

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

**Desenvolvimento de algoritmo de visão para
VANTS sobre Linux Embocado**

Autor: Nícolas dos Santos Rosa
Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2015

Nícolas dos Santos Rosa

Desenvolvimento de algoritmo de visão 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

2015

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 a minha mãe, meu pai, minha irmã, meus padrinhos e 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 atazanar 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. Farçoni, Plínio F. G. Bueno, 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 à todos os outros amigos.

Aos membros do grupo extracurricular Warthog Robotics, por partilhar um interesse em comum e pelas incontáveis horas de dedicação gastos 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 **Desenvolvimento de algoritmo de visão para VANTS sobre Linux Embarcado.** Trabalho de Conclusão de Curso – Escola de Engenharia de São Carlos, Universidade de São Paulo, 2015.

Atualmente, veículos aéreos não tripulado (VANT) vem tornando-se um assunto recorrente no âmbito científico. Estes veículos, devido a sua mobilidade e inteligência artificial, vem 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, estuda-se a utilização de visão estéreo em sistemas embarcados para reconhecimento de obstáculos que ameacem a locomoção do veículo autônomo. Por fim, será desenvolvido um algoritmo utilizando visão computacional que estime as distâncias de objetos próximos ao veículo móvel.

Palavras-Chave: Visão estéreo, Detecção de Obstáculos, Sistemas Embarcados, Veículos Aéreos Não Tripulado - VANT, Visão Computacional.

Abstract

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 paper, it will be studied the use of stereoscopic vision in embedded systems to recognize obstacles that threaten the mobility of the autonomous vehicle. Finally, an algorithm using computer vision to estimate the distances of objects near the mobile vehicle will be developed.

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

Lista de Figuras

2.1	Modelo Idealizado de um sistema de visão estéreo. Imagem retirada de [1]	30
2.2	Geometria Epipolar. Imagem retirada de [2]	31
2.3	Calibração estéreo dos parâmetros intrínsecos - Modelo de Distorções da câmera esquerda e direita, respectivamente.	33
2.4	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].	34
2.5	Calibração estéreo dos parâmetros extrínsecos - Estimativa do posicionamento da câmera direita ²	35
2.6	Retificação de Imagens - Processo de retificação estéreo. Imagem retirada de [1]	35
2.7	Correspondência de pontos homólogos em um par de imagens estéreo. Imagem original retirada de [3].	36
2.8	Caminhão Autônomo desenvolvido pelo Laboratório de Robótica Móvel - LRM - ICMC/USP em parceria com a Scania	37
2.9	Plataformas experimentais de aeronaves com o Sistema Estéreo - FPGA (frente) e o Sistema Estéreo - Pushbroom (trás). Câmaras são montados na parte dianteira das asas na mesma linha de base (34 cm) em ambas células.	38
2.10	Veículo Submarino Autônomo - Girona 500	39
3.1	3D Webcam Minoru	42
3.2	Câmera Digital Fujifilm FinePix Real 3D W3	43
3.3	Plataforma de Desenvolvimento - BeagleBone Black	44
3.4	Plataforma de Desenvolvimento - Jetson TK1	44
3.5	Padrão de Calibração	46
3.6	Quadricóptero 3DR X8 com suporte para a Câmera 3D	46
3.7	Cenário 1 - Ambiente Externo - Árvore	47

3.8	Cenário 2 - Ambiente Interno - Mesa/Cadeira/Estantes	48
3.9	Cenário 3 - Bancada/Quadricóptero	49
3.10	Etapas do Processamento de Imagens	50
4.1	Interface Gráfica - Visualização simultânea dos quadros das câmeras esquerda e direita	56
4.2	Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB	57
4.3	Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB	57
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	58
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	59
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	60

Lista de Tabelas

3.1	Especificações - 3D Webcam Minoru	42
3.2	Especificações - Câmera Digital Fujifilm FinePix Real 3D W3	43
3.3	Especificações - BeagleBone Black	44
3.4	Especificações - Jetson TK1	45
3.5	Especificações - Asus Q550LF	45
3.6	Versões utilizadas de CUDA e OpenCV	51
4.1	Desempenho atingido de cada plataforma utilizando o método de correspondências BM	60

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
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
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
SGBM	<i>Semi-Global Block Matching</i> - Método Estéreo Semi-global

SLAM *Simultaneous Localization and Mapping* - Localização e Mapeamento Simultâneo

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

Sumário

1	Introdução	25
1.1	Objetivos	26
1.2	Justificativa	26
1.3	Motivação	27
1.4	Organização do trabalho	27
2	Fundamentos Teóricos	29
2.1	Visão Estéreo - <i>Stereo Imaging</i>	29
2.1.1	Triangulação - <i>Triangulation</i>	29
2.1.2	Geometria epipolar - <i>Epipolar Geometry</i>	30
2.1.3	Calibração das Câmeras- <i>Calibration</i>	31
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	36
3	Materiais e Métodos	41
3.1	Materiais	41
3.1.1	Câmeras estéreo	41
3.1.2	Unidades de Processamento	43
3.1.3	Equipamentos auxiliares	45
3.2	Métodos	47
3.2.1	Cenários	47
3.2.2	Processamento de Imagem	49
3.2.3	Aceleração em Hardware - <i>Hardware Acceleration</i>	50
3.2.4	Jetson TK1 - Configuração da Plataforma	51

4 Resultados	55
4.1 Interface Gráfica - <i>StereoVisionGUI</i>	55
4.2 Comparaçao de Desempenho: Desktop x BBB x Jetson TK1	60
4.2.1 Desktop	61
4.2.2 BeagleBone Black(BBB)	61
4.2.3 Jetson TK1	61
5 Conclusão	63
A Apêndice 1	67
I Anexo 1	69

Capítulo 1

Introdução

A pesquisa em veículos aéreos não tripulado (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 enfrenta o problema de localização e mapeamento simultâneo, mais conhecido como SLAM (*Simultaneous Localization and Mapping*) [4]. SLAM apresenta quatro etapas: Mapeamento, Percepção, Localização e Modelagem. A complexidade deste problema encontra-se no fato de que o veículo necessita navegar em um espaço desconhecido, extrair 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).

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*) [5], planejamento de caminho (*Path Planning*) e arquitetura de máquina de estados (*Decision Making*).

Um sistema autônomo também implica o processamento de navegação, detecção de obs-

táculos, tomada de decisão, sejam embarcados, isto é, todo processamento necessário deve ser realizado *Online*. Este trabalho objetiva-se no estudo das ações que devam ser tomadas para que o algoritmo de detecção de obstáculos utilizando visão estéreo seja embarcado. Deste modo, as plataformas de desenvolvimento BeagleBone Black e NVIDIA Jetson TK1 serão analisadas e suas performances avaliadas ao executar o algoritmo desenvolvido [6].

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. Desenvolvimento de algoritmos de visão estéreo para Linux embarcado para aplicação em Quadricópteros.
4. Comparativo de desempenho do algoritmo desenvolvido em diferentes plataformas.
5. Estudo e aplicação de métodos para a aceleração em hardware dos algoritmos desenvolvidos.

