

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

**Mapeamento de ambientes baseado em
algoritmos de visão estéreo para VANTs sobre
Linux Embarcado**

Autor: Nícolas dos Santos Rosa

Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2016

Nícolas dos Santos Rosa

Mapeamento de ambientes baseado em algoritmos de visão estéreo para VANTs sobre Linux Embarcado

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Evandro Luís Linhari Rodrigues

São Carlos

2016

Página com a ficha catalográfica (em página par).

página com a folha de aprovação (página ímpar).

Dedicatória

Este trabalho de conclusão de curso é dedicado à minha mãe, ao meu pai, à minha irmã, aos meus padrinhos e a toda minha família.

Nícolas dos Santos Rosa.

Agradecimentos

Primeiramente, agradeço a Deus por propiciar saúde e felicidade a todos aqueles que me rodeiam.

À minha família: à minha mãe, Valdenilce, pela ferrenha dedicação em mostrar a mim e a minha irmã a importância da educação para nossa formação pessoal; ao meu pai, Francisco, por fazer o possível e o impossível para sustentar nossa família e por possibilitar condições para que me tornasse engenheiro; à minha irmã, Natália, pelo suporte e carinho e por ser a dupla perfeita para atazarar nossos pais; às minhas tias, Elisabeth, Vera Lúcia e Maria Regina, por serem minhas segundas mães, visto o tamanho do suporte, preocupação, e apreço dado.

Ao meu orientador, Prof. Evandro Luís Linhari Rodrigues, pelo apoio para o desenvolvimento deste trabalho e por despertar meu interesse em sistemas embarcados, confirmando ainda mais minha paixão pelo meu curso.

Aos meus orientadores de iniciação científica, Prof. Dr. Cláudio F. M. Toledo e Márcio S. Arantes, por introduzir-me ao âmbito da pesquisa acadêmica. Aos membros do SARLab: ao Prof. Dr. Samir Rawashdeh por ser o idealizador deste trabalho e por oferecer a oportunidade de desenvolvê-lo; a Benjamin Dale e a Miguel Rocha Jr., pelo companheirismo e por sempre estarem sempre de prontidão.

Aos meus amigos de curso, pelos ótimos momentos que vivemos juntos nestes anos, especialmente, Alexandre B. Moretti, Leonardo B. Farconi, Plínio G. B. Ferreira, Marília L. Dourado, Jéssica B. da Vida, Pedro Arantes, Augusto Martins, Gustavo Oliveira, Aline Midori, Vitor Martins, Anderson M. Tsai, Victor Morini, João F. Corsini, Caio Martins e a todos os outros amigos.

Aos membros do grupo extracurricular Warthog Robotics, por partilhar um interesse em comum e pelas incontáveis horas de dedicação gastas no laboratório.

Por fim, agradeço a todos os envolvidos no desenvolvimento deste trabalho.

Nícolas dos Santos Rosa.

"O dinheiro faz homens ricos, o conhecimento faz homens sábios e a humildade faz grandes homens."

Mahatma Gandhi

"You can't put a limit on anything. The more you dream, the farther you get."

Michael Phelps

"Se eu vi mais longe, foi por estar sobre ombros de gigantes."

Isaac Newton

Resumo

Rosa, Nícolas **Mapeamento de ambientes baseado em algoritmos de visão estéreo para VANTs sobre Linux Embarcado.** Trabalho de Conclusão de Curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2016.

Atualmente, veículos aéreos não tripulados (VANT) vêm tornando-se um assunto recorrente no âmbito científico. Estes veículos, devido a sua mobilidade e inteligência artificial, vêm sendo adaptados para a atuação em diferentes ambientes, desempenhando assim diversas atividades que vão desde aplicações militares, agronômicas, espaciais, cinematográficas, entre outras. Entretanto, essa atuação só não é mais ampla devido a problemas relacionados ao reconhecimento do ambiente ao seu redor e detecção de objetos e obstáculos. Neste trabalho, estudou-se a utilização de visão estéreo em sistemas embarcados para mapeamento de ambientes e obstáculos ameaçam a locomoção do veículo autônomo. Os métodos estéreo mais conhecidos pela literatura, BM (*Block Matching*) e SGBM (*Semi-Global Block Matching*), foram implementados e também foi desenvolvido uma interface que facilite a extração de informações e a comparação de performance destes métodos. Após análise, o algoritmo mais robusto para a aplicação em veículos aéreos foi o método BM para ambas as plataformas, BBB e *Jetson TK1*. Visto que a *Jetson TK1* permite a aceleração em hardware do método BM, foi possível implementar o método BMGPU (*Block Matching with GPU Acceleration*) fornecido pelo OpenCV nesta plataforma. Por fim, os algoritmos utilizados permitiram que as distâncias de objetos próximos ao veículo móvel pudessem ser estimadas.

Palavras-Chave: Visão estéreo, Detecção de Obstáculos, Sistemas Embarcados, Veículos Aéreos Não Tripulados - VANT, Visão Computacional, OpenCV, Aceleração em GPU, CUDA.

Abstract

Rosa, Nícolas **Environment Mapping based on Stereo Vision algorithms for UAVs on Embedded Linux.** Completion of course work – São Carlos School of Engineering, University of São Paulo, 2016.

Currently, Unmanned Aerial Vehicles (UAV) are becoming a recurring theme in the scientific realm. These vehicles, because of their mobility and artificial intelligence, have been adapted to perform in different environments, thus performing various activities ranging from military applications, agronomic, spacial, cinematographic, among others. However, this performance is not wider due to problems related to the recognition of the surrounding environment and the detection objects and obstacles. In this work, it was studied the use of stereoscopic vision in embedded systems for environment mapping and obstacles that threaten the mobility of the autonomous vehicle. The most well known stereo methods in the literature, BM (*Block Matching*) and SGBM (*Semi-Global Block Matching*) were implemented and was developed a graphical user interface, which facilitates the extraction of the information and comparing performance of these methods. After analysis, the most robust algorithm for use in aerial vehicles was the BM method for both platforms, BBB and *Jetson TK1*. Since the *Jetson TK1* allows hardware acceleration of the method BM, it was possible to implement the method BMGPU (*Block Matching with GPU Acceleration*) provided by OpenCV on this platform. Finally, the used algorithms allowed the distances of obstacles near the moving vehicle could be estimated.

Keywords: Stereo Vision, Obstacle Detection, Embedded Systems, Unmanned aerial Vehicle - UAV, Computational Vision, OpenCV, GPU Acceleration, CUDA.

Lista de Figuras

2.1	Modelo Idealizado de um sistema de visão estéreo. Imagem retirada de [1]	32
2.2	Geometria Epipolar. Imagem retirada de [2]	33
2.3	O deslocamento do plano Π_r com relação ao plano Π_l pode ser descrito pela matriz R e o vetor de translação T. Imagem retirada de [1].	35
2.4	Retificação de Imagens - Processo de retificação estéreo. Imagem retirada de [1].	36
2.5	Correspondência de pontos homólogos em um par de imagens estéreo. Imagem original retirada de [3].	36
2.6	Relação inversamente proporcional entre distância e disparidade. Imagem retirada de [1]	37
2.7	Caminhão Autônomo desenvolvido pelo Laboratório de Robótica Móvel - LRM - ICMC/USP em parceria com a Scania.	39
2.8	Plataformas experimentais de aeronaves com o Sistema Estéreo - FPGA e o Sistema Estéreo - Pushbroom. Câmeras são montados na parte dianteira das asas na mesma linha de base (34 cm) em ambas células. Imagem retirada de [4].	39
2.9	Veículo Submarino Autônomo - Girona 500	40
3.1	3D Webcam Minoru	44
3.2	Câmera Digital Fujifilm FinePix Real 3D W3	45
3.3	Plataforma de Desenvolvimento - BeagleBone Black	46
3.4	Plataforma de Desenvolvimento - <i>Jetson TK1</i>	46
3.5	Padrão de Calibração	48
3.6	Quadricóptero 3DR X8 com suporte para a Câmera 3D	48
3.7	Cenário 1 - Ambiente Externo - Árvore	50
3.8	Cenário 2 - Ambiente Interno - Mesa/Cadeira/Estantes	50

3.9	Cenário 3 - Bancada/Quadricóptero	51
3.10	Calibração estéreo dos parâmetros intrínsecos - Modelo de Distorções da câmera esquerda e direita, respectivamente. Imagem obtida utilizando <i>Camera Calibration Toolbox for MATLAB</i> [5].	52
3.11	Calibração estéreo dos parâmetros extrínsecos - Estimativa do posicionamento da câmera direita ² em relação à câmera esquerda ¹ . Imagem obtida utilizando o aplicativo <i>Stereo Calibration App</i> do MATLAB [6].	53
3.12	Etapas do Processamento de Imagens	54
4.1	Interface Gráfica - Visualização simultânea dos quadros das câmeras esquerda e direita	61
4.2	Interface Gráfica - Visualização dos Mapa de disparidades em Escala de Cinza e RGB	61
4.3	Interface Gráfica - Visualização dos Mapa de disparidades em Escala de Cinza e RGB	62
4.4	Interface Gráfica - Visualização da Imagem da Câmera Esquerda com o indicador de objeto rastreado e da Imagem binária resultante da limiarização por distância	62
4.5	Interface Gráfica - Visualização da imagem resultante do processo de detecção de movimentos e imagem resultante do processo de detecção de movimentos limiarizada por distância	63
4.6	Interface Gráfica - Visualização da Imagem resultante da adição da imagem à direita com a Imagem da Câmera Esquerda e Imagem resultante do processo de realce das bordas dos objetos em movimento próximos ao veículo	63
4.7	Cenário 1 - Comparativo do Mapa de Disparidades gerado pelos Métodos BM, SGBM e BMGPU, respectivamente. Árvore como principal obstáculo.	65
4.8	Cenário 2 - Comparativo do Mapa de Disparidades gerados pelos Métodos BM, SGBM e BMGPU, respectivamente. Bancadas, Cone de segurança como principais obstáculos.	66
4.9	Cenário 3 - Comparativo do Mapa de Disparidades gerados pelos Métodos BM, SGBM e BMGPU, respectivamente. Piloto e Veículo aéreo (Quadricóptero) como principais obstáculos.	67

4.10 Gráfico - Comparação de Desempenho dos Métodos Estéreo para cada plataforma. 69

Lista de Tabelas

3.1	Especificações - 3D Webcam Minoru	44
3.2	Especificações - Câmera Digital Fujifilm FinePix Real 3D W3	45
3.3	Especificações - BeagleBone Black	46
3.4	Especificações - <i>Jetson TK1</i>	47
3.5	Especificações - Asus Q550LF	47
3.6	Versões utilizadas de CUDA e OpenCV	55
4.1	Configuração utilizada para Avaliação de Performance	68
4.2	Desempenho atingido por cada plataforma dos Métodos Utilizados	69

Siglas

AAVC	<i>Autonomous Aerial Vehicle Competition</i>
AFRL	<i>Air Force Research Laboratory</i> - Laboratório de pesquisas da Força Aérea Americana
ANAC	Agência Nacional de Aviação Civil
API	<i>Application Programming Interface</i> - Interface de Programação de Aplicações
AUV	<i>Autonomous Underwater Vehicle</i> - Veículo Submarino Autônomo
BBB	<i>BeagleBone Black</i>
BM	<i>Block Matching</i> - Método Estéreo Local
BMGPU	<i>Block Matching with GPU Acceleration</i> - Método Estéreo Local com Aceleração de GPU
CPU	<i>Central Processing Unit</i> - Unidade central de Processamento
CUDA	<i>Compute Unified Device Architecture</i> - Plataforma de computação paralela
DECEA	Departamento de Controle do Espaço Aéreo
FAA	<i>Federal Aviation Administration</i> - Administração Federal de Aviação
FPGA	<i>Field-programmable gate array</i> - Arranjo de Portas Programável em Campo
FPS	<i>Frames per Second</i> - Quadros por Segundo
GUI	<i>Graphical User Interface</i> - Interface gráfica do usuário
GPU	<i>Graphics Processing Unit</i> - Unidade de Processamento Gráfico
GPGPU	<i>General Purpose Graphics Processing Unit</i> - Unidade de Processamento Gráfico de Propósito Geral
GPS	<i>Global Positioning System</i> - Sistema de Posicionamento Global
LIDAR	<i>Light Detection And Ranging</i>
LMR	Laboratório de Robótica Móvel
MAVS	<i>Micro Air Vehicle</i> - Micro Veículo Aéreo
MIT	<i>Massachusetts Institute of Technology</i>
OACI	Organização de Aviação Civil Internacional
RPA	<i>Remotely Piloted Aircraft</i> - Aeronave Remotamente Pilotada
RTOS	<i>Real-Time Operating System</i> - Sistema Operacional em Tempo Real
SGBM	<i>Semi-Global Block Matching</i> - Método Estéreo Semi-global
SIMD	<i>Single Instruction, Multiple Data</i>
SLAM	<i>Simultaneous Localization and Mapping</i> - Localização e Mapeamento Simultâneo

UAV/VANT *Unmanned aerial vehicle* - Veículo Aéreo não tripulados

Sumário

1	Introdução	27
1.1	Objetivos	28
1.2	Justificativa	28
1.3	Motivação	29
1.4	Organização do trabalho	30
2	Fundamentos Teóricos	31
2.1	Visão Estéreo - <i>Stereo Vision</i>	31
2.1.1	Triangulação - <i>Triangulation</i>	31
2.1.2	Geometria epipolar - <i>Epipolar Geometry</i>	32
2.1.3	Calibração das Câmeras - <i>Calibration</i>	34
2.1.4	Retificação das Imagens - <i>Rectification</i>	35
2.1.5	Correspondência Estéreo - <i>Stereo Correspondence</i>	36
2.1.6	Aplicações em Robótica	38
2.2	OpenCV	40
2.3	Linux Embarcado	41
2.4	Aceleração em Hardware - CUDA	41
3	Materiais e Métodos	43
3.1	Materiais	43
3.1.1	Câmeras estéreo	43
3.1.2	Unidades de Processamento	45
3.1.3	Equipamentos auxiliares	47
3.2	Métodos	49
3.2.1	Cenários	49
3.2.2	Calibração	51

3.2.3	Processamento de Imagem	53
3.2.4	Aceleração em Hardware - <i>Hardware Acceleration</i> - CUDA	54
3.2.5	Jetson TK1 - Configuração da Plataforma	57
4	Resultados	59
4.1	Interface Gráfica - <i>StereoVisionGUI</i>	59
4.2	Cenários - Resultados Visuais	64
4.3	Comparação de Desempenho: Desktop x BBB x Jetson TK1	68
4.3.1	Desktop	70
4.3.2	BeagleBone Black (BBB)	70
4.3.3	Jetson TK1	71
5	Conclusão e Trabalhos Futuros	73
5.1	Conclusão	73
5.2	Trabalhos Futuros	75
A	Apêndice 1	83
I	Anexo 1	85

Capítulo 1

Introdução

A pesquisa em veículos aéreos não tripulados (VANTs) vem se tornando um assunto recorrente no âmbito científico. A real motivação para seu desenvolvimento levanta diversas questões éticas e legais, visto que foram inicialmente motivados para fins militares. Por outro lado, esse tipo de plataforma também possui aplicações tais como: cultivo e pulverização de culturas, produção cinematográfica, operações de busca e salvamento, inspeção de linhas elétricas de alta tensão, entrega de mercadorias e encomendas.

A navegação de um micro veículo aéreo em espaço confinado é um desafio significante. Atualmente, a navegação autônoma procura soluções para o chamado SLAM (*Simultaneous Localization and Mapping*) [7] que é um problema computacional que refere-se a dificuldade de construir e atualizar um mapa de um ambiente desconhecido enquanto simultaneamente rastreia-se os *landmarks* dentro dele. A solução deste problema é uma interação de quatro processos (Mapeamento, Percepção, Localização e Modelagem), cujos resultados são inteiramente dependentes. A complexidade do SLAM encontra-se no fato de que o veículo necessita navegar em um espaço desconhecido, extraír características importantes do ambiente, construir um mapa com os dados obtidos e simultaneamente localizar-se dentro deste. O sensoriamento pode ser realizado tanto por visão computacional utilizando câmeras ou por sensores ópticos, como, por exemplo, o LIDAR (*Light Detection And Ranging*) [8].

O processo de desenvolvimento do veículo consiste em quatro passos: estrutura, circuito, controle e navegação. Os dois primeiros itens compõem o hardware, o qual estabelece as conexões físicas necessárias para integrar os sistemas de alimentação, comunicação e controle. A parte de software engloba o desenvolvimento de algoritmos visando o controle e navegação, mais especificamente o desenvolvimento do código para visão estéreo (*Stereo Vision*), planejamento de caminho (*Path Planning*) e arquitetura de máquina de estados (*Decision*

Making) [9].

Um sistema autônomo também implica que o processamento de navegação, detecção de obstáculos, tomada de decisão, sejam embarcados, isto é, todo processamento necessário deve ser realizado *online*.

Este trabalho concentra-se no estudo de métodos estéreo e na aceleração destes algoritmos, realizando as otimizações cabíveis para que possam ser embarcados. Deste modo, as plataformas de desenvolvimento BeagleBone Black [10] e NVIDIA *Jetson TK1* [11] serão analisadas e suas performances avaliadas ao executar o algoritmo desenvolvido [12].

1.1 Objetivos

1. Estudo e aplicação de técnicas de visão computacional para visão estéreo.
2. Desenvolvimento de uma interface de apoio para o monitoramento de um veículo autônomo.
3. Utilização dos algoritmos do OpenCV para visão estéreo destinados a Linux embarcado e aplicação em Quadricópteros.
4. Comparativo de desempenho do algoritmo implementado em diferentes plataformas.
5. Estudo e aplicação de métodos para a aceleração em hardware dos algoritmos implementados.

1.2 Justificativa

Há pouco mais de trinta anos, o VANT BQM-1BR realizava seu primeiro voo em espaço aéreo brasileiro [13]. Deste então, mesmo a após a recente popularização dos *Drones*, a legislação com relação à esses veículos ainda é obscura. José Luiz Boa Nova Filho, gerente-adjunto do projeto VANT da Polícia Federal, apresenta o histórico e introduz a atual conjuntura na qual se encontra o processo legislativo [14]. No dia 19/11/2015, o Departamento de Controle do Espaço Aéreo da Aeronáutica (DECEA) e a Agência Nacional de Aviação Civil (ANAC) publicaram a nova regulamentação para a utilização dos *Remotely-Piloted Aircraft* (RPA), termo adotado para se referir aos VANTS, traduzido como "Aeronave Remotamente Pilotada", substituindo assim a Circular de Informações Aeronáuticas AIC N 21/10. A regulação segue o modelo proposto pela Organização de Aviação Civil Internacional (OACI), a

qual preza integralmente pela priorização da segurança, tanto da aeronave quanto dos civis e propriedades [15].

A *Federal Aviation Administration* (FAA) apresenta as mesmas dificuldades para a integração destes dispositivos em seu espaço aéreo, visto que este é o mais complexo e movimentado do mundo. Deste modo, mesmo sem uma legislação madura, rígidas restrições são impostas para voos em ambientes abertos. Entretanto, assim como as agências brasileiras, a FAA ainda não apresentou uma regulamentação clara envolvendo veículos totalmente autônomos.

Conclui-se que, mesmo sem um posicionamento concreto das agências reguladoras, a segurança destas aeronaves deve ser priorizada, assim permitindo a utilização e ampliação dessa nova tecnologia. Deste modo, o aprimoramento dos sensores para a percepção do ambiente ao redor destes equipamentos torna-se um passo crucial, justificando a execução deste trabalho, o qual estuda a utilização de visão estéreo para a detecção de obstáculos.

