UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMA DE AUXÍLIO A NAVEGAÇÃO EM AMBIENTES INTERNOS PARA PESSOAS COM DEFICIÊNCIA VISUAL USANDO VISÃO ESTEREOSCÓPICA

THIAGO SOUSA CHIQUETO

ORIENTADOR: PROF. DR. EMERSON CARLOS PEDRINO

São Carlos - SP
Agosto/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SISTEMA DE AUXÍLIO A NAVEGAÇÃO EM AMBIENTES INTERNOS PARA PESSOAS COM DEFICIÊNCIA VISUAL USANDO VISÃO ESTEREOSCÓPICA

THIAGO SOUSA CHIQUETO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Processamento de Imagens e Sinais.

Orientador: Prof. Dr. Emerson Carlos Pedrino

São Carlos - SP
Agosto/2016

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Thiago Sousa Chiqueto, realizada em 01/08/2016.

Prof. Dr. Emerson Carlos Pedrino
(UFSCar)

Prof. Dr. Jose Hiroki Saito
(UFSCar)

****

Prof. Dr. Valentin Obac Roda
(UFRN)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Prof. Dr. Valentin Obac Roda. Depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Thiago Sousa Chiqueto.

Prof. Dr. Emerson Carlos Pedrino
Coordenadora da Comissão Examinadora
(UFSCar)

Dedico este trabalho primeiramente à minha família, principalmente a minha esposa que me incentivou durante todo esse período de estudos e me manteve firme para a conclusão desse projeto, e ao meu orientador que me ajudou e norteou para o cumprimento dos objetivos.

AGRADECIMENTO

Agradeço primeiramente a Deus por me dar a vida e a grande oportunidade de estudar e aprimorar meus conhecimentos. A minha família que sempre me apoiou e guiou-me em todas as minhas decisões. Agradecer em especial a minha esposa, que sempre esteve do meu lado e me apoiou nos momentos mais difíceis, além de me manter firme durante toda a pesquisa. Agradeço ao meu orientador Emerson por sempre me ouvir em diversos momentos, sempre me auxiliando, guiando e ajudando no desenvolvimento deste trabalho. Agradeço a todos os meus amigos que sempre torceram para o meu sucesso, e, o mais importante, para o sucesso desta pesquisa.

"Intensão sem ação é ilusão, ouse fazer e o poder lhe será dado! "

Autor Desconhecido

RESUMO

Considerando o alto índice de pessoas portadoras de algum tipo de deficiência visual, e além disso, boa parte dessas pessoas possui baixa renda, neste trabalho é proposto a criação de um sistema de navegação em ambientes internos por meio de equipamentos de baixo custo. Sendo utilizado *Python* como linguagem de programação em conjunto com a biblioteca *OpenCV*, o objetivo principal desta pesquisa será extrair um mapa de disparidade, e realizar testes em diversas plataformas, em conjunto com aplicações de filtros sobre imagens, geradas por meio da câmera de visão estereoscópica *Minoru3D* para o mapeamento de ambientes internos, informando ao usuário o caminho livre para locomoção.

Palavras-chave: Deficiência visual, Visão Estereoscópica, *OpenCV*, *Python*, *Minoru3D*, Mapa de Disparidade.

ABSTRACT

Considering the high rate of people with some kind of visual impairment, and, moreover, that most of these people have a low income, this work is proposed to create a navigation system in indoors environment through equipment low cost. Using Python as programming language, together with the *OpenCV* library, the main objective of this research is to extract a disparity map and perform testing on different platforms with application filters images generated by the stereoscopic vision camera Minoru3D for mapping indoor environments, informing the user the free way for locomotion.

Palavras-chave: Visual Impairment, Stereoscopic Vision, OpenCV, Python, Minoru3D, Disparity Map.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

6DOF - *Six Degrees Of Freendom*

API - *Application Programming Interface*

BM - *Block-Matching*

BSD - *Berkeley Software Distribution*

GPS - *Global Position System*

HSV - *Hue, Saturation e Brightness*

IBGE – *Instituto Brasileiro de Geografia e Estatística*

IMU - *Inertial Measurement Unit*

RAM - *Random Access Memory*

RFID - *Radio-Frequency IDentification*

RGB – *Red Green Blue*

SGBM - *Semi-Global Block Matching*

SLAM - *Simultaneous Localization and Mapping*

USB - *Universal Serial Bus*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

O presente capítulo aborda o contexto, a motivação e os objetivos da atual proposta, além da estrutura em que se encontra este trabalho.

1.1 Contexto, Motivação e Objetivos

A dificuldade dos deficientes visuais em se locomoverem de forma segura, e partindo de estudos sobre os diversos trabalhos na área, o presente trabalho tem como objetivo principal propor um sistema com algoritmos eficientes para a navegação de cegos em recintos internos, através de uma solução desenvolvida em ambientes de programação abertos com o objetivo de reduzir custos de forma eficaz.

De acordo com o Censo realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE) em 2010, 18,8% da população brasileira possui algum tipo de deficiência visual, totalizando 35,8 milhões de brasileiros. Dessas pessoas com deficiência visual, ainda segundo o próprio Censo 2010, 65% têm uma renda máxima de dois salários mínimos. (“IBGE - Instituto Brasileiro de Geografia e Estatística”, 2010)

O principal objetivo do trabalho proposto é a implementação de um método capaz de fornecer dados suficientes para que uma pessoa portadora de algum tipo de deficiência visual, seja capaz de navegar em um ambiente que lhe informará um caminho sem obstáculos.

O sistema deverá mapear o recinto verificando os possíveis caminhos, indicando ao usuário qual a melhor rota, sem obstáculos, enquanto o usuário se movimenta.

A preocupação com o baixo custo é um dos requisitos do sistema, tornando o sistema acessível ao público. A utilização de linguagens de programação livres e equipamentos de baixo custo é o princípio da implementação da proposta, bem como a implementação de algoritmos eficientes.

O presente trabalho está relacionado a um projeto mais abrangente intitulado “Projeto Ouvir para Olhar” (Pedrino; Hospital, 2015), sendo o principal objetivo desenvolver ferramentas, métodos e sistemas capazes de auxiliar a navegação em ambientes diversos por pessoas portadoras de algum tipo de deficiência visual, ou de sistemas robóticos de navegação. A Figura 1.1 mostra a distribuição do projeto “Projeto Ouvir para Olhar”.

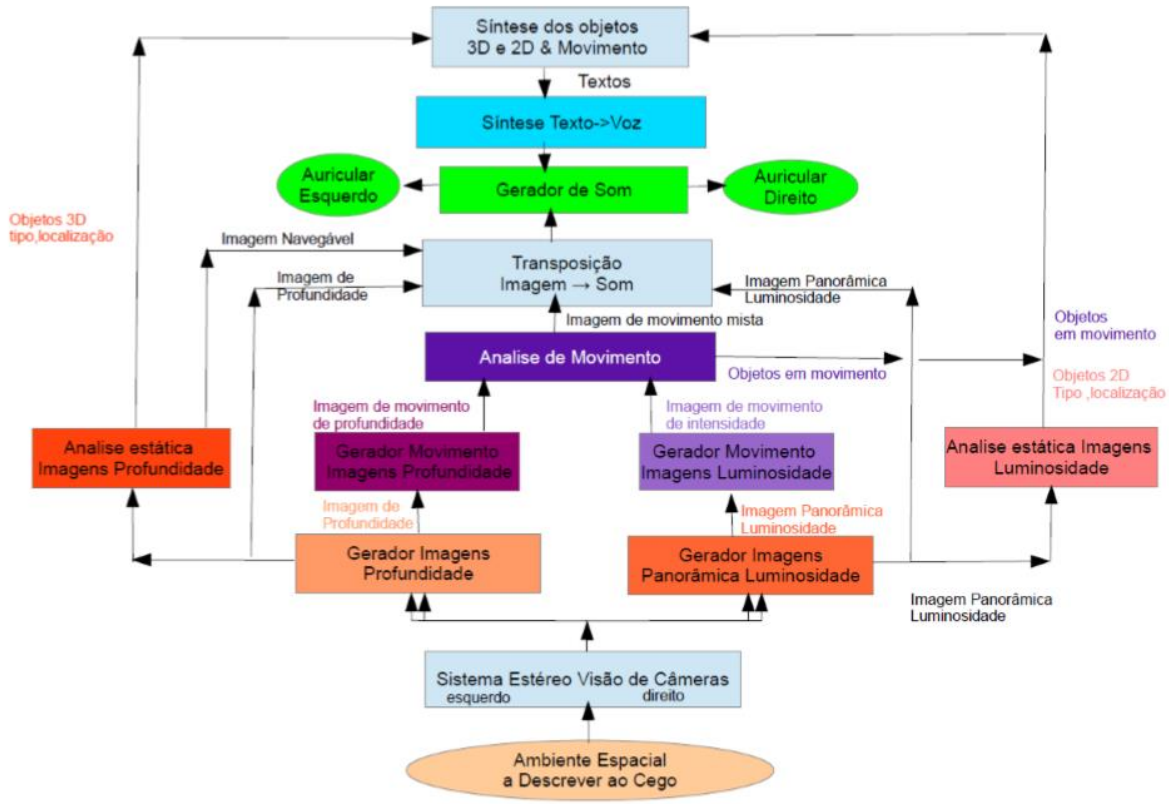


Figura 1.1 - Diagrama em blocos do “Projeto Ouvir para Olhar”.

1.2 Organização do Trabalho

O presente trabalho está organizado em cinco capítulos: o primeiro capítulo é a introdução contendo a motivação e organização da pesquisa. No segundo capítulo

está o estudo teórico necessário à construção da proposta, e, como itens fundamentais, estão o estudo sobre a visão humana, sobre a visão estéreo, além do estudo sobre processamento de imagens de visão estéreo como a geometria epipolar, a retificação estéreo, a calibração e a disparidade. O terceiro capítulo tem o objetivo de apresentar alguns projetos desenvolvidos utilizando visão estéreo para auxiliar a navegação de pessoas portadoras de deficiência visual. Em seguida, no quarto capítulo, serão apresentados a metodologia para o desenvolvimento deste trabalho e os resultados dos testes realizados no método proposto. O último capítulo apresenta as conclusões extraídas do presente projeto, bem como os pontos positivos e sugestões para futuros projetos.

Capítulo 2

VISÃO ESTÉREO E PROCESSAMENTO DIGITAL DE IMAGENS

Neste capítulo estão os estudos realizados sobre a área de processamento de imagens necessários para o desenvolvimento do trabalho, iniciando-se pelas considerações iniciais e evoluindo para o estudo da visão humana, passando pelo funcionamento da visão estéreo, processamento de imagens, geometria epipolar, retificação estéreo, disparidade estéreo, técnicas de detecção e extração de características, linguagem Python, biblioteca OpenCV, câmera de visão estéreo.

2.1 Considerações Iniciais

O propósito deste capítulo é realizar uma fundamentação teórica dos conceitos e técnicas a partir da literatura pesquisada para o desenvolvimento do projeto de mestrado.

2.2 Visão Humana

Os humanos, assim como os animais, possuem a visão como um dos sentidos essenciais para conseguir obter a máxima experiência do mundo externo. O olho humano é, essencialmente, um órgão esférico, contendo uma abertura circular frontal, a pupila, por onde entram os raios de luz. A pupila é recoberta por uma

membrana transparente denominada córnea, que constitui a parte transparente da membrana que cobre a superfície do olho (TOMMASELLI, 2007).

A imagem se forma a partir da luz que passa pela córnea, atingindo o cristalino. A íris é a parte colorida do olho, que é responsável pela variação de diâmetro da pupila, controlando a quantidade de luz que entra no globo ocular. Músculos apoiam o controle do foco pela sua contração, pressionando o cristalino e podendo alterar sua curvatura para que a imagem possa chegar até a retina onde é convertida em impulsos nervosos transmitidos ao cérebro através do nervo ótico. (TOMMASELLI, 2007)

Segundo (TOMMASELLI, 2007), a visão obtida por dois olhos, chamada de visão binocular, ajuda com segurança na percepção das relações espaciais de profundidade, naturalmente existente entre os diversos tipos de objetos que podemos encontrar numa cena captada pelos olhos humanos. O sentido tridimensional que o cérebro produz é o resultado da fusão das duas imagens captadas em cada retina, as quais estão ligeiramente deslocadas pela distância entre os olhos.

2.3 Visão Estéreo

Em visão computacional, a visão estéreo é o ramo que analisa o problema da reconstrução da informação tridimensional de objetos, a partir de um par de imagens capturadas simultaneamente, mas que possuem um pequeno deslocamento entre si (FUSIELLO; TRUCCO; VERRI, 2000). Esta é razão de os sistemas de câmeras tridimensionais comerciais imitarem o sistema visual humano, mantendo uma distância entre as câmeras de aproximadamente 6,5 cm – a mesma encontrada nos olhos humanos (SEUBA, 2009).

Segundo Seuba (2009), outro ponto importante para sistemas de processamento de imagens de visão estéreo é o campo visual, como no exemplo dos olhos humanos, o campo visual é o espaço que um olho é capaz de capturar encarando um ponto. Cada olho tem um campo visual diferente, e juntos produzem a visualização em três dimensões conforme a Figura 2.1.

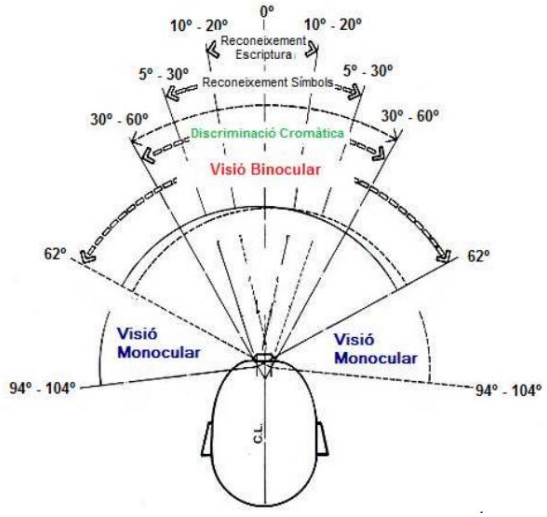


Figura 2.1 - Visão Monocular e Binocular (SEUBA, 2009)

A Figura 2.1 mostra um diagrama representando o campo visual horizontal. Existem áreas para considerar: primeiro, a área de visão monocular onde não é possível obter a sensação de profundidade. A segunda, responsável pela sensação profundidade, é obtida pela área de visão binocular (SEUBA, 2009).

Para um dispositivo que pretende obter a visão estéreo, o primeiro passo é extrair características das imagens observadas por meio de duas câmeras, e, então, efetuar a fusão binocular dessas características com dados geométricos para a recuperação da posição tridimensional do ambiente em observação, simulando o sistema visual humano (MADALENA; GOMES, 2012).

2.4 Processamento Digital de Imagens

As aplicações resultantes dos estudos da área de processamento digital de imagens contribuem para dois grandes grupos de interesse:

(1) Interpretação humana: através de imagens pictóricas aprimoradas

(2) Análise automatizada pelo computador de informações extraídas de uma determinada cena (FILHO; VIEIRA NETO, 1999).

Mas existe uma ampla gama de aplicações nas diversas áreas da rotina humana. Na medicina, os avanços na área de processamento digital de imagens vêm facilitando a interpretação de imagens produzidas por equipamentos como os

de raios *X*. Na área da biologia, a contagem automática do número de células pode ser obtida através do processamento de imagens geradas por microscópios. Imagens geradas por satélites são processadas auxiliando os diversos trabalhos realizados na área da geografia, geoprocessamento, meteorologia, entre outras (PEDRINO, 2003).

Para um sistema de processamento de imagens, segundo Filho e Vieira Neto (1999), alguns elementos básicos são necessários e podem ser visualizados na Figura 2.2.

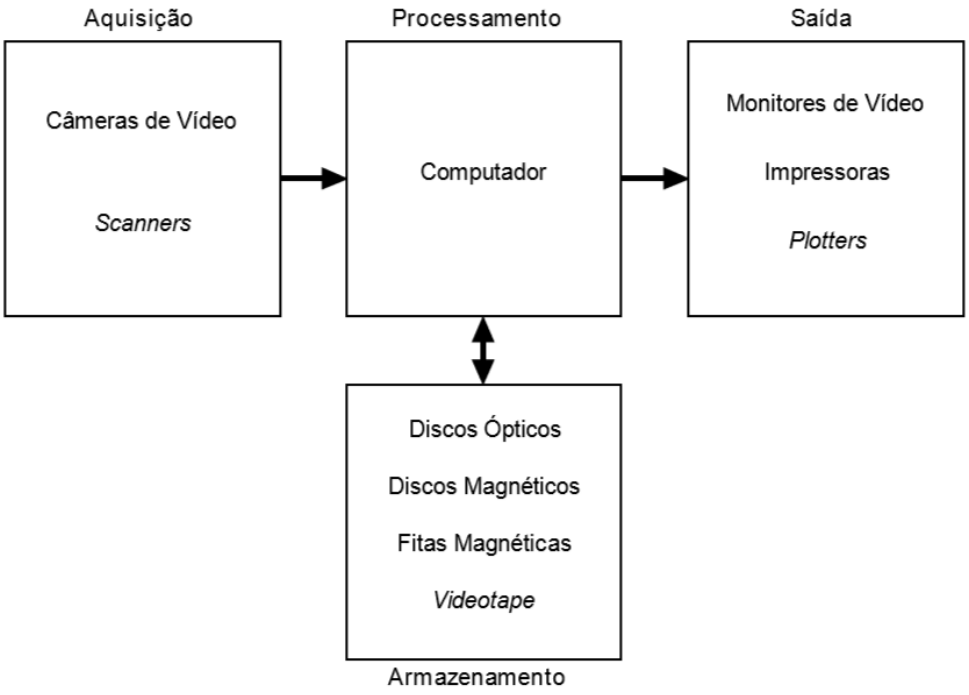


Figura 2.2 - Elementos de um sistema de processamento de imagens (FILHO; VIEIRA NETO, 1999)

A Figura 2.2 descreve sistemas, desde aqueles de baixo custo até sofisticadas estações de trabalho utilizadas em sistemas para processamento de imagens. As principais operações podem ser visualizadas: aquisição, processamento, armazenamento e exibição.

Na aquisição, é necessária a captação de uma faixa de energia no espectro eletromagnético através de um dispositivo físico capaz de transformar o nível de energia captado em sinal elétrico de forma proporcional. Nessa etapa, o dispositivo envolvido consegue através do digitalizador converter o sinal elétrico-analógico em informação digital.

O processamento fica expresso pelos procedimentos em forma algorítmica, que na maioria das vezes são implementadas via *software*. O primeiro passo nessa etapa é a correção de imperfeições da imagem resultante da aquisição, como pixels ruidosos, contraste e/ou brilho inadequado, bem como outras correções necessárias de acordo com o domínio do problema. No caso do projeto proposto, é necessária também a segmentação da imagem para encontrar os objetos de interesse e, destes, obter as suas características. O último passo nessa etapa é o reconhecimento desses objetos, extraídos através de suas características processadas por seus descritores. Para que estas tarefas sejam executadas deve existir uma base de conhecimento sobre o domínio do problema. De acordo com Filho e Vieira Neto (1999), o uso de hardware especializado para processamento de imagens é fundamental quando se encontra limitações no dispositivo responsável central, como, por exemplo, a baixa transferência de dados dos barramentos envolvidos podendo ser totalmente intoleráveis.

O armazenamento é um dos desafios de um projeto de processamento de imagens pela quantidade de *bytes* necessários. Pode ser dividido em três categorias: armazenamento de curta duração – armazenado em memória *RAM (Random Access Memory)*, armazenamento de massa para recuperação rápida de imagens e arquivamento de imagens para recuperação futura.

A exibição é realizada através de dispositivos como os monitores e impressoras capazes de reproduzirem a imagem captada (FILHO; VIEIRA NETO, 1999).

2.5 Geometria Epipolar

Geometria Epipolar, segundo Aires (2010), é a relação geométrica da interseção entre os planos de imagens e o conjunto de planos epipolares, considerando a linha da reta de base como ponto de interseção.

Um exemplo é a captura por duas câmeras de imagens sobre uma mesma cena, cada uma com o seu centro de projeção de forma não coincidentes. Cada par de imagens que estas câmeras capturam possuem duas perspectivas diferentes da

mesma cena estática. A Geometria Epipolar estabelece uma relação entre essas duas vistas capturadas sobre essas condições (SANTOS, 2012).

