UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

JILMAR TEIXEIRA DE ALMEIDA JUNIOR

DESENVOLVIMENTO DE UM MÓDULO DE RETIFICAÇÃO DE IMAGENS PARA VISÃO ESTÉREO

RECIFE

2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

JILMAR TEIXEIRA DE ALMEIDA JUNIOR

DESENVOLVIMENTO DE UM MÓDULO DE RETIFICAÇÃO DE IMAGENS PARA VISÃO ESTÉREO

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pela professora Edna Natividade da Silva Barros.

RECIFE

2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

JILMAR TEIXEIRA DE ALMEIDA JUNIOR

DESENVOLVIMENTO DE UM MÓDULO DE RETIFICAÇÃO DE IMAGENS PARA VISÃO ESTÉREO

Monografia submetida ao corpo docente da Universidade Federal de Pernambuco, defendida e aprovada em 04 de Julho de 2018.

Banca Examinadora:

Orientador

Edna Natividade da Silva Barros

Doutor

Examinador

Manoel Eusebio de Lima

Doutor

Dedico esse trabalho à minhas duas famílias:
Os Almeidas e os Felicianos. O apoio deles
deixou o caminho até aqui muito mais fácil e
prazeroso. Amo vocês.

AGRADECIMENTOS

Quem me conhece sabe que não sou muito religioso, mas ainda assim gostaria de agradecer a Deus. Durante toda a jornada reconheci várias vezes Sua ajuda e tenho certeza que isso me levou às minhas vitorias.

Agradeço também à minha mãe por estar sempre comigo, mesmo longe. Seu apoio me tranquilizava quando estava preocupado e me motivava quando estava desmotivado.

Para os Almeidas nós sabemos das dificuldades que passamos e vocês estiveram lá para me ajudar com suas palavras e com carinho de uma família que mesmo separados estão unidos. Para os Felicianos que me acolheram e fizeram com que eu sentisse que havia um lar, independente de onde estivesse. Para essas duas famílias, não tenho nada menos que minha eterna gratidão e puro amor.

A professora Edna Natividade por me acolher nesse trabalho e a Lucas Cambuim por terem me ajudado em todas as etapas com muita paciência e gentileza.

E por fim, a meus amigos, meu muito obrigado. Vocês fazem meus dias mais felizes. Aline Policarpo sempre disposta a me ouvir, sendo para contar minhas felicidades ou reclamações, Lucas Virgílio com quem posso contar em todas as situações e meus amigos dessa longa jornada de engenharia Luã Lazaro, Rafael Dias, Maria Luiza, Larissa Camila e Vinicius Sanguinete. Espero que possamos contar com o apoio um do outro como contamos nesses anos.

“Transportai um punhado de terra todos os dias,
e fareis uma montanha.”
Confucio

RESUMO

O processo de retificação assegura que imagens vindas de duas câmeras diferentes de um sistema estéreo estarão paralelas, assim é possível achar os pontos referentes a um ponto 3D na mesma linha em duas imagens diferentes. No trabalho desenvolvido em seu mestrado, o aluno Lucas Fernando da Silva Cambuim desenvolveu duas implementações de um sistema de correspondência para um sistema de visão estéreo, ambos com o propósito final de computar de um mapa de disparidade. Um dos sistemas foi implementado em C++ e o outro em FPGA na placa DE2i-150 da Intel. Esses sistemas contavam com o uso da biblioteca OpenCV, porém como não há uma implementação de retificação para FPGA na biblioteca, a placa utiliza tinha que estar sempre conectada a um computador para receber as imagens retificadas como entrada. O presente trabalho propõe a implementação de um algoritmo de retificação de imagens em C++ que possa ser usado como referência para uma posterior implementação em FPGA. Adicionalmente, foi implementada uma versão do algoritmo em uma linguagem de descrição de hardware, SystemVerilog, no nível RTL, visando facilitar a futura prototipação do sistema em FPGA. Os resultados desse trabalho foram comparados aos resultados obtidos no trabalho em C++ desenvolvido no mestrado do aluno Lucas Fernando da Silva Cambuim e obtiveram uma taxa de erro de 0.2197265%. Os resultados da simulação funcional também foram comparados ao mesmo trabalho base e uma taxa de erro de 0.0973307% foi obtida. Conclui-se que devido a baixa taxa de erro, o sistema desenvolvido nesse trabalho pode ser integrado ao sistema de correspondência sem grandes perdas na geração do mapa de disparidade.

Palavras-chave: Visão Computacional, Retificação de Imagens, FPGA.

ABSTRACT

The rectification assures that images coming from two different cameras of a stereo system will be parallel. Thus, it is possible to find in two different images, points related to a 3D point on the same line. During his masters' work, the student Lucas Fernando da Silva Cambuim developed two systems of correspondence to a system of stereo vision. Both aimed the creation of a disparity map. One of the systems was implemented using C++; the other one through Intel's DE2i-150 FPGA board. These systems used OpenCV library; however, because there is not any implementation of rectification for FPGA in the library, the board in use had to be uninterruptedly connected to a computer in order to receive the rectified images as input. The present work proposes the implementation of a algorithm of rectification of images in C ++ that can be used as reference for a later implementation in FPGA. In addition, a version of the algorithm was implemented in a hardware description language, SystemVerilog, at the RTL level, to facilitate the future prototyping of the FPGA system. The results taken through this work were compared to the ones taken in the work with C++, developed during Lucas Fernando da Silva Cambuim's masters, and the error rate was 0.2197265%. The results of the functional simulation were also compared to the same work, obtaining the error rate of 0.0973307%. As a conclusion, it is possible to state that due to the low error rate, the system presented in this work can be integrated to the system of correspondence without big losses in the generation of the disparity map.

Keywords: Computer Vision, Image Rectification, FPGA.

Sumário

Lista de Figuras

LISTA DE TABELAS

TABELA DE SIGLAS

Sigla	Significado	Página
3D	Três dimensões	14
FPGA	Field Programmable Gate Array	15
RTL	Register-Transfer Level	16
PPM	Perspective Projection Matrix	19
2D	Duas dimensões	19
LUT	Look up Table	26

1. Introdução

1.1. Motivação

A área de visão computacional é essencial para sistemas embarcados modernos. Os sistemas embarcados estão usando reconhecimento de imagens e de padrões para aplicações como sistemas de navegação, reconhecimento de objetos, localização [1], etc.

Uma das técnicas da área de visão computacional é a visão estéreo. Visão estéreo é baseada na visão humana e em grande parte dos animais, sendo composta por duas câmeras que capturam imagens de um mesmo objeto para encontrar informações 3D de uma cena [2]. Sistemas robóticos usam visão estéreo para detectar a profundidade dos objetos em cena sem a necessidade da instalação de sensores adicionais.

Um dos desafios da técnica de visão estéreo é a correspondência de dois pontos da mesma cena em diferentes imagens, pois é a partir da distância desses pontos que a maioria das informações é obtida.

Como já foi dito, a profundidade de um objeto em cena pode ser obtida utilizando um sistema estéreo, mais especificamente através da técnica de correspondência estéreo (*Stereo Matching)*. Nessa técnica, os pixels correspondentes são buscados horizontalmente nas imagens das duas câmeras do sistema estéreo e a distância entre esses pixels (disparidade) é calculada [7]. Para fazer essa busca horizontal, as imagens precisam estar retificadas. A imagem gerada que contém os valores de disparidade para cada pixel é chamada de mapa de disparidade.

A retificação de imagens (Figura 1) é um método que garante que as linhas de duas imagens capturadas em um sistema estéreo estarão paralelas entre si, assim reduzindo a busca de um ponto à uma busca em uma linha.

A retificação é uma etapa importante e pode ser utilizada por vários sistemas. Um exemplo de sistema que utiliza a retificação é o *Google car* que usa dados de imagem de várias câmeras com imagens retificadas para calcular a profundidade dos pontos capturados e utiliza esse dados para fazer a reconstrução 3D dos locais das imagens capturadas.

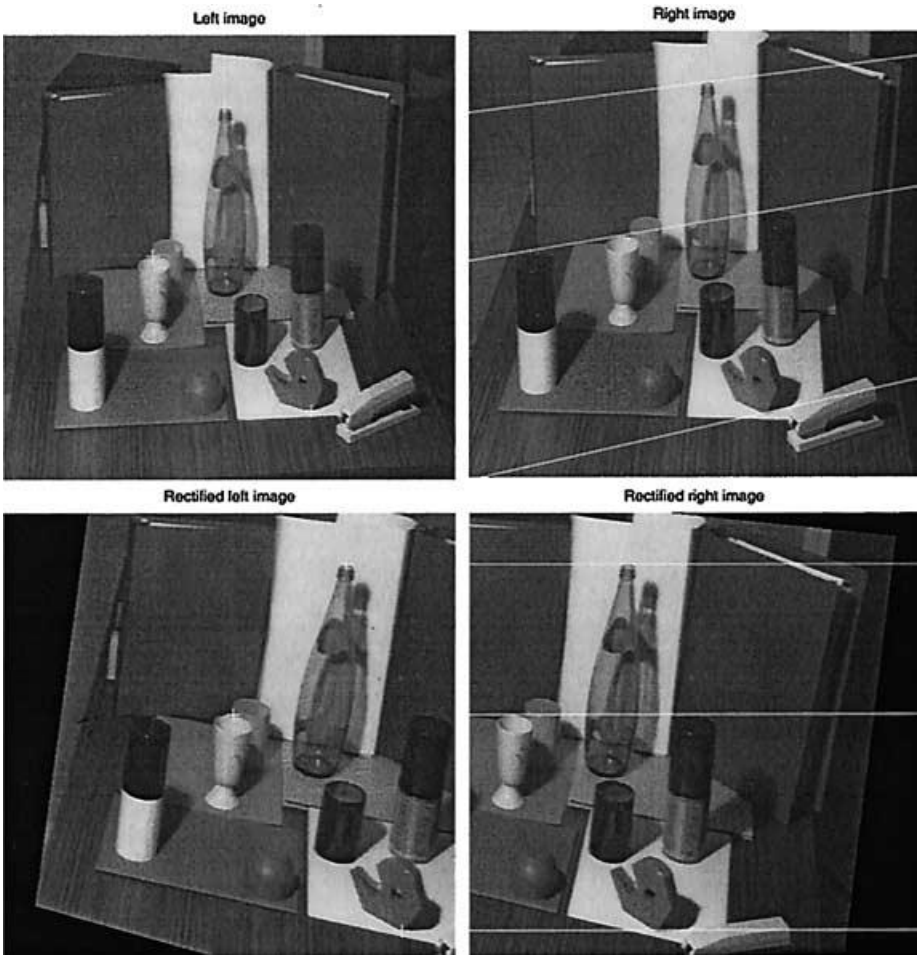

Figura 1: Exemplo de retificação de imagens de um sistema estéreo [1].