1.3 Motivação

A proposta deste trabalho de conclusão de curso é contribuir tecnicamente no conceito de percepção de ambientes através de visão estéreo. Também é motivado pela tentativa de reproduzir-se o desafio proposto pela *Autonomous Aerial Vehicle Competition* (AAVC) [16], competição organizada pelo Laboratório de Pesquisas da Força Aérea Americana (AFRL) e sediada em Dayton-OH. Esta competição incentiva o estudo de veículos aéreos autônomos, convidando diversas universidades a compartilhar seus avanços nesta área de pesquisa. O competidor é motivado a adaptar um modelo de quadricóptero 3DRobotics[©], assim este veículo precisa cumprir um certo percurso com caixas como obstáculos, detectar e reportar à estação base a posição de um objeto.

A segunda motivação para o desenvolvimento deste trabalho é o crescente número de aplicações de visão estéreo em plataformas robóticas. Atualmente, existem diversas pesquisas voltadas ao desenvolvimento de veículos autônomos para navegação terrestre, aérea e subaquática. A seção 2.1.6 traz mais detalhes.

1.4 Organização do trabalho

Esta monografia encontra-se estruturada em 5 capítulos: Introdução, Fundamentos Teóricos, Materiais e Métodos, Resultados e Conclusão. O primeiro capítulo sintetiza o trabalho desenvolvido e apresenta ao leitor suas reais pretensões. O capítulo 2 tem como conteúdo os principais conceitos teóricos para o seu entendimento, onde todo equacionamento e trabalhos similares são apresentados. O terceiro capítulo descreve todos os elementos necessários e técnicas utilizadas para sua realização. O quarto capítulo apresenta os resultados obtidos, onde são interpretados no último capítulo, o de Conclusão.

Capítulo 2

Fundamentos Teóricos

A visão estéreo possibilita a identificação de um espaço tridimensional, visto que sua estrutura permite a triangulação de pontos chaves, assim determinando o seu correto posicionamento. Deste modo, comprehende-se o porquê deste sistema visual ser amplamente difundido na evolução humana e animal. Em visão computacional, deseja-se emular os sistemas de visão mais eficientes para identificação de objetos e reconhecimento de ambientes. Este processo pode ser realizado computacionalmente, porém com alguns conceitos como Triangulação, Geometria Epipolar, Calibração e Retificação e Correspondência Estéreo. Estes conceitos encontram-se apresentados nas próximas seções.

2.1 Visão Estéreo - *Stereo Vision*

2.1.1 Triangulação - *Triangulation*

Idealmente, a triangulação de um ponto P de coordenadas globais (X, Y, Z) pode ser realizada caso tenha-se uma estrutura estéreo, cujas lentes não apresentem distorção e estejam perfeitamente alinhadas. Deste modo, matematicamente, é possível abstrair os sensores das câmeras como dois planos coplanares entre si. Nessas condições, tem-se que os eixos ópticos das câmeras são paralelos. O eixo óptico, também conhecido como raio principal, é a reta que intercepta o ponto de centro de projeção O e o ponto principal da lente c . Assumindo que as câmeras sejam exatamente iguais e alinhadas, tem-se que os pontos focais da câmera esquerda e da câmera direita são iguais $f_l = f_r$ e os pontos principais c_x^{left} e c_x^{right} apresentam as mesmas coordenadas [1]. A figura 2.1 ilustra a representação do modelo idealizado de um sistema estéreo.

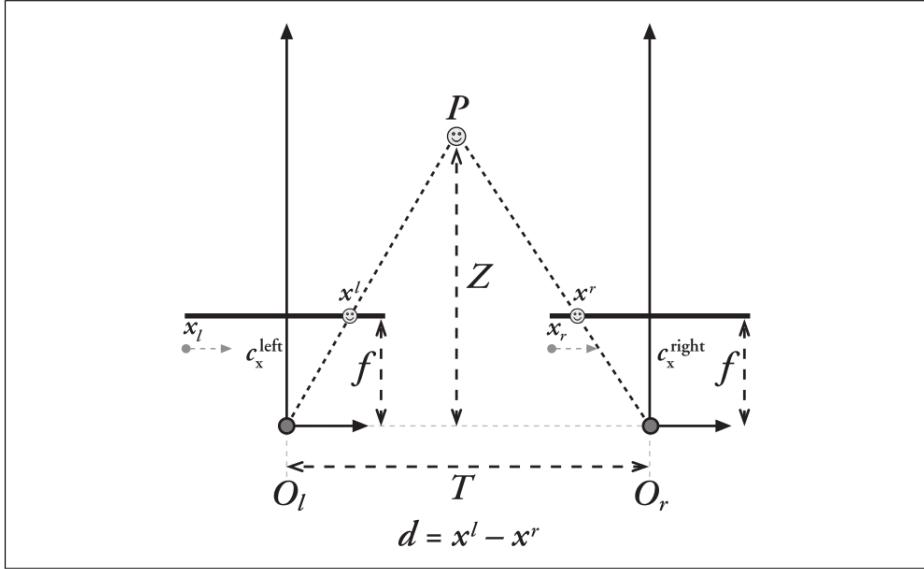


Figura 2.1: Modelo Idealizado de um sistema de visão estéreo. Imagem retirada de [1]

A distância presente entre os pontos x^l e x^r é dada pela equação $d = x^l - x^r$, o valor de d é comumente também chamado de disparidade. Caso os pontos x^l e x^r , a distância focal f , a distância entre os centros das câmeras (T , *baseline*) sejam conhecidos é possível determinar a distância entre o ponto P à base das câmeras (Z). Por meio de semelhanças de triângulos, é possível estabelecer uma relação entre os triângulos O_lPO_r e x^lPx^r , a qual está apresentada na equação 2.1.

$$\frac{T - (x^l - x^r)}{Z - T} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{(x^l - x^r)} = \frac{fT}{d} \quad (2.1)$$

2.1.2 Geometria epipolar - *Epipolar Geometry*

Geometria epipolar corresponde à estrutura básica de um sistema estéreo, na qual leva em consideração os modelos *pinhole* de ambas câmeras utilizadas e encontra-se ilustrada pela figura 2.2.

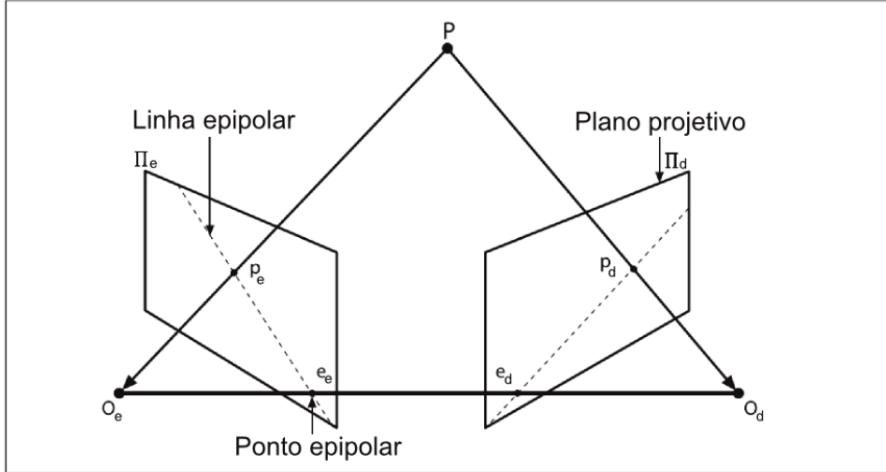


Figura 2.2: Geometria Epipolar. Imagem retirada de [2]

Projeta-se o ponto P nos centros de projeção O^e e O^d , as linhas que interligam o ponto P a esses centros interceptam os planos Π_e e Π_d nos pontos p_e e p_d . Os pontos epipolares (*epipoles*) e_e e e_d estão localizados nos pontos de intersecção da linha que interliga os centros de projeção e os planos projetivos [2]. O conhecimento desta geometria é importante pra o entendimento da chamada restrição epipolar (*epipolar constraint*).

Considerando um sistema estéreo e um certo ponto P , deseja-se localizar na imagem da direita o seu ponto homólogo P' , isto é, sua projeção no plano projetivo direito. Sem a restrição, faz-se necessário a busca bidimensional em todo espaço do plano Π_d . Por outro lado, essa restrição garante que o ponto homólogo deve estar sobre a linha epipolar da imagem direita. Deste modo, é possível restringir a busca a uma única dimensão, busca somente sobre a linha epipolar, reduzindo consideravelmente o custo computacional [1]. Todavia, o sistema estéreo deve estar corretamente calibrado para que possa tomar mão desta restrição. Em visão computacional, a chamada matriz fundamental refere-se a matrix 3x3 utilizada para relacionar os pontos correspondentes do par de imagens estéreo. A expressão 2.2 que representa a restrição epipolar é dada pela relação entre a matriz fundamental F e os dois pontos correspondentes, em coordenadas de pixel [17].

$$p'^T F p = 0 \quad (2.2)$$

2.1.3 Calibração das Câmeras - *Calibration*

Até o presente momento, todos os conceitos apresentados assumiam que as câmeras são idealmente alinhadas e que suas lentes não apresentavam distorção. Na realidade, os centros ópticos não são perfeitamente alinhados e a lente introduz distorções à imagem projetada no sensor da câmera. Deste modo, faz-se necessário o processo de calibração, o qual mensura estas deformidades e estima os parâmetros que consigam anular ou minimizar estas imperfeições. Isso permite que os métodos computacionais obtenham resultados mais precisos. Estes parâmetros são classificados em dois tipos específicos: intrínsecos e extrínsecos.

Parâmetros intrínsecos

Correspondem às propriedades intrínsecas de cada câmera, as quais são descritas pela matriz M (*Intrinsic Matrix*) e o vetor D (*Distortion Coefficients Vector*). A figura 3.10 ilustra o modelo completo, componente radial e componente tangencial do modelo de distorções das lentes da câmera esquerda e direita, respectivamente [18].

$$M = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (2.3)$$

f_x, f_y : Distância focal

c_x, c_y : Compensação do ponto principal

$$D = (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6]]) \quad (2.4)$$

k_1, k_2, k_3 : Coeficientes de distorção radial simétrica

p_1, p_2 : Coeficientes de distorção tangencial (descentrada)

Parâmetros extrínsecos

Correspondem às propriedades extrínsecas, isto é, demonstram a disposição espacial da segunda câmera, direita, com relação à câmera da esquerda em coordenadas globais. Assim como ilustrado na figura 2.3, a matriz de rotação R (3x3) e o vetor de translação T (3x1) são responsáveis por descrever este deslocamento. Geralmente, quando o quadro de referência não se encontra no centro de projeção da câmera, faz-se necessário a adição da matriz de rotação e o vetor de translação. Neste contexto, utiliza-se o sistema de coordenadas homogêneas,

isto é, pontos 2D são representados como vetores 3×1 e pontos 3D como vetores 4×1 . Deste modo, tem-se que a equação 2.6 descreve a relação do ponto $P(X, Y, Z)$ e o ponto projetado p' no plano Π_r [17].

$$s.p' = M \begin{bmatrix} R & | & t \end{bmatrix} P \quad (2.5)$$

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2.6)$$

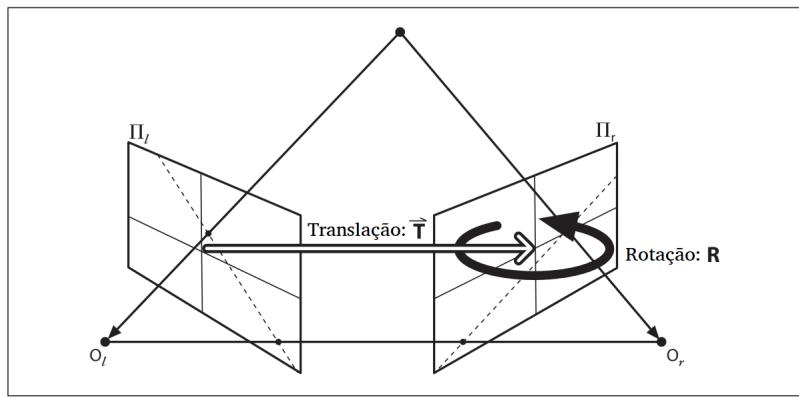


Figura 2.3: O deslocamento do plano Π_r com relação ao plano Π_l pode ser descrito pela matriz R e o vetor de translação T . Imagem retirada de [1].

Devido ao desalinhamento entre as câmeras, torna-se necessário estimar estas variáveis, aumentando, assim, a eficiência dos métodos estéreo ao procurarem pelas correspondências entre as duas imagens. A figura 3.11, ilustra o processo de calibração dos parâmetros extrínsecos, no qual utiliza-se um conjunto de imagens do padrão de calibração. Ao fim deste processo, é possível aferir o posicionamento da segunda câmera com relação à primeira [5].

2.1.4 Retificação das Imagens - *Rectification*

O processo de retificação é responsável por realizar as correções com relação à distorção das lentes e ao alinhamento das câmeras, de acordo com os parâmetros obtidos pelo processo de calibração apresentado no tópico anterior. Ao fim deste processo, deseja-se que o par de imagens estéreo esteja retificado e sem distorções, preparado para a aplicação dos métodos estéreo, assim como ilustrado na figura 2.4.

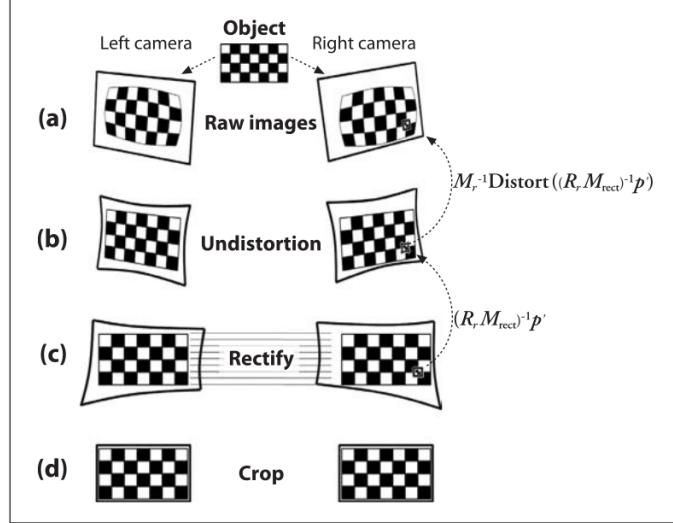


Figura 2.4: Retificação de Imagens - Processo de retificação estéreo. Imagem retirada de [1].

2.1.5 Correspondência Estéreo - *Stereo Correspondence*

Correspondência estéreo não é nada mais que a utilização de métodos estéreo, os quais são responsáveis pela procura de pontos homólogos nos pares de imagens estéreo. Como visto anteriormente, devido ao distanciamento das câmeras (T) e ao distanciamento do objeto com relação às câmeras (Z), o mesmo ponto apresenta diferentes posicionamento nos planos das câmeras x^l e x^r . A figura 2.5 ilustra um par de imagens estéreo (esquerda e direita) e o mapa de disparidades gerado pela cena, nas quais os pixels homólogos estão destacados. Os métodos estéreo utilizam da restrição epipolar o que reduz o espaço de busca e de diferentes meios para encontrá-los. Mesmo com essa restrição e com a retificação das imagens, os métodos ainda assim apresentam um elevado custo computacional, além do que ainda estão sujeitos à encontrarem falsas correspondências.



Figura 2.5: Correspondência de pontos homólogos em um par de imagens estéreo. Imagem original retirada de [3].

Mapa de disparidades - *Disparity Map*

Como já foi dito anteriormente, disparidade é o deslocamento dos pontos homólogos entre as duas imagens. Nos métodos estéreo, o valor da disparidade é codificado em escala de cinza, a qual é inversamente proporcional à distância do objeto, assim como representado na figura 2.6. Assim, níveis de cinza mais altos (tons claros) correspondem a disparidades maiores (perto) e níveis de cinza mais baixos (tons escuros) correspondem a disparidades mais baixas (distante). Devido a sua relação com a distância, este conceito é comumente atrelado à percepção de profundidade.

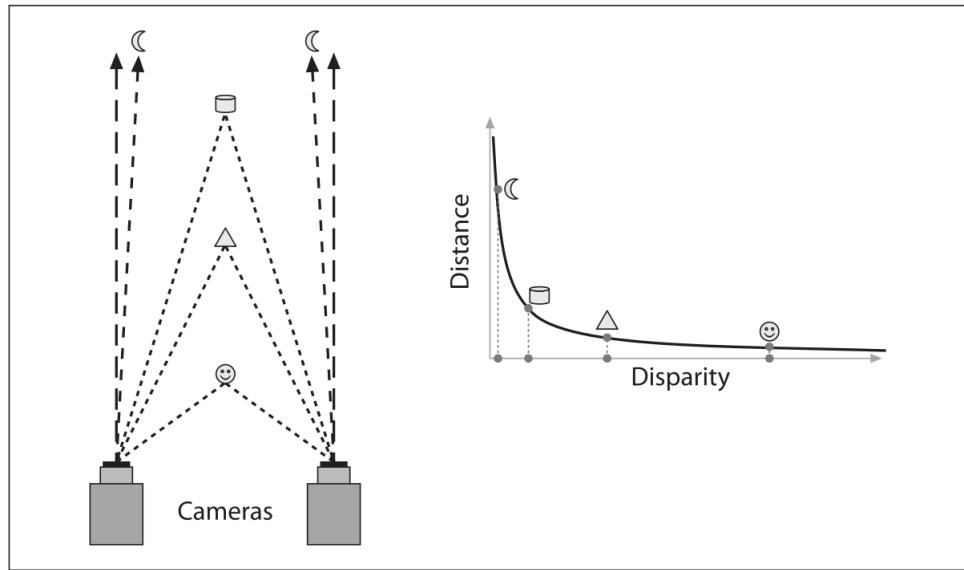


Figura 2.6: Relação inversamente proporcional entre distância e disparidade. Imagem retirada de [1]

Métodos Estéreo

A seguir, encontra-se apresentado um pequeno resumo de como são os operadores dos métodos estéreo utilizados neste trabalho.

1. **Block Matching - BM:** Este algoritmo é um método local, no qual utiliza soma das diferenças ao quadrado (SSD - *Sum of Squared Differences*) para a determinação de correspondências. O tamanho da vizinhança deve ser avaliado, visto que a densidade do mapa gerado, nível de detalhamento e intensidade de ruído dependem disso. Caso o bloco seja pequeno, o mapa apresentará detalhes mais nítidos, porém adicionará ruídos ao mapa. Por outro lado, um bloco maior reduz o nível de ruído, porém diminui o nível de detalhamento do mapa. As limitações serviram de motivação para a criação do método a seguir [19].

2. ***Semi-global Block Matching - SGBM:*** Este algoritmo é um método global e também visa a obtenção de uma correspondência estéreo mais precisa, para isso, além da etapa de agregação de custo (cálculo da similaridade), modela-se o sistema como um problema de minimização energética, o que adiciona restrições que suavizam o mapa ao penalizar descontinuidades. [20]. Por conta disso, este método é mais lento que o apresentado anteriormente, porém é mais denso ao comparar-se o nível de detalhamento para um bloco de tamanho pequeno [19].

3. ***Block Matching with GPU Acceleration - BMGPU:*** Método é igual ao primeiro apresentado, diferindo do fato que as suas bibliotecas e métodos implementados são voltados para a aceleração em GPU (*Graphics Processing Unit*). Dado o paralelismo intrínseco dos aceleradores gráficos de hoje, esse tipo de *hardware* permite o processamento simultâneo dos dados.

2.1.6 Aplicações em Robótica

Nesta seção, serão apresentados alguns dos trabalhos que têm como principais sensores câmeras estéreo para a navegação autônoma em ambientes terrestre, aérea e subaquática.