1.2 Justificativa

A pouco mais de trinta anos, o VANT BQM-1BR realizava seu primeiro voo em espaço aéreo brasileiro. 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 [7]. 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 [8].

A *Federal Aviation Administration* (FAA) apresenta as mesmas dificuldades para a integração deste 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 auxiliar o primeiro passo de mapeamento de ambientes através de visão estéreo. Também é motivado pela tentativa de reproduzir-se o desafio proposto pela *Autonomous Aerial Vehicle Competition* (AAVC)[9], 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 (veja seção 2.1.6).

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 prentensões. 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 boa 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 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 Imaging*

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.

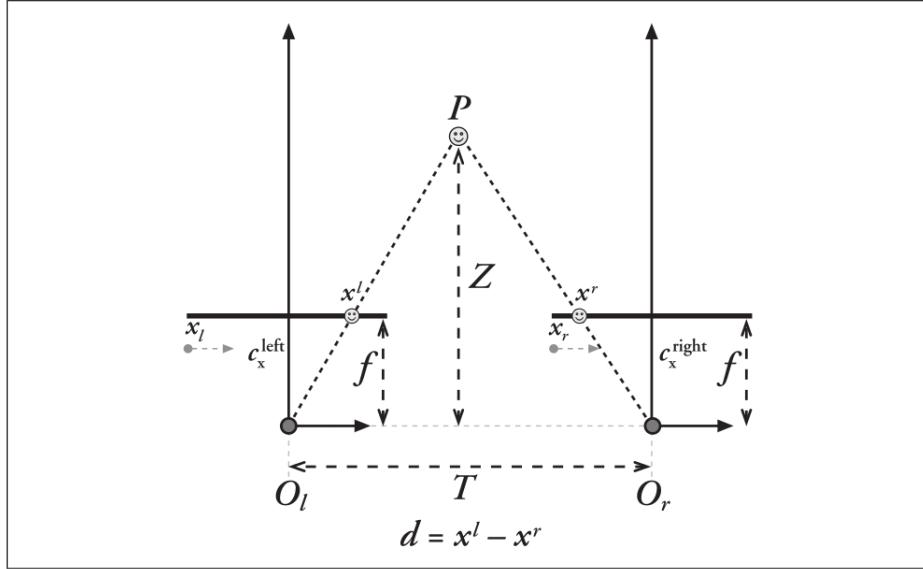


Figura 2.1: Modelo Idealizado de um sistema de visão estéreo. Imagem retirada de [1]

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.

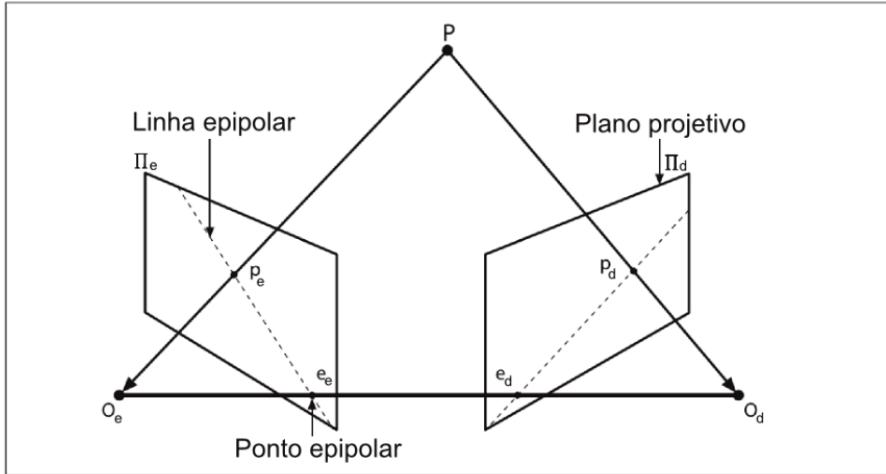


Figura 2.2: Geometria Epipolar. Imagem retirada de [2]

Projeta-se o ponto P no centros de projeção O^e e O^d , as linha que interligam o ponto P à 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 . Entretanto, a restrição epipolar garante que o ponto homólogo deve estar sobre a linha epipolar da imagem direita. Deste modo, é possível restringir a busca à 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. A expressão 2.2 representa a restrição epipolar é dada relação entre a matriz fundamental F e os dois pontos correspondentes, em coordenadas de pixel [10].

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

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. Isto permite que os métodos computacionais obtenham resultados mais coerentes com a realidade. Estes parâmetros são classificadas 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 2.3 ilustra o modelo completo, componente radial e componente tangencial do modelo de distorções das lentes da câmera esquerda e direita, respectivamente. [11].

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

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)