1.2. Sistema de Correspondência em [7]

No seu trabalho de mestrado, o aluno Lucas Fernando da Silva Cambuim criou dois sistemas de correspondência para o calcular o mapa de disparidade. Um desses sistemas foi implementado em C++, criado para computadores desktops e o outro implementado em SystemVerilog RTL e prototipado em FPGA.

A retificação de imagens garante que as imagens em uma linha podem ser encontradas na mesma linha em outras imagens retificadas e o trabalho de Lucas em [7] utiliza essa técnica para reduzir a complexidade do sistema de correspondência, fazendo com que a busca por um pixel numa imagem retificada se reduza a uma busca em uma linha em uma outra imagem retificada.

O trabalho proposto por Lucas utiliza a biblioteca OpenCV para gerar imagens retificadas. Isso significa que para garantir o funcionamento na placa FPGA, essa placa deve sempre estar ligada a um computador para receber as imagens retificadas geradas pelo OpenCV.

1.3. Objetivos

O objetivo desse trabalho é implementar dois sistemas de retificação de imagens. O primeiro sistema será implementado em C++ e o segundo será implementado em System Verilog RTL. Os sistemas implementados nesse trabalho serão comparados com os sistemas desenvolvidos no trabalho em [7] e tem como objetivo principal analisar o impacto na precisão entre as abordagens e analisar se esse sistema pode ser adicionado ao trabalho em [7] para criar um sistema completo em FPGA sem que haja grande perda de precisão no cálculo do mapa de disparidade.

1.4. Organização do Trabalho

Esse trabalho está organizado como segue:

- Seção 2: Conceitos básicos.
- Seção 3: Trabalhos Relacionados.
- Seção 4: Implementações para o algoritmo de retificação de imagens.
- Seção 5: Resultados e análises.
- Seção 6: Conclusões e trabalhos futuros.
- Seção 7: Referências bibliográficas.

2. Conceitos Básicos

Em [5], visão computacional é definida como a ciência que desenvolve base matemática e algorítmica pelas quais informações de alto nível de ambiente são extraídas para análise e processamento com o uso de sensores óticos, técnicas de processamento digital de imagens e reconhecimento de padrões.

O principal objetivo da visão computacional é prover para os computadores a capacidade de percepção humana [4]. Como exemplo disso, pode-se citar a capacidade de computadores de reconhecer objetos, contextos em cenas, emoções humanas e profundidades de objetos em fotos.

Visão estéreo é uma técnica de visão computacional baseada na visão de grande parte dos animais, incluindo humanos. Essa técnica utiliza duas câmeras para capturar uma mesma cena, tendo como principal objetivo obter características de duas imagens simultâneas sem a necessidade de outros sensores adicionais.

A geometria epipolar está diretamente ligada a geometria de um sistema estéreo e é geralmente utilizada para *Stereo Matching* [2]. *Stereo matching* é o processo de encontrar as representações de um ponto no mundo real em duas imagens que foram capturas no mesmo sistema estéreo.

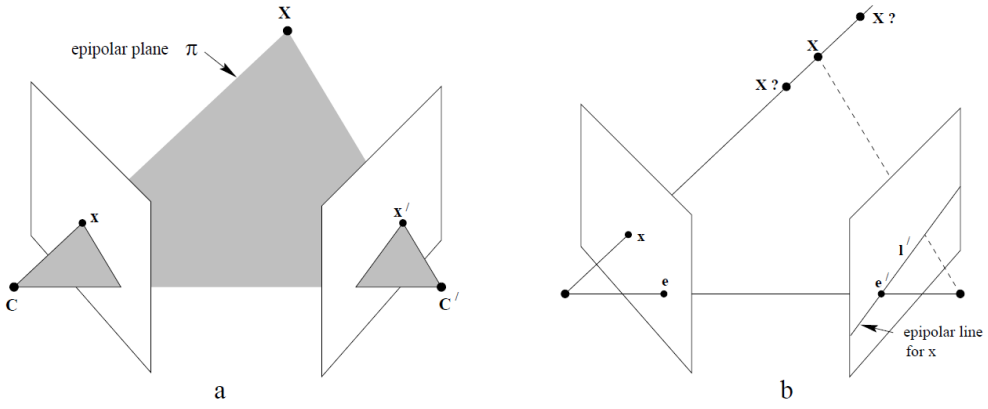

Figura 2: Representação da Geometria epipolar em [2].

Na Figura 2 (a), as câmeras são indicadas pelos seus centros C e C', já os pontos x e x' representam respectivamente o ponto X do mundo real projetado nos planos da imagem da câmera da esquerda e da direita. Sobre a geometria epipolar, deve-se destacar os seguintes pontos importantes:

- Plano epipolar: Na figura 2 (a) é possível ver o plano epipolar π que liga os centros das câmeras e o ponto X do mundo real.
- Linha epipolar: Na figura 2 (b), a linha epipolar l' no plano da imagem da câmera da direita contém o ponto X e outros pontos do mundo real, representados por na figura por *X?*. As linhas epipolares podem ser encontradas na intersecção do plano da imagem com o plano epipolar. Um exemplo de linhas epipolares em imagens pode ser visto na câmera direita da figura 1 antes e depois da retificação das imagens.
- Ponto epipolar ou epipólo: Na figura 2 (b) os pontos epipolares *e* e *e'* respectivamente da câmera esquerda e câmera direita. O ponto epipolar pode ser encontrado na intersecção do plano da imagem com a linha que liga os centros das câmeras.

A retificação de imagens é uma técnica inerente da geometria de duas ou mais câmeras. O objetivo da retificação de imagens é transformar imagens da mesma cena capturadas simultaneamente de modo que as imagens resultantes tenham suas linhas epipolares paralelas entre si e que todos os pontos encontrados em uma imagem possam ser encontrados na mesma linha nas outras imagens [1].

Existem dois tipos de retificação: a retificação horizontal e a retificação vertical. Na retificação horizontal todos os pontos de uma imagem podem ser encontrados numa mesma linha horizontal nas outras imagens retificadas. Já na retificação vertical todos os pontos de uma imagem podem ser encontrados numa mesma linha vertical nas outras imagens retificadas. A retificação horizontal é a mais comum e poder ser encontrada nos artigos referenciados por esse trabalho e é também o tipo de retificação usado nesse trabalho, assim todas as vezes que a palavra retificação for citada, deve ser considerada uma retificação horizontal.

Para se obter um melhor entendimento da retificação de imagens, é necessário conhecer os parâmetros das câmeras e a projeção de imagens, como será descrito na seção 2.1. O fluxo de uma retificação se inicia obtendo parâmetros das câmeras do sistema estéreo com a calibração, como será descrito na seção 2.2. Depois da calibração é necessário calcular as matrizes de rotação e projeção do sistema estéreo para executar a retificação, como será definido na seção 2.3.

2.1. Projeção da Imagem e Parâmetros da Câmera

Uma câmera pode ser representada algebricamente como uma matriz, chamada de matriz de projeção. A matriz de projeção de perspectiva (PPM - Perspective Projection Matrix) é uma matriz 3x4 que descreve o mapeamento de pontos no mundo real 3D para o plano das imagens em 2D.

A matriz de projeção de perspectiva, ou simplesmente matriz de projeção *P*, é formada de duas outras matrizes de parâmetros, a matriz de parâmetros intrínsecos e a matriz de paramentos extrínsecos.

Os parâmetros intrínsecos de uma câmera contêm informações associadas às características inerentes à câmera. A representação de uma matriz intrínseca é criada levando em conta centro ótico *C* da câmera e o plano da imagem gerada pela câmera. Esses pontos podem ser vistos na figura 3.

A distância do ponto C até o plano das imagens é chamada de distância focal f. As distancias focais f_x e f_y são chamadas respectivamente de distância focal horizontal e distância focal vertical e podem ser calculadas usando a distância focal f e a altura (m_x) e largura (m_y) dos pixels da imagem em milímetros, como mostrado nas equações *1* e *2*.

$$f_x = f.m_x \tag{1}$$

$$f_y = f.m_y \tag{2}$$

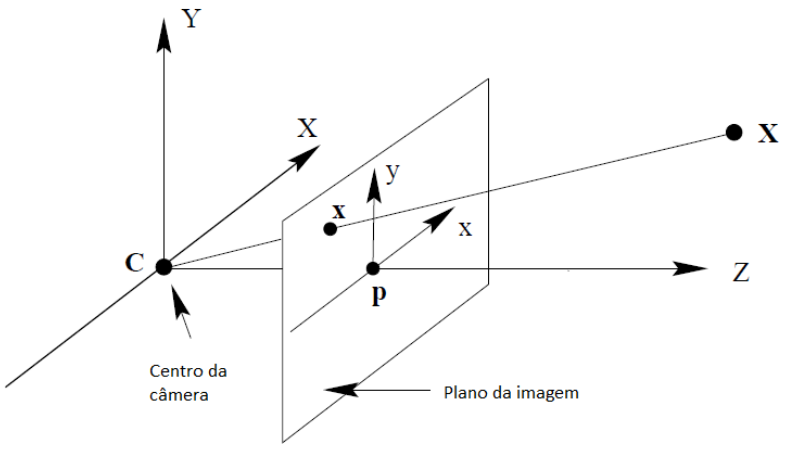

Figura 3: Representação da captura de imagem do ponto X. Adaptado de [2].

Por fim, o ponto (p_x, p_y) é chamado de ponto principal de uma imagem. O ponto principal pode ser visto na figura 2 (representado pelo ponto *p*) e é encontrado na intersecção da reta com o eixo ótico. O eixo ótico por sua vez é a reta que contém o ponto C e é ortogonal ao plano da imagem. Na figura 3, o eixo ótico coincide com o eixo Z.