Nos dias de hoje, as empresas automobilísticas vêm participando de uma verdadeira corrida tecnológica para o desenvolvimento de automóveis totalmente autônomos e economicamente viáveis. As universidades não ficaram para trás e também apresentam pesquisas envolvendo desenvolvimento de algoritmos de controle e sensores, tornando essa corrida ainda mais acirrada. Os resultados apresentados são realmente promissores, e concretizam cada vez mais essa realidade tida até então como distante. Na figura 2.7, é possível observar o projeto de caminhão autônomo desenvolvido pelo Laboratório de Robótica Móvel - LRM - ICMC/USP. O caminhão conta com diversos sensores, dentre eles câmeras estéreo, que identificam outros automóveis, pessoas e faixas de sinalização [21].



Figura 2.7: Caminhão Autônomo desenvolvido pelo Laboratório de Robótica Móvel - LRM - ICMC/USP em parceria com a Scania.

No caso de navegação autônoma para ambiente aéreo, o projeto utilizando *Micro Air Vehicle* (MAVS) desenvolvido pelo *Massachusetts Institute of Technology* (MIT) permite que pequenas aeronaves consigam navegar autonomamente e desviar de obstáculos voando a uma velocidade de 30 mph (48 km/h). A figura 2.8 apresenta o trabalho desenvolvido, o qual é um comparativo de desempenho de uma implementação em hardware utilizando *Field-programmable gate array* (FPGA) e um processador ARM para processamento embarcado do método SGBM [4].

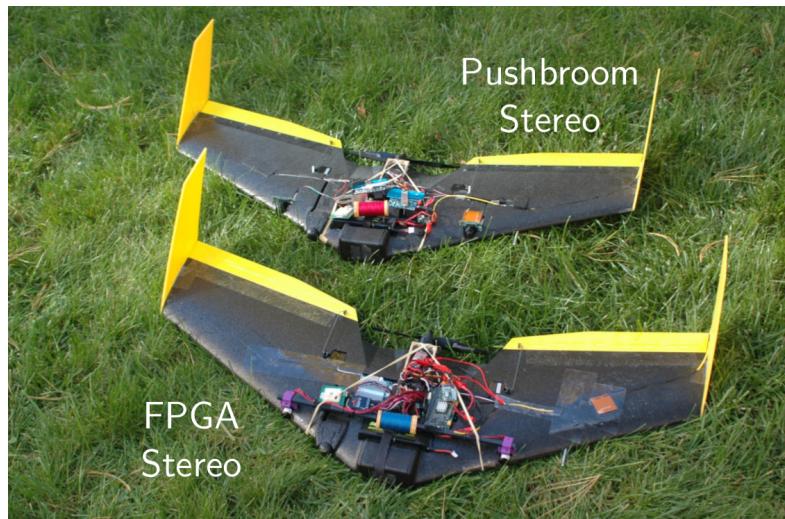


Figura 2.8: Plataformas experimentais de aeronaves com o Sistema Estéreo - FPGA e o Sistema Estéreo - Pushbroom. Câmeras são montados na parte dianteira das asas na mesma linha de base (34 cm) em ambas células. Imagem retirada de [4].

No caso de navegação autônoma para ambiente subaquático, um exemplo de *autonomous underwater vehicle* (AUV) é o projeto desenvolvido pela Universidade espanhola de Girona (veja figura 2.9). O trabalho propõe a utilização do método de Mapeamento e Localização Simultânea (SLAM), juntamente com câmera estéreo, para o reconhecimento do ambiente, aprimorando assim o erro de rastreamento dos objetos [22].

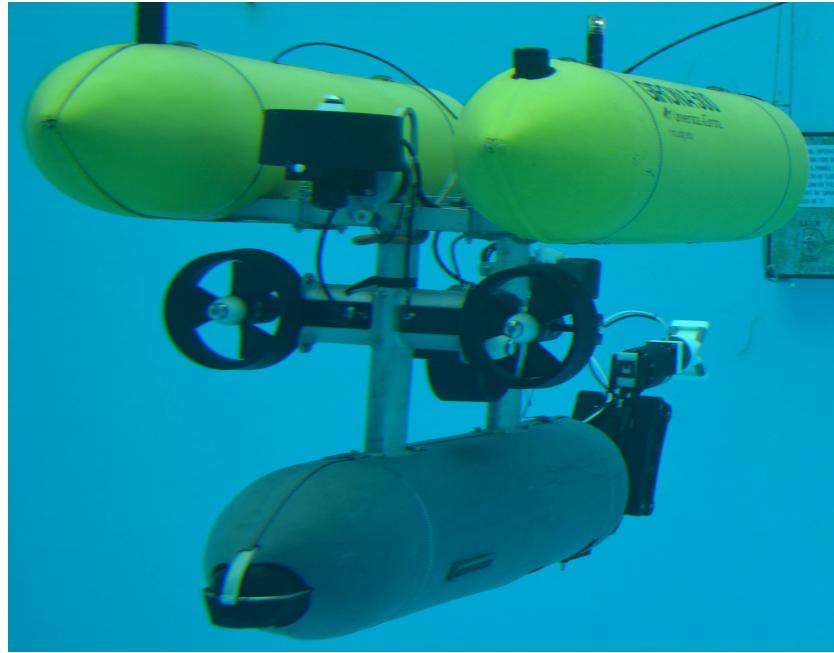


Figura 2.9: Veículo Submarino Autônomo - Girona 500

2.2 OpenCV

Open Source Computer Vision Library (OpenCV) é uma biblioteca de código aberto, a qual reúne diversos algoritmos relacionados a visão computacional. O OpenCV foi criado pelo centro de pesquisa da *Intel*, e atualmente é mantido pelo time de pesquisadores da companhia *Itseez* [23]. A biblioteca é estruturada em diferentes módulos, os quais agrupam os algoritmos relacionados a tópicos como processamento de imagem, estimativa de movimento, correspondência estéreo, extractores de características, criação de interfaces de usuário e aceleração em *hardware*. O OpenCV foi criado com a intensão de ser multiplataforma, por conta disso, em sua maior parte, foi escrito em linguagem C/C++, o que o torna facilmente portátil para diversas plataformas como Windows, Linux, Android, MacOS, iOS, dentre outros. Recentemente, módulos dando suporte à CUDA (seção 2.4) e OpenGL foram incorporados à biblioteca expandindo ainda mais o número de aplicações possíveis [24].

2.3 Linux Embarcado

Linux é um sistema operacional de código-aberto que é amplamente utilizado, cujo *Kernel*, originalmente escrito por Linus Torvalds, pode suportar diversos designs de sistemas podendo ser um computador pessoal, um supercomputador ou *cluster*, ou até mesmo sistemas *System-on-Chip* (SoC). Este sistema é capaz de suportar diversas arquiteturas de processadores como x86_64, PowerPC, ARM, MIPS, StrongARM, XScale, dentre outras [25].

A popularização do termo Linux fez com que atualmente exista uma certa confusão sobre a distinção de um *Kernel* de um sistema operacional e do sistema operacional propriamente dito. A maior razão desta confusão é a variedade de distribuições desenvolvidas pela comunidade, visto a facilidade para customizar seus módulos, pacotes, sistemas de arquivo e servidores gráficos. Dada sua versatilidade, logo tornou-se a peça chave para o desenvolvimento de sistemas embarcados, os quais são computadores completamente especializados para a realização de tarefas específicas. Esses sistemas diferem de computadores de propósito geral e comumente apresentam certas restrições de dimensão, condições de operação, ou limitação de recursos computacionais como menor poder de processamento ou menor capacidade de armazenamento.

Assim, começaram a surgir distribuições de Linux, os chamados *Embedded Linux*, que apresentavam menores requisitos de sistemas, devido à remoção de módulos que não são utilizados em determinados sistemas ou à otimização desses módulos para aquela atividade específica [26].

2.4 Aceleração em Hardware - CUDA

Compute Unified Device Architecture (CUDA) é uma tecnologia desenvolvida pela NVIDIA e em sua essência é uma plataforma de computação paralela de propósito geral, cujo objetivo é tirar proveito das unidades de processamento gráfico (GPU), assim, acelerando consideravelmente a execução de algoritmos computacionais complexos ao comparar-se com o desempenho da CPU.

Atualmente, a maioria das placas de vídeo da NVIDIA contam com essa tecnologia, por conta disso os pesquisadores e desenvolvedores de software estão voltando suas pesquisas para o desenvolvimento e otimização de algoritmos que possam explorar todo o poder de processamento destes dispositivos. Alguns exemplos de aplicações são identificação de placas ocultas em artérias, análise do fluxo de tráfego aéreo [27] e visualização de moléculas

[28].

Historicamente, a implementação de algoritmos em GPU mostrou-se bastante obscura, visto que sua programação era bastante atrelada ao hardware a ser utilizado, mesmo com linguagens de programação gráfica como o OpenGL. Em 2003, um grupo de pesquisadores de Stanford, apresentou o primeiro compilador, Brook, que facilitaria a implementação de software, visto que permitia lidar de uma maneira mais maleável o fluxo de dados, podendo principalmente paralelizá-los. Em 2006, a NVIDIA juntamente com Ian Buck, criador do Brook, desenvolveu uma plataforma (CUDA) mais intuitiva e que permitisse a utilização de uma linguagem de alto nível e tornasse a GPU em um processador de propósito geral (GPGPU) [29].

Capítulo 3

Materiais e Métodos

Nesta seção, serão apresentados os equipamentos necessários e métodos utilizados para o desenvolvimento do projeto. No caso dos equipamentos, serão apresentados todas as especificações técnicas e sua importância para o trabalho. Na seção destinada aos métodos, os algoritmos desenvolvidos para identificação e reconhecimento de objetos serão descritos.

3.1 Materiais

Com relação aos equipamentos é possível classificá-los em três grupos distintos: câmeras estéreo, unidades de processamento, e equipamentos auxiliares.

3.1.1 Câmeras estéreo

As câmeras utilizadas para aplicações em visão estéreo apresentam uma série de requisitos para que seja possível desenvolver um sistema que seja facilmente embarcável e que gere um mapa de disparidades denso de qualidade, isto é, um mapa com elevado grau de detalhamento e com menor susceptibilidade a erros. Um dos requisitos que este trabalho exige é que *Stereo Rig*, estrutura na qual as câmeras são fixadas, seja o mais alinhado possível. Vale ressaltar que o espaçamento das câmeras está estritamente relacionado com o espaçamento das lentes (*Baseline*). Outro requisito é que essa mesma estrutura seja coerente com o tamanho do veículo e seja leve, consequentemente, diminui-se o esforço exigido pelo veículo, por exemplo, para alçar voo no caso de quadricópteros. Outro requisito é que as câmeras apresentem uma elevada taxa de captura de quadros e que sejam sincronizados, isto é, os quadros de ambas as câmeras sejam capturados no mesmo instante. Os quadros capturados devem ser disponibilizados para a plataforma embarcada via conexão USB, *FireWire*, ou algum outro tipo de

Tabela 3.1: Especificações - 3D Webcam Minoru

Sensor de Imagem	VGA CMOS Sensor
Resolução Máxima	800x600
Distância entre Sensores (<i>Baseline</i>)	6 cm
Taxa de Captura	30 fps
Distância Focal	10 cm até ∞
Campo de Visão	42°
Peso	249.48 g

conexão que permita a transmissão em tempo real.

O projeto utilizou duas câmeras estéreo. Primeiramente, utilizou-se a *webcam* Minoru (veja figura 3.1), visto que apresentava preço totalmente acessível e cumpria o requisito de realizar *streaming* via USB. Deste modo, tornou-se um equipamento essencial para a implementação dos métodos para encontro de correspondências entre as câmeras. A tabela 3.1 apresenta as especificações da *webcam*.



Figura 3.1: 3D Webcam Minoru

Atualmente, a câmera utilizada é a digital 3D W3 fabricada pela Fujifilm (veja figura 3.2). A primeira câmera foi substituída, pois o controlador USB não permitia que a webcam realizasse *streaming* na máxima resolução. Deste modo, optou-se por uma com maior resolução e que apresentasse lentes com baixa distorção. Entretanto, essa não apresenta *streaming* via USB, assim é necessário que os vídeos sejam processados *offline*. Visto que o projeto se preocupa principalmente na geração do mapa de disparidades, isso não oferece nenhuma desvantagem para o desenvolvimento do trabalho. Todavia, para uma aplicação real, a câmera instalada no veículo deve apresentar esse aspecto. A tabela 3.2 apresenta as especificações da

Tabela 3.2: Especificações - Câmera Digital Fujifilm FinePix Real 3D W3

Sensor de Imagem	10 MP CCD Sensor
Resolução Máxima	1280x720
Distância entre Sensores (<i>Baseline</i>)	7.5 cm
Taxa de Captura	24 - 30 fps
Distância Focal	60 cm até ∞
Peso	250g

câmera em questão.



Figura 3.2: Câmera Digital Fujifilm FinePix Real 3D W3

3.1.2 Unidades de Processamento

Visto que este trabalho busca a implementação dos métodos estéreo em quadricópteros, tem-se como objetivo sua implementação para Linux embarcado (*Embedded Linux*). Em seguida, estão apresentadas as plataformas que foram utilizadas para este propósito.

BeagleBone Black

Uma das unidades de processamento utilizadas foi a plataforma aberta BeagleBone Black (BBB), ilustrada pela figura 3.3. Esta plataforma foi escolhida devido ao seu tamanho reduzido, podendo ser facilmente embarcada, isto é, é possível adaptá-la mecanicamente ao veículo. Com relação ao seu poder de processamento, ela apresenta um processador ARM Cortex-A8 operando à 1 GHz. A tabela 3.3 apresenta as especificações da plataforma.

Tabela 3.3: Especificações - BeagleBone Black

Processador	1GHz TI Sitara AM3359 ARM Cortex-A8
RAM	512 MB DDR3L @ 400 MHz
Armazenamento	2 GB on-board eMMC, MicroSD
Sistemas Operacionais	Angstrom (Default), Ubuntu, Android, dentre outros...
Consumo de energia	210-460 mA @ 5V
Pinos de GPIO	65/92 pinos
Periféricos	1 USB Host, 1 Mini-USB Client, 1 10/100 Mbps Ethernet

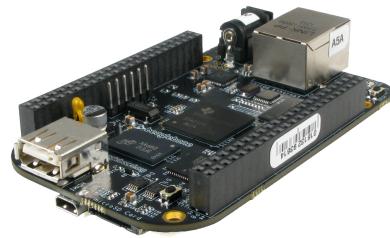


Figura 3.3: Plataforma de Desenvolvimento - BeagleBone Black

Jetson TK1

Outra unidade de processamento utilizada foi a plataforma *Jetson TK1* produzida pela NVI-DIA, ilustrada pela figura 3.4. Essa plataforma conta com um processador de 32-bits Tegra K1 baseado na tecnologia ARM Cortex-A15. O motivo pelo qual esta plataforma foi escondida é devido ao seu poder de processamento gráfico, visto que apresenta 192 núcleos gráficos, sendo assim adequada para aplicações envolvendo processamento de imagens. A tabela 3.4 apresenta as especificações da plataforma. Outro fator interessante desta plataforma é que ela oferece suporte à tecnologia CUDA, a qual será discutida mais adiante.

Figura 3.4: Plataforma de Desenvolvimento - *Jetson TK1*

Tabela 3.4: Especificações - *Jetson TK1*

Processador	NVIDIA 2.32GHz ARM quad-core Cortex-A15
Processador Gráfico	NVIDIA Kepler "GK20a"GPU with 192 SM3.2 CUDA cores
DRAM	2GB DDR3L 933MHz EMC x16 using 64-bit data width
Armazenamento	16GB fast eMMC 4.51 (routed to SDMMC4)
Sistemas Operacionais	Platform 64-bit Linux Ubuntu 14.04
Consumo de energia	0.6W to 3W @ 12 V
Pinos de GPIO	7 x GPIO pins (1.8V)
Periféricos	USB, mini-PCIe, SATA, SD-card, HDMI, audio

Tabela 3.5: Especificações - *Asus Q550LF*

Especificações CPU	
CPU	Intel Core i7 (4th Gen) 4500U / 1.8 3.0 GHz
Number of Cores	Dual-Core
Memory	DDR3 SDRAM 8 GB
Especificações de GPU	
Chipset	NVIDIA
Architecture	Kepler
GPU	GK107 384 @ 837 MHz
Memory	DDR3 - 2048 MB - 128 Bit @ 1800 MHz
CUDA Cores	384 Cores
Features	Optimus, GPU Boost 2.0, PhysX, Verde Drivers, CUDA, 3D Vision, 3DTV Play

Notebook Asus Q550LF

Também foi utilizado um *Notebook* Asus Q550LF, ele foi utilizada para o desenvolvimento de todos os programas contidos neste trabalho e para estudo de desempenho comparativo com as plataformas embarcadas. A interface desenvolvida, *StereoVisionGUI*, foi desenvolvida para ser executada nesta máquina e em computadores com arquitetura x86 e x64. As especificações desta plataforma estão apresentadas pela tabela 3.5.

3.1.3 Equipamentos auxiliares

Nesta seção, estão apresentados os equipamentos auxiliares para o desenvolvimento do trabalho.

Os métodos para a identificação de correspondências entre as câmeras requerem que as imagens estejam calibradas e retificadas. Por conta disso, utiliza-se o padrão de calibração de dimensão 7x10, apresentado na figura 3.5, para este propósito. Deste modo, é possível caracterizar as distorções das lentes, parâmetros intrínsecos, e o posicionamento de uma das câmeras com relação a outra, parâmetros extrínsecos.

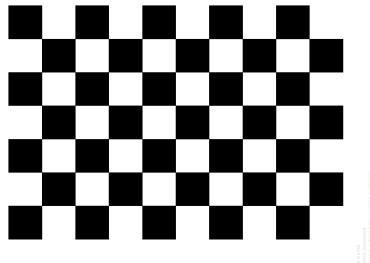


Figura 3.5: Padrão de Calibração

A motivação deste trabalho é a sua utilização em veículos aéreos. Por conta disso, é indispensável que se tenha algum desses veículos. O trabalho conta com a utilização de um quadricóptero produzido pela 3DR, porém este apresenta modificações visando o seu desenvolvimento para navegação autônoma. Deste modo, tem-se a adição de *propellers guards*, objetivando o aumento da segurança do veículo e das pessoas que o operam. Além disso, o *drone* conta com suportes para a câmera estéreo e para a plataforma embarcada. Como pode ser observado na figura 3.6, todas as peças foram produzidas utilizando impressora 3D.



Figura 3.6: Quadricóptero 3DR X8 com suporte para a Câmera 3D

3.2 Métodos

Nesta seção, serão apresentados os cenários e os procedimentos utilizados na implementação dos métodos estéreo apresentados.

3.2.1 Cenários

O pós-processamento do mapa de disparidades foi voltado para a identificação e detecção de obstáculos. Deste modo, os seguintes cenários propõem uma série de adversidades, as quais o algoritmo implementado tenta contorná-las. Propõe-se que ele deve ser flexível a variações na luminosidade, capaz de detectar obstáculos estáticos e móveis, e ser imune a vibrações. Deste modo, dois cenários em ambiente confinado e um em ambiente aberto foram analisados. Deseja-se a navegação autônoma ocorra até mesmo em casos que o sinal do Sistema de Posicionamento Global (GPS) seja perdido, por conta disso escolheu-se a utilização dos ambientes confinados. O ambiente externo foi escolhido devido a quantidade de fatores externos que poderiam atrapalhar a detecção de obstáculos. O tratamento destes percalços torna o programa ainda mais robusto.

Cenário 1

O cenário da figura 3.7 foi utilizado para estudo das condições de ambiente externo, o qual está sujeito grandes variações de luminosidade e um número menor de movimentos, o que permite uma análise de alcances maiores. O principal obstáculo deste cenário é uma árvore.

Cenário 2

O cenário da figura 3.8 foi utilizado para estudo das condições de ambiente interno, o qual também apresenta certa variação de luminosidade, porém apresenta um número maior de movimentos, permitindo uma análise de objetos estáticos à curta e média distância. Os principais obstáculos deste cenário são uma mesa, uma cadeira e duas estantes.

