

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 férrea 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, confirmado ainda mais minha paixão pelo meu curso.

Aos meus orientadores de iniciação científica, Prof. Dr. Cláudio F. M. Toledo, 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 estereoscópica 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 estereoscópica, Detecção de Obstáculos, Sistemas Embarcados, Veículos Aéreos Não Tripulado - VANT, Visão Computacional.

Abstract

Currently, unmanned aerial vehicles (UAV) is 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

1.1	Caminhão Autônomo desenvolvido pelo Laboratório de Robótica Móvel - LRM - ICMC/USP	27
1.2	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.	28
3.1	Modelo Idealizado de um sistema de Visão Estéreo	32
4.1	3D Webcam Minoru	36
4.2	Câmera Digital Fujifilm FinePix Real 3D W3	37
4.3	Plataforma de Desenvolvimento - BeagleBone Black	38
4.4	oi	38
4.5	Quadricóptero 3DR X8 com suporte para a Câmera 3D	39
4.6	Padrão de Calibração	39
5.1	Interface Gráfica - Visualização simultânea dos quadros das câmeras esquerda e direita	42
5.2	Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB	42
5.3	Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB	43
5.4	Interface Gráfica - Visualização da Imagem da Câmera Esquerda com o indicador de objeto rastreado e da Imagem binária resultante da limiarização por distância	44
5.5	Interface Gráfica - Visualização da imagem resultante do processo de detecção de movimentos e imagem resultante do processo de detecção de movimentos limiarizada por distância	45

5.6 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 46

Lista de Tabelas

4.1	Especificações - 3D Webcam Minoru	36
4.2	Especificações - Câmera Digital Fujifilm FinePix Real 3D W3	37
4.3	Especificações - BeagleBone Black	37
4.4	Especificações - Jetson TK1	38

Siglas

MVC	<i>Model-View-Controller</i> - Modelo-Visão-Controlador
POO	Programação Orientada a Objetos
UI	<i>User Interface</i> - Interface do Usuário
UML	<i>Unified Modeling Language</i> - Linguagem de Modelagem Unificada

Sumário

1	Introdução	25
1.1	Objetivos	26
1.2	Justificativa	26
1.3	Motivação	26
1.4	Organização do trabalho	28
2	Especificação do Projeto	29
2.1	Seção 1	29
2.2	Seção 2	29
3	Visão Estereoscópica - <i>Stereo Imaging</i>	31
3.1	Triangulação - <i>Triangulation</i>	31
3.2	Geometria epipolar - <i>Epipolar Geometry</i>	33
3.3	Calibração <i>Stereo Calibration</i>	33
3.3.1	Parâmetros intrínsecos	33
3.3.2	Parâmetros extrínsecos	33
3.4	Retificação de Imagens - <i>Stereo Rectification</i>	33
3.5	Correspondência Estéreo - <i>Stereo Correspondence</i>	33
3.6	Mapa de disparidades - <i>Disparity Map</i>	33
3.7	Projeção Tridimensional - <i>3D Reprojection</i>	33
3.8	Reconhecimento de Objeto - <i>Object Recognition</i>	33
3.8.1	Segmentação - <i>Image Segmentation</i>	33
3.8.2	Identificação - <i>Object Identification</i>	33
4	Materiais e Métodos	35
4.1	Materiais	35

4.1.1	Câmeras estereoscópicas	35
4.1.2	Unidades de Processamento	37
4.1.3	Equipamentos auxiliares	39
4.2	Métodos	39
5	Resultados Parciais	41
6	Conclusão	47
A	Apêndice 1	49
A.1	Interface Gráfica - StereoVision System	49
A.1.1	Bibliotecas	49
A.1.2	Códigos-Fonte	63
I	Anexo 1	111

Capítulo 1

Introdução

A pesquisa em VANTs – Veículos Aéreos não Tripulado 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) [?]. 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 ou por sensores ópticos.

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).

Um sistema autônomo também implica o processamento de imagens ser embarcado, isto é, o processamento para a navegação do veículo ser realizada *On-line*. Deste modo, as pla-

formas de desenvolvimento BeagleBone Black e Nvidia jetson TK1 serão analisadas e avaliadas com relação performance ao executar o algoritmo desenvolvido [?].

1.1 Objetivos

1. Estudo e aplicação de técnicas de visão computacional para visão estéreo.
2. Desenvolvimento de uma interface de apoio para o controle de um veículo autônomo.
3. Desenvolvimento de algoritmo de visão estéreo para Linux Embarcado e aplicação em Quadricópteros.
4. Comparativo de desempenho do algoritmo desenvolvido em diferentes plataformas.

1.2 Justificativa

1.3 Motivação

A proposta deste Trabalho de conclusão de Curso é auxiliar o primeiro passo de Mapeamento de ambientes através de visão estereoscópica. Este trabalho, é motivado pela tentativa de reproduzir-se o desafio proposto pela Autonomous Aerial Vehicle Competition (AAVC)[?], 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 estereoscópica em plataformas robóticas. Atualmente, existem diversas pesquisas voltadas ao desenvolvimento de veículos autonômios para navegação terrestre, aérea e subaquática. Nesta seção, serão apresentados alguns dos trabalhos envolvendo cada um destes ambientes, os quais câmeras estereoscópicas auxiliam ou são os principais sensores destas plataformas.

Nos dias de hoje, as empresas automobilísticas vem participando de uma verdadeira corrida tecnológica para o desenvolvimento de automóveis totalmente autônomos e economicamente viáveis. As universidades não ficaram para trás e também participam desta corrida.

Além disso, vêm apresentando resultados promissores, o que concretizam cada vez mais essa realidade tida até então como distante. Na figura 1.1 abaixo é possível observar o projeto de caminhão autônomo desenvolvido pelo laboratório de robótica móvel - LRM - ICMC/USP. O caminhão conta com diversos sensores, dentro eles câmeras estereoscópicas, que identificam outros automóveis, pessoas e faixas de sinalização [?].



Figura 1.1: Caminhão Autônomo desenvolvido pelo Laboratório de Robótica Móvel - LRM - ICMC/USP

No caso de navegação autônoma para ambiente áereo, 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 1.2 abaixo apresenta o trabalho desenvolvido, o qual é um comparativo de desempenho de uma implementação em hardware utilizando FPGA e um processador ARM para processamento embarcado do método *Semi-Global Block Matching* [?].

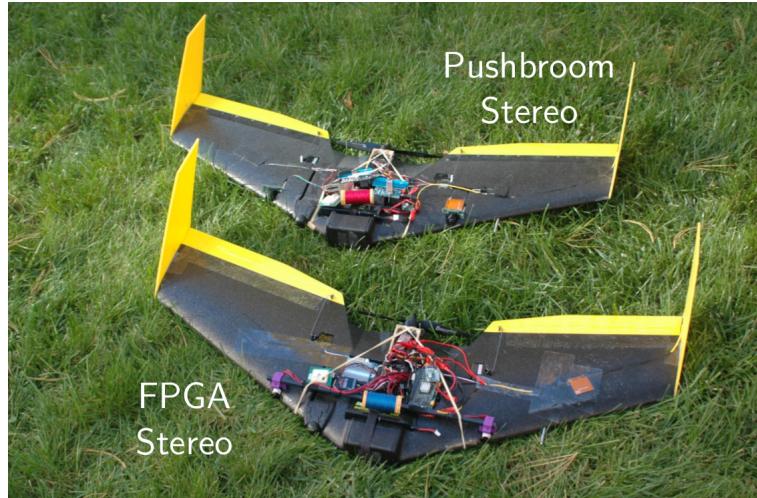


Figura 1.2: Plataformas experimentais de aeronaves com o Sistema Estéreo - FPGA (frente) e o Sistema Estéreo - Pushbroom (trás). Câmaras são montados na parte dianteira das asas na mesma linha de base (34 cm) em ambas células.

underwater [?]

1.4 Organização do trabalho

Dica: Apresente o que tem em cada capítulo.

Capítulo 2

Especificação do Projeto

Especificação do projeto.

2.1 Seção 1

Seção dentro de um capítulo.

2.2 Seção 2

Outra seção dentro do capítulo.

Capítulo 3

Visão Estereoscópica - *Stereo Imaging*

Dica: Embasamento Teórico ou Fundamentação Teórica: revisão da literatura dos tópicos que sustentam a ciência e o conhecimento, relativos aos objetivos e aos métodos escolhidos para o desenvolvimento do trabalho; - Itens como Considerações Iniciais e Finais não são obrigatórios, mas completam muito bem qualquer capítulo.

A visão estereoscópica possibilita uma boa identificação de um espaço tridimensional, visto que sua estrutura permite a triangulação de pontos chaves, assim determinando-se 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. Computacionalmente, este processo pode ser realizado em quatro etapas: retificação, Nas próximas seções encontram-se apresentadas o modelamento matemático, a triangulação, a calibração, a retificação e a reprojeção tridimensional para este tipo de sistema.

3.1 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 câmeras 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 camera 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 [?].

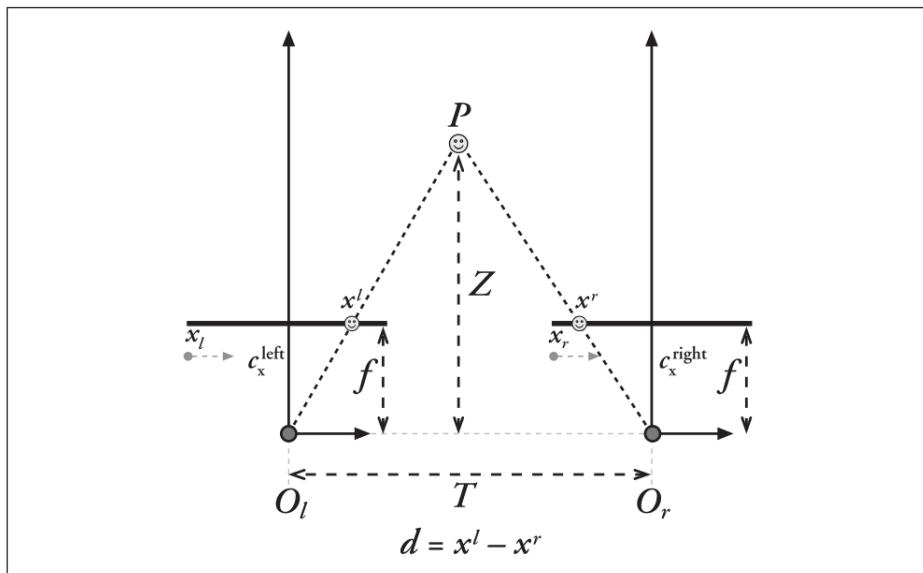


Figura 3.1: Modelo Idealizado de um sistema de Visão Estéreo

Referênciar esta imagem à figura 12-4 do livro do Bradski

A Figure 12-4. With a perfectly undistorted, aligned stereo rig and known correspondence, the depth Z can be found by similar triangles; the principal rays of the imagers begin at the centers of projection O_l and O_r and extend through the principal points of the two image planes at c_l and c_r

3.2 Geometria epipolar - *Epipolar Geometry*

3.3 Calibração *Stereo Calibration*

3.3.1 Parâmetros intrínsecos

3.3.2 Parâmetros extrinsecos

3.4 Retificação de Imagens - *Stereo Rectification*

3.5 Correspondência Estéreo - *Stereo Correspondence*

3.6 Mapa de disparidades - *Disparity Map*

3.7 Projeção Tridimensional - *3D Reprojection*

3.8 Reconhecimento de Objeto - *Object Recognition*

3.8.1 Segmentação - *Image Segmentation*

3.8.2 Identificação - *Object Identification*

Capítulo 4

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.

4.1 Materiais

Com relação aos equipamentos é possível classificá-los em três grupos distintos: Câmeras estereoscópicas, Unidades de Processamento, e Equipamentos auxiliares.

4.1.1 Câmeras estereoscópicas

O projeto já utilizou duas câmeras estereoscópicas. Primeiramente, utilizou-se a webcam Minoru(veja figura 4.1), 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 4.1 apresenta as espeficações da webcam.

Tabela 4.1: Especificações - 3D Webcam Minoru

Sensor de Imagem	VGA CMOS Sensor
Resolução Máxima	800 × 600
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 4.1: 3D Webcam Minoru

Atualmente, a câmera utilizada é uma câmera digital 3D W3 fabricada pela Fujifilm(veja figura) 4.2. 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 câmera com maior resolução e que apresentasse lentes com baixa distorção. Entretanto, essa câmera não apresenta streaming via USB, assim é necessário que os vídeos sejam processados *of-line*. 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 4.2 apresenta as especificações da câmera.

Tabela 4.2: Especificações - Câmera Digital Fujifilm FinePix Real 3D W3

Sensor de Imagem	10 MP CCD Sensor
Resolução Máxima	1280 × 720
Tamanho Linha de Base	7.5 cm
Taxa de Captura	24 - 30 fps
Distância Focal	60 cm até ∞
Peso	250g

Tabela 4.3: Especificações - BeagleBone Black

Processador	1GHz TI Sitara AM3359 ARM Cortex-A8
RAM	512 MB DDR3L @ 400 MHz
Armanezamento	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 4.2: Câmera Digital Fujifilm FinePix Real 3D W3

4.1.2 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. Abaixo estão apresentadas as plataformas que serão utilizadas para este propósito.