A matriz dos parâmetros intrínsecos de uma câmera *c* é representada pela matriz K_c abaixo, onde fx e fy são as distancias focais apresentadas nas equações *1* e *2* e p_x e p_y são as coordenadas do ponto principal descrito no parágrafo anterior.

$$K_c = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

A matriz dos parâmetros extrínsecos *E* é uma matriz 3x4 que contém a posição e orientação da câmera. A orientação da câmera é representada por uma matriz 3x3 e é chamada de matriz de rotação R_c. A posição da câmera é representada por uma matriz 3x1, chamada de matriz de translação T_c. Essa matriz é a distância do centro da câmera até o sistema de coordenadas adotado como principal. A matriz de parâmetros extrínsecos da câmera *c* é representada abaixo como E_c.

$$E_c = [R_c|T_c]$$

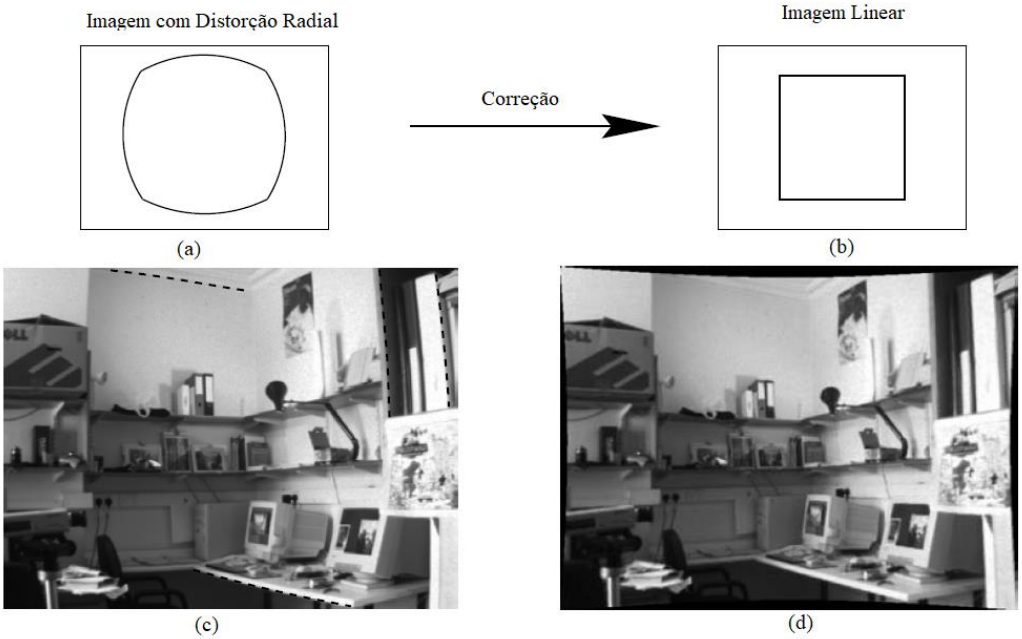

Figura 4: Comparação de imagens com distorção radial e corrigida sem distorção. Adaptado de [2].

Finalmente, o último parâmetro associado a uma câmera é a distorção radial. Essa distorção pode ser causada pelo ambiente (como ambientes subaquáticos) ou pelo uso de lentes. Na figura 4 (a) e (c) é possível ver imagens com distorção radial, já nas figuras 4 (b) e (d) é possível ver como as imagens ficam sem essa distorção. Para utilizar a projeção e encontrar o local exato de um ponto no plano das imagens, as imagens devem passar por uma correção para ter a distorção radial removida.

Como foi dito no inicio dessa seção, a matriz de projeção é uma matriz de mapeamento, sendo assim, se a matriz de projeção *P* da figura 3 for conhecida, é possível saber em quais coordenadas o ponto x no plano das imagens (que representa o ponto X no mundo real 3D) pode ser encontrado. A equação *3* que define a matriz de projeção P_c de uma câmera é detalhada em [1]. A equação *4* define o mapeamento de um ponto de um ponto X do mundo real (3D) para um ponto x no plano da imagem (2D), onde o elemento 1 na segunda matriz está presente apenas para possibilitar a multiplicação das matrizes.

$$P_c = K_c \,.\, E_c = K_c \,.\, [R_c \mid T_c] \tag{3}$$

$$x = P_c \,.\, \begin{bmatrix} X^T \\ 1 \end{bmatrix} \tag{4}$$

2.2. Parâmetros e Calibração de um Sistema Estéreo

Um sistema de visão estéreo conta com duas câmeras. Essas câmeras contém, respectivamente, os parâmetros intrínsecos K_1 e K_2, parâmetros extrínsecos E_1 e E_2 (que são formados pelas matrizes de Rotação R_1, R_2 e pelas matrizes de translação T_1, T_2) e distorções radiais associadas a cada uma delas, que foram os parâmetros descritos anteriormente.

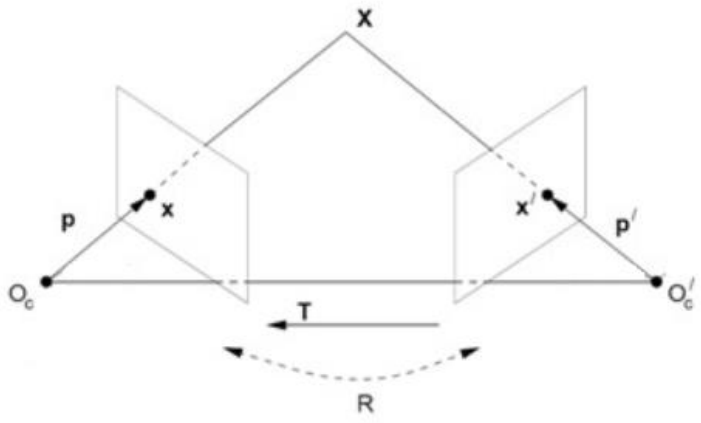

Figura 5: Parâmetros de um sistema estéreo. Adaptado de [3].

Além dos parâmetros descritos no paragrafo acima, um sistema estéreo conta também com a matriz de translação T do sistema, que é a distância entre os centros das câmeras e a matriz de rotação R, a rotação entre as câmeras do sistema. Esses parâmetros são mostrados na figura 5.

Os parâmetros K_1, K_2, R e T e parâmetros de distorção descritos nessa seção podem ser obtidos através da calibração do sistema estéreo. Um modo de executar essa calibração é utilizar a função *stereocalibrate* [8] na biblioteca OpenCV. Essa função tem como entrada diversas imagens de tabuleiros de xadrez capturadas com o sistema estéreo a ser calibrado. O algoritmo localiza as arestas no tabuleiro de xadrez, como mostrado na figura 6, e utiliza essas informações para calcular os parâmetros do sistema estéreo.

Figura 6: Detecção de arestas no tabuleiro de xadrez para calibração de sistema estéreo em [8].

2.3. Retificação de imagens

A retificação de imagens pode ser dividida em duas partes. O objetivo do primeiro passo é encontrar as matrizes de rotação que quando aplicadas as câmeras já existentes, crie projeções de câmera que tenham as linhas epipolares horizontais, como é possível ver no primeiro passo na figura 7.

Em [3], um método para calcular essas matrizes de rotação é descrito. Esse método consiste em criar um sistema mutualmente ortogonal, isto é, um sistema no qual o plano da imagem da câmera esteja localizado onde os planos X, Y e Z do sistema sejam ortogonais entre si.

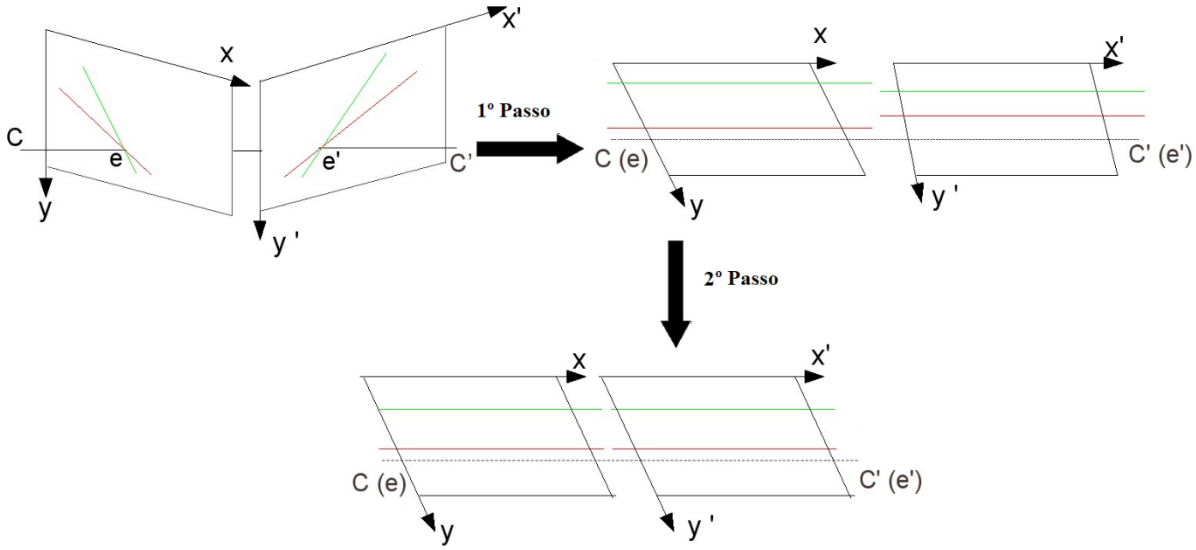

Figura 7: Passos para obtenção de imagens retificadas. Adaptado de [9].

Para a escolha do primeiro vetor E_1, pode-se usar uma escolha arbitraria, pois, esse vetor ainda não precisa ser ortogonal a nenhum outro. A escolha utilizada pelo autor de [3] é o vetor de translação do sistema, por ele ser um vetor conhecido dos parâmetros do sistema estéreo. Para facilitar os cálculos e criar vetores unitários, tanto nesse passo quanto nos demais, o vetor escolhido é dividido pelo seu módulo.

$$E_1 = \frac{T}{|T|} \tag{5}$$

No segundo passo, o vetor E_2 deve ser ortogonal ao vetor E_1. Existem alguns vetores que satisfazem essa restrição. O vetor $[-Ty, Tx, 0]$ mostrado na equação *6* foi o escolhido e a divisão pelo seu módulo também é mostrada na equação.

$$E_2 = \frac{[-Ty, Tx, 0]^T}{\sqrt{Tx^2 + Ty^2}} \tag{6}$$

Por fim, o terceiro vetor E_3 deve ser ortogonal aos vetores E_1 e E_2, assim podemos usar o produto cartesiano para calcular seu valor, como é mostrado na equação *7*.

$$E_3 = E_1 \times E_2 \tag{7}$$

A primeira matriz de Rotação R_1 é formada pelos vetores E_1, E_2 e E_3 como é mostrado na equação 8.

$$R_1 = \begin{pmatrix} {E_1}^T \\ {E_2}^T \\ {E_3}^T \end{pmatrix} \tag{8}$$

Como é possível ver no segundo passo da figura 7, após a aplicação de R_1 nas câmeras, ainda pode haver uma rotação entre as câmeras. Essa rotação é o parâmetro R, descrito na seção 2.2. O segundo passo da retificação é rotacionar essa segunda câmera para que a rotação das câmeras seja igual. Aplicando a rotação R_1 calculada anteriormente e a rotação R, obtemos a rotação total que será aplicada a segunda câmera na equação *9*.

$$R_2 = R.R_1 \tag{9}$$

Aplicando a equação *3* é possível calcular as projeções das câmeras de um sistema estéreo. Nesse caso a câmera *1* é a câmera de referência do sistema, logo, a translação dela é zero.

$$P_1 = K_1.[R_1 \mid 0] \tag{10}$$

$$P_2 = K_2.[R_2 \mid T] \tag{11}$$

Na figura 8 é possível ver os planos π_{r0} e π_{l0} representando as projeções das câmeras de um sistema estéreo antes da aplicação da retificação e os planos π_{r1} e π_{l1} representando as matrizes de projeção após a retificação.

