições necessárias. Dividindo-se a equação 2.32 por n e usando as médias, tem-se $b = \bar{y} - a\bar{x}$. Para se obter a, resolve-se 2.32 com a regra de Cramer:

$$a = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n(n-1)s_x^2} \tag{2.33}$$

2.6.1 Regressão Linear para FPGAs

A solução de uma regressão pelo método dos Mínimos Quadrados exige uma certa quantidade de cálculos computacionalmente custosos, como multiplicações e divisões. Quando se deseja ocupar poucos elementos em uma FPGA e reduzir a quantidade de operações, algumas simplificações podem ser feitas. Isto mostrado nesta seção.

De acordo com [34], o primeiro passo é definir que os valores em x são inteiros e começando à partir de 0. Desta a equação 2.20 pode ser reescrita da seguinte forma:

$$q = \sum_{i=0}^{n-1} (y_i - b - ax + i)^2 \tag{2.34}$$

Ao se aplicar o quadrado e separar a equação 2.34 em diferentes somas tem-se:

$$q = \sum_{i=0}^{n-1} y_i^2 + a^2 \sum_{i=0}^{n-1} i^2 - 2a \sum_{i=0}^{n-1} iy_i + nb^2 + 2ab \sum_{i=0}^{n-1} i - 2b \sum_{i=0}^{n-1} y_i \tag{2.35}$$

Aplicando-se derivadas parciais na equação 2.35:

$$\frac{\partial q}{\partial a} = 2a \sum_{i=0}^{n-1} i^2 - 2 \sum_{i=0}^{n-1} iy_i + 2b \sum_{i=0}^{n-1} i = 0 \tag{2.36}$$

$$a = \left(\sum_{i=0}^{n-1} iy_i - b \sum_{i=0}^{n-1} i \right) / \sum_{i=0}^{n-1} i^2 \tag{2.37}$$

$$\frac{\partial q}{\partial b} = 2nb + 2a \sum_{i=0}^{n-1} i - 2 \sum_{i=0}^{n-1} y_i = 0 \tag{2.38}$$

$$b = \left(2 \sum_{i=0}^{n-1} y_i - a \sum_{i=0}^{n-1} i \right) / n \tag{2.39}$$

No final tem-se:

$$a = \left(n \sum_{i=0}^{n-1} iy_i - \sum_{i=0}^{n-1} y_i \sum_{i=0}^{n-1} i \right) / \left(n \sum_{i=0}^{n-1} i^2 - \sum_{i=0}^{n-1} i \sum_{i=0}^{n-1} i \right) \tag{2.40}$$

$$b = \left(\sum_{i=0}^{n-1} y_i \sum_{i=0}^{n-1} i^2 - \sum_{i=0}^{n-1} iy_i \sum_{i=0}^{n-1} i \right) / \left(n \sum_{i=0}^{n-1} i^2 - \sum_{i=0}^{n-1} i \sum_{i=0}^{n-1} i \right) \tag{2.41}$$

Como visto nas equações 2.40 e 2.40, ambos os coeficientes dependem da quantidade de pontos utilizados. São necessárias muitos somatórios com multiplicações e duas divisões.

Se esta quantidade for limitada para um valor constante N, obtém-se equações muito mais simplificadas, pois os somatórios que não dependem de x ou y passam a ter valores constantes. Esta limitação reduz boa parte dos cálculos dos fatores, incluindo os denominadores.

Considere-se que:

$$\sum_{i=0}^{N-1} i = \frac{N^2}{2} - \frac{N}{2} \tag{2.42}$$

$$\sum_{i=0}^{N-1} i^2 = \frac{N^3}{3} - \frac{N^2}{2} + \frac{N}{6} \tag{2.43}$$

Os coeficientes podem ser então encontrados apenas multiplicando-se constantes por somatórios e subtraindo resultados como pode ser visto em 2.44 e 2.45 [34].

$$a = \left(\frac{N}{N\sum i^2 - \sum i \sum i}\right) \cdot \sum_{i=0}^{N-1} i d_i - \left(\frac{\sum i}{N\sum i^2 - \sum i \sum i}\right) \cdot \sum_{i=0}^{N-1} d_i \tag{2.44}$$

$$b = \left(\frac{\sum i^2}{N\sum i^2 - \sum i \sum i}\right) \cdot \sum_{i=0}^{N-1} d_i - \left(\frac{\sum i}{N\sum i^2 - \sum i \sum i}\right) \cdot \sum_{i=0}^{N-1} i d_i \tag{2.45}$$

2.6.2 Regressão Robusta

2.7 Filtro de Kalman

Retirar ruídos dos dados gerados pelo processamento de imagens. Geração do sinal de controle à partir dos vetores gerados pelo algoritmo de processamento.

Capítulo 3

Desenvolvimento

3.1 Introdução

O trabalho consistiu basicamente em duas etapas. Na primeira foi desenvolvido um algoritmo de processamento em um computador com software de alto nível de abstração, o Matlab, e na segunda foi feita a implementação em hardware do mesmo algoritmo utilizando um FPGA.

Inicialmente, no algoritmo em Matlab, foi criado o algoritmo para fazer as medidas necessárias em apenas uma imagem, até que se percebeu que seria necessário usar média de imagens. Depois passaram a ser processadas grandes grupos de imagens armazenadas. O sistema em FPGA, escrito em VHDL, foi concebido para processar imagens à medida que são capturadas, em tempo real.

Infelizmente não foi possível obter dados reais com a câmera, pois não havia uma interface física entre o cabo de comunicação da câmera e o FPGA. Por esse motivo foi utilizada a memória da placa para fornecer dados ao processador de imagens desenvolvido.

3.2 Arquitetura geral

O sistema deverá fazer parte de um controlador em malha fechada, que teoricamente será superior ao controle de malha aberta comumente utilizado em sistemas soldas automatizadas. Ou seja, o sinal de entrada deve ser o resultado da soma do sinal de referência, que será o trajeto definido na programação de um robô, e do sinal de controle, que será calculado por um computador.

Para que o sistema seja realimentado é necessário obter medidas da saída do sistema e, se possível, outros estados. O estado de saída no caso será obtido através de um algoritmo que irá processar fotos do processo e retornar valores numéricos relevantes à operação. Esta etapa é feita com o uso de uma câmera de alta taxa de captura de imagens, capaz de fotografar o arame e a poça de soldagem no instante em que há um curto circuito. O algoritmo processa as imagens e obtém valores numéricos de tamanho da poça, desvio e inclinação do arame.

3.3 Implementação em Matlab

O primeiro algoritmo foi desenvolvido para funcionamento de forma sequencial, pois seria implementado com o software Matlab. A sequência de operações foi definida de acordo com a descrição abaixo. As descrições entre parênteses indicam os parâmetros utilizados em testes que obtiveram bons resultados.