A primeira plataforma a ser utilizada e estudada é a plataforma aberta BeagleBone Black. Esta plataforma foi escolhida devido ao seu tamanho reduzido, podendo ser facilmente embarcado, e ao seu poder de processamento, utiliza Cortex-A8 operando à 1 GHz. A tabela 4.3 apresenta as especificações da plataforma.

Tabela 4.4: 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



Figura 4.3: Plataforma de Desenvolvimento - BeagleBone Black

A segunda plataforma a ser utilizada e estudada é a plataforma Jetson TK1 produzida pela NVIDIA. 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 4.4 apresenta as especificações da plataforma.



Figura 4.4: Plataforma de Desenvolvimento - Jetson TK1

4.1.3 Equipamentos auxiliares

Abaixo 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 4.6, 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.

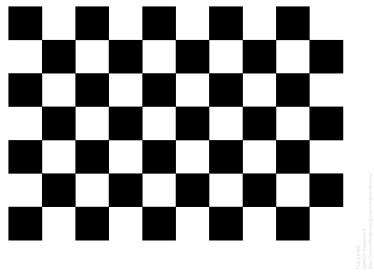


Figura 4.5: Padrão de Calibração

A motivação deste trabalho é a sua utilização em veículos áereos. 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 estereoscópica e para a plataforma embarcada. Como pode ser observado pela figura 4.5, todas as peças foram produzidas utilizando impressora 3D.

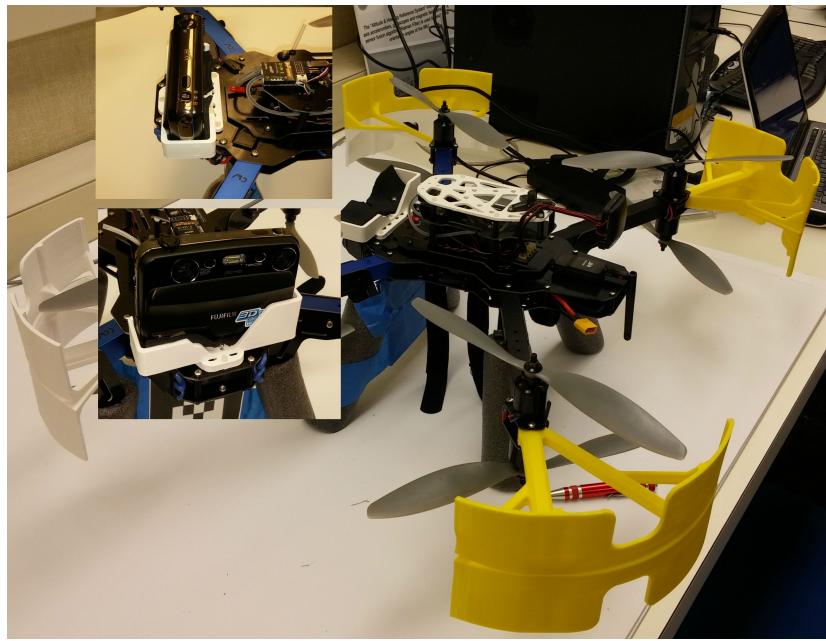


Figura 4.6: Quadricóptero 3DR X8 com suporte para a Câmera 3D

4.2 Métodos

Métodos utilizados no projeto.

Explicar as abordagens utilizando BM e SGBM

Capítulo 5

Resultados Parciais

Os resultados obtidos estão inteiramente relacionados ao desenvolvimento do programa e o algoritmo para a detecção de objetos e obstáculos. Nesta seção, serão apresentados os resultados desse *software* obtidos até então.

O software desenvolvido apresenta uma interface gráfica(GUI – *Graphical User Interface*) amigável, a qual facilita a visualização das imagens obtidas pela câmera estereoscópica e das imagens resultantes dos método de processamento de imagens desenvolvido. Atualmente, o software conta 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 5.1 ilustra o comporta-

mento do software quando essa opção é selecionada.

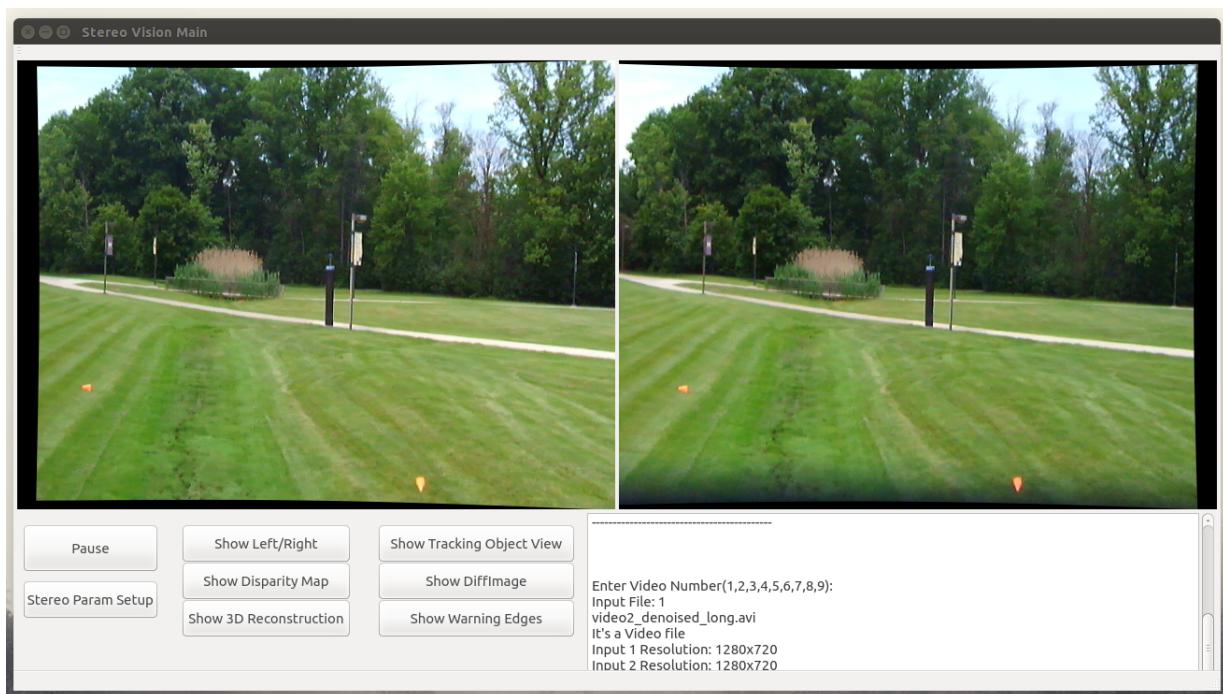


Figura 5.1: Interface Gráfica - Visualização simultânea dos quadros das câmeras esquerda e direita

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 5.2 ilustra o comportamento do software quando essa opção é selecionada.

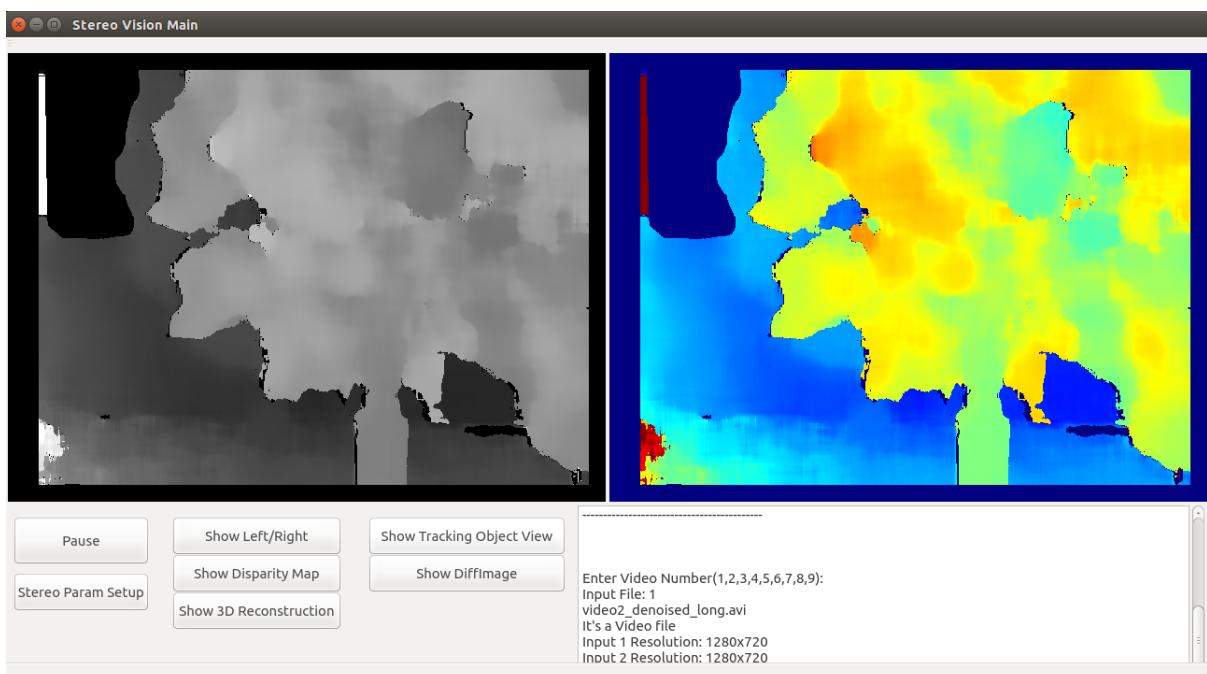


Figura 5.2: Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB

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 5.3 ilustra o comportamento do software quando essa opção é selecionada.

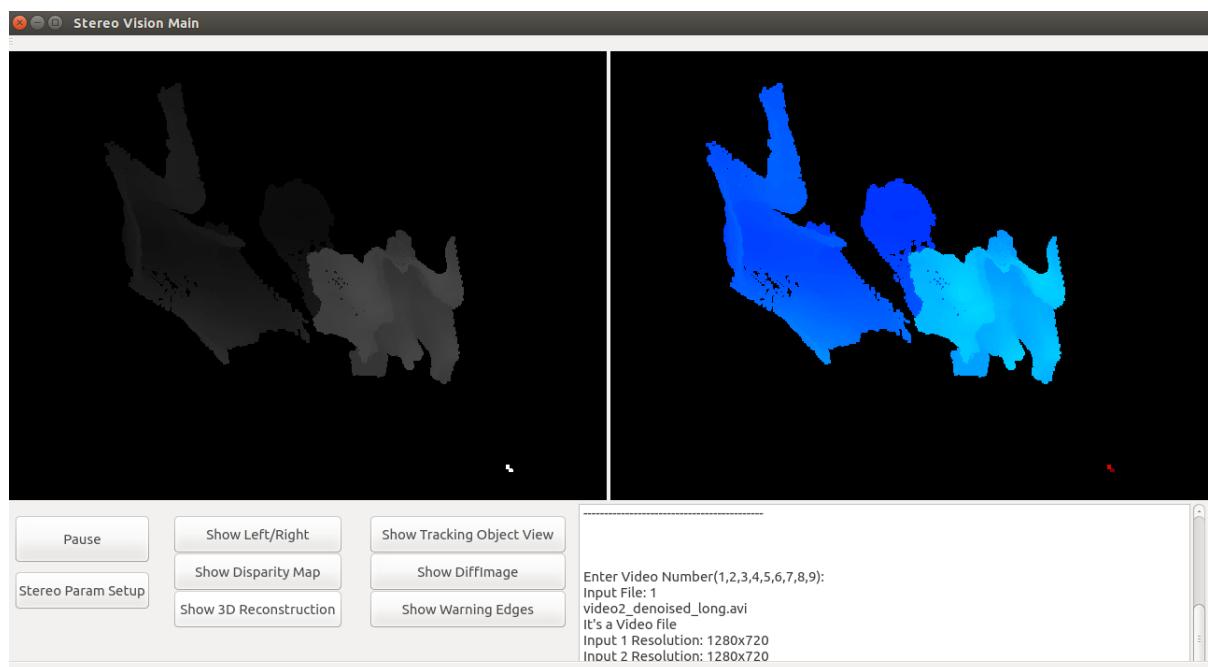


Figura 5.3: Interface Gráfica - Visualização dos Mapa de disparidade em Escala de Cinza e RGB

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 5.4 ilustra o comportamento do software quando essa opção é selecionada.

