Universidade Federal
do Rio de Janeiro
Escola Politécnica

DETECÇÃO E MODELAGEM EFICIENTE DE ESCADAS PARA CONTROLE DE SUBIDA DE ROBÔS MÓVEIS

Derek Kevin Shu Chan

Projeto de Graduação apresentado ao Corpo Docente do Curso de Engenharia de Controle e Automação da Escola Politécnica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientadores: Fernando Cesar Lizarralde
João Carlos Espiúca Monteiro

Rio de Janeiro
Setembro de 2017

DETECÇÃO E MODELAGEM EFICIENTE DE ESCADAS PARA CONTROLE DE SUBIDA DE ROBÔS MÓVEIS

Derek Kevin Shu Chan

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

Prof. Fernando Cesar Lizarralde, D.Sc.

Prof. Liu Hsu, Dr. d'État

Eng. João Carlos Espiúca Monteiro, BEE

Eng. Ivanko Yannick Yanque Tomasevich, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2017

Shu Chan, Derek Kevin

Detecção e Modelagem Eficiente de Escadas para Controle de Subida de Robôs Móveis / Derek Kevin Shu Chan. – Rio de Janeiro: UFRJ/Escola Politécnica, 2017.

XII, 51 p.: il.; 29, 7cm.

Orientadores: Fernando Cesar Lizarralde

João Carlos Espiúca Monteiro

Projeto de Graduação – UFRJ/Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2017.

Referências Bibliográficas: p. 49 – 50.

1. Detecção de objetos. 2. Modelagem. 3. Robótica Móvel. I. Lizarralde, Fernando Cesar *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Dedico à minha família e amigos.

Agradecimentos

Gostaria de agradecer, primeiramente, à minha família, pais, Jane Shu e Leandro Chan, irmã, Priscilla Chan, avós e tios; que sempre me incentivaram, apoiaram e acreditaram em mim para obter sucesso em todas as etapas da minha vida.

Ao meu orientador, Fernando Lizarralde, pelos grandes ensinamentos, críticas, dicas e sugestões durante todo o desenvolvimento deste projeto e ao longo da minha formação acadêmica. Agradeço também ao meu amigo João Monteiro por toda a paciência, conselhos e ensinamentos durante o projeto.

Agradeço aos amigos antigos: Pedro Truppel e Rodrigo Rocha; e aos amigos do Ciência sem Fronteiras: Pedro Samuel, Isabela Leme Cruz, Rafael Rigaud, Abner Cabral e Isabella Dall'Asta. Obrigado por dividirem comigo tantos momentos de alegria e por me ajudarem a superar tantos desafios.

Aos amigos de Controle: Rôb Klér, Tiago Azevedo, Isabelle Contreras, Pedro Gomes, Vinícius Plácido, Gabriel Lira, Gabriel Ribeiro, Lívia Paravidino, Gabriel Evangelista, Ricardo Oliveira e José Guilherme Monteiro, que trilharam comigo este caminho durante todos esses anos de graduação.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação

DETECÇÃO E MODELAGEM EFICIENTE DE ESCADAS PARA CONTROLE DE SUBIDA DE ROBÔS MÓVEIS

Derek Kevin Shu Chan

Setembro/2017

Orientadores: Fernando Cesar Lizarralde
João Carlos Espiúca Monteiro

Departamento: Engenharia de Controle e Automação

Este trabalho apresenta um algoritmo de detecção de escada computacionalmente eficiente, preciso e robusto a ruídos de medição. O método proposto foca na identificação de características bidimensionais (2D) de escadas à partir de uma nuvem de pontos 3D. Cada degrau é considerado como uma sequência de linhas que possuem domínio de definição semelhantes, sendo a escada um conjunto formado por essas sequências.

Para reduzir o custo computacional do algoritmo, utiliza-se uma *octree* de forma a reduzir a quantidade de pontos analisados da nuvem de pontos do ambiente. Os centróides resultantes são armazenados em n vetores, onde cada um contém pontos com a mesma altura. Este procedimento segmenta os dados 3D em n fatias horizontais 2D do mapa. A transformada de Hough é aplicada para buscar retas em cada fatia. O restante do algoritmo agrupa as retas que possuem o mesmo domínio de definição e determina se existem sequências que podem corresponder à escadas. As sequências que atendem às tolerâncias pré-estabelecidas são reconhecidas como escadas.

Para cada escada detectada, um modelo contendo a altura, largura, profundidade e quantidade de degraus é obtido. O algoritmo proposto é capaz de localizar e modelar de forma eficiente escadas em diferentes poses e é robusto ao ruído dos dados medidos, como indicado pelos resultados experimentais.

Abstract of Graduation Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Control and Automation Engineer

EFFICIENT STAIRWAY DETECTION AND MODELLING FOR CLIMB CONTROL OF MOBILE ROBOTS

Derek Kevin Shu Chan

September/2017

Advisors: Fernando Cesar Lizarralde
João Carlos Espiúca Monteiro

Department: Control and Automation Engineering

This work presents a computationally efficient and accurate stairway detection algorithm, robust to measurement noise. The proposed method focuses on identifying two-dimensional (2D) features of stairways embedded in 3D data. Each stair of a staircase is a sequence of lines with similar domain of definition, and the staircase is a sequence of these elements.

To reduce the computational cost, an octree is used to down-sample the environment *point-cloud* data. The resulting centroids are stored in n arrays, each one containing points with equal height. This process segments the 3D data into n 2D horizontal slices. The Hough transform is applied to scan each array for lines. The remainder of the algorithm groups lines with the same domain of definition and determines if sequences that may correspond to stairways exist. The sequences that meet the required tolerances are detected as a staircase.

For each detected stairway, a model containing that contains the height, the depth, the width and the amount of stairs is obtained. The proposed algorithm is able to locate and model staircases on different poses and is robust to noise from the measurement data, as indicated by the experimental results.

Sumário

Lista de Figuras **x**

Lista de Tabelas **xii**

1 Introdução **1**
1.1 Motivação . . . 2
1.2 Objetivos . . . 3
1.3 Revisão de Literatura . . . 3
1.3.1 Detecção e modelagem de escadas . . . 4
1.3.2 Plataforma ROS . . . 5
1.4 Plataforma OctoMap . . . 6
1.5 Estrutura do Texto . . . 7

2 Robô móvel com esteiras **8**
2.1 Sensores utilizados . . . 10
2.2 Computador embarcado . . . 12

3 Metodologia de detecção e modelagem de escadas **14**
3.1 Transformada de Hough . . . 14
3.2 Mapeamento do ambiente . . . 16
3.2.1 Mapeamento por *Kinect* . . . 16
3.2.2 Mapeamento com o sensor *laser scan VLP-16* . . . 17
3.3 Armazenamento dos dados . . . 18
3.4 Algoritmo de detecção de escadas . . . 19
3.4.1 Inicialização e Pré-Processamento . . . 21
3.4.2 Ordenação por Z e Filtragem . . . 21
3.4.3 Detecção de Retas . . . 22
3.4.4 Sequenciamento . . . 23
3.4.5 Segmentação e Re-sequenciamento . . . 23
3.4.6 Pós-Processamento . . . 25
3.5 Modelagem das escadas . . . 25
3.5.1 Cálculo das propriedades das escadas . . . 26

4 Implementação do algoritmo **28**
4.1 Pacote *diane_octomap* 28
4.1.1 Parâmetros de inicialização 29
4.1.2 Tópicos 30
4.1.3 Serviços 31
4.1.4 Utilização do pacote 32
4.2 Utilização através da interface gráfica Web 36

5 Resultados Experimentais da Detecção de Escada **38**
5.1 Resultados 38
5.1.1 Configuração do algoritmo 41
5.1.2 Discussão 42