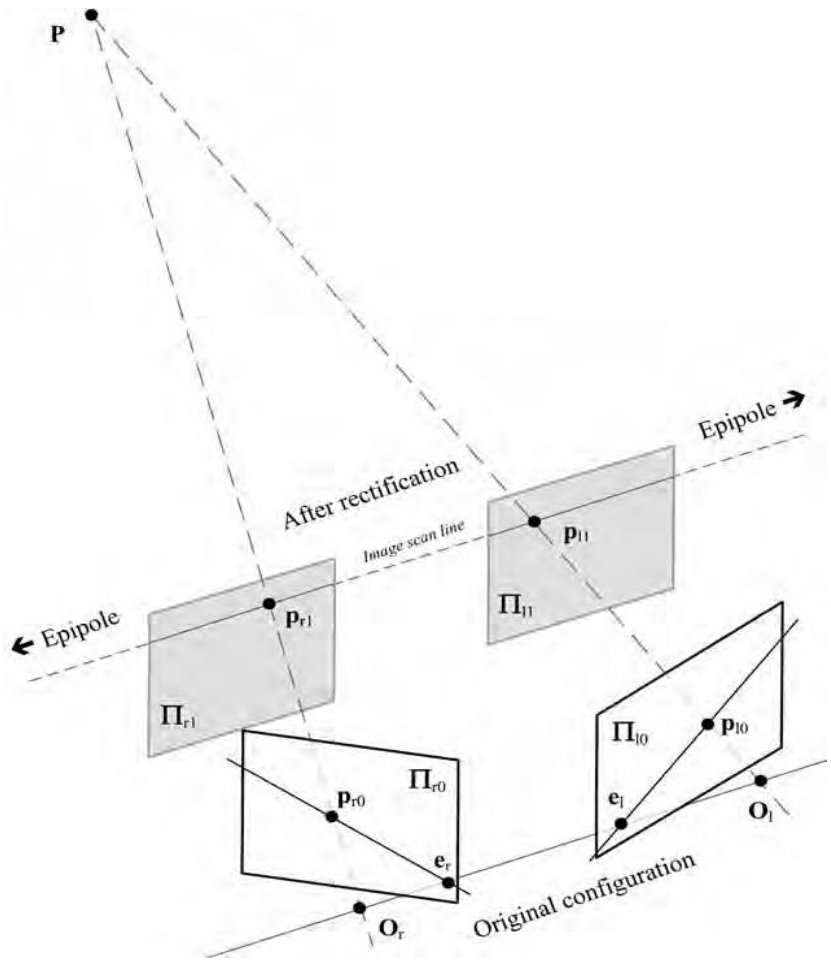

Figura 8: Diferença na projeção das câmeras antes e depois da aplicação das rotações [3].

3. Trabalhos Relacionados

Essa seção mostra alguns trabalhos relacionados com a área da retificação de imagens, o que esses trabalhos se propõem a fazer e o motivo destes trabalhos serem ou não possíveis de adaptar para as implementações que esse trabalho propõe.

3.1. Algoritmo Compacto de retificação

O artigo em [1] é um trabalho de base com várias citações. O trabalho propõe o aprimoramento das matrizes de projeção que é mostrada na seção anterior para criação de um mapa de substituição de posições com menos erros.

Um problema que esse algoritmo gera para uma implementação em hardware é a alto número de multiplicações de matrizes e vetores. Multiplicações de matrizes são séries de multiplicações e somas. Quando passado para hardware, seria necessária a criação de várias instancias de módulos para lidar com esse aspecto.

Outro problema é a inversa das matrizes utilizadas em seus cálculos. Matrizes inversas são altamente custosas para serem calculadas mesmo para um projeto em software. O algoritmo desenvolvido nesse trabalho foi implementado na linguagem de programação MATLAB, assim facilitando o desenvolvimento do algoritmo, as multiplicações de matrizes e as inversas das mesmas.

3.2. Implementação de Tabela de Consulta (LUT)

O artigo em [6] é uma implementação em FPGA de uma LUT (*Look up Table*), que também pode ser chamado de tabela de consulta. Essa proposta visa calcular todas posições de cada câmera de um sistema estéreo na retificação e utilizar esses pontos para as imagens que chegam através das câmeras.

Assim como o artigo apresentado na seção 3.1, os cálculos desse artigo também apresentam multiplicações de matrizes e a adição de uma matriz inversa. Os autores dos artigos conseguiram passar por esse fato carregando as matrizes resultantes dessa operação como constantes, porém isso remove a possibilidade do projeto ser portado para outros sistemas estéreos.

Outro ponto do trabalho deste artigo é o uso de uma memória para acessar os dados. No trabalho imagens de 640x512 são utilizadas. Isso significa que *327680* pontos (*x*, *y*) são armazenados para cada câmera, resultando em um espaço de aproximadamente 5,24MB.

O objetivo final do trabalho aqui proposto é adicionar esse módulo de retificação a um sistema já existente e esse sistema não conta com memória o suficiente para armazenar dados dessa grandeza.

3.3. Projeção Simples

Em [2] é mostrado um método de projeção simples que é a projeção de 3D para 2D, que facilita a localização dos pontos referentes a um ponto no mundo real no plano das imagens. Isso pode ser executado com matrizes de projeção de perspectiva, somas, multiplicações e divisões.

O problema é que há distorções associadas a câmera que não são mostradas neste trabalho, assim, podendo gerar pontos incorretos.

4 Implementações propostas para o Algoritmo de Retificação de imagens

Nesse capítulo serão apresentados uma implementação do algoritmo de retificação em C++ e uma implementação RTL em SystemVerilog para posterior implementação em FPGA. .

4.1 Visão Geral

O diagrama na figura 9 mostra a sequência de passos utilizados na retificação de imagens. O diagrama abaixo representa o processo de retificação feito para uma imagem. Como em um sistema de visão estéreo há duas imagens, o processo deve ser realizado para cada uma delas.

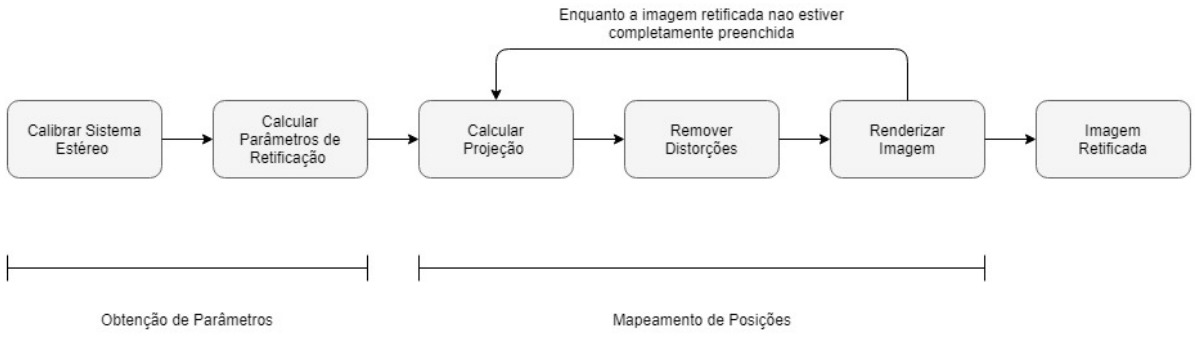

Figura 9: Diagrama da retificação utilizada no projeto.

A etapa de obtenção de parâmetros, como seu próprio nome já diz é necessária para se obter os parâmetros utilizados na retificação de imagens. Essa etapa precisa ser feita apenas uma vez para o mesmo sistema estéreo e os dados obtidos podem ser usados em várias execuções da retificação, contanto que não haja nenhuma alteração no sistema estéreo. Um sistema estéreo é alterado quando a distância entre as câmeras ou a rotação das delas é alterada. Outras alterações no sistema estéreo são a troca de câmeras, adição ou remoção de lentes nas câmeras ou troca de ambientes para locais que causem distorção na captura de imagens, como por exemplo ambientes subaquáticos.

A primeira parte da obtenção de parâmetros é a calibração do sistema estéreo. Como foi citado na seção 2.2, a calibração do sistema estéreo é feita usando a biblioteca OpenCV. Essa calibração tem como saída as matrizes de parâmetros intrínsecos K_1 *e* K_2 da câmera *1* e da

câmera *2* do sistema estéreo, além da matriz de rotação *R* entre as câmeras e a matriz de translação T entre as câmeras.

A segunda parte da obtenção de parâmetros é o cálculo dos parâmetros de rotação R_1 *e* R_2 das câmeras do sistema para que seja possível o cálculo das matrizes de projeções P_1 *e* P_2 . Esses cálculos foram descritos na seção 2.3.

A segunda etapa de retificação de imagens é o mapeamento de posições. O objetivo dessa etapa é preencher a imagem retificada com os pixels oriundos da imagem não retificada, como ilustrado na figura 10 para os dois primeiros pixels da imagem retificada.

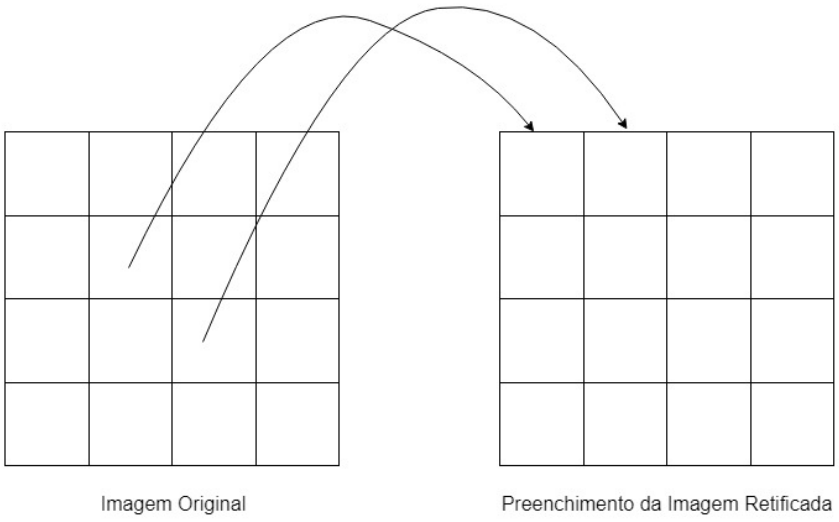

Figura 10: Preenchimento de uma imagem retificada com os pixels da imagem original.