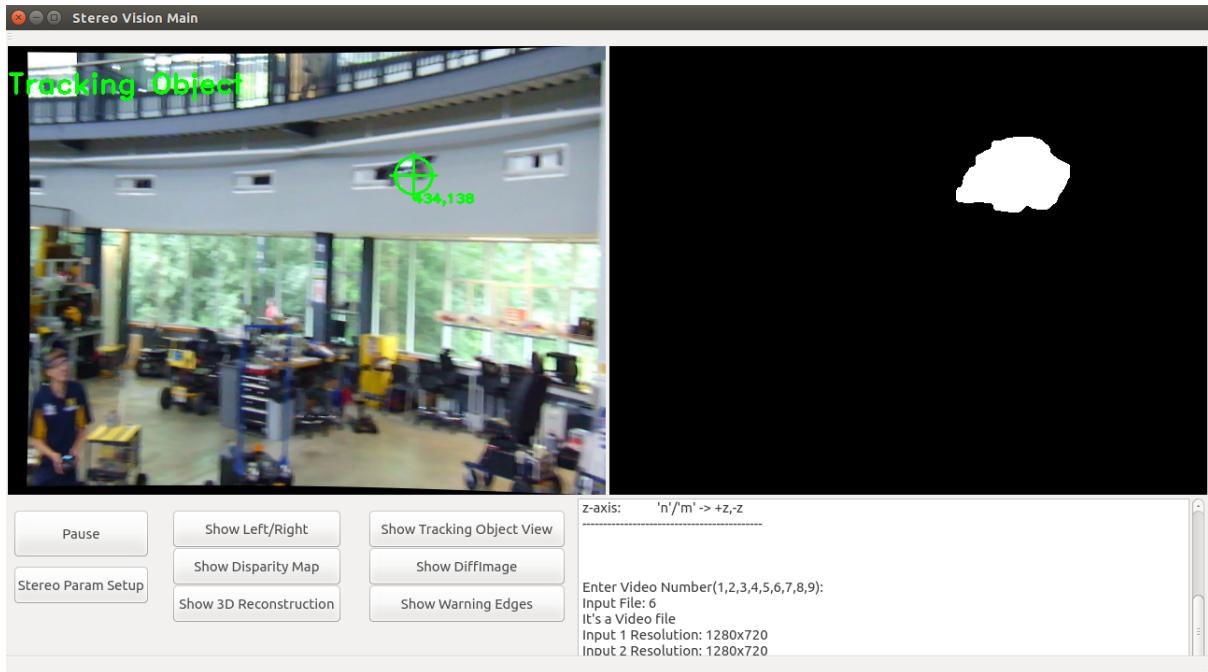


Figura 5.4: Interface Gráfica - Visualização da Imagem da Câmera Esquerda com o indicador de objeto rastreado e da Imagem binária resultante da limiarização por distância

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 5.5 ilustra o comportamento do software quando essa opção é selecionada.

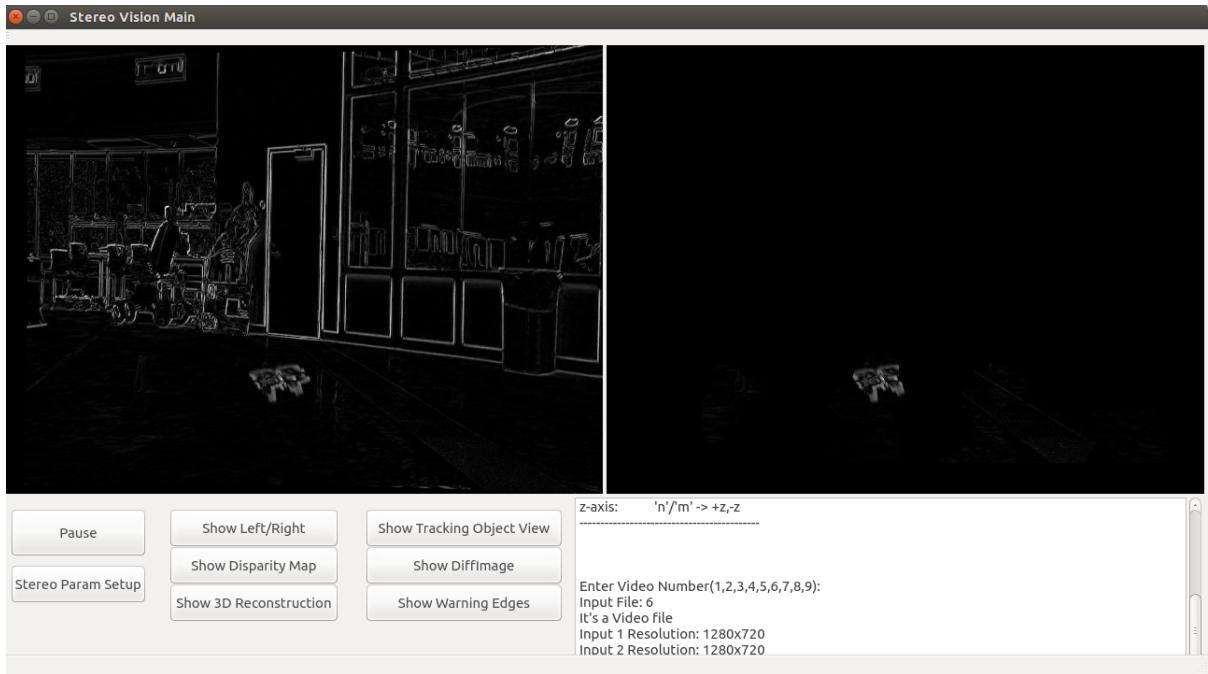


Figura 5.5: Interface Gráfica - Visualização da imagem resultante do processo de detecção de movimentos e imagem resultante do processo de detecção de movimentos limiarizada por distância

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 5.6 ilustra o comportamento do software quando essa opção é selecionada.

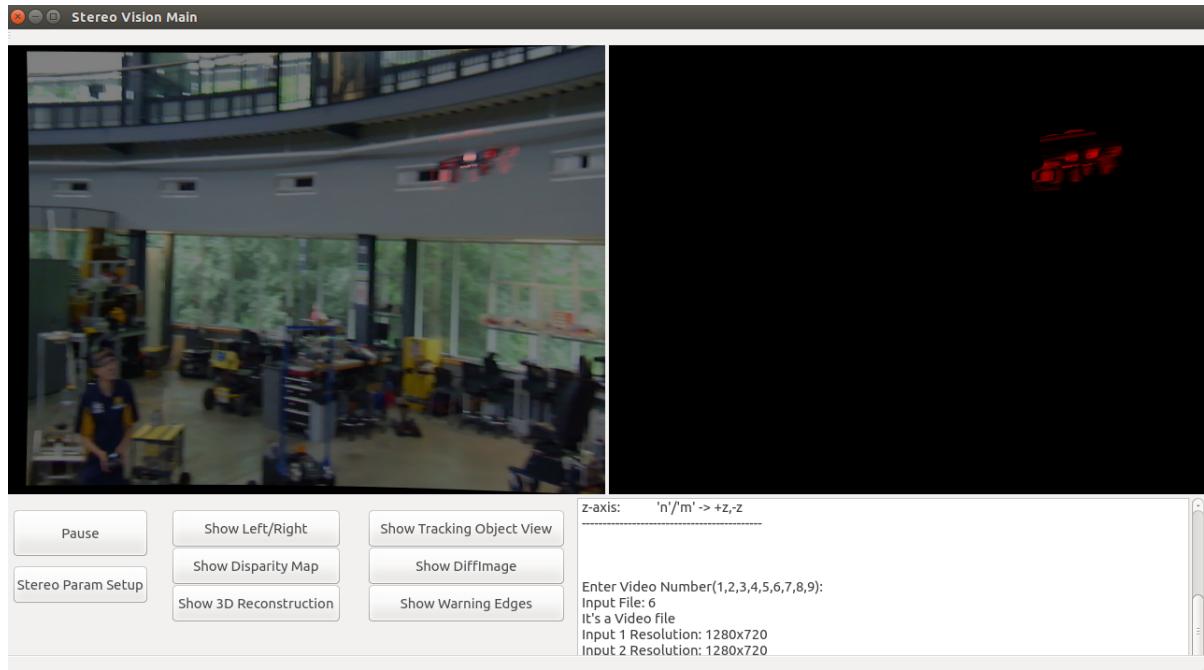


Figura 5.6: Interface Gráfica - Visualização da Imagem resultante da adição da imagem à direita com a Imagem da Câmera Esquerda e Imagem resultante do processo de realce das bordas dos objetos em movimento próximos ao veículo

Capítulo 6

Conclusão

Dica: Conclusões: "fecha" com os objetivos? (respondem aos objetivos?) - aqui é que "se vende o peixe- elas é que valorizam (ou não) o trabalho realizado. Normalmente é uma parte do trabalho "um pouco desprezada", pois o autor já está "cansado....". Mas é aqui que realmente se mede se o trabalho tem ou não valor. - Contém o item Trabalhos futuros, que é uma orientação sobre as possibilidades de continuação do desenvolvimento do trabalho.

Apêndice A

Apêndice 1

Abaixo estão apresentados todos os trechos de código desenvolvido para a interface gráfica implementada para a estação-base. Essa seção está subdividada em duas partes, bibliotecas e código-fonte.

A.1 Interface Gráfica - StereoVision System

A.1.1 Bibliotecas

mainwindow.h

```
/*
 * mainwindow.h
 *
 * Created on: Oct 1, 2015
 * Author: nicolasrosa
 */
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

/* Libraries */
#include <QMainWindow>
#include <opencv2/opencv.hpp>
```

```

/* Custom Libraries */
#include "StereoProcessor.h"
#include "setstereoparams.h"

using namespace cv;

namespace Ui{
    class MainWindow;
}

class MainWindow : public QMainWindow{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    void StereoVisionProcessInit();
    void printHelp();
    void openStereoSource(int inputNum);
    void createTrackbars();
    QImage putImage(const Mat& mat);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    StereoProcessor *stereo;
    SetStereoParams *stereoParamsSetupWindow;

    QImage qimageL, qimageR;

    QTimer* tmrTimer;

public slots:
    void StereoVisionProcessAndUpdateGUI();
}

```

```

private slots :
    void on_btnPauseOrResume_clicked();
    void on_btnShowDisparityMap_clicked();
    void on_btnShowStereoParamSetup_clicked();
    void on_btnShow3DReconstruction_clicked();
    void on_btnShowInputImages_clicked();
    void on_btnShowTrackingObjectView_clicked();
    void on_btnShowDiffImage_clicked();
    void on_btnShowDiffImage_2_clicked();
};

#endif // MAINWINDOW_H

```

reprojectImageTo3D.h

```

/*
 * reprojectImageTo3D.h
 *
 * Created on: Jun 18, 2015
 * Author: nicolasrosa
 */

```

```

#ifndef reproject_Image_To_3D_LIB_H_
#define reproject_Image_To_3D_LIB_H_

```

```

/* Libraries */
#include <opencv2/opencv.hpp>
#include <fstream>

```

```

using namespace cv;

```

```

/* Calibration */
#define RESOLUTION_640x480

```

```

//#define RESOLUTION_1280x720
#define CALIBRATION_ON

/* Functions Scope */
void on_trackbar(int ,void *);

void imageProcessing1(Mat img , Mat imgMedian , Mat imgMedianBGR );
void imageProcessing2(Mat src , Mat imgE , Mat imgED,Mat cameraFeedL ,bool

void resizeFrames(Mat* frame1 ,Mat* frame2 );
void changeResolution(VideoCapture* cap_l ,VideoCapture* cap_r );
void contrast_and_brightness(Mat &left ,Mat &right ,float alpha ,float beta

/* Global Variables */
bool isVideoFile=false ;
bool isImageFile=false ;
bool needCalibration=false ;
bool isStereoParamSetupTrackbarsCreated=false ;
bool isTrackingObjects=true ;

#endif /* reproject_Image_To_3D_LIB_H_ */

setstereoparams.h

/*
* setstereoparams.h
*
* Created on: Oct 1, 2015
* Author: nicolasrosa
*/
#ifndef SETSTEREOPARAMS_H
#define SETSTEREOPARAMS_H

```

```

#include <QDialog>

class StereoProcessor; // forward-declaration

namespace Ui {
    class SetStereoParams;
}

class SetStereoParams : public QDialog{
    Q_OBJECT

public:
    explicit SetStereoParams(QWidget *parent = 0, StereoProcessor *stereoProcessor = 0);
    void loadStereoParamsUi(int preFilterSize, int preFilterCap, int SADWindowSize, int disp12MaxDiff);

    ~SetStereoParams();

    bool isAlreadyShowing;

signals:
    void valuesChanged(int preFilterSize, int preFilterCap, int sadWindowSize);

private slots:
    /* Sliders */
    void on_preFilterSize_slider_valueChanged(int value);
    void on_preFilterCap_slider_valueChanged(int value);
    void on_SADWindowSize_slider_valueChanged(int value);
    void on_minDisparity_slider_valueChanged(int value);
    void on_numberOfDisparities_slider_valueChanged(int value);
    void on_textureThreshold_slider_valueChanged(int value);
    void on_uniquenessRatio_slider_valueChanged(int value);
    void on_speckleWindowSize_slider_valueChanged(int value);
    void on_speckleRange_slider_valueChanged(int value);
    void on_disp12MaxDiff_slider_valueChanged(int value);
}

```

```

/* SpinBoxes */
void on_preFilterSize_spinBox_valueChanged(int value);
void on_preFilterCap_spinBox_valueChanged(int value);
void on_SADWindowSize_spinBox_valueChanged(int value);
void on_minDisparity_spinBox_valueChanged(int value);
void on_numberOfDisparities_spinBox_valueChanged(int value);
void on_textureThreshold_spinBox_valueChanged(int value);
void on_uniquenessRatio_spinBox_valueChanged(int value);
void on_speckleWindowSize_spinBox_valueChanged(int value);
void on_speckleRange_spinBox_valueChanged(int value);
void on_disp12MaxDiff_spinBox_valueChanged(int value);

void on_buttonBox_accepted();
void on_buttonBox_rejected();

private:
Ui::SetStereoParams *ui;
StereoProcessor *stereo;

void updateValues();

};

#endif // SETSTEREOPARAMS_H

StereoCalib.h

/*
* StereoCalib.h
*
* Created on: Dec 3, 2015
* Author: nicolasrosa
*/