6 Conclusões e Trabalhos Futuros **47**
6.1 Trabalhos futuros 48

Referências Bibliográficas **49**

A Informações dos sensores **51**

Lista de Figuras

1.1 Exemplos de robôs EOD. 2
1.2 Diagrama de uma *octree*. 6

2.1 Comportamento do sensor Kinect de acordo com a distância. 8
2.2 Robô DIANE manipulando um objeto. 8
2.3 Movimentação das esteiras. 9
2.4 Detalhamento dos sensores do *Kinect*. 10
2.5 Velodyne VLP-16. 11
2.6 Velodyne VLP-16. 12
2.7 Placa ADLQM67PC. 13

3.1 Mapeamento entre a representação de uma linha no espaço Euclidiano e sua representação no espaço de Hough. 15
3.2 Comportamento do sensor Kinect de acordo com a distância. 17
3.3 Sistema de coordenadas adotada pelo sensor Kinect. 17
3.4 Nuvem de Pontos obtida do *Kinect*. 18
3.5 Nuvem de Pontos obtida do Laser *Velodyne VLP-16*. 18
3.6 Mapeamento com resolução de 1 cm. 19
3.7 Mapeamento com resolução de 5 cm. 19
3.8 Ilustração das etapas que compõem o algoritmo de detecção de escada 19
3.9 Resultado da Inicialização e do Pré-Processamento 21
3.10 Resultado da Ordenação por Z e Filtragem 21
3.11 Resultado da Detecção de Retas com a Transformada de Hough . . . 22
3.12 Resultado da filtragem das retas durante o Sequenciamento inicial . . 22
3.13 Resultado da segmentação com os pontos detectados 25
3.14 Escada modelada com os pontos pertencentes à escada 27
3.15 Escada modelada 27

4.1 Interface Gráfica Web para operação do DIANE 36
4.2 Sessão de detecção de escadas - Botão para iniciar a detecção 37
4.3 Resultado da detecção de escadas 37

5.1 Escadas utilizadas para a avaliação do algoritmo proposto, com dados obtidos de diferentes perspectivas. 40
5.2 Ruído observado ao detectar os últimos 6 degraus da escada B com o *Kinect* 40
5.3 Dados experimentais obtidos durante o "Sequenciamento". Elementos em vermelho são descartados e os azuis são submetidos ao sequenciamento. 43
5.4 Resultado da Detecção à partir de dados obtidos com o *Kinect* da Escada A (a, b), Escada B (c, d) e Escada C (e, f). 44
5.5 Resultado da Detecção à partir de dados obtidos com o sensor laser *Velodyne VLP-16* da Escada A (a, b) e Escada B (c, d). 45
5.6 Resultado da Detecção à partir de dados obtidos simultaneamente com o *Kinect* e com sensor laser *Velodyne VLP-16* da Escada A. . . . 46

Lista de Tabelas

5.1 Relacionamento entre os parâmetros de inicialização e os do algoritmo. 41
5.2 Erro de modelagem de escada com o *Kinect*. 44
5.3 Erro de modelagem de escada com o *Velodyne VLP-16*. 45
5.4 Erro de modelagem de escada com o *Kinect* e *Velodyne VLP-16*. . . . 45

A.1 Informações do Kinect Fonte: https://msdn.microsoft.com/en-us/library/ 51
A.2 Informações do laser VLP-16 Informações retiradas do manual 51
A.3 Informações do laser UST-10LX Informações retiradas do manual 51

Capítulo 1

Introdução

Na área da robótica, a interação com o ambiente se faz muitas vezes necessária. Para que tal interação ocorra de uma maneira segura e previsível, robôs autônomos dependem fortemente de modelos confiáveis do ambiente e dos outros objetos presentes nele. Em situações nas quais tais modelos não são previamente conhecidos, o robô precisa identificá-los durante o decorrer da missão. Por esse motivo, a tarefa de detecção de objetos deve ser realizada frequentemente, caso contrário, torna-se um risco para a continuidade da missão. Desse modo, muitas vezes a mesma se torna custosa para o processamento e por conta do tempo necessário para sua finalização.

Neste cenário de exploração de ambientes desconhecidos, vêm crescendo a utilização de sistemas robóticos capazes de realizar a exploração, reconhecimento, observação e manipulação de objetos, sem a necessidade da presença do ser humano.

Um grupo mais específico de robôs amplamente utilizados em aplicações militares de exploração de ambientes urbanos são os denominados *Bomb Disposal/Explosive Ordnance Disposal (EOD)*. A função deste robô móvel sobre esteiras consiste em explorar, identificar e desarmar bombas e explosivos quando possível. A maior parte dos robôs desta classe apresenta também braços robóticos, que auxiliam na transposição de obstáculos, e um manipulador robótico, em sua parte superior, com graus de liberdade que permite a manipulação e o transporte de objetos. Exemplos de robôs dessa classe podem ser vistos na figura 1.1.

Em relação aos tipos de operação utilizados para a locomoção desses robôs, podemos classificá-los em três tipos: teleoperado, semiautônomo e autônomo.

Na teleoperação utiliza-se exclusivamente a intervenção humana para operar o robô, no qual o operador é capaz de reagir adequadamente à diferentes situações encontradas durante a missão. Contudo, a visão limitada das câmeras combinada com as informações limitadas de outros sensores dificulta a operação e exige que os operadores sejam altamente habilidosos e treinados na manipulação do robô, aumentando o tempo de treinamento do operador.

(a) Robô EOD. (b) Robô EOD subindo escada.

Figura 1.1: Exemplos de robôs EOD.
Fonte: https://br.pinterest.com/lmorhac/eod-robots/

Por sua vez, a operação semiautônoma também necessita da intervenção humana, mas a locomoção do robô é auxiliada por um computador. Isto fornece uma maior segurança ao sistema robótico e precisão no movimento, al'em de não necessitar mais de um operador altamente treinado e habilidoso. O tempo de treinamento de um operador para um robô semiautônomo é bastante reduzido comparado com o de um robô teleoperado.

Já em uma operação autônoma, não ocorre nenhuma intervenção humana, cabendo à uma elevada inteligência computacional controlar todos os aspectos do sistema robótico. Este tipo de operação traz, além dos benefícios citados na operação semiautônoma, uma maior velocidade de reação devido à falta de atrasos na comunicação e ao fato de computadores realizarem tarefas muito mais rápido do que um ser humano. Entretanto, ocorre uma limitação em relação aos cenários de operação, dado que depende da programação de rotinas de identificação das situações, podendo não agir de forma adequada à situações que saiam da operação normal.

1.1 Motivação

Durante operações de robôs móveis em aplicações de exploração e resgate em cenários urbanos, deparar-se com a necessidade de transpor uma escada como obstáculo é bem comum, principalmente no interior de construções civis. Desta forma, a importância de um método de locomoção eficiente sobre escadas torna-se evidente para a exploração de novos níveis.

A tarefa de subir uma escada é uma das tarefas mais difíceis de ser executada por um robô móvel. Este problema pode ser dividido em três etapas: (1) detectar a escada, (2) obter um modelo confiável e (3) subir a escada. Para a execução

desta tarefa de forma autônoma ou semiautônoma, é muito importante a existência de métodos precisos e robustos que realizem as duas primeiras etapas, detecção e modelagem. Caso o modelo final da escada não esteja de acordo com a realidade, uma ação incorreta, do operador ou da própria inteligência computacional do robô, pode causar danos ao mesmo e até mesmo levar ao fracasso da missão.

1.2 Objetivos

O objetivo principal deste projeto é propor uma metodologia de detecção e modelagem de uma escada reta e regular, a ser utilizada em conjunto com o controle para locomoção semiautônoma em escada de um robô sobre esteiras, proposto por de Lima De Lima (2016). Deste modo, o operador do robô poderá utilizar esse sistema de suporte para realizar esta tarefa.

O programa para controle do robô sobre esteiras foi desenvolvido baseado na plataforma ROS, que disponibiliza bibliotecas que facilitam a comunicação entre componentes e o gerenciamento do software. O mesmo foi implementado para ser escalável, segmentado e reutilizável. Este programa funciona por comandos externos, enviados por um operador.