Segundo Fernandez (2009), a geometria epipolar é independente da estrutura da cena e depende somente dos parâmetros internos das câmeras e da posição relativa entre elas. A definição da posição relativa entre as câmeras é dada através da transformação composta de uma matriz de rotação e de um vetor de translação que permite referenciar o sistema de coordenadas da primeira para a segunda câmera. A matriz de rotação e o vetor de translação são considerados os parâmetros extrínsecos que relacionam o sistema de visão estéreo.

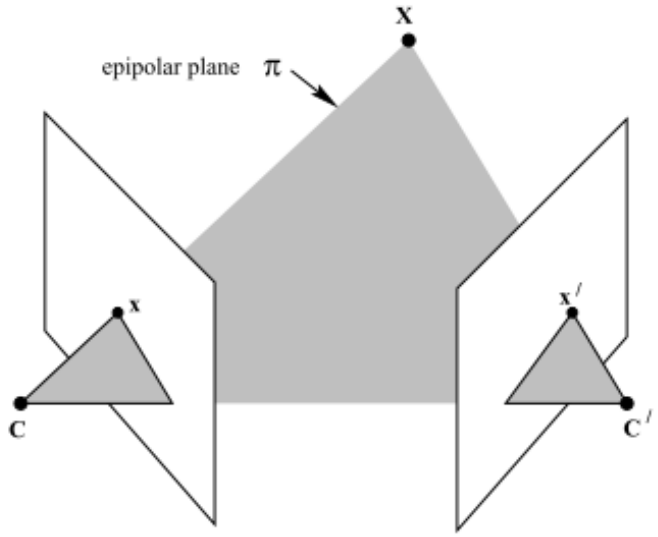


Figura 2.3 - Correspondência entre pontos utilizando geometria epipolar. Duas câmeras indicadas pelos seus centros C e C' e planos de imagem. O ponto de observação tridimensional é representado por x, que é representado nas câmeras pelas imagens x e x' em um plano π comum. (HARTLEY; ZISSERMAN, 2004)

2.6 Calibração

A calibração de câmeras é o processo que leva em consideração diversas constantes que definem as propriedades geométricas associadas às câmeras de acordo com o domínio do problema (GUERRA, 2004). Para definir a calibração, é necessário estimar os parâmetros intrínsecos e extrínsecos. Segundo Fernandez (2009), a calibração de câmera é o processo realizado para calcular os valores dos parâmetros intrínsecos, extrínsecos, além dos coeficientes de distorção da lente.

2.6.1 Parâmetros Intrínsecos:

São parâmetros associados a sua construção interna, em que quase todos não dependem da posição e orientação da câmera no espaço. O único parâmetro intrínseco que influencia a posição e a orientação é o de projeção perspectiva, que é o comprimento do cone de projeção (AIRES, 2010).

2.6.2 Parâmetros Extrínsecos:

São parâmetros da câmera associados à orientação e ao posicionamento do sistema de coordenadas da câmera em relação ao sistema de coordenadas global (AIRES, 2010).

2.6.3 Aquisição de Dados para a Calibração:

Nessa etapa existem diferentes métodos e técnicas utilizados na captura de dados para entrada, que serão posteriormente utilizados para se calcular os parâmetros de calibração das câmeras.

Os métodos e técnicas existentes, segundo Fernandez (2009), são compostos por diferentes algoritmos da área de processamento de imagem e reconhecimento de padrões, que são utilizados em conjunto com o uso e a definição de alguns tipos de padrões sintéticos, os quais auxiliam a captura de pontos de referência, também conhecido como marcadores.

Os algoritmos de processamento de imagem têm o objetivo de ressaltar as características inerentes do padrão escolhido, o que possibilita isolá-lo na imagem de outros objetos presentes. Após o isolamento do objeto, algoritmos de reconhecimento de padrão, segundo Fernandez (2009), se encarregam de extrair e identificar individualmente os marcadores contidos em um padrão.

O uso de um padrão possibilita a captura de uma amostra grande dos pontos de referência reconhecidos pelas câmeras em todo o espaço de visão capturada.

De acordo com Fernandez (2009), diferentes tipos de padrões encontrados na bibliografia sobre calibração de câmeras propõem cumprir alguns requisitos:

- Uma estrutura nítida, para diferenciação de outros objetos semelhantes, deve ser definida para evitar erros de, por exemplo, falsos positivos no reconhecimento de padrão.
- Para possibilitar diferentes vistas do mesmo objeto, esse deve possuir um formato que viabilize sua manipulação e movimentação no espaço de rastreamento. A captura por diferentes pontos de vistas representará um maior número de pontos de referência corretamente identificados e espalhados no campo visual das câmeras.
- Para permitir a recuperação de várias informações em relação aos mesmos pontos de referência, a quantidade e a distribuição dos pontos contidos no padrão devem ser levados em consideração. As identificações em grupo dos pontos de referência determinam um formato único do padrão, além de distâncias, ângulos e outras medidas que possam ser recuperadas a partir do prévio conhecimento da distribuição tridimensional desses pontos de referência, formando o padrão.

2.6.4 Padrões utilizados na calibração:

Os padrões de calibração têm o objetivo permitir o reconhecimento de pontos de referência a partir da captura, sobre um determinado formato, de algumas características.

A partir de uma análise e de um processamento de diversas vistas capturadas do padrão são extraídos os parâmetros para a calibração das câmeras.

2.6.4.1 Padrão Planar 2D:

Considera-se um padrão planar 2D aquele cujos pontos de referência se encontram sobre uma superfície plana 2D. O trabalho de Wang, Li e Zheng (2010) mostra a utilização de um padrão planar 2D para a calibração de câmeras.

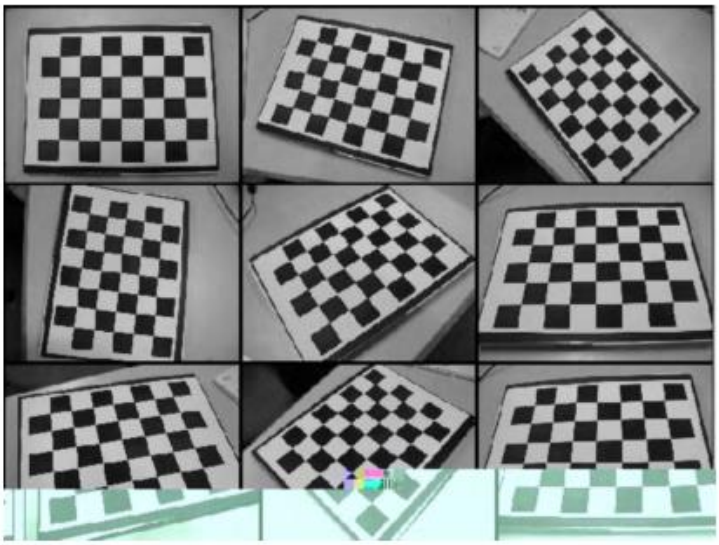


Figura 2.4 - Calibração de câmera por (WANG; LI; ZHENG, 2010)

2.6.4.2 Padrão Unidimensional 1D:

O padrão unidimensional foi proposto inicialmente por Borghese e Cerveri (2000). De acordo com a Figura 2.5, o padrão é composto por dois marcadores esféricos acoplados nos extremos de uma vara metálica. O trabalho deBorghese e Cerveri (2000) propôs esse padrão como um novo método de calibração de sistemas de câmera estéreo.

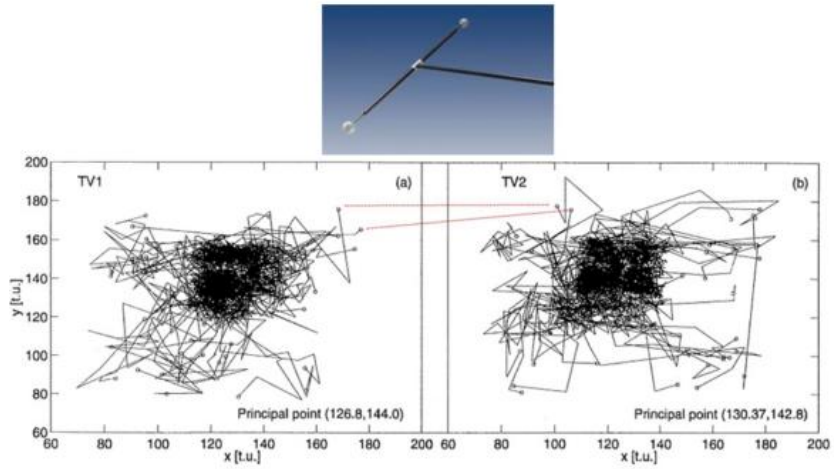


Figura 2.5 – Sistema de Calibração Unidimensional (ALBERTO BORGHESE; CERVERI, 2000)

2.6.4.3 Padrão Adimensional 0D:

O padrão adimensional foi inicialmente apresentado por Svoboda, Martinec e Pajdla (2005), sendo proposto um método de calibração para múltiplas câmeras. A diferença é a utilização de um único marcador para o reconhecimento dos parâmetros, conforme a Figura 2.6.

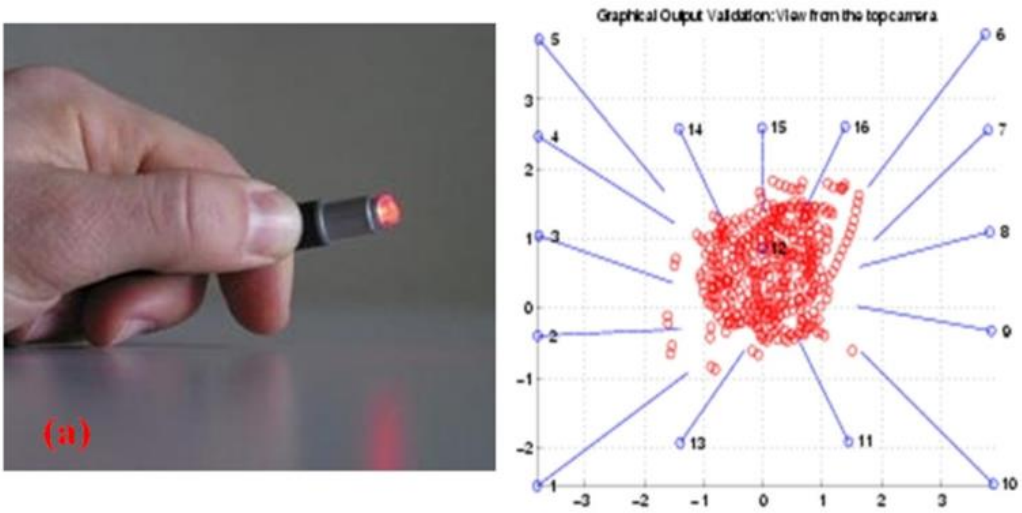


Figura 2.6 - Padrão adimensional proposto por (SVOBODA; MARTINEC; PAJDLA, 2005)

2.7 Retificação Estéreo

Posto que a geometria epipolar reduza o problema de correspondência ao espaço unidimensional, o uso da retificação estéreo busca tornar as linhas epipolares paralelas às linhas da imagem (NOGUEIRA; TOZZI, 1998) .

Segundo Stivanello (2008), observando a restrição epipolar, dado um ponto em uma imagem, pode-se encontrar seu ponto correspondente em uma outra imagem sobre uma linha específica reduzindo a complexidade do problema de correspondência entre os pontos. A geometria epipolar permite reduzir um

processamento essencialmente 2D, buscando pontos correspondentes em toda a imagem, a uma busca 1D, limitando-se a uma linha específica.

A retificação estéreo é definida pela transformação de cada imagem de forma que os pares de linhas epipolares fiquem colineares e paralelas ao eixo *x* da imagem. Nas novas imagens obtidas pelo processo de retificação, as linhas epipolares são paralelas ao eixo *x*, não existindo disparidade no eixo *y*.

De acordo com Fusiello, Trucco e Verri (2000), para reduzir o problema de correspondência, a geometria epipolar encontrada em um par de câmeras pode-se utilizar da transformação do par de imagens estéreo por transformações de projeções apropriadas, conforme Figura 2.7.

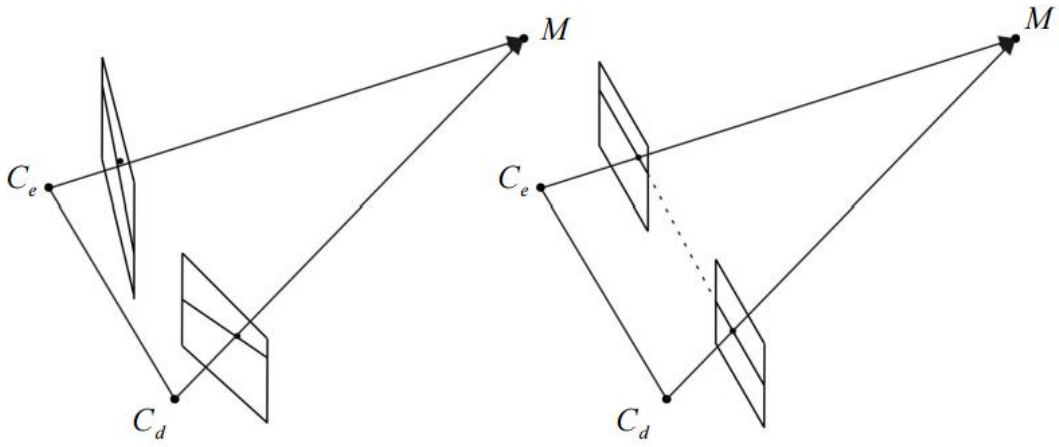


Figura 2.7 - Transformação da configuração estéreo pela retificação (STIVANELLO, 2008)

2.8 Disparidade Estéreo

Um sistema de visão estéreo básico, possui duas lentes horizontalmente deslocadas que captam simultaneamente duas imagens similares, alterando ligeiramente o ponto de vista capturado do ambiente. Ao processar a diferença de posição de um ponto entre uma imagem e outra é possível estimar sua profundidade relativa, semelhante a maneira como a visão humana proporciona a noção de profundidade (MENDES; WOLF, 2011). A Figura 2.8 ilustra a geometria de um sistema estéreo padrão.

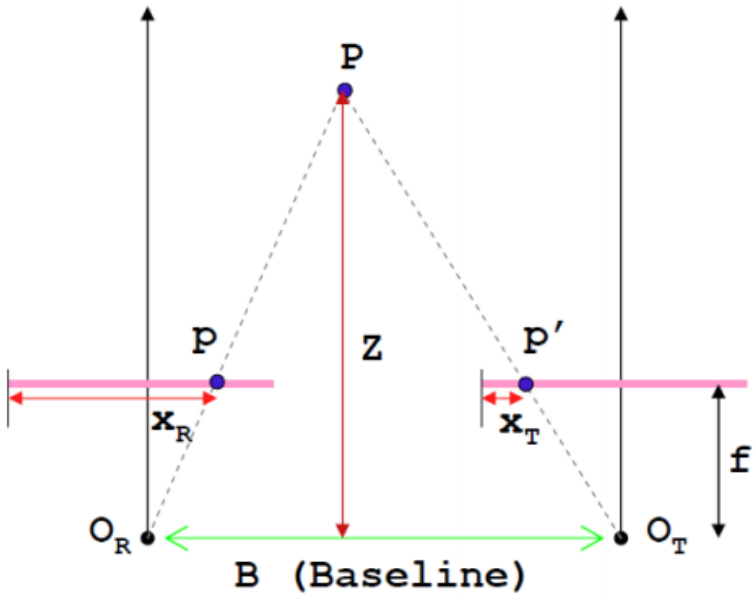


Figura 2.8 - Sistema padrão de visão estéreo com duas câmeras e distância focal *f*, lentes deslocadas por uma distância *B*, deslocamento base são representados por *Xr* (*r – reference)* e *X*t (*t* target). (MATTOCCIA, 2013)

Segundo Souza (2012), a disparidadeestéreo pode ser entendida como a diferença entre as coordenadas de imagem de um ponto no mundo capturadas por duas câmeras do sistema estéreo. Pode ser entendido como uma transformação projetiva tridimensional do espaço 3D.

Para a estimação da disparidade, ou seja, a distância do eixo *x* dos pontos correspondentes *p* e *p'* é necessária a utilização de algoritmo de correspondência ou método estéreo com objetivo de encontrar os pontos correspondentes entre as duas imagens. A procura por pontos correspondentes, segundo Mendes e Wolf (2011), possui um alto custo computacional e a diminuição da área de busca é essencial para o controle do processamento. A limitação da busca pelo eixo horizontal traz um ganho significativo de tempo de processamento. Mas tal limitação exige câmeras perfeitamente calibradas e retificadas para a correção de distorções e alinhamentos entre os pares de imagens. A Figura 2.9 ilustra o fluxo na utilização de uma câmera estéreo proposta no presente trabalho.

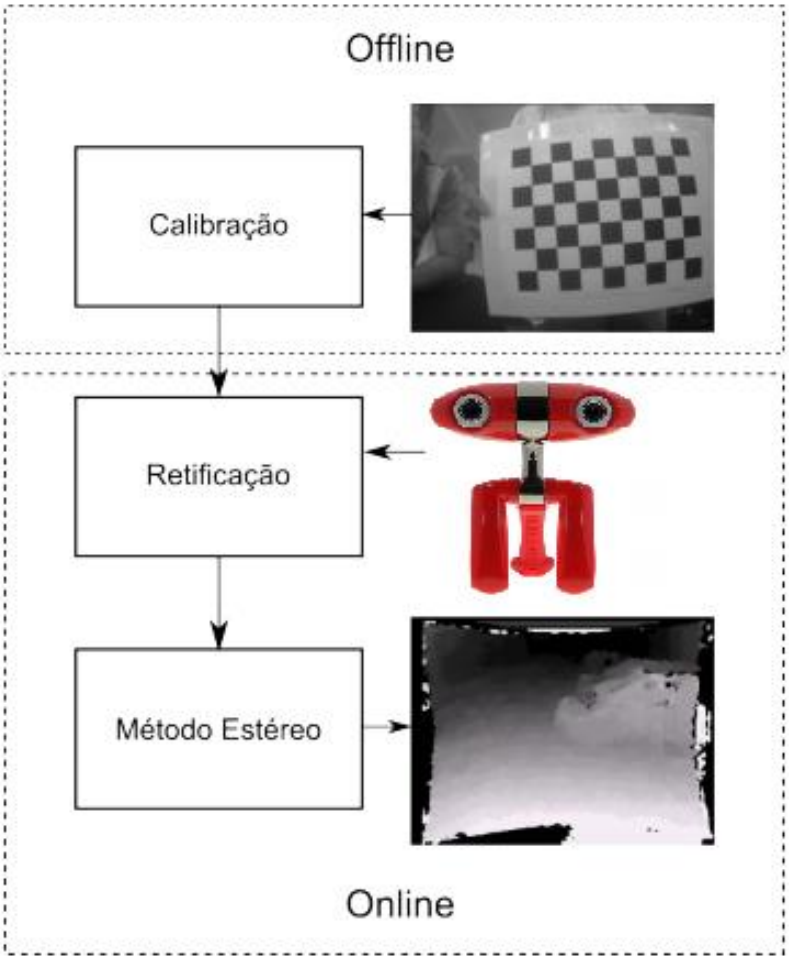


Figura 2.9 - Fluxo de utlização de câmera estéreo.

Partindo de imagens já retificadas, segundo Nogueira e Tozzi (1998), a disparidade (d) para um par de pontos homólogos pode ser determinada pela equação 1:

$$d = x_E - x_D$$

Equação 1

Em um sistema estéreo, de acordo com a Figura 2.8, sabendo-se a distância focal e o deslocamento base é possível por triangulação saber a distância relativa do ponto *P* à base da câmera. A equação 2 mostra como o cálculo é realizado, sabendo-se que *Z* é a distância, *B* o deslocamento base, *f* a distância focal e *d* a disparidade.

$$Z = \frac{B \cdot f}{x_r - x_t} = \frac{B \cdot f}{d}$$

Equação 2

A profundidade de um ponto associado no espaço objeto pode ser dada pela fórmula da equação 3:

$$Z = \frac{b.f}{d}$$

Equação 3

Na fórmula, *Z* representa a profundidade, *b* a base e *f* a distância focal.

De acordo com Stivanello, (2008), pode-se concluir que a profundidade é inversamente proporcional a disparidade. O resultado do cálculo de disparidade é dado pela busca em duas imagens de um ponto correspondente. O valor da disparidade será armazenado como o valor de intensidade na posição do ponto da imagem de referência em uma terceira imagem, conhecida como mapa de disparidade.