Cenário 3

O cenário da figura 3.9 foi utilizado para estudo das condições de ambiente interno, o qual é semelhante ao cenário anterior com relação à luminosidade e o alcance analisado. A cena difere apenas na inserção de uma outra aeronave, a qual realiza o papel de um obstáculo

móvel. Os principais obstáculos deste cenário é uma bancada e um outro quadricóptero no campo de visão do veículo pilotado.



Figura 3.7: Cenário 1 - Ambiente Externo - Árvore

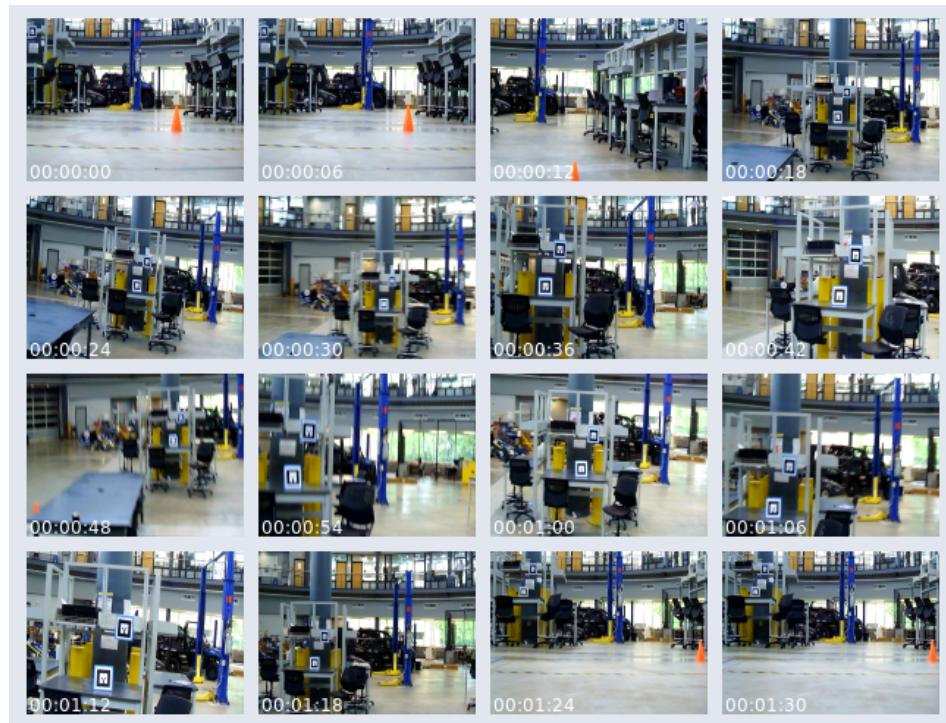


Figura 3.8: Cenário 2 - Ambiente Interno - Mesa/Cadeira/Estantes

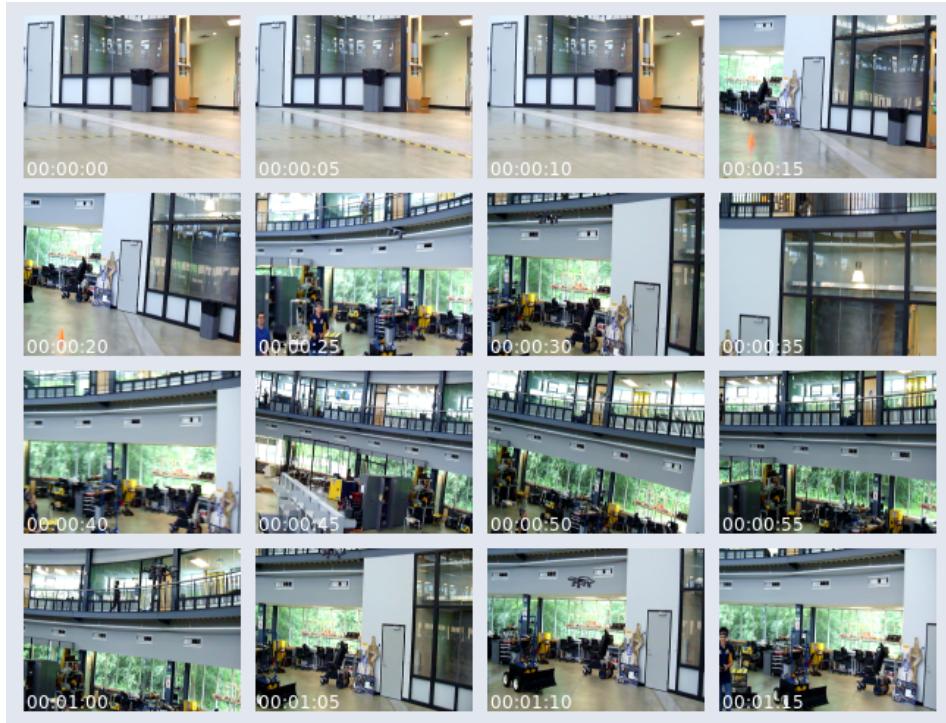


Figura 3.9: Cenário 3 - Bancada/Quadricóptero

3.2.2 Calibração

O processo de calibração pode ser realizado de duas maneiras diferentes.

O primeiro método é utilizando a rotina disponibilizada pelo OpenCV [30], no qual algumas informações relevantes, como o tipo do padrão utilizado, o tamanho do padrão e o número de quadros, precisam ser informadas ao executá-la. Este método dispensa um conjunto de imagens de entrada para o processo de calibração, pois o próprio, automaticamente, se encarrega de detectar o padrão de calibragem e gravar as imagens utilizadas no processo. Ao fim da execução, as imagens são analisadas e os arquivos "*intrinsics.yml*" e "*extrinsics.yml*" são gerados, os quais contêm as matrizes que corrigem as distorções apresentadas na seção 2.1.3.

O segundo método é utilizando o aplicativo *Stereo Camera Calibrator* presente no MATLAB. Ele também gera as mesmas informações anteriores, porém permite uma análise ainda mais profunda das distorções das lentes, como o erro de reprojeção 2D de cada imagem ou por pixel. Entretanto, é necessário que um conjunto de pares de imagens seja disponibilizado para o aplicativo para que a calibração seja realizada. Além disso, o MATLAB disponibiliza gráficos que representam as distorções das ambas as lentes, assim como apresentado na imagem 3.10, e a estimativa de reconstrução da cena utilizada no processo de calibração,

onde estima-se o posicionamento de cada perspectiva das imagens utilizadas com relação às câmeras, assim como ilustrado na figura 3.11.

Parâmetros intrínsecos

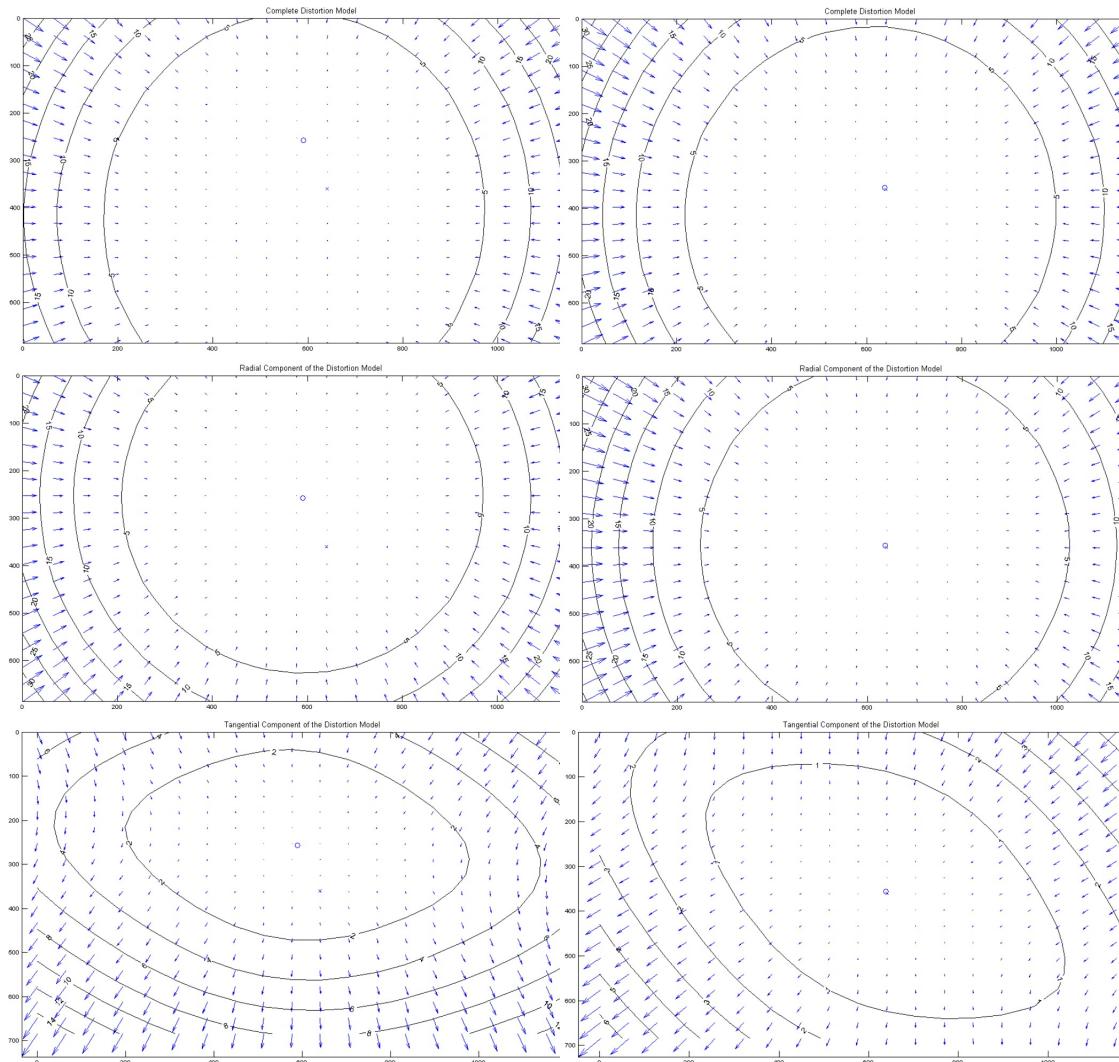


Figura 3.10: Calibração estéreo dos parâmetros intrínsecos - Modelo de Distorções da câmera esquerda e direita, respectivamente. Imagem obtida utilizando *Camera Calibration Toolbox for MATLAB* [5].

Parâmetros extrínsecos

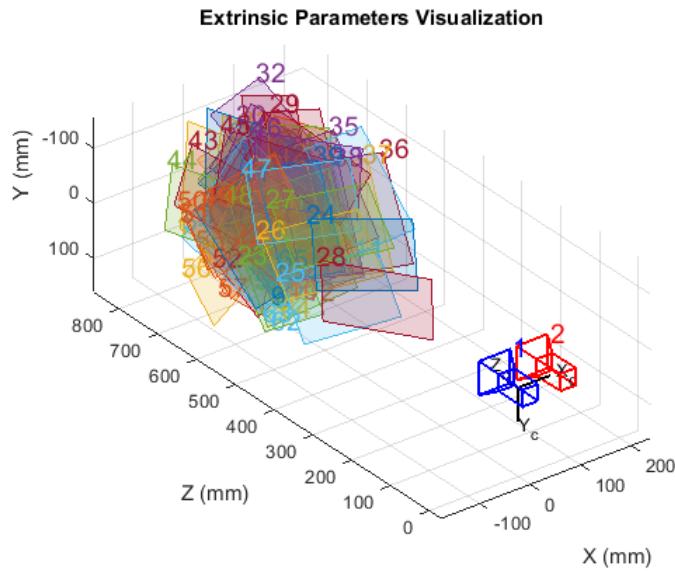


Figura 3.11: Calibração estéreo dos parâmetros extrínsecos - Estimativa do posicionamento da câmera direita² em relação à câmera esquerda¹. Imagem obtida utilizando o aplicativo *Stereo Calibration App* do MATLAB [6].

3.2.3 Processamento de Imagem

Nessa seção será apresentado o processamento de imagens utilizado para a identificação de obstáculos. Como pode ser visto na figura 3.12, todo o processo conta com seis etapas.

Câmeras: O primeiro passo do processo é a captura das imagens da câmera estéreo. Idealmente, as imagens devem ser capturadas ao mesmo instante e as lentes não apresentarem distorções.

Calibração e Retificação: Na prática, as lentes apresentam distorção. Com base nos parâmetros obtidos após a calibração das câmeras é possível retificá-las.

Correspondência Estéreo: Aplica-se os métodos para encontrar as correspondências entre as duas câmeras, gerando assim o mapa de disparidades.

Filtragem: Este passo, pode ser aplicado tanto nas imagens retificadas ou no mapa de disparidades. Atualmente, aplica-se a operação morfológica de abertura e um filtro de mediana sobre o mapa de disparidades.

Limiarização por Distância: Visto que a disparidade apresenta uma relação com a distância, aplica-se uma operação de limiarização. Deste modo, apenas os obstáculos a uma certa distância são segmentados.

Identificação de Obstáculos: Após o passo anterior, o objeto é identificado e sua posição é rastreada. Essa informação pode ser utilizada pelo sistema de controle da aeronave para manter distância do obstáculo identificado. Esta etapa encontra-se realçada na figura 3.12, pois o processo de segmentação e identificação de objetos varia conforme o tipo de obstáculo e as condições da imagem (claro, escuro, muito brilho, baixo contraste). Deste modo, uma árvore de opções surge a partir deste processo, todos com o intuito de contornar as adversidades presentes no ambiente.



Figura 3.12: Etapas do Processamento de Imagens

3.2.4 Aceleração em Hardware - *Hardware Acceleration - CUDA*

Neste trabalho, utilizou-se diferentes versões desta API (*Application Programming Interface*) para o desenvolvimento para *Desktop* e *Jetson TK1*. Neste contexto, essas versões não oferecem nenhuma alteração visível, visto que não foi desenvolvido nenhum tipo de rotina especificamente para execução em GPU. Na verdade, utilizou-se as funções disponíveis na biblioteca OpenCV, as quais opcionalmente podem ser executadas pela CPU ou GPU. A tabela 3.6 informa as versões do pacote de ferramentas da plataforma CUDA e da biblioteca OpenCV utilizadas.

Tabela 3.6: Versões utilizadas de CUDA e OpenCV

	CUDA ToolKit	OpenCV
Desktop	7.5	3.0.0
Jetson TK1	6.5	2.4.12

Desktop - Instalação e Configuração

Abaixo, estão apresentadas as instruções necessárias para a instalação e a configuração do ambiente de desenvolvimento para que plataforma CUDA funcione corretamente [31]. O seguinte processo também pode ser executado para a configuração do ambiente na *Jetson TK1*. Entretanto, um processo muito mais intuitivo e recomendado para essa plataforma está apresentado na seção 3.2.5.

1. Instalando a CUDA

Primeiramente, as seguintes instruções são compatíveis somente em máquinas com Ubuntu 14.04+ e com processadores gráficos da NVIDIA com capacidade de computação 3.5 ou maior.

1.1. Execute o seguinte comando:

```
$ ubuntu@ubuntu:~$ sudo apt-get install build-essential
```

Caso o usuário esteja utilizando uma máquina virtual, é necessário que os seguintes comandos sejam executados:

```
$ ubuntu@ubuntu:~$ sudo apt-get update
$ ubuntu@ubuntu:~$ sudo apt-get install linux-generic
```

1.2. Baixe o arquivo .deb da CUDA no seguinte link: <https://developer.nvidia.com/cuda-downloads>, e verifique se a versão do arquivo é correspondente ao sistemas operacional utilizado. Neste trabalho, utilizou-se o Linux Ubuntu 14.04 64-bit e o arquivo condizente chamava-se "cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb" e foi possível instalá-lo recorrendo aos seguintes comandos:

1.3. Execute os seguintes comandos:

```
$ ubuntu@ubuntu:~$ cd Downloads
$ ubuntu@ubuntu:~/Downloads$ sudo dpkg -i cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb
$ ubuntu@ubuntu:~/Downloads$ sudo apt-get update
$ ubuntu@ubuntu:~/Downloads$ sudo apt-get install cuda
```

1.4. Configure as variáveis do ambiente necessárias para o desenvolvimento em programação CUDA:

```
$ ubuntu@ubuntu:~/Downloads$ nano ~/.bashrc

Edite o arquivo /home/<user>/.bashrc e adicione o seguinte trecho:

# CUDA Environment Setup
# CUDA 6.5
#export PATH=/usr/local/cuda-6.5/bin:$PATH
#export LD_LIBRARY_PATH=/usr/local/cuda-6.5/lib64:$LD_LIBRARY_PATH

# CUDA 7.5 (active)
export PATH=/usr/local/cuda-7.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH

$ ubuntu@ubuntu:~/Downloads$ source ~/.bashrc
```

1.5. Reinicie o computador.

2. Instalando do OpenCV com Suporte à CUDA [32]

2.1. Instalando os pacotes necessários

```
$ ubuntu@ubuntu:~$ sudo apt-get update

$ ubuntu@ubuntu:~$ sudo apt-get install libopencv-dev build-essential checkinstall
      cmake pkg-config yasm libtiff4-dev libjpeg-dev libjasper-dev
      libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev
      libxine-dev libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev
      libv4l-dev python-dev python-numpy libtbb-dev libqt4-dev libgtk2.0-dev
      libfaac-dev libmp3lame-dev libopencore-amrnb-dev libopencore-amrwb-dev
      libtheora-dev libvorbis-dev libxvidcore-dev x264 v4l-utils
$ ubuntu@ubuntu:~$ sudo add-apt-repository ppa:jon-severinsson/ffmpeg
$ ubuntu@ubuntu:~$ sudo apt-get update
$ ubuntu@ubuntu:~$ sudo apt-get install ffmpeg
$ ubuntu@ubuntu:~$ sudo apt-get install frei0r-plugins
```

2.2. Clonando o repositório do OpenCV. Neste trabalho, utilizou-se a versão 3.0.0:

```
$ ubuntu@ubuntu:~$ mkdir OpenCV
$ ubuntu@ubuntu:~$ cd OpenCV
$ ubuntu@ubuntu:~/OpenCV$ git clone https://github.com/Itseez/opencv.git
```

2.3. Instalando e realizando o *build* da biblioteca OpenCV. Este passo é de extrema importância, pois é aqui que configura-se o OpenCV para dar suporte à CUDA. Certifique-se que a arquitetura de GPU configurada na flag CUDA_GENERATION corresponda realmente a da sua plataforma:

```
$ ubuntu@ubuntu:~/OpenCV$ mkdir build && cd build
$ ubuntu@ubuntu:~/OpenCV/build$ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local
-D WITH_TBB=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON
-D WITH_QT=ON -D WITH_OPENGL=ON -D ENABLE_FAST_MATH=1 -D CUDA_FAST_MATH=1 -D WITH_CUBLAS=1 -D CUDA_GENERATION=Kepler ...
$ ubuntu@ubuntu:~/OpenCV/build$ make -j4
$ ubuntu@ubuntu:~/OpenCV/build$ sudo make install
```

2.4. Configurando o *LIBRARY SEARCH PATH*:

```
$ ubuntu@ubuntu:~/$ sudo sh -c 'echo "/usr/local/lib" > /etc/ld.so.conf.d/opencv.conf'
$ ubuntu@ubuntu:~/$ sudo ldconfig
```

3.2.5 Jetson TK1 - Configuração da Plataforma

A execução das rotinas de visão estéreo com aceleração via GPU requisitam que a plataforma de desenvolvimento esteja corretamente configurada. Abaixo, encontra-se o tutorial para configurá-la para a operação desejada, isto é, basicamente, configurá-la para o correto funcionamento da CUDA e do OpenCV2.