```

```

#ifndef STEREOCALIB_H
#define STEREOCALIB_H

/* Libraries */
#include <string>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

class StereoCalib{
public:
    StereoCalib(); // Constructor
    void readQMatrix();
    void calculateQMatrix();
    void createKMatrix();

    string intrinsicsFileName;
    string extrinsicsFileName;
    string QmatrixFileName;
    string StereoParamFileName;

    Point2d imageCenter;

    Mat K,Q;
    double focalLength;
    double baseline;
    bool is320x240;
    bool is640x480;
    bool is1280x720;

    Mat M1,D1,M2,D2;
    Mat R,T,R1,P1,R2,P2;
}

```

```

    Rect roi1, roi2;
    bool isKcreated;
};

#endif // STEREOCALIB_H

```

StereoConfig.h

```

/*
 * StereoConfig.h
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */

```

```

#ifndef STEREOCONFIG_H
#define STEREOCONFIG_H

class StereoConfig {
public:
    StereoConfig(); // Constructor
    // StereoConfig getConfig();

    int preFilterSize;
    int preFilterCap;
    int SADWindowSize;
    int minDisparity;
    int numberOfDisparities;
    int textureThreshold;
    int uniquenessRatio;
    int speckleWindowSize;
    int speckleRange;
    int disp12MaxDiff;
};

```

```
#endif // STEREOCONFIG_H
```

StereoCustom.h

```
/*
 * StereoCustom.h
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */
```

```
#ifndef STEREOCUSTOM_H
```

```
#define STEREOCUSTOM_H
```

```
/* Libraries */
```

```
#include <string>
```

```
using namespace std;
```

```
/* Global Variables */
```

```
const string trackbarWindowName = "Stereo_Param_Setup";
```

```
//bool isVideoFile=false , isImageFile=false , needCalibration=false , isStereo
```

```
/* Threshold , Erosion , Dilation and Blur Constants */
```

```
#define THRESH_VALUE 100
```

```
#define EROSION_SIZE 5
```

```
#define DILATION_SIZE 5
```

```
#define BLUR_SIZE 3
```

```
/* Trackbars Variables
```

```
* Initial min and max BM Parameters values. These will be changed using
*/
```

```
const int preFilterSize_MAX = 100;
```

```

const int preFilterCap_MAX = 100;
const int SADWindowSize_MAX = 100;
const int minDisparity_MAX = 100;
const int numberOfDisparities_MAX = 16;
const int textureThreshold_MAX = 100;
const int uniquenessRatio_MAX = 100;
const int speckleWindowSize_MAX = 100;
const int speckleRange_MAX = 100;
const int disp12MaxDiff_MAX = 1;

```

#endif // STEREOCUSTOM_H

StereoDiff.h

```

/*
 * StereoDiff.h
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */

```

```

#ifndef STEREODIFF_H
#define STEREODIFF_H

/* Libraries */
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

class StereoDiff{
public:
    StereoDiff(); // Constructor
    void createDiffImage(Mat, Mat);
}

```

```

void createResAND(Mat , Mat );
void convertToBGR ();
void addRedLines ();

bool StartDiff ;
Mat diffImage ;

Mat res_AND ;
Mat imageL ;
Mat res_AND_BGR ;
Mat res_AND_BGR_channels [3];

double alpha ;
double beta ;
Mat res_ADD ;
};

#endif // STEREODIFF_H

```

StereoDisparityMap.h

```

/*
 * StereoDisparityMap.h
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */

```

```

#ifndef STEREODISPARITYMAP_H
#define STEREODISPARITYMAP_H

/* Libraries */
#include <opencv2/opencv.hpp>

```

```

using namespace cv;
using namespace std;

class StereoDisparityMap {
public:
    StereoDisparityMap(); // Constructor

    Mat disp_16S;
    Mat disp_8U;
    Mat disp_BGR;
};

#endif // STEREODISPARITYMAP_H

```

StereoFlags.h

```

/*
 * StereoFlags.h
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */

```

```

#ifndef STEREOFLAGS_H
#define STEREOFLAGS_H

class StereoFlags {
public:
    StereoFlags(); // Constructor

    bool showInputImages;
    bool showXYZ;
    bool showStereoParam;
    bool showStereoParamValues;
}

```

```

bool showFPS;
bool showDisparityMap;
bool show3Dreconstruction;
bool showTrackingObjectView;
bool showDiffImage;
bool showWarningLines;
};

#endif // STEREOFFLAGS_H

```

StereoProcessor.h

```

/*
 * StereoProcessor.h
 *
 * Created on: Oct 20, 2015
 * Author: nicolasrosa
 */

```

```

#ifndef STEREOPROCESSOR_H
#define STEREOPROCESSOR_H

/* Libraries */
#include <opencv2/opencv.hpp>

/* Custom Libraries */
#include "StereoCalib.h"
#include "StereoCustom.h"
#include "StereoConfig.h"
#include "StereoDiff.h"
#include "StereoDisparityMap.h"
#include "StereoFlags.h"

#include "3DReconstruction.h"

```

```

using namespace cv;
using namespace std;

class StereoProcessor : public StereoConfig {
public:
    StereoProcessor(int inputNum); // Constructor
    int getInputNum();

    void readConfigFile();
    void readStereoConfigFile();

    void stereoInit();
    void stereoCalib();
    void setStereoParams();
    void setValues(int preFilterSize, int preFilterCap, int sadWindowSize);

    void imageProcessing(Mat src, Mat imgE, Mat imgED, Mat trackingView, b

    void saveLastFrames();

    Mat imageL[2], imageR[2];
    Mat imageL_grey[2], imageR_grey[2];
    VideoCapture capL, capR;

    Ptr<StereoBM> bm;
    StereoCalib calib;
    StereoConfig stereocfg;
    StereoDisparityMap disp;
    Reconstruction3D view3D;
    StereoDiff diff;
    StereoFlags flags;
    Size imageSize;
}

```

```

int numRows;

/* Results */
Mat imgThreshold;
Mat trackingView;

bool showStereoParamsValues;

private:
    int inputNum;
};

#endif // STEREOPROCESSOR_H

```

A.1.2 Códigos-Fonte

main.cpp

```

#include "mainwindow.h"

#include <QApplication>

//#include <QHBoxLayout>
//#include <QSlider>
//#include <QSpinBox>

int main(int argc, char *argv[]){
    QApplication app(argc, argv);
    MainWindow mainwindow;

    mainwindow.show();

//     QWidget *window = new QWidget;
//     window->setWindowTitle("Enter Your Age");

```

```

//      QSpinBox *spinBox = new QSpinBox;
//      QSlider *slider = new QSlider(Qt::Horizontal);
//      spinBox->setRange(0, 130);
//      slider->setRange(0, 130);

//      QObject::connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
//      QObject::connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));
//      spinBox->setValue(35);

//      QHBoxLayout *layout = new QHBoxLayout;

//      layout->addWidget(spinBox);
//      layout->addWidget(slider);
//      window->setLayout(layout);

//      window->show();

return app.exec();
}

```

mainwindow.cpp

```

/* Project: reprojectImageTo3D - BlockMatching Algorithm
 * mainwindow.cpp
 *
 * Created on: June, 2015
 * Author: nicolasrosa
 *
 * // Credits: http://opencv.jp/opencv2-x-samples/point-cloud-rendering
 * // Credits: Kyle Hounslow - https://www.youtube.com/watch?v=bSeFrPrqZ2I
 */

/* Libraries */

```

```

#include <QtCore>
#include <opencv2/imgproc/imgproc.hpp>

/* Custom Libraries */
#include "reprojectImageTo3D.h"
#include "mainwindow.h"
#include "ui_mainwindow.h"

using namespace cv;
using namespace std;

void writeMatToFile(cv::Mat& m, const char* filename);

//MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new Ui::MainWindow):
MainWindow::MainWindow(QWidget *parent): QMainWindow(parent), ui(new Ui::MainWindow){
    ui->setupUi(this);

    this->stereo = new StereoProcessor(1);
    StereoVisionProcessInit();

    tmrTimer = new QTimer(this);
    connect(tmrTimer, SIGNAL(timeout()), this, SLOT(StereoVisionProcessAndUpdateUI()));
    tmrTimer->start(20);
}

MainWindow::~MainWindow(){
    delete ui;
}

void MainWindow::on_btnPauseOrResume_clicked(){
    if(tmrTimer->isActive() == true){
        tmrTimer->stop();
        cout << "Paused!" << endl;
    }
}

```

```

    ui->btnPauseOrResume->setText( "Resume" );
} else {
    tmrTimer->start( 20 );
    cout << "Resumed!" << endl;
    ui->btnPauseOrResume->setText( "Pause" );
}
}

void MainWindow::on_btnShowStereoParamSetup_clicked(){
    stereoParamsSetupWindow = new SetStereoParams( this , stereo );

    cout << "[ Stereo_Param_Setup ]_Stereo_Parameters_Configuration_Loaded"
    this->stereoParamsSetupWindow->loadStereoParamsUi( stereo->stereocfg .)
                                                stereo->stereocfg .)

    stereoParamsSetupWindow->show();
}

void MainWindow::StereoVisionProcessInit(){
    cerr << "Arrumar_a_Matrix_K,_os_valores_das_últimas_colunas_estão_errados"
    cerr << "Arrumar_a_função_StereoProcessor::calculateQMatrix()." << endl;
    cerr << "Arrumar_o_Constructor_da_classe_StereoDisparityMap_para_Algoritmo"
    cerr << "Arrumar_o_tipo_de_execução_da_Stereo_Param_Setup,_fazer_com_o"
    cerr << "Arrumar_a_funcionalidade_do_Botão_Pause/Resume,_não_está_funcionando"

    printHelp();
}

```

```

//(1) Open Image Source
openStereoSource(stereo->getInputNum());
stereo->readConfigFile();
stereo->readStereoConfigFile();

//(2) Camera Setting

// Checking Resolution
stereo->calib.is320x240 = false;
stereo->calib.is640x480 = true;
stereo->calib.is1280x720 = false;

if(isVideoFile){
    stereo->imageSize.width = stereo->capL.get(CV_CAP_PROP_FRAME_WIDTH);
    stereo->imageSize.height = stereo->capL.get(CV_CAP_PROP_FRAME_HEIGHT);
} else{
    stereo->imageSize.width = stereo->imageL[0].cols;
    stereo->imageSize.height = stereo->imageL[0].rows;
}

if(stereo->imageSize.width==0 && stereo->imageSize.height==0){
    cerr << "Number_of_Cols_and_Number_of_Rows_equal_to_ZERO!" << endl;
} else{
    cout << "Input_Resolution(Width,Height): (" << stereo->imageSize.width
        << "," << stereo->imageSize.height << ")";
}

//(3) Stereo Initialization
stereo->bm = StereoBM::create(16,9);
stereo->stereoInit();

//(4) Stereo Calibration
if(needCalibration){

```

```

cout << "Calibration:_ON" << endl;
stereo->stereoCalib ();

// Compute the Q Matrix
stereo->calib.readQMatrix(); // true=640x480 false=others

//Point2d imageCenter = Point2d((imageL[0].cols - 1.0)/2.0,(imageL[0].rows - 1.0)/2.0);
//calculateQMatrix();

// Compute the K Matrix
////// // Checking Intrinsic Matrix
////// if(stereo->calib.isKcreated){
//////     cout << "The Intrinsic Matrix is already Created." << endl;
////// } else {
//         //createKMatrix();
//     }
stereo->calib.createKMatrix();

} else {
    cout << "Calibration:_OFF" << endl << endl;
    cerr << "Warning:_Couldn't generate_3D_Reconstruction._Please ,_check your camera parameters." << endl;

//stereo->readQMatrix(); // true=640x480 false=others
//stereo->createKMatrix();
}

// Setting StereoBM Parameters
stereo->setStereoParams();

//(5) Point Cloud Initialization
stereo->view3D.PointCloudInit(stereo->calib.baseline/10,true);

stereo->view3D.setViewPoint(20.0,20.0,-stereo->calib.baseline*10);

```

```

stereo->view3D.setLookAtPoint(22.0,16.0,stereo->calib.baseline*10.0)

}

void MainWindow::StereoVisionProcessAndUpdateGUI(){
    // Local Variables
    char key=0;

    // Timing
    int frameCounter=0;
    int fps ,lastTime = clock();

    // (6) Rendering Loop
    while(key != 'q'){
        if(isVideoFile){
            stereo->capL >> stereo->imageL[0];
            stereo->capR >> stereo->imageR[0];

            resizeFrames(&stereo->imageL[0],&stereo->imageR[0]);

            if(needCalibration){
                stereo->imageSize = stereo->imageL[0].size();
                stereoRectify(stereo->calib.M1,stereo->calib.D1,stereo->
                Mat rmap[2][2];

                initUndistortRectifyMap(stereo->calib.M1, stereo->calib.I1,
                initUndistortRectifyMap(stereo->calib.M2, stereo->calib.I2,
                Mat imageLr, imageRr;
                remap(stereo->imageL[0], imageLr, rmap[0][0], rmap[0][1]);
                remap(stereo->imageR[0], imageRr, rmap[1][0], rmap[1][1]);

                stereo->imageL[0] = imageLr;
                stereo->imageR[0] = imageRr;
            }
        }
    }
}
```

```

    }

}

// Setting StereoBM Parameters
// stereo ->setStereoParams();

// Convert BGR to Grey Scale
cvtColor( stereo ->imageL[0] , stereo ->imageL_grey [0] ,CV_BGR2GRAY );
cvtColor( stereo ->imageR[0] , stereo ->imageR_grey [0] ,CV_BGR2GRAY );

stereo ->bm->compute( stereo ->imageL_grey [0] , stereo ->imageR_grey [0]
//fillOcclusion( disp ,16 ,false );

normalize( stereo ->disp . disp_16S , stereo ->disp . disp_8U , 0 , 255 , CV_
applyColorMap( stereo ->disp . disp_8U , stereo ->disp . disp_BGR , COLORMAP_HSV);

/* Image Processing */
Mat disp_8Ueroded ;Mat disp_8U_eroded_dilated ;

stereo ->imageProcessing( stereo ->disp . disp_8U , disp_8Ueroded , disp_8U_ero

//(7) Projecting 3D point cloud to image
if( stereo ->flags . show3Dreconstruction){
    cv :: reprojectImageTo3D( stereo ->disp . disp_16S , stereo ->view3D . depth );
    Mat xyz = stereo ->view3D . depth . reshape (3 , stereo ->view3D . depth . rows * stereo ->view3D . depth . cols );
    xyz . t . at <double>(0 ,0 )= stereo ->view3D . viewpoint . x;
    xyz . t . at <double>(1 ,0 )= stereo ->view3D . viewpoint . y;
    xyz . t . at <double>(2 ,0 )= stereo ->view3D . viewpoint . z;

    if( stereo ->flags . showXYZ){
        //cout<< stereo ->view3D . t << endl ;
    }
}