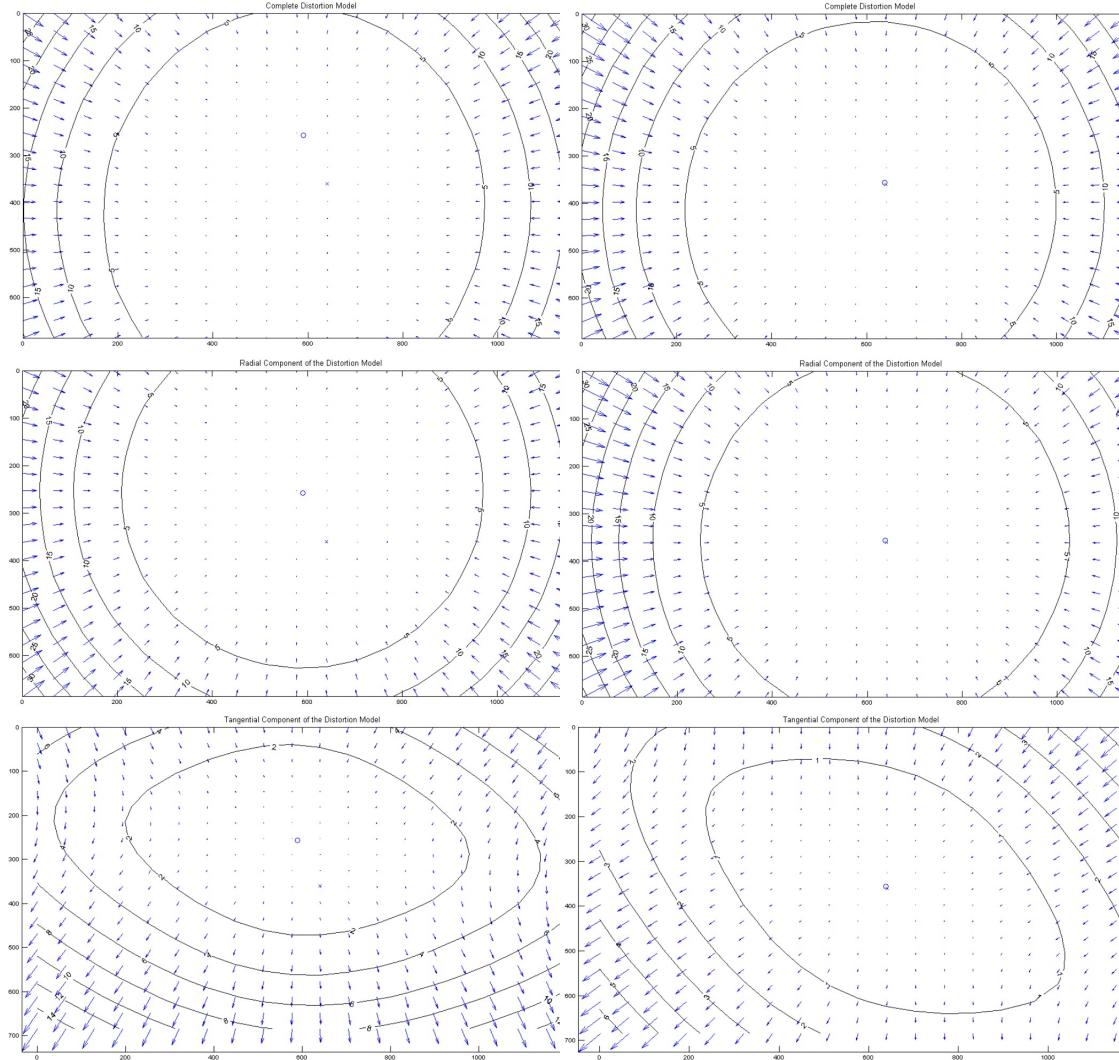


Figura 2.3: Calibração estéreo dos parâmetros intrínsecos - Modelo de Distorções da câmera esquerda e direita, respectivamente.

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 2.4, a matriz de rotação R (3×3) e o vetor de translação T (3×1) são responsáveis por descrever este deslocamento. Geralmente, quando o quadro de referência não encontra-se 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 [10].

$$s \cdot p' = M \left[\begin{array}{c|c} R & t \end{array} \right] P \quad (2.5)$$

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

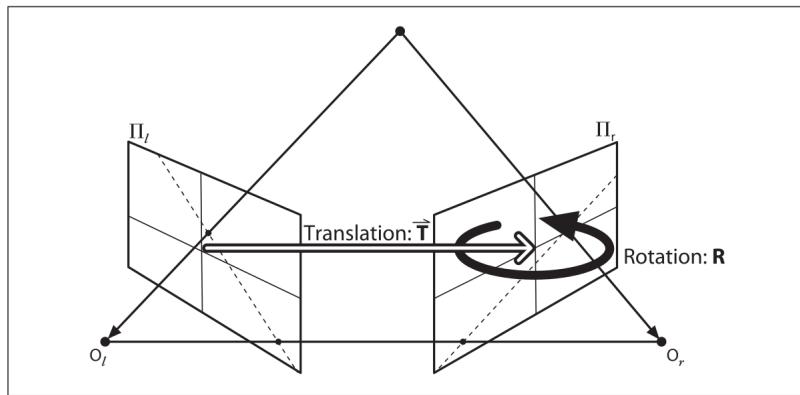


Figura 2.4: O deslocamento do plano Π_r com relação ao plano Π_l pode ser descrito pela matriz R e o vetor de translação T . Imagem retirada de [1].

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 2.5, 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[12].

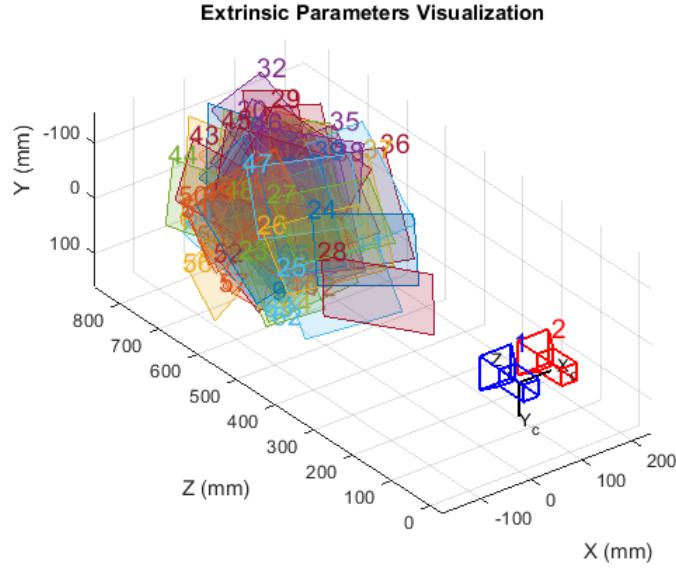


Figura 2.5: Calibração estéreo dos parâmetros extrínsecos - Estimativa do posicionamento da câmera direita²

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 estejam retificadas e sem distorções, preparadas para a aplicação dos métodos estéreo, assim como ilustrado na figura 2.6.

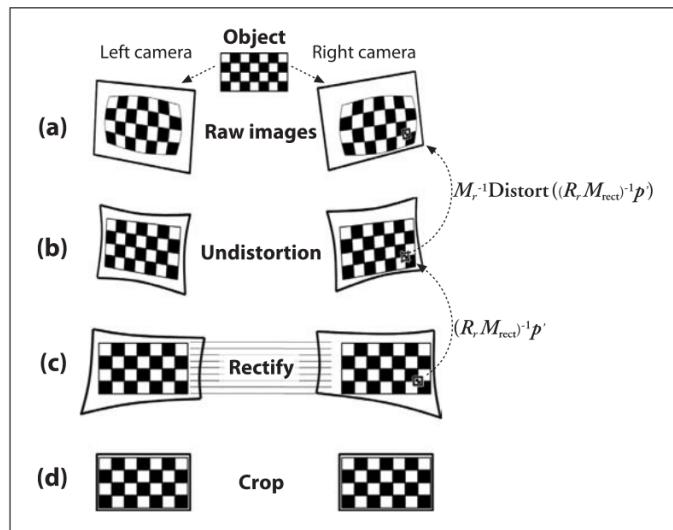


Figura 2.6: Retificação de Imagens - Processo de retificação estéreo. Imagem retirada de [1]

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 2.7 ilustra um par de imagens estéreo (esquerda e direita), nas quais pixels homólogos estão destacados. Os métodos estéreo utilizam da restrição epipolar o que reduzir o espaço de busca e de diferentes meios para encontrá-los. Mesmo com essa essa restrição e com a retificação das imagens, os métodos ainda assim apresenta um elevado custo computacional, além de que ainda estão sujeitos à encontrarem falsas correspondências.



Figura 2.7: Correspondência de pontos homólogos em um par de imagens estéreo. Imagem original retirada de [3].

Abaixo, encontra-se apresentado um pequeno resumo de como são os operadores dos métodos estéreo utilizado neste trabalho.

Método Estéreo - *Block Matching* - BM

Método Estéreo Semi-global - *Semi-global Block Matching* - SGBM

Método Estéreo Local com Aceleração de GPU - *Block Matching with GPU Acceleration* - BMGPU

Mapa de disparidades - *Disparity Map*

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 vem participando de uma verdadeira cor-

rida 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, o que concretizam cada vez mais essa realidade tida até então como distante. Na figura 2.8, 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 [13].