3.3.1 Algoritmo Base

O diagrama da figura 3.1 demonstra a sequência de instruções criadas no Matlab para processar as imagens:

A sequência de passos pode ser lida com mais detalhes abaixo:

1. Limpar memória e fechar figuras
2. Carregar quantidadeImagens imagens (quantidadeImagens=3)
3. Criar imagem média das imagens carregadas -> I
4. Retirar pixels abaixo de threshold1 para retirar scanlines (threshold1 = 10)
5. Filtro de mediana em I para retirar ruído -> I2
6. Retirar pixels abaixo de threshold2 de I2 -> B (threshold2 = 40)
7. Encontrar topo e base do arame

   Criar vetor de soma do perfil horizontal de B -> somaHor

   Criar vetor de derivadas de somaHor -> derivadaHor

   Encontrar topo do arame

   Valor máximo de derivadaHor entre o topo da imagem e o provável meio do arame (meioVert = 125) -> posArameTopo

   Encontrar base do arame

   Valor mínimo de derivadaHor entre o meio do arame e o fim da imagem - posArameBase
8. Encontrar borda esquerda e direita da poça de soldagem

   Criar vetor de soma do perfil vertical de B -> somaVert

   Criar vetor de derivadas de somaVert -> derivadaVert

   Encontrar borda esquerda da poça

   Valor máximo de derivadaVert entre o começo e o meio da imagem -> limEsqPoca

   Encontrar borda direita da poça

   Valor mínimo de derivadaVert entre o meio e o final da imagem -> limDirPoca

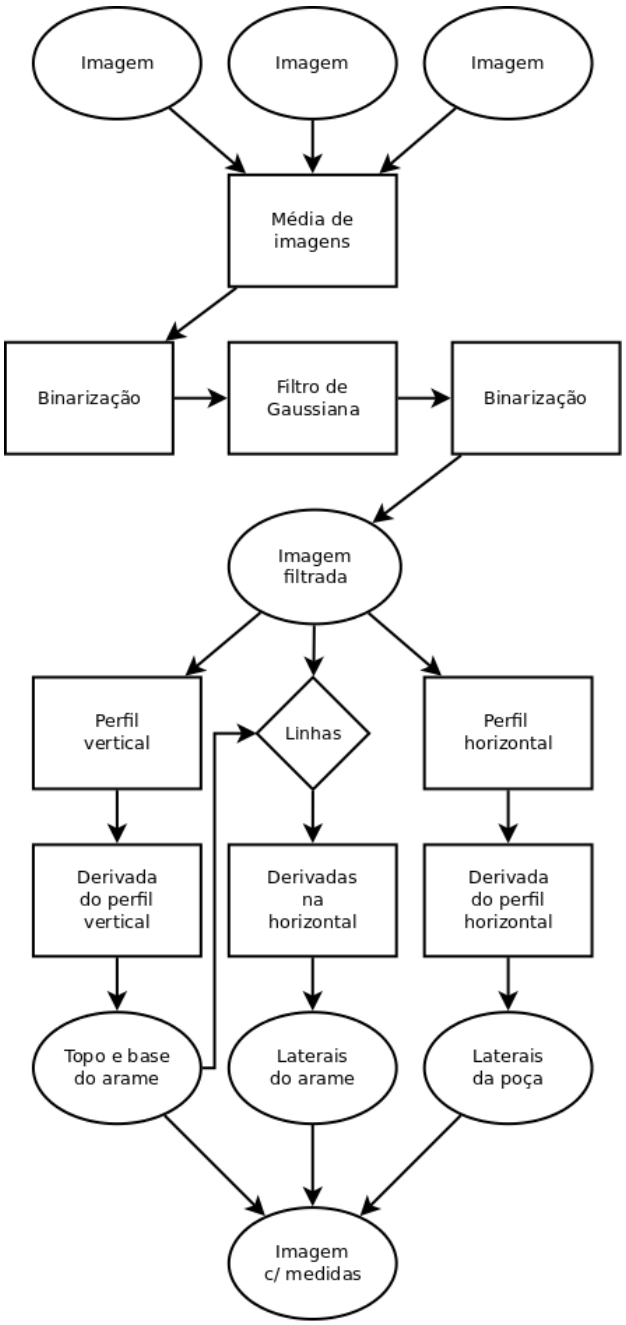


Figura 3.1: Diagrama do algoritmo utilizado

9. Encontrar laterais do arame

   Criar vetor de derivadas de B na horizontal entre topo e base do arame -> derivadaArame

   Valores mínimos de derivadaArame -> vetor inicioArame

   Valores máximos de derivadaArame -> vetor fimArame

   Linearizar inicioArame e fimArame

Esses passos com os valores para os parâmetros definidos acima obtiveram bons resultados para uma grande quantidade de imagens (antigas). O algoritmo demonstrou certa robustez à ruídos,

porém quando a imagem foge muito ao padrão esperado, o algoritmo fornece valores equivocados. Um exemplo é quando há uma grande gota de solda fora da área da poça.

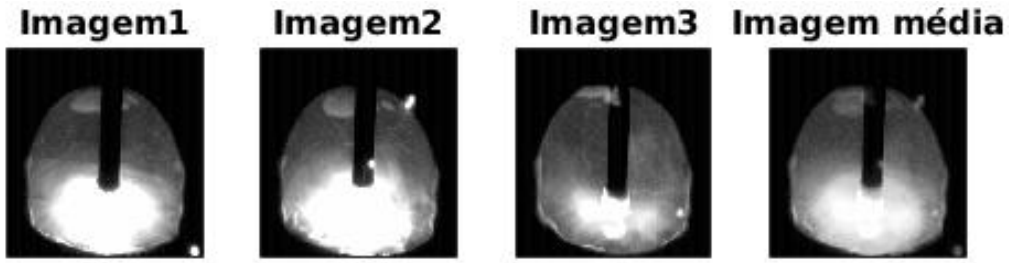
3.3.2 Filtros

A escolha dos filtros utilizados foi baseada nas características do processo e das próprias imagens, já conhecidas. Abaixo segue a sequência de filtro utilizados.

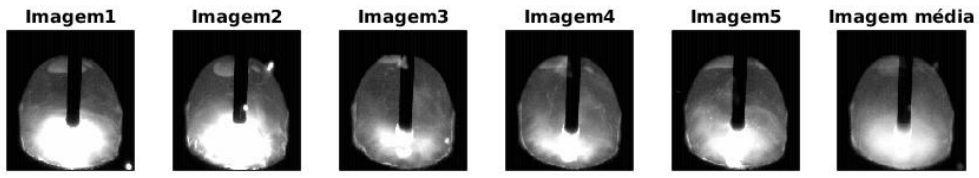
1. Média de imagens

   Não é um filtro muito comumente utilizado, mas se provou útil neste trabalho. Observou-se em várias imagens objetos brilhosos que aparecem aleatoriamente, provavelmente respingos de solda e fagulhas. São ruídos muito grandes para serem retirados com filtros de janela. O interessante é que estes objetos não costumam estar presentes em mais de uma imagem. As características do processo favorecem o uso da média, pois as imagens as formas a serem identificadas mudam pouco de *frame* a *frame*.

   A média de imagens reduz bastante este efeito e quanto maior o número de imagens utilizadas, menor o efeito de deste tipo de ruído. Em contrapartida, o uso de muitas imagens pode propagar o ruído durante muitas leituras, o que comprometeria a eficácia do processamento de sinais mais tarde. Portanto, deve-se escolher uma quantidade razoável de imagens a serem utilizadas na média. Durante as simulações observou-se que três a cinco imagens são o suficiente para atenuar consideravelmente estes ruídos.



(a)



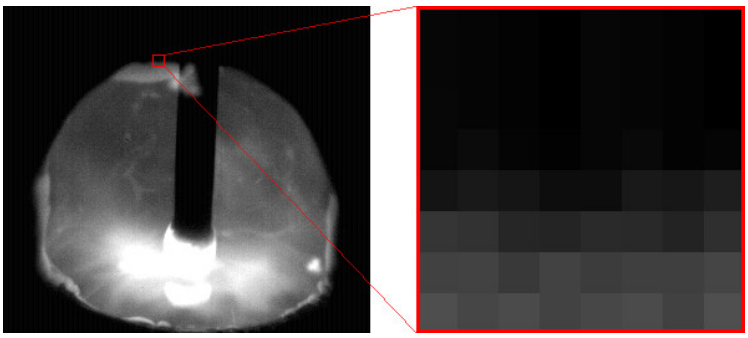
(b)

Figura 3.2: Média de três imagens (a). Média de cinco imagens(b)

2. Binarização

   Esta binarização retira as *scanlines* verticais que estão presentes em todas as imagens. Essas *scanlines* são pouco mais claras que o preto total (geralmente menor que 6/255), portanto é

utilizado um filtro de binarização simples. Quase nenhuma informação útil é perdida neste passo. Pois a maior parte das informações úteis estão contidas em pixels com brilho médio a alto (acima de 100/255).