1. Baixando o JetPack L4T

- 1.1. Clique aqui para ir ser redirecionado para a página do pacote de desenvolvimento Jetpack L4T. Caso o link esteja quebrado, vá até a página da NVIDIA e procure pela localização correta do pacote.
- 1.2. Na máquina *Host* rodando Ubuntu, crie um novo diretório para armazenar os pacotes de instalação utilizando as seguintes linhas de comando.

```
$ ubuntu@ubuntu:~$ mkdir flash
$ ubuntu@ubuntu:~$ cd flash
$ ubuntu@ubuntu:~/flash$
```

Uma ação importante que merece destaque é baixar o arquivo JetPack-\$VERSION.run dentro do diretório criado (o caminho NÃO DEVE conter espaços).

- 1.3. Dê permissão de execução para o arquivo baixado.

```
$ ubuntu@ubuntu:~/flash$ chmod +x JetPack-${VERSION}.run
$ ubuntu@ubuntu:~/flash$ ./JetPack-${VERSION}.run
```

2. Instalando o JetPack L4T

Siga as instruções no manual de usuário do kit de desenvolvimento da *Jetson TK1*

- 2.1. Baixe o guia de instruções no seguinte link e clique na aba "Install Guide". Link:
<https://developer.nvidia.com/embedded/jetpack>
- 2.2. Siga as instruções do instalador do Jetpack L4T.

Uma informação que merece destaque é que o computador *Host* e o dispositivo alvo (*Jetson TK1*) DEVEM estar conectados à mesma rede. Isso pode ser feito conectando:

- 2.2.1. Máquina *Host* e dispositivo alvo à mesma Intranet, no caso o dispositivo alvo tenha um endereço IP estático.

Edite o /etc/network/interfaces :

auto lo

```

iface lo inet loopback
auto eth0
iface eth0 inet static
address 10.235.0.133
netmask 255.255.252.0
network 10.235.3.0
gateway 10.235.0.1
pre-up ifconfig eth0 hw ether 00:01:02:03:05:09
dns-nameservers 143.107.225.6 143.107.182.2 8.8.8.8

```

2.2.2. Máquina *Host* e dispositivo alvo ao mesmo roteador. Neste caso, o dispositivo alvo DEVE ser capaz de encontrar o endereço IP por protocolo DHCP.

Edite o /etc/network/interfaces:

```

auto eth0
allow-hotplug eth0
iface eth0 inet dhcp

```

3. Instalando o OpenCV com Módulo GPU na *Jetson TK1*

Primeiramente, deve-se baixar e instalar o pacote de ferramentas CUDA, uma vez que é necessário para OpenCV. Cabe ao usuário decidir se deseja se utilizar a biblioteca pré-compilada ou compilar a biblioteca do código-fonte. A primeira opção é a mais recomendada, visto que a biblioteca pré-compilada é a OpenCV4Tegra, uma versão otimizada em CPU e GPU do OpenCV para a *Jetson TK1*. A segunda opção é recomendada caso deseja-se adicionar módulos que não estão presentes na versão OpenCV4Tegra [33].

Capítulo 4

Resultados

Os resultados apresentados neste capítulo estão divididos em duas partes. Os resultados da seção 4.1 estão inteiramente relacionados às funções fundamentais presentes na interface gráfica desenvolvida. Já na seção 4.3, têm-se presente os resultados que apresentam os dados obtidos por meio dos testes de desempenho dos métodos de *Stereo Matching* nas plataformas utilizadas.

4.1 Interface Gráfica - *StereoVisionGUI*

Nesta seção, o software desenvolvido apresenta uma interface gráfica (GUI – *Graphical User Interface*) amigável, a qual facilita a visualização das imagens da câmera estéreo, dos mapas de disparidades, da reconstrução tridimensional e do método de processamento de imagens desenvolvido. Atualmente, o software conta com três métodos para encontrar correspondências estéreo (BM, SGBM e BMGPU) e com 8 opções das quais 6 delas são destinadas a visualização das seguintes perspectivas:

1. Imagens retificadas das câmeras esquerda e direita
2. Mapa de disparidades em Escala de Cinza e RGB
3. Mapa tridimensional do ambiente reconstruído em Escala de Cinza e RGB
4. Imagem da Câmera Esquerda com o indicador de objeto rastreado e Imagem binária resultante da limiarização por distância.
5. Imagem resultante do processo de detecção de movimentos e Imagem resultante do processo de detecção de movimentos limiarizada por distância.

6. Imagem resultante da adição da imagem à direita com a Imagem da Câmera Esquerda e Imagem resultante do processo de realce das bordas dos objetos em movimento próximos ao veículo.

O botão *Show Left/Right* seleciona a opção na qual a interface gráfica permite a visualização simultânea das imagens retificadas de ambas câmeras. A figura 4.1 ilustra o comportamento do software quando essa opção é selecionada.

O botão *Show Disparity Map* seleciona a opção na qual a interface gráfica permite a visualização simultânea dos mapas de disparidade em escala de cinza e RGB. A figura 4.2 ilustra o comportamento do software quando essa opção é selecionada.

O botão *Show 3D Reconstruction* seleciona a opção na qual a interface gráfica permite a visualização simultânea dos mapas tridimensionais do ambiente reconstruído em escala de cinza e RGB. A figura 4.3 ilustra o comportamento do software quando essa opção é selecionada.

O botão *Show Tracking Object View* seleciona a opção na qual a interface gráfica permite a visualização simultânea da imagem da câmera Esquerda com o indicador de objeto rastreado e imagem binária resultante da limiarização por distância. A figura 4.4 ilustra o comportamento do software quando essa opção é selecionada.

O botão *Show DiffImage* seleciona a opção na qual a interface gráfica permite a visualização simultânea da imagem resultante do processo de detecção de movimentos e imagem resultante do processo de detecção de movimentos limiarizada por distância. A figura 4.5 ilustra o comportamento do software quando essa opção é selecionada.

O botão *Show Warning Edges* seleciona a opção na qual a interface gráfica permite a visualização simultânea da imagem resultante da adição da imagem à direita com a imagem da câmera esquerda e imagem resultante do processo de realce das bordas dos objetos em movimento próximos ao veículo. A figura 4.6 ilustra o comportamento do software quando essa opção é selecionada.

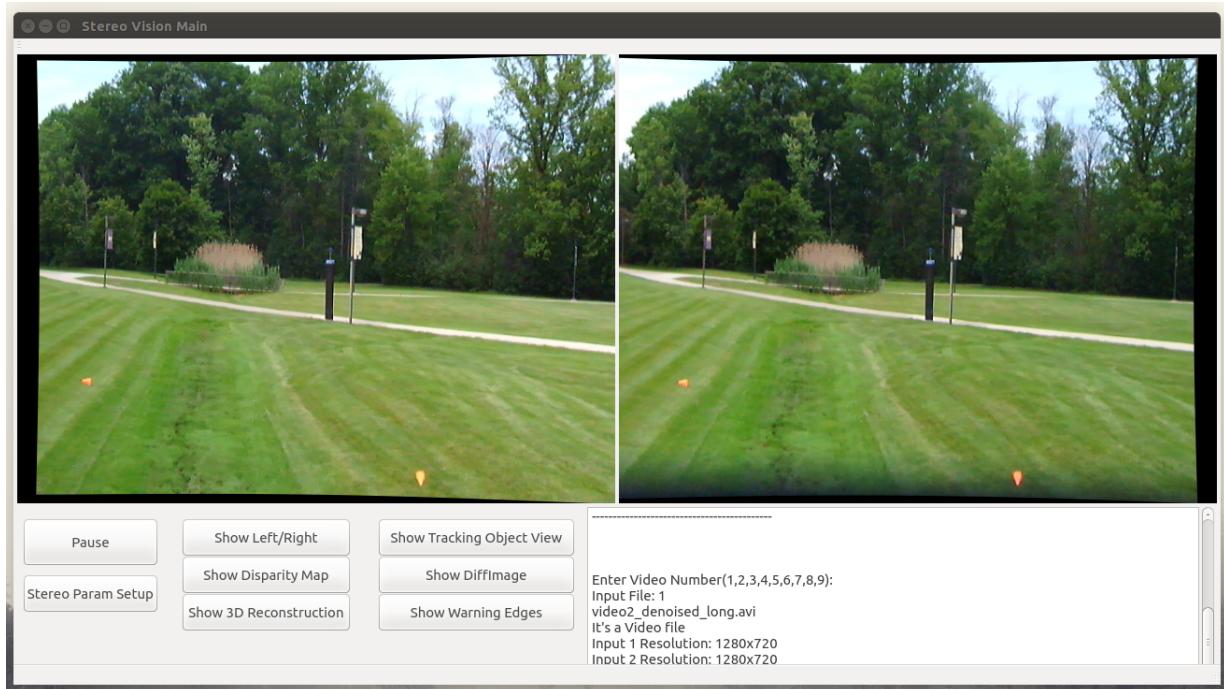


Figura 4.1: Interface Gráfica - Visualização simultânea dos quadros das câmeras esquerda e direita

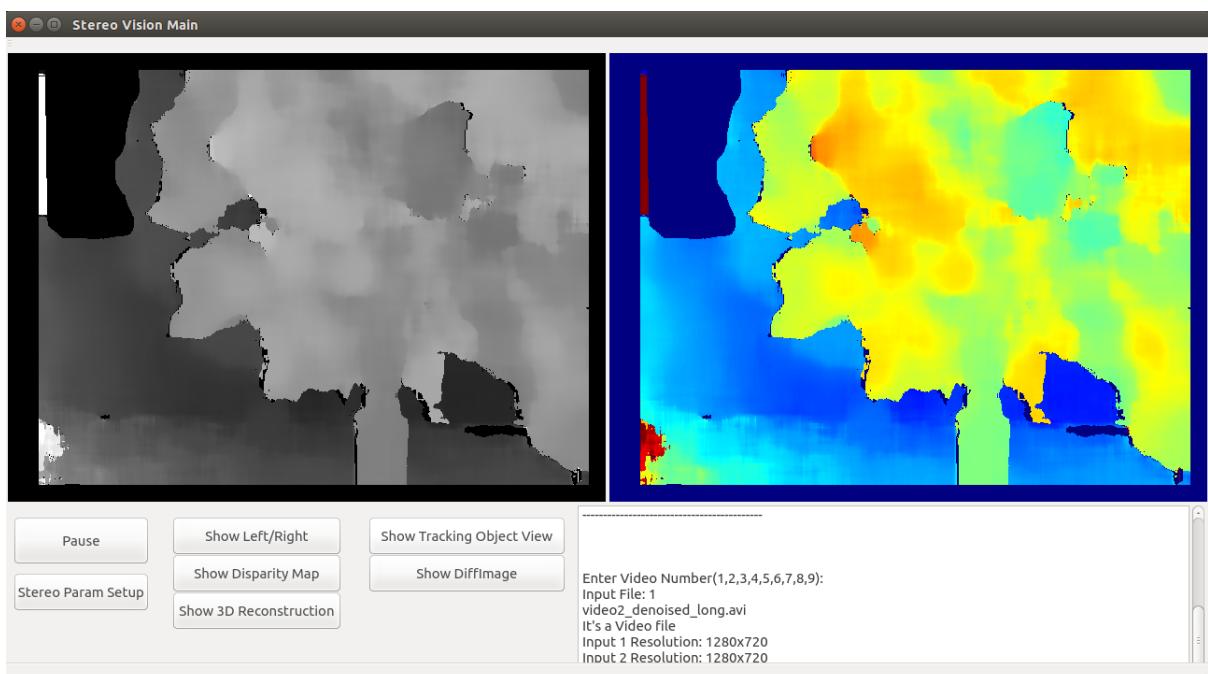


Figura 4.2: Interface Gráfica - Visualização dos Mapa de disparidades em Escala de Cinza e RGB

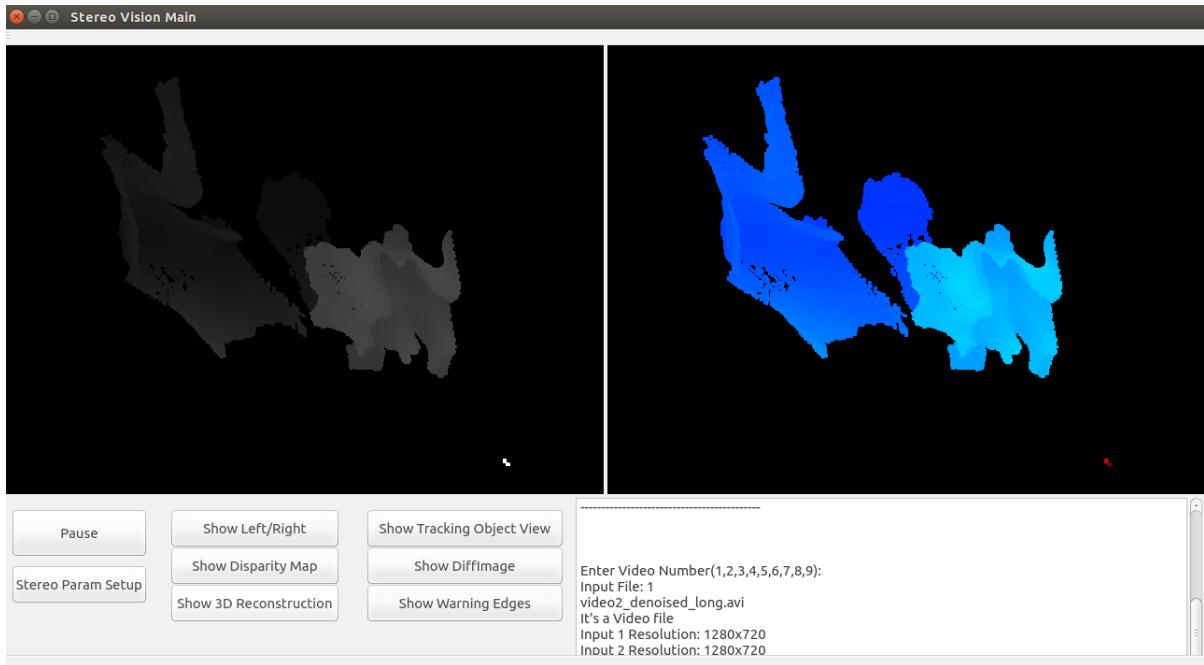


Figura 4.3: Interface Gráfica - Visualização dos Mapa de disparidades em Escala de Cinza e RGB

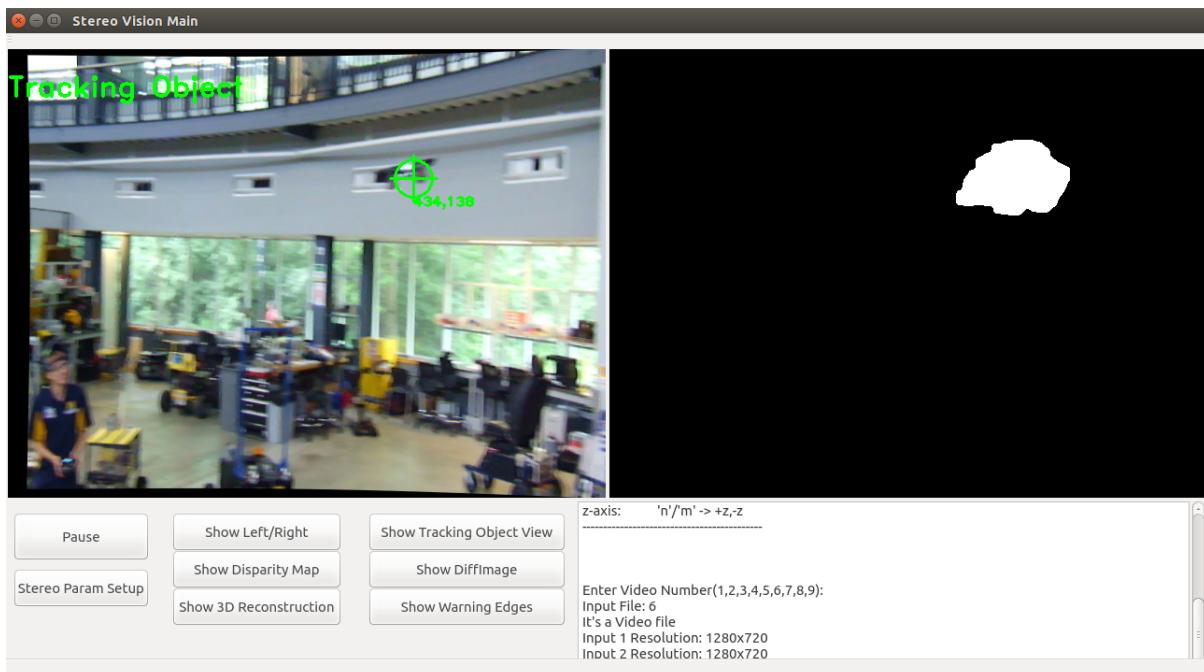


Figura 4.4: Interface Gráfica - Visualização da Imagem da Câmera Esquerda com o indicador de objeto rastreado e da Imagem binária resultante da limiarização por distância

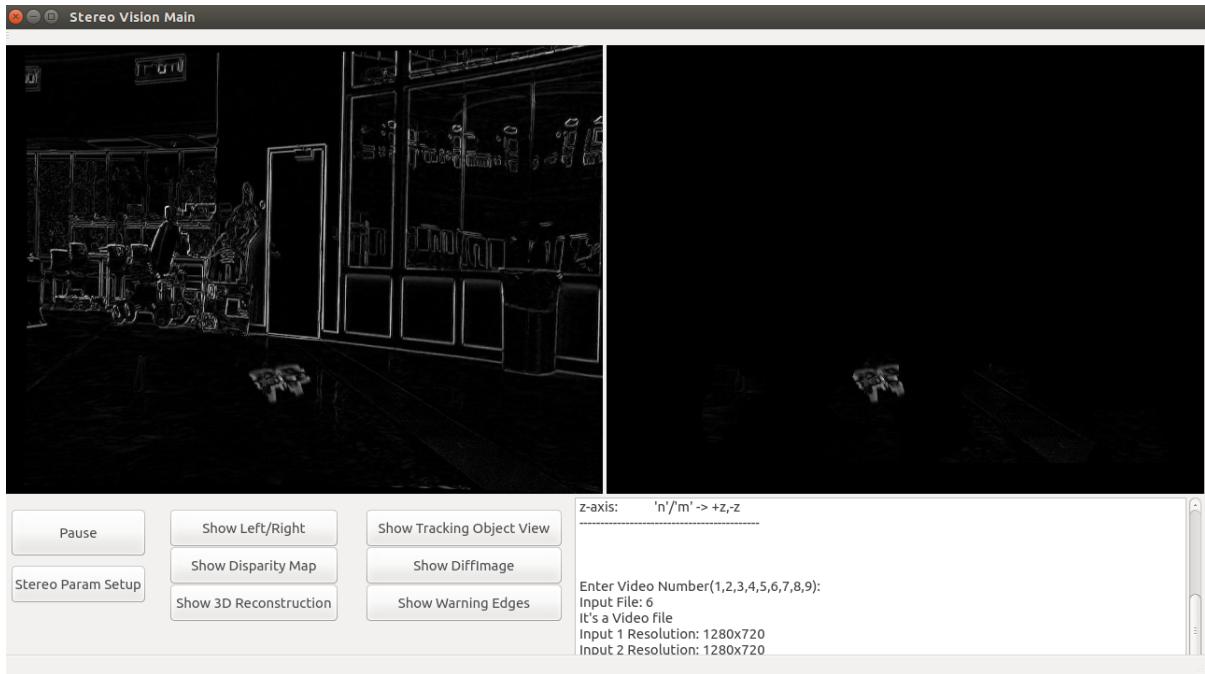


Figura 4.5: Interface Gráfica - Visualização da imagem resultante do processo de detecção de movimentos e imagem resultante do processo de detecção de movimentos limiarizada por distância