```

```

    cout << "x:_" << stereo->view3D.t.at<double>(0,0) << endl
    cout << "y:_" << stereo->view3D.t.at<double>(1,0) << endl
    cout << "z:_" << stereo->view3D.t.at<double>(2,0) << endl
}

stereo->view3D.t=stereo->view3D.Rotation*stereo->view3D.t;

// projectImagefromXYZ( imageL[0], disp3Dviewer , disp , disp3D , xyz )
stereo->view3D.projectImagefromXYZ( stereo->disp . disp_BGR , ster

// GUI Output
stereo->view3D.disp3D.convertTo( stereo->view3D.disp3D_8U , CV_8U );
//imshow( "3D Depth" , disp3D );
//imshow( "3D Viewer" , disp3Dviewer );
//imshow( "3D Depth RGB" , disp3DBGR );

QImage qimageL = putImage( stereo->view3D.disp3D_8U );
QImage qimageR = putImage( stereo->view3D.disp3D_BGR );

ui->lblOriginalLeft->setPixmap( QPixmap::fromImage( qimageL ) );
ui->lblOriginalRight->setPixmap( QPixmap::fromImage( qimageR ) )
}

//(8) Movement Difference between Frames
// if( stereo->diff . StartDiff ){
if( stereo->flags . showDiffImage || stereo->flags . showWarningLines )
    if( stereo->diff . StartDiff ){
        stereo->diff . createDiffImage( stereo->imageL_grey[0] , stereo->imageL_grey[1] );
        stereo->diff . createRes( stereo->diff . diffImage , stereo->diff . res );
        stereo->diff . convertToBGR();
        stereo->imageL[0].copyTo( stereo->diff . imageL );
        stereo->diff . copyTo( stereo->diff . imageR );
    }
}

```

```

    stereo ->diff . addRedLines ();

    //imshow( " imgThreshold ", stereo ->imgThreshold );
    //imshow( " DiffImage ", stereo ->diff . diffImage );
    //imshow( " Bitwise_AND ", stereo ->diff . res_AND );

}

stereo ->saveLastFrames ();
} else {
    stereo ->saveLastFrames ();
    stereo ->diff . StartDiff = 1;
}
}

//(9) OpenCV and GUI Output
if ( stereo ->flags . showInputImages ){

    QImage qimageL = putImage( stereo ->imageL [ 0 ] );
    QImage qimageR = putImage( stereo ->imageR [ 0 ] );

    ui ->lblOriginalLeft ->setPixmap ( QPixmap :: fromImage ( qimageL ) );
    ui ->lblOriginalRight ->setPixmap ( QPixmap :: fromImage ( qimageR ) )
}

if ( stereo ->flags . showDisparityMap ){

    QImage qimageL = putImage( stereo ->disp . disp_8U );
    QImage qimageR = putImage( stereo ->disp . disp_BGR );

    ui ->lblOriginalLeft ->setPixmap ( QPixmap :: fromImage ( qimageL ) );
    ui ->lblOriginalRight ->setPixmap ( QPixmap :: fromImage ( qimageR ) )
}

if ( stereo ->flags . showTrackingObjectView ){

    QImage qimageL = putImage( stereo ->trackingView );
}

```

```

QImage qimageR = putImage( stereo->imgThreshold );

ui->lblOriginalLeft->setPixmap( QPixmap::fromImage( qimageL ) );
ui->lblOriginalRight->setPixmap( QPixmap::fromImage( qimageR ) )
}

if( stereo->flags . showDiffImage && stereo->diff . StartDiff ){
    this->qimageL = putImage( stereo->diff . diffImage );
    this->qimageR = putImage( stereo->diff . res_AND );

    ui->lblOriginalLeft->setPixmap( QPixmap::fromImage( qimageL ) );
    ui->lblOriginalRight->setPixmap( QPixmap::fromImage( qimageR ) )
}

if( stereo->flags . showWarningLines && stereo->diff . StartDiff ){
    this->qimageL = putImage( stereo->diff . res_ADD );
    this->qimageR = putImage( stereo->diff . res_AND_BGR );

    ui->lblOriginalLeft->setPixmap( QPixmap::fromImage( qimageL ) );
    ui->lblOriginalRight->setPixmap( QPixmap::fromImage( qimageR ) )
}

//      // if( showStereoParam && !isStereoParamSetupTrackbarsCreated )
//      if( showStereoParam && !isStereoParamSetupTrackbarsCreated )
//          isStereoParamSetupTrackbarsCreated=true ;
//          createTrackbars();
//          cout << "oi" << endl;
//      } else {
//          destroyWindow( trackbarWindowName );
//          isStereoParamSetupTrackbarsCreated=false ;
//      }

//(10) Shortcuts

```

```

key = waitKey(1);
if(key==' ')
    printHelp();
//           if(key=='1')
//               showInputImages = !showInputImages;
//           if(key=='2')
//               showDisparityMap = !showDisparityMap;
//           if(key=='3')
//               show3Dreconstruction = !show3Dreconstruction;
if(key=='4')
    stereo->flags.showXYZ = !stereo->flags.showXYZ;
if(key=='5')
    stereo->flags.showFPS = !stereo->flags.showFPS;
if(key=='6')
    stereo->flags.showStereoParamValues = !stereo->flags.showStereoParamValues;
if(key=='7')
    stereo->flags.showDiffImage = !stereo->flags.showDiffImage;

if(key=='f')
    stereo->view3D.isSub=stereo->view3D.isSub?false:true;
if(key=='h')
    stereo->view3D.viewpoint.x+=stereo->view3D.step;
if(key=='g')
    stereo->view3D.viewpoint.x-=stereo->view3D.step;
if(key=='l')
    stereo->view3D.viewpoint.y+=stereo->view3D.step;
if(key=='k')
    stereo->view3D.viewpoint.y-=stereo->view3D.step;
if(key=='n')
    stereo->view3D.viewpoint.z+=stereo->view3D.step;
if(key=='m')
    stereo->view3D.viewpoint.z-=stereo->view3D.step;

```

```

if(key=='q')
    break;

//(1) Video Loop - If the last frame is reached, reset the capture
frameCounter += 1;

if(frameCounter == stereo->capR.get(CV_CAP_PROP_FRAME_COUNT)){
    frameCounter = 0;
    stereo->capL.set(CV_CAP_PROP_POS_FRAMES, 0);
    stereo->capR.set(CV_CAP_PROP_POS_FRAMES, 0);
}

if(1){
    // if(stereo->flags.showFPS){
        // cout << "Frames: " << frameCounter << "/" << capR.get(CV_CAP_PROP_FRAME_COUNT);
        // cout << "Current time(s): " << current_time << endl;
        // cout << "FPS: " << (frameCounter/current_time) << endl;
        fps = (int) (1000/((clock() / 1000) - lastTime)); // time stuff
        lastTime = clock() / 1000;
        // cout << clock() << endl;
        cout << "FPS: " << fps << endl;
    }
}

cout << "END" << endl;

// return 0;
}

void MainWindow::printHelp(){
    // Console Output
    cout << "-----Help_Menu-----\n"
    << "Run_command_line: ./reprojectImageTo3D\n"
    << "Keys:\n"
}

```

```

<< " , ' _-\tShow_Help\n"
<< "'1' _-\tShow_L/R_Windows \t\t'4' _-\tShow_XYZ\n"
<< "'2' _-\tShow_Disparity_Map \t\t'5' _-\tShow_FPS\n"
<< "'3' _-\tShow_3D_Reconstruction \t\t'6' _-\tShow_Stereo_Parameters\n"
<< "\n3D_Viewer_Navigation :\n"
<< "x-axis :\t'g'/'h' _->_+x,-x\n"
<< "y-axis :\t'l'/'k' _->_+y,-y\n"
<< "z-axis :\t'n'/'m' _->_+z,-z\n"
<< "_________________________________\n"
<< "\n\n";

//GUI
ui->txtOutputBox ->appendPlainText
( QString("_________________________________Help_Menu_________________________________\n")+
QString("Run_command_line : ./reprojectImageTo3D\n")+
QString("Keys :\n")+
QString(" , ' _-\tShow_Help\n")+
QString(" '1' _-\tShow_L/R_Windows \t\t'4' _-\tShow_XYZ\n")+
QString(" '2' _-\tShow_Disparity_Map \t\t'5' _-\tShow_FPS\n")+
QString(" '3' _-\tShow_3D_Reconstruction \t\t'6' _-\tShow_Stereo_Parameters\n")
QString("\n3D_Viewer_Navigation :\n")+
QString("x-axis :\t'g'/'h' _->_+x,-x\n")+
QString("y-axis :\t'l'/'k' _->_+y,-y\n")+
QString("z-axis :\t'n'/'m' _->_+z,-z\n")+
QString("_________________________________\n")+
QString("\n\n"));
}

void MainWindow::openStereoSource(int inputNum){
    string imageL_filename;
    string imageR_filename;

    // Create an object that decodes the input Video stream.

```

```

cout << "Enter_Video_Number(1,2,3,4,5,6,7,8,9):" << endl;
ui->txtOutputBox->appendPlainText(QString("Enter_Video_Number(1,2,3,
//    scanf("%d",&inputNum);
cout << "Input_File :" << inputNum << endl;
ui->txtOutputBox->appendPlainText(QString("Input_File :") + QString::number(inputNum));
switch(inputNum){
case 1:
    imageL_filename = ".../.../workspace/data/video10_l.avi";
    imageR_filename = ".../.../workspace/data/video10_r.avi";
    needCalibration=true;
    //ui->txtOutputBox->appendPlainText(QString("video2_denoised_long"));
    break;
case 2:
    imageL_filename = ".../.../workspace/data/video12_l.avi";
    imageR_filename = ".../.../workspace/data/video12_r.avi";
    needCalibration=true;
    //ui->txtOutputBox->appendPlainText(QString("video0.avi"));
    break;
case 3:
    imageL_filename = ".../data/left/video1.avi";
    imageR_filename = ".../data/right/video1.avi";
    needCalibration=true;
    //ui->txtOutputBox->appendPlainText(QString("video1.avi"));
    break;
case 4:
    imageL_filename = ".../data/left/video2_noised.avi";
    imageR_filename = ".../data/right/video2_noised.avi";
    needCalibration=true;
    //ui->txtOutputBox->appendPlainText(QString("video2_noised.avi"));
    break;
case 5:
    imageL_filename = ".../data/left/20004.avi";
    imageR_filename = ".../data/right/30004.avi";
}

```

```

needCalibration=false;
break;

case 6:
    imageL_filename = ".../.../workspace/data/left/video15.avi";
    imageR_filename = ".../.../workspace/data/right/video15.avi";
    needCalibration=true;
    break;

case 7:
    imageL_filename = ".../.../workspace/data/left/left1.png";
    imageR_filename = ".../.../workspace/data/right/right1.png";
    needCalibration=false;
    break;

case 8:
    imageL_filename = ".../data/left/left2.png";
    imageR_filename = ".../data/right/right2.png";
    needCalibration=false;
    break;

case 9:
    imageL_filename = ".../data/left/left3.png";
    imageR_filename = ".../data/right/right3.png";
    needCalibration=false;
    break;

}

if(imageL_filename.substr(imageL_filename.find_last_of(".")+1) ==
   cout << "It's a Video file" << endl;
ui->txtOutputBox->appendPlainText(QString("It's a Video file"));
isVideoFile=true;

stereo->capL.open(imageL_filename);
stereo->capR.open(imageR_filename);

if(!stereo->capL.isOpened() || !stereo->capR.isOpened())){

```

```

    cerr << "Could_not_open_or_find_the_input_videos!" << endl
    ui->txtOutputBox->appendPlainText(QString( "Could_not_open_o
        // return -1;
    }

    cout << "Input_1_Resolution:" << stereo->capR.get(CV_CAP_PROP_FR
    cout << "Input_2_Resolution:" << stereo->capL.get(CV_CAP_PROP_FR
    ui->txtOutputBox->appendPlainText(QString("Input_1_Resolution:" )
    ui->txtOutputBox->appendPlainText(QString("Input_2_Resolution:" )

} else {
    cout << "It_is_not_a_Video_file" << endl;
    ui->txtOutputBox->appendPlainText(QString( "It_is_not_a_Video_file
    if(imageL_filename.substr(imageL_filename.find_last_of("." ) + 1)
        cout << "It's_a_Image_file" << endl;
        ui->txtOutputBox->appendPlainText(QString( "It's_a_Image_file
        isImageFile=true;

    stereo->imageL[0] = imread(imageL_filename, CV_LOAD_IMAGE_CO
    stereo->imageR[0] = imread(imageR_filename, CV_LOAD_IMAGE_CO

    if(!stereo->imageL[0].data || !stereo->imageR[0].data){
// Check for invalid input
        ui->txtOutputBox->appendPlainText(QString("Could_not_open
        return;
    }
} else {
    cout << "It_is_not_a_Image_file" << endl;
    ui->txtOutputBox->appendPlainText(QString( "It_is_not_a_Imag
}
}

void MainWindow::on_btnShowInputImages_clicked(){