A detecção e modelagem de escada também foi implementada utilizando esta plataforma, de forma a facilitar a integração com o controle do robô. Utilizando sensores RGD-B, como o *Kinect*, e sensores *laser scan*, obtém-se a nuvem de pontos com as informações do ambiente. Através do framework *Octomap*, os dados são armazenados dentro de uma estrutura de *octree* e ,em seguida, os pontos pertencentes à escadas são extraídos. Com estas informações, é possível encontrar um modelo aproximado para a escada.

Desta maneira, ao final do projeto, o robô deverá ser capaz de identificar e modelar uma escada do ambiente, de forma a obter os parâmetros necessários para o controle e para determinar se é possível transpor este obstáculo.

1.3 Revisão de Literatura

A detecção e a modelagem de uma escada são as primeiras etapas necessárias para fazer um sistema robótico transpor uma escada. Para tal, diversos trabalhos relacionados à esses dois temas foram avaliados. A nova metodologia de detecção e de modelagem descrita neste projeto foi desenvolvida em conjunto com as características do robô, descritas no capítulo 2, podendo ser utilizada com outros robôs.

1.3.1 Detecção e modelagem de escadas

Para se determinar se um robô será capaz de transpor ou não uma escada, a existência de um modelo da mesma é essencial. As características geométricas também são utilizadas durante a fase de planejamento do posicionamento e no controle do sistema robótico visando sua transposição.

Nos métodos de detecção e modelagem de escadas avaliados, foram identificados a utilização de três tipos de sensores: câmeras monoculares (como em Carbonara e Garagnella Carbonara & Guaragnella (2014)), sensores *laser scan* (como em Eilering et al. Eilering et al. (2014)) e sensores 3D (como em Delmerico et al. Delmerico et al. (2013)), como o *Kinect*.

Em Eilering et al. Eilering et al. (2014) é proposto um método probabilístico para a detecção de superfícies de suporte à partir da análise de dados armazenados em um *point cloud* (nuvem de pontos). Estes dados são segregados em um conjunto de partes disjuntas e coerentes no espaço, que são rotuladas com um tipo. Após a segregação, as partes são agrupadas e, caso passem por uma validação, são utilizadas na modelagem de diferentes objetos, como escadas, cadeiras e outros objetos. Para a obtenção dos dados a serem armazenados no *point cloud*, foram utilizados nos experimentos um Hokuyo UTM-30LX, um Kinect e um *TOF* (time-of-flight depth camera). Entretanto, este método obteve uma acurácia relativamente baixa de 75%.

Oßwald et al. Oßwald et al. (2011) compara dois métodos para realizar a detecção de uma escada. O primeiro deles se baseia em uma técnica chamada *Scan-Line Grouping*, na qual busca-se detectar superfícies em uma *depth image* (imagem em profundidade). Primeiro, para cada linha da imagem, são identificados segmentos de retas e depois aplica-se um crescimento de região utilizando as linhas como as primitivas para formar regiões planares e assim detectar os planos referentes às superfícies da escada. O segundo método utiliza a técnica de amostragem aleatória sistemática para determinar as principais direçõe presentes na nuvem de pontos. À partir destas direções encontradas, buscam-se os planos que as contém, que são identificados como os planos onde estão as superfícies da escada. A imagem em profundidade foi obtida através de um sensor laser scan instalado na cabeça do robô Nao e obtida a cada degrau. Apesar de os resultados deste trabalho serem bastante satisfatórios, a imagem em profundidade era composta principalmente pela escada.

Já em Harms et al. Harms et al. (2015), é proposto um algoritmo de detecção e modelagem de escadas utilizando uma visão estéreo como entrada. As imagens são obtidas de duas câmeras estéreo RIGs acopladas à um capacete. Estas imagens são processadas e transformadas em uma imagem em profundidade. O método proposto identifica as bordas de uma escada presente na imagem em profundidade, no qual cada degrau é representado por duas linhas (referentes à quina inferior e a quina

superior de um degrau). A escada é, então modelada através de um conjunto de degraus.

Em Delmerico et al. Delmerico et al. (2013), propõe-se um método de detecção e modelagem de escadas regulares, que identifica e analisa as descontinuidades contidas em uma *depth image* obtida de um *Kinect*. Primeiro, aplica-se um detector de bordas Canny para se evidenciar um conjunto de linhas paralelas das descontinuidades geradas pelos degraus. Após este passo, utiliza-se a transformada de Hough em 2D para identificar as retas da imagem. Estas são devidamente agrupadas e depois faz-se uma correspondência delas com a matriz *point cloud* da mesma imagem, obtendo-se os pontos referentes às bordas de cada degrau. Por fim, ajusta-se um plano pelos pontos identificados como referentes às bordas para determinar uma inclinação aproximada da escada. Os demais parâmetros necessários para a modelagem da escada, como a largura e profundidade dos degraus, podem ser obtidos pela relação geométrica entre os pontos agrupados de degraus distintos.

1.3.2 Plataforma ROS

O *framework* ROS (do ingês *Robot Operating System*) é uma plataforma flexível que auxilia o desenvolvimento de softwares com aplicações robóticas. Trata-se de um conjunto de bibliotecas, ferramentas e convenções com a finalidade de facilitar a tarefa de desenvolver o comportamento de robôs em multiplataformas.

O ROS foi desenvolvido de forma distribuída e modular e possui características similares à um sistema operacional, gerenciando a execução de processos, troca de mensagens entre os mesmos e um sistema de arquivos. Este permite a programação das suas rotinas e processos nas linguagens *Python* e *C++*, que são linguagens orientadas à objeto. Isto permite que os desenvolvedores segmentem seus programas em diversos processos e realizem a comunicação entre os mesmos por meio da troca de mensagens, inclusive entre computadores na mesma rede.

Em Loureiro Loureiro (2017), encontra-se uma boa definição desta plataforma, além de uma utilização da mesma. Também utiliza-se a funcionalidade de inicialização de processos do ROS, o *launch*. A execução de processos específicos com configurações pré-definidas podem ser configuradas em um arquivo XML (do inglês *eXtensible Markup Language*) e inicializada através de simples comandos.

1.4 Plataforma OctoMap

Este trabalho foi desenvolvido utilizando o *framework* Octomap, composto por um conjunto de ferramentas e bibliotecas com estruturas e funcionalidades com o objetivo de gerar um modelo volumétrico 3D do ambiente. O mapeamento feito por esta plataforma é baseada em *octrees*, sendo esta uma estrutura de dados em formato de árvore, onde cada nó interno possui interligação com oito nós filhos. Isto permite que o espaço tridimensional seja modelado e dividido recursivamente em espaços volumétricos menores, os *voxels*, até que seja alcançada a resolução desejada. O diagrama desta estrutura de dados pode ser vista em 1.2.

Em conjunto com esta estrutura de dados, o OctoMap utiliza uma estimação probabilística de ocupação para determinar a probabilidade de cada nó desta *octree* estar ocupado. Para isto, o estado de ocupação de um *voxel* é definido pela quantidade de vezes que o espaço referente a este *voxel* foi detectado como ocupado.

O OctoMap foi desenvolvido em *C++* e possui integração com o ROS. A utilização deste *framework* com ROS é feita através dos pacotes *octomap_ros*, *octomap_msgs* e *octomap_server*. Os pacotes *octomap_ros* e *octomap_msgs* fornecem funções de conversão e de serialização entre os tipos de dados utilizados no ROS e tipos de dados nativos do OctoMap. O pacote *octomap_server* possui funcionalidades para criação incremental de mapas 3D baseados em *octree* e para exercer o papel de servidor, sendo possível atualizar e incrementar o mapa 3D armazenado em um nó de *octomap_server* utilizando os dados obtidos de sensores.

A plataforma OctoMap já foi utilizada em diversas aplicações robóticas. Mais informações sobre as ferramentas e funcionalidades do OctoMap podem ser encontradas em Hornung et al. Hornung et al. (2013), e em seu sítio (https://octomap.github.io/).