Figura 2.8: 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 estas pequenas aeronaves consigam navegar autonomamente e desviar de obstáculos voando a um velocidade de 30 mph (48 km/h). A figura 2.9 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 *Semi-Global Block Matching* (SGBM) [14].

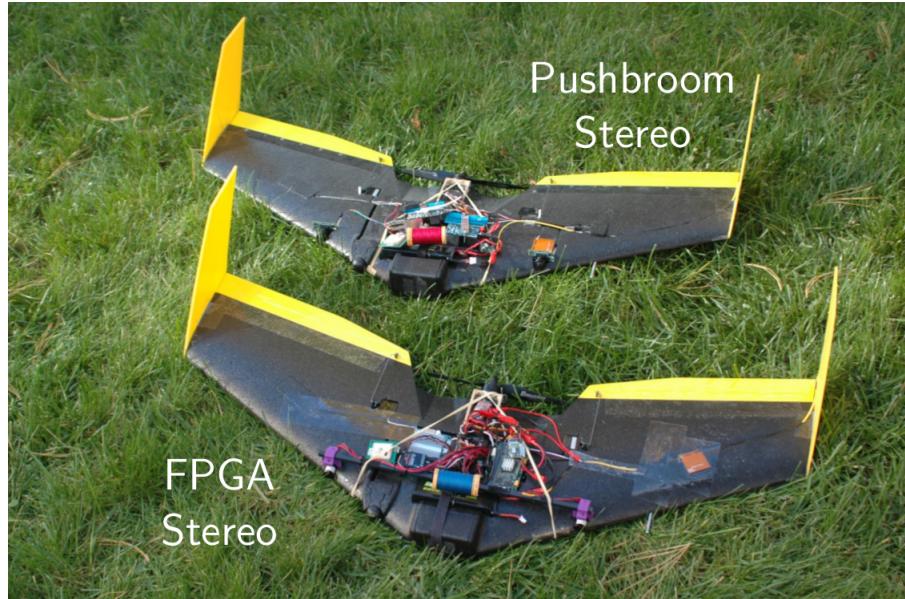


Figura 2.9: Plataformas experimentais de aeronaves com o Sistema Estéreo - FPGA (frente) e o Sistema Estéreo - Pushbroom (trás). Câmaras são montados na parte dianteira das asas na mesma linha de base (34 cm) em ambas células.

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 2.10). 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 [15].

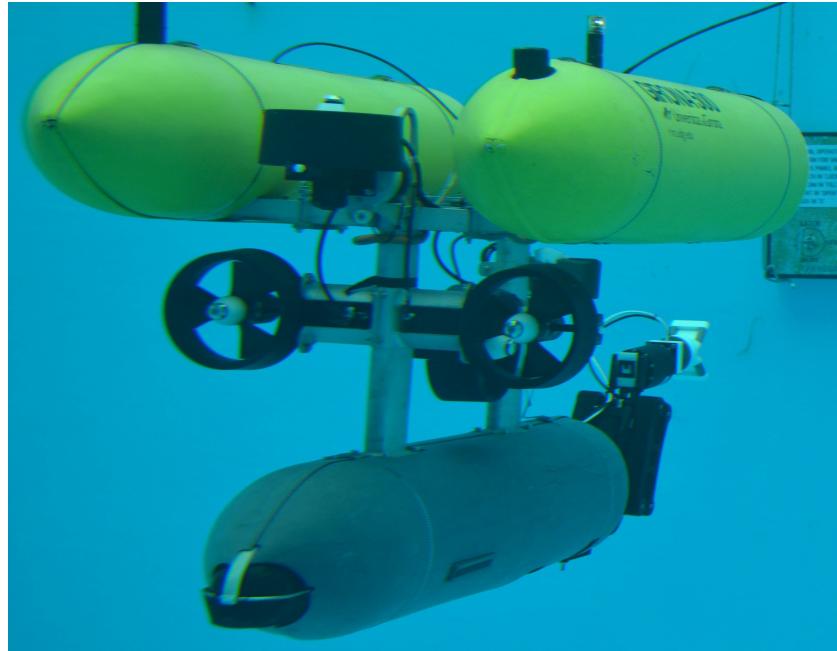


Figura 2.10: Veículo Submarino Autônomo - Girona 500

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

O projeto já 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*.

Tabela 3.1: Especificações - 3D Webcam Minoru

Sensor de Imagem	VGA CMOS Sensor
Resolução Máxima	800x600
Tamanho Linha de Base	6 cm
Taxa de Captura	30 fps
Distância Focal	10 cm até ∞
Campo de Visão	42°
Peso	249.48 g



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 preocupa-se principalmente na identificação de obstáculos, isso não oferece nenhuma desvantagem para o desenvolvimento do algoritmo. Todavia, para uma aplicação real, a câmera instalada no veículo deve apresentar esse aspecto. A tabela 3.2 apresenta as especificações da câmera em questão.

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

Sensor de Imagem	10 MP CCD Sensor
Resolução Máxima	1280x720
Tamanho Linha de Base	7.5 cm
Taxa de Captura	24 - 30 fps
Distância Focal	60 cm até ∞
Peso	250g



Figura 3.2: Câmera Digital Fujifilm FinePix Real 3D W3

3.1.2 Unidades de Processamento

Visto que este trabalho também busca a implementação dos algoritmos de detecção de obstáculos em quadricópteros, tem-se como objetivo sua implementação para Linux embarcado (*Embedded Linux*). Em seguida, estão apresentadas as plataformas que serão utilizadas para este propósito.

BeagleBone Black

A primeira a ser utilizada e estudada é 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, e ao seu poder de processamento, utiliza 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

A segunda a ser utilizada e estudada é a plataforma Jetson TK1 produzida pela NVIDIA, 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 escolhida 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
Armanezamento	16GB fast eMMC 4.51 (routed to SDMMC4)
Sistemas Operacionais	Platforma 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

A terceira plataforma utilizada é um *Notebook* Asus Q550LF, ela 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 a imagem 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.

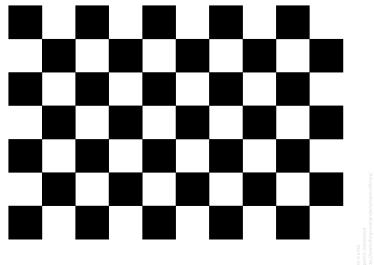


Figura 3.5: Padrão de Calibração

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 pela 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 métodos utilizado para o desenvolvimento do algoritmo para a detecção de obstáculos.

3.2.1 Cenários

O algoritmo de detecção de obstáculos desenvolvido tenta contornar todas as adversidades propostas pelos seguintes cenários. Propõe-se que ele deve ser flexível à variações na luminosidade, capaz de detectar obstáculos estáticos e móveis, e ser imune à 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 que poderiam atrapalhar e tornar o algoritmo 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.



Figura 3.7: Cenário 1 - Ambiente Externo - 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.