```

```

this ->stereo ->flags . showInputImages = true ;
this ->stereo ->flags . showDisparityMap = false ;
this ->stereo ->flags . show3Dreconstruction = false ;
this ->stereo ->flags . showTrackingObjectView = false ;
this ->stereo ->flags . showDiffImage = false ;
this ->stereo ->flags . showWarningLines = false ;
}

void MainWindow :: on_btnShowDisparityMap_clicked(){
    this ->stereo ->flags . showInputImages = false ;
    this ->stereo ->flags . showDisparityMap = true ;
    this ->stereo ->flags . show3Dreconstruction = false ;
    this ->stereo ->flags . showTrackingObjectView = false ;
    this ->stereo ->flags . showDiffImage = false ;
    this ->stereo ->flags . showWarningLines = false ;
}

void MainWindow :: on_btnShow3DReconstruction_clicked(){
    this ->stereo ->flags . showInputImages = false ;
    this ->stereo ->flags . showDisparityMap = false ;
    this ->stereo ->flags . show3Dreconstruction = true ;
    this ->stereo ->flags . showTrackingObjectView = false ;
    this ->stereo ->flags . showDiffImage = false ;
    this ->stereo ->flags . showWarningLines = false ;
}

void MainWindow :: on_btnShowTrackingObjectView_clicked(){
    this ->stereo ->flags . showInputImages = false ;
    this ->stereo ->flags . showDisparityMap = false ;
    this ->stereo ->flags . show3Dreconstruction = false ;
    this ->stereo ->flags . showTrackingObjectView = true ;
    this ->stereo ->flags . showDiffImage = false ;
    this ->stereo ->flags . showWarningLines = false ;
}

```

}

```
void MainWindow::on_btnShowDiffImage_clicked(){
    this ->stereo ->flags . showInputImages = false ;
    this ->stereo ->flags . showDisparityMap = false ;
    this ->stereo ->flags . show3Dreconstruction = false ;
    this ->stereo ->flags . showTrackingObjectView = false ;
    this ->stereo ->flags . showDiffImage = true ;
    this ->stereo ->flags . showWarningLines = false ;
}
```

```
void MainWindow::on_btnShowDiffImage_2_clicked(){
    this ->stereo ->flags . showInputImages = false ;
    this ->stereo ->flags . showDisparityMap = false ;
    this ->stereo ->flags . show3Dreconstruction = false ;
    this ->stereo ->flags . showTrackingObjectView = false ;
    this ->stereo ->flags . showDiffImage = false ;
    this ->stereo ->flags . showWarningLines = true ;
}
```

```
QImage MainWindow::putImage( const Mat& mat){
    // 8-bits unsigned , NO. OF CHANNELS=1
    if(mat . type ()==CV_8UC1)
    {
        // Set the color table (used to translate colour indexes to qRgb
        QVector<QRgb> colorTable;
        for (int i=0; i<256; i++)
            colorTable . push_back(qRgb(i ,i ,i ));
        // Copy input Mat
        const uchar *qImageBuffer = (const uchar *)mat . data ;
        // Create QImage with same dimensions as input Mat
        QImage img(qImageBuffer , mat . cols , mat . rows , mat . step , QImage :: F
        img . setColorTable(colorTable);
```

```

return img;
}

// 8-bits unsigned , NO. OF CHANNELS=3
if( mat . type ()==CV_8UC3)
{
    // Copy input Mat
    const uchar *qImageBuffer = (const uchar*)mat . data ;
    // Create QImage with same dimensions as input Mat
    QImage img( qImageBuffer , mat . cols , mat . rows , mat . step , QImage :: F
    return img . rgbSwapped () ;

}
else
{
    qDebug () << "ERROR:_Mat_could_not_be_converted_to_QImage ." ;
    return QImage () ;
}

}

void writeMatToFile (cv :: Mat& m, const char* filename)
{
    ofstream fout (filename);

    if (!fout)
    {
        cout<<" File _Not _Opened "<< endl ; return ;
    }

    for (int i=0; i<m. rows ; i++)
    {
        for (int j=0; j<m. cols ; j++)
        {
            fout<<m. at<float>(i , j)<<" \t " ;
        }
    }
}

```

```

fout << endl ;
}

fout.close();
}

void MainWindow::createTrackbars(){ //Create Window for trackbars
char TrackbarName[50];

// Create TrackBars Window
namedWindow(trackbarWindowName ,0);

// Create memory to store Trackbar name on window
sprintf( TrackbarName , "preFilterSize");
sprintf( TrackbarName , "preFilterCap");
sprintf( TrackbarName , "SADWindowSize");
sprintf( TrackbarName , "minDisparity");
sprintf( TrackbarName , "numberOfDisparities");
sprintf( TrackbarName , "textureThreshold");
sprintf( TrackbarName , "uniquenessRatio");
sprintf( TrackbarName , "speckleWindowSize");
sprintf( TrackbarName , "speckleRange");
sprintf( TrackbarName , "disp12MaxDiff");

//Create Trackbars and insert them into window

createTrackbar( "preFilterSize" , trackbarWindowName , &this->stereo->
createTrackbar( "preFilterCap" , trackbarWindowName , &this->stereo->s
createTrackbar( "SADWindowSize" , trackbarWindowName , &this->stereo->
createTrackbar( "minDisparity" , trackbarWindowName , &this->stereo->s
createTrackbar( "numberOfDisparities" , trackbarWindowName , &this->ste
createTrackbar( "textureThreshold" , trackbarWindowName , &this->stereo->
```

```

        createTrackbar( "uniquenessRatio" , trackbarWindowName , &this->stereo ->
        createTrackbar( "speckleWindowSize" , trackbarWindowName , &this->stereo ->
        createTrackbar( "speckleRange" , trackbarWindowName , &this->stereo ->s
        createTrackbar( "disp12MaxDiff" , trackbarWindowName , &this->stereo ->
    }

void on_trackbar(int ,void*){ } ; //This function gets called whenever a tra

void resizeFrames(Mat* frame1 ,Mat* frame2){
    if(frame1->cols != 0 || !frame2->cols != 0){
#ifdef RESOLUTION_320x240
        resize(*frame1 , *frame1 , Size(320,240) , 0 , 0 , INTER_CUBIC);
        resize(*frame2 , *frame2 , Size(320,240) , 0 , 0 , INTER_CUBIC);
#endif

#ifdef RESOLUTION_640x480
        resize(*frame1 , *frame1 , Size(640,480) , 0 , 0 , INTER_CUBIC);
        resize(*frame2 , *frame2 , Size(640,480) , 0 , 0 , INTER_CUBIC);
#endif

#ifdef RESOLUTION_1280x720
        resize(*frame1 , *frame1 , Size(1280,720) , 0 , 0 , INTER_CUBIC);
        resize(*frame2 , *frame2 , Size(1280,720) , 0 , 0 , INTER_CUBIC);
#endif
    }
}

void change_resolution(VideoCapture* capL ,VideoCapture* capR){
#ifdef RESOLUTION_320x240
    capL->set(CV_CAP_PROP_FRAME_WIDTH, 320);
    capL->set(CV_CAP_PROP_FRAME_HEIGHT,240);
    capR->set(CV_CAP_PROP_FRAME_WIDTH, 320);
    capR->set(CV_CAP_PROP_FRAME_HEIGHT,240);
}

```

```

#endif

#ifdef RESOLUTION_640x480
    capL->set(CV_CAP_PROP_FRAME_WIDTH, 640);
    capL->set(CV_CAP_PROP_FRAME_HEIGHT, 480);
    capR->set(CV_CAP_PROP_FRAME_WIDTH, 640);
    capR->set(CV_CAP_PROP_FRAME_HEIGHT, 480);
#endif

#ifdef RESOLUTION_1280x960
    capL->set(CV_CAP_PROP_FRAME_WIDTH, 1280);
    capL->set(CV_CAP_PROP_FRAME_HEIGHT, 720);
    capR->set(CV_CAP_PROP_FRAME_WIDTH, 1280);
    capR->set(CV_CAP_PROP_FRAME_HEIGHT, 720);
#endif

cout << "Camera_1_Resolution : " << capL->get(CV_CAP_PROP_FRAME_WIDTH);
cout << "Camera_2_Resolution : " << capR->get(CV_CAP_PROP_FRAME_WIDTH);
}

void imageProcessing1(Mat Image, Mat MedianImage, Mat MedianImageBGR){

    // Apply Median Filter
    medianBlur(Image, MedianImage, 5);
    applyColorMap(MedianImage, MedianImageBGR, COLORMAP_JET);

    // Output
    imshow("Disparity_Map_Median_Filter_3x3", MedianImage);
    imshow("Disparity_Map_Median_Filter_3x3_-_RGB", MedianImageBGR);
}

void contrast_and_brightness(Mat &left, Mat &right, float alpha, float beta)
// Contrast and Brightness. Do the operation: new_image(i, j) = alpha*left(i, j) + beta*right(i, j);
}

```

```

for( int y = 0; y < left.rows; y++ ){
    for( int x = 0; x < left.cols; x++ ){
        for( int c = 0; c < 3; c++ ){
            left .at<Vec3b>(y,x)[c] = saturate_cast<uchar>( alpha*(
                right .at<Vec3b>(y,x)[c] = saturate_cast<uchar>( alpha*(
            )
        )
    )
}
}

```

setstereoparams.cpp

```

/*
 * setstereoparams.cpp
 *
 * Created on: Oct 1, 2015
 * Author: nicolasrosa
 */
#include "setstereoparams.h"
#include "ui_setstereoparams.h"
#include "iostream"
#include "StereoProcessor.h"

#include <QHBoxLayout>
#include <QSlider>
#include <QSpinBox>

//using namespace cv;
using namespace std;

SetStereoParams::SetStereoParams(QWidget *parent, StereoProcessor *stereo
    ui->setupUi(this);
//stereocfg_pointer = &stereocfg;

```

```

this->stereo = stereo;

connect(ui->preFilterSize_slider, SIGNAL(valueChanged(int)), ui->preF
connect(ui->preFilterCap_slider, SIGNAL(valueChanged(int)), ui->preFil
connect(ui->SADWindowSize_slider, SIGNAL(valueChanged(int)), ui->SADW
connect(ui->minDisparity_slider, SIGNAL(valueChanged(int)), ui->minDi
connect(ui->numberOfDisparities_slider, SIGNAL(valueChanged(int)), ui->n
connect(ui->textureThreshold_slider, SIGNAL(valueChanged(int)), ui->te
connect(ui->uniquenessRatio_slider, SIGNAL(valueChanged(int)), ui->un
connect(ui->speckleWindowSize_slider, SIGNAL(valueChanged(int)), ui->spe
connect(ui->speckleRange_slider, SIGNAL(valueChanged(int)), ui->speckl
connect(ui->disp12MaxDiff_slider, SIGNAL(valueChanged(int)), ui->disp1
connect(ui->preFilterSize_spinBox, SIGNAL(valueChanged(int)), ui->preF
connect(ui->preFilterCap_spinBox, SIGNAL(valueChanged(int)), ui->preF
connect(ui->SADWindowSize_spinBox, SIGNAL(valueChanged(int)), ui->SADW
connect(ui->minDisparity_spinBox, SIGNAL(valueChanged(int)), ui->minD
connect(ui->numberOfDisparities_spinBox, SIGNAL(valueChanged(int)), ui->n
connect(ui->textureThreshold_spinBox, SIGNAL(valueChanged(int)), ui->t
connect(ui->uniquenessRatio_spinBox, SIGNAL(valueChanged(int)), ui->u
connect(ui->speckleWindowSize_spinBox, SIGNAL(valueChanged(int)), ui->s
connect(ui->speckleRange_spinBox, SIGNAL(valueChanged(int)), ui->speckl
connect(ui->disp12MaxDiff_spinBox, SIGNAL(valueChanged(int)), ui->disp1

}

//Ui::SetStereoParams* SetStereoParams::getUi(){
//    return (this->ui);
//}

void SetStereoParams::loadStereoParamsUi(int preFilterSize, int preFilterC
    this->ui->preFilterSize_slider->setValue(preFilterSize);
    this->ui->preFilterSize_spinBox->setValue(preFilterSize);

    this->ui->preFilterCap_slider->setValue(preFilterCap);