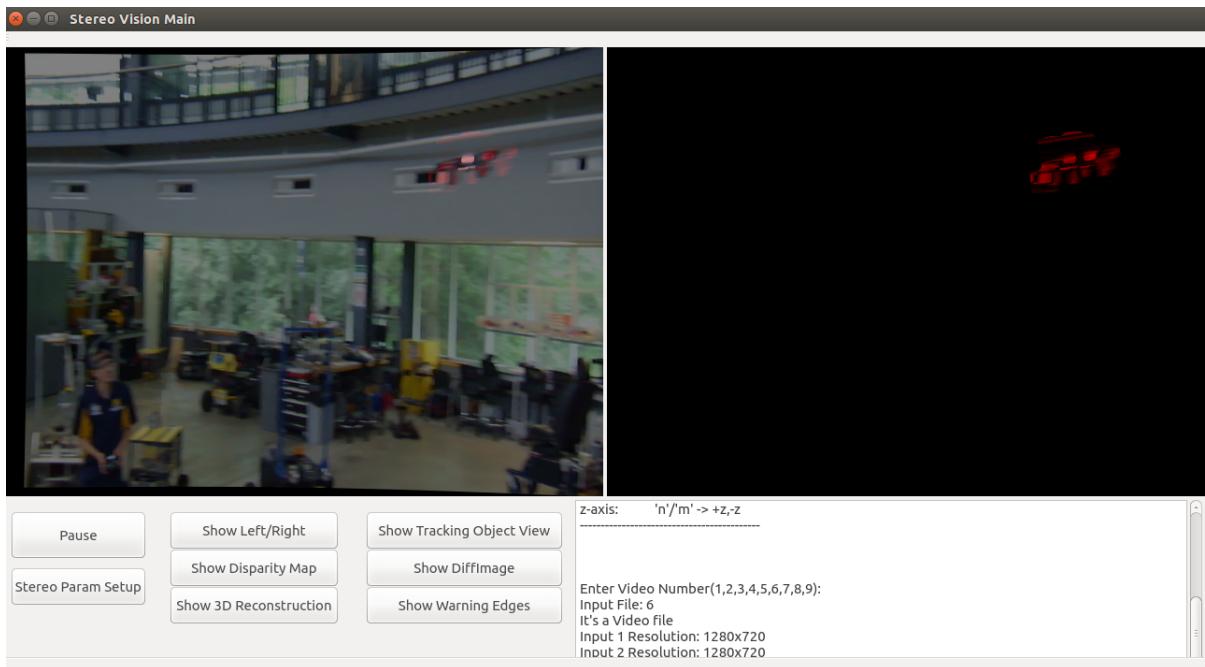


Figura 4.6: Interface Gráfica - Visualização da Imagem resultante da adição da imagem à direita com a Imagem da Câmera Esquerda e Imagem resultante do processo de realce das bordas dos objetos em movimento próximos ao veículo

4.2 Cenários - Resultados Visuais

Nesta seção, estão apresentados os resultados obtidos de cada método estéreo para os cenários propostos na seção 3.2.1. Para facilitar a comparação dos mapas de disparidades foi apresentado uma colagem para cada cenário, contendo o par de imagens capturadas da cena, o mapa de disparidade em nível de cinza e em RGB para os métodos BM, SGBM e BMGPU, respectivamente. As regiões que não contêm informações (regiões com grandes porções de cor escura, no caso da imagem RGB, de cor azul-marinho) estão muito próximas às câmeras ou áreas com superfícies refletivas. Nestas superfícies, os métodos estéreo apresentam baixa relação de unicidade, isto é, a cada iteração, mesmo em uma cena parada, o método apresenta dificuldade para definir os corretos valores de correspondências. Deste modo, é preferível identificá-las e desconsiderá-las do mapa de disparidades.

Como apresentado pela figura 4.7, a árvore é tida como principal obstáculo nesta cena e todos os métodos conseguiram identificá-la e realçá-la corretamente. O método SGBM apresenta cores mais frias, pois a configuração que permitia a correta segmentação do objeto apresenta um maior número de níveis de cinza para o mapa de disparidades. No caso do Método BMGPU, temos uma elevada porcentagem de região desconhecida, porém este consegue identificar essas regiões ao passo que o veículo se aproxima dessas regiões.

Na figura 4.8, é possível observar que o Método SGBM, diferentemente dos métodos BM e BMGPU, conseguiu identificar as bancadas, as regiões muito próximas ao quadricóptero e o cone de segurança, e não apenas as bancadas como ocorreu nos outros métodos. Entretanto, vale lembrar que este método é pesado computacionalmente e a performance do sistema também deve ser tomado em conta, não somente a qualidade do mapa gerado.

Os resultados obtidos para o terceiro cenário proposto está ilustrado pela figura 4.9. Nela foi possível, observar que todos os métodos foram capazes de identificar o piloto e o quadricóptero. Este resultado é bastante importante, visto que se provou que é possível utilizar visão estéreo para a identificação não somente de obstáculos estáticos, como também obstáculos dinâmicos, sendo estes terrestres ou aéreos. O aperfeiçoamento dos equipamentos para detecção de objetos por intermédio de visão estéreo é um dos passos que talvez possibilitem *drones* a utilizar o tráfego aéreo.

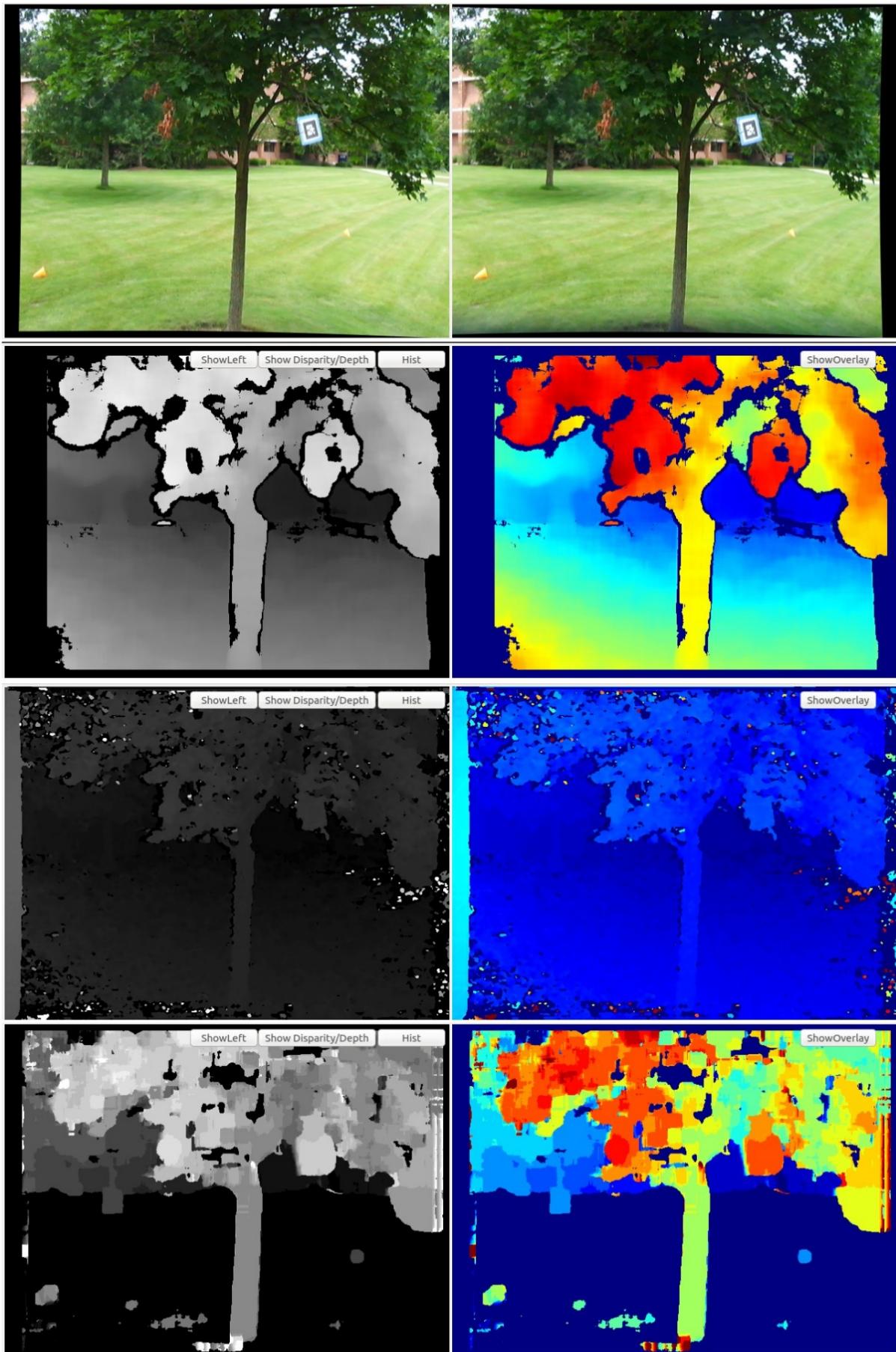


Figura 4.7: Cenário 1 - Comparativo do Mapa de Disparidades gerado pelos Métodos BM, SGBM e BMGPU, respectivamente. Árvore como principal obstáculo.

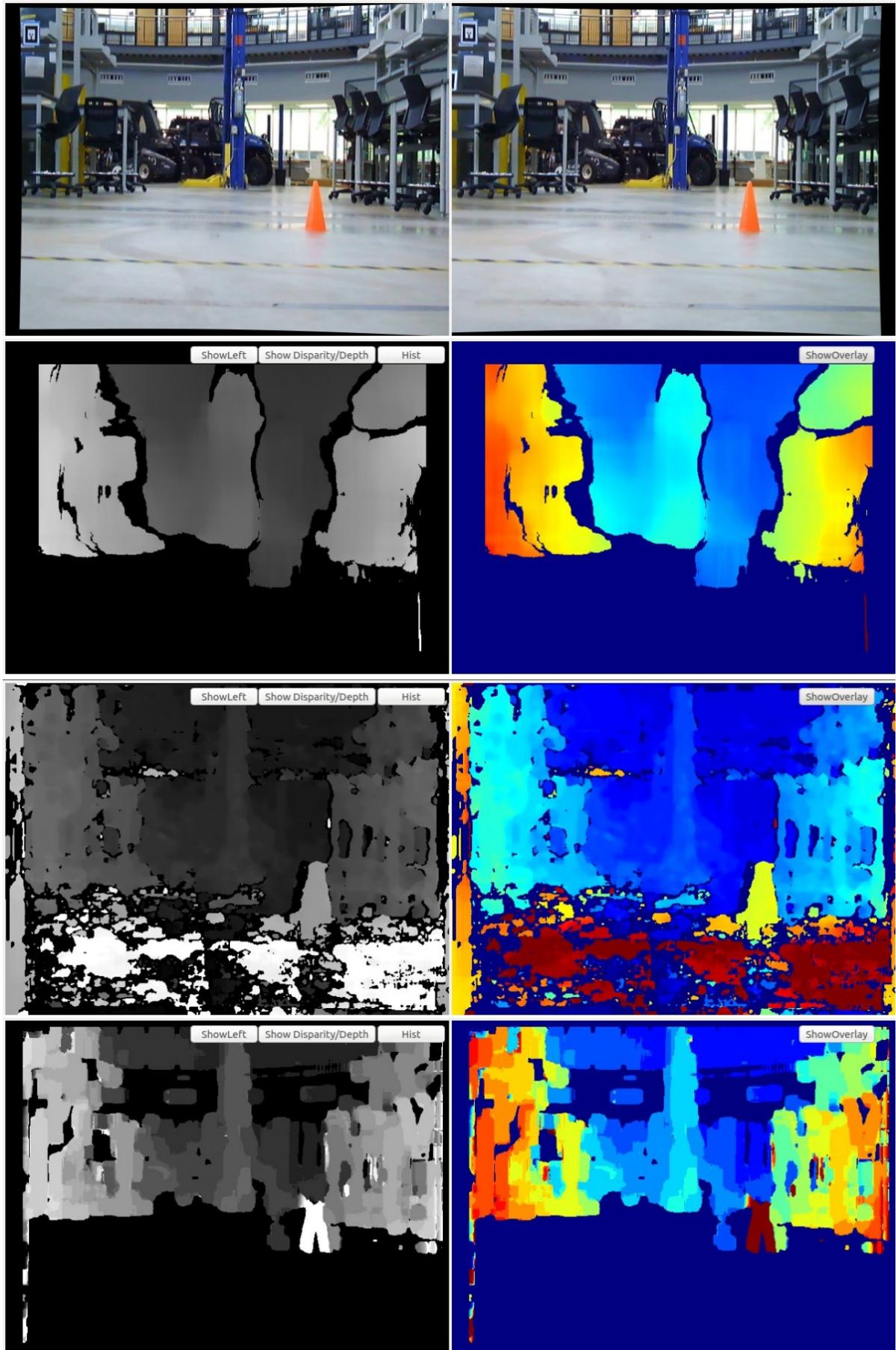


Figura 4.8: Cenário 2 - Comparativo do Mapa de Disparidades gerados pelos Métodos BM, SGBM e BMGPU, respectivamente. Bancadas, Cone de segurança como principais obstáculos.

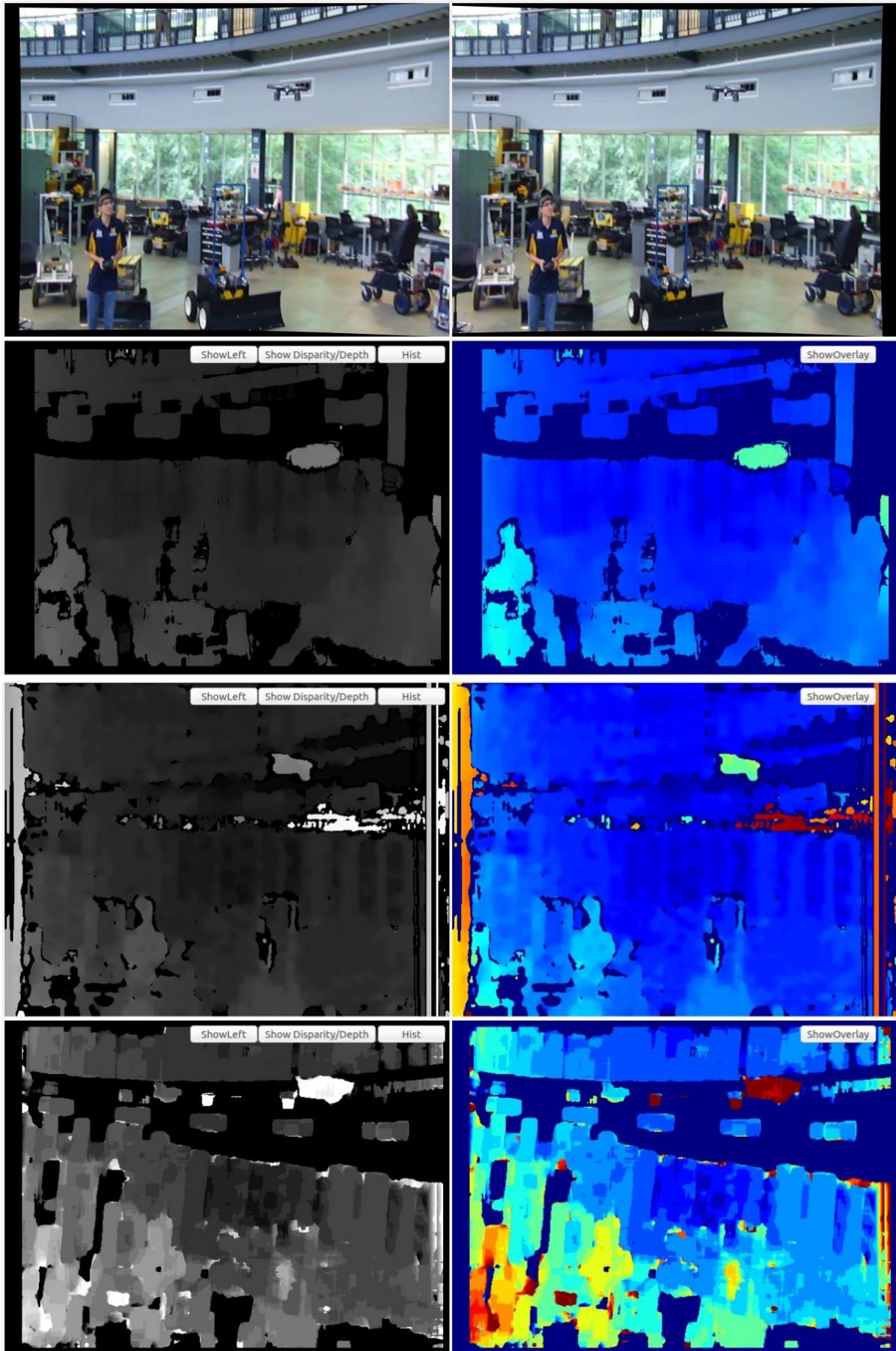


Figura 4.9: Cenário 3 - Comparativo do Mapa de Disparidades gerados pelos Métodos BM, SGBM e BMGPU, respectivamente. Piloto e Veículo aéreo (Quadricóptero) como principais obstáculos.

Tabela 4.1: Configuração utilizada para Avaliação de Performance

	Configuração
Resolução	640x480
SADWindowSize	15
NumberOfDisparities	16

4.3 Comparação de Desempenho: Desktop x BBB x Jetson TK1

Nesta seção, estão apresentados os resultados práticos das diferentes implementações dos métodos de visão estéreo nas plataformas abordadas. Os Métodos utilizados foram BM, SGBM e BMGPU para a comparação das plataformas. O método BMGPU mostrou-se o de melhor performance, visto que é o que requisita menor processamento dentre os outros métodos mais conhecidos (SGBM, BP, CSBP, AD Census, ...) além de ser acelerado em hardware. A tabela 4.2 abaixo, apresenta o resultado de desempenhos obtidos nas plataformas, apresentando comparativamente seu desempenho quando o processado utilizando CPU ou GPU/NEON (Caso da BBB). Com relação à configuração utilizada para a realização dos testes, utilizou-se uma resolução de 640x480, um tamanho médio da janela usada para combinar blocos de pixels (*SADWindowSize*) de 15 e um tamanho da gama de disparidades (*NumberOfDisparities*) igual à 16, como pode ser observado na tabela 4.1. O critério de escolha desses valores foi dado pelo balanço entre performance e qualidade do mapa de disparidades gerado, isto é, um mapa que fosse confiável e permitisse as corretas correspondências do mapa de disparidades.

O gráfico da figura 4.10 apresenta o comparativo entre o desempenho de cada plataforma para os métodos estéreo BM, SGBM e BMGPU. O resultado da BBB para o método BM é muito inferior aos resultados obtidos nas demais plataformas. O teste para o método SGBM foi abortado, visto que naturalmente este método é um algoritmo mais complexo que o BM. Decorrente a isso, espera-se que sua performance seja inferior à um FPS. Segundo o manual da BBB, o processador ARM Cortex A8 apresenta suporte ao NEONTM, o qual é um acelerador de ponto flutuante que utiliza instruções SIMD (*Single instruction, Multiple Data*) para a aceleração de algoritmos de multimídia e processamento de sinais [34]. Os testes dos métodos estéreo na BBB utilizando essa tecnologia também não foram executados, pois sua implementação tomaria um tempo considerável e provavelmente não apresentaria um resultado expressivo.