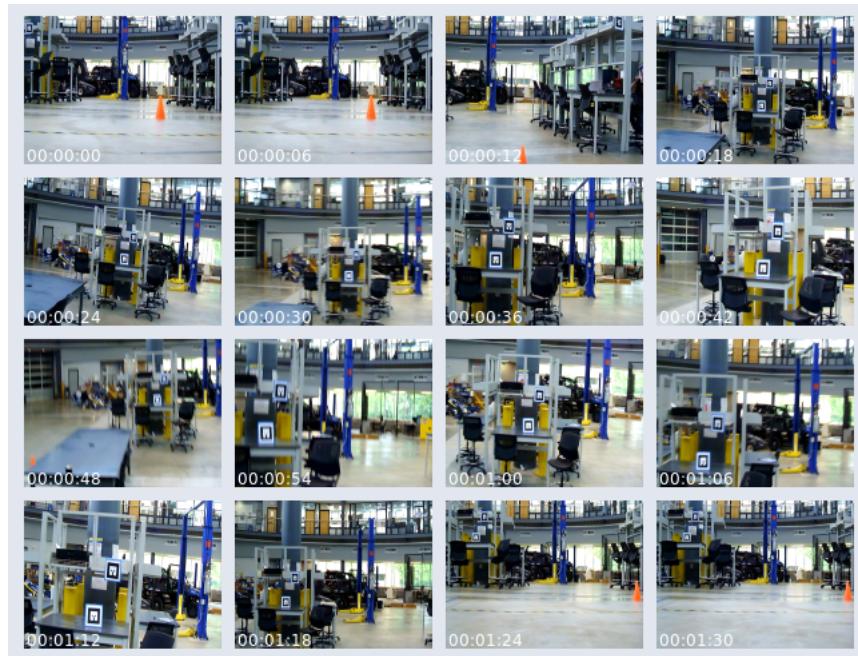


Figura 3.8: Cenário 2 - Ambiente Interno - Mesa/Cadeira/Estantes

Cenário 3

O cenário da figura 3.9 foi utilizado para estudo das condições de ambiente interno, o qual também apresenta certa variação de luminosidade, apresenta um número muito maior de movimentos e permite a análise de objetos móveis à curto e médio alcance. Os principais obstáculos deste cenário é uma bancada e um outro quadricóptero no campo de visão do veículo pilotado.



Figura 3.9: Cenário 3 - Bancada/Quadricóptero

3.2.2 Processamento de Imagem

Nessa seção será apresentado o processamento de imagens utilizado para a identificação de obstáculos. Como pode ser visto na figura 3.10, todo o processo conta com seis etapas.

Câmeras: O primeiro passo do processo é a captura das imagens da câmera estéreo. Idealmente, as imagens deve 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.

Pré-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.

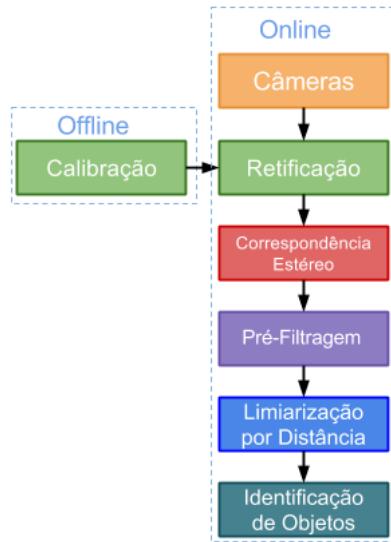


Figura 3.10: Etapas do Processamento de Imagens

3.2.3 Aceleração em Hardware - *Hardware Acceleration*

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 e visualização de moléculas

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, apresentaram o primeiro compilador, Brook, que facilitaria a implementação de software, visto que permitia a lidar de uma maneira mais maleável o fluxo de dados, podendo principalmente paralelizá-los. Em 2006, a NVIDIA juntamente com Ian Buck, criador do

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

Brook, desenvolveram 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)[16].

Neste trabalho, utilizou-se diferentes versões desta plataforma 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.