```

```

this ->ui ->preFilterCap_spinBox ->setValue( preFilterCap );

this ->ui ->SADWindowSize_slider ->setValue( SADWindowSize );
this ->ui ->SADWindowSize_spinBox ->setValue( SADWindowSize );

this ->ui ->minDisparity_slider ->setValue( minDisparity );
this ->ui ->minDisparity_spinBox ->setValue( minDisparity );

this ->ui ->numberOfDisparities_slider ->setValue( numberOfDisparities );
this ->ui ->numberOfDisparities_spinBox ->setValue( numberOfDisparities )

this ->ui ->textureThreshold_slider ->setValue( textureThreshold );
this ->ui ->textureThreshold_spinBox ->setValue( textureThreshold );

this ->ui ->uniquenessRatio_slider ->setValue( uniquenessRatio );
this ->ui ->uniquenessRatio_spinBox ->setValue( uniquenessRatio );

this ->ui ->speckleWindowSize_slider ->setValue( speckleWindowSize );
this ->ui ->speckleWindowSize_spinBox ->setValue( speckleWindowSize );

this ->ui ->speckleRange_slider ->setValue( speckleRange );
this ->ui ->speckleRange_spinBox ->setValue( speckleRange );

this ->ui ->disp12MaxDiff_slider ->setValue( disp12MaxDiff );
this ->ui ->disp12MaxDiff_spinBox ->setValue( disp12MaxDiff );
}

// void SetStereoParams :: getStereoParamsUi () {

// }

SetStereoParams ::~ SetStereoParams ()
{

```

```
    delete ui;
}

/* Sliders */

void SetStereoParams :: on_preFilterSize_slider_valueChanged(int value)
{
    cout << "Bar1:" << value << endl;
    updateValues();
}

void SetStereoParams :: on_preFilterCap_slider_valueChanged(int value)
{
    cout << "Bar2:" << value << endl;
    updateValues();
}

void SetStereoParams :: on_SADWindowSize_slider_valueChanged(int value)
{
    cout << "Bar3:" << value << endl;
    updateValues();
}

void SetStereoParams :: on_minDisparity_slider_valueChanged(int value)
{
    cout << "Bar4:" << value << endl;
    updateValues();
}

void SetStereoParams :: on_numberOfDisparities_slider_valueChanged(int value)
{
    cout << "Bar5:" << value << endl;
    updateValues();
}
```

```
void SetStereoParams::on_textureThreshold_slider_valueChanged(int value)
{
    cout << "Bar6:" << value << endl;
    updateValues();
}

void SetStereoParams::on_uniquenessRatio_slider_valueChanged(int value)
{
    cout << "Bar7:" << value << endl;
    updateValues();
}

void SetStereoParams::on_speckleWindowSize_slider_valueChanged(int value)
{
    cout << "Bar8:" << value << endl;
    updateValues();
}

void SetStereoParams::on_speckleRange_slider_valueChanged(int value)
{
    cout << "Bar9:" << value << endl;
    updateValues();
}

void SetStereoParams::on_disp12MaxDiff_slider_valueChanged(int value)
{
    cout << "Bar10:" << value << endl;
    updateValues();
}

/* SpinBoxes */
void SetStereoParams::on_preFilterSize_spinBox_valueChanged(int value)
```

```

{

    cout << "Spin1:_" << value << endl;
    updateValues();

}

void SetStereoParams :: on_preFilterCap_spinBox_valueChanged(int value)
{
    cout << "Spin2:_" << value << endl;
    updateValues();
}

void SetStereoParams :: on_SADWindowSize_spinBox_valueChanged(int value)
{
    cout << "Spin3:_" << value << endl;
    updateValues();
}

void SetStereoParams :: on_minDisparity_spinBox_valueChanged(int value)
{
    cout << "Spin4:_" << value << endl;
    updateValues();
}

void SetStereoParams :: on_numberOfDisparities_spinBox_valueChanged(int va
{
    cout << "Spin5:_" << value << endl;
    updateValues();
}

void SetStereoParams :: on_textureThreshold_spinBox_valueChanged(int value)
{
    cout << "Spin6:_" << value << endl;
    updateValues();
}

```

}

```
void SetStereoParams :: on_uniquenessRatio_spinBox_valueChanged(int value)
{
    cout << "Spin7:_" << value << endl;
    updateValues ();
}
```

```
void SetStereoParams :: on_speckleWindowSize_spinBox_valueChanged(int value)
{
    cout << "Spin8:_" << value << endl;
    updateValues ();
}
```

```
void SetStereoParams :: on_speckleRange_spinBox_valueChanged(int value)
{
    cout << "Spin9:_" << value << endl;
    updateValues ();
}
```

```
void SetStereoParams :: on_disp12MaxDiff_spinBox_valueChanged(int value)
{
    cout << "Spin10:_" << value << endl;
    updateValues ();
}
```

```
void SetStereoParams :: on_buttonBox_accepted (){
```

}

```
void SetStereoParams :: on_buttonBox_rejected (){
```

}

```

void SetStereoParams :: updateValues () {
    std :: cout << "UPDATE_VALUES !\n";
    stereo ->setValues ( ui ->preFilterSize_slider ->value () ,
                           ui ->preFilterCap_slider ->value () ,
                           ui ->SADWindowSize_slider ->value () ,
                           ui ->minDisparity_slider ->value () ,
                           ui ->numberOfDisparities_slider ->value () ,
                           ui ->textureThreshold_slider ->value () ,
                           ui ->uniquenessRatio_slider ->value () ,
                           ui ->speckleWindowSize_slider ->value () ,
                           ui ->speckleRange_slider ->value () ,
                           ui ->disp12MaxDiff_slider ->value ()) ;

    this ->stereo ->setStereoParams ();
}

```

StereoCalib.cpp

```

/*
 * StereoCalib . cpp
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */

#include "StereoCalib.h"

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

void StereoCalib :: createKMatrix (){
    this ->K=Mat :: eye (3 ,3 ,CV_64F);
}

```

```

this ->K. at<double>(0,0)=this ->focalLength ;
this ->K. at<double>(1,1)=this ->focalLength ;
// this ->K. at<double>(0,2)=(this ->imageSize . width - 1.0)/2.0;
// this ->K. at<double>(1,2)=(this ->imageSize . height - 1.0)/2.0;
this ->K. at<double>(0,2)=(0 - 1.0)/2.0;
this ->K. at<double>(1,2)=(0 - 1.0)/2.0;
cout << "K: " << endl << this ->K << endl ;
}

/* *** Read Q Matrix function
** Description: Reads the Q Matrix in the *.yml file
** Receives:      Matrices Addresses for storage
** Returns:       Nothing
**
** Perspective transformation matrix(Q)
** [ 1   0   0           -cx      ]
** [ 0   1   0           -cy      ]
** [ 0   0   0           f        ]
** [ 0   0   -1/Tx      (cx-cx')/Tx]
***/

void StereoCalib :: readQMatrix(){
FileStorage fs(this ->QmatrixFileName , FileStorage ::READ);

if(this ->is640x480){
    if(!fs . isOpened()){
        cerr << "Failed_to_open_Q.yml_file" << endl ;
        return ;
    }

    fs [ "Q" ] >> this ->Q;
    // Check
    if(!this ->Q. data){
        cerr << "Check_Q_Matrix_Content!" << endl ;
    }
}

```

```

    return ;
}

// Display
cout << "Q: " << endl << this->Q << endl;

this->focalLength = this->Q.at<double>(2,3); cout << "f:" << this->focalLength;
this->baseline = -1.0/this->Q.at<double>(3,2); cout << "baseline" << this->baseline;

} else {
    cerr << "Check_Q.yml_file !\n" << endl;
    return ;
}
}

void StereoCalib::calculateQMatrix(){
    this->Q = Mat::eye(4,4,CV_64F);
    this->Q.at<double>(0,3)=-this->imageCenter.x;
    this->Q.at<double>(1,3)=-this->imageCenter.y;
    this->Q.at<double>(2,3)=this->focalLength;
    this->Q.at<double>(3,3)=0.0;
    this->Q.at<double>(2,2)=0.0;
    this->Q.at<double>(3,2)=1.0/this->baseline;
    cout << "Q: " << endl << this->Q << endl;
}
}
```

StereoConfig.cpp

```

/*
 *  StereoConfig.cpp
 *
 *  Created on: Dec 3, 2015
 *      Author: nicolasrosa
 */

```

```
#include "StereoConfig.h"

/* Constructor */
StereoConfig::StereoConfig(){}
```

StereoCustom.h

```
/*
 * StereoCustom.h
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */
```

```
#include "StereoCustom.h"
```

StereoDiff.cpp

```
/*
 * StereoDiff.cpp
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */
```

```
#include "StereoDiff.h"
```

```
/* Constructor */
StereoDiff::StereoDiff(){
    StartDiff=false;
    alpha = 0.5;
    beta = (1.0-alpha);
}
```

```
void StereoDiff::createDiffImage(Mat input1, Mat input2){
    absdiff(input1, input2, diffImage);
```

```
}
```

```
void StereoDiff::createResAND(Mat input1, Mat input2){
```

```
    bitwise_and(input1, input2, this->res_AND);
```

```
}
```

```
void StereoDiff::convertToBGR(){
```

```
    cvtColor(res_AND, res_AND_BGR, CV_GRAY2BGR);
```

```
}
```

```
void StereoDiff::addRedLines(){
```

```
    split(res_AND_BGR, res_AND_BGR_channels);
```

```
// Set the Blue and Green Channels to 0
```

```
res_AND_BGR_channels[0] = Mat::zeros(res_AND.rows, res_AND.cols, CV_8UC3);
```

```
res_AND_BGR_channels[1] = Mat::zeros(res_AND.rows, res_AND.cols, CV_8UC3);
```

```
cv::merge(res_AND_BGR_channels, 3, res_AND_BGR);
```

```
addWeighted(this->imageL, alpha, res_AND_BGR, beta, 0.0, res_ADD);
```

```
// imshow("Add", res_ADD);
```

```
}
```

StereoDisparityMap.cpp

```
/*
```

```
* StereoDisparityMap.cpp
```

```
*
```

```
* Created on: Dec 3, 2015
```

```
* Author: nicolasrosa
```

```
*/
```

```
#include "StereoDisparityMap.h"
```

```
/* Constructor */
StereoDisparityMap :: StereoDisparityMap () {
    // Allocate Memory
    //Mat disp      = Mat( imageR[0].rows , imageR[0].cols , CV_16UC1 );
    //Mat disp_8U   = Mat( stereo ->imageR[0].rows , stereo ->imageR[0].cols ,
    //Mat disp_BGR = Mat( stereo ->imageR[0].rows , stereo ->imageR[0].cols ,
}
```

StereoFlags.cpp

```
/*
 * StereoFlags.cpp
 *
 * Created on: Dec 3, 2015
 * Author: nicolasrosa
 */
```

```
#include "StereoFlags.h"
```

```
/* Constructor */
StereoFlags :: StereoFlags () {
    showInputImages=true ;
    showXYZ=false ;
    showStereoParam=false ;
    showStereoParamValues=false ;
    showFPS=false ;
    showDisparityMap=false ;
    show3Dreconstruction=false ;
    showTrackingObjectView=false ;
    showDiffImage=false ;
    showWarningLines=false ;
}
```

StereoProcessor.cpp

```
/*
```

```

*  StereoProcessor.cpp
*
*  Created on: Oct 20, 2015
*      Author: nicolasrosa
*/
#include "trackObject.h"

/* Constructor */
#include "StereoProcessor.h"

StereoProcessor::StereoProcessor(int number) {
    inputNum=number;
}

int StereoProcessor::getInputNum(){
    return inputNum;
}

void StereoProcessor::readConfigFile(){
//FileStorage fs( "../reprojectImageTo3D_calibON_bm_GUI/config.yml",
FileStorage fs( "/home/nicolas/repository/StereoVision/Qt_Creator/repr

    if (!fs.isOpened()){
        cerr << "Failed_to_open_config.yml_file!" << endl;
        return ;
    }
    fs["Intrinsics_Path"] >> this->calib.intrinsicsFileName;
    fs["Extrinsics_Path"] >> this->calib.extrinsicsFileName;
    fs["Q_Matrix_Path"] >> this->calib.QmatrixFileName;
    fs["Stereo_Parameters_Path"] >> this->calib.StereoParamFileName;

    fs.release();
}

```

```

cout << "-----Config .yml-----"
cout << "Intrinsics_Path : "           << this->calib . intrinsicsFileName
<< endl;
cout << "Extrinsics_Path : "           << this->calib . extrinsicsFileName
<< endl;
cout << "Q_Matrix_Path : "             << this->calib . QmatrixFileName
<< endl;
cout << "Stereo_Parameters_Path : "   << this->calib . StereoParamFileName
cout << "Config .yml_Read_Successfully ." << endl << endl ;
cout << "-----"
}

void StereoProcessor :: readStereoConfigFile (){
    FileStorage fs (this->calib . StereoParamFileName , FileStorage ::READ);
    if (!fs . isOpened ()) {
        cerr << "Failed_to_open_stereo .yml_file !" << endl;
        return;
    }

    fs [ "preFilterSize" ] >> this->stereocfg . preFilterSize ;
    fs [ "preFilterCap" ] >> this->stereocfg . preFilterCap ;
    fs [ "SADWindowSize" ] >> this->stereocfg . SADWindowSize ;
    fs [ "minDisparity" ] >> this->stereocfg . minDisparity ;
    fs [ "numberOfDisparities" ] >> this->stereocfg . numberOfDisparities ;
    fs [ "textureThreshold" ] >> this->stereocfg . textureThreshold ;
    fs [ "uniquenessRatio" ] >> this->stereocfg . uniquenessRatio ;
    fs [ "speckleWindowSize" ] >> this->stereocfg . speckleWindowSize ;
    fs [ "speckleRange" ] >> this->stereocfg . speckleRange ;
    fs [ "disp12MaxDiff" ] >> this->stereocfg . disp12MaxDiff ;

    fs . release ();
}