Para preencher os pixels da imagem retificada é necessário calcular para cada ponto da imagem retificada (que será referido nesse trabalho como (x_r, y_r)), o pixel correspondente na imagem original (que será referido nesse trabalho como (x_o, y_o)). Esse processo de mapeamento começa com o cálculo da projeção, que irá calcular qual a localização do pixel na imagem original que deve ser usada para preencher o ponto em análise na imagem retificada.

O processo de cálculo da projeção não conta com a remoção de distorção, o que pode causar um erro na localização dos pontos (x_o, y_o). Por isso, a segunda parte do mapeamento de posições é a remoção de distorções.

Por fim, as coordenadas calculadas estarão sem distorções. A parte de renderização irá transformar as coordenadas sem distorção recebidas em pixel, para que a posição (x_o, y_o) seja obtida e preencher a posição em análise da imagem retificada (x_r, y_r) com o pixel correspondente (x_o, y_o).

4.2 Implementação em C++

4.2.1 Calibração do Sistema Estéreo

Como foi descrito na seção 2.2 deste trabalho, a calibração do sistema estéreo consiste em adquirir parâmetros inerentes a esse sistema. Esses parâmetros são as matrizes de parâmetros intrínsecos K_1 e K_2, a matriz de rotação R entre as câmeras do sistema, a matriz de translação T e os parâmetros de distorção. Este trabalho utiliza os parâmetros de calibração do sistema estéreo em [7]. Em [7] a calibração foi feita utilizando função de calibração estéreo do OpenCV, *stereocalibrate* [8], como explicado na seção 2.2.

4.2.2 Calculo de Parâmetros de Retificação

Na etapa de cálculo de parâmetros de retificação, os parâmetros de rotação R_1, R_2 são aplicados às câmeras do sistema para se obter as matrizes de projeção P_1, P_2. Os parâmetros de rotação são calculados como é mostrado nas equações *5*, *6*, *7*, *8* e *9* da seção 2.3.

Os parâmetros de rotação calculados serão usados para calcular as matrizes de projeção P_1, P_2, como apresentado nas equações *10* e *11* da seção 2.3.

No pseudocódigo abaixo é descrito o fluxo para se obter as matrizes de rotação e de projeção, utilizando as equações descritas na seção 2.3

```
Funcao calcular_projecoes(R, T)
     /*Calcular E1, E2, E3 respectivamente mostrados nas
     equações 5, 6 e 7*/
     E1 = T/modulo(T)
     E2 = [-T[1], T[0], 0]/raiz(T[0]* T[0] + T[1]* T[1])
     E3 = produto_vetorial(E1, E2)

     /*Prencher o vetor R1 com os vetores calculados acima,
     como mostrado na equação 8*/
     R1[0] = transposta(E1)
     R1[1] = transposta(E2)
     R1[1] = transposta(E3)

     /*Calcular os valores da matriz de rotação R2 como
     mostrado na equação 9*/
     R2 = R1*R

     /*Calcular os valores das matrizes de rotação P1 e P2 como
     mostrado nas equações 10 e 11*/
```

```
P1 = K1*[R1|0]
P2 = K2*[R2|T]
```

4.2.3 Cálculo da Projeção de Pontos

Como descrito na seção 2.1, a matriz de projeção calculada na obtenção de parâmetros é uma matriz de mapeamento. A equação 4 na seção 2.1 será repetida abaixo, onde *X* é o ponto no mundo real em 3D e *x* o ponto projetado no plano das imagens.

$$x = P_c \cdot \begin{bmatrix} X^T \\ 1 \end{bmatrix} \tag{4}$$

Expandindo as matrizes:

$$x = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Na projeção da câmera de referência os elementos da quarta coluna são iguais a zero, já para a segunda câmera, a quarta coluna contém informação sobre as distâncias das câmeras. Como o que está sendo calculado são os pontos de uma câmera não retificada, para uma mesma câmera retificada, as informações de distância entre câmeras não são uteis para cálculo, então os valores da quarta coluna serão considerados como zero para as duas projeções, assim o cálculo pode ser simplificado para:

$$x = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Aplicando a multiplicação de matrizes:

$$x = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} P_{11}.X + P_{12}.Y + P_{13}.Z \\ P_{21}.X + P_{22}.Y + P_{23}.Z \\ P_{31}.X + P_{32}.Y + P_{33}.Z \end{bmatrix} \tag{12}$$

Com isso, é possível descobrir a projeção do ponto X (X, Y, Z), para o ponto x (x', y', z'). Esse mapeamento pode ser visto na figura 3 na seção 2.1.

Por fim, com o ponto x encontrado, é necessário encontrar uma conversão do plano euclidiano em 3D para o plano das imagens em 2D. O método para este cálculo é descrito em [6], que consiste em dividir todos os elementos do vetor x pelo elemento z', assim o último elemento do vetor será fixo, como é mostrado na equação *13*.

$$x = (x', y', z') = (\frac{x'}{z'}, \frac{y'}{z'}, 1) \tag{13}$$

O ponto $(\frac{x'}{z'}, \frac{y'}{z'})$ é o ponto no plano das imagens que projeta o ponto X no mundo real, porém ele pode ter uma distorção associada que pode fazer com que o ponto esteja deslocado. Por esse motivo, esse ponto será chamado de $(x_{distorcido}, y_{distorcido})$.

4.2.4 Remoção de Distorções

A segunda parte do mapeamento de posições é a remoção de distorções. Como dito na seção 2.1, uma imagem pode ter distorções associadas ao meio ou a lentes usadas na câmera, como pode ser visto na figura 4, que contém imagens com e sem distorção radial. Os coeficientes distorções radiais são $(k_1, k_2, k_3, k_4, k_5, k_6, k_7)$ são obtidos na fase de calibração do sistema estéreo. Em [6], as equações para remoção das distorções são descritas. A remoção da distorção é realizada conforme as equações abaixo:

$$DR_X = x_{distorcido}.(k1.r^2 + k2.r^4 + \cdots) \tag{14}$$

$$DR_Y = y_{distorcido}.(k1.r^2 + k2.r^4 + \cdots) \tag{15}$$

$$x_{semdistorcao} = x_{distorcido} + DR_X = x_{distorcido}.(1 + k1.r^2 + k2.r^4 + \cdots) \tag{16}$$

$$y_{semdistorcao} = y_{distorcido} + DR_X = x_{distorcido}.(1 + k1.r^2 + k2.r^4 + \cdots) \tag{17}$$

Onde:

$$r^{2=} {x_{distorcido}}^2 + {y_{distorcido}}^2 \tag{18}$$

4.2.5 Renderização

Por fim, a última parte da etapa de mapeamento de posições consiste na renderização das imagens. Esse processo tem como objetivo encontrar e preencher cada ponto (pixel) das imagens retificadas com o seu ponto correspondente na imagem original.

Para fazer a renderização é necessário usar o ponto $(x_{semdistorcao}, y_{semdistorcao})$ calculado na remoção de distorções. Esses valores contém a posição no plano das coordenadas do ponto que será atualizado na imagem retificada. Para finalizar a renderização é necessário calcular a posição do pixel que se refere ao ponto $(x_{semdistorcao}, y_{semdistorcao})$. Este cálculo é feito de acordo com as equações abaixo:

$$x_r = arredondar(x_{semdistorcao}.fx + px) \quad (19)$$

$$y_r = arredondar(y_{semdistorcao}.fy + py) \quad (20)$$

Onde os parâmetros *fx, px, fy* e *py* são parâmetros intrínsecos encontrados na matriz das câmeras e *arredondar* ( ) é a função de arredondamento de real para inteiro. O pseudocódigo a seguir, descreve o fluxo para obter imagens retificadas. Vale ressaltar que esse processo gera a imagem retificada de uma das câmeras. Para obter a imagem da segunda câmera do sistema estéreo, o algoritmo deve ser executado novamente com os parâmetros da segunda câmera.

```
Funcao retificar_imagem(P, K, D)
     /* Os parâmetros p11, p12, p13, p21, p22, p23, p31, p32,
     p33 são obtidos na matriz projeção P
     Já fx, fy, px e py são obtidas na matriz de câmera K
     Os parâmetros k1, k2 e k3 são obtidos no vetor de distorção
     D*/

     z=1
     enquanto (linha < altura_img)
          enquanto (coluna < largura_img)
               /* Calculo da projeção conforme descrito na
               seção 4.2.3 */

               x3d = p11*linha + p12*coluna + p13*z
               y3d = p21*linha + p22*coluna + p23*z
               z3d = p31*linha + p32*coluna + p33*z
               x2d = x3d/z3d
               y2d = y3d/z3d
```

```
            /* Calculo e remoção da distorção conforme
            descrito na seção 4.2.4 */

            raioquadadro = r2 = x*x + y*y
            distorcao = 1 + k1*r2 + k2*r2*r2 + k3*r2*r2*r2
            x_d = x*distorcao
            y_d = y*distorção

            //Renderização conforme descrito na seção 4.2.5

            x_pixel = arredondar(fx*x_d + px)
            y_pixel = arredondar(fy*y_d + py)
            imagem_retificada[linha][coluna] =
            imagem_original[x_pixel][y_pixel]
            coluna = coluna + 1
        linha = linha + 1
```

4.3 Implementação em SystemVerilog RTL

Diferentemente da implementação na linguagem C++, a implementação em uma linguagem de descrição de hardware realizada neste trabalho considerou apenas a etapa de mapeamento de posições. Os parâmetros obtidos na etapa de obtenção dos parâmetros em C++ são utilizados como valores de entrada para os módulos descritos em SystemVerilog RTL.

As três partes da etapa de mapeamento de posições mostradas na figura 9 foram implementadas na linguagem SystemVerilog RTL. Nas seções abaixo as máquinas de estados que implementam cada uma dessas partes serão descritas e detalhadas.

Além dos módulos que compõe o mapeamento de posições, é necessário também que haja um controle de linhas e colunas. Esse controle na implementação em software é facilmente especificado com estruturas de repetição aninhadas, porém a implementação desse controle no nível RTL é um pouco mais complexa.