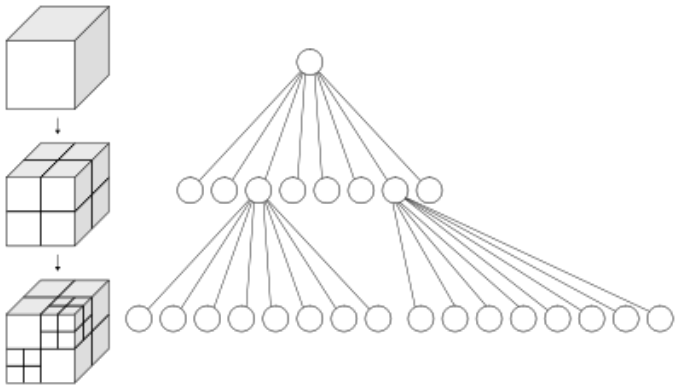


Figura 1.2: Diagrama de uma *octree*.

1.5 Estrutura do Texto

No Capítulo 1, foram apresentados a motivação deste trabalho e dos objetivos que este pretende alcançar. Além disso, foram apresentados o *framework* ROS, o robô EOD DIANE e o *framework* OctoMap, que são utilizados neste trabalho.

No Capítulo 2, o robô utilizado é descrito, bem como suas características fundamentais necessárias para o desenvolvimento do algoritmo de deteção e modelagem de escada e do controle são discutidas.

No Capítulo 3, apresenta-se a metodologia proposta para detectar e modelar a escada à partir de um conjunto de dados em *point cloud*, obtidos através do sensoreamento do ambiente ao redor do robô.

A integração do algoritmo de detecção com o algoritmo de controle do robô, bem como a integração com a interface gráfica feita em HTML são discutidas no Capítulo 4 deste trabalho.

No último Capítulo, o 5, apresenta-se os resultados e as conclusões acerca do trabalho, assim como os trabalhos em desenvolvimento e os futuros.

Capítulo 2

Robô móvel com esteiras

Neste trabalho, consideram-se robôs móveis com esteiras e braços articulados do tipo EOD. Sem perda de generalidade, será considerado o robô móvel DIANE desenvolvido pelo PEE COPPE/UFRJ, que pode ser visto na figura 2.1.

O robô DIANE é composto por duas esteiras nas laterais (uma de cada lado), 4 braços (2 frontais e 2 traseiros) com esteiras, que se movimentam na mesma velocidade das esteiras laterais, e um manipulador com três graus de liberdade, utilizado para o manejamento de objetos. A figura 2.2 mostra o robô manipulando um objeto.

A movimentação deste robô é determinada pelas velocidades de movimentação das esteiras laterais e, caso os braços estejam em contato com a superfície, dos braços. Estes também são utilizados para a transposição de obstáculos.

Figura 2.1: Comportamento do sensor Kinect de acordo com a distância.

Figura 2.2: Robô DIANE manipulando um objeto.

Originalmente, o DIANE carega um braço manipulador de 3 graus de liberdade. No entanto, este manipulador não será considerado neste trabalho, uma vez que a detecção da escada e a locomoção do robô independem do mesmo. Entretanto, deve-

se considerar que a posição do manipulador pode obstruir a captura de dados pelos sensores para a detecção da escada, podendo também alterar o centro de massa do robô, tornando instáveis certas posições do robô.

Durante a movimentação do DIANE no solo, o mesmo utiliza principalmente as esteiras laterais, que compõem como resultado as velocidades linear e angular do robô. Aplicando-se velocidades com o mesmo módulo e sentido nas esteiras, o robô é capaz de se movimentar para frente e para trás; aplicando-se velocidades com o mesmo módulo e sentidos diferentes, gera uma rotação do robô em torno do próprio eixo; e aplicando-se velocidades com módulos diferentes, permite que se realize um movimento em curva.

Na equação 2.1 , mostra-se o comportamento de movimentação do sistema robótico, no qual *v(t)* é a velocidade linear do mesmo, ω(t) é a velocidade angular, $\dot{\phi}_e$ trata-se da velocidade tangencial da esteira esquerda e $\dot{\phi}_d$ trata-se da velocidade tangencial da esteira direta. A figura 2.3 mostra a relação entre as velocidades.

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{\dot{\phi}_d}{2} + \frac{\dot{\phi}_e}{2} \\ \frac{\dot{\phi}_d}{2} - \frac{\dot{\phi}_e}{2} \end{bmatrix} \tag{2.1}$$

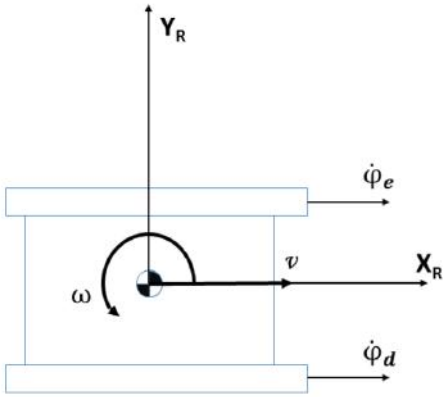


Figura 2.3: Movimentação das esteiras.

O robô também conta com os 4 braços para a transposição de obstáculos. Os pares de braços, frontais e traseiros, podem ser movimentados e posicionados de acordo com a necessidade da situação. Os braços são movimentados em pares, frontais e traseiros, o que aumenta a estabilidade do robô.

Define-se agora os sensores do robô, componentes responsáveis pela percepção e captura de informações do ambiente. Após os sensores, define-se o computador, componente principal responsável por gerenciar todos os demais componentes do robô e por interpretar os comandos enviados pelo controle e pelo operador.

2.1 Sensores utilizados

O robô DIANE possui 3 sensores: um *kinect*, utilizado para o mapeamento de distâncias, para a teleoperação e para determinar a inclinação do robô; e dois sensores *laser scan*, utilizados no mapeamento do ambiente ao redor do robô.

O *kinect* trata-se de um dispositivo RGB-D (do inglês, *Red Green Blue - Depth*) desenvolvido pela *Microsoft* em conjunto com a companhia Prime Sense, composto por uma câmera RGB, um projetor infravermelho, uma câmera infravermelho, além de um acelerômetro que fornece dados de inclinação do *kinect* em relação ao solo, cujas características da versão 1.0 estão descritas na tabela A.1, como mostrado na figura 2.4.

A comunicação deste instrumento com o computador é feita através de uma porta USB (do inglês, *Universal Serial Bus*). Sua utilização em aplicações robóticas de localização, mapeamento e detecção vem crescendo devido à sua alta variedade de informações e à sua acessibilidade. No robô DIANE, a câmera RGB deste dispositivo é utilizada para teleoperação e na metodologia de detecção de escadas descrita em Azevedo Azevedo (2017).

Este sensor também pode ser utilizado em conjunto com o ROS, através do pacote *freenect_launch*. Este pacote contém arquivos de *launch* para inicialização de nós que obtém os dados de sensoreamento do *Kinect* e os disponibilizam em tópicos. O principal arquivo de *launch* deste pacote, *freenect.launch*, pode ser executado através do comando 2.1. Após a execução deste comando, os dados em nuvem de pontos obtidos pelo *Kinect* ficam disponíveis no tópico */camera/depth/points*.

Código 2.1: Execução do launch freenect.launch

```
$ roslaunch freenect_launch freenect.launch
```

Figura 2.4: Detalhamento dos sensores do *Kinect*.
Fonte: de Lima De Lima (2016), pp.37

Os sensores *laser scan* são ferramentas amplamente utilizadas na robótica em aplicações como: posicionamento, navegação, mapeamento de ambientes, dentre outras. O sensor escolhido para realizar o mapeamento de ambiente foi o *VLP-16*, que pode ser visto na figura 2.5.