6	5	3	0	6	5	3	0
6	5	3	0	6	5	3	0
7	5	3	0	6	5	3	0
7	11	5	2	6	9	3	5
20	25	21	13	13	25	23	30
53	50	38	36	42	41	35	47
65	66	57	66	60	63	63	69
78	70	75	66	71	75	65	79

Figura 3.3: Detalhe de imagem e valores numéricos dos pixels

A figura 3.3 demonstra um pedaço da imagem onde podem-se ver as *scanlines* e como a parte mais brilhosa não sofre este efeito. Ao lado os valores numéricos são mostrados. Concluiu-se que o valor de $threshold1 = 7$ para a retirada das scanlines era mais que suficiente, sem afetar informações úteis.

3. Gaussiana

   Este filtro é de gaussiana com janela 3x3 para tirar pequenos ruídos. Também contribui para suavizar as bordas antes do próximo filtro.

4. Binarização

   O segundo filtro de binarização tem o objetivo de retirar mais ruídos e acentuar as bordas da imagem. Usado depois do filtro de gaussiana tem-se a impressão de bordas mais definidas. Por exemplo, o perfil fica mais reto onde deveria haver uma reta. Ambos os filtros de binarização são parciais e mantêm o brilho dos pixels com valor acima do $threshold2 = 40$ e abaixo do $threshold3 = 100$.

   É importante ressaltar que a ordem de utilização dos filtros afeta o resultado final, por isso foi feita a primeira binarização antes do filtro de gaussiana para que as *scanlines* não interferissem nas bordas com informações úteis.

3.3.3 Perfis Verticais e Horizontais

Uma das ferramentas essenciais neste trabalho foi o uso de perfis verticais e horizontais e suas derivadas. A escolha deste método levou em consideração algumas características particulares do processo em questão:

1. As imagens são em tons de cinza, portanto apenas há apenas um valor numérico a ser lido em cada píxel;

2. O objeto de interesse nas imagens é bastante definido em relação ao fundo e é único;

3. Este método permite uma comparação relativamente precisa entre imagens seguidas.

São gerados dois vetores de perfis, o $somaHor$ e o $somaVert$ e suas derivadas $derivadaHor$ e $derivadaVert$, respectivamente.

O vetor $somaHor$ representa a luminosidade somada em cada linha (horizontal) da imagem analisada. Analogamente O vetor $somaVert$ representa a luminosidade somada em cada coluna (vertical) da imagem analisada.

O vetor $derivadaHor$ representa as variações de luminosidade a cada linha (horizontal) da imagem e é utilizado para encontrar as coordenadas Y do topo ($posArameTopo$) e da base ($posArameBase$) do eletrodo na imagem. Essas posições são encontradas nos máximos locais de $derivadaHor$. É possível perceber na imagem **??**

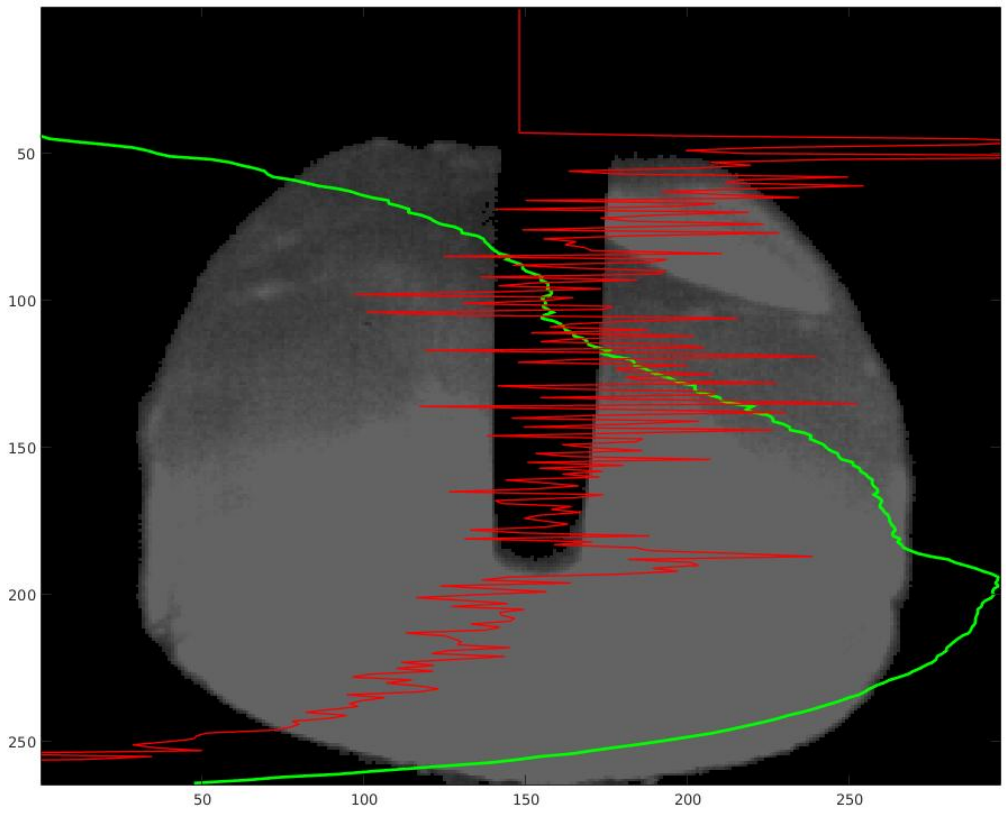


Figura 3.4: Imagem com perfil horizontal(verde) e derivada do perfil(vermelho)

3.3.4 Regressão Robusta

Um dos passos do algoritmo é encontrar os pontos de interesse do eletrodo. Para simplificação dos cálculos considerou-se o perfil do eletrodo em cada imagem como um trapézio. Desta forma apenas são necessários encontrar quatro pontos para se determinar a posição, comprimento e desvio

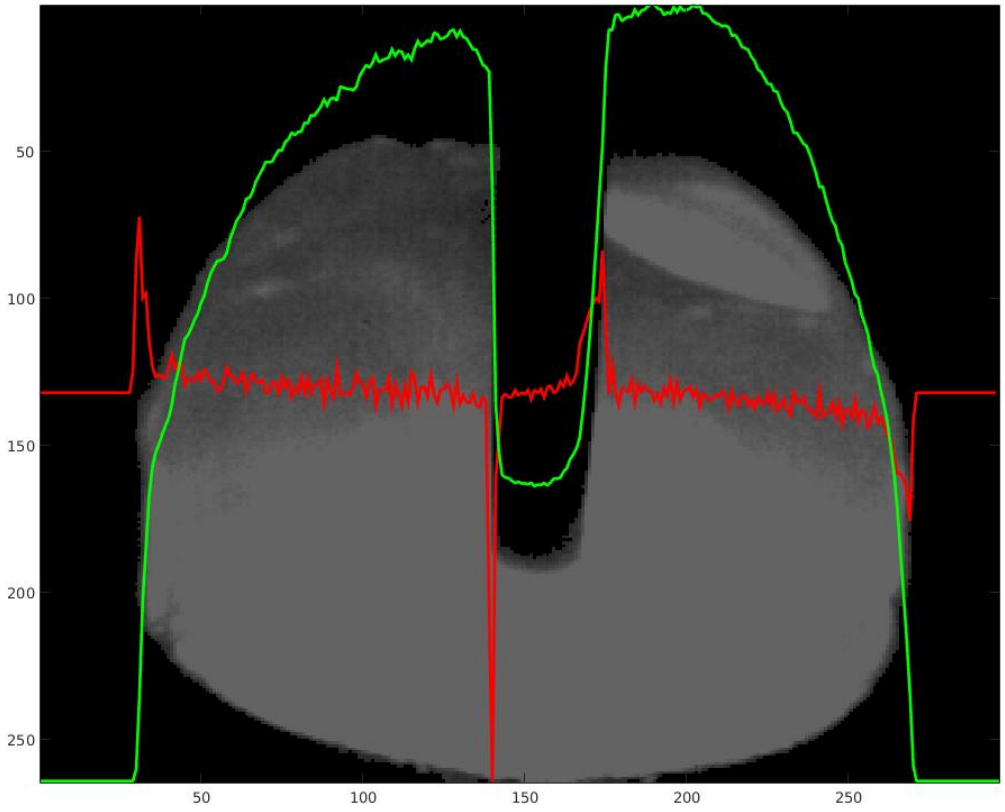


Figura 3.5: Imagem com perfil vertical(verde) e derivada do perfil(vermelho)

angular do eletrodo. Estes pontos são definidos pelas intersecções entre base e topo do eletrodo e suas laterais.