2.9 Segmentação

A segmentação consiste em técnicas para extração de informação de uma imagem. O termo vem do inglês "*image segmentation*", criado nos anos 80. É uma área de constantes estudos por compreender a base de todo o processamento de informação em uma imagem.

A segmentação consiste em dividir a imagem em regiões diferentes, possibilitando a análise através de algoritmos específicos em busca de informações mais precisas. Diversas técnicas são utilizadas para obter a segmentação, não existindo nenhuma genérica para diversas imagens (ALBUQUERQUE, 2000).

2.10 Propriedades de uma Imagem Digital

Uma imagem digital é uma imagem *f(x,y)* discretizada espacialmente e em amplitude. A imagem digital pode ser vista como uma matriz cujas linhas e colunas identificam um ponto na imagem que possui um valor equivalente ao nível de cinza naquele ponto. As letras *p* e *q* são utilizadas para representar um pixel, assim como a letra *S* representa um subconjunto de *f(x,y)* (FILHO; VIEIRA NETO, 1999). De acordo com Gonzalez e Woods (2002), qualquer imagem digital pode ser representada por uma função bidimensional de intensidade de luz *f*(*x,y*), onde *x* e *y* compreendem as coordenadas espaciais. Por exemplo, para imagens em nível de cinza, o valor de *f* é proporcional à intensidade do tom de cinza naquele ponto.

2.11 Técnicas de Detecção de Bordas de *Canny*

Os estudos de detecção de bordas sobre imagens digitais, segundo Filho e Vieira Neto (1999), desafia pesquisadores da área há muito tempo com constantes pesquisas sobre novas técnicas, principalmente sobre imagens consideradas difíceis.

A definição de borda pode ser dita como a fronteira entre duas regiões sobre a imagem cujo níveis de cinza são razoavelmente diferentes. Outros autores associam a borda à descontinuidade de luminosidade de uma determinada cena, onde as informações de textura e cor são mais relevantes (FILHO; VIEIRA NETO, 1999).

Segundo Canny (1986), a detecção de bordas ajuda a simplificar o processamento de imagens por reduzir os dados que são processados mantendo assim a informação estrutural útil.

Alguns procedimentos, de acordo com Canny (1986), são essenciais em um algoritmo de detecção de bordas:

- ➔ Baixa taxa de erros: uma detecção de bordas deve manter a estrutura básica da imagem, detectando todas as bordas;

- ➔ Manter a localização: o algoritmo deve manter a localização original dos pixels detectados como borda na imagem gerada;
- ➔ Evitar duplicação de pixels de bordas: o processo deve marcar os pixels detectados para evitar duplicação de bordas.

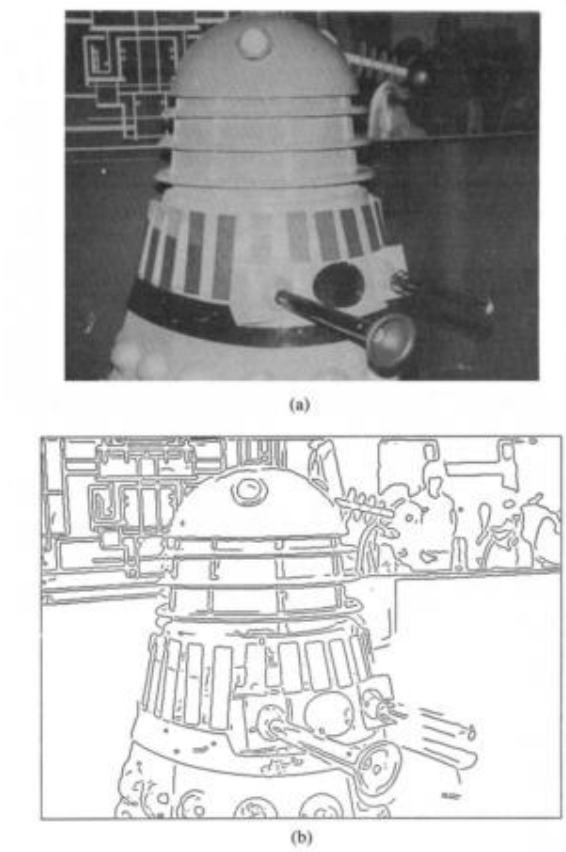


Figura 2.10 - Detecção de bordas (Algoritmo Canny) (CANNY, 1986)

2.12 Limiarização

A técnica da limiarização consiste em separar as regiões de uma imagem para a extração de duas classes: o fundo e o objeto (FILHO; VIEIRA NETO, 1999). Outra denominação para esse processo é o de binarização, uma vez que a imagem produzida pelo método é uma imagem em preto e branco. A forma mais simples de limiarização, segundo Filho e Vieira Neto(1999) seria a bipartição do histograma, fazendo a conversão dos pixels cujo tom de cinza é maior a um valor de limiar (T)

em branco e os pixels restantes convertidos em preto. A figura 2.11 ilustra o processo:

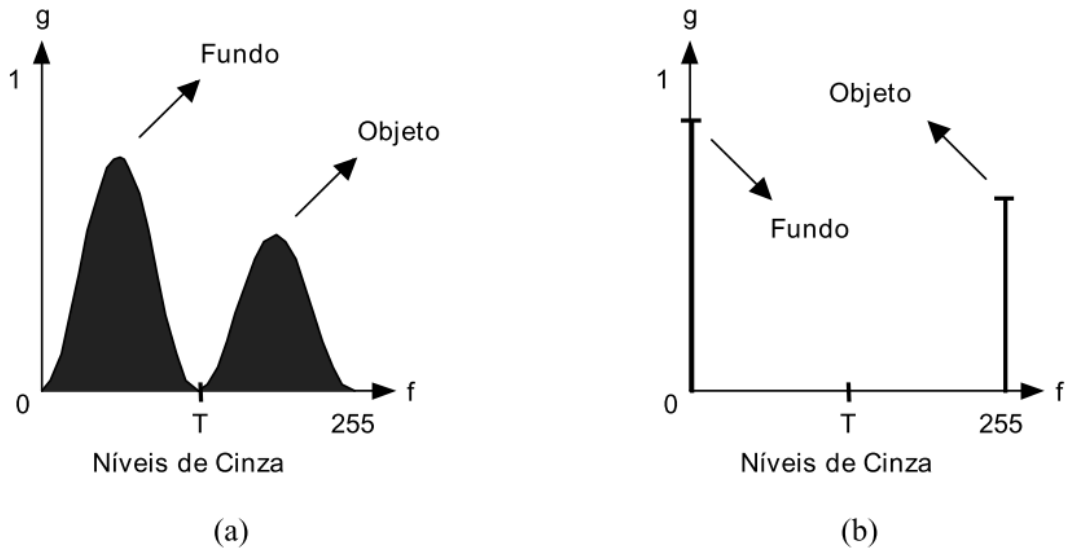


Figura 2.11 - Limiarização de uma imagem monocromática utilizando um limiar T. (a) histograma original, (b) histograma da imagem binarizada (FILHO; VIEIRA NETO, 1999)

Capítulo 3

TRABALHOS RELACIONADOS

Apresentação dos trabalhos e estado da arte relacionados com a proposta desse trabalho de mestrado.

3.1 Considerações Iniciais

Neste capítulo estão relacionados os principais trabalhos relacionados à navegação de pessoas portadoras de deficiência visual. São levantados os pontos positivos e sugestões de melhoras para cada projeto.

3.2 Trabalhos Relacionados

O trabalho de Lin, Han e Hahn (2011) propõe a criação de um sistema utilizando apenas uma câmera para o auxílio a navegação de cegos em ambientes abertos, como em uma calçada ao ar livre. Eles propõem um algoritmo de transformação *top-view* da imagem da câmera para obter as posições dos possíveis obstáculos, como árvores, postes e outros pedestres. Além disso o sistema faz uma estimativa dos possíveis caminhos livres. Cada caminho recebe uma pontuação e a mais alta faz relação ao caminho mais seguro, com menos obstáculos (LIN; HAN; HAHN, 2011).

O ponto forte desse projeto é o tratamento da imagem através de uma câmera, diminuindo a complexidade na criação do equipamento. Mas o sistema proposto não estabelece uma comunicação sonora com o cego sobre a informação do melhor caminho a percorrer.

Figura 3.1 - Imagem fonte e resultado do tratamento proposto. (LIN; HAN; HAHN, 2011)

No trabalho de Pradeep, Medioni e Weiland (2010), entitulado “Visão Robótica para Deficientes Visuais” (*Robot Vision for the Visually Impaired*), a proposta é a criação de um equipamento de visão estéreo, que possibilita a navegação através do mapeamento do ambiente. Esse mapeamento basicamente é realizado mediante a criação de um modelo 3D através da triangulação estéreo das imagens obtidas. O sistema orienta os usuários através de motores que realizam microvibrações fornecendo dicas sobre o movimento a ser tomado. O sistema utiliza o conceito de SLAM (Localização e Mapeamento Simultâneos - *Simultaneous Localization and Mapping*), que consiste em técnicas de construção de mapas de navegação em tempo real. O que chama a atenção do projeto é a interface com o usuário através de motores de vibração, mas o trabalho falha no quesito mobilidade, uma vez que todo o equipamento fica preso ao usuário através de um colete que possui fios de alimentação e cabos de dados (PRADEEP; MEDIONI; WEILAND, 2010).

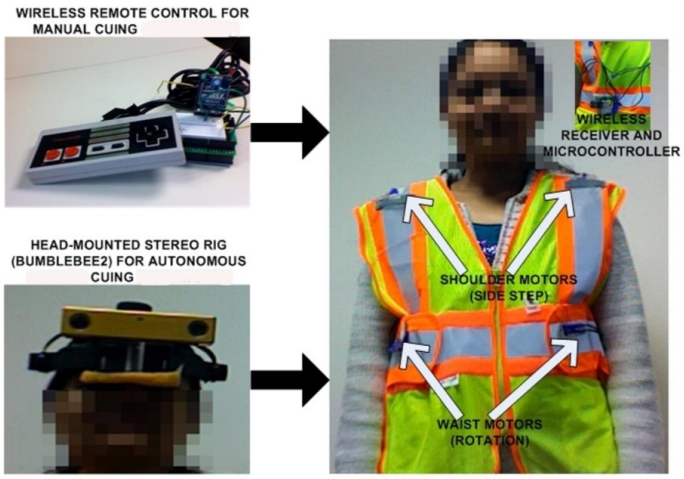


Figura 3.2 - Visão Robótica para Deficientes Visuais. (PRADEEP; MEDIONI; WEILAND, 2010)

Leung e Medioni (2014) realizaram um projeto de ajuda à navegação para cegos em ambientes diversos. O projeto utiliza um equipamento *Vuzix Wrap 920 AR* integrado com *IMU* (*Inertial Measurement Unit*) que é capaz de identificar o plano terrestre decompondo-o em um sistema 6DOF (*Six Degrees Of Freedom*). O plano terrestre é estimado em cada quadro por análise da matriz de disparidade. A vantagem do projeto é a utilização de um equipamento leve e confortável. A limitação é a qualidade das imagens geradas e o preço elevado do equipamento (LEUNG; MEDIONI, 2014).

Figura 3.3 - Utilização do Óculos Vuzix Wrap 920 AR e imagens geradas

O trabalho de Bourbakis, Makrogiannis e Dakopoulos (2013), intitulado "*A System-Prototype Representing 3D Space via Alternative-Sensing for Visually Impaired Navigation*", propõe a implementação de um protótipo de sistema para transferir por meio vibrações o espaço tridimensional mapeado. A proposta é a detecção de obstáculos e padrões de movimento na sequência das imagens para a navegação segura. São utilizadas imagens das câmeras da direita e esquerda na formação de novas imagens estéreo. Após isso, é realizada a estabilização do vídeo como uma etapa do pré-processamento. Uma etapa importante do projeto é a representação da informação processada das cenas capturadas em padrões detectáveis de vibrações aplicadas sobre o corpo do usuário para criar uma sensação tridimensional do espaço durante a navegação.

O projeto desenvolvido por Costa *et al*. (2011), tem como objetivo realizar o auxílio à navegação de uma pessoa portadora de deficiência visual através da utilização do sistema *SmartVision.* De acordo com os autores, a proposta possui como entrada do sistema diversos dados oriundos de diferentes sensores, como câmera, GPS (*Global Position System*), RFID (*Radio-Frequency IDentification*). O principal objetivo do trabalho é a criação de marcações de navegação com os dados adquiridos pelo *SmartVision* através das câmeras e dos quais é extraído o mapa de disparidade do ambiente o qual verifica a distância do usuário com as marcações. A saída é realizada por dois meios, vibração e sonora. A vibração auxilia na navegação e os pontos de interesse são informados por áudio. O trabalho desenvolvido tem como pontos positivos as saídas adequadas para o usuário.

Um algoritmo para detecção de obstáculos foi proposto por Nalpantidis, Kostavelis e Gasteratos (2009). O trabalho tem como objetivo a detecção de obstáculos em uma cena real propiciando uma resposta para a tomada de decisão de robôs autônomos. Duas fases são utilizadas para a execução do algoritmo: uma para o processamento de imagens obtidas através de um par de câmeras para visão estéreo e a fase final relacionada com a tomada de decisão para uma navegação livre de obstáculos.

Na primeira fase são aplicadas técnicas de processamento de imagens para detecção de regiões com características marcantes como superfícies texturizadas, tornando essas regiões mais evidentes e facilitando o próximo passo que é a realização da correspondência estéreo da imagem. Para a realização do realce sobre as características marcantes, é aplicado o método de Laplace para a detecção

de bordas e um núcleo Laplaciano marcando as regiões de alta intensidade. No passo seguinte, as novas imagens processadas são sobrepostas com asoriginais conforme pode ser observado na figura 3.4.

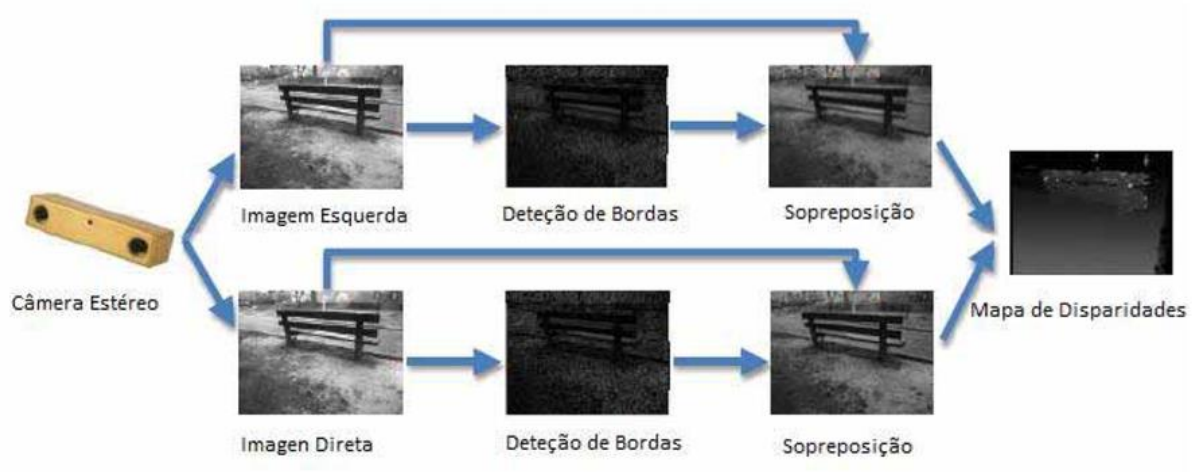


Figura 3.4 - Processamento de imagens proposto pelo algoritmo (adaptado de NALPANTIDIS; KOSTAVELIS; GASTERATOS, 2009)

Com o mapa de disparidade adquirido, o algoritmo realiza a classificação da cena em regiões navegáveis. A imagem é dividida em três partes e para cada parte são sumarizados os pixels de modo que o valor da disparidade seja maior que um limiar preestabelecido sobre experimentos para verificar qual a principal característica de imagens com obstáculos. Sobre o somatório dos pixels de cada região é realizada a saída sugerindo a navegação livre de obstáculos como direita, esquerda ou frente.

Testes foram executadas sobre o algoritmo proposto utilizando uma base de 25 imagens em um ambiente externo urbano, calculando-se as saídas esperadas, e o sistema acertou. Na figura 3.5 pode ser observado algumas imagens e a saída resultante depois da inferência sobre o algoritmo proposto.

Figura 3.5 - Imagens e saída pelo processo proposto por (adaptado de NALPANTIDIS; KOSTAVELIS; GASTERATOS, 2009)

3.3 Considerações Finais

Neste capítulo foram descritos os principais trabalhos relacionados que utilizam algum sistema de auxílio à navegação. No próximo capítulo está a descrição do sistema proposto neste trabalho, seus métodos e testes realizados.

Capítulo 4

SISTEMA DE AUXÍLIO À NAVEGAÇÃO

Apresentação de estudos sobre métodos de obtenção de disparidade estéreo bem como dos filtros utilizados para extrair os resultados esperados. Apresentação de todos os testes efetuados para encontrar os parâmetros adequados de acordo com o resultado esperado.

4.1 Considerações Iniciais

Com base nos conceitos apresentados nos capítulos anteriores, no atual serão detalhados todos os procedimentos executados durante a realização deste trabalho, tais como a apresentação dos objetivos, a apresentação de todos os testes realizados sobre os métodos utilizados e o estudo de viabilidade para o embarque do sistema em um computador portátil de baixo custo. Todos os resultados serão apresentados levando-se em consideração as saídas esperadas.

4.2 Requisitos do Projeto

Este trabalho de mestrado teve por objetivo a criação de um sistema de auxílio à navegação em ambientes internos utilizando equipamentos de baixo custo, além da realização de um estudo sobre a viabilidade da utilização de técnicas desenvolvidas sobre um computador mais acessível.

Todas as técnicas utilizadas e algoritmos implementados foram testados para fornecer uma taxa de resposta eficiente. O tempo de execução e as variações realizadas sobre os métodos serão apresentados para justificar os parâmetros escolhidos durante o projeto.

Os objetivos específicos do trabalho estão descritos a seguir:

- ➔ Estudo sobre uma câmera estéreo de baixo custo.
- ➔ Aplicação da calibração nas câmeras utilizadas.
- ➔ Encontro do melhor método estéreo para adquirir mapa de disparidade ideal para o objetivo geral do projeto.
- ➔ Implementação de um método para detecção do caminho livre com base na imagem de disparidade.
- ➔ Identificação de melhorias nos códigos implementados visando um tempo de execução satisfatório.
- ➔ Implementação um sistema capaz de identificar o caminho livre com base na imagem obtida.
- ➔ Projeto de uma saída eficiente visando a navegação de uma pessoa portadora de deficiência visual.
- ➔ Teste dos métodos em uma plataforma menor para embarque.
- ➔ Validação dos métodos propostos.

A seguir está uma breve descrição de cada etapa. Mais detalhes estão relacionados nos próximos subtópicos deste capítulo.

As etapas exibidas na figura 4.1 estão divididas em dois processos, *off-line e online*. No processo *off-line* estão o procedimento de cálculo da matriz de correção das câmeras (CMC) e o processo para se extrair dessas matrizes de parâmetros intrínsecos e extrínsecos utilizando imagens de (IEM) adquiridos em diversos ângulos e posições pela câmera estéreo e inferidos sobre o método de calibração. Esse passo será detalhado posteriormente.