A figura 11 mostra a máquina de estados da implementação em SystemVerilog RTL. O primeiro módulo é responsável por controlar as linhas e colunas na imagem retificada que serão preenchidas durante execução do algoritmo. O segundo módulo, o módulo de cálculo de projeção, tem como entrada a linha x_r e coluna y_r a serem preenchidas na imagem retificada e tem como retorno o ponto $(x_{distorcido}, y_{distorcido})$. O terceiro módulo tem como entrada o ponto $(x_{distorcido}, y_{distorcido})$ e é responsável por retirar a distorção desse ponto, retornando o ponto $(x_{semdistorcao}, y_{semdistorcao})$. Por fim, o módulo de renderização calcula o ponto (x_o, y_o) da imagem original que deve ser colocado na linha e coluna (x_r, y_r) em análise.

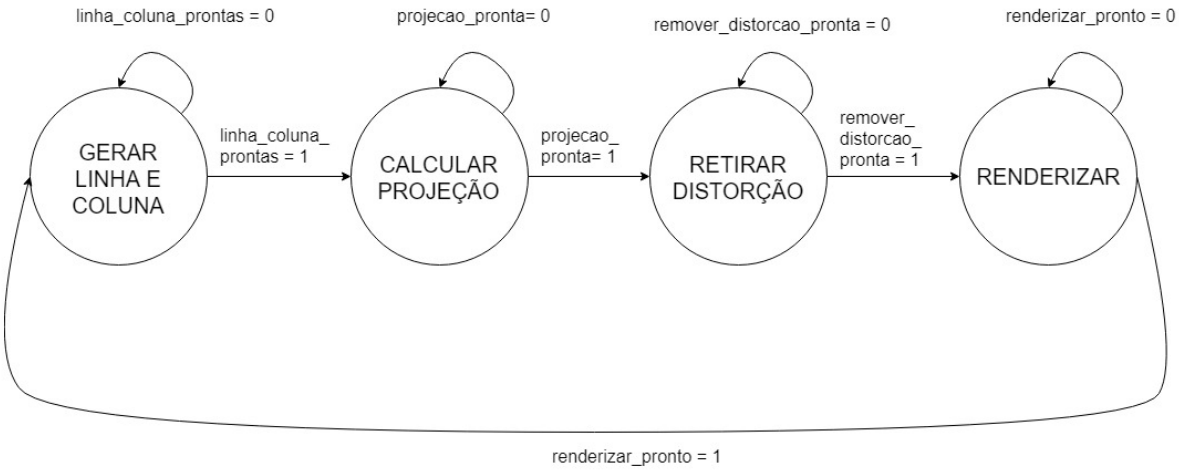

Figura 11: Máquina de estados do módulo de controle da implementação SystemVerilog RTL.

Os somadores, multiplicadores utilizados nesse projeto foram disponibilizados pelo autor do trabalho em [7]. Esses módulos operam com números na representação ponto flutuante com precisão simples, que contém 32 bits para representar um número em ponto flutuante como binário.

Na descrição dos módulos a seguir, os estados de resetar os módulos, fazendo com que os módulos sejam configurados com seus valores iniciais foram omitidos.

4.3.1 Módulo de geração de linha e coluna

Como dito anteriormente na especificação RTL, o controle de linhas e colunas tem que ser feito por um módulo separado. Nesse módulo o valor das linhas e colunas são adicionados um a um e suas saídas são a linha x_r e a coluna y_r a ser preenchida na imagem retificada. Como é possível ver na figura 12, primeiro, o valor da coluna é somado até que esse valor alcance o valor da largura da imagem, então, o valor da linha é somado e o processo de somar a coluna e linha se repete até que todas as linhas estejam completamente varridas.

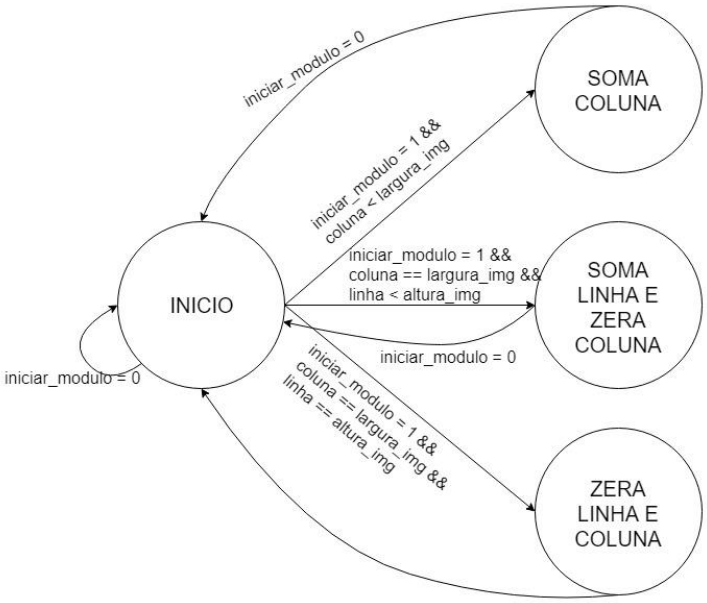

Figura 12: Máquina de estados do módulo de controle de linhas e colunas.

4.3.2 Módulo de cálculo de projeção

Na seção 2.1 foi descrito que a matriz de projeção contém parâmetros que torna possível calcular o ponto $(x_o,\ y_o)$ que irá preencher o ponto $(x_r,\ y_r)$ na imagem retificada. Na seção 4.2.3 esse processo de utilização da matriz de projeção foi descrito com mais detalhes.

Como as linhas x_r e colunas y_r geradas no módulo descrito anteriormente são representadas como número inteiros, é necessário converter esses dois parâmetros para a notação de ponto flutuante, assim eles serão compatíveis com os módulos de multiplicação e soma de ponto flutuante usados nos demais módulos. Isso é feito nos estados "transforma linha ponto flutuante" e "transforma coluna ponto flutuante" na figura 13.

Quando os valores de linha e coluna são convertidos para a notação de ponto flutuante, é possível aplicar a equação *12* para encontrar os valores que compõe a matriz $x = [x' \quad y' \quad z']^T$.

Finalmente, com os elementos da matriz x, é possível calcular a conversão do plano 3D para o plano 2D a partir da equação *13*.

As saídas desse módulo é o ponto $(x_{distorcido}, y_{distorcido})$ com as distorções da câmera ainda aplicados a ele.

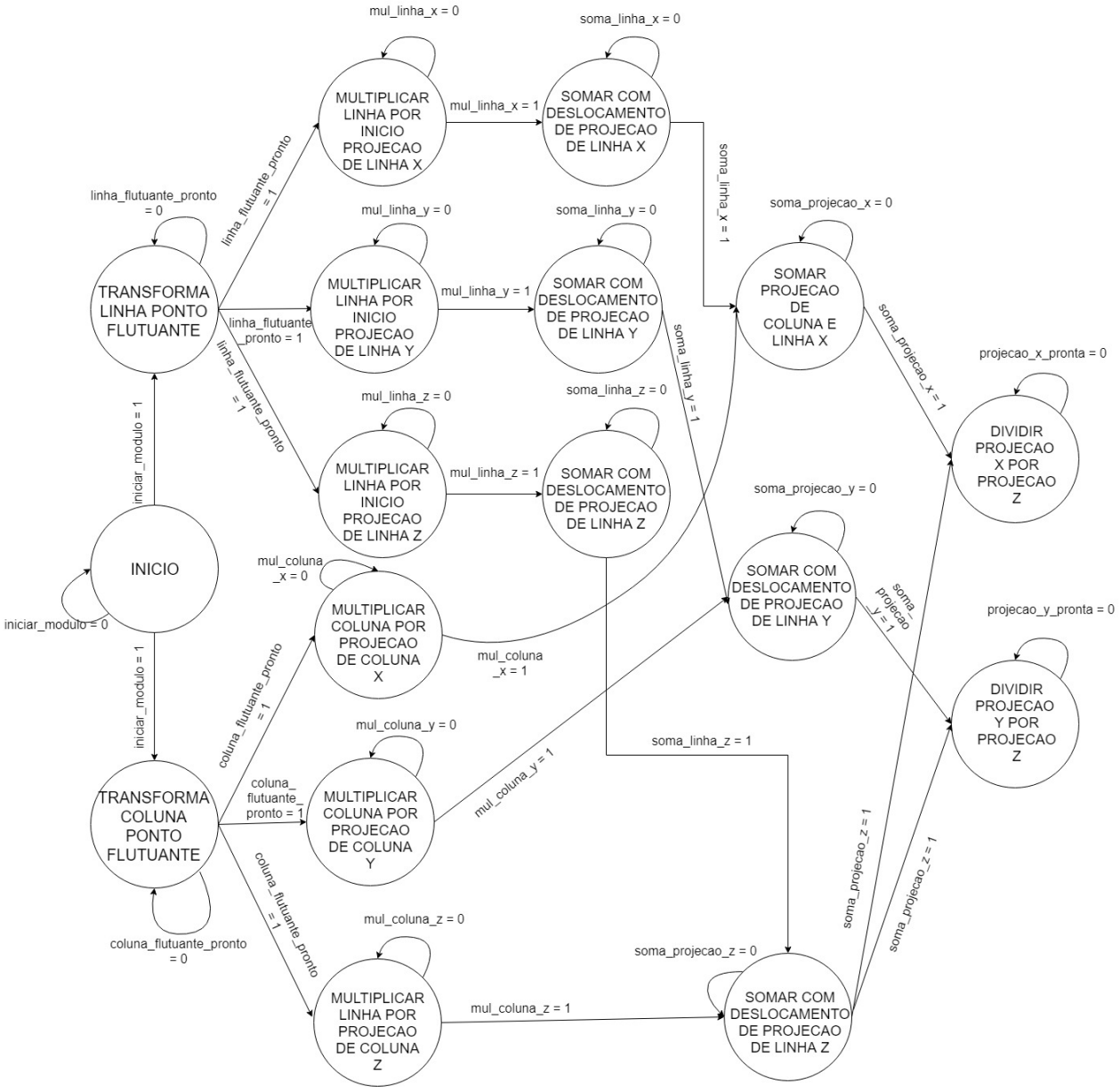

Figura 13: Máquina de estados do módulo de cálculo de projeção.

4.3.3 Módulo de remoção de distorção

Na seção 2.1 as distorções associadas a uma câmera foram explicadas e na seção 4.2.4 são mostradas equações para remoção dessa distorção. Enquanto uma câmera pode ter vários coeficientes k de distorção radial, segundo a metodologia discutida em [6] o uso de três coeficientes (k_1, k_2, k_3) são mais que suficientes para compensar todas as distorções.