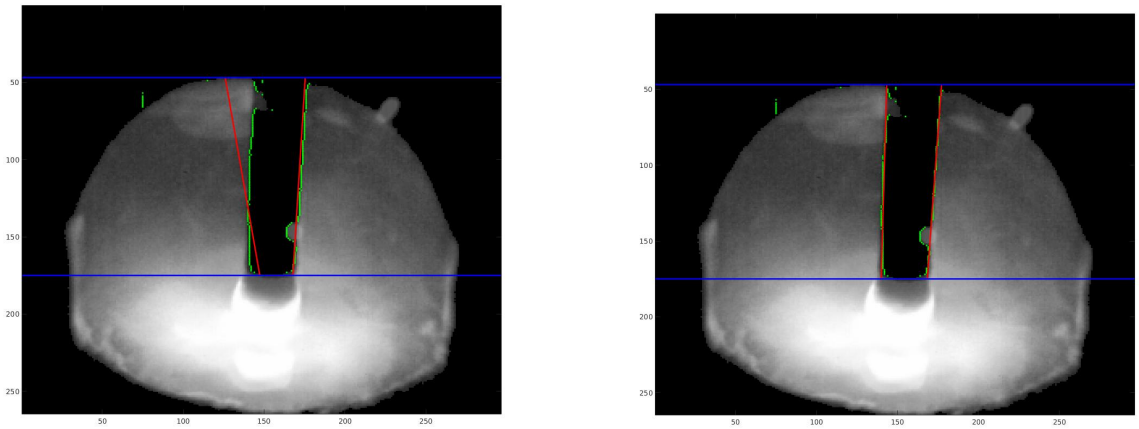
A princípio pensou-se em identificar apenas os pontos das laterais do eletrodo mais próximos ao topo e à base do mesmo para se definir esses pontos. Esta identificação seria pelo método dos perfis verticais (2.2.5). Contudo, verificou-se que é comum haverem objetos brilhosos sobre alguma parte do eletrodo e inclusive sobre parte de suas bordas. Também, a ponta (base) do eletrodo geralmente está deformada por conta da fusão do material neste ponto. Esses objetos ou deformações sobre os pontos desejados causam grandes erros de medição ao simplesmente se utilizar este método.

Portanto, considerou-se decidiu-se que estes pontos deveriam ser encontrados de forma mais robusta. A primeira tentativa foi obter diversos pontos ao longo do arame e se fazer uma regressão linear de tais pontos com a função $polyfit$ do Matlab. Essas retas foram definidas como vetores de dois elementos que representam os coeficientes das equações de reta, armazenados em $ladoEsqArame$ e $ladoDirArame$. Com esses coeficientes, seriam calculadas as coordenadas no eixo X onde as retas encontradas se cruzavam com as medidas Y da base e topo do eletrodo.

A regressão linear comum reduziu alguns dos erros mas ainda não estava satisfatória. Portanto, foi utilizada uma regressão robusta em vez de uma de mínimos quadrados para se obter a equação

da reta de cada lateral do eletrodo.

A função utilizada para essa tarefa foi a *robustfit* do Matlab, com a configuração padrão "bisquare". Esta função retorna o mesmo formato de coeficientes da equação de reta que a função *polyfit*. Dessa forma, as coordenadas dos pontos de interesse são obtidos com o mesmo procedimento.



(a) (b)

Figura 3.6: Regressão por mínimos quadrados (a) versus regressão robusta (b)

A figura 3.6(a) demonstra como a regressão por mínimos quadrados foi capaz de corrigir encontrar corretamente os pontos do eletrodo no lado direito mas falhou para o lado esquerdo. Já a figura 3.6(b) demonstra a vantagem da regressão robusta.

3.3.5 Distorção de perspectiva

A câmera deve ser posicionada obliquamente em relação à poça de soldagem não apenas por questões físicas mas também para que inclua tanto poça de soldagem quanto arame. Isso significa que os elementos (poça e arame) ficam em diagonal com o plano da imagem, o que causa uma distorção de perspectiva do que seria a imagem ideal para processamento. A posição da câmera que capturou as imagens utilizadas nesta parte do experimento pode ser vista na figura 3.7.

Dessa forma, a distorção de perspectiva da imagem deve ser conhecida para que seja possível obter valores corretos no processamento. Uma forma de obter essa informação seria por inserção de dados como ângulo, distância entre câmera e tocha, e distância focal da lente por um operador no sistema. Mas considerando que o sistema possa ser utilizado com câmeras, atuadores e posicionamentos diferentes, inclusive com a possibilidade de alteração de alguma variável entre dois *setups* diferentes, há a possibilidade de erros e aumento do tempo de *setup*.

A princípio, desejava-se calcular a distorção de perspectiva e fazer os ajustes necessários durante o processamento das imagens do processo de solda. A ideia seria utilizar os cantos do eletrodo de solda (dois formados entre o topo do eletrodo e o bocal da tocha e dois entre a base do eletrodo e

Figura 3.7: Posição da câmera

a poça de soldagem) para identificar os pontos necessários para este cálculo. No entanto, alguns fatos foram observados:

1. O eletrodo pode, em dado momento, estar em uma posição oblíqua à peça de soldagem.

   Independente do motivo, à partir de imagens sabe-se que isso acontece no plano ortogonal à direção da soldagem. Presume-se que o mesmo ocorre também no plano que é paralelo à direção de soldagem e perpendicular à peça. Isso significa que o eletrodo pode estar mais à frente ou atrás do centro da tocha de solda, aproximando-se ou afastando-se da câmera. Ou seja, os pontos que deveriam ser conhecidos na base do eletrodo movem-se para locais desconhecidos, impossibilitando o cálculo correto da matriz de transformação.

2. O tamanho do eletrodo varia.

   Fato também observado em imagens, tem o mesmo efeito que a variação de direção do eletrodo. A ponta do eletrodo está hora mais perto hora mais longe da peça de soldagem, mesmo quando em regime estável da soldagem. Isso também muda a posição de pontos que deveriam ser conhecidos.

Devido aos motivos listados o sistema deve obter a matriz de transformação através de uma

pré-calibração. Pensou-se em uma forma simples e rápida: esta pré calibração é feita com o processamento de uma imagem inicial e a matriz de transformação fica armazenada no sistema até que seja feito um novo *setup*. A imagem deve conter o mínimo de quatro pontos conhecidos, como descrito em 2.2.3.2 no espaço tridimensional para que seja possível calcular a matriz de transformação.