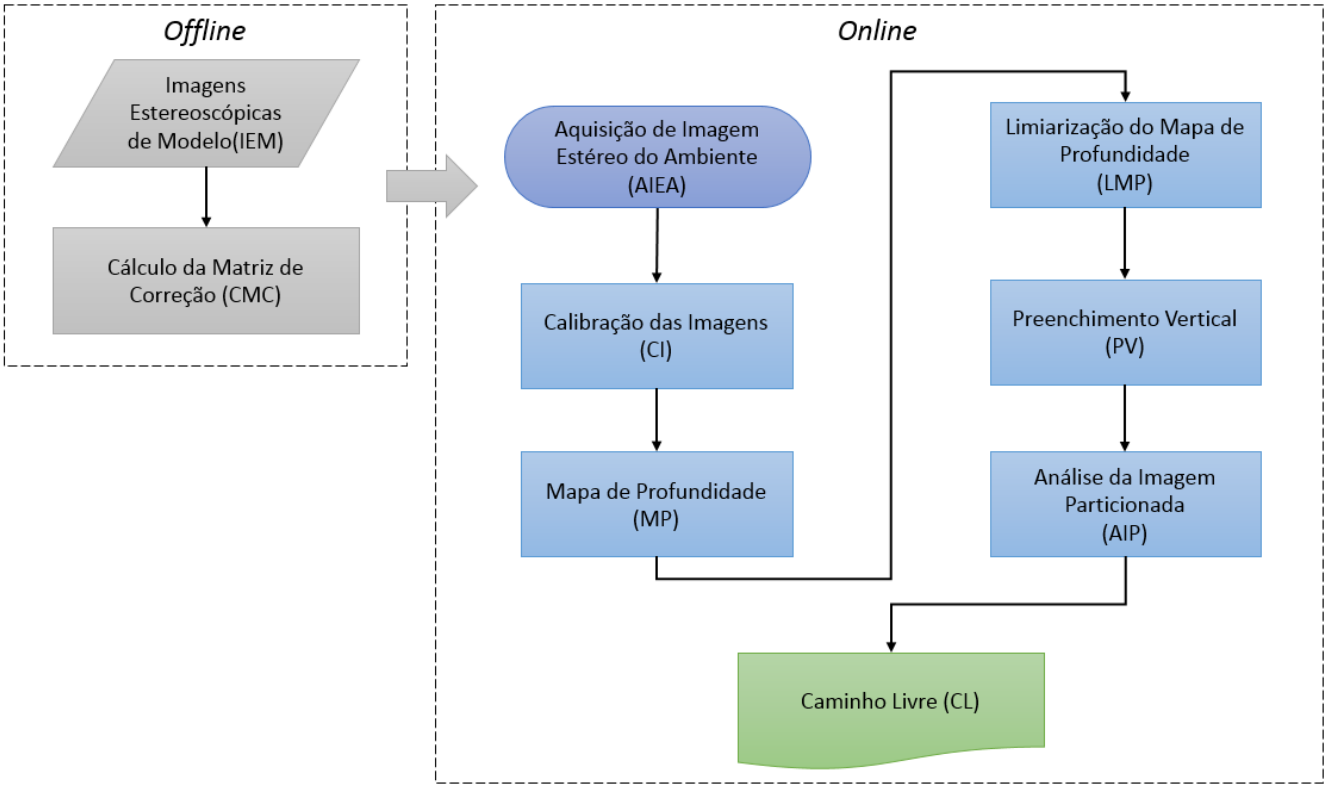


Figura 4.1 – Principais etapas do projeto

O processo *online* ocorre em tempo real durante a navegação. A primeira etapa é a aquisição de imagem estéreo do ambiente (AIEA) visando o auxílio a navegação em um ambiente interno. Parte-se do princípio que haverá iluminação natural ou artificial ideal no local, ambos propiciando condições para qualquer indivíduo sem deficiência identificar todos os objetos e obstáculos. A partir dos requisitos acima citados, serão adquiridas em tempo real, pelo par de câmeras, as imagens direita e esquerda.

O procedimento de Calibração (CI) é realizado aplicando-se as matrizes de correção sobre as imagens adquiridas que são utilizadas no próximo processo de extração do Mapa de Profundidade (MP) ou Mapa de disparidade, que é uma imagem na escala de cinza que possui os dados de profundidade de cada objeto em cena, sendo que as tonalidades mais claras indicam objetos mais próximos da câmera. Com o mapa de profundidade calculado, aplica-se a limiarização (LMP) para evidenciar somente os objetos mais próximos.

Um procedimento de preenchimento vertical (PV) é aplicado para a encontrar os quadrantes da imagem que possuem um possível caminho livre através de um

algoritmo. Finalizando o processo, a imagem é particionada em três e uma análise (AIP) é realizada para a verificação da porção que contem mais pixels brancos indicando um possível caminho livre (CL). O caminho livre será determinado pela maior área livre encontrada onde a saída deverá ser realizada indicando a direção da esquerda, centro ou direita.

A figura 4.2 descreve um possível cenário simulando um ambiente interno de uma residência com mesa, cadeiras, sofá distribuídos de forma natural em que o sistema deverá indicar o caminho livre:

Figura 4.2 – Ambiente para navegação

Foram utilizados dois equipamentos para os testes de comparação de desempenho, um PC notebook com a seguinte configuração, processador i7-4500U 2.40GHz com memória RAM de 8GB e placa de vídeo dedicada de 2GB de memória, o outro equipamento é a *Raspberry* PI 2 com processador A? 900MHz quad-core ARM Cortex-A7 CPU e memória de 1GB RAM. O objetivo é comparar a execução nos dois equipamentos, sendo que o segundo será objeto de estudo para o embarque do sistema proposto. A seguir estão descritos os procedimentos, testes e resultados obtidos durante a execução do projeto.

4.3 Ferramentas de Apoio

Para a implementação do sistema, algumas ferramentas foram utilizadas, como a Linguagem de Programação *Python* incluindo, a biblioteca de visão computacional *OpenCV*. A câmera utilizada foi a *Minoru3D* e visando um dispositivo de tamanho reduzido foi utilizado a *Raspberry* PI2. Todas essas ferramentas estão descritas a seguir.

4.3.1 Linguagem de Programação *Python*

Python é uma linguagem de programação de alto nível, interpretada, orientada a objetos de tipagem dinâmica e forte. Criada por *Guido Van Rossum* em 1991. Atualmente é gerenciado pela *Python Software Foudation*.

A linguagem *Python* vem sendo utilizada para determinadas tarefas computacionais e de visualização. Uma das principais vantagens do *Python* é possibilidade de um desenvolvimento através de pacotes que podem (estender a língua para fornecer capacidades avançadas, como o processamento de imagens digitais, processamento de sinais digitais, manipulação de matrizes e visualização (ANDERSON; PRESTON; SILVA, 2010).

Segundo Anderson, Preston e Silva (2000), a linguagem *Python* fornece uma base sólida para a construção de aplicações flexíveis, pois permitem a importação de bibliotecas criadas sobre outras linguagens como C, C++ e *Fortran*.

O *Python* é uma linguagem que se assemelha ao *MatLab* quando comparado na sua flexibilidade na construção de aplicações. Mas embora equivalentes, a linguagem *Python* é de código aberto tornando-se uma solução atraente para diversas aplicações. Pode-se afirmar por conta dessa vantagem, que as aplicações desenvolvidas em *Python* podem ser amplamente distribuídas, tornando possível uma maior colaboração dos profissionais da área (ANDERSON; PRESTON; SILVA, 2010).

4.3.2 Biblioteca OpenCV (Open Source Computing Vision)

O *OpenCV* é uma biblioteca multiplataforma desenvolvida pela *Intel* no ano 2000, totalmente livre ao uso acadêmico e comercial, para desenvolvimento de aplicações na área de Visão Computacional. *OpenCV* é liberado sobre a licença BSD (*Berkeley Software Distribution*), de código aberto. Possui interfaces C++, C, *Python* e *Java* suportando os sistemas operacionais *Windows, Linux, Mac OS, iOS,* e *Android*. O *OpenCV* foi projetado com forte foco em aplicações de tempo real (“OpenCV.org”, 2014).

4.3.3 Câmera Estereoscópica *Minoru3D*

A *Minoru3D* é uma *webcam* que filma e transmite vídeos 3D. A câmera usa técnicas de estereoscopia e anáglifo para gerar o efeito 3D. Pode ser empregada em robótica como opção para sensores de visão estéreo. Pesquisadores de *Nagoya Institute of Technology* realizaram alguns trabalhos na área de visão computacional utilizando a Minoru com a API OpenCV (“Minoru3D.com”, 2015).

Figura 4.3 - Câmera estéreo Minoru3D (“Minoru3D.com”, 2015)

4.3.4 Microcomputador *Raspberry* PI

O *Raspberry PI* é um computador do tamanho de um cartão de crédito que se conecta com um monitor de computador ou TV utilizando como entrada de dados um mouse ou um teclado padrão. Atualmente o dispositivo é utilizado em uma ampla gama de projetos educacionais (“Raspberrypi.org”, 2015).

A versão em teste neste projeto é o *Raspberry* PI 2, com as seguintes características:

Processador: A 900MHz quad-core ARM Cortex-A7 CPU

Memória:1GB RAM

Conexões:

- 4 portas USB
- 40 GPIO *pins*
- *Full HDMI port*
- *Ethernet port*
- *Combined 3.5mm audio jack and composite video*
- *Camera interface* (CSI)
- *Display interface (*DSI)
- Slot para cartão Micro SD
- *VideoCore IV 3D graphics core*

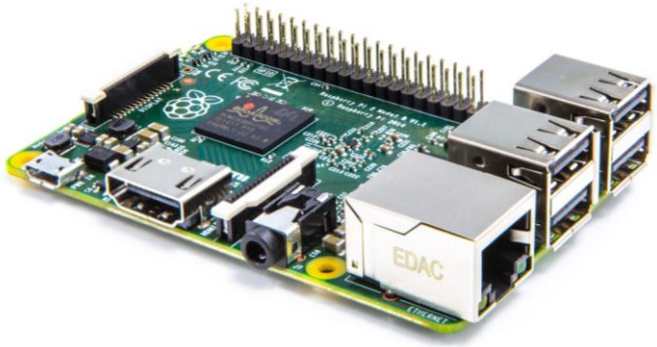


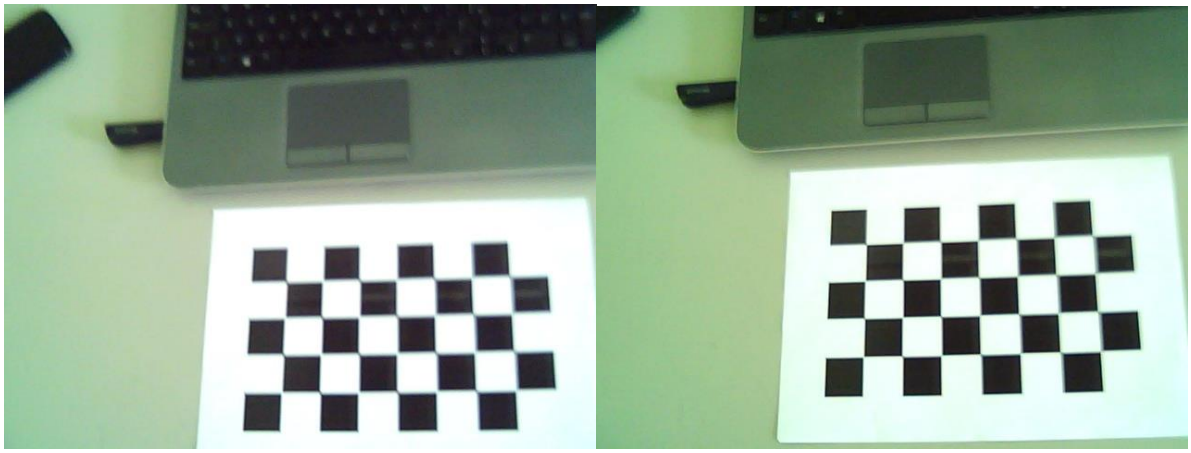
Figura 4.4 - Raspberry PI2 (“Raspberrypi.org”, 2015)

4.4 Calibração das Câmeras

A aquisição de imagens pelo sistema é uma das etapas mais importantes a considerar. De acordo com a revisão bibliográfica, a devida calibração das câmeras deve ser levada em consideração para melhorar a qualidade do mapa de disparidade. Utilizando a biblioteca *OpenCV* sobre a linguagem *Python* é possível efetuar o primeiro passo, que é a calibração estéreo.

4.4.1 Imagens Estereoscópicas de Modelo

Para a calibração, optou-se por utilizar o padrão planar 2D na forma de um *grid* ou tabuleiro de xadrez. Considerando a câmera *Minoru3D* instalada, pode-se observar na figura 4.3 algumas imagens extraídas do padrão escolhido.



(a) Esquerda **(b) Direita**

Figura 4.5 – Aquisição de imagens para calibração

Foram captadas 60 pares de imagens estereoscópicas (esquerda, direita) sendo que em cada par o padrão de calibração foi movido em diversos ângulos e distâncias com o objetivo de extrair os pontos ocultos das câmeras. A Figura 4.6 exibe a extração dos pontos pelo sistema de calibração.

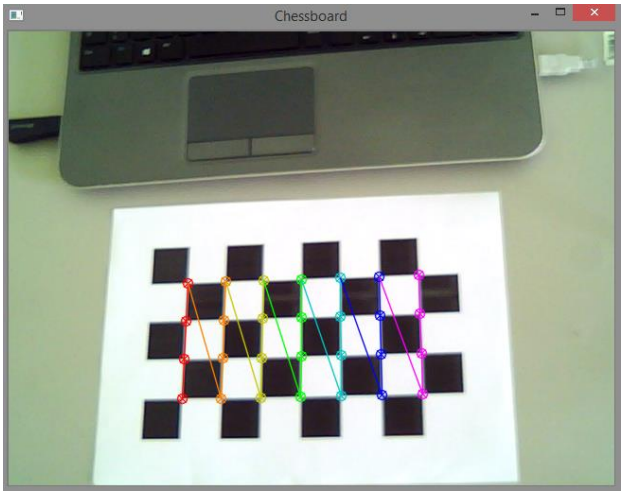


Figura 4.6 - Identificação dos pontos do padrão de calibração pelo sistema.

4.4.2 Cálculo da Matriz de Correção

Este processo é realizado previamente e apenas uma vez antes da execução do sistema, e, após extraídos os parâmetros intrínsecos e extrínsecos, esses são aplicados sobre as imagens adquiridas, resultando na devida correção das distorções.

O método da biblioteca *OpenCV* aplicado sobre as imagens extraídas para calibração foi o *cv2.findChessboardCorners()*. O método se encarrega de encontrar o padrão previamente definido, no caso sete pontos internos de largura por quatro pontos de altura e é aplicado a todas as imagens na busca de pontos ocultos. Cada ponto oculto gera um erro associado aos parâmetros intrínsecos, como distorções da lente, que é calculado e armazenado em matrizes.

4.4.3 Aquisição de Imagem Estéreo do Ambiente e Calibração das Imagens

O sistema proposto utiliza as imagens obtidas da câmera *Minoru3D* para a inferência no sistema de auxílio à navegação. A escolha da câmera foi baseada no baixo custo para a realização do projeto.

Cada par de imagens obtido em tempo real é processado e aplicado ao algoritmo para verificação do possível caminho livre como será descrito posteriormente.

Aplicando os parâmetros obtidos pela calibração, é possível lograr a retificação estéreo das câmeras sobre imagens capturadas em tempo real. O método da biblioteca *OpenCV* utilizado para a retificação estéreo é o *cv2.initUndistortRectifyMap().* Na figura 4.7 é possível observar o alinhamento obtido pelo processo de retificação entre as duas imagens capturadas.

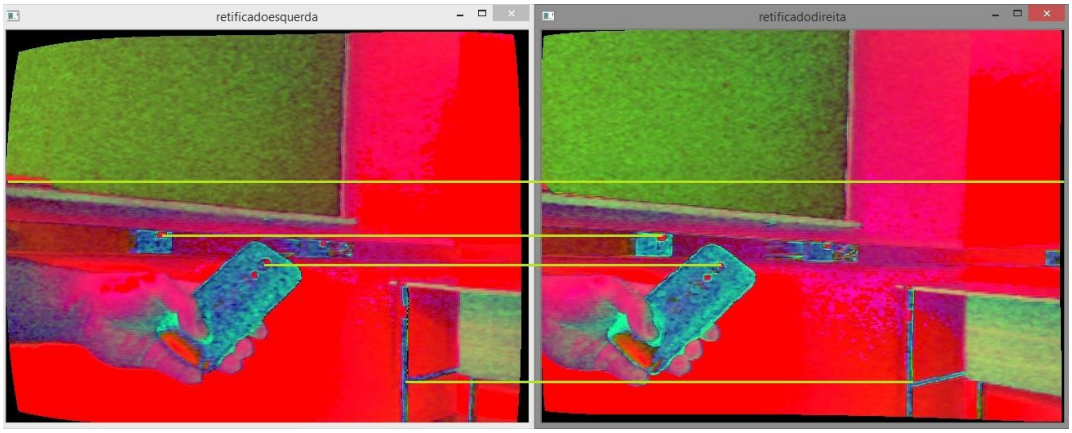


Figura 4.7 – Imagens geradas com aplicação da retificação

4.5 Mapa de Disparidade

Utilizando as funções já implementadas e incorporadas na biblioteca *OpenCV* foi possível realizar a comparação dos dois principais métodos para obtenção do mapa de disparidade, o método estéreo *semi-global block matching* (SGBM) e o algoritmo de correspondência local *block-matching* (BM). Para a realização dos testes foram consideradas pelo menos 30 pares de imagens diferentes extraídas de um ambiente interno seguindo os requisitos de iluminação e obstáculos anteriormente descritos.

Os primeiros testes serviram para definir a obtenção dos melhores parâmetros para um mapa de disparidade com uma taxa de erro mínima, ou seja, melhorar a qualidade do mapa. Para a escolha do melhor método, foram ponderados em duas arquiteturas diferentes os seguintes parâmetros:

- Resolução.
- Tipos de Imagens RGB e HSV.
- Métodos SGBM e BM.

- Tamanho do bloco de disparidade em pixels. Exemplo: um bloco de valor 5 corresponde a uma busca em lotes pela correspondência em um bloco de 5 x 5 pixels.

No primeiro teste realizado no PC foram comparados os dois métodos, tipos de imagem e resolução diferentes. Foi criado um algoritmo que verifica as disparidades com valores negativos para a obtenção da taxa de erro de disparidade. Esse método pode ser observado no trabalho dos autores Mendes e Wolf (2011).

De acordo com os gráficos apresentados nas figuras 4.8 e 4.9 pode-se observar que o método BM é um método bem menos custoso em termos de processamento, mas que a taxa de erro é bem elevada em todas as variações de tamanhos de bloco.

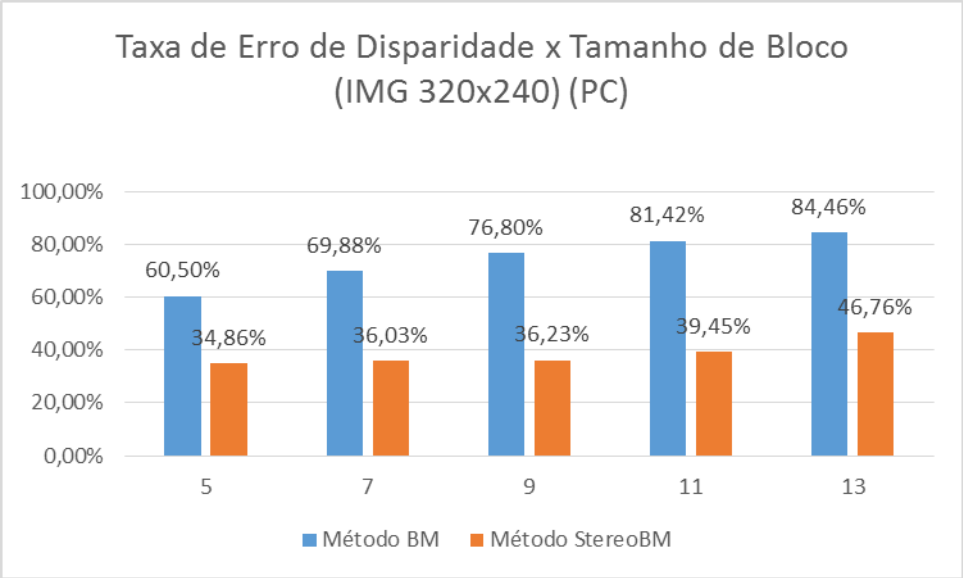


Figura 4.8 – Gráfico de comparação entre o Tamanho de Bloco e a Taxa de Erro de Disparidade sobre os métodos BM e SGBM realizadas no PC

Já o método SGBM possui um processamento um pouco mais lento, mas a eficiência desse método pode ser visualizada nos valores encontrados, tendo uma taxa de erro média na casa dos 35%. O mesmo teste foi realizado na *Raspberry PI2* e os resultados seguem nos próximos gráficos apresentados nas figuras 4.10 e 4.11.