NEON

3.2.4 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 o página da NVIDIA e procure pelo 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:~$ cd /home/<user>
$ ubuntu@ubuntu :~/home/<user>$ mkdir flash
$ ubuntu@ubuntu :~/home/<user>$ cd flash
$ ubuntu@ubuntu :~/home/<user>/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:~/home/<user>/flash$ chmod +x JetPack-${VERSION}.run
$ ubuntu@ubuntu:~/home/<user>/flash$ ./JetPack-${VERSION}.run
```

2. Instalando o JetPack L4T

Sigua 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. Sigua 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 OpenCV4Tetra [17].

Capítulo 4

Resultados

Os resultados apresentados neste capítulo estão inteiramente relacionados as funções fundamentais presentes na interface gráfica desenvolvida. Além disso, tem-se presente a seção que apresenta os dados obtidos através 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 disparidade 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 câmera. A figura 4.1 ilustra o comportamento do software quando essa opção é selecionada.

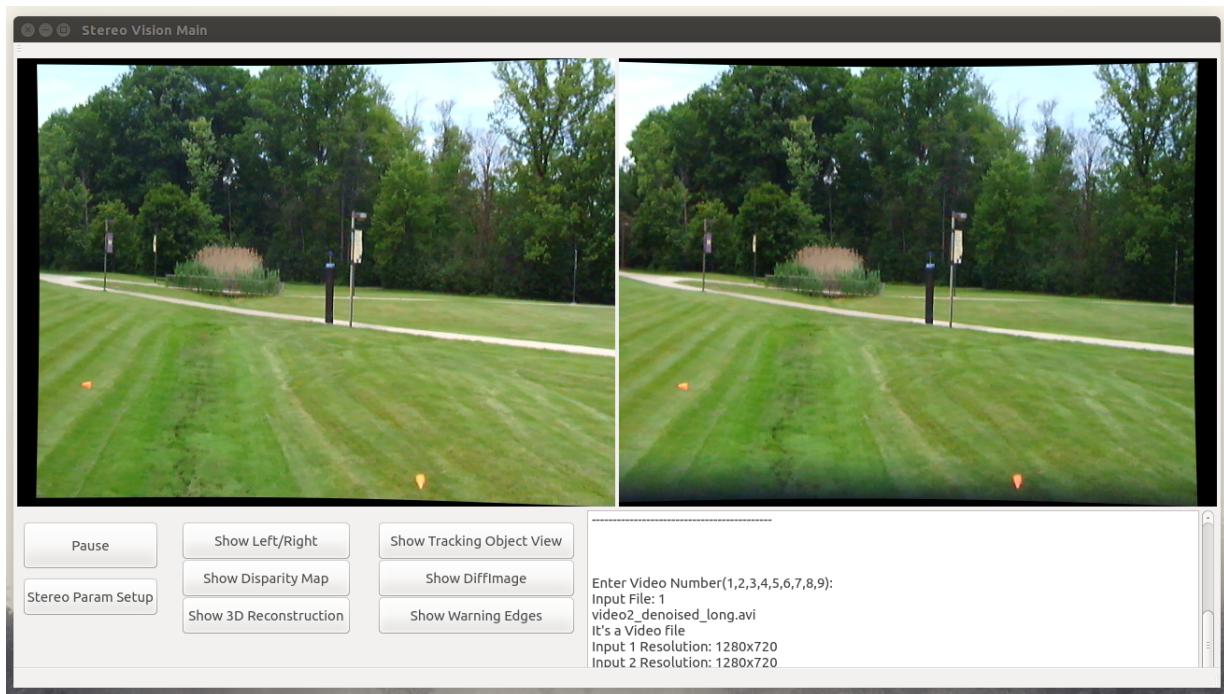


Figura 4.1: Interface Gráfica - Visualização simultânea dos quadros das câmeras esquerda e direita

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.

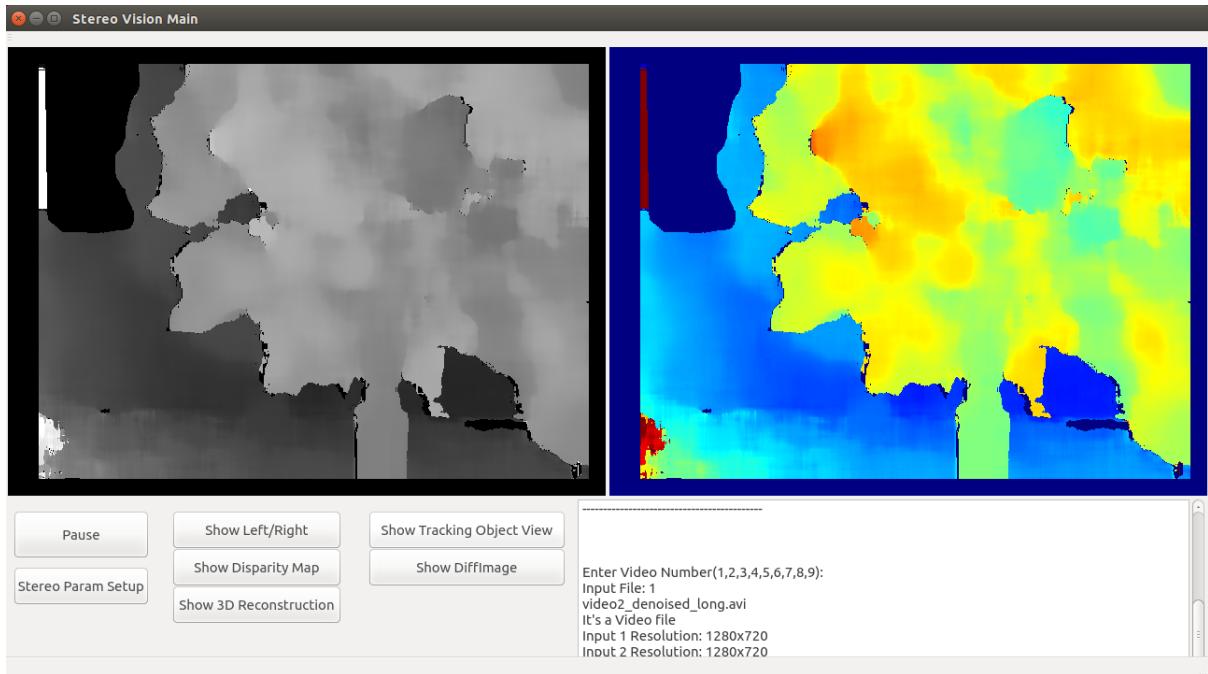


Figura 4.2: Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB

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.

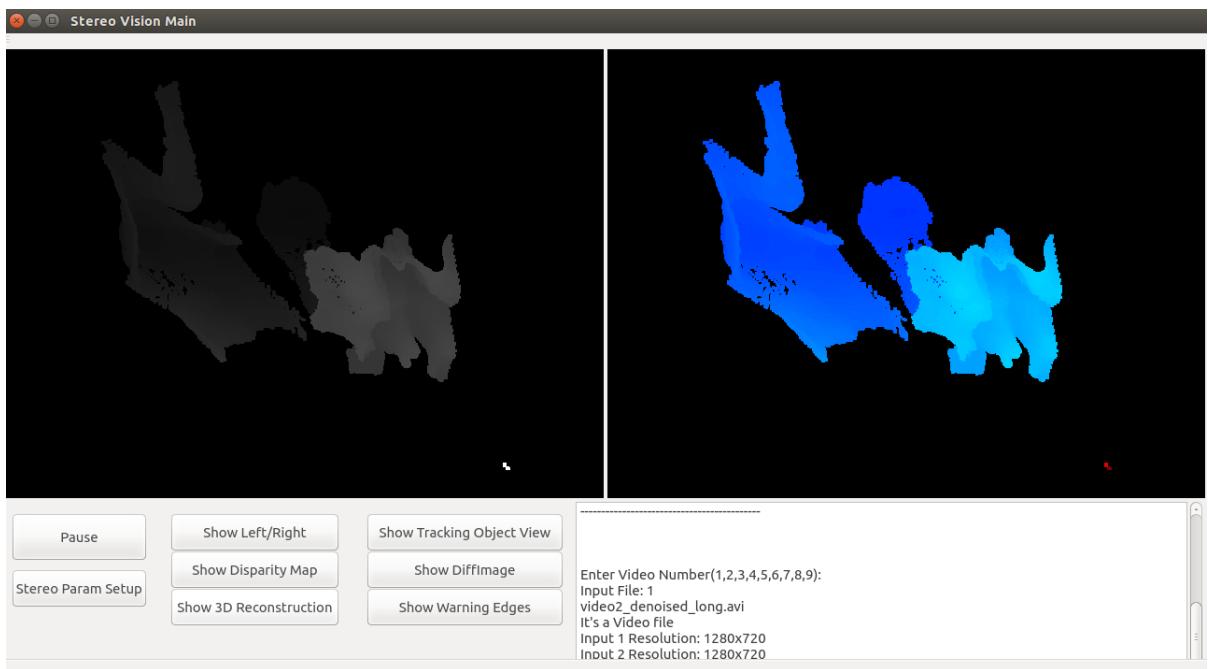


Figura 4.3: Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB

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.

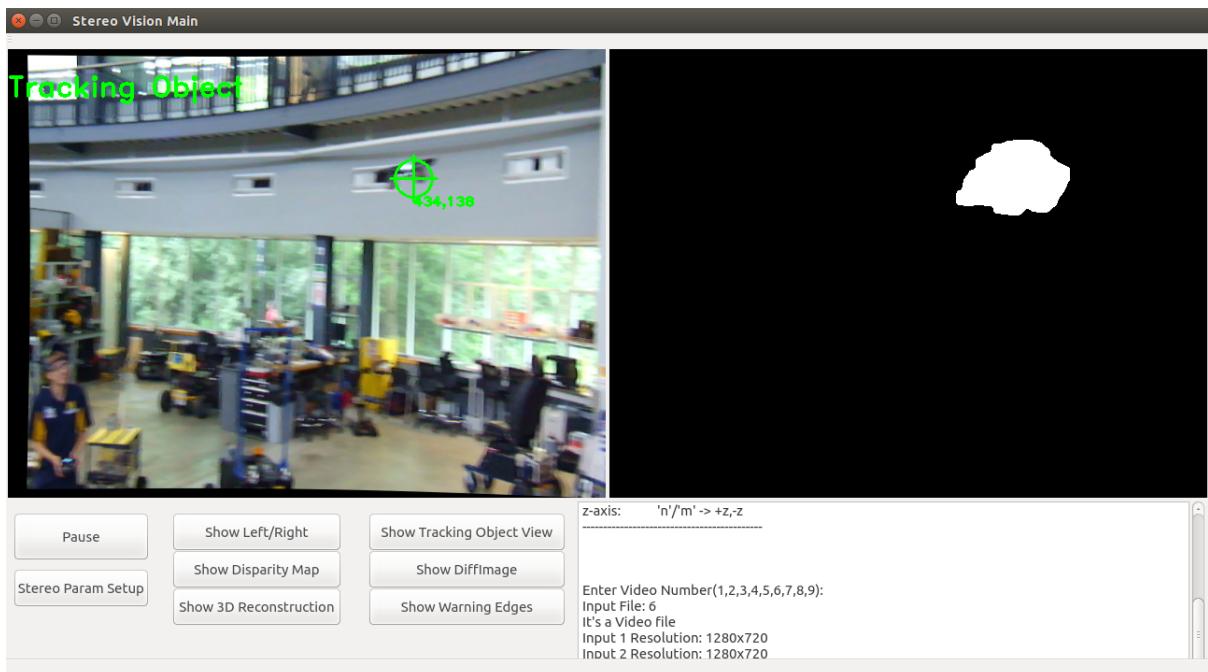


Figura 4.4: Interface Gráfica - Visualização da Imagem da Câmera Esquerda com o indicador de objeto rastreado e da Imagem binária resultante da limiarização por distância

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.

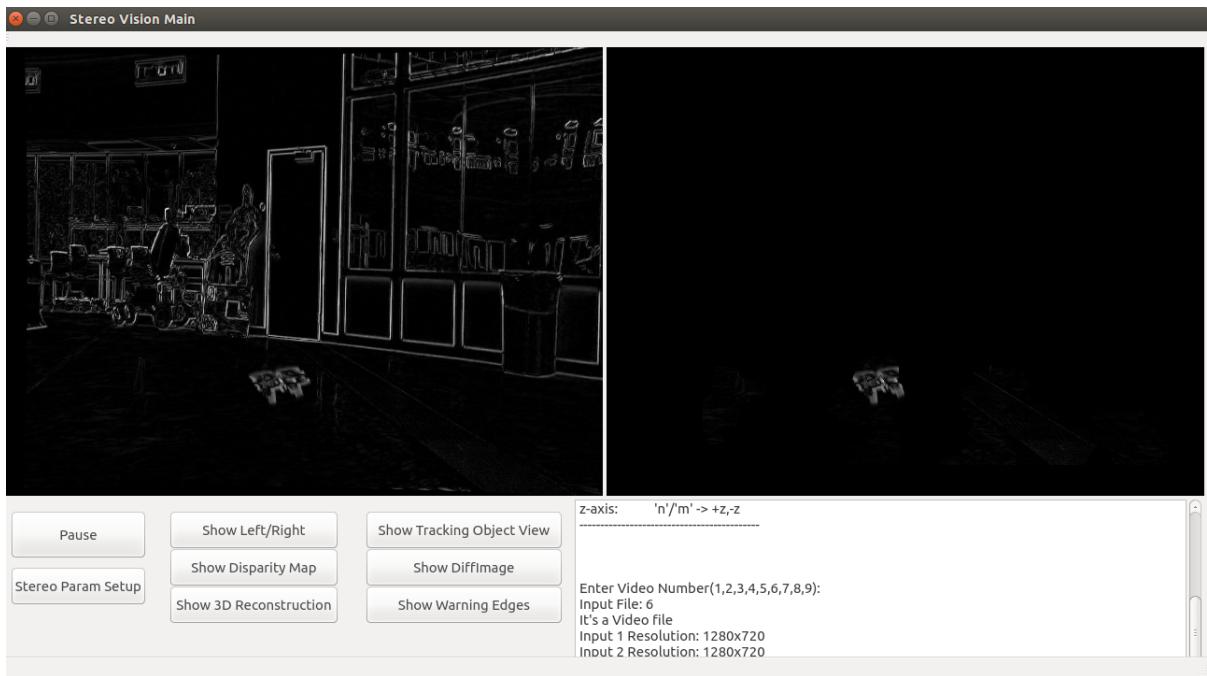


Figura 4.5: Interface Gráfica - Visualização da imagem resultante do processo de detecção de movimentos e imagem resultante do processo de detecção de movimentos limiarizada por distância

O botão *Show DiffImage* 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.

Tabela 4.1: Desempenho atingido de cada plataforma utilizando o método de correspondências BM

	CPU(FPS)	GPU/NEON(FPS)
Desktop	1	1
BBB	1	1
Jetson TK1	1	1

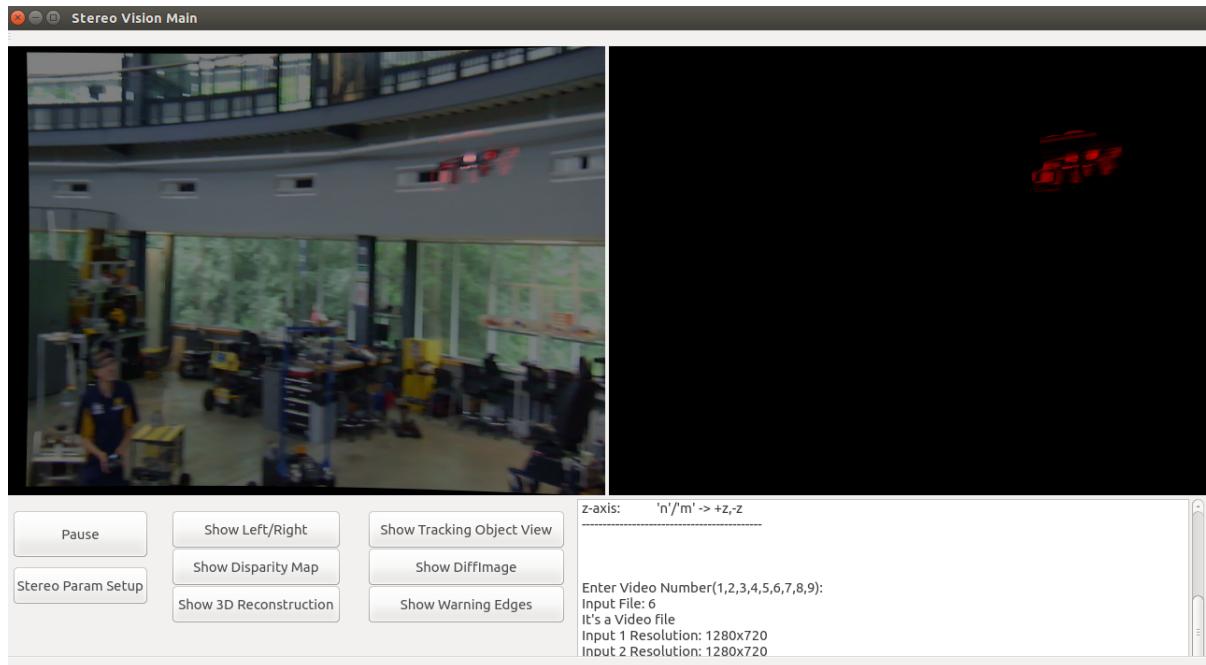


Figura 4.6: Interface Gráfica - Visualização da Imagem resultante da adição da imagem à direita com a Imagem da Câmera Esquerda e Imagem resultante do processo de realce das bordas dos objetos em movimento próximos ao veículo

4.2 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. O Método utilizado foi o BM para a comparação das plataformas, visto que é o que requisita menor processamento dentre os outros métodos mais conhecidos (SGBM, BP, CSBP, AD Census, ...). A tabela 4.1 abaixo, apresenta o resultado de desempenhos obtidos nas plataformas, apresentando comparativamente seu desempenho quando o processado utilizando CPU ou GPU/NEON (Caso da BBB).

4.2.1 Desktop

CPU

GPU

4.2.2 BeagleBone Black(BBB)

CPU2

NEON

4.2.3 Jetson TK1

CPU

GPU

Capítulo 5

Conclusão

Acredita-se que o trabalho desenvolvido até então está em dia com o cronograma, visto que a interface gráfica desenvolvida encontra-se totalmente funcional e o algoritmo detecta e segmenta relativamente bem os obstáculos dos cenários propostos. Entretanto, a taxa de processamento tornou-se uma preocupação, visto que o algoritmo desenvolvido consegue processar a uma taxa de captura de aproximadamente oito quadros por segundo. A alteração de plataforma poderia ser uma solução para o aumento da taxa de atualização, visto que o código implementado nos sistemas embarcados dispensam interface gráfica e diversas outras funcionalidades, como exemplo a funcionalidade para reconstrução tridimensional do ambiente. Todavia, essas plataformas não são tão poderosas quanto o computador da estação-base, logo, possivelmente essas plataformas apresentem performance inferior.