A pré calibração é feita utilizando-se um objeto quadrado de dimensões conhecidas no plano paralelo à peça que será soldada. Isso pode ser feito colocando-se uma pequena chapa metálica quadrada sobre o plano de soldagem. Apenas uma imagem é necessária para se obter a matriz de transformação, desde que o resto do ambiente contido na imagem seja bem controlado no momento da calibração. Os algoritmo gerado com Matlab para gerar a matriz de transformação se mostrou eficiente, mas pode apresentar erros caso haja na imagem cantos mais definidos que os cantos do quadrado.

O algoritmo da pré-calibração em Matlab é razoavelmente simples pois utiliza as funções prontas no software. Abaixo seguem os passos utilizados:

1. Filtragem da imagem

   Basicamente são retirados os pixels abaixo de um threshold e depois passado um filtro de gaussiana para retirada de ruídos.

2. Detecção de cantos

   A função "corner" do Matlab é chamada e utiliza o método de Harris (seção 2.2.4) para a detecção. Os cantos são ordenados descendentemente.

3. Cálculo da matriz de transformação

   As coordenadas reais dos cantos são utilizadas em conjunto com as coordenadas obtidas na detecção do passo anterior para o cálculo. (seção 2.2.3.2)

4. Transformação da imagem

   Aqui o algoritmo apresenta a imagem transformada com a matriz de transformação.

5. Correção de valores calculados em etapas anteriores

   As distâncias e posições encontradas anteriormente são corrigidas e sofrem menos influência da distorção.

O interessante de se fazer a correção desta forma, é que também so torna possível estimar desvios do arame na direção do eixo de avanço. Considerando-se uma imagem corrigida após as distorção, observa-se o seguinte: Caso a largura da base do eletrodo tenha a mesma medida horizontal que o topo do eletrodo, este está corretamente alinhado. Caso a base seja menor, o eletrodo só pode estar mais distante do que o topo. Ou seja, está mais longe da câmera, o que significa que o eletrodo está em diagonal, com a ponta mais atrás do que o centro da tocha de soldagem. A mesma lógica aplica-se caso a base tenha uma medida maior que o topo, o que significa que o eletrodo está com a ponta à frente da tocha.

O programa em Matlab utilizou uma imagem definida como um padrão aceitável para calcular a matriz de transformação. Em seguida, o algoritmo passa a obter os valores à partir de imagens transformadas com esta matriz.

3.4 Implementação do processamento em FPGA

O algoritmo utilizado para implementação em Matlab foi adaptado para uma placa FPGA, utilizando-se a linguagem VHDL. Boa parte dos procedimentos continuou com a característica sequencial, pois as imagens são obtidas desta forma. Alguns procedimentos utilizaram do paralelismo para economia de tempo e recursos. Por exemplo: a obtenção de topo e base do arame e largura da poça de soldagem ocorrem simultaneamente. A seguir é demonstrado como foi feita a implementação de cada item do algoritmo.

3.4.1 Aquisição de imagens

Diferentemente do projeto em matlab, o algoritmo para a FPGA engloba faz captura de dados diretamente da câmera. Os dados fornecidos serialmente foram desserializados e disponibilizados em apenas um *std_logic_vector* pois apenas eram necessários níveis de cinza.

A implementação em FPGA desenvolvida apenas recebe imagens para a função *Freerunning Mode* da câmera. A princípio, o algoritmo não precisa de nenhum gatilho de sincronia enviado à câmera pois deve selecionar as imagens por conta própria como descrito em 3.4.4.

O primeiro passo para o recebimento do vídeo pela placa é a sincronização com o clock da câmera. Apesar deste clock ser fornecido pelo *Channel Link*, as palavras de dados não são sincronizadas com a subida do clock. Também é necessario um sinal com 7 vezes a frequencia do clock fornecido para fazer a leitura dos 7 bits de cada palavra transmitida por ciclo.

3.4.2 Retirada de scanlines

Assim que os dados estejam corretamente disponíveis para processamento começam a passar por etapas semelhantes às descritas na seção 3.3. A primeira etapa é a retirada de *scanlines*.

Este filtro de binarização é feito antes de qualquer outro, no momento de recepção de cada pixel. O código para tal é tão simples que foi deixado no corpo da entidade de projeto pois não justifica a criação de uma entidade para tal e não seria adequado colocá-lo em outra.

A binarização consiste basicamente transformar em zero qualquer pixel que seja maior que $threshold1$. O valor utilizado para testes $threshold1$ foi de 10, sendo o brilho máximo 255.

3.4.3 Constantes e tipos customizados

Para agilizar o desenvolvimento do projeto foi criada um pacote com valores e tipos que são recorrentemente utilizados nas diversas entidades. As constantes e tipos dependem basicamente

das dimensões das imagens que se pretende processar. São estes:

1. Constantes
    - *numcols*: Largura da imagem, ou quantidade de colunas (pixels) na dimensão horizontal;
    - *numlin*: Altura da imagem, ou quantidade de linhas (pixels) na dimensão vertical.
2. Tipos
    - *vetorHor*: Vetor utilizado para guardar algum tipo de informação de cada linha. Possui *numlin* elementos do tipo inteiro;
    - *vetorVert*: Vetor utilizado para guardar algum tipo de informação de cada coluna. Possui *numcols* elementos do tipo inteiro;
    - *MatrizImagem* Matriz utilizada para armazenar imagens inteiras na simulação. Tem tamanho de *numlin* por *numcols* do tipo *unsigned* de 8b.

3.4.4 Filtro de imagem média e seleção de imagens propícias

Para que fosse implementado este filtro foi utilizado um bloco de memória subdividido em quatro blocos, cada um dimensionado para receber uma imagem. A placa recebe sequencialmente os dados da câmera e armazena três imagens em memória. A imagem média não necessita ser armazenada pois cada pixel pode ser obtido através da média dos pixels das imagens armazenadas à medida que for necessário.

A seleção de imagens propícias ao processamento foi feito no momento de obtenção da imagem. Uma nova imagem é armazenada no bloco marcado como *bloco_atual*. Ao final do recebimento desta, o sinal *bloco_atual* permanece inalterado caso a imagem apresente brilho excessivo ou tem seu valor aumentado de 1 caso contrário. Se *bloco_atual* permanece com o mesmo valor, a imagem recebida mais recentemente é sobrescrita pela nova imagem que será recebida em seguida. Se *bloco_atual* for acrescido, então a imagem mais antiga é sobrescrita. Desta forma, à partir da terceira imagem sempre há três imagens úteis carregadas e uma sendo recebida.

Para que isto fosse possível durante a simulação, foi implementado um bloco de RAM utilizando uma *megafunction* do Quartus II. O tamanho desta memória calculado para armazenar quatro vezes a quantidade de bits de uma imagem. Para os primeiros estágios do desenvolvimento e teste, foram utilizadas imagens de 296x264 pixels reduzidas a 60x60 pixels (para economia de tempo e melhor visualização da simulação). O tamanho desta memória deveria comportar $60 \times 60 \times 4 = 1440B$, portanto foi utilizada uma memória de 16 KB.

O interessante dessa implementação é que o endereçamento de leitura e escrita pode utilizar diretamente o sinal *bloco_atual* com uma simples concatenação, o que poupa operações na implementação em VHDL e recursos na placa.