Este sensor foi desenvolvido pela *Velodyne LiDAR* e suas características podem ser encontradas na tabela A.2 e foi escolhido por ser compacto, possuir um amplo campo de visão tanto horizontal (360°) quanto vertical (±15°) e um longo alcance de medição (até 100m). Sua utilização permite mapear o ambiente ao redor do robô com uma alta precisão e sem a necessidade de movimentação do mesmo.

Este sensor também possui integração com o ROS, através do pacote *velodyne_pointcloud*. A inicialização dos nós e tópicos necessários para obtenção dos dados obtidos pelo modelo *Velodyne VLP-16* é feita através da execução do código 2.2, onde o arquivo de *launch VLP16_points.launch* é nativo do pacote *velodyne_pointcloud*. Após a inicialização destes nós, os dados em nuvem de pontos obtidos por este sensor ficam disponíveis no tópico */velodyne_points*.

Código 2.2: Execução do launch VLP16_points.launch

```
$ roslaunch velodyne_pointcloud VLP16_points.launch
```

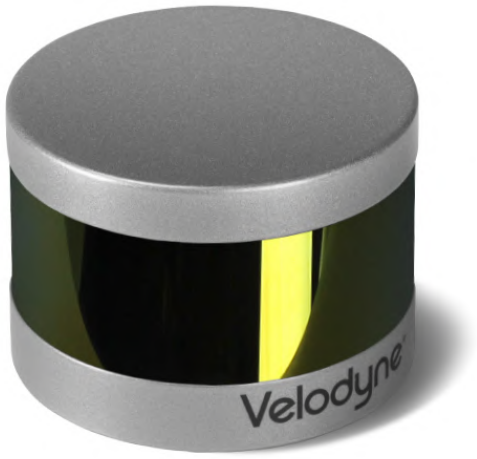


Figura 2.5: Velodyne VLP-16.
Fonte: http://velodynelidar.com

Por fim, o segundo sensor *laser scan* presente no robô DIANE é o *UST-10LX*, desenvolvido pela *Hokuyo*, cujas características estão descritas na tabela A.3. Este sensor é pequeno e leve, facilitando sua utilização em sistemas robóticos móveis, dado que quase não há impactos no peso e na movimentação do mesmo. A estrutura compacta dos dados fornecida por ele permite um rápido processamento de seus dados. As comunicações destes dois sensores com o computador utilizam o protocolo TCP/IP (do inglês, *Transmission Control Protocol/Internet Protocol*), através de um cabo Ethernet conectado à rede do DIANE.

Este sensor também possui integração com o ROS, através do pacote *hokuyo_node*. Após conectar o sensor *Hokuyo* no computador, é possível inicializar o nó do mesmo executando o arquivo de *launch* mostrado no código 2.3. Para utilizar este arquivo de *launch*, é necessário configurar no argumento *ip_address* o IP do *Hokuyo* utilizado.

Código 2.3: Inicialização do nó hokuyo_node

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
    <arg name="ip_address" default="172.16.0.10"/>
    <arg name="namespace" default="r"/>
    <arg name="manager" default=""/>

    <node ns="$(arg namespace)" pkg="urg_node" type="urg_node" name="urg_node"
    args="load urg_node $(arg manager) _ip_address:=$(arg ip_address) "/>
</launch>
```

Figura 2.6: Velodyne VLP-16.
Fonte: https://www.hokuyo-aut.jp/

2.2 Computador embarcado

Para gerenciar todos os componentes embarcados no DIANE, foi escolhida a placa *ADLQM67PC*, desenvolvida pela *ADL Embedded Solutions*. A arquitetura desta permite que dispositivos periféricos sejam conectados à um barramento PCI-e na posição vertical, o que reduz muito o espaço necessário no robô, comparado à um computador normal. Esta placa possui um processador Intel i7 da segunda geração e 4Gb de memória RAM, o que fornece um grande poder de processamento. Devido à essas características, esta se torna uma excelente opção de computador para sistemas de controle moderno que necessitem de um poderoso processamento para realizar tarefas complexas.

O computador possui, instalada nele, uma placa de interface com a rede CAN (do inglês, *Controller Area Network*), utilizada para a comunicação com os *drivers*

dos motores. Esta placa possui duas portas para comunicação com a rede CAN, e foi acoplada na placa *ADLQM67PC* por baixo da mesma, através de uma interface *PCI-express*. O computador também possui interfaces *Ethernet* e USB nativas, o que possibilita a comunicação com os sensores, no qual a interface *Ethernet* permite a comunicação por meio da rede interna do DIANE.

Figura 2.7: Placa ADLQM67PC.

Conectado à placa, encontra-se um SSD (do inglês, *Solid State Drive*), escolhido como o dispositivo de armazenamento devido a seu ótimo desempenho. Neste componente, as velocidades de leitura e escrita são muito superiores aos discos rígidos encontrados no mercado. Esta característica é essencial para aplicações de controle em tempo real, uma vez que permite que tarefas realizadas pelo computador sejam completadas em tempos muito menores. Além disso, como não possui partes mecânicas, seu funcionamento não é tão prejudicado por choques e vibrações, que ocorrem frequentemente no funcionamento de um robô do tipo EOD.

Para o sistema operacional, optou-se pelo Linux, mais especificamente a distribuição Ubuntu. Este sistema operacional foi selecionado devido ao seu suporte prolongado, robustez, grande comunidade colaborativa e acessibilidade. Além disso, a execução do framework robótico ROS, ferramenta utilizada como base para o desenvolvimento do software de controle do DIANE, é suportada por este sistema operacional. O Ubuntu é multi-tarefa e permite uma rápida resolução de tarefas e comunicação, essenciais para o controle em tempo real do DIANE.

Capítulo 3

Metodologia de detecção e modelagem de escadas

Neste trabalho, é apresentada uma solução para o problema de detecção e modelagem de escadas, onde cada etapa do método proposto é explicada e analisada separadamente. Esta metodologia foi apresentada de forma mais resumida em Chan et al. (2017).

Inicialmente, explica-se a Transformada de Hough, técnica utilizada na metodologia de detecção apresentada. Vale lembrar que o algoritmo proposto neste capítulo foi implementado em *C++* e baseado em ROS.

3.1 Transformada de Hough

A Transformada de Hough é um método frequentemente utilizado na detecção de linhas e círculos em conjuntos de dados bidimensionais, sendo uma das técnicas mais utilizadas em computação gráfica, como indicado por Hart em Hart (2009). Desde que o método foi desenvolvido e patenteado por Hough em Hough (1962), foram feitas diversas modificações no mesmo (Illingworth realizou uma pesquisa dessas alterações em Illingworth & Kittler (1988)). Uma generalização deste método para conjuntos de dados tridimensionais foi desenvolvida em Borrmann et al. Borrmann et al. (2011), além de conter um guia detalhado para a implementação.

Para a detecção de retas, cada linha é representada em coordenadas polares, como mostrado na equação 3.1, onde ρ corresponde à distância da origem até a linha e θ corresponde ao ângulo entre o eixo da coordenada X e a linha, como ilustrado na figura 3.1. Desta maneira, cada reta é definida por um par (ρ, θ), como utilizado em Kuo et al. Kuo et al. (2005). Esta parametrização evita que ocorram singularidades na representação de retas com inclinação de 90^o.

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{3.1}$$

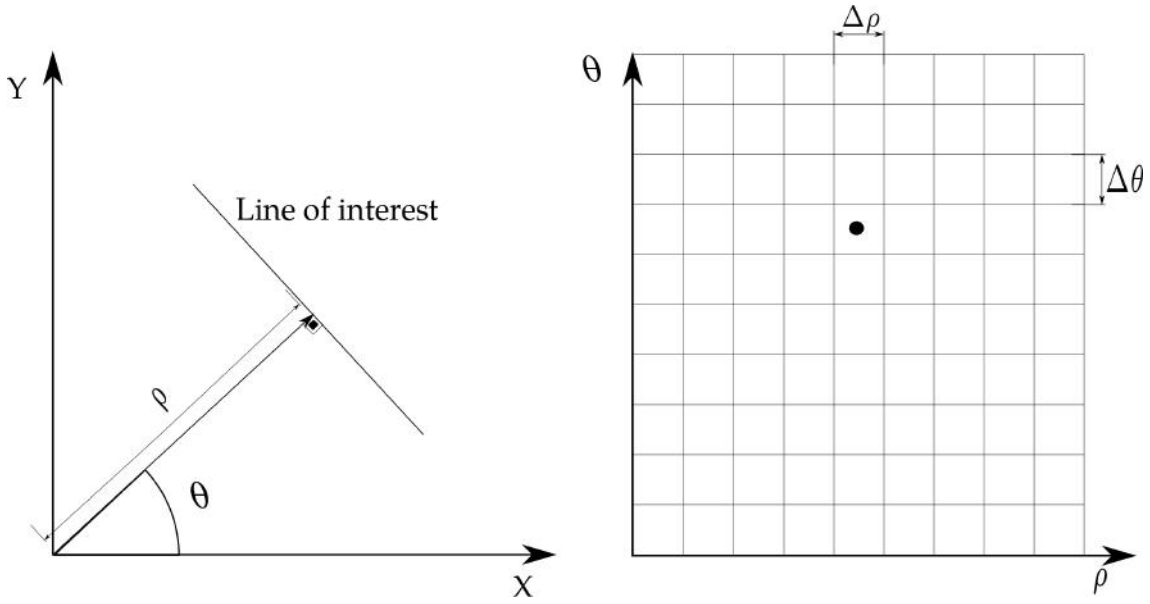


Figura 3.1: Mapeamento entre a representação de uma linha no espaço Euclidiano e sua representação no espaço de Hough.

Após a parametrização, é necessário definir o espaço de Hough (um espaço discreto) em (ρ, θ), onde $\rho_{min} \leq \rho \leq \rho_{max}$ e $\theta_{min} \leq \theta \leq \theta_{max}$. Este espaço é discretizado, formando uma grade na qual células adjacentes são igualmente espaçadas em ambas as coordenadas. As variações no eixo ρ, $\Delta\rho$, e no eixo θ, $\Delta\theta$, são definidas pelo usuário da metologia, e o resultado da discretização do espaço pode ser observado na figura 3.1. Note que cada reta no espaço Euclidiano é mapeada como um ponto no espaço de Hough e estará localizada dentro de uma das células da grade.

Utilizando os parâmetros calculados previamente, cria-se uma matriz de tamanho (N_ρxN_θ), chamada *acumulador*, onde N_ρ e N_θ correspondem à quantidade de divisões existentes nos eixos ρ e θ, respectivamente. O acumulador é utilizado no processo de votação. Neste processo, para cada célula do espaço de Hough, é verificado quantos pontos do espaço Euclidiano estão mais próximos da reta do que uma dada tolerância. Cada ponto, que esteja próximo o suficiente da reta referente à uma célula, é considerado um voto para a célula correspondente no acumulador.

Após o processo de votação, as células que não receberam uma quantidade mínima de votos são descartadas. As retas referentes às células que passaram por este filtro são consideradas como detectadas. O algoritmo 1 descreve o passo à passo deste processo.

Algorithm 1 Transformada de Hough