O módulo apresentado nessa seção recebe como entrada um ponto $(x_{distorcido}, y_{distorcido})$ calculado no módulo anterior (módulo do cálculo de projeção) e tem

como saída outro ponto, porém sem as distorções associadas à câmera. Esse módulo utiliza as equações *16* e *17* descritas na seção 4.2.4 deste trabalho.

A máquina de estados desse módulo pode ser vista na figura 14.

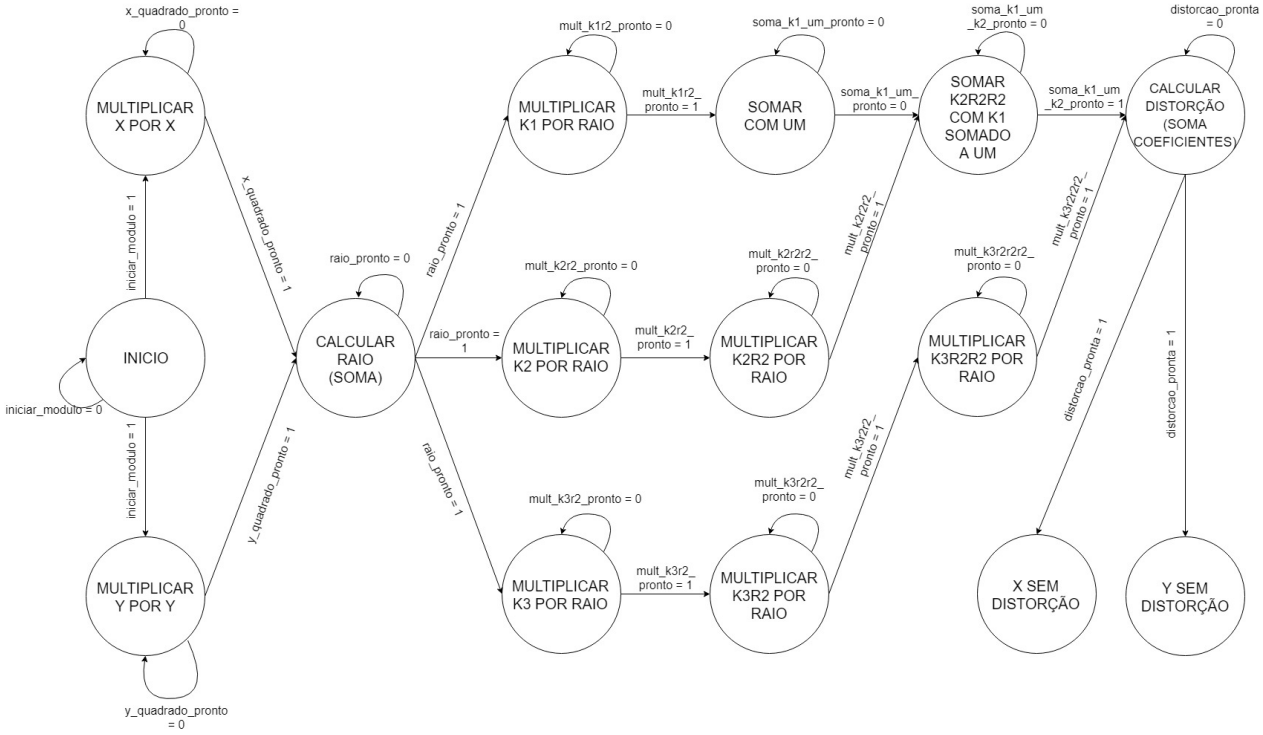

Figura 14: Máquina de estados do módulo de remoção de distorção.

Como todos os parâmetros das equações calculadas por esse módulo dependem do raio da distorção, calculado segundo a equação *18* na seção 4.2.4, então esse raio será o primeiro parâmetro calculado. Para isso, é necessário calcular os quadrados de $x_{distorcido}$ e $y_{distorcido}$ (ilustrados na máquina de estados da figura 14 como X e Y). Quando as multiplicações para obter os quadrados estão prontas, é necessário somar esses dois quadrados para assim obter o raio da distorção.

Após o cálculo do raio, é necessário achar cada parte dos coeficientes, como descrito nas equações *16* e *17*. Para obter a parte referente ao coeficiente k_n deve-se multiplicar o termo k_n por *n* vezes o raio. Nos estados "multiplicar *kn* por raio" na figura 14 são implementadas essas multiplicações.

Após as multiplicações dos raios, é necessário somar todos os coeficientes obtidos para calcular a distorção *k* total e somar esse valor a um, como mostra as equações *16* e *17*. Isso é

feito nos estados denominados “somar com um”, “somar k2r2r2 com k1 somado a um” e “calcular distorção (soma dos coeficientes).”

Por fim, com o valor da distorção calculado, é necessário aplicar esse valor à variável $x_{distorcido}$ e à variável $y_{distorcido}$, assim obtendo a saída do módulo que são as variáveis $x_{semdistorcao}$ e $y_{semdistorcao}$. Isso é feito nos estados denominados “x sem distorção” e “y sem distorção” conforme as equações *16* e *17*.

4.3.4 Módulo de Renderização

O módulo de renderização é assim chamado, pois na parte da programação em software a imagem retificada é preenchida usando as posições calculadas durante o processo de mapeamento de posições. Na implementação RTL, esse módulo poderia facilmente ter sido renomeado para “calcular posição final,” porém, para manter o padrão com o modelo de software, esse trabalho continuará chamando este módulo de renderização.

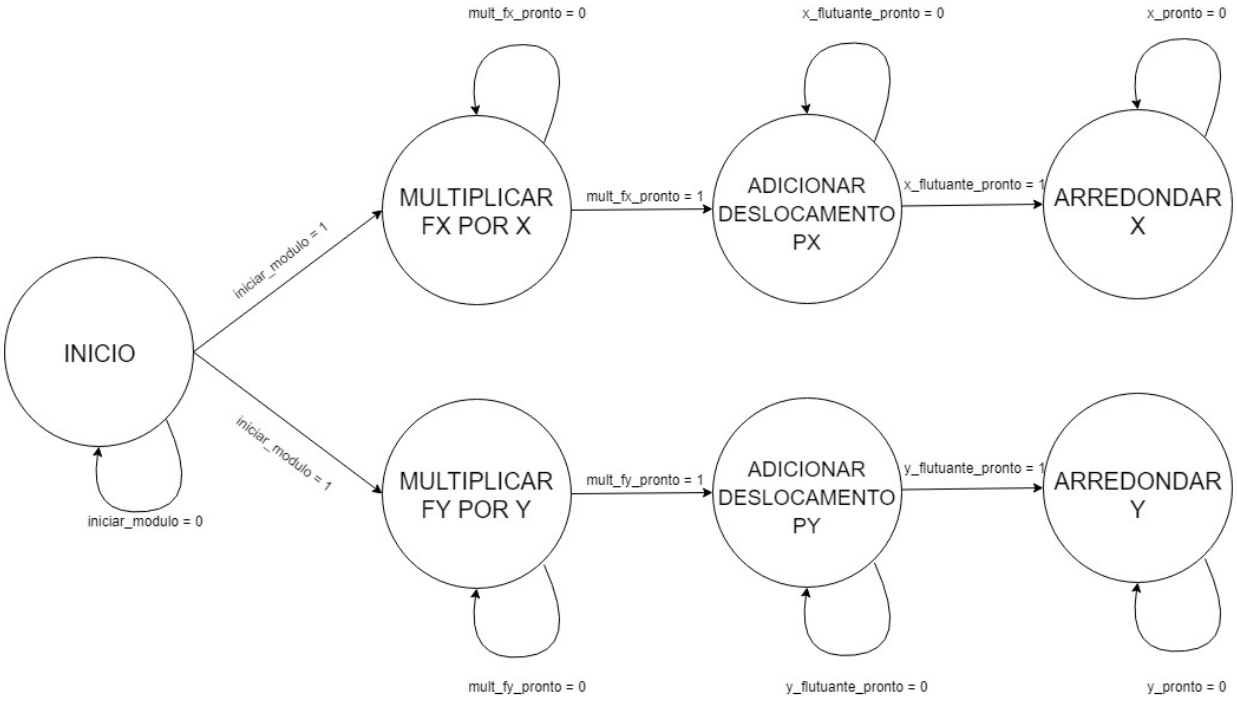

Figura 15: Máquina de estados do módulo de renderização.

Esse módulo recebe como entrada os parâmetros $(x_{semdistorcao}, y_{semdistorcao})$. Como já foi dito no módulo de cálculo de projeção, os pontos *x* e *y* representam a projeção do mundo real 3D no plano da imagem 2D. Para descobrir qual a posição do pixel (também referido como

ponto na matriz da imagem) que esse ponto ocupa na imagem, é necessário aplicar as equações *19* e *20*. A máquina de estados desse módulo é mostrada na figura 15.

Quando o módulo receber o sinal de início, as posições $x_{semdistorcao}$ e $y_{semdistorcao}$ são multiplicadas pela distância focal da câmera relativa a cada parâmetro. Isso é feito nos estados denominados “multiplicar fx por x” e “multiplicar fy por y.”

Quando os resultados dos estados de multiplicação são obtidos, é preciso somar os deslocamentos px e py. Por fim, é necessário arredondar o valor obtido na soma dos deslocamentos para se obter os números inteiros x_o e y_o para serem usados na obtenção do pixel no ponto $(x_o,\ y_o)$ calculado. O ponto $(x_r,\ y_r)$ na imagem retificada será preenchido com o pixel vindo do ponto $(x_o,\ y_o)$.

5 Resultados e Análises

Como foi mencionado na seção anterior, experimentos foram realizados para validar o algoritmo implementado em C++, bem como a implementação em SystemVerilog RTL. Nessa seção serão mostrados os resultados obtidos nas duas validações.

As câmeras utilizadas nesse trabalho geram imagens com a resolução de *480x640*, ou seja, cada imagem gerada em uma câmera contém *307200* pixels.

Como apresentado na seção 4, para cada ponto (x_r, y_r) na imagem retificada existe um (x_o, y_o) correspondente na imagem original. Os pontos (x_o, y_o) calculados nas seções 4.2.5 e 4.2.4 são números reais e para serem inseridos nas posições de uma matriz de inteiros, esses números precisaram ser arredondados para inteiros. Os erros nas tabelas 1, 2 e 3 são calculados antes do arredondamento para inteiro, já o número de pontos calculados errados é calculado após esse arredondamento.

Todos os resultados das análises abaixo foram arredondados na sétima casa decimal.