Tabela 4.2: Desempenho atingido por cada plataforma dos Métodos Utilizados

Quadros por Segundo (FPS)	BM	SGBM	BM_GPU
Desktop	28	12	30
BBB	1	-	-
Jetson TK1	6	2	7~8

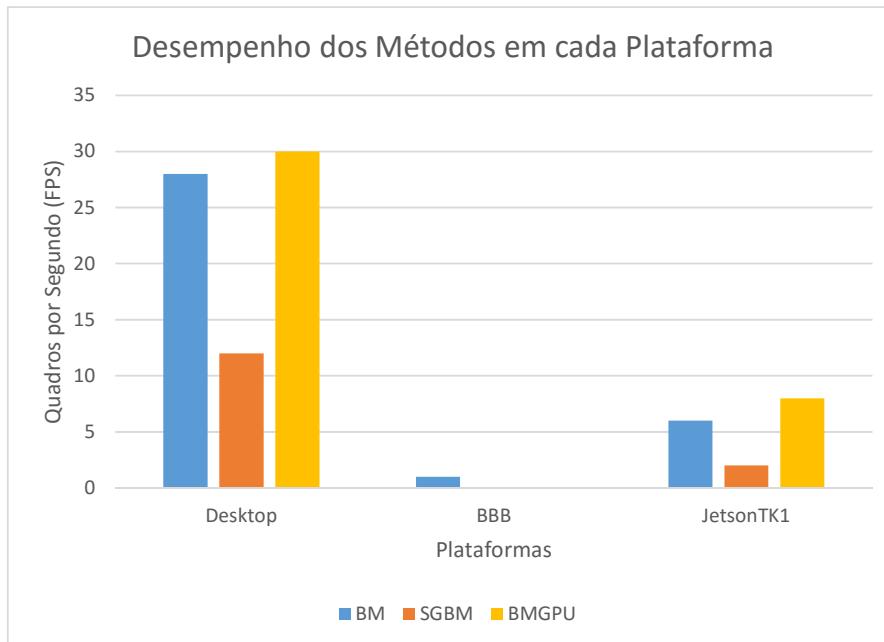


Figura 4.10: Gráfico - Comparação de Desempenho dos Métodos Estéreo para cada plataforma.

Este trabalho não tem como escopo realizar a avaliação dos resultados apresentados acima com relação ao atendimento dos requisitos da aplicação. O trabalho realiza a busca pelas limitações dos sistemas para a execução dos métodos estéreo. No que diz respeito às condições do teste, em nenhuma das plataformas realizou-se a priorização dos processos no sistema. Vale realçar que nenhuma distribuição do tipo RTOS (Sistema Operacional em Tempo Real) foi utilizada.

4.3.1 Desktop

CPU

Primeiramente, as rotinas para execução em CPU foram implementadas no *Desktop* e só então foram adaptadas para as plataformas embarcadas. Os valores de desempenho obtidos foram usados como referência para a comparação com os resultados acelerados por GPU e com as plataformas embarcadas. Como esperado, o SGBM mostrou-se mais lento que o BM, isso deve-se ao fato do algoritmo ser mais complexo por conta da análise global realizada (Mais detalhes em 2.1.5).

GPU

Como previsto, o desempenho do método BM acelerado em GPU é superior ao executado em CPU. O ganho em desempenho é pequeno, isto deve-se ao fato do tempo necessário para transferir as imagens entre a memória utilizada pela CPU e a memória da GPU é comparativamente grande ao ganho de processamento pela execução em GPU. Consequentemente, este resultado motivou a implementação do mesmo na plataforma *Jetson TK1*, visto que sua implementação é bastante similar a em Desktop por conta de possuírem a mesma arquitetura de GPU (Kepler).

4.3.2 BeagleBone Black (BBB)

CPU

O resultado obtido para o método BM desmotivou qualquer outra implementação nesta plataforma. Dificilmente, algum outro recurso de aceleração apresentaria um resultado que realizasse o processamento de imagens necessários com resposta em tempo real.

NEON

Como já fora dito, o resultado obtido pela performance em CPU, desestimulou o estudo e implementação desta tecnologia.

4.3.3 Jetson TK1

CPU

A implementação em CPU nesta plataforma foi realizada para efeito de comparação entre o desempenho obtido em CPU e com a implementação utilizando CUDA. Como esperado, o método SGBM também se mostrou mais lento que o BM, apresentando uma taxa de processamento relativamente baixa, aproximadamente dois FPS. Com relação ao BM, a taxa atingida foi de aproximadamente seis quadros por segundo, taxa que começa a possibilitar aplicações em tempo real para alguns tipos sistemas.

GPU

Como esperado, a implementação da tecnologia CUDA na *Jetson TK1* acelerou a execução do método BM em aproximadamente 30%. O ganho obtido é expressivo, porém, assim como dito acima, aplicações que garantam resposta em tempo real necessitam apresentar taxas de processamento muito superiores à essa. Idealmente, espera-se que a taxa de processamento seja pelo menos igual à taxa de captura da câmera.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusão

O trabalho cumpriu todos os objetivos apresentados na seção 1.1.

No que diz respeito à visão estéreo, foram realizados o estudo e a aplicação das técnicas de visão computacional para o problema proposto, mais precisamente estudou-se os processos de calibração, retificação, filtragem e os métodos estéreo para a obtenção de um mapa de disparidades denso.

No que concerne o monitoramento de um veículo autônomo, foi desenvolvida uma plataforma de apoio que facilita o processo de calibração dos métodos estéreo para as plataformas embarcadas, visto que são destinadas a uma aplicação que não necessita de interface gráfica. Além disso, ela possibilita o monitoramento via estação-base, caso deseja-se que o processamento das imagens não seja realizado *on-board*. Os métodos foram implementados e testados em cenários capturados pela câmera fixada no quadricóptero. Entretanto, a extensão desse trabalho não se resume a veículos aéreos, as plataformas utilizadas podem muito bem serem empregadas em veículos terrestres e até mesmo em veículos aquáticos, evidentemente com a devida vedação dos equipamentos. Outro importante objetivo foi a comparação de desempenho dos métodos nas plataformas disponíveis, BBB e Jetson TK1. No caso da BBB, mesmo a rotina desenvolvida mostrando-se totalmente funcional, isto é, os algoritmos conseguem detectar e segmentar relativamente bem os obstáculos dos cenários propostos, ocorreu que a taxa de processamento tornou-se uma preocupação. O desempenho foi extremamente baixo, aproximadamente um quadro por segundo, o que claramente inviabiliza sua utilização em uma aplicação real. A alteração de plataforma mostrava-se ser uma solução para o aumento da taxa de atualização, visto que a Jetson TK1 é uma plataforma mais recente e mais

poderosa. O desempenho desenvolvido pela Jetson TK1 foi de aproximadamente seis FPS (Método BM), performance ainda baixa para um sistema que deve apresentar resposta em tempo real (O critério de avaliação tomado foi de pelo menos 10 FPS).

Diante deste impasse, duas providências foram cruciais para a continuidade do trabalho. A primeira foi a otimização das rotinas utilizadas, sendo essa uma tentativa para a melhora do desempenho geral dos métodos. A segunda foi a execução de uma ampla revisão bibliográfica, incluindo principalmente trabalhos que apresentassem comparativos de desempenho entre plataformas e que tivessem realizados algum tipo de aceleração por *hardware*, sejam eles envolvendo paralelização de processos, implementação em FPGA ou utilizando a plataforma de computação paralela CUDA. Ao fim, a última alternativa foi tomada visto que a plataforma Jetson TK1 apresenta suporte a tecnologia CUDA.

Com relação à utilização de uma plataforma embarcada em quadricóptero, a plataforma Jetson TK1 juntamente com o método BMGPU foi a melhor alternativa para a implementação em um sistema real, mesmo apresentando uma taxa de performance reduzida, foi a melhor solução existente durante o período de desenvolvimento do trabalho. Vale ressaltar que as rotinas implementadas utilizaram os métodos estéreo presentes no OpenCV, isto é, não se desenvolveu rotinas em programação CUDA, diretivas para a manipulação de dados em GPU. Deste modo, existe a expectativa de que essas bibliotecas sejam otimizadas em novas versões do OpenCV ou que a NVIDIA atualize a versão do CUDA *Toolkit*, reduzindo assim o tempo de execução.

Quanto à qualidade do mapa de disparidades, a câmera utilizada, FinePix 3D W3, mesmo sendo uma câmera comercial que não necessariamente precisa se preocupar com o perfeito alinhamento do *Stereo Rig* e qualidade das lentes apresentou resultados satisfatórios, isto é, mapas com as corretas correspondências dos pontos homólogos e com baixa porcentagem de região de desconhecido. Com o devido processo de calibração foi capaz de capturar os vídeos utilizados para o desenvolvimento desse trabalho, além de cumprir todos os requisitos do projeto (Leve, *Baseline* rígida, Taxa de quadros elevada).

Em suma, o projeto atendeu seus principais objetivos e proporcionou o estudo de diferentes conceitos essencialmente relacionados com visão computacional, sistemas embarcados e aceleração em GPU. Com relação aos mapas de disparidades gerados pelos métodos estéreo, os resultados obtidos, tanto em CPU quanto em GPU, são idênticos, porém ainda se apresentam problemas referentes à performance do sistema. Por hora, eles podem ser resolvidos com a substituição dos equipamentos por melhores câmeras ou plataformas embarcadas mais

poderosas. Ao fim do trabalho, os algoritmos desenvolvidos podem ser totalmente adaptados para a sua incorporação a sistemas robóticos mais conhecidos como *Player/Stage* [35], *Gazebo* [36], *ROS* [37], dentre outros. Esse tipo de integração pode ser realizada modificando-se a plataforma desenvolvida para que esta publique corretamente as imagens nos tópicos do ROS, por exemplo. Os resultados obtidos permitiram que uma nova fase de pesquisa se iniciasse e também possibilitasse o estudo de novos conceitos como SLAM, algoritmos para desvio de obstáculos, planejamento de rotas.

5.2 Trabalhos Futuros

Como apresentado acima, concluiu-se que o trabalho desenvolvido apresentou bons resultados em diversos aspectos, porém esses resultados podem ser melhorados pela simples troca ou atualização das câmeras e plataformas utilizadas. Além disso, ao fim do desenvolvimento do projeto, indagou-se quais modificações deveriam ser realizadas para a real implementação do sistema autônomo. Consequentemente, a maior parte das sugestões de atividades futuras descritas nessa seção estão diretamente relacionadas com as possíveis melhorias e implementação de novos conceitos.

No que se refere à atualização de plataformas, a NVIDIA lançou recentemente uma nova plataforma - Jetson TX1 que apresenta um aumento expressivo de núcleos CUDA, aproximadamente 30% em relação à versão anterior [38]. A implementação do trabalho desenvolvido nesta nova plataforma é bastante válida e interessante, visto que existe a possibilidade da melhora na performance do sistema. Além disso, o estudo de novas alternativas de aceleração em *hardware* também é válido. Uma delas é a utilização de FPGAs, as quais são dispositivos que possibilitam o desenvolvimento de arquiteturas totalmente especializadas e também podem ser dedicadas a esse tipo de aplicação, como visto em [8]. Outra alternativa ainda mais recente, seria a utilização da tecnologia presente nos processadores gráficos MaliTM da ARM®. Eles também permitem a aceleração em GPU, com um elevado grau de especialização, por meio da utilização de programação paralela OpenCL [39]. Ambas arquiteturas apoiam-se na utilização de *pipelines*, as quais proporcionam que mais de um processo sejam executados comitadamente, o que reduz drasticamente o tempo para a execução de cada quadro. Embora a Jetson TK1 suporte à tecnologia CUDA, ela ainda é uma plataforma de desenvolvimento e de propósito geral, por conta disso, acredita-se que o desempenho das alternativas apresentadas sejam melhores.

No que diz respeito à atualização das câmeras, existem atualmente no mercado algumas câmeras que também cumprem todos os requisitos do projeto e poderia muito bem serem incorporadas. As câmeras estéreo mais indicadas para esse tipo de aplicação são a *Bumblebee®2* da *Point Grey Research* [40], *ZED™* da *StereoLabs* [41] e as câmeras *DUO 3D™* da *Code Laboratories* [42]. Cada qual possui sua especialidade: Alta taxa de quadros, Alta Resolução, Tamanho Pequeno, respectivamente. Atualmente, a StereoLab e a NVI-DIA têm colaborado para o desenvolvimento de sistemas que utilizem a câmera *ZED™* e as plataformas da família Jetson, o que realmente incentiva a comunidade de desenvolvedores optarem por essa solução [43]. Isso é um indicativo que o caminho para o desenvolvimento de plataformas embarcadas utilizando visão estéreo será mediante à combinação de equipamentos dessas empresas.

Referências Bibliográficas

- [1] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*, volume 1. 2008.
- [2] Caio César Teodoro Mendes. *Navegação de robôs móveis utilizando visão estéreo*. PhD thesis, University of São Paulo, 2012.
- [3] D Scharstein and R Szeliski. High-accuracy stereo depth maps using structured light. *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, 1(June):I–195 – I–202, 2003.
- [4] Andrew J Barry, Helen Oleynikova, Dominik Honegger, Marc Pollefeys, and Russ Tedrake. FPGA vs . Pushbroom Stereo Vision for MAVs. *Vision-based Control and Navigation of Small Lightweight UAVs, IROS Workshop*, pages 1–6, 2015.
- [5] Jean-Yves Bouguet. Complete Camera Calibration Toolbox for MATLAB, 1999.
- [6] MATLAB MathWorks Inc. Stereo Camera Calibrator. <http://www.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html>, Acesso em: 20 de Abril de 2016.
- [7] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM). *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.
- [8] Andrew J. Barry, Helen Oleynikova, Dominik Honegger, Marc Pollefeys, and Russ Tedrake. Fast Onboard Stereo Vision for UAVs. page 7, 2015.
- [9] Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix. Vision-Based SLAM: Stereo and Monocular Approaches. *International Journal of Computer Vision*, 74(3):343–364, 2007.

- [10] BeagleBoard.org. BeagleBone Black. <https://beagleboard.org/>, Acesso em: 20 de Abril de 2016.
- [11] NVIDIA. Jetson TK1 Developer Kit. <https://developer.nvidia.com/embedded/develop/hardware>, Acesso em: 20 de Abril de 2016.
- [12] Sunil Shah. Real-time Image Processing on Low Cost Embedded Computers. pages 1–29, 2014.
- [13] Victor M. G. Paula. O vanto jato brasileiro. *Centro de Pesquisas Estratégicas “Paulino Soares de Sousa”*, pages 1–3, Acesso em: 20 de Abril de 2016.
- [14] José Luiz Boanova Filho. Aeronaves Não Tripuláveis no Brasil e sua Regulação. *Revista Brasileira de Direito Aeronáutico e Espacial*, 2014.
- [15] Departamento de Controle do Espaço Aéreo da Aeronáutica. DECEA publica nova regulamentação para voos de RPAS (drones). <http://www.decea.gov.br/?p=8465>, Acesso em: 06 de Dezembro de 2015.
- [16] FLYAAVC. Autonomous Aerial Vehicle Competition. <http://www.flyaavc.org/>, Acesso em: 09 de Novembro de 2015.
- [17] Robert Laganière. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, 2011.
- [18] OpenCV Development Team. Camera Calibration and 3D Reconstruction. http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, Acesso em: 30 de Março de 2016.
- [19] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008.
- [20] Junhwan Kim, Kolmogorov, and Zabih. Visual correspondence using energy minimization and mutual information. In *Proceedings Ninth IEEE International Conference on Computer Vision*, number Iccv, pages 1033–1040 vol.2, 2003.
- [21] D. F. Shinzato, P. Y. ; Osório, F. S. ; Wolf. Visual Road Recognition Using Artificial Neural Networks and Stereo Vision. *IEEE/RSJ International Conference on Intelligent Robots and Systems - IROS - Workshop*.

- [22] Sharad Nagappa, Narcís Palomeras, Chee Sing Lee, Nuno Gracias, Daniel E. Clark, and Joaquim Salvi. Single cluster PHD SLAM: Application to autonomous underwater vehicles using stereo vision. *2013 MTS/IEEE OCEANS-Bergen*, (Ref 288273):1–9, 2013.
- [23] Itseez. OpenCV Information. <http://itseez.com/opencv/>, Acesso em: 20 de Abril de 2016.
- [24] Itseez. OpenCV Platforms. <http://opencv.org/platforms.html>, Acesso em: 20 de Abril de 2016.
- [25] Shuai Wang, Heng Zhang, Han Qing Tan, and Lin Ying Jiang. Implementation of step motor control under embedded linux based on S3C2440. In *Energy Procedia*, volume 16, pages 1541–1546, 2011.
- [26] Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, and Philippe Gerum. *Building Embedded Linux Systems*. O'Reilly Media, 2008.
- [27] DT Monish and Sandy Wiraatmadja. High-Speed Prediction of Air Traffic for Real-Time Decision Support. *AIAA Guidance, Navigation, and Control Conference*, (August):1–15, 2011.
- [28] John E. Stone, Michael J. Hallock, James C. Phillips, Joseph R. Peterson, Kirby L. Vandivort, Zaida Luthey-Schulten, and Klaus Schulten. Evaluation of Emerging Energy-Efficient Heterogeneous Computing Platforms for Biomolecular and Cellular Simulation Workloads. *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2016 IEEE International*, 2015.
- [29] NVIDIA. CudaTM - plataforma paralela de computação. http://www.nvidia.com.br/object/cuda_home_new_br.html, Acesso em: 03 de Março de 2015.
- [30] OpenCV Development Team. Camera Calibration with OpenCV. http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html, Acesso em: 27 de Abril de 2016.
- [31] Facebook. CUDA Installation. <https://github.com/facebook/fbcunn/blob/master/INSTALL.md>, Acesso em: 24 de Maio de 2016.

- [32] Adam Harasimowicz. Install OpenCV with CUDA. <http://blog.aicry.com/ubuntu-14-04-install-opencv-with-cuda/>, Acesso em: 24 de Maio de 2016.
- [33] Embedded Linux Wiki. Installing OpenCV (including the GPU module) on Jetson TK1. http://elinux.org/Jetson/Installing_OpenCV, Acesso em: 16 de Março de 2016.
- [34] ARM Ltd. NEONTM. <http://www.arm.com/products/processors/technologies/neon.php>, Acesso em: 23 de Maio de 2016.
- [35] Brian Gerkey, Richard Vaughan, Andrew Howard, and Nathan Koenig. Player/Stage Project. <http://playerstage.sourceforge.net/>, Acesso em: 25 de Maio de 2016.
- [36] Open Source Robotics Foundation. Gazebo. <http://www.gazebosim.org/>, Acesso em: 25 de Maio de 2016.
- [37] Open Source Robotics Foundation. ROS - Robot Operating System. <http://www.ros.org/>, Acesso em: 25 de Maio de 2016.
- [38] NVIDIA. Jetson TX1 Developer Kit. <http://www.nvidia.com/object/jetson-tx1-dev-kit.html>, Acesso em: 21 de Maio de 2016.
- [39] Gian Marco Iodice; Media Processing Group; ARM. Real-time Dense Passive Stereo Vision: A Case Study in Optimizing Computer Vision Applications Using OpenCLTM on ARM®, 2015.
- [40] Point Grey Research Inc. Bumblebee®2. <https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-systems>, Acesso em: 24 de Maio de 2016.
- [41] StereoLabs. ZED - 3D Camera for AR/VR and Autonomous Navigation. <https://www.stereolabs.com/zed/specs/>, Acesso em: 24 de Maio de 2016.
- [42] Code Laboratories Inc. DUO 3DTM. <https://duo3d.com/>, Acesso em: 25 de Maio de 2016.
- [43] NVIDIA. NVIDIA Jetson Partner Story: Stereolabs. <https://developer.nvidia.com/embedded/learn/success-stories/stereolabs>, Acesso em: 25 de Maio de 2016.

- [44] Etsuko; UEDA, Masanao; Koeda, Tsuyoshi; Suenaga, and Kentaro; Takemura. 3D Reconstruction and Point Cloud Rendering. <http://opencv.jp/opencv2-x-samples/point-cloud-rendering>, Acesso em: 05 de Dezembro de 2015.
- [45] Kyle Hounslow. Real-Time Object Tracking Using OpenCV. <https://www.youtube.com/watch?v=bSeFrPrqZ2A>, Acesso em: 05 de Dezembro de 2015.