Como será visto mais detalhadamente abaixo em 3.4.5, os cálculos e operações são feitos a cada novo pixel durante o carregamento de uma imagem. Assim, a média de cada pixel das últimas

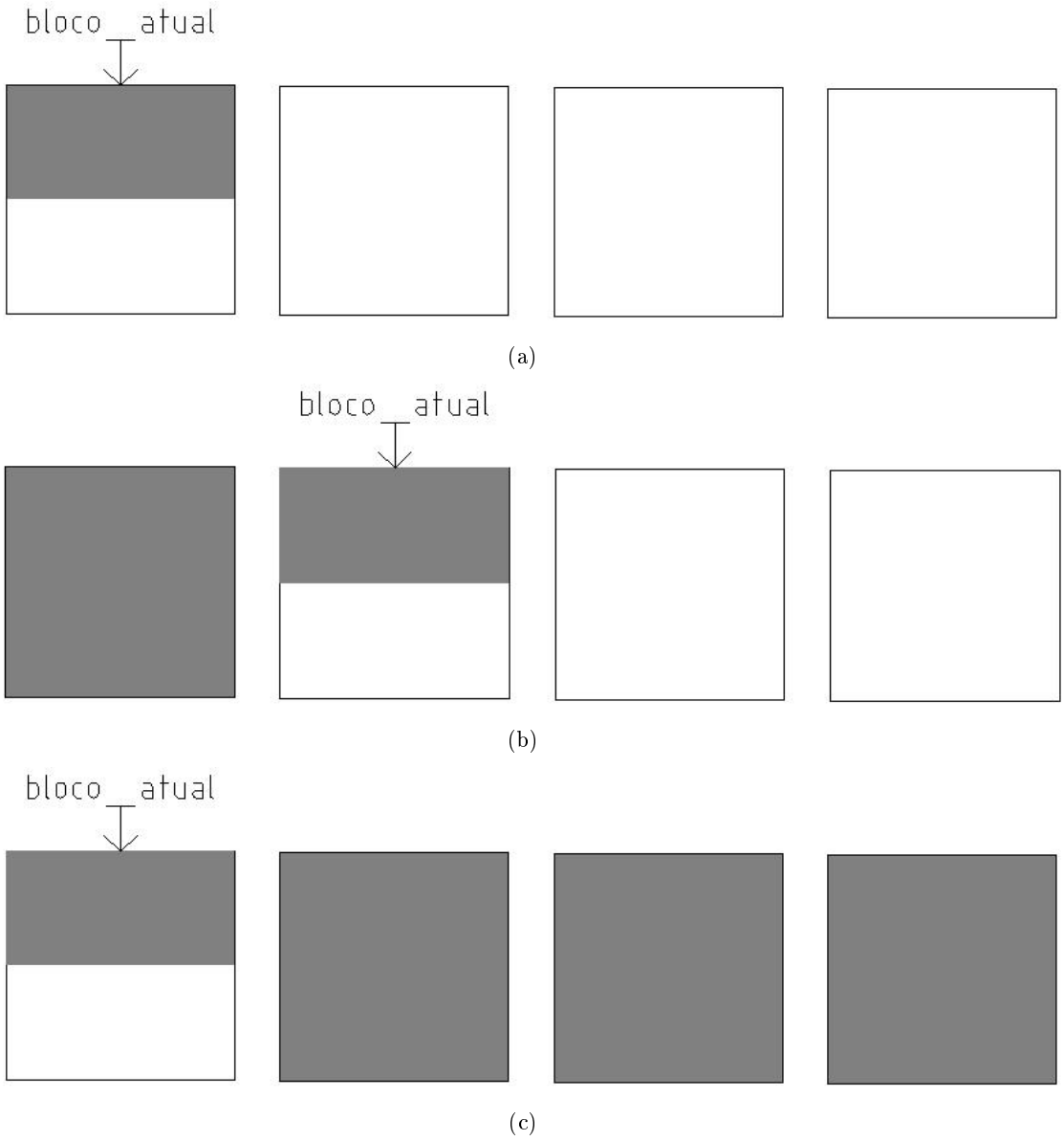


Figura 3.8: Funcionamento dos blocos de memória.

três imagens válidas deve ser calculada a cada ciclo de *clock*. Como os valor absoluto não são de interesse durante os cálculos ficou decidido que a soma das três imagens seria suficiente, o que evita uma divisão desnecessária.

Durante a implementação foi constatado que fazer a leitura de diversos pixels em diferentes endereços de memória podeira consumir muito tempo e talvez se tornar proibitivo caso se deseje uma quantidade maior de imagens no filtro de média. A solução para este problema foi utilizar uma espécie de média móvel nos sinais que são utilizados para os cálculos. Em vez de se fazer a leitura de dois endereços de memória mais recentes para somá-los com o novo pixel recebido ($dado\_escrita$), foi feita apenas a leitura do mais antigo par ser subtraído da soma. Por exemplo: quando era feito o recebimento de um pixel da linha *linha*, coluna *coluna* o valor do elemento $somaHor(linha)$ era atualizado da seguinte forma:

$$somaHor(linha) = somaHor(linha) + dado\_escrita - q(bloco\_atual - 3, linha, coluna) \quad (3.1)$$

Onde q é o byte armazenado no endereço de memória dado por $bloco\_atual$ concatenado com $linha \times numcols + coluna$, sendo $numcols$ a largura da imagem.

3.4.5 Cálculo de topo e base do arame

Os dados de topo e base do arame foram obtidos com a mesma lógica criada para Matlab e de forma sequencial. Foi utilizado um processo que tem in_janela e in_clock como *sensitivity list*. O sinal in_janela foi utilizado para sincronização de final de imagem e o in_clock era o *clock* de entrada da câmera e define o fim e o começo da transmissão de cada pixel.

Foram criados os mesmos vetores $somaHor$ e o $derivadaHor$ ambos do tipo $vetorHor$. Como citado em 3.4.4 os elementos de $somaHor$ foram calculados a cada pixel recebido, mas os valores de $derivadaHor$ apenas foram calculados ao final da leitura de cada linha.

Ao mesmo tempo foram definidos os valores $posArameTopo$ e $posArameBase$ com simples testes ao final de cada linha. Ao final da imagem estes valores já estão devidamente definidos.

3.4.6 Laterais do Eletrodo

As laterais do eletrodo são definidas em dois estágios diferentes. Primeiro os pontos prováveis destas laterais são encontrados, depois é calculada a equação de reta que descreve essas laterais. O cálculo desta equação de reta é descrita em 3.4.6.1

3.4.6.1 Regressão linear

Esta regressão é feita de forma simplificada para poupar elementos utilizados na placa e diminuir o tempo de cálculo. Para tanto foi decidido que seriam utilizados apenas uma quantidade limitada de pontos por reta no momento do desenvolvimento. Essa quantidade foi a princípio definida como $qtdPontosArame = 16$ para a simulação com imagens de 60x60 pixels. Este valor foi escolhido por ser uma potência de 2 e também por ter sido testado com o programa em Matlab, com resultados satisfatórios.

Esta forma de se calcular permite que seja utilizado o método descrito em 2.6.1. As constantes utilizadas para as multiplicações pelos somatórios são previamente calculadas no momento do desenvolvimento do código de acordo com as equações 2.44, e 2.45.

Como os valores destas constantes são quase sempre muito pequenos (da ordem de 10^{-4} para 16 pontos), esses valores foram multiplicados por potências de 2 para serem utilizados na multiplicação pelos somatórios. Essas potências de 2 equivalem a *shifts* para a esquerda nas constantes. O resultado de cada uma passa posteriormente por um *shift* para a direita equivalentes à potência pela qual foram multiplicadas as constantes.

Descrição	Valor	Inteiro usado	Utilizada com fator
constRegressao1	0,0029	$0,0029 * 2^{26}$	a
constRegressao2	0,0221	$0,0221 * 2^{23}$	a,b
constRegressao3	0,2279	$0,2279 * 2^{20}$	b

Tabela 3.1: Constantes utilizadas em regressão com 16 pontos

Os valores calculados para as contantes para 16 pontos, com seus proporcionais utilizados na no código VHDL, e as quantidades de *shifts* são descritos na tabela 3.1. É importante notar que os valores inteiros foram todos calculados de forma a se obter a máxima precisão com 18 bits, pois os multiplicadores da placa utilizada são de 18x18 bits.