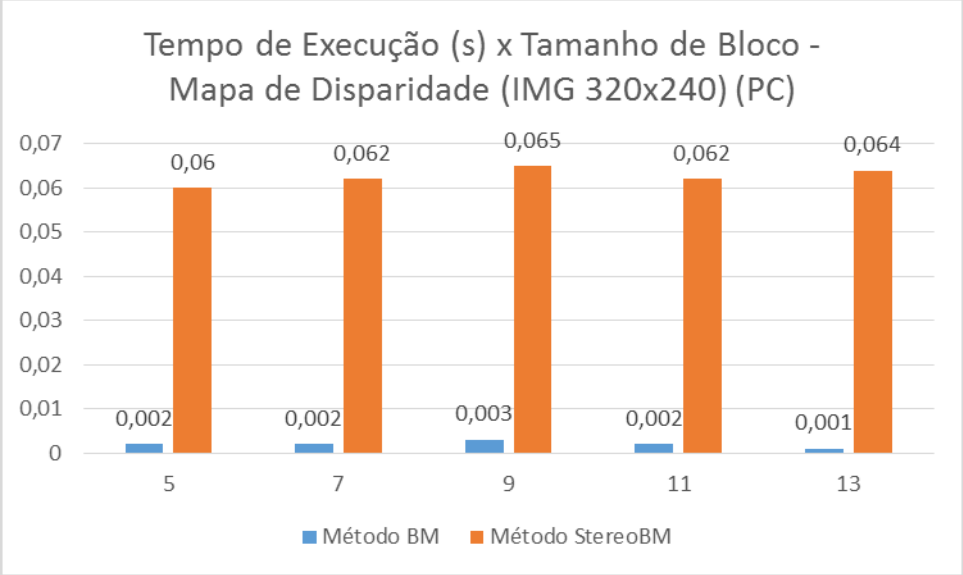


Figura 4.9 – Gráfico de comparação entre o Tempo de Execução e o Tamanho de Bloco sobre os métodos BM e SGBM realizadas no PC

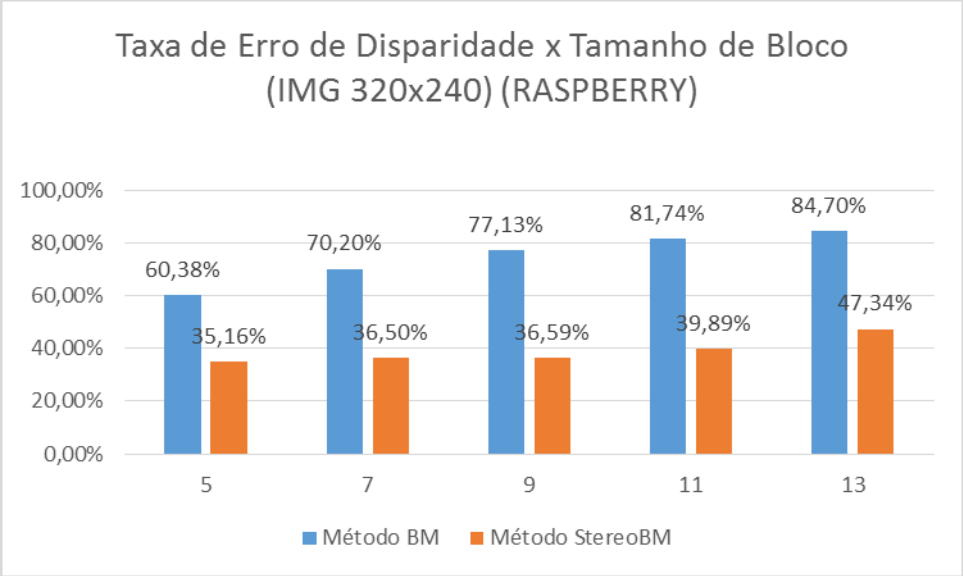


Figura 4.10 – Gráfico de comparação entre o Tamanho de Bloco e a Taxa de Erro de Disparidade sobre os métodos BM e SGBM realizadas na Raspberry

Pode-se observar pela figura 4.10 a comparação entre o tamanho de bloco e a taxa de erros da *Raspberry*, cujo os resultados são bem semelhantes aos obtidos no PC. Mas, no gráfico da figura 4.11, é nítida a diferença de tempo de processamento.

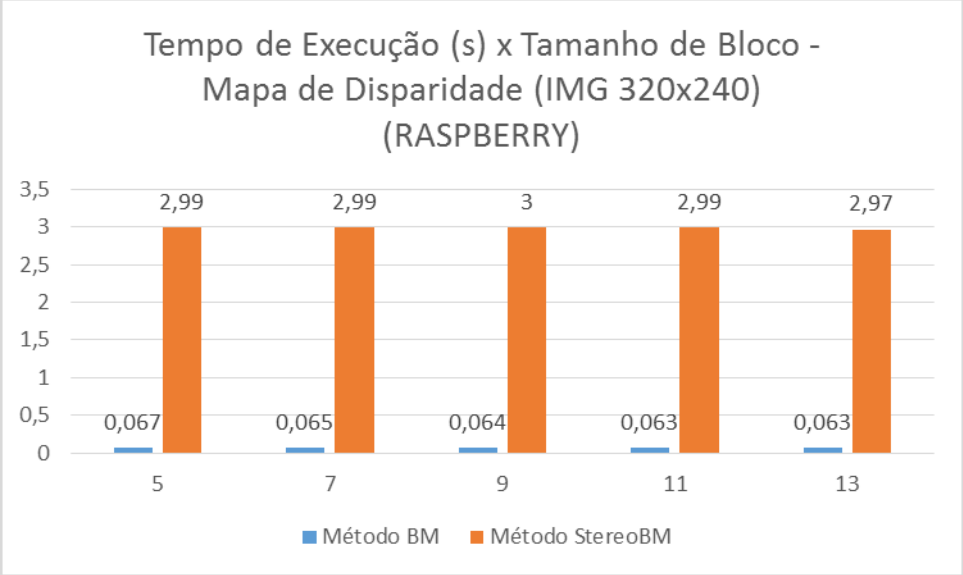


Figura 4.11 – Gráfico de comparação entre o Tempo de Execução e o Tamanho de Bloco sobre os métodos BM e SGBM realizadas na Raspberry

A seguir estão apresentados os resultados com a inferência das imagens convertidas para o padrão HSV. Este teste foi baseado na modificação do tipo de imagens para a correção de possíveis ruídos causados pela iluminação do ambiente. Os testes foram aplicados no PC e *Raspberry*, utilizando o mesmo conjunto de imagens agora convertidos para HSV e os resultados podem ser visualizados no gráfico da figura 4.12.

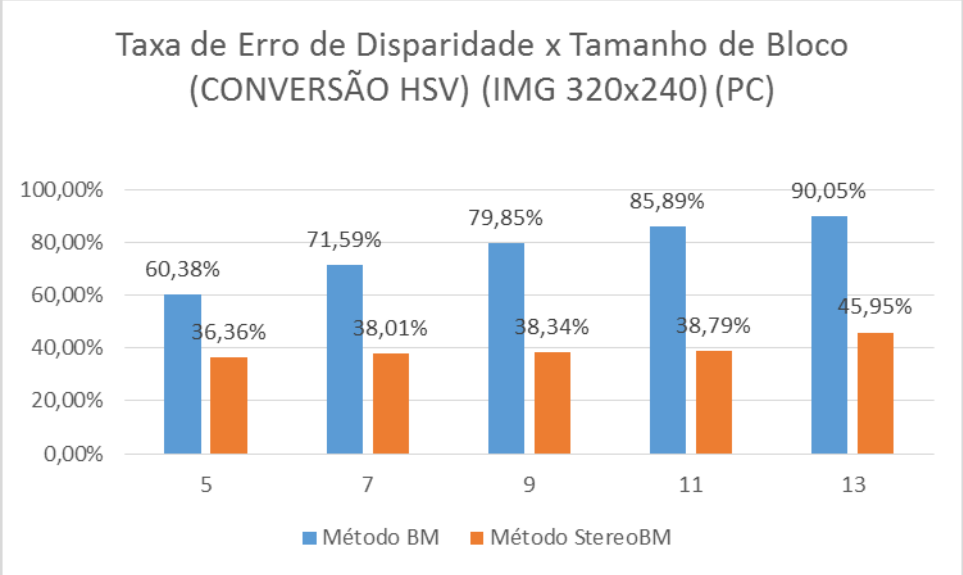


Figura 4.12 – Gráfico de comparação entre o Tamanho de Bloco e a Taxa de Erro de Disparidade sobre os métodos BM e SGBM realizadas no PC com imagens em HSV

Pode-se observar pela figura 4.12 que as taxas de erro da disparidade foram maiores que os resultados dos testes com as imagens RGB.

No gráfico da figura 4.13 é possível observar que o tempo de execução se mantém estável bem próximo dos valores encontrados no teste das imagens em RGB.

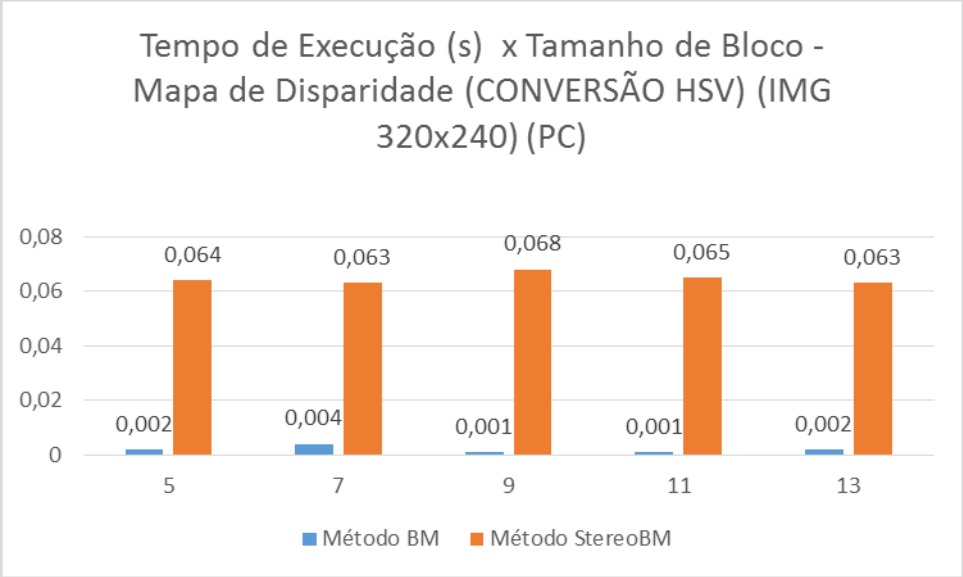


Figura 4.13 – Gráfico de comparação entre o Tempo de Execução e o Tamanho de Bloco sobre os métodos BM e SGBM realizadas no PC com imagens HSV

Para efeito de comparação entre as duas arquiteturas foram testados os mesmos métodos com imagens HSV na *Raspberry*. A seguir, na figura 4.14, são exibidos os resultados obtidos relacionando a taxa de erro de disparidade com o tamanho de bloco. É possível observar no gráfico a elevada taxa de erros com valores acima dos testes realizados com imagens RGB. No resultado apresentado na figura 4.15, é possível verificar que o tempo de processamento é muito próximo dos resultados com as imagens RGB.

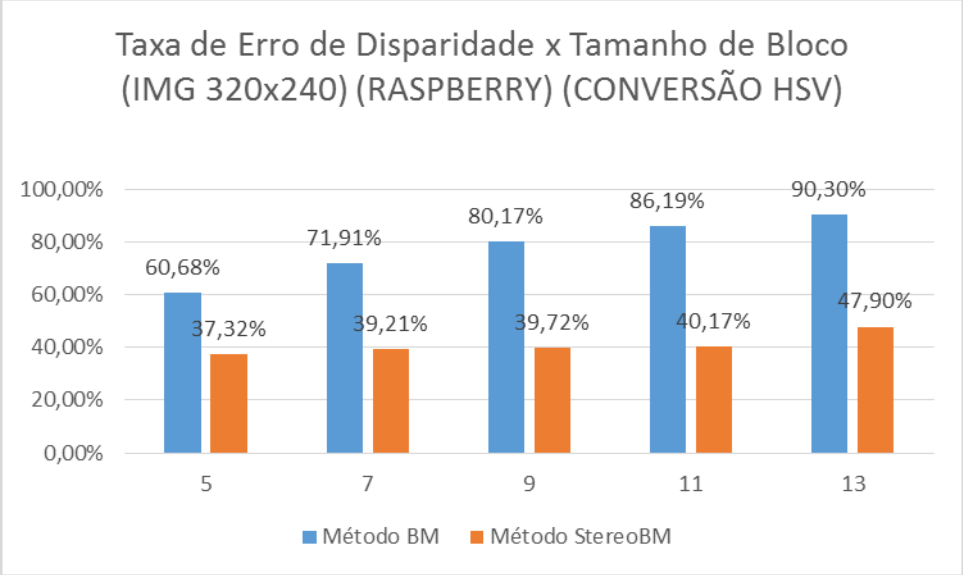


Figura 4.14 – Gráfico de comparação entre o Tamanho de Bloco e a Taxa de Erro de Disparidade sobre os métodos BM e SGBM realizadas na Raspberry com imagens HSV

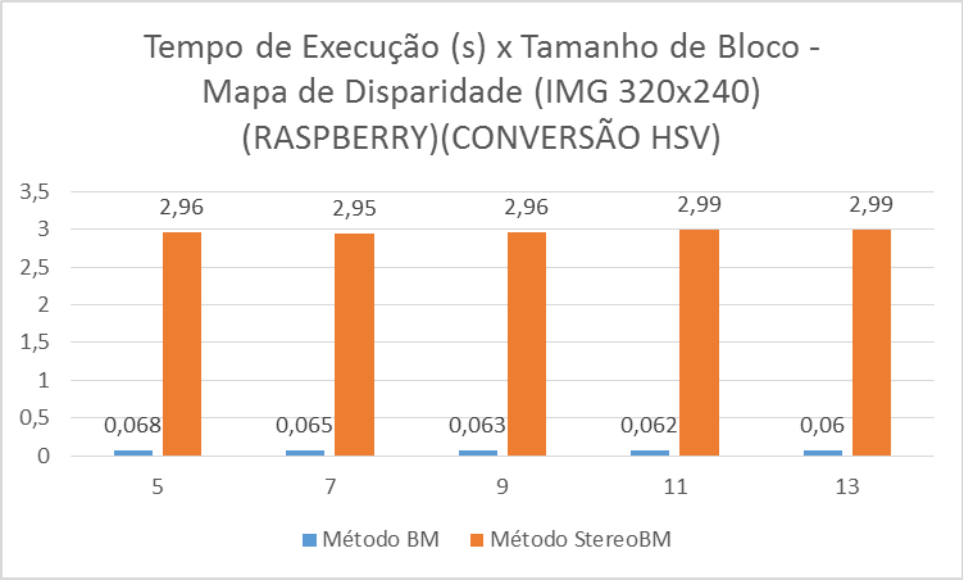


Figura 4.15 – Gráfico de comparação entre o Tempo de Execução e o Tamanho de Bloco sobre os métodos BM e SGBM realizadas na Raspberry com imagens HSV

Com os resultados apresentados é possível determinar que, para o sistema desenvolvido, ao se utilizar imagens RGB, elimina-se a necessidade de conversão para HSV.

Ainda foram efetuados testes com variação de resolução nas duas arquiteturas (*PC e Raspberry*). A seguir seguem os resultados obtidos com a utilização das mesmas imagens em uma resolução reduzida para 160 x 120 pixels.

Uma vez que a câmera utilizada é uma câmera de baixo custo e de uma captação de imagens de baixa qualidade, foram feitos testes específicos para determinar a resolução ideal a ser utilizada pelo sistema. Analisando os resultados das comparações entre os tamanhos das imagens e as taxas de erros, as imagens de resolução 160x120 (figura 4.16) obtiveram maior taxa de erro comparadas com as imagens de resolução 320x240 (figura 4.8). Por essa razão, foi definido o uso das imagens de resolução 320x240 para o sistema.

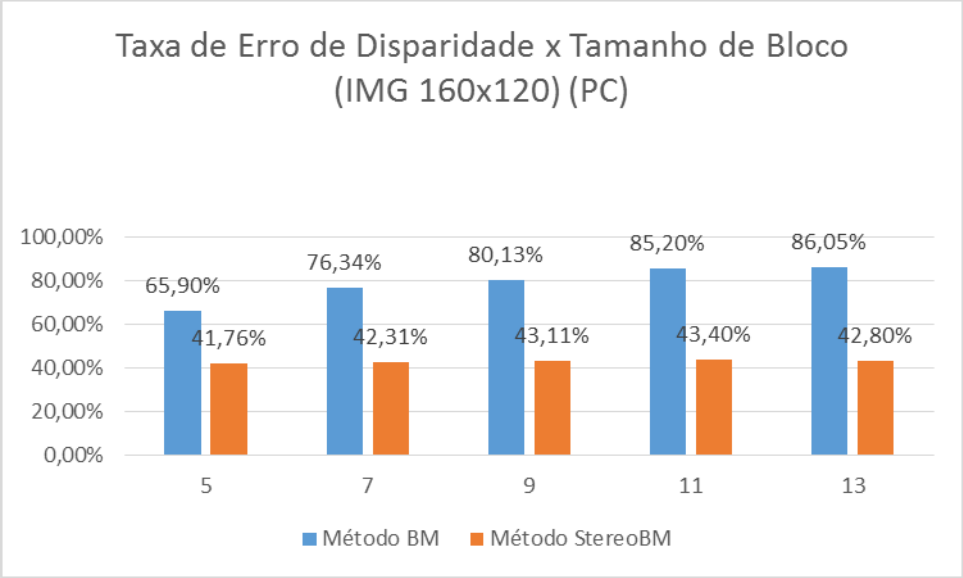


Figura 4.16 – Gráfico de comparação entre o Tamanho de Bloco e a Taxa de Erro de Disparidade sobre os métodos BM e SGBM realizadas no PC resolução de 160x120

Na próxima etapa é gerado o mapa de disparidade pelo método *semiglobal* com base nas imagens obtidas em tempo real já corrigidas pelas distorções de cada lado. A correção é gerada com base na calibração realizada com as câmeras.

A figura 4.17 mostra um mapa de disparidade gerado no PC utilizando imagens RGB, resolução de 320 x 240 e métodos SGBM com janela de bloco com valor 5.

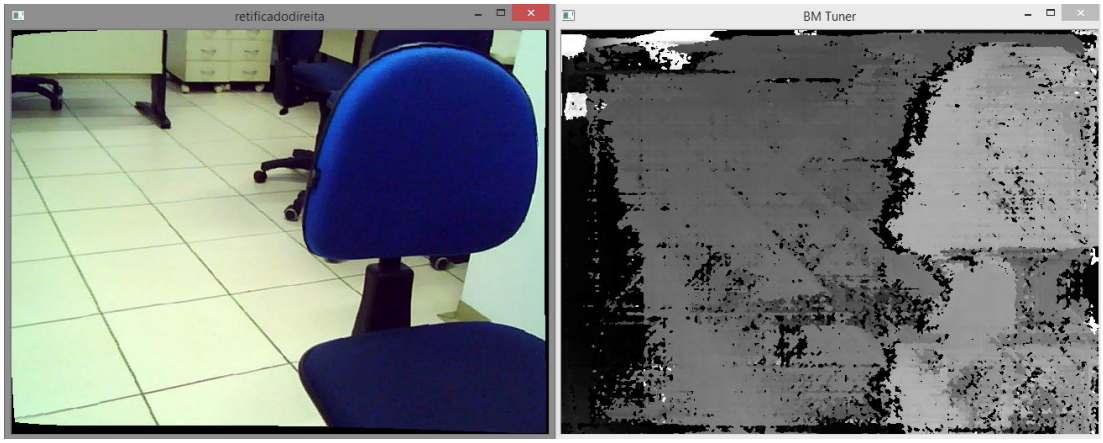


Figura 4.17 – Mapa de Disparidade

4.6 Sistema para Detecção de Obstáculos

Com o mapa de disparidade gerado são aplicados filtros e métodos para a detecção do caminho livre de obstáculos. Para o sistema criado aplicaram-se duas diferentes técnicas: uma baseada nas bordas extraídas do mapa de disparidade e outra na aplicação de um limiar binário. As duas técnicas apresentaram resultados diferentes, sendo que uma delas apresentou uma saída satisfatória para os objetivos do trabalho.