Deste modo, duas providências são cruciais para a continuidade do trabalho. A primeira é a otimização das rotinas utilizadas pelo algoritmo, sendo essa uma tentativa para a melhora do desempenho geral do método. A segunda é a execução de uma ampla revisão bibliográfica, incluindo principalmente trabalhos que apresentem comparativos de desempenho entre plataformas e que tenham realizados algum tipo de aceleração por *hardware*, sejam eles envolvendo paralelização de processos, implementação em FPGA ou utilizando a plataforma de computação paralela CUDA (*Compute Unified Device Architecture*). Deste modo, será possível identificar a plataforma mais adequada para este propósito.

Atividades Futuras:

- Otimizar Código
- Alterar Interface gráfica para suportar o Método SGBM
- Implementar Segmentação por Distância/Limiarização do mapa de disparidades utili-

zando Método de Otsu

- Aprimorar a suavização da Nuvem de Pontos Gerada
- Analisar Métodos de Abertura e Fechamento para aprimoramento na Segmentação
- Implementar funções para medir a Distância de objetos próximos
- Reavaliar a Calibração das Câmeras
- Realizar ensaio de Ground Truth para validação da Distância Detectada
- Implementar os algoritmos de Visão na BeagleBone e Jetson TK1
- Implementar os algoritmos de Visão na Jetson TK1
- Comparação de Desempenho: Desktop x BBB x Jetson TK1

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] 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)\nproblem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.
- [5] 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.
- [6] Sunil Shah. Real-time Image Processing on Low Cost Embedded Computers. pages 1–29, 2014.
- [7] José Luiz Boanova Filho. Aeronaves Não Tripuláveis no Brasil e sua Regulação. *Revista Brasileira de Direito Aeronáutico e Espacial*, 2014.
- [8] Departamento de Controle do Espaço Aéreo da Aeronáutica. DECEA publica nova regulamentação para voos de RPAS (drones), 2015.
- [9] FLYAAVC. Autonomous Aerial Vehicle Competition. <http://www.flyaavc.org/>, 2015.

- [10] Robert Laganière. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, 2011.
- [11] OpenCV Development Team. Camera Calibration and 3D Reconstruction. http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, 2014.
- [12] Jean-Yves Bouguet. Complete Camera Calibration Toolbox for Matlab, 1999.
- [13] 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*.
- [14] 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.
- [15] 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.
- [16] NVIDIA. Cuda - plataforma paralela de computação. http://www.nvidia.com.br/object/cuda_home_new_br.html, 2016.
- [17] Embedded Linux Wiki. Installing OpenCV (including the GPU module) on Jetson TK1. http://elinux.org/Jetson/Installing_OpenCV.
- [18] Etsuko; UEDA, Masanao; Koeda, Tsuyoshi; Suenaga, and Kentaro; Takemura. 3D Reconstruction and Point Cloud Rendering, 2011.
- [19] Kyle Hounslow. Real-Time Object Tracking Using OpenCV, 2013.

Apêndice A

Apêndice 1

Todos os trechos de código desenvolvidos para a estação-base da interface gráfica e para as outras plataformas podem ser encontradas no seguinte repositório:

Link para o Repositório: <https://github.com/nicolasrosa/StereoVision>

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 à parte de reconstrução tridimensional e renderização da nuvem de pontos [18]. A segunda funcionalidade corresponde à parte de rastreamento do objeto desenvolvida por Kyle Hounslow [19].

3DReconstruction.h