Os somatórios são calculados com os valores dos vetores $inicioArame$ e $fimArame$ entre os índices $posArameTopo$ e $posArameBase$. Assim como no algoritmo para Matlab, aqui também foi utilizado um afastamento entre o topo e a base para se evitar os cantos onde podem haver inconsistências. Este foi definido inicialmente como $afastamento = 2$ para a imagem de 60x60 pixels.

Para a definição dos índices nos vetores $inicioArame$ e $fimArame$ que seriam utilizados para a leitura dos valores foi feito um processo semelhante ao utilizado com as multiplicações de constantes acima. Isto foi necessário pois o para se utilizar uma quantidade limitada de pontos distribuídos o mais homogeneamente possível entre dois limites, sem a utilização de pontos flutuantes. Para melhor demostrar o problema: Imagine que se deseja 16 índices inteiros distribuídos entre os valores 14 e 42 de forma mais homogênea possível com ferramentas que não permitem o uso de números fracionários.

A solução encontra-se em utilizar valores inteiros e *shifts*. Neste caso específico as variáveis de entrada, $posArameTopo$, $posArameBase$, $qtdPontosArame$ e $afastamento$ são todas inteiras. O que se necessita para a obtenção do n-ésimo índice pode ser melhor entendido com a equação 3.2.

$$indice = i + (n-1) \times \frac{intervalo}{qtdPontosArame} \tag{3.2}$$

Onde i é o primeiro índice da série, n se refere ao n-ésimo ponto e $intervalo/qtdPontosArame$ é a diferença entre dois pontos subsequentes que se deseja na série, sendo $intervalo$ calculado acordo com a equação 3.3.

$$intervalo = posArameBase - posArameTopo - afastamento - afastamento \tag{3.3}$$

O valor de $intervalo$ precisa ser sempre inteiro para se obter um índice válido no vetor. Caso a divisão ocorra antes da multiplicação, o valor truncado perde precisão. Por este motivo a equação 3.2 foi adaptada na equação 3.4 para manter o melhor inteiro possível.

$$indice = i + [(n-1) \times intervalo] >> log_2(qtdePontosArame) \tag{3.4}$$

O operador $>>$ denota a operação de *shift*. Neste caso, serão feitos 4 *shifts*, o que equivale a uma divisão por 16. O resultado final pode ser normalmente somado e multiplicado por ferramentas comuns da FPGA.

Os coeficientes obtidos com o cálculo simplificado devem ser corrigidos pois até este ponto as observações são consideradas como tendo avanço unitário no eixo X, e não aquelas calculadas por $intervalo/qtdPontosArame$.

3.5 Simulações

A maior parte do projeto foi desenvolvida sem a utilização de placas FPGA e câmeras reais, portanto foi essencial o uso de simulações. Cada criação ou alteração de código foi testada com auxílio das simulações a fim de evitar excessivo retrabalho em correções de erros.

O desenvolvimento em Matlab, utilizou arquivos de imagens como entrada, pois o *software* tem diversas funções que facilitam a leitura de arquivos. A implementação em FPGA, por sua vez é bem mais complexa e utilizou o *software* externo Modelsim com arquivos de texto como entrada.

3.5.1 Simulações em Matlab

A simulação em Matlab ocorre de forma bastante simples: basta rodar o programa gerado. Foi possível aproveitar das facilidades do software para imprimir as imagens de entrada processadas, ou seja, após a passagem dos filtros e com os valores calculados representados graficamente.

Os programas desenvolvidos em Matlab utilizaram de imagens previamente obtidas em outro trabalho [29]. A câmera utilizada foi uma DALSA citada em 2.5 e foi montada conforme a figura 3.7.

O experimento utilizou um eletrodo de $r_{eletrodo} = 1mm$ de diâmetro, com $standoff = 15mm$. Parte do eletrodo fica coberta pelo bocal na captura das imagens, mas sabe-se que a câmera estava posicionada a $\alpha = 30°$ em relação ao plano horizontal. Considerando o bocal da tocha de soldagem com $r_{bocal} = 10mm$ de diâmetro, a altura da h parte visível do eletrodo é dada por:

$$h = standoff - (r_{bocal} - r_{eletrodo}) \cdot sen(\alpha) \tag{3.5}$$

$$h = 15 - (10 - 0,5) \cdot sen(30) = 10,25mm \tag{3.6}$$

Os valores de $r_{eletrodo}$ e h foram utilizados para calcular a distorção de perspectiva à partir das próprias imagens. Apesar de essa não ser a opção desejada (o ideal seria uma calibração em separado) é suficiente para o propósito de simulação.

OU OU OU OU OU OU OU OU OU OU OU OU OU OU OU OU OU OU OU OU

Esta câmera foi posicionada a 30° em relação ao plano horizontal, o plano da poça de soldagem. Com esse conhecimento foi gerada a matriz de transformação correspondente, 3.7. Também se assumiu que a câmera estava perfeitamente alinhada no plano formado entre a direção de avanço da solda e a direção do eletrodo.

DESCREVER QUAL EIXO É QUAL

$$\begin{bmatrix} a & b & c \\ a & b & c \\ a & b & c \end{bmatrix} \tag{3.7}$$

3.5.2 Simulações em FPGA

Inicialmente, o algoritmo foi desenvolvido como se os dados de entrada (pixels) e clock viessem de alguma forma já desserializados para a placa. Ou seja, havia apenas a entrada de um clock e um byte, além dos outros sinais de configuração. As entidades de processamento foram todas feitas de forma a receber informações desta forma genérica, o que se mostrou eficiente após serem criados os blocos descritos em 3.4.1.

Dessa forma foram utilizadas imagens previamente capturadas no trabalho de [29], da mesma forma que as simulações em Matlab. Porém, as imagens foram transformadas em blocos de texto para poderem ser usadas em um arquivo *testbench*.

3.5.2.1 Conversão de Imagens para Simulação

Este processo foi feito com a ajuda de um programa bastante simples disponível em distribuições Linux, chamado "od". Este programa traduz qualquer tipo de arquivo, inclusive imagens, para códigos hexadecimais. Um pequeno script foi criado para facilitar a utilização do programa. O resultado é um arquivo com palavras hexadecimais separadas por espaços, com a mesma quantidade de linhas e colunas que a imagem fornecida.

O texto deste arquivo é então inserido no arquivo de *testbench* a ser carregado pelo Modelsim. Abaixo segue um trecho de imagem em formato de texto hexadecimal.

03	03	03	07	07	07	31	31	31
04	04	04	1c	1c	1c	5b	5b	5b
0b	0b	0b	49	49	49	7d	7d	7d
26	26	26	7f	7f	7f	89	89	89
47	47	47	93	93	93	6c	6c	6c

Note que em cada linha os valores são repetidos três a três. Isso acontece pois as imagens estavam armazenadas em arquivos do tipo *bitmab* de três cores RGB, apesar de serem imagens em escala de cinza. Ou seja, cada um dos três valores repetidos representa uma cor de vermelho,

verde ou azul. Apenas retirando dois de cada três desses valores tem-se o desejado. Esta etapa é facilmente feita com o uso de uma expressão regular, ou *Regex*.

O resultado final dentro do *testbench* é uma matriz com os valores desejados. Esses valores podem ser lidos por indexação de linha e coluna. Abaixo, segue o exemplo do começo, meio e final de uma dessas matrizes:

```
CONSTANT imagem_teste0 : MatrizImagem :=

(( X"03",  X"02",  X"03", ...  X"03",  X"02",  X"03", ...  X"02",  X"03",  X"03"),
 ( X"03",  X"02",  X"03", ...  X"da",  X"f5",  X"fe", ...  X"09",  X"03",  X"03"),
 ( X"03",  X"02",  X"03", ...  X"03",  X"02",  X"03", ...  X"02",  X"03",  X"03"));
```

3.5.2.2 Sincronização

A primeira versão do *testbench* foi feita de forma a fornecer os pixels sequencialmente, sem um sinal de início de imagem ou de linhas. Isso obrigava os blocos que funcionavam em paralelo a fazer uma contagem para saber quando uma imagem ou linhas acabavam ou iniciavam. Cada entidade possuía um valor de *linha* e um de *coluna* para fazer essa contagem. Essa redundância gerava um custo desnecessário de recursos.