A proposta para a extração do caminho livre pelo método de utilização de bordas segue o processo de navegação de robôs do *RobotRealm* (“Roborealm.com”, 2016). O processo se inicia com a aquisição de imagens e tem como passo principal a aplicação de detecção de bordas. No passo seguinte aplica-se um algoritmo para criar linhas verticais até a borda encontrada e em seguida é realizada uma erosão seguida de uma dilatação da imagem obtida. A seguir os dados obtidos são interpretados e apresentam como saída o caminho livre. Com base nesse método, segundo o “Roborealm.com" (2016), é possível detectar com base no pixel mais elevado da imagem o caminho livre para locomoção.

Seguindo o mesmo processo, será proposta a substituição da utilização de detecção de bordas pela técnica de *thresholding*.

4.6.1 Aquisição de Imagem

O processo se inicia com a aquisição das imagens da direita e esquerda. As imagens são captadas pela câmera *Minoru3D* e enviadas para o sistema. As matrizes de calibração são aplicadas a cada par de imagens por frame, tornando possível a retificação das imagens e extração do mapa de disparidade. O método escolhido foi o SGBM (*semiglobal*) com tamanho de bloco 5. Na figura 4.18 é possível observar na primeira imagem a aquisição da câmera direita e no cenário um obstáculo que é uma cadeira com possível caminho livre a esquerda. Na outra imagem segue a disparidade gerada mostrando em tons mais claros o objeto mais próximo da câmera.

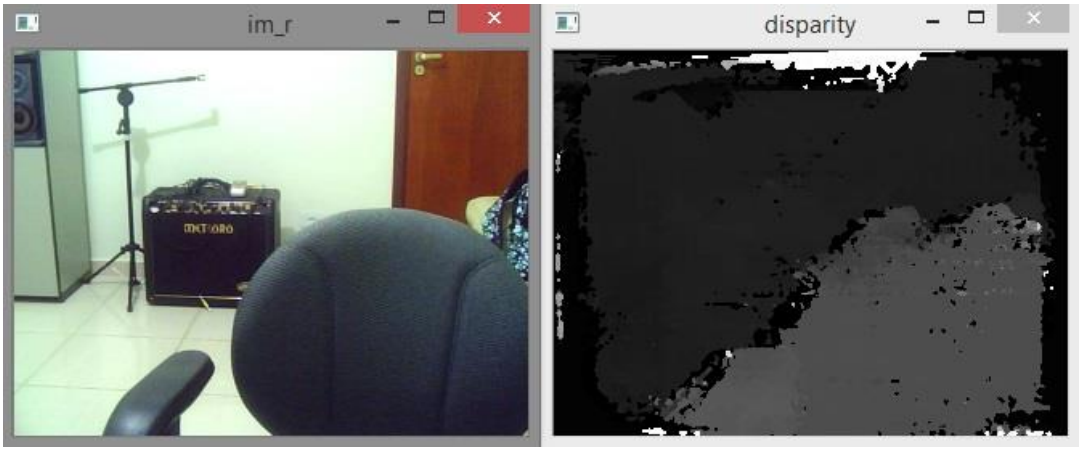


Figura 4.18 – Aquisição de Imagem e Mapa de Disparidade

4.6.2 Preenchimento Vertical

A próxima etapa consiste no tratamento da disparidade gerada para a aplicação de linhas verticais indicando um possível caminho livre de obstáculos. Após o tratamento será aplicado um algoritmo para a criação de linhas verticais brancas, seguindo a premissa de que para cada pixel branco encontrado é cancelada a aplicação da linha, seguindo para a próxima coluna, partindo-se da base da imagem.

Neste tópico serão descritas duas técnicas utilizadas: uma por utilização de detecção de bordas e outra por aplicação de um *thresholding*. Processos estes que antecedem a aplicação do preenchimento vertical.

4.6.2.1 Detecção de Bordas

Para a detecção de bordas o método escolhido foi o de *Canny* (CANNY, 1986). O processo consiste em aplicar sobre o mapa de disparidade a detecção de bordas.

A aplicação da técnica de detecção de bordas neste sistema trouxe algumas falhas conforme será apresentado a seguir.

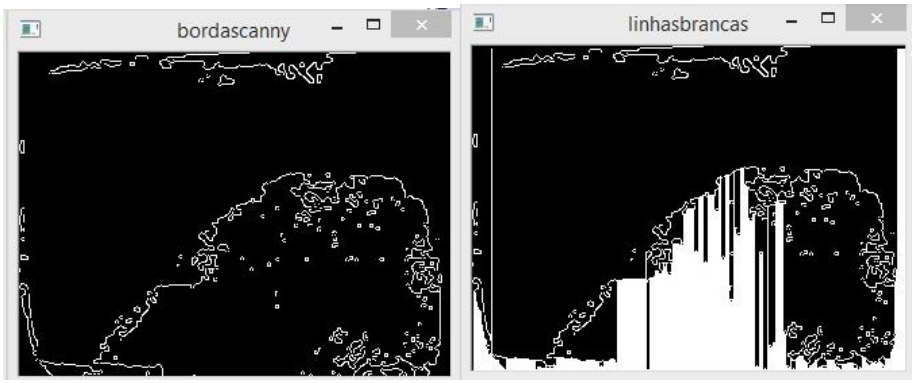


Figura 4.19 – Aplicação de Detecção de Bordas no Mapa de Disparidade

Pode-se observar na figura 4.19, que o algoritmo de criação de linhas verticais passou sobre o objeto da cena, obtendo assim um dado inconsistente sobre o possível caminho livre de obstáculos. Para o propósito deste projeto de extrair o caminho livre pelo mapa de disparidade, o método de detecção de bordas foi descartado e para substituir essa técnica foi utilizado um *thresholding,* conforme será descrito no próximo tópico.

4.6.2.2 Limiarização do Mapa de Profundidade

O mapa de disparidade como já descrito anteriormente é um mapa em níveis de cinza que possuem nos tons mais claros os objetos em cena mais próximos da câmera estéreo. Partindo-se desse princípio foi utilizado no sistema, como substituição da detecção de bordas, a limiarização da imagem de disparidade. A partir de um valor de nível de cinza executa-se o *threshold* binário, em que os pixels recebem o valor máximo. Os pixels que não possuírem valor igual ou superior são zerados, obtendo-se uma imagem binária, em que os objetos são evidenciados pela aplicação sobre o mapa de disparidade. A figura 4.20 mostra a substituição do processo de detecção de bordas pela aplicação do *thresholding*.

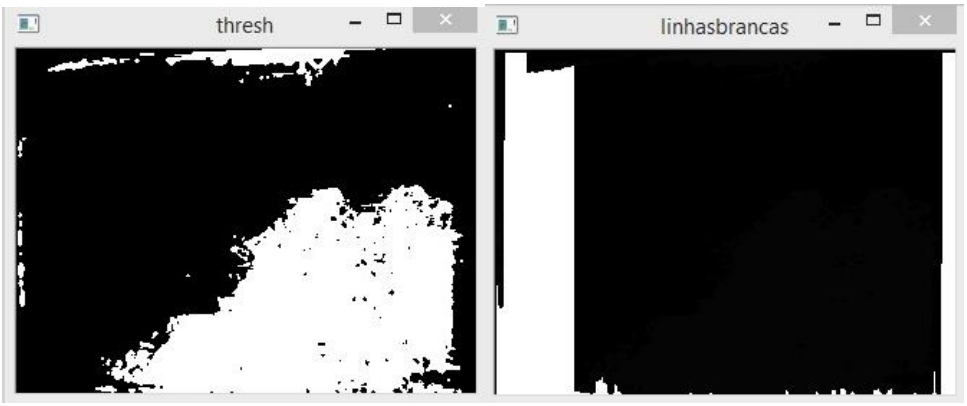


Figura 4.20 – Aplicação do *Thresholding* Binário sobre Mapa de Disparidade

4.6.3 Erosão

Com o preenchimento vertical aplicado na imagem tratada, é necessário aplicar a erosão para retirar de cena dados inconsistentes, bordas ou linhas atravessando objetos que possam ser considerados ruídos. Essa técnica tende a diminuir ou remover linhas que possuam uma área pequena. Na figura 4.21 pode-se observar que as linhas menores foram diminuídas ou totalmente removidas neste processo de erosão, deixando mais evidente o possível caminho livre de obstáculos.

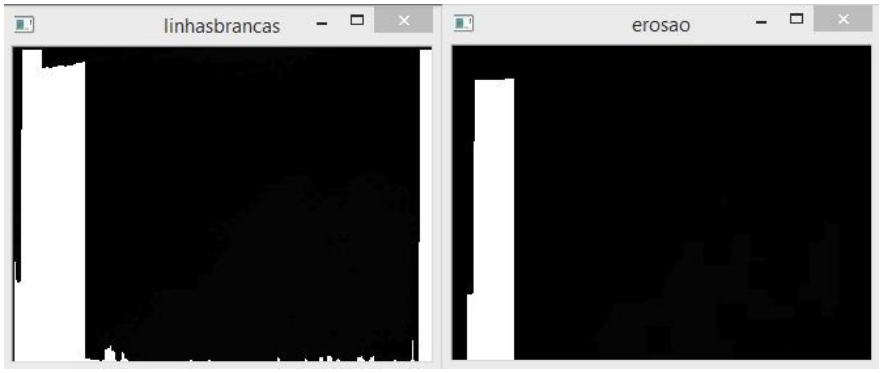


Figura 4.21 – Aplicação da Erosão sobre as Linhas Verticais

4.6.4 Dilatação

A partir do processo de erosão para compensar a diminuição de alguns componentes da imagem, é realizada a dilatação, processo pelo qual é realizado o aumento dos objetos em cena, evidenciando mais ainda o possível caminho livre de obstáculos.

A figura 4.22 mostra o resultado da aplicação da dilatação na imagem erodida.

Figura 4.22 – Aplicação da Dilatação sobre imagem erodida

4.6.5 Interpretação e Saída

A ideia principal é encontrar a parte da imagem que possua maior área livre para locomoção. Partindo desse princípio a proposta para encontrar o caminho livre é dividir a imagem final dilatada em três imagens: esquerda, centro e direita.

4.6.5.1 Análise da Imagem Particionada

Em cada parte de imagem particionada será aplicado um algoritmo para contabilizar quantos pixels brancos, que seria o caminho livre, cada parcela possui. A parcela que contiver o maior número de pixels brancos será identificada como possível caminho livre da imagem. Outra condição implementada é a checagem dos últimos três *frames* com a mesma saída antes de informar ao usuário o caminho

livre. Isso foi necessário devido a possibilidade de alguns ruídos aparecerem na imagem devido a movimentação da câmera.

Outra condição para a saída do sistema é o levantamento do pixel branco mais alto na imagem, sendo ele identificado do lado esquerdo, centro ou direito, tornando-se parte da verificação antes da efetiva resposta para o usuário.

A seguir serão descritos os procedimentos utilizados no algoritmo implementado:

- A imagem dilatada é dividida em três partes iguais, esquerda, centro e direita;
- Cada parcela de imagem passa por uma contagem de pixels brancos;
- A parcela com o maior número de pixels é marcada;
- Uma verificação auxiliar armazena a partição que contenha o pixel branco mais alto;
- Para a saída com o lado que contenha o caminho livre de obstáculo são analisadas as seguintes condições:
    - Partição com as últimas três marcações de maior número de pixels brancos, aumentando a confiança na resposta do sistema;
    - A mesma partição deverá conter o pixel mais alto na imagem;
    - A parcela deverá conter pelo menos 30% de pixels brancos;
    - Caso alguma dessas condições não forem satisfeitas o sistema o sistema informará um possível obstáculo em cena;
- O sistema terá como saída um áudio indicando os lados com caminho livre de obstáculo: esquerda, centro ou direita. Caso nenhum caminho seja identificado será informado: obstáculo.

4.7 Resultados Obtidos

Para validação do sistema implementado neste trabalho foram realizados alguns testes exaustivos de inferência de imagens captadas através da câmera Minoru 3D. Os testes consistiram em se obter a maior eficiência de resposta sobre as imagens testadas. Foram captadas através do par de câmeras pelo menos 90 pares de imagens de um ambiente determinando para a navegação proposta. Em

seguida foram divididas em imagens com caminho livre à esquerda, centro e à direita. Cada conjunto de imagens foi inserido no sistema para se auferir a capacidade de resposta em porcentagem de acerto e tempo de execução médio. Os mesmos testes também foram aplicados em dois equipamentos diferentes, o PC e a *Rapsberry* PI, com as configurações já detalhadas no início deste capítulo.

4.7.1 Resultados obtidos pela inferência no PC

A seguir seguem os resultados dos testes realizados no PC. Nos gráficos a seguir é possível observar a taxa de acerto de resposta do sistema e tempo de execução médio de cada par de imagens. Para os testes foram variados alguns parâmetros fundamentais, como o valor do limiar para a aplicação do *thresholding* binário e número de interações de erosão e dilatação

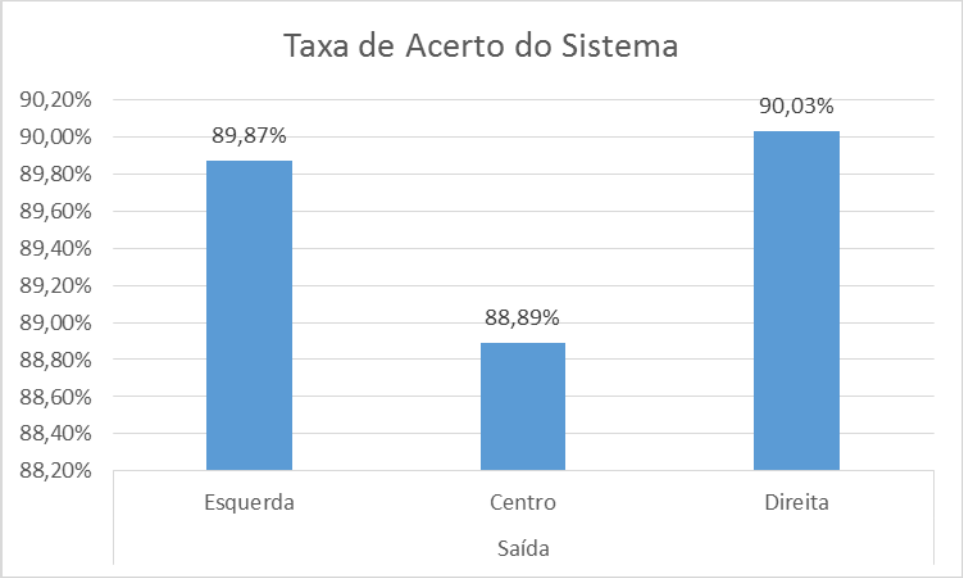


Figura 4.23 – Gráfico Taxa de Acerto do Sistema para cada par de imagens

Na figura 4.23 é possível observar os melhores resultados adquiridos pelo sistema, com valor definido de 2 interações de erosão e 4 interações de dilatação. Para o conjunto de imagens com caminho livre à esquerda a taxa de acerto foi de 89,87%; para o conjunto de imagens do centro, o valor foi de 88,89% de acerto; e para imagens da direita, o sistema respondeu com 90,03% de precisão.

No gráfico da figura 4.24, pode-se observar que o melhor valor de *thresholding* binário foi de 23% e para essa taxa de acerto o tempo médio de execução foi de 0,4682s como mostra a figura 4.25.

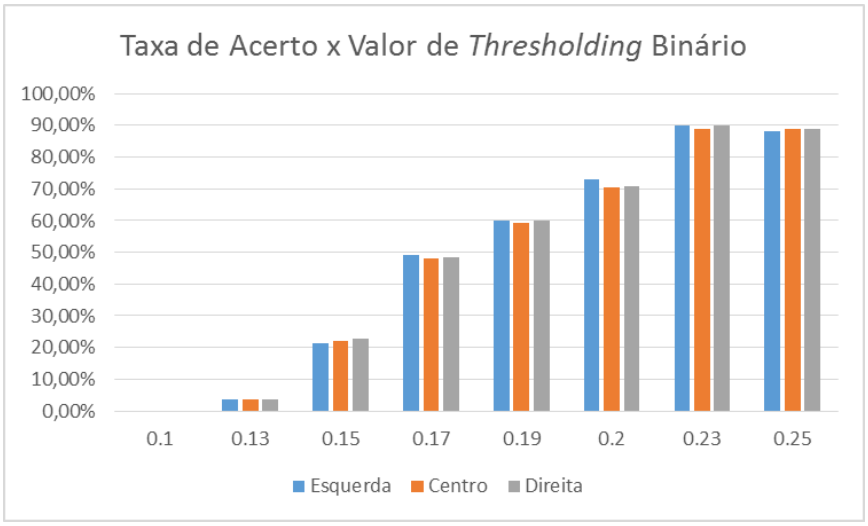


Figura 4.24 – Gráfico Taxa de Acerto x Valor de *Thresholding* Binário

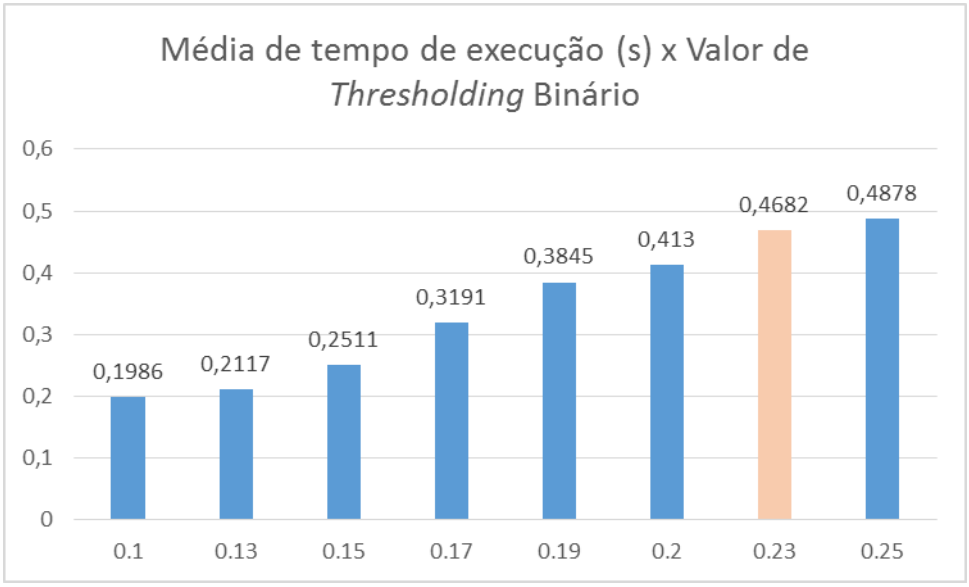


Figura 4.25 - Gráfico Tempo Médio de Execução x Valor de *Thresholding* Binário

4.7.2 Resultados obtidos pela inferência na Raspberry PI

Uma das propostas deste projeto foi testar o método implementado em uma arquitetura menor, possibilitando uma maior mobilidade para o sistema. A seguir seguem os resultados dos testes realizados com os mesmos conjuntos de imagens.

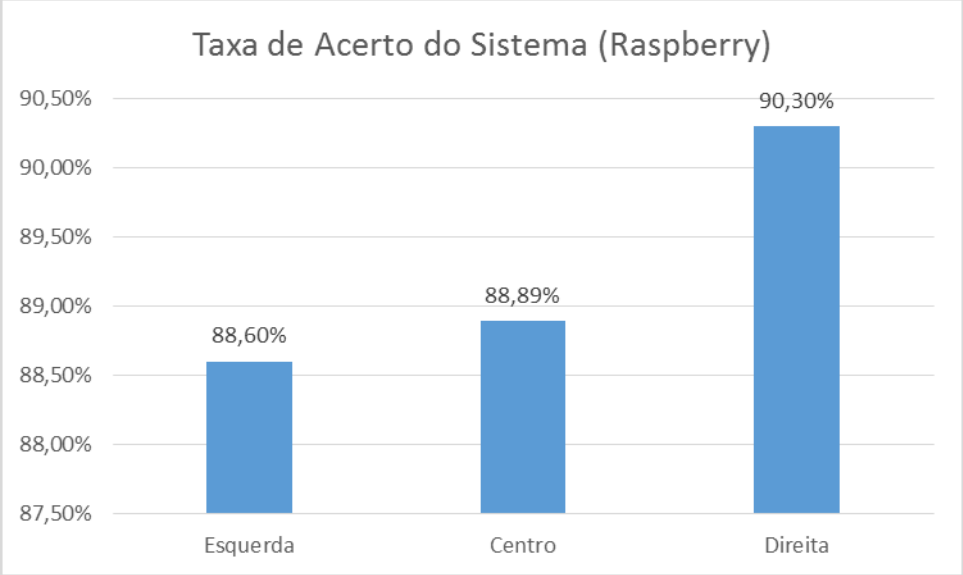


Figura 4.26 – Gráfico Taxa de Acerto do Sistema para cada par de imagens (Raspberry)

É possível observar na figura 4.26 que a taxa de acerto da *Raspberry* é bem semelhante à resposta dos testes realizados no PC. No gráfico 4.27, é mostrado o tempo de execução médio do processamento na *Raspberry* variando-se o *thresholding* binário.