```
/*
 * 3DReconstruction.h
 *
 * Created on: Oct 25, 2015
 * Author: nicolasrosa
 */

#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& disp3Dviewer);

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 &disp3Dviewer);
    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&
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 mask;
if(mask.empty())mask=Mat::zeros(image.size(),CV_8U);
if(disp.type()==CV_8U){
    projectImagefromXYZ_<unsigned char>(image,destimage, disp, destdi
}
else if(disp.type()==CV_16S){
    projectImagefromXYZ_<short>(image,destimage, disp, destdisp, xyz
}
else if(disp.type()==CV_16U){
    projectImagefromXYZ_<unsigned short>(image,destimage, disp, destd
}
else if(disp.type()==CV_32F){
    projectImagefromXYZ_<float>(image,destimage, disp, destdisp, xyz
}
else if(disp.type()==CV_64F){
    projectImagefromXYZ_<double>(image,destimage, disp, destdisp, xyz
}

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,
    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>
            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 [ co
                    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)
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 func
    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 noise
        if (numObjects < MAX_NUM_OBJECTS) {
            for (int index = 0; index >= 0; index = hierarchy
```

```

    Moments moment = moments((cv::Mat)contour);
    double area = moment.m00;

    // if the area is less than 20 px by 20px
    // if the area is the same as the 3/2 of ...
    // we only want the object with the large ...
    // iteration and compare it to the area in ...
    if (area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA && area > ...
        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(x, y));
    // draw object location on screen
    drawObject(x, y, cameraFeed);
}

} else putText(cameraFeed, "TOO MUCH NOISE!_ADJUST_FILTER");
}

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 wi

    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);

}

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

```