Logo chegou-se a conclusão que essa abordagem se tornaria um problema no caso real onde se deseja processar durante a captura de imagens. Então foram implementados os sinais $FVAL$ e $LVAL$ do padrão *Camera Link* para simular a sincronia da captura e também evitar problemas futuros.

3.6 Identificação do ângulo da câmera e distorção de perspectiva

Para implementação em uma FPGA, devem-se separar blocos da imagem para serem processados paralelamente. Dessa forma uma imagem pode ser separada em setores para a aplicação do algoritmo.

Capítulo 4

Resultados Experimentais

Resumo opcional.

4.1 Introdução

Na introdução deverá ser feita uma descrição geral dos experimentos realizados.

Para cada experimentação apresentada, descrever as condições de experimentação (e.g., instrumentos, ligações específicas, configurações dos programas), os resultados obtidos na forma de tabelas, curvas ou gráficos. Por fim, tão importante quando ter os resultados é a análise que se faz deles. Quando os resultados obtidos não forem como esperados, procurar justificar e/ou propor alteração na teoria de forma a justificá-los.

4.2 Avaliação do algoritmo de resolução da equação algébrica de Riccati

O algoritmo proposto para solução da equação algébrica de Riccati foi avaliado em diferentes máquinas. Os tempos de execução são mostrados na Tabela 4.1. Nesta tabela, os algoritmos propostos receberam a denominação CH para Chandrasekhar e $CH + LYAP$ para Chandrasekhar com Lyapunov. As implementações foram feitas em linguagem *script* MATLAB.

Observa-se que o algoritmo $CH + LYAP$ apresenta tempos de execução superiores com relação ao algoritmo CH. Entretanto, era esperado que o algoritmo CH fosse mais rápido. Este

Tabela 4.1: Tempos de execução em segundos para diferentes máquinas

Algoritmo	**Laptop 1.8 GHz**	**Desktop PIII 850 MHz**	**Desktop MMX 233**	**Laptop 600 MHz**
Matlab ARE	649,96	1.857,5	7.450,5	9.063,9
CH	259,44	606,4	2.436,5	2.588,5
$CH + LYAP$	357,86	952,9	3.689,2	3.875,0

resultado se justifica pelo fato de o algoritmo CH fazer uso de funções embutidas do MATLAB. Já o aloritmo $CH + LYAP$ faz uso também de funções *script* externas, aumentando bastante seu tempo computacional.

Capítulo 5

Conclusões

Este capítulo é em geral formado por: um breve resumo do que foi apresentado, conclusões mais pertinentes e propostas de trabalhos futuros.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] MODENESI, P. J.; MARQUES, P. V.; SANTOS, D. B. *IntroduÃğÃčo Ãă Metalurgia da Soldagem*. [S.l.: s.n.], 2012.

[2] MACHADO, I. G. *Soldagem e tÃľcnicas conexas*. [S.l.: s.n.], 2007.

[3] CRIMINISI, A.; REID, I.; ZISSERMAN, A. A plane measuring device. *British Machine Vision Conferencel*, July 1997.

[4] ALTERA DE2-115 Development and Education Board. Disponível em: <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=502&PartNo=1>.

[5] IN the Beginning. Disponível em: <https://www.altera.com/solutions/technology/system-design/articles/_2013/in-the-beginning.html>.

[6] WOLF, W. *FPGA-Based System Design*. [S.l.: s.n.].

[7] VENTURA, R. MudanÃğas no perfil do consumo no brasil. August 2010.

[8] BROERING, C. E. *Desenvolvimento de Sistemas para a AutomaÃğÃčo da Soldagem e corte TÃľrmico*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2005.

[9] TRANDEL, G. A. The bias due to omitting quality when estimating automobile demand. *The Review of Economics and Statistics*, August 1991.

[10] MARQUES, P. V.; MODENESI, P. J.; BRACARENSE, A. Q. *Soldagem: Fundamentos e Tecnologia*. [S.l.]: Editora UFMG, 2005.

[11] ADOLFSSON, S. On-line quality monitoring in short-circuit gas metal arc welding. *Welding Journal*, December 1997.

[12] BALFOUR, C.; SMITH, J. S.; AL-SHAMMA'A, A. I. A novel edge feature correlation algorithm for real-time computer vision-based molten weld pool measurements.

[13] FILHO, O. M.; NETO, H. V. *Processamento Digital de Imagens*. Rio de Janeiro: Brasport, 1999.

[14] GONZALEZ, R. C.; WOODS, R. E. *Processamento de Imagens Digitais*. 1Âł. ed. [S.l.]: Edgard BlÃijcher LTDA, 2000.

[15] MUNDY JOSEPH. ZISSERMAN, A. Geometric invariance in computer vision. In: _____. [S.l.]: MIT Press, 1992. cap. 23.

[16] HARRIS, M. S. C. A combined corner and edge detector. *Plessey Research Roke Manor*, 1998.

[17] MORAVEC, H. P. Towards automatic visual obstacle avoidance. *Proc. 5th International Joint Conference on Artificial Intelligence*, p. 584, 1977.

[18] JR., G.; J., L. *Machine Vision and Digital Image Processing Fundamentals*. [S.l.]: Prentie Hall, 1990.

[19] HISTORY of FPGAs. Disponível em: <https://web.archive.org/web/20070412183416/http://filebox.vt.edu/

[20] David W. Page e LuVerne R. Peterson. *Re-programmable PLA*. abr. 2 1985. US Patent 4,508,977. Disponível em: <http://www.google.com/patents/US4508977>.

[21] XILINX (Ed.). *Xcell*.

[22] KINNUNEN, M. *Examining the limits of Moore's law*. Bachelor of Engineering.

[23] MOORE, G. E. Cramming more components onto integrated circuits.

[24] D'AMORE, R. *VHDL: descriçao e síntese de circuitos digitais*. LTC, 2005. ISBN 9788521614524. Disponível em: <https://books.google.com.br/books?id=6I26AAAACAAJ>.

[25] IEEE Standard VHDL Language Reference Manual. *IEEE Std 1076-1987*, p. 01–, 1988.

[26] IEEE Standard VHDL Language Reference Manual. *ANSI/IEEE Std 1076-1993*, p. i–, 1994.

[27] IEEE Standard VHDL Synthesis Packages. *IEEE Std 1076.3-1997*, p. 1–52, June 1997.

[28] VHDL for Simulation and Synthesis. [S.l.].

[29] FRANCO, L. D. N. *SincronizaÃğÃčo, captura e anÃąlise de imagens da poÃğa de soldagem no processo GMAW convencional, no modo de transferÃłncia metÃąlica por curto circuito*. Dissertação (Mestrado) — Universidade de BrasÃŋlia, 2007.

[30] SPECIFICATIONS of The Camera Link Interface Standard for Digital Cameras and Frame Grabbers. [S.l.].

[31] MICROTRONIX Camera Link IP Core. [S.l.].

[32] 1M28, 1M75, and 1M150 UserâĂŹs Manual. [S.l.].

[33] KREYSIG, E. *Advanced Engineering Mathematics*. [S.l.: s.n.].

[34] ROYER, P. et al. Area-efficient linear regression architecture for real-time signal processing on fpgas.

ANEXOS

I. DIAGRAMAS ESQUEMÁTICOS

II. DESCRIÇÃO DO CONTEÚDO DO CD