```
1: tol, min_votos ← definido pelo usuário
2: Δ_ρ, ρ_max, ρ_min ← definido pelo usuário
3: Δ_θ, θ_max, θ_min ← definido pelo usuário
4: N_ρ = (ρ_max − ρ_min)/Δ_ρ
5: N_θ = (θ_max − θ_min)/Δ_θ
6: A = array[N_ρ][N_θ]
7: retas = array vazio
8: for all cell(ρ, θ) no acumulador A do
9:     for all p em pontos do
10:        if distancia de p para a reta(ρ, θ) < tol then
11:            cell(ρ, θ)++
12:    if cell(ρ, θ) > min_votos then
13:        Cria a reta e adiciona a retas
14: return retas
```

3.2 Mapeamento do ambiente

O primeiro procedimento necessário para a detecção de escadas no ambiente é o mapeamento do mesmo. Para realizar a acquisição de dados, utilizou-se dois sensores: o sensor de profundidade *Kinect*; e o sensor *laser scan Velodyne LiDAR* (do inglês, *Light Detection And Ranging*) *VLP-16*.

Ambos os sensores são capazes fornecer dados do ambiente no formato de uma Nuvem de Pontos (conjunto de dados tridimensionais expresso em um mesmo sistema de coordenadas), à partir da qual será aplicada a lógica definida para a detecção de escadas.

3.2.1 Mapeamento por *Kinect*

O *Kinect* foi o primeiro sensor utilizado para mapeamento neste trabalho. Apesar de ruidoso, o dado retornado pelo *Kinect* é bastante acurado, sendo capaz de captar informações com boa precisão de objetos à distâncias entre $(0,8 \sim 4)$ metros, como mostrado na figura 3.2.

É importante ressaltar que o *Kinect* adota um sistema de coordenadas diferente do utilizado neste trabalho. Para este projeto, adota-se um referencial localizado no chão logo abaixo do *Kinect*, com o eixo Y apontando para a mesma direção da frente do sensor e com o eixo Z sendo vertical para cima. O sistema de coordenadas adotado pelo *Kinect* é mostrado na figura 3.3.

Devido a esta diferença, foi necessário aplicar uma transformação nos pontos mapeados pelo *Kinect*, modificando as coordenadas desses pontos para que sejam referentes ao referencial utilizado neste trabalho. Esta tranformação é composta por rotações dos eixos do sistemas de coordenadas e por uma translação.

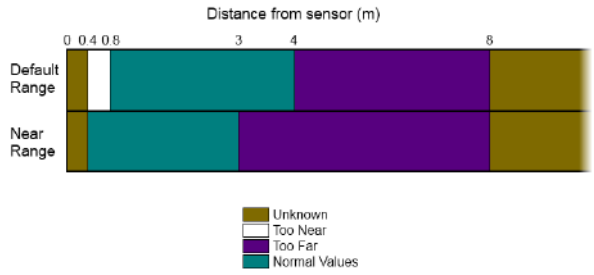


Figura 3.2: Comportamento do sensor Kinect de acordo com a distância.

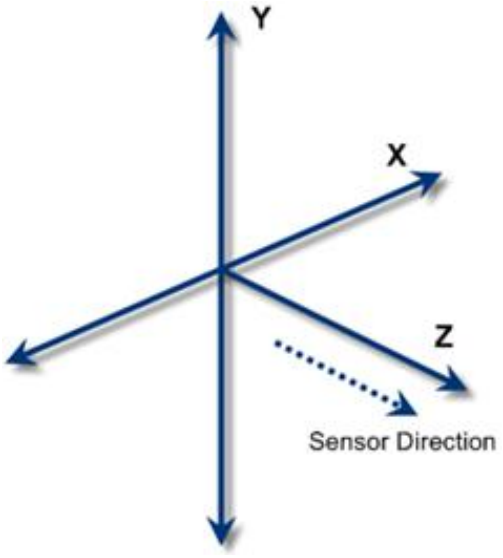


Figura 3.3: Sistema de coordenadas adotada pelo sensor Kinect.

Após realizar o ajuste no sistema de coordenadas dos dados retornados, obtém-se um conjunto de dados no formato de uma nuvem de pontos, como o mostrado na figura 3.4. Esta nuvem de pontos é então utilizada nas próximas etapas do algoritmo.

3.2.2 Mapeamento com o sensor *laser scan VLP-16*

O segundo sensor utilizado para o mapeamento do ambiente foi o sensor *laser scan LiDAR VLP-16*. Este envia feixes de luz e capta suas reflexões para calcular a sua distância até os objetos ao seu redor. Este equipamento possui um ângulo de varredura vertical de $\pm 15°$ (com uma resolução ângular de $2°$) e um ângulo de varredura horizontal de $360°$. Aliado a esse amplo campo de visão, este sensor é capaz de detectar objetos situados a distâncias de um a cem metros e possui uma precisão maior do que a do *Kinect*.

Para realizar a acquisição de dados, optou-se por utilizar este sensor na posição vertical, para que fossem obtidas mais informações pertencentes à uma escada localizada na frente do robô do que as pertencentes ao ambiente ao redor do robô. Dessa maneira, o modelo gerado para a escada é muito mais próximo do real, quando comparado com o resultado ao utilizar o sensor na posição horizontal. Por este motivo, também foi necessário aplicar um ajuste no sistema de coordenadas dos dados retornados para torná-los conpatíveis com o algoritmo de detecção.

Da mesma maneira que para o *Kinect*, obtém-se um conjunto de dados no formato de nuvem de pontos, como o mostrado na figura 3.5. Esta nuvem de pontos também pode ser utilizada nas próximas etapas do algoritmo.