5.1 Implementação em C++

Os resultados obtidos na tabela abaixo foram obtidos usando como base os algoritmos para comparação o trabalho em [7]. Esse trabalho utiliza a biblioteca OpenCV para calcular a posição das imagens retificadas, então pode-se assumir que o trabalho aqui apresentado está comparando diretamente os seus resultados com os resultados da biblioteca OpenCV.

A tabela abaixo mostra o erro médio e o erro médio quadrático das posições x_o e y_o antes da transformação em inteiro com a função de arredondamento.

	x	*y*
Erro médio	0.0012276	0.0004736
Erro médio quadrático	0.0012277	0.0010156

Tabela 1: Erro médio e erro médio quadrático das posições x_o e y_o calculados no algoritmo implementado em C++ comparado ao trabalho em [7].

Para essa simulação, um total de 675 pontos (x_o, y_o) não se encontram na mesma posição que o trabalho base de comparação. A porcentagem de pontos (x_o, y_o) calculados que não são iguais aos calculados no trabalho em [7] é de 0.2197265%.

5.2 Implementação SystemVerilog RTL

Assim como os resultados obtidos na seção 5.1, os resultados da tabela abaixo também foram obtidos comparando os pontos (*x, y*) do trabalho em [7].

	x	*y*
Erro médio	0.0004121	0.0000976
Erro médio quadrático	0.0004247	0.0002471

Tabela 2: Erro médio e erro médio quadrático das posições x_o e y_o calculados no algoritmo em implementado em SystemVerilog RTL comparado ao trabalho em [7].

Para a implementação RTL, um total de 299 pontos (x_o, y_o) não se encontram na mesma posição que o trabalho base de comparação. A porcentagem de pontos (x_o, y_o) calculados que não são iguais aos calculados no trabalho base de comparação é de 0.0973307%.

Como implementação RTL foi desenvolvida tendo como base o algoritmo em C++ também implementado nesse trabalho, os resultados da implementação RTL também são comparados com a implementação em C++, os resultados obtidos estão apresentados na tabela abaixo.

	x	*y*
Erro médio	0.0008154	0.0003754
Erro médio quadrático	0.0009648	0.0010702

Tabela 3: Erro médio e erro médio quadrático das posições x_o e y_o calculados no algoritmo em implementado em SystemVerilog RTL comparado ao algoritmo implementado em C++ deste trabalho.

Para comparação entre as implementações C++ e RTL, um total de 624 pontos (x_o, y_o) não se encontram na mesma posição que a implementação em C++ desse trabalho. Isso representa uma porcentagem de 0.02031250%.

5.3 Análise

Como foi mostrado nas seções acima, os resultados alcançados nesse trabalho no algoritmo em software têm um erro médio na ordem de 10^{-3} e um erro médio quadrático na

mesma ordem, o que pode ser considerado um erro baixo quando se observa a taxa de erro de pontos que é apenas de 0.2197265%.

Os *675* pontos calculados errados estão diretamente ligados a função de transformação em inteiro por arredondamento quando a primeira casa decimal está próxima ao número *5*. Um exemplo disso pode ser dado enquanto o OpenCV calcula uma posição (x_o, y_o) como (2.49, 33.32), após a função de arredondamento, a posição do OpenCV será (2, 33), já a implementação em C++ desse trabalho calcula para o mesmo ponto (2.51, 33,38), assim após o arredondamento o ponto será (3, 33), fazendo com que o ponto final calculado esteja errado.

A taxa de erro que é mostrada na implementação em SystemVerilog RTL se deve principalmente ao modo de representação de números ponto flutuante. Mesmo que a representação do número em binário seja próxima da representação real, ainda há um erro que se propaga durante os módulos. Além disso ainda há o erro associado à função de transformação em inteiro por arredondamento. Assim como mostrado no parágrafo anterior, mesmo erros pequenos (como o erro do arredondamento) podem fazer com que ponto calculado seja errado.

Como se pode ver nas seções 5.1 e 5.2, a implementação SystemVerilog RTL obteve menores taxas de erro e menor número de pontos gerados errados. Isso pode ser explicado por dois motivos. O primeiro motivo é que na implementação em C++, os números reais estavam sendo arredondados na quarta casa decimal. O segundo motivo, como pode ser visto no pseudocódigo da seção 4.2.5, é cálculo dos pontos $(x',\ y',\ z')$, que são os pontos 3D usados para calcular os pontos 2D como foi descrito na seção 4.2.3. Esses pontos são somados a cada coluna, assim quando há um erro associado, esse erro é propagado para as demais colunas até a próxima linha.

6 Conclusões e Trabalhos futuros

6.1 Conclusões

Este trabalho apresentou a implementação da técnica de retificação de imagens na linguagem de programação, C++, e na linguagem de descrição de hardware SystemVerilog no nível de abstração RTL, desde a aquisição dos parâmetros intrínsecos e extrínsecos da câmera até o mapeamento dos pontos.

As taxas de erro nos pontos calculados pelos algoritmos implementados nesse trabalho podem ser consideradas baixas, já que no pior dos casos, na implementação em C++, o algoritmo teve uma taxa de erro de pontos menor que 0.3%. A diferença entre o mapa de disparidade gerado no trabalho em [7] (figura 16 (a)) e o mapa gerado na implementação em C++ desse trabalho (figura 16 (b)) é dificilmente perceptível, logo, o erro na retificação não causa grande impacto na geração do mapa de disparidade.

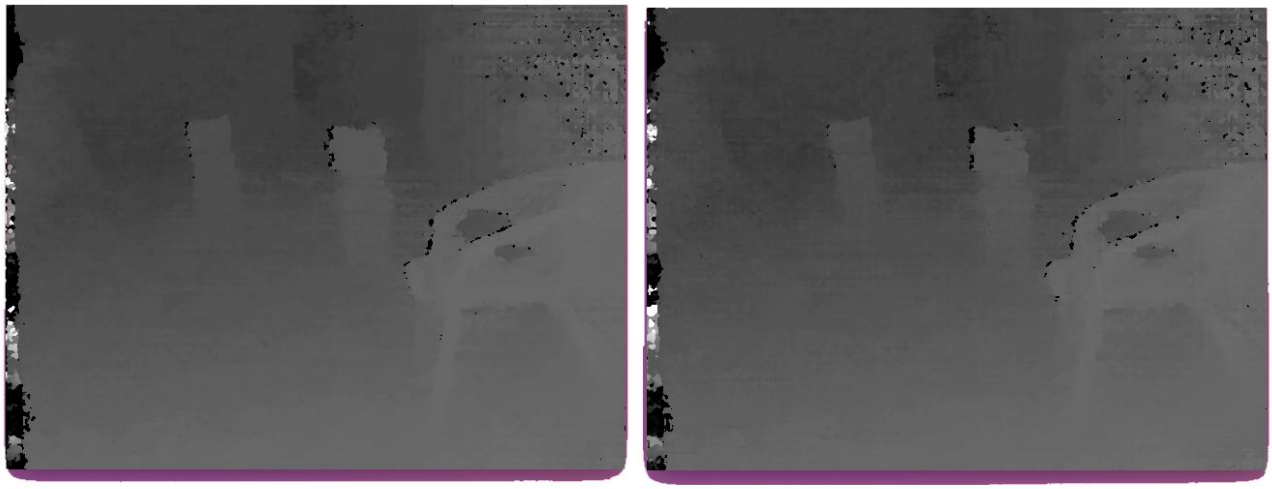

Figura 16: Diferença entre mapa de disparidade gerado no trabalho em [7] e o mapa de disparidade gerado na implementação C++ desse trabalho.

6.2 Trabalhos Futuros

Para diminuir o erro na implementação em C++, como trabalho futuro, as implementações devem ser alteradas para utilizar ponto flutuante com precisão dupla no lugar

da precisão simples utilizada nesse trabalho. É também importante retirar a dependência no cálculo dos pontos (x', y', z'), assim não haverá propagação de erro.

Como trabalho futuro, ainda é necessário realizar a simulação temporal da especificação SystemVerilog, para descobrir a qual taxa de clock quando o projeto for prototipado para execução em FPGA. A placa FPGA que esse projeto deve ser validado é a Cyclone IV, que é a mesma usada no projeto em [7].

Outro objetivo do trabalho é carregamento dos parâmetros, como as matrizes de câmera, matrizes de rotações e vetores de distorções através da comunicação entre hardware e software. Atualmente esses parâmetros estão sendo carregados como constantes para os módulos de hardware, porém isso limita otrabalho a apenas um sistema estéreo, que é o mesmo sistema estéreo do trabalho em [7].

Após a validação do módulo de retificação completa, o objetivo final é unir o módulo de retificação aos módulos desenvolvidos no trabalho descrito em [7], assim criando um trabalho de correspondência para visão estéreo que não precisa estar conectado a um computador para receber imagens retificadas.

7 Bibliografia

[1] Andrea Fusiello, Emanuele Trucco, and Alessandro Verri. 2000. A compact algorithm for rectification of stereo pairs. Mach. Vision Appl. 12, 1 (July 2000), 16-22.

[2] Richard Hartley and Andrew Zisserman. 2003. Multiple View Geometry in Computer Vision (2 ed.). Cambridge University Press, New York, NY, USA.

[3] B. Cyganek. 2007. An Introduction to 3D Computer Vision Techniques and Algorithms. John Wiley & Sons, Inc., USA.

[4] Nicu Sebe, Ira Cohen, Ashutosh Garg, and Thomas S. Huang. 2005. Machine Learning in Computer Vision (Computational Imaging and Vision). Springer-Verlag, Berlin, Heidelberg.

[5] LÚLIO, L. C. Técnicas de visão computacional aplicadas ao reconhecimento de cenas naturais e locomoção autônoma em robôs agrícolas móveis, 2011. Universidade de São Paulo.

[6] Vancea, Cristian and Sergiu Nedevschi. LUT-based Image Rectification Module Implemented in FPGA. 2007 IEEE International Conference on Intelligent Computer Communication and Processing (2007): 147-154.

[7] Cambuim, Lucas Fernando da Silva. Um módulo de Hardware de Tempo Real de Correspondência Semi Global para um Sistema de Visão Estéreo, 2017.

[8] OpenCV. Camera Calibration. Disponivel em <https://docs.opencv.org/2.4/modules/calib3d/doc/calib3d.html>. Acesso em: 5 de junho de 2018.

[9] Pascal Monasse, Jean-Michel Morel, Zhongwei Tang. Three-step image rectification. BMVC 2010 - British Machine Vision Conference, Aug 2010, Aberystwyth, United Kingdom. BMVA Press, pp.89.1--89.10, 2010