Apêndice A

Apêndice 1

Todos os trechos de código desenvolvidos e os arquivos de calibração utilizados para a estação-base da interface gráfica e para as outras plataformas podem ser encontradas nos seguintes repositórios:

- Desktop: <https://github.com/nicolasrosa/StereoVision>
- BBB: <https://github.com/nicolasrosa/StereoVisionBBB>
- Jetson TK1: <https://github.com/nicolasrosa/StereoVisionJetsonTK1>

Anexo I

Anexo 1

Os seguintes trechos de código foram incorporados ao projeto da interface gráfica para a estação-base. Duas funcionalidades foram adicionadas ao código. A primeira funcionalidade corresponde a parte de reconstrução tridimensional e renderização da nuvem de pontos [44]. A segunda funcionalidade corresponde a parte de rastreamento do objeto desenvolvida por Kyle Hounslow [45].

3DReconstruction.h

```
#ifndef RECONSTRUCTION_3D_H
#define RECONSTRUCTION_3D_H

/* Libraries */
#include "opencv2/opencv.hpp"

using namespace cv;
using namespace std;

/* 3D Reconstruction Classes */
template <class T>
static void projectImagefromXYZ_(Mat& image, Mat& destimage, Mat& disp,
                                Mat& destdisp, Mat& xyz, Mat& R, Mat& t,
                                Mat& K, Mat& dist, Mat& mask, bool isSub);

template <class T>
static void fillOcclusionInv_(Mat& src, T invalidvalue);

template <class T>
static void fillOcclusion_(Mat& src, T invalidvalue);

// 3D Reconstruction
class Reconstruction3D{
public:
    Reconstruction3D(); //Constructor
    void setViewPoint(double x,double y,double z);
    void setLookAtPoint(double x,double y,double z);
    void PointCloudInit(double baseline,bool isSub);

/* 3D Reconstruction Functions */
    void eular2rot(double yaw,double pitch, double roll,Mat& dest);
    void lookat(Point3d from, Point3d to, Mat& destR);
    void projectImagefromXYZ(Mat &image, Mat &destimage, Mat &disp,
```

```

        Mat &destdisp , Mat &xyz , Mat &R, Mat &t ,
        Mat &K, Mat &dist , bool isSub);

void fillOcclusion(Mat& src , int invalidvalue , bool isInv);

Mat disp3Dviewer;
Mat disp3D;
Mat disp3D_8U;
Mat disp3D_BGR;

Point3d viewpoint;
Point3d lookatpoint;

Mat dist;
Mat Rotation;
Mat t;

Mat depth;

double step;
bool isSub;
};

#endif // RECONSTRUCTION_3D_H

```

trackObject.h

```

/*
 * trackObject.h
 *
 * Author: Kyle Hounslow
 * Link: https://www.youtube.com/watch?v=bSeFrPrqZ2A
 * Published in: March 11, 2013
 */

#ifndef SRC_TRACKOBJECT_H_
#define SRC_TRACKOBJECT_H_

/* Libraries */
#include "opencv2/opencv.hpp"

using namespace cv;
using namespace std;

//default capture width and height
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
//max number of objects to be detected in frame
const int MAX_NUM_OBJECTS=50;
//minimum and maximum object area
const int MIN_OBJECT_AREA = 20*20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;

string intToString(int number);
void drawObject(int x, int y, Mat &frame);
void trackFilteredObject(int &x, int &y, Mat threshold , Mat &cameraFeed);

#endif /* SRC_TRACKOBJECT_H_ */

```

3DReconstruction.cpp

```

/*
 * 3DReconstruction.cpp
 *
 *
```

```

*   Created on: Oct 25, 2015
*       Author: nicolasrosa
*/
#include "3DReconstruction.h"

/* Constructor */
Reconstruction3D :: Reconstruction3D(){}

void Reconstruction3D :: setViewPoint(double x, double y, double z){
    this->viewpoint.x = x;
    this->viewpoint.y = y;
    this->viewpoint.z = z;
}

void Reconstruction3D :: setLookAtPoint(double x, double y, double z){
    this->lookatpoint.x = x;
    this->lookatpoint.y = y;
    this->lookatpoint.z = z;
}

void Reconstruction3D :: PointCloudInit(double baseline, bool isSub){
    this->dist=Mat::zeros(5,1,CV_64F);
    this->Rotation=Mat::eye(3,3,CV_64F);
    this->t=Mat::zeros(3,1,CV_64F);

    this->isSub = isSub;
    this->step = baseline/10;
}

void Reconstruction3D :: eular2rot(double yaw, double pitch, double roll, Mat& dest){
    double theta = yaw/180.0*CV_PI;
    double pusai = pitch/180.0*CV_PI;
    double phi = roll/180.0*CV_PI;

    double datax[3][3] = {{1.0,0.0,0.0},{0.0,cos(theta),-sin(theta)},{0.0,sin(theta),cos(theta)}};
    double datay[3][3] = {{cos(pusai),0.0,sin(pusai)},{0.0,1.0,0.0},{-sin(pusai),0.0,cos(pusai)}};
    double dataz[3][3] = {{cos(phi),-sin(phi),0.0},{sin(phi),cos(phi),0.0},{0.0,0.0,1.0}};

    Mat Rx(3,3,CV_64F,datax);
    Mat Ry(3,3,CV_64F,datay);
    Mat Rz(3,3,CV_64F,dataz);
    Mat rr=Rz*Rx*Ry;
    rr.copyTo(dest);
}

void Reconstruction3D :: lookat(Point3d from, Point3d to, Mat& destR){
    double x=(to.x-from.x);
    double y=(to.y-from.y);
    double z=(to.z-from.z);

    double pitch = asin(x/sqrt(x*x+z*z))/CV_PI*180.0;
    double yaw = asin(-y/sqrt(y*y+z*z))/CV_PI*180.0;

    eular2rot(yaw, pitch, 0,destR);
}

void Reconstruction3D :: projectImagefromXYZ(Mat &image, Mat &destimage, Mat &disp,
                                             Mat &destdisp, Mat &xyz, Mat &R, Mat &t,
                                             Mat &K, Mat &dist, bool isSub){
    Mat mask;
    if(mask.empty()) mask=Mat::zeros(image.size(),CV_8U);
    if(disp.type()==CV_8U){
        projectImagefromXYZ_<unsigned char>(image,destimage, disp, destdisp, xyz, R, t, K, dist, mask,isSub);
    }
    else if(disp.type()==CV_16S){

```

```

    projectImagefromXYZ_<short>(image , destimage , disp , destdisp , xyz , R, t , K, dist , mask , isSub );
}

else if ( disp . type ()==CV_16U){
    projectImagefromXYZ_<unsigned short>(image , destimage , disp , destdisp , xyz , R, t , K, dist , mask , isSub );
}

else if ( disp . type ()==CV_32F){
    projectImagefromXYZ_<float>(image , destimage , disp , destdisp , xyz , R, t , K, dist , mask , isSub );
}

else if ( disp . type ()==CV_64F){
    projectImagefromXYZ_<double>(image , destimage , disp , destdisp , xyz , R, t , K, dist , mask , isSub );
}

}

template <class T>
static void fillOcclusionInv_(Mat& src , T invalidvalue){

    int bb=1;
    const int MAX_LENGTH=src . cols *0.8;
    //#pragma omp parallel for
    for (int j=bb;j<src . rows-bb;j++){
        T* s = src . ptr<T>(j);
        //const T st = s[0];
        //const T ed = s[src . cols -1];
        s[0]=0;
        s[src . cols -1]=0;
        for (int i=0;i<src . cols ;i++){
            if (s[i]==invalidvalue){
                int t=i;
                do{
                    t++;
                    if (t>src . cols -1)break ;
                }while (s[t]==invalidvalue );

                const T dd = max(s[i-1],s[t]);
                if (t-i>MAX_LENGTH){
                    for (int n=0;n<src . cols ;n++){
                        s[n]=invalidvalue ;
                    }
                }
                else {
                    for (;i<t;i++){
                        s[i]=dd;
                    }
                }
            }
        }
    }
}

template <class T>
static void projectImagefromXYZ_(Mat& image , Mat& destimage , Mat& disp ,
                                Mat& destdisp , Mat& xyz , Mat& R, Mat& t,
                                Mat& K, Mat& dist , Mat& mask , bool isSub){

    if (destimage . empty ()) destimage=Mat:: zeros (Size (image . size ()) ,image . type ());
    if (destdisp . empty ()) destdisp=Mat:: zeros (Size (image . size ()) ,disp . type ());

    vector<Point2f> pt;
    if (dist . empty ()) dist = Mat:: zeros (Size (5 ,1) ,CV_32F);
    cv::projectPoints (xyz ,R, t ,K, dist ,pt);
    destimage . setTo (0);
    destdisp . setTo (0);

    //#pragma omp parallel for
    for (int j=1;j<image . rows -1;j++){
        int count=j*image . cols ;
        uchar* img=image . ptr<uchar>(j);
    }
}

```

```

uchar* m=mask.ptr<uchar>(j);
for(int i=0;i<image.cols;i++,count++){
    int x=(int)(pt[count].x+0.5);
    int y=(int)(pt[count].y+0.5);
    if(m[i]==255) continue;
    if(pt[count].x>=1 && pt[count].x<image.cols-1 && pt[count].y>=1 && pt[count].y<image.rows-1){
        short v=destdisp.at<T>(y,x);
        if(v<disp.at<T>(j,i)){
            destimage.at<uchar>(y,3*x+0)=img[3*i+0];
            destimage.at<uchar>(y,3*x+1)=img[3*i+1];
            destimage.at<uchar>(y,3*x+2)=img[3*i+2];
            destdisp.at<T>(y,x)=disp.at<T>(j,i);

            if(isSub){
                if((int)pt[count+image.cols].y-y>1 && (int)pt[count+1].x-x>1{
                    destimage.at<uchar>(y,3*x+3)=img[3*i+0];
                    destimage.at<uchar>(y,3*x+4)=img[3*i+1];
                    destimage.at<uchar>(y,3*x+5)=img[3*i+2];

                    destimage.at<uchar>(y+1,3*x+0)=img[3*i+0];
                    destimage.at<uchar>(y+1,3*x+1)=img[3*i+1];
                    destimage.at<uchar>(y+1,3*x+2)=img[3*i+2];

                    destimage.at<uchar>(y+1,3*x+3)=img[3*i+0];
                    destimage.at<uchar>(y+1,3*x+4)=img[3*i+1];
                    destimage.at<uchar>(y+1,3*x+5)=img[3*i+2];

                    destdisp.at<T>(y,x+1)=disp.at<T>(j,i);
                    destdisp.at<T>(y+1,x)=disp.at<T>(j,i);
                    destdisp.at<T>(y+1,x+1)=disp.at<T>(j,i);
                }
                else if((int)pt[count-image.cols].y-y<-1 && (int)pt[count-1].x-x<-1){
                    destimage.at<uchar>(y,3*x-3)=img[3*i+0];
                    destimage.at<uchar>(y,3*x-2)=img[3*i+1];
                    destimage.at<uchar>(y,3*x-1)=img[3*i+2];

                    destimage.at<uchar>(y-1,3*x+0)=img[3*i+0];
                    destimage.at<uchar>(y-1,3*x+1)=img[3*i+1];
                    destimage.at<uchar>(y-1,3*x+2)=img[3*i+2];

                    destimage.at<uchar>(y-1,3*x-3)=img[3*i+0];
                    destimage.at<uchar>(y-1,3*x-2)=img[3*i+1];
                    destimage.at<uchar>(y-1,3*x-1)=img[3*i+2];

                    destdisp.at<T>(y,x-1)=disp.at<T>(j,i);
                    destdisp.at<T>(y-1,x)=disp.at<T>(j,i);
                    destdisp.at<T>(y-1,x-1)=disp.at<T>(j,i);
                }
                else if((int)pt[count+1].x-x>1){
                    destimage.at<uchar>(y,3*x+3)=img[3*i+0];
                    destimage.at<uchar>(y,3*x+4)=img[3*i+1];
                    destimage.at<uchar>(y,3*x+5)=img[3*i+2];

                    destdisp.at<T>(y,x+1)=disp.at<T>(j,i);
                }
                else if((int)pt[count-1].x-x<-1){
                    destimage.at<uchar>(y,3*x-3)=img[3*i+0];
                    destimage.at<uchar>(y,3*x-2)=img[3*i+1];
                    destimage.at<uchar>(y,3*x-1)=img[3*i+2];

                    destdisp.at<T>(y,x-1)=disp.at<T>(j,i);
                }
                else if((int)pt[count+image.cols].y-y>1){
                    destimage.at<uchar>(y+1,3*x+0)=img[3*i+0];
                    destimage.at<uchar>(y+1,3*x+1)=img[3*i+1];

```

```

destimage . at<uchar>(y+1,3*x+2)=img[3*i+2];

destdisp . at<T>(y+1,x)=disp . at<T>(j , i);
}

else if((int)pt [ count -image . cols ].y -y <-1){
destimage . at<uchar>(y -1,3*x+0)=img[3*i+0];
destimage . at<uchar>(y -1,3*x+1)=img[3*i+1];
destimage . at<uchar>(y -1,3*x+2)=img[3*i+2];

destdisp . at<T>(y -1,x)=disp . at<T>(j , i);
}
}

}

}

}

}

if(isSub)
{
Mat image2;
Mat disp2;
destimage . copyTo(image2 );
destdisp . copyTo(disp2 );
const int BS=1;
//#pragma omp parallel for
for(int j=BS;j<image . rows -BS;j++){
uchar* img=destimage . ptr<uchar>(j );
T* m = disp2 . ptr<T>(j );
T* dp = destdisp . ptr<T>(j );
for(int i=BS;i<image . cols -BS;i++){
if(m[ i ]==0){
int count=0;
int d=0;
int r=0;
int g=0;
int b=0;
for(int l=-BS;l<=BS;l++){
T* dp2 = disp2 . ptr<T>(j+l );
uchar* imageR = image2 . ptr<uchar>(j+l );
for(int k=-BS;k<=BS;k++){
if(dp2 [ i+k ]!=0){
count++;
d+=dp2 [ i+k ];
r+=imageR [ 3*( i+k )+0 ];
g+=imageR [ 3*( i+k )+1 ];
b+=imageR [ 3*( i+k )+2 ];
}
}
}
if(count !=0){
double div = 1.0/count;
dp[ i ]=d*div ;
img[ 3*i+0 ]=r*div ;
img[ 3*i+1 ]=g*div ;
img[ 3*i+2 ]=b*div ;
}
}
}
}
void fillOcclusion(Mat& src , int invalidvalue , bool isInv){
if(isInv){
if(src . type ()==CV_8U){

```

```

        fillOcclusionInv_<uchar>(src , (uchar)invalidvalue );
    }

    else if (src.type() == CV_16S){
        fillOcclusionInv_<short>(src , (short)invalidvalue );
    }

    else if (src.type() == CV_16U){
        fillOcclusionInv_<unsigned short>(src , (unsigned short)invalidvalue );
    }

    else if (src.type() == CV_32F){
        fillOcclusionInv_<float>(src , (float)invalidvalue );
    }

}

else {

    if (src.type() == CV_8U){
        fillOcclusion_<uchar>(src , (uchar)invalidvalue );
    }

    else if (src.type() == CV_16S){
        fillOcclusion_<short>(src , (short)invalidvalue );
    }

    else if (src.type() == CV_16U){
        fillOcclusion_<unsigned short>(src , (unsigned short)invalidvalue );
    }

    else if (src.type() == CV_32F){
        fillOcclusion_<float>(src , (float)invalidvalue );
    }

}

}

template <class T>
static void fillOcclusion_(Mat& src , T invalidvalue){
    int bb=1;
    const int MAX_LENGTH=src.cols*0.5;
    //#pragma omp parallel for
    for(int j=bb;j<src.rows-bb;j++){
        T* s = src.ptr<T>(j);
        //const T st = s[0];
        //const T ed = s[src.cols-1];
        s[0]=255;
        s[src.cols-1]=255;
        for(int i=0;i<src.cols;i++){
            if(s[i]<=invalidvalue){
                int t=i;
                do{
                    t++;
                    if(t>src.cols-1)break;
                }while(s[t]<=invalidvalue);

                const T dd = min(s[i-1],s[t]);
                if(t-i>MAX_LENGTH){
                    for(int n=0;n<src.cols;n++){
                        s[n]=invalidvalue ;
                    }
                }
                else{
                    for(;i<t;i++){
                        s[i]=dd;
                    }
                }
            }
        }
    }
}

```

trackObject.cpp

```

* trackObject.cpp
*
* Created on: Oct 19, 2015
* Author: nicolasrosa
*/
#include "trackObject.h"

void trackFilteredObject(int &x, int &y, Mat threshold, Mat &cameraFeed){

    Mat temp;
    threshold.copyTo(temp);
    //these two vectors needed for output of findContours
    std::vector<std::vector<Point>> contours;
    std::vector<Vec4i> hierarchy;
    //find contours of filtered image using openCV findContours function
    findContours(temp, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE );
    //use moments method to find our filtered object
    double refArea = 0;
    bool objectFound = false;
    if (hierarchy.size() > 0) {
        int numObjects = hierarchy.size();
        //if number of objects greater than MAX_NUM_OBJECTS we have a noisy filter
        if(numObjects>MAX_NUM_OBJECTS){
            for (int index = 0; index >= 0; index = hierarchy[index][0]) {

                Moments moment = moments((cv::Mat)contours[index]);
                double area = moment.m00;

                //if the area is less than 20 px by 20px then it is probably just noise
                //if the area is the same as the 3/2 of the image size, probably just a bad filter
                //we only want the object with the largest area so we save a reference area each
                //iteration and compare it to the area in the next iteration.
                if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea){
                    x = moment.m10/area;
                    y = moment.m01/area;
                    objectFound = true;
                    refArea = area;
                } else objectFound = false;
            }
            //let user know you found an object
            if(objectFound ==true){
                putText(cameraFeed, "Tracking_Object", Point(0,50),2,1,Scalar(0,255,0),2);
                //draw object location on screen
                drawObject(x,y,cameraFeed);}
        } else putText(cameraFeed, "TOO MUCH NOISE!_ADJUST FILTER", Point(0,50),1,2,Scalar(0,0,255),2);
    }
}

void drawObject(int x, int y, Mat &frame){

    //use some of the openCV drawing functions to draw crosshairs
    //on your tracked image!

    //UPDATE:JUNE 18TH, 2013
    //added 'if' and 'else' statements to prevent
    //memory errors from writing off the screen (ie. (-25,-25) is not within the window!)

    circle(frame, Point(x,y),20,Scalar(0,255,0),2);
    if(y-25>0)
        line(frame, Point(x,y), Point(x,y-25), Scalar(0,255,0),2);
    else line(frame, Point(x,y), Point(x,0), Scalar(0,255,0),2);
}

```

```
if (y+25<FRAME_HEIGHT)
line (frame , Point(x,y) , Point(x,y+25) , Scalar(0,255,0),2);
else line (frame , Point(x,y) , Point(x,FRAME_HEIGHT) , Scalar(0,255,0),2);
if (x-25>0)
line (frame , Point(x,y) , Point(x-25,y) , Scalar(0,255,0),2);
else line (frame , Point(x,y) , Point(0,y) , Scalar(0,255,0),2);
if (x+25<FRAME_WIDTH)
line (frame , Point(x,y) , Point(x+25,y) , Scalar(0,255,0),2);
else line (frame , Point(x,y) , Point(FRAME_WIDTH,y) , Scalar(0,255,0),2);

putText (frame , intToString (x)+" "+intToString (y) , Point(x,y+30) , 1,1 , Scalar(0,255,0),2);

}

string intToString (int number){
stringstream ss;
ss << number;
return ss.str();
}
```