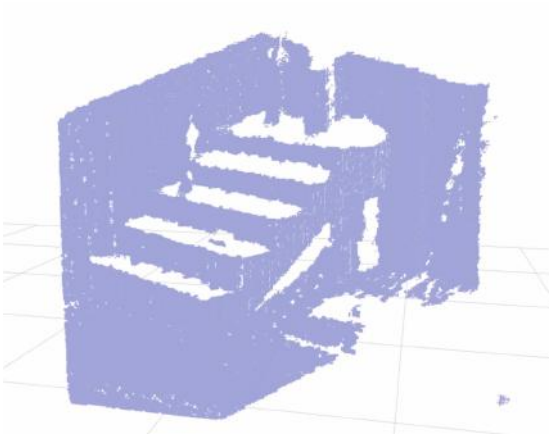


Figura 3.4: Nuvem de Pontos obtida do *Kinect*.

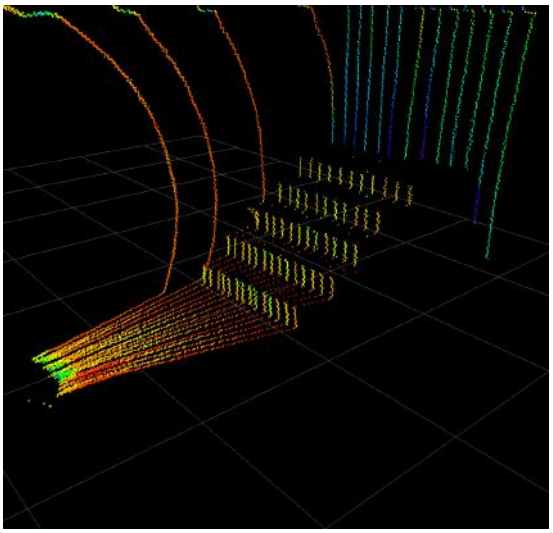


Figura 3.5: Nuvem de Pontos obtida do Laser *Velodyne VLP-16*.

3.3 Armazenamento dos dados

Nesta etapa, os dados obtidos no formato de nuvem de pontos dos sensores são transformados para um referencial comum, localizado no chão com o eixo Z positivo sendo vertical e para cima. Após a transformação, os pontos resultantes são armazenados em uma estrutura de *octree* utilizando o *framework* OctoMap.

Diferentemente de outras estruturas 3D, como a nuvem de pontos, o OctoMap é capaz de diferenciar áreas desconhecidas de áreas livres, sendo possível saber se um espaço ainda não foi explorado ou se não há nenhum obstáculo presente no mesmo. Além disso, com esta estrutura de dados, é possível coletar mapas com altas resoluções e, no momento da busca pela escada nesses dados, utilizar resoluções mais baixas. Ao reduzir a resolução no momento da busca, diminui-se a quantidade de dados a serem utilizados e, consequentemente, o processamento necessário.

Observa-se, na figura 3.6, o mapeamento obtido com resolução de 1 centímetro e, na figura 3.7, observa-se o mesmo mapeamento com uma resolução reduzida de 5 centímetros, utilizado na detecção da escada. Como para a detecção de escada não é necessário uma alta resolução comparado a outras tarefas, esta característica se torna um recurso importante para este trabalho.

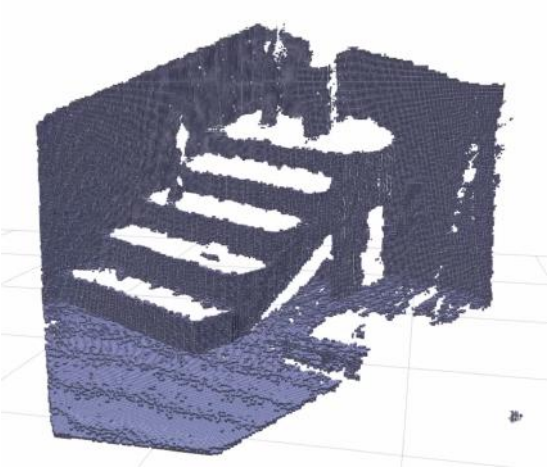


Figura 3.6: Mapeamento com resolução de 1 cm.

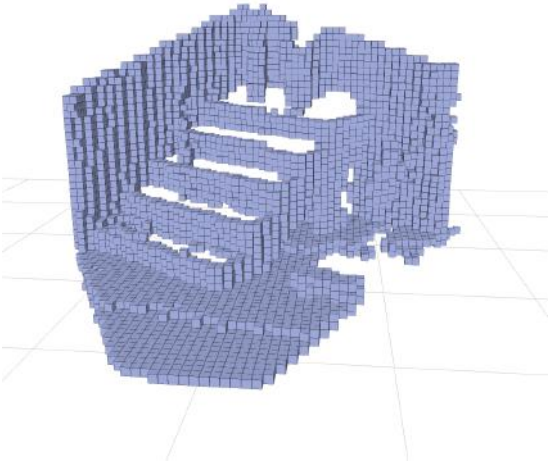


Figura 3.7: Mapeamento com resolução de 5 cm.

3.4 Algoritmo de detecção de escadas

O método proposto para a detecção de escadas pode ser dividido em 6 etapas principais: a inicialização e pré-processamento; a ordenação por Z e filtragem; a detecção de retas; o sequenciamento; a segmentação e ressequenciamento; e o pós-processamento.

O algoritmo 2 descreve o passo à passo do método implementado neste trabalho e uma ilustração dessas etapas pode ser visualizada na figura 3.8. As etapas são descritas a seguir.

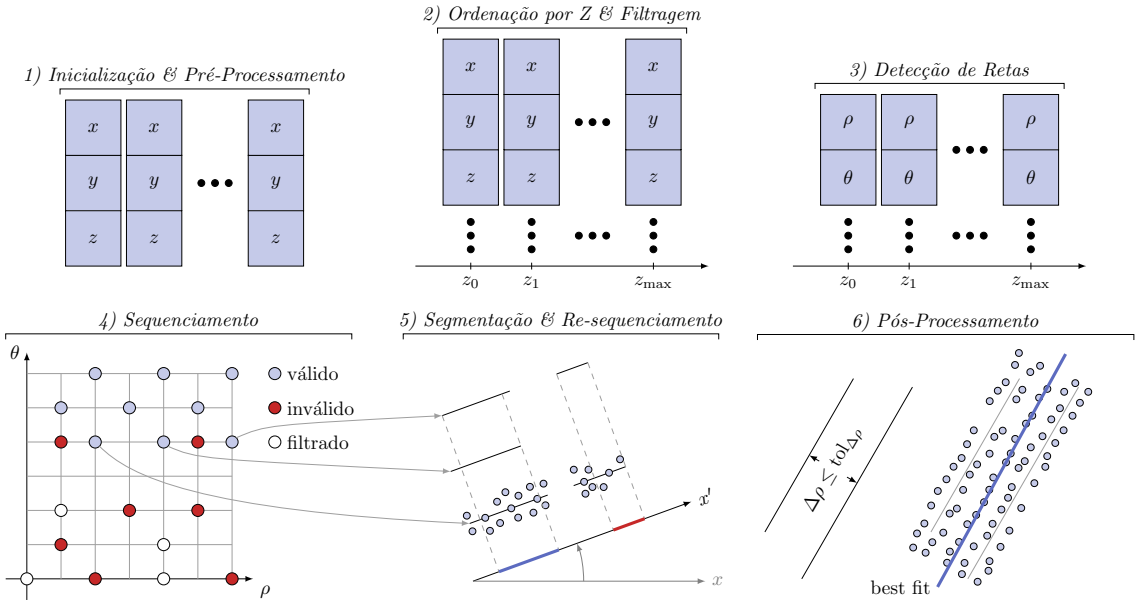


Figura 3.8: Ilustração das etapas que compõem o algoritmo de detecção de escada

Algorithm 2 Detecção de escada

Agrupar os pontos iniciais pelas coordenadas (X,Y) ▷ Pré-Processamento
Remover as colunas que contenham mais do que um número máximo de pontos
Pontos = Array multidimensional vazio ▷ Detecção de retas
Retas = Array vazio
Agrupar os pontos filtrados pela coordenada Z e armazenar em Pontos
for all Conjunto de pontos do array Pontos **do**
    Aplicar a Transformada de Hough
    Descartar as Retas que não receberam um número mínimo de votos
    Incluir as retas detectadas e filtradas em Retas
Planos = Array multidimensional vazio ▷ Sequenciamento
Agrupar as retas que possuam os mesmos parâmetros (ρ, θ)
Descartar os conjuntos que possuam menos do que a quantidade mínima de retas
Armazenar os conjuntos de retas válidos em Planos
Agrupar os planos contidos em Planos por θ, ordenados por ρ
Sequências = Array vazio
for all Conjunto de planos com o mesmo θ **do**
    Buscar sequências de planos espaçados igualmente em ρ
    Descartas as sequências que possuam menos do que 3 planos
    Armazenar as sequências válidas em Sequências
for all Sequência em Sequências **do** ▷ Popular os Planos
    for all Plano pertencente à Sequência **do**
        Buscar e armazenar os pontos iniciais próximos ou pertencentes ao Plano
        Ordena os pontos por suas coordenadas X
Escadas = Array vazio
for all Sequência em Sequências **do**
    for all Plano populado com pontos **do** ▷ Segmentação
        if Encontrar um espaçamento maior do que o tolerado **then**