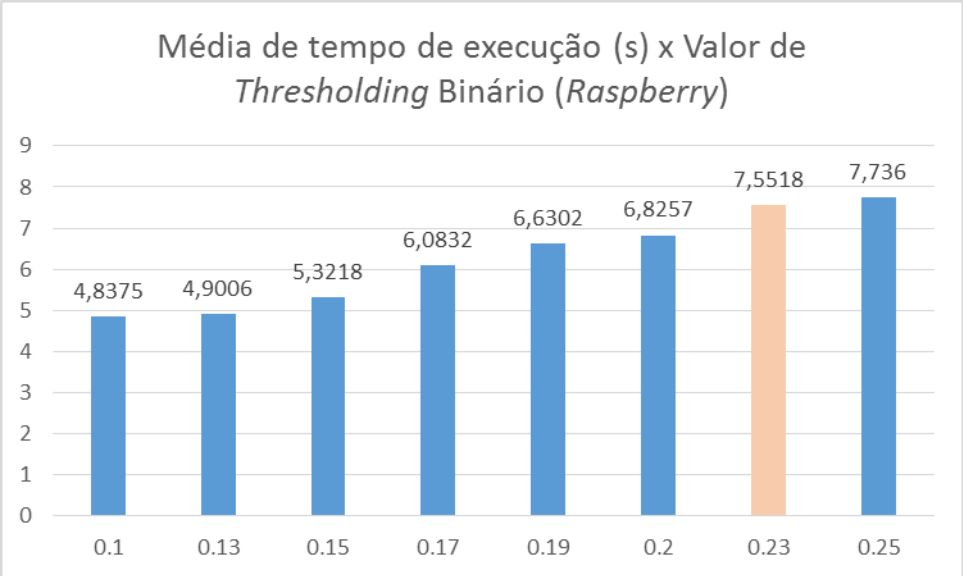


Figura 4.27 – Gráfico Tempo Médio de Execução x Valor de *Thresholding* Binário Raspberry

Observando o gráfico na figura 4.25 é possível verificar como o tempo de execução na *Raspberry* é alto. Para cada par de imagens o tempo é de 7,5518 segundos. De acordo com testes já apresentados no início do capítulo, uma das possíveis causas é a resolução das imagens, por isso um novo teste foi realizado

alterando a resolução das imagens para 160x120, mas a taxa de acerto foi de 0% em todas as variações realizadas neste trabalho.

4.8 Considerações Finais

Neste capítulo foram apresentados a metodologia e os testes realizados com os procedimentos implementados para este projeto. No próximo capítulo, serão apresentadas as conclusões obtidas com os processos empregados nesse trabalho.

Capítulo 5

CONCLUSÕES

Apresentação das conclusões tendo por base os métodos e testes desenvolvidos nesse trabalho

5.1 Considerações Iniciais

No capítulo anterior foi possível descrever a metodologia utilizada para a criação do sistema de auxílio à navegação utilizando uma câmera de baixo custo. A seguir estão descritas as conclusões tendo por base os métodos e testes realizados.

5.2 Conclusões Obtidas

O sistema proposto nesse trabalho de mestrado tem por objetivo identificar o caminho livre para a navegação de um usuário portador de alguma deficiência visual por meio de um único sensor, através de uma câmera estereoscópica e em um ambiente interno comum, auxiliando-o na locomoção segura por este cenário. Outro ponto relevante para o projeto foi o baixo custo do sistema, utilizando uma câmera estéreo de baixo valor e uma solução criada em uma linguagem de programação aberta.

O sistema implementado segue referências de navegação em ambientes internos, e, para os objetivos desse projeto, foi proposta uma modificação no

processo de tratamento do mapa de disparidade para a extração do possível caminho livre de obstáculos.

O projeto foi testado em duas arquiteturas diferentes, com o objetivo de comparar os resultados, bem como implementar a solução a partir de um equipamento menor possibilitando, portanto, maior mobilidade para o usuário.

Pôde-se observar que nas duas arquiteturas testadas (PC e *Raspberry* PI) o índice de eficiência do método em média é de quase 90% de acerto para cada par de imagens. Em contrapartida, o tempo de execução na *Raspberry PI* é 16 vezes maior comparado ao tempo médio de processamento no PC.

Muitos fatores podem influenciar os resultados, como iluminação e texturas diversas encontradas nos ambientes testados, resultantes da aplicação de um único sensor de entrada para o sistema. A utilização de sensores de proximidade pode ser estudada futuramente para aprimoramento dos resultados. O teste realizado com a arquitetura de menor porte mostrou-se ineficaz pela baixa capacidade de processamento, podendo ser alvo de trabalhos futuros com arquiteturas menores, mas com maior capacidade de processamento.

O projeto criado pode ser considerado aceitável quando observado o alto índice de eficiência das imagens testadas, resultado da modificação do processo de análise do mapa de disparidade analisado.

Para trabalhos futuros, algumas melhorias podem ser realizadas e testadas para uma maior eficiência do processo:

- Embarque em arquiteturas menores com maior capacidade de processamento que a *Raspberry PI*;
- Melhorias nos processos e algoritmos utilizados neste projeto;
- Testes com outras câmeras e sensores para captação de dados e inferência no sistema;
- Testes e validação em pessoas;

5.3 Considerações Finais

Neste capítulo foram apresentadas as conclusões e resultados positivos do trabalho e possíveis sugestões de melhorias para projetos futuros.

REFERÊNCIAS

AIRES, G. DE S. **ESTIMACÃO DE MOVIMENTO DE UMA CÂMERA VIA GEOMETRIA EPIPOLAR**. [s.l.] UFRJ, 2010.

ALBERTO BORGHESE, N.; CERVERI, P. Calibrating a video camera pair with a rigid bar. **Pattern Recognition**, v. 33, n. 1, p. 81–95, jan. 2000.

ALBUQUERQUE, M. P. DE. Processamento de Imagens: Métodos e Análises. **Revista Científica**, p. 12, 2000.

ANDERSON, E. W.; PRESTON, G. A.; SILVA, C. T. Using Python for Signal Processing and Visualization. **Computing in Science & Engineering**, v. 12, n. 4, p. 90–95, jul. 2010.

BOURBAKIS, N.; MAKROGIANNIS, S. K.; DAKOPOULOS, D. A System-Prototype Representing 3D Space via Alternative-Sensing for Visually Impaired Navigation. **IEEE Sensors Journal**, v. 13, n. 7, p. 2535–2547, jul. 2013.

CANNY, J. A Computational Approach to Edge Detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. PAMI-8, n. 6, p. 679–698, 1986.

COSTA, P. et al. Fiducials Marks Detection to Assist Visually Impaired People Navigation. **JDCTA: International Journal of Digital Content Technology and its Applications**, v. 5, n. 5, p. 342–350, 2011.

FERNANDEZ, M. E. L. **Calibração de Múltiplas Câmeras baseado em um Padrão Invariante**. [s.l.] Pontifícia Universidade Católica do Rio de Janeiro, 2009.

FILHO, O. M.; VIEIRA NETO, H. **Processamento Digital de Imagens**. Processamento Digital de Imagens: Brasport, 1999.

FUSIELLO, A.; TRUCCO, E.; VERRI, A. A compact algorithm for rectification of stereo pairs. **Machine Vision and Applications**, v. 12, n. 1, p. 16–22, 1 jul. 2000.

GUERRA, R. DA S. **Calibração automática de sistemas de visão estéreo a partir de movimentos desconhecidos**. [s.l.] Universidade Federal do Rio Grande do Su, Porto Alegre, RS, 2004.

HARTLEY, R.; ZISSERMAN, A. **Multiple View Geometry in Computer Vision**. second ed ed. New York - USA: Cambridge University Press, 2004.

IBGE - Instituto Brasileiro de Geografia e Estatística.

LEUNG, T.-S.; MEDIONI, G. Visual Navigation Aid for the Blind in Dynamic Environments. **2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops**, p. 579–586, jun. 2014.

LIN, Q.; HAN, Y.; HAHN, H. Vision-Based Navigation Using Top-view Transform and Beam-ray Model. p. 978–979, 2011.

MADALENA, I. D. M.; GOMES, D. Auto-Calibração de Câmeras em Visão Estéreo. **decom.ufop.br**, 2012.

MATTOCCIA, S. **Stereo Vision : Algorithms and Applications**.

MENDES, C. C. T.; WOLF, D. F. Desvio de Obstáculos Utilizando um Método Estéreo. **Congresso da Sociedade Brasileira de Computação (CSBC) - Encontro Nacional de Inteligência Artificial**, p. 788–799, 2011.

Minoru3D.com. Disponível em: <http://www.minoru3d.com/>. Acesso em: 10 fev. 2015.

NALPANTIDIS, L.; KOSTAVELIS, I.; GASTERATOS, A. Stereovision-based algorithm for obstacle avoidance. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 5928 LNAI, p. 195–204, 2009.

NOGUEIRA, F. M. D. A.; TOZZI, C. L. Geração Automática de Mapas de Disparidade em Visão Estéreo. **Anais do XI SIBGRAPI**, p. 1–8, 1998.

OpenCV.org. Disponível em: <http://opencv.org/>. Acesso em: 1 dez. 2014.

PEDRINO, E. C. **Apostila de Tratamento Digital de Imagens**São CarlosUfscar, , 2003.

PRADEEP, V.; MEDIONI, G.; WEILAND, J. Robot vision for the visually impaired. **2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops**, p. 15–22, jun. 2010.

Raspberrypi.org. Disponível em: <https://www.raspberrypi.org>. Acesso em: 10 abr. 2015.

Roborealm.com. Disponível em: <http://www.roborealm.com/tutorial/Obstacle_Avoidance/slide010.php>. Acesso em: 25 jun. 2016.

SANTOS, M. C. DOS. **Câmera , Geometria Epipolar , Mapas de Profundidade e Varredura de Planos**Campinas - SP, 2012.

SEUBA, P. J. M. **Estimació trinocular de mapes de profunditat**. [s.l.] Universitat Politècnica de Catalunya, 2009.

SOUZA, A. A. DE S. **Mapeamento Robótico 2,5-D com Representação em Grade de Ocupação-Elevação**. [s.l.] UFRN, 2012.

STIVANELLO, M. Desenvolvimento de uma biblioteca para sistemas de visão estereoscópia para robótica móvel. 2008.

SVOBODA, T.; MARTINEC, D.; PAJDLA, T. A Convenient Multicamera Self-Calibration for Virtual Environments. **Presence: Teleoperators and Virtual Environments**, v. 14, n. 4, p. 407–422, ago. 2005.

TOMMASELLI, A. M. G. **Fotogrametria Básica – Estereoscopia e Paralaxe**Curso Fotogrametria I da Unesp – Presidente Prudente, , 2007.

WANG, Y. M.; LI, Y.; ZHENG, J. B. A camera calibration technique based on OpenCV. **The 3rd International Conference on Information Sciences and Interaction Sciences**, p. 403–406, jun. 2010.

Apêndice A

CÓDIGOS EM PYTHON

A seguir seguem os códigos já implementados para a aquisição de imagens, calibração das câmeras e geração do mapa de disparidade. O pré-requisito para a execução dos códigos é a instalação das bibliotecas OpenCV, NumPy e ProgressBar.

1 – Classe para aquisição das imagens:

```
from argparse import ArgumentParser
import os

import cv2
from progressbar import ProgressBar, Bar, Percentage
from stereo_cameras import ChessboardFinder
from ui_utils import calibrate_folder, CHESSBOARD_ARGUMENTS
from ui_utils import find_files
import time

def main():
    left=1
    right=2
    num_pictures=60
    output_folder="c:\imagensppm3"

    progress = ProgressBar(maxval=num_pictures,widgets=[Bar("=", "[", "]")," ",
Percentage()])
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    with ChessboardFinder((left, right)) as pair:
        for i in range(num_pictures):
            frames = pair.get_chessboard(7, 4, True)

            for side, frame in zip(("left", "right"), frames):
                number_string = str(i + 1).zfill(len(str(num_pictures)))
                filename = "{}_{}.ppm".format(side, number_string)
                output_path = os.path.join(output_folder, filename)
                cv2.imwrite(output_path, frame)
            progress.update(progress.maxval - (num_pictures - i))
            for i in range(10):
                pair.show_frames(1)
```

```
            time.sleep(3)
        progress.finish()


if __name__ == "__main__":
    main()
```

2 – Classe para calibração das câmeras:

```
import os

import cv2

import numpy as np
from exceptionss import ChessboardNotFoundError


class StereoCalibration(object):

    """
    A stereo camera calibration.

    The ``StereoCalibration`` stores the calibration for a stereo pair. It can
    also rectify pictures taken from its stereo pair.
    """

    def __str__(self):
        output = ""
        for key, item in self.__dict__.items():
            output += key + ":\n"
            output += str(item) + "\n"
        return output

    def _copy_calibration(self, calibration):
        """Copy another ``StereoCalibration`` object's values."""
        for key, item in calibration.__dict__.items():
            self.__dict__[key] = item

    def _interact_with_folder(self, output_folder, action):
        """
        Export/import matrices as *.npy files to/from an output folder.

        ``action`` is a string. It determines whether the method reads or writes
        to disk. It must have one of the following values: ('r', 'w').
        """
        if not action in ('r', 'w'):
            raise ValueError("action must be either 'r' or 'w'.")
        for key, item in self.__dict__.items():
            if isinstance(item, dict):
                for side in ("left", "right"):
                    filename = os.path.join(output_folder,
                                            "{}_{}.npy".format(key, side))
                    if action == 'w':
                        np.save(filename, self.__dict__[key][side])
                    else:
                        self.__dict__[key][side] = np.load(filename)
            else:
                filename = os.path.join(output_folder, "{}.npy".format(key))
                if action == 'w':
                    np.save(filename, self.__dict__[key])
```

```
            else:
                self.__dict__[key] = np.load(filename)

    def __init__(self, calibration=None, input_folder=None):
        """
        Initialize camera calibration.

        If another calibration object is provided, copy its values. If an input
        folder is provided, load ``*.npy`` files from that folder. An input
        folder overwrites a calibration object.
        """
        #: Camera matrices (M)
        self.cam_mats = {"left": None, "right": None}
        #: Distortion coefficients (D)
        self.dist_coefs = {"left": None, "right": None}
        #: Rotation matrix (R)
        self.rot_mat = None
        #: Translation vector (T)
        self.trans_vec = None
        #: Essential matrix (E)
        self.e_mat = None
        #: Fundamental matrix (F)
        self.f_mat = None
        #: Rectification transforms (3x3 rectification matrix R1 / R2)
        self.rect_trans = {"left": None, "right": None}
        #: Projection matrices (3x4 projection matrix P1 / P2)
        self.proj_mats = {"left": None, "right": None}
        #: Disparity to depth mapping matrix (4x4 matrix, Q)
        self.disp_to_depth_mat = None
        #: Bounding boxes of valid pixels
        self.valid_boxes = {"left": None, "right": None}
        #: Undistortion maps for remapping
        self.undistortion_map = {"left": None, "right": None}
        #: Rectification maps for remapping
        self.rectification_map = {"left": None, "right": None}
        if calibration:
            self._copy_calibration(calibration)
        elif input_folder:
            self.load(input_folder)

    def load(self, input_folder):
        """Load values from ``*.npy`` files in ``input_folder``."""
        self._interact_with_folder(input_folder, 'r')

    def export(self, output_folder):
        """Export matrices as ``*.npy`` files to an output folder."""
        if not os.path.exists(output_folder):
            os.makedirs(output_folder)
        self._interact_with_folder(output_folder, 'w')

    def rectify(self, frames):
        """
        Rectify frames passed as (left, right) pair of OpenCV Mats.

        Remapping is done with nearest neighbor for speed.
        """
        new_frames = []
        for i, side in enumerate(("left", "right")):
            new_frames.append(cv2.remap(frames[i],
                                        self.undistortion_map[side],
                                        self.rectification_map[side],
                                        cv2.INTER_NEAREST))
```

```
        return new_frames


class StereoCalibrator(object):

    """A class that calibrates stereo cameras by finding chessboard corners."""

    def _get_corners(self, image):
        """Find subpixel chessboard corners in image."""
        temp = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        ret, corners = cv2.findChessboardCorners(temp,
                                                 (self.rows, self.columns))
        if not ret:
            raise ChessboardNotFoundError("No chessboard could be found.")
        cv2.cornerSubPix(temp, corners, (11, 11), (-1, -1),
                         (cv2.TERM_CRITERIA_MAX_ITER + cv2.TERM_CRITERIA_EPS,
                          30, 0.01))
        return corners

    def _show_corners(self, image, corners):
        """Show chessboard corners found in image."""
        temp = image
        cv2.drawChessboardCorners(temp, (self.rows, self.columns), corners,
                                  True)
        window_name = "Chessboard"
        cv2.imshow(window_name, temp)
        if cv2.waitKey(0):
            cv2.destroyWindow(window_name)

    def __init__(self, rows, columns, square_size, image_size):
        """
        Store variables relevant to the camera calibration.

        ``corner_coordinates`` are generated by creating an array of 3D
        coordinates that correspond to the actual positions of the chessboard
        corners observed on a 2D plane in 3D space.
        """
        #: Number of calibration images
        self.image_count = 0
        #: Number of inside corners in the chessboard's rows
        self.rows = rows
        #: Number of inside corners in the chessboard's columns
        self.columns = columns
        #: Size of chessboard squares in cm
        self.square_size = square_size
        #: Size of calibration images in pixels
        self.image_size = image_size
        pattern_size = (self.rows, self.columns)
        corner_coordinates = np.zeros((np.prod(pattern_size), 3), np.float32)
        corner_coordinates[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)
        corner_coordinates *= self.square_size
        #: Real world corner coordinates found in each image
        self.corner_coordinates = corner_coordinates
        #: Array of real world corner coordinates to match the corners found
        self.object_points = []
        #: Array of found corner coordinates from calibration images for left
        #: and right camera, respectively
        self.image_points = {"left": [], "right": []}

    def add_corners(self, image_pair, show_results=False):
        """
        Record chessboard corners found in an image pair.
```

```
        The image pair should be an iterable composed of two CvMats ordered
        (left, right).
        """
        side = "left"
        self.object_points.append(self.corner_coordinates)
        for image in image_pair:
            corners = self._get_corners(image)
            if show_results:
                self._show_corners(image, corners)
            self.image_points[side].append(corners.reshape(-1, 2))
            side = "right"
            self.image_count += 1

    def calibrate_cameras(self):
        """Calibrate cameras based on found chessboard corners."""
        criteria = (cv2.TERM_CRITERIA_MAX_ITER + cv2.TERM_CRITERIA_EPS,
                    100, 1e-5)
        flags = (cv2.CALIB_FIX_ASPECT_RATIO + cv2.CALIB_ZERO_TANGENT_DIST +
                 cv2.CALIB_SAME_FOCAL_LENGTH)
        calib = StereoCalibration()
        (calib.cam_mats["left"], calib.dist_coefs["left"],
         calib.cam_mats["right"], calib.dist_coefs["right"],
         calib.rot_mat, calib.trans_vec, calib.e_mat,
         calib.f_mat) = cv2.stereoCalibrate(self.object_points,
                                            self.image_points["left"],
                                            self.image_points["right"],
                                            self.image_size,
                                            criteria=criteria,
                                            flags=flags)[1:]
        (calib.rect_trans["left"], calib.rect_trans["right"],
         calib.proj_mats["left"], calib.proj_mats["right"],
         calib.disp_to_depth_mat, calib.valid_boxes["left"],
         calib.valid_boxes["right"]) = cv2.stereoRectify(calib.cam_mats["left"],
                                                         calib.dist_coefs["left"],
                                                         calib.cam_mats["right"],
                                                         calib.dist_coefs["right"],
                                                         self.image_size,
                                                         calib.rot_mat,
                                                         calib.trans_vec,
                                                         flags=0)
        for side in ("left", "right"):
            (calib.undistortion_map[side],
             calib.rectification_map[side]) = cv2.initUndistortRectifyMap(
                                                           calib.cam_mats[side],
                                                           calib.dist_coefs[side],
                                                           calib.rect_trans[side],
                                                           calib.proj_mats[side],
                                                           self.image_size,
                                                           cv2.CV_32FC1)
        # This is replaced because my results were always bad. Estimates are
        # taken from the OpenCV samples.
        #width, height = self.image_size
        #focal_length = 0.8 * width
        #calib.disp_to_depth_mat = np.float32([[1, 0, 0, -0.5 * width],
        #                                      [0, -1, 0, 0.5 * height],
        #                                      [0, 0, 0, -focal_length],
        #                                      [0, 0, 1, 0]])
        return calib

    def check_calibration(self, calibration):
        """
```

```
        Check calibration quality by computing average reprojection error.

        First, undistort detected points and compute epilines for each side.
        Then compute the error between the computed epipolar lines and the
        position of the points detected on the other side for each point and
        return the average error.
        """
        sides = "left", "right"
        which_image = {sides[0]: 1, sides[1]: 2}
        undistorted, lines = {}, {}
        for side in sides:
            undistorted[side] = cv2.undistortPoints(
                        np.concatenate(self.image_points[side]).reshape(-1,
                                                                        1, 2),
                        calibration.cam_mats[side],
                        calibration.dist_coefs[side],
                        P=calibration.cam_mats[side])
            lines[side] = cv2.computeCorrespondEpilines(undistorted[side],
                                            which_image[side],
                                            calibration.f_mat)
        total_error = 0
        this_side, other_side = sides
        for side in sides:
            for i in range(len(undistorted[side])):
                total_error += abs(undistorted[this_side][i][0][0] *
                                   lines[other_side][i][0][0] +
                                   undistorted[this_side][i][0][1] *
                                   lines[other_side][i][0][1] +
                                   lines[other_side][i][0][2])
            other_side, this_side = sides
        total_points = self.image_count * len(self.object_points)
        return total_error / total_points
```