```

```

// Display
cout << "-----StereoConfig-----"
cout << "preFilterSize : " << this->stereocfg . preFilterSize
<< endl;
cout << "preFilterCap : " << this->stereocfg . preFilterCap
<< endl;
cout << "SADWindowSize : " << this->stereocfg . SADWindowSize
<< endl;
cout << "minDisparity : " << this->stereocfg . minDisparity
<< endl;
cout << "numberOfDisparities : " << this->stereocfg . numberOfDisparities
<< endl;
cout << "textureThreshold : " << this->stereocfg . textureThreshold
<< endl;
cout << "uniquenessRatio : " << this->stereocfg . uniquenessRatio
<< endl;
cout << "speckleWindowSize : " << this->stereocfg . speckleWindowSize
<< endl;
cout << "speckleRange : " << this->stereocfg . speckleRange
<< endl;
cout << "disp12MaxDiff : " << this->stereocfg . disp12MaxDiff
<< endl;
cout << "stereo . yml_Read_Successfully ." << endl << endl;
cout << "-----"
}


```

```

/*** Stereo Initialization function
** Description: Executes the PreSetup of parameters of the StereoBM object
** @param StereoBM bm: Correspondence Object
** Returns: Nothing
*/
void StereoProcessor::stereoInit(){
    this->bm->setPreFilterCap(31);
}
```

```

this ->bm->setBlockSize(25 > 0 ? 25 : 9);
this ->bm->setMinDisparity(0);
this ->bm->setNumDisparities(3);
this ->bm->setTextureThreshold(10);
this ->bm->setUniquenessRatio(15);
this ->bm->setSpeckleWindowSize(100);
this ->bm->setSpeckleRange(32);
this ->bm->setDisp12MaxDiff(1);
}

/* *** Stereo Calibration function

** Description: Reads the Calibrations in *.yml files
** Receives: Matrices Addresses for storage
** @param Mat M1,M2: Intrinsic Matrices from camera 1 and 2
** @param Mat D1,D2: Distortion Coefficients from camera 1 and 2
** @param Mat R: Rotation Matrix
** @param Mat t: Translation Vector
** Returns: Nothing
***/

void StereoProcessor::stereoCalib(){
    FileStorage fs(this->calib.intrinsicsFileName, FileStorage::READ);
    if (!fs.isOpened()){
        cerr << "Failed_to_open_intrinsics.yml_file!" << endl;
        return;
    }

    fs["M1"] >> this->calib.M1;
    fs["D1"] >> this->calib.D1;
    fs["M2"] >> this->calib.M2;
    fs["D2"] >> this->calib.D2;

    fs.release();
}

```

```

float scale = 1.f;
this->calib.M1 *= scale;
this->calib.M2 *= scale;

fs . open( this->calib.extrinsicsFileName , FileStorage ::READ);
if (! fs . isOpened()){
    cerr << "Failed_to_open_extrinsics .yml_file !" << endl;
    return;
}

fs [ "R" ] >> this->calib.R;
fs [ "T" ] >> this->calib.T;

fs . release ();

// Check
if (! this->calib.M1.data || !this->calib.D1.data || !this->calib.M2.d
    cerr << "Check_instrinsics_and_extrinsics_Matrixes_content !" << endl;
    return;
}

// Display
cout << "-----Intrinsics-----"
cout << "M1:" << endl << this->calib.M1 << endl;
cout << "D1:" << endl << this->calib.D1 << endl;
cout << "M2:" << endl << this->calib.M2 << endl;
cout << "D2:" << endl << this->calib.D2 << endl << endl;
cout << "intrinsics .yml_Read_Successfully ." << endl << endl;

cout << "-----Extrinsics-----"
cout << "R:" << endl << this->calib.R << endl;
cout << "T:" << endl << this->calib.T << endl << endl;
cout << "extrinsics .yml_Read_Successfully ." << endl;

```

```

cout << "-----"
}

/* *** Stereo Parameters Configuration function
** Description: Executes the setup of parameters of the StereoBM object
** @param rect roi1: Region of Interest 1
** @param rect roi2: Region of Interest 2
** @param StereoBM bm: Correspondence Object
** @param int numRows: Number of Rows of the input Images
** @param bool showStereoBMparams
** Returns: Nothing
***/

void StereoProcessor::setStereoParams(){
    int trackbarsAux[10];

    trackbarsAux[0] = this -> stereocfg . preFilterSize * 2.5 + 5;
    trackbarsAux[1] = this -> stereocfg . preFilterCap * 0.625 + 1;
    trackbarsAux[2] = this -> stereocfg . SADWindowSize * 2.5 + 5;
    trackbarsAux[3] = this -> stereocfg . minDisparity * 2.0 - 100;
    trackbarsAux[4] = this -> stereocfg . numberOfDisparities * 16;
    trackbarsAux[5] = this -> stereocfg . textureThreshold * 320;
    trackbarsAux[6] = this -> stereocfg . uniquenessRatio * 2.555;
    trackbarsAux[7] = this -> stereocfg . speckleWindowSize * 1.0;
    trackbarsAux[8] = this -> stereocfg . speckleRange * 1.0;
    trackbarsAux[9] = this -> stereocfg . disp12MaxDiff * 1.0;

    this -> bm -> setROI1(this -> calib . roi1 );
    this -> bm -> setROI1(this -> calib . roi2 );

    this -> numRows = imageL[0].rows;

    if (trackbarsAux[0] % 2 == 1 && trackbarsAux[0] >= 5 && trackbarsAux[0] <= 25)
        //bm.state -> preFilterSize = trackbarsAux[0];
}

```

```

bm->setPreFilterSize (trackbarsAux [0]);

}

if (trackbarsAux [1]>=1 && trackbarsAux [1]<=63){
    //bm. state ->preFilterCap = trackbarsAux [1];
    bm->setPreFilterCap (trackbarsAux [1]);
}

if (trackbarsAux [2]%2==1 && trackbarsAux [2]>=5 && trackbarsAux [2]<=25)
    //bm. state ->SADWindowSize = trackbarsAux [2];
    bm->setBlockSize (trackbarsAux [2]);
}

if (trackbarsAux [3]>=-100 && trackbarsAux [3]<=100){
    //bm. state ->minDisparity = trackbarsAux [3];
    bm->setMinDisparity (trackbarsAux [3]);
}

if (trackbarsAux [4]%16==0 && trackbarsAux [4]>=16 && trackbarsAux [4]<=16)
    //bm. state ->numberOfDisparities = trackbarsAux [4];
    bm->setNumDisparities (trackbarsAux [4]);
}

if (trackbarsAux [5]>=0 && trackbarsAux [5]<=32000){
    //bm. state ->textureThreshold = trackbarsAux [5];
    bm->setTextureThreshold (trackbarsAux [5]);
}

if (trackbarsAux [6]>=0 && trackbarsAux [6]<=255){
    //bm. state ->uniquenessRatio = trackbarsAux [6];
    bm->setUniquenessRatio (trackbarsAux [6]);
}

```

```

if (trackbarsAux [7]>=0 && trackbarsAux [7]<=100){
    //bm. state ->speckleWindowSize = trackbarsAux [7];
    bm->setSpeckleWindowSize (trackbarsAux [7]);
}

if (trackbarsAux [8]>=0 && trackbarsAux [8]<=100){
    //bm. state ->speckleRange = trackbarsAux [8];
    bm->setSpeckleRange (trackbarsAux [8]);
}

if (trackbarsAux [9]>=0 && trackbarsAux [9]<=100){
    //bm. state ->disp12MaxDiff = trackbarsAux [9];
    bm->setDisp12MaxDiff (trackbarsAux [9]);
}

if (showStereoParamsValues){
    cout << getTrackbarPos ("preFilterSize", trackbarWindowName)
    cout << getTrackbarPos ("preFilterCap", trackbarWindowName)
    cout << getTrackbarPos ("SADWindowSize", trackbarWindowName)
    cout << getTrackbarPos ("minDisparity", trackbarWindowName)
    cout << getTrackbarPos ("numberOfDisparities", trackbarWindowName)
    cout << getTrackbarPos ("textureThreshold", trackbarWindowName)
    cout << getTrackbarPos ("uniquenessRatio", trackbarWindowName)
    cout << getTrackbarPos ("speckleWindowSize", trackbarWindowName)
    cout << getTrackbarPos ("speckleRange", trackbarWindowName)
    cout << getTrackbarPos ("disp12MaxDiff", trackbarWindowName)
}
}

void StereoProcessor :: imageProcessing (Mat src , Mat imgE , Mat imgED, Mat ca
    Mat erosionElement = getStructuringElement( MORPH_RECT, Size( 2*EROSIO
    Mat dilationElement = getStructuringElement( MORPH_RECT, Size( 2*DILA
    Mat imgEBGR, imgEDBGR;

```

```

Mat imgEDMedian, imgEDMedianBGR;
int x, y;

// Mat imgThreshold;
static Mat lastImgThreshold;
int nPixels, nTotal; // static int lastThresholdSum=0;

// Near Object Detection

// Prefiltering
// Apply Erosion and Dilation to take out spurious noise
erode(src, imgE, erosionElement);
dilate(imgE, imgED, erosionElement);

applyColorMap(imgE, imgEBGR, COLORMAP_JET);
applyColorMap(imgED, imgEDBGR, COLORMAP_JET);

// Apply Median Filter
// GaussianBlur(imgED, imgEDMedian, Size(3,3), 0, 0);
medianBlur(imgED, imgEDMedian, 5);
applyColorMap(imgEDMedian, imgEDMedianBGR, COLORMAP_JET);

// Thresholding
// adaptiveThreshold(imgEDMedian, imgThreshold, 255, ADAPTIVE_THRESH_MEAN_C);
// adaptiveThreshold(imgEDMedian, imgThreshold, 255, ADAPTIVE_THRESH_GAUSSIAN_C);
threshold(imgEDMedian, imgThreshold, THRESH_VALUE, 255, THRESH_BINARY);
erode(imgThreshold, imgThreshold, erosionElement);
dilate(imgThreshold, imgThreshold, dilationElement);

// Solving Lighting Noise Problem
nPixels = sum(imgThreshold)[0]/255;
nTotal = imgThreshold.total();

```

```

// cout << "Number of Pixels :" << nPixels << endl;
// cout << "Ratio is : " << ((float)nPixels)/nTotal << endl << endl;

if (((float)nPixels)/nTotal)>0.5{
    // sleep(1);
    // cout << "Lighting Noise !!!" << endl;
    // cout << "Number of Pixels :" << nPixels << endl;
    // cout << "Ratio is : " << ((float)nPixels)/nTotal << endl;

    // Invalidates the last frame
    imgThreshold = lastImgThreshold;
} else {
    // Saves the last valid frame
    lastImgThreshold=imgThreshold;
    //lastThresholdSum = CurrentThresholdSum;
}

// Output
//imshow("Eroded Image",imgE);
//imshow("Eroded Image BGR",imgEBGR);
//imshow("Eroded+Dilated Image",imgED);
//imshow("Eroded+Dilated Image BGR",imgEDBGR);
//imshow("Eroded+Dilated+Median Image",imgEDMedian);
//imshow("Eroded+Dilated+Median Image BGR",imgEDMedianBGR);

//imshow("Thresholded Image",imgThreshold);

// Tracking Object
if(isTrackingObjects){
    cameraFeedL.copyTo(trackingView);
    trackFilteredObject(x,y,imgThreshold,trackingView);
    //imshow("Tracking Object",trackingView);
}

```

}

// Saving Previous Frame

void StereoProcessor :: saveLastFrames () {

 imageL [0]. copyTo (imageL [1]);

 imageR [0]. copyTo (imageR [1]);

 imageL_grey [0]. copyTo (imageL_grey [1]);

 imageR_grey [0]. copyTo (imageR_grey [1]);

}

void StereoProcessor :: setValues (**int** preFilterSize , **int** preFilterCap , **int**

 stereocfg . preFilterSize = preFilterSize ;

 stereocfg . preFilterCap = preFilterCap ;

 stereocfg . SADWindowSize = sadWindowSize ;

 stereocfg . minDisparity = minDisparity ;

 stereocfg . numberOfDisparities = numOfDisparities ;

 stereocfg . textureThreshold = textureThreshold ;

 stereocfg . uniquenessRatio = uniquenessRatio ;

 stereocfg . speckleRange = speckleWindowRange ;

 stereocfg . speckleWindowSize = speckleWindowSize ;

 stereocfg . disp12MaxDiff = disp12MaxDiff ;

 std :: cout << "SET_VALUES ! \n " ;

}

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 [?]. A segunda funcionalidade corresponde à parte de rastreamento do objeto desenvolvida por Kyle Hounslow [?].

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){
```

$$\text{Mat temp;} \\ \text{threshold.copyTo(temp);} \\ //these two vectors needed for output of findContours \\ \text{std::vector< std::vector<Point> > contours;} \\ \text{std::vector<Vec4i> hierarchy;} \\ //find contours of filtered image using openCV findContours func \\ \text{findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX)} \\ //use moments method to find our filtered object \\ \text{double refArea = 0;} \\ \text{bool objectFound = false;} \\ \text{if (hierarchy.size() > 0) { } \\ \quad \text{int numObjects = hierarchy.size();} \\ //if number of objects greater than MAX_NUM_OBJECTS we have a no \\ \text{if (numObjects < MAX_NUM_OBJECTS) { } \\ \quad \text{for (int index = 0; index >= 0; index = hierarchy)}$$

```

    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();
}

```