3 – Classe para exibir o mapa de disparidade (incluem os métodos para preenchimento vertical, particionamento da imagem e verificação de quantidade de pixels em cada lado):

```
from argparse import ArgumentParser
import os
import time
import cv2
import numpy as np
from blockmatchers import StereoBM, StereoSGBM
from calibration import StereoCalibration
from ui_utils import find_files, BMTuner, STEREO_BM_FLAG
import pygame
pygame.init()
from pygame.locals import *
con_dir=0
con_esq=0
con_cen=0
aux=0
auxaudio=0
obstasculo=0
audio_cen = pygame.mixer.Sound('sons\centro.wav')
audio_dir = pygame.mixer.Sound('sons\direita.wav')
audio_esq = pygame.mixer.Sound('sons\esquerda.wav')
```

```
audio_obs = pygame.mixer.Sound('sons\obstaculo.wav')

def main():


    calibration_folder="c:\calibracao_resolmenor" #resultado bom resolucao 320

    image_folder="c:\imagens"
    parser = ArgumentParser(description="Read images taken from a calibrated "
                            "stereo pair, compute disparity maps from them and "
                            "show them interactively to the user, allowing the "
                            "user to tune the stereo block matcher settings in "
                            "the GUI.", parents=[STEREO_BM_FLAG])

    parser.add_argument("--bm_settings",
                        help="File to save last block matcher settings to.",
                        default="")
    args = parser.parse_args()
    audio = pygame.mixer.Sound('sons\inicializando_sistema_navegacao.wav')

    audio.play(0)


    calibration = StereoCalibration(input_folder=calibration_folder)
    input_files = find_files(image_folder)
    if args.use_stereobm:
        block_matcher = StereoBM(window_size=40)
    else:

        block_matcher = StereoSGBM(min_disparity=1, num_disp=96, sad_window_size=5,
                 uniqueness=1, speckle_window_size=3, speckle_range=32,
                 p1=216, p2=84, max_disparity=1, full_dp=False)
    capL = cv2.VideoCapture(2)
    capR = cv2.VideoCapture(1)

    capL.set(3, 320.)
    capL.set(4, 240.)
    capL.set(5, 40)
    capR.set(3, 320.)
    capR.set(4, 240.)
    capR.set(5, 40)



    ret, esquerda = capL.read()


    ret2, direita = capR.read()

    image_pair = [esquerda,direita]

    rectified_pair = calibration.rectify(image_pair)
    tuner = BMTuner(block_matcher, calibration, rectified_pair)

    kernel = np.ones((10,10),np.uint8)
    kernel2 = np.ones((50,5),np.uint8)

    while(True):


        ret, esquerda = capL.read()
```

```
        ret2, direita = capR.read()



        image_pair = [esquerda,direita]



        rectified_pair = calibration.rectify(image_pair)
        cv2.imshow('im_e',rectified_pair[0])


        disparity = block_matcher.get_disparity(rectified_pair)
        norm_coeff = 255 / disparity.max()
        cv2.imshow('disparity', disparity * norm_coeff / 255)
        disp=disparity * norm_coeff / 255



        #Thresholding
        ret,thresh4 = cv2.threshold(disp,0.2,5,cv2.THRESH_BINARY)
        cv2.imshow('thresh',thresh4)

        cv_img = thresh4.astype(np.uint8)



        img_linhasbrancas=pintabranco(cv_img)
        cv2.imshow('linhasbrancas',img_linhasbrancas)



        #erosao

        erosion = cv2.erode(img_linhasbrancas,kernel,iterations = 2)
        cv2.imshow('erosao',erosion)

        #dilatacao

        dilation = cv2.dilate(erosion,kernel,iterations = 4)
        cv2.imshow('dilatacao',dilation)
        contapixelsbrancos(dilation,320,240)


        #input_files = input_files[2:]
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break



    for param in block_matcher.parameter_maxima:
        print("{}\n".format(tuner.report_settings(param)))

    if args.bm_settings:
        block_matcher.save_settings(args.bm_settings)



def pintabranco(img):
    for j in range(1,320):
        i = 239
        while (i-1>0)&(img[i,j]==0):
```

```
            img[i,j]=255
            i=i-1
    return img;

def contapixelsbrancos(img,largura,altura):
    global
aux,con_dir,con_cen,con_esq,audio_cen,audio_dir,audio_esq,auxaudio,audio_obs,obstasculo


    esquerda = img[1:altura,1:(largura/3)+1]
    centro = img[1:altura,(largura/3)+2:(largura/3+(largura/3+1))]
    direita = img[1:altura,(largura/3+(largura/3+1)):largura]

    esq=0
    dir=0
    cen=0
    pixelalto=altura
    ladopixelalto=0
    for j in my_range(1,(largura/3)-1,3):
        i=altura-2
        while((i-1)>0):
            if(esquerda[i,j]==255):
                esq=esq+1;
                if(i<pixelalto):#selecao para pixel mais alto
                    pixelalto=i
                    ladopixelalto=1
            if(direita[i,j]==255):
                dir=dir+1
                if(i<pixelalto):
                    pixelalto=i
                    ladopixelalto=3
            i=i-1;
    for j in my_range(1,(largura/3)-2,3):
        i=altura-2
        while(i-1>=0):
            if(centro[i,j]==255):
                cen=cen+1
                if(i<pixelalto):
                    pixelalto=i
                    ladopixelalto=2
            i=i-1
    if(cen>dir) & (cen>esq) & (cen>(((largura/3)*altura)*0.2)):
        obstasculo=0
        if(aux==0)|(aux==2):
            con_cen=con_cen+1
            con_dir=0
            con_esq=0
        aux=2
        #print 'Centro'
    elif(esq>dir) & (esq>(((largura/3)*altura)*0.2)):
        obstasculo=0
        if(aux==0)|(aux==1):
            con_esq=con_esq+1
            con_cen=0
            con_dir=0
        aux=1
        #print 'Esquerda'
    elif (dir>(((largura/3)*altura)*0.2)):
        obstasculo=0
        if(aux==0)|(aux==3):
            con_dir=con_dir+1
```

```
            con_esq=0
            con_cen=0
        aux=3
    else:
        obstasculo=1

    #verifica se os ultimos 3 quadros foram realmente os que contiveram maior numero
de pixels, se a msg ja foi exibida(auxaudio) e se o maior pixel realmente esta no lado
com maior numero de pixels



    if(con_cen>3)&(auxaudio!=2)&(ladopixelalto==2):
        audio_cen.play(0)
        auxaudio=2
        print 'Centro'
    elif(con_esq>3)&(auxaudio!=1)&(ladopixelalto==1):
        audio_esq.play(0)
        auxaudio=1
        print 'Esquerda'
    elif(con_dir>3)&(auxaudio!=3)&(ladopixelalto==3):
        audio_dir.play(0)
        auxaudio=3
        print 'Direita'
    elif(obstasculo==1)&(auxaudio!=0):
        auxaudio=0
        audio_obs.play(0)



def my_range(start, end, step):
    while start <= end:
        yield start
        start += step

if __name__ == "__main__":
    main()
```

4 – Classe com as funções do mapa de disparidade:

```
import cv2
import simplejson

import numpy as np
from exceptionss import (InvalidSearchRangeError,
                                    InvalidWindowSizeError,
                                    InvalidBMPresetError,
                                    InvalidNumDisparitiesError,
                                    InvalidSADWindowSizeError,
                                    InvalidUniquenessRatioError,
                                    InvalidSpeckleWindowSizeError,
                                    InvalidSpeckleRangeError,
                                    InvalidFirstDisparityChangePenaltyError,
                                    InvalidSecondDisparityChangePenaltyError)


class BlockMatcher(object):



    #: Dictionary of parameter names associated with their maximum values
```

```
    parameter_maxima = {}

    def __init__(self, settings=None):
        """Set block matcher parameters and load from file if necessary."""
        #: Block matcher object used for computing point clouds
        self._block_matcher = None
        self._replace_bm()
        if settings:
            self.load_settings(settings)

    def load_settings(self, settings):
        """Load settings from file"""
        with open(settings) as settings_file:
            settings_dict = simplejson.load(settings_file)
        for key, value in settings_dict.items():
            self.__setattr__(key, value)

    def save_settings(self, settings_file):
        """Save block matcher settings to a file object"""
        settings = {}
        for parameter in self.parameter_maxima:
            settings[parameter] = self.__getattribute__(parameter)
        with open(settings_file, "w") as settings_file:
            simplejson.dump(settings, settings_file)

    @classmethod
    def get_3d(cls, disparity, disparity_to_depth_map):
        """Compute point cloud."""
        return cv2.reprojectImageTo3D(disparity, disparity_to_depth_map)

    def _replace_bm(self):
        """Replace block matcher with new parameters"""
        raise NotImplementedError

    def get_disparity(self, image_pair):
        """Compute disparity map from image pair."""
        raise NotImplementedError


class StereoBM(BlockMatcher):

    """A stereo block matching ``BlockMatcher``."""

    parameter_maxima = {"search_range": None,
                        "window_size": 255,
                        "stereo_bm_preset": cv2.STEREO_BM_NARROW_PRESET}

    @property
    def search_range(self):
        """Return private ``_search_range`` value."""
        return self._search_range

    @search_range.setter
    def search_range(self, value):
        """Set private ``_search_range`` and reset ``_block_matcher``."""
        if value == 0 or not value % 16:
            self._search_range = value
        else:
            raise InvalidSearchRangeError("Search range must be a multiple of "
                                          "16.")
        self._replace_bm()
```

```
    @property
    def window_size(self):
        """Return private ``_window_size`` value."""
        return self._window_size

    @window_size.setter
    def window_size(self, value):
        """Set private ``_window_size`` and reset ``_block_matcher``."""
        if (value > 4 and
            value < self.parameter_maxima["window_size"] and
            value % 2):
            self._window_size = value
        else:
            raise InvalidWindowSizeError("Window size must be an odd number "
                                         "between 0 and {}.".format(
                                         self.parameter_maxima["window_size"] + 1))
        self._replace_bm()

    @property
    def stereo_bm_preset(self):
        """Return private ``_bm_preset`` value."""
        return self._bm_preset

    @stereo_bm_preset.setter
    def stereo_bm_preset(self, value):
        """Set private ``_stereo_bm_preset`` and reset ``_block_matcher``."""
        if value in (cv2.STEREO_BM_BASIC_PRESET,
                     cv2.STEREO_BM_FISH_EYE_PRESET,
                     cv2.STEREO_BM_NARROW_PRESET):
            self._bm_preset = value
        else:
            raise InvalidBMPresetError("Stereo BM preset must be defined as "
                                       "cv2.STEREO_BM_*_PRESET.")
        self._replace_bm()

    def _replace_bm(self):
        """Replace ``_block_matcher`` with current values."""
        self._block_matcher = cv2.StereoBM(preset=self._bm_preset,
                                           ndisparities=self._search_range,
                                           SADWindowSize=self._window_size)

    def __init__(self, stereo_bm_preset=cv2.STEREO_BM_BASIC_PRESET,
                 search_range=80,
                 window_size=21,
                 settings=None):
        self._bm_preset = cv2.STEREO_BM_BASIC_PRESET
        self._search_range = 0
        self._window_size = 5
        #: OpenCV camera type for ``_block_matcher``
        self.stereo_bm_preset = stereo_bm_preset
        #: Number of disparities for ``_block_matcher``
        self.search_range = search_range
        #: Search window size for ``_block_matcher``
        self.window_size = window_size
        super(StereoBM, self).__init__(settings)

    def get_disparity(self, pair):
        """
        Compute disparity from image pair (left, right).

        First, convert images to grayscale if needed. Then pass to the
        ``_block_matcher`` for stereo matching.
```

```
        """
        gray = []
        if pair[0].ndim == 3:
            for side in pair:
                gray.append(cv2.cvtColor(side, cv2.COLOR_BGR2GRAY))
        else:
            gray = pair
        return self._block_matcher.compute(gray[0], gray[1],
                                           disptype=cv2.CV_32F)


class StereoSGBM(BlockMatcher):

    """A semi-global block matcher."""

    parameter_maxima = {"minDisparity": None,
                        "numDisparities": None,
                        "SADWindowSize": 11,
                        "P1": None,
                        "P2": None,
                        "disp12MaxDiff": None,
                        "uniquenessRatio": 15,
                        "speckleWindowSize": 200,
                        "speckleRange": 2,
                        "fullDP": 1}

    @property
    def minDisparity(self):
        """Return private ``_min_disparity`` value."""
        return self._min_disparity

    @minDisparity.setter
    def minDisparity(self, value):
        """Set private ``_min_disparity`` and reset ``_block_matcher``."""
        self._min_disparity = value
        self._replace_bm()

    @property
    def numDisparities(self):
        """Return private ``_num_disp`` value."""
        return self._num_disp

    @numDisparities.setter
    def numDisparities(self, value):
        """Set private ``_num_disp`` and reset ``_block_matcher``."""
        if value > 0 and value % 16 == 0:
            self._num_disp = value
        else:
            raise InvalidNumDisparitiesError("numDisparities must be a "
                                             "positive integer evenly "
                                             "divisible by 16.")
        self._replace_bm()

    @property
    def SADWindowSize(self):
        """Return private ``_sad_window_size`` value."""
        return self._sad_window_size

    @SADWindowSize.setter
    def SADWindowSize(self, value):
        """Set private ``_sad_window_size`` and reset ``_block_matcher``."""
        if value >= 1 and value <= 11 and value % 2:
```

```
            self._sad_window_size = value
        else:
            raise InvalidSADWindowSizeError("SADWindowSize must be odd and "
                                            "between 1 and 11.")
        self._replace_bm()

    @property
    def uniquenessRatio(self):
        """Return private ``_uniqueness`` value."""
        return self._uniqueness

    @uniquenessRatio.setter
    def uniquenessRatio(self, value):
        """Set private ``_uniqueness`` and reset ``_block_matcher``."""
        if value >= 5 and value <= 15:
            self._uniqueness = value
        else:
            raise InvalidUniquenessRatioError("Uniqueness ratio must be "
                                              "between 5 and 15.")
        self._replace_bm()

    @property
    def speckleWindowSize(self):
        """Return private ``_speckle_window_size`` value."""
        return self._speckle_window_size

    @speckleWindowSize.setter
    def speckleWindowSize(self, value):
        """Set private ``_speckle_window_size`` and reset ``_block_matcher``."""
        if value >= 0 and value <= 200:
            self._speckle_window_size = value
        else:
            raise InvalidSpeckleWindowSizeError("Speckle window size must be 0 "
                                                "for disabled checks or "
                                                "between 50 and 200.")
        self._replace_bm()

    @property
    def speckleRange(self):
        """Return private ``_speckle_range`` value."""
        return self._speckle_range

    @speckleRange.setter
    def speckleRange(self, value):
        """Set private ``_speckle_range`` and reset ``_block_matcher``."""
        if value >= 0:
            self._speckle_range = value
        else:
            raise InvalidSpeckleRangeError("Speckle range cannot be negative.")
        self._replace_bm()

    @property
    def disp12MaxDiff(self):
        """Return private ``_max_disparity`` value."""
        return self._max_disparity

    @disp12MaxDiff.setter
    def disp12MaxDiff(self, value):
        """Set private ``_max_disparity`` and reset ``_block_matcher``."""
        self._max_disparity = value
        self._replace_bm()
```

```
    @property
    def P1(self):
        """Return private ``_P1`` value."""
        return self._P1

    @P1.setter
    def P1(self, value):
        """Set private ``_P1`` and reset ``_block_matcher``."""
        if value < self.P2:
            self._P1 = value
        else:
            raise InvalidFirstDisparityChangePenaltyError("P1 must be less "
                                                          "than P2.")
        self._replace_bm()

    @property
    def P2(self):
        """Return private ``_P2`` value."""
        return self._P2

    @P2.setter
    def P2(self, value):
        """Set private ``_P2`` and reset ``_block_matcher``."""
        if value > self.P1:
            self._P2 = value
        else:
            raise InvalidSecondDisparityChangePenaltyError("P2 must be greater "
                                                           "than P1.")
        self._replace_bm()

    @property
    def fullDP(self):
        """Return private ``_full_dp`` value."""
        return self._full_dp

    @fullDP.setter
    def fullDP(self, value):
        """Set private ``_full_dp`` and reset ``_block_matcher``."""
        self._full_dp = bool(value)
        self._replace_bm()

    def _replace_bm(self):
        """Replace ``_block_matcher`` with current values."""
        self._block_matcher = cv2.StereoSGBM(minDisparity=self._min_disparity,
                        numDisparities=self._num_disp,
                        SADWindowSize=self._sad_window_size,
                        uniquenessRatio=self._uniqueness,
                        speckleWindowSize=self._speckle_window_size,
                        speckleRange=self._speckle_range,
                        disp12MaxDiff=self._max_disparity,
                        P1=self._P1,
                        P2=self._P2,
                        fullDP=self._full_dp)

    def __init__(self, min_disparity=16, num_disp=96, sad_window_size=3,
                 uniqueness=10, speckle_window_size=100, speckle_range=32,
                 p1=216, p2=864, max_disparity=1, full_dp=False,
                 settings=None):
        """Instantiate private variables and call superclass initializer."""
        #: Minimum number of disparities. Normally 0, can be adjusted as
        #: needed
        self._min_disparity = min_disparity
```

```
        #: Number of disparities
        self._num_disp = num_disp
        #: Matched block size
        self._sad_window_size = sad_window_size
        #: Uniqueness ratio for found matches
        self._uniqueness = uniqueness
        #: Maximum size of smooth disparity regions to invalid by noise
        self._speckle_window_size = speckle_window_size
        #: Maximum disparity range within connected component
        self._speckle_range = speckle_range
        #: Penalty on disparity change by +-1 between neighbor pixels
        self._P1 = p1
        #: Penalty on disparity change by multiple neighbor pixels
        self._P2 = p2
        #: Maximum left-right disparity. 0 to disable check
        self._max_disparity = max_disparity
        #: Boolean to use full-scale two-pass dynamic algorithm
        self._full_dp = full_dp
        #: StereoSGBM whose state is controlled
        self._block_matcher = cv2.StereoSGBM()
        super(StereoSGBM, self).__init__(settings)

    def get_disparity(self, pair):
        """Compute disparity from image pair (left, right)."""
        return self._block_matcher.compute(pair[0],
                                           pair[1]).astype(np.float32) / 16.0
```