            Criar um novo plano que conterá os pontos após o espaçamento
            Incluir esse novo plano populado na Sequência
            Remover do plano original os pontos contidos após o espaçamento
    Eixo X' = Eixo X rotacionado por $(\theta - 90°)$ ▷ Ressequenciamento
    Projetar os planos segmentados no Eixo X'
    Buscar as sequências de segmentos que possuem domínios similares em X'
    Descartar as sequências com menos do que 3 segmentos
    Armazenar as sequências válidas em Escadas
for all Sequência de segmentos em Escadas **do** ▷ Pós-Processamento
    Buscar os melhores parâmetros (ρ, θ) dos planos com mínimos quadrados

3.4.1 Inicialização e Pré-Processamento

Conforme descrito acima, os dados recebidos são obtidos no formato de uma nuvem de pontos. Esses dados são então processados e armazenados em uma estrutura de *octree* com uma alta resolução (1 centímetro) utilizando o *framework* OctoMap.

Após o armazenamento, diminui-se a resolução da *octree* para 5 centímetros. Além disso, aplica-se um filtro nesta *octree*, removendo os voxels correspondentes à espaços com coordenadas Z menores do que 0. Esse pré-processamento tem como objetivo reduzir o custo computacional do algoritmo, ao diminuir a quantidade de voxels necessários para representar o espaço mapeado e ao diminuir a quantidade de dados que devem ser considerados na detecção da escada. O resultado desse pré-processamento pode ser visualizado na figura 3.9. Nas próximas etapas, será considerado somente o conjunto de pontos correspondentes aos centros dos voxels que passaram pelo pré-processamento.

3.4.2 Ordenação por Z e Filtragem

Nesta etapa, os pontos que possuem as mesmas coordenadas (X,Y) são agrupados, de forma que cada um desses conjuntos representa uma coluna de pontos no espaço tridimensional. Os conjuntos, que possuem mais do que um número máximo de pontos ou menos do que um mínimo, são descartados, uma vez que possuem alturas diferentes das de um degrau padrão de uma escada.

Este filtro tem como objetivo remover as colunas de pontos correspondentes à uma parede ou à objetos mais baixos do que um degrau típico. Desta maneira, diminui-se ainda mais a quantidade de pontos que devem ser processados. O resultado desta etapa pode ser visualizado na figura 3.10.

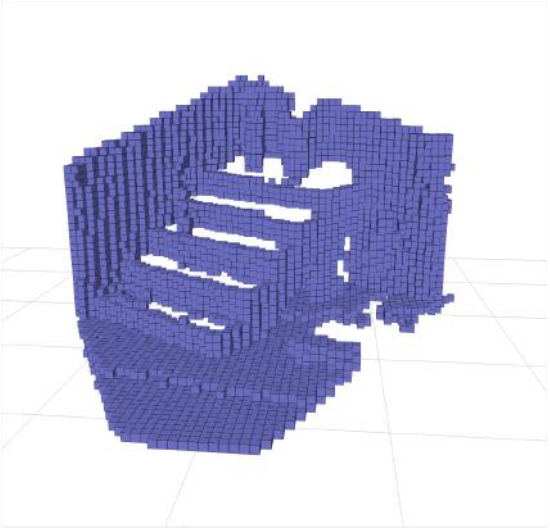


Figura 3.9: Resultado da Inicialização e do Pré-Processamento

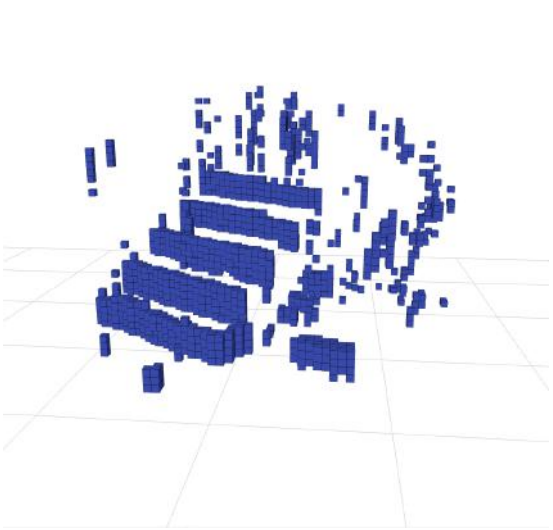


Figura 3.10: Resultado da Ordenação por Z e Filtragem

3.4.3 Detecção de Retas

Após a aplicação do filtro anterior, os pontos remanescentes são agrupados por altura, ou seja, por coordenada Z e armazenadas em *arrays*. Cada um desses conjuntos de pontos pode ser entendido como uma seção horizontal dos dados tridimensionais (3D). Isto permite que o algoritmo processe estes dados utilizando métodos bidimensionais (2D).

Aplica-se então o método da Transformada de Hough bidimensional em cada um desses conjuntos, para que sejam detectadas as retas contidas em uma mesma altura. Para cada conjunto de pontos, computa-se o parâmetro ρ_{max} utilizando a distância máxima da origem até um ponto do conjunto, e $0 \leq \theta \leq 2\pi$, para que o espaço de Hough seja determinado.

Durante o processo de votação da Transformada de Hough, as retas do espaço de Hough que não receberam um número mínimo de votos de

$$n_{minvotos} = w_{min}/\delta_{res} \tag{3.2}$$

são descartadas, onde $w_{min} > 0$ corresponde à largura mínima definida para um degrau e $\delta_{res} > 0$ é a resolucão utilizada para a *octree*. As informações de cada reta detectada são armazenadas em uma estrutura de dados, que contém os parâmetros polares (ρ, θ) e a coordenada Z da reta obtida. O resultado da detecção de retas pode ser visualizado na figura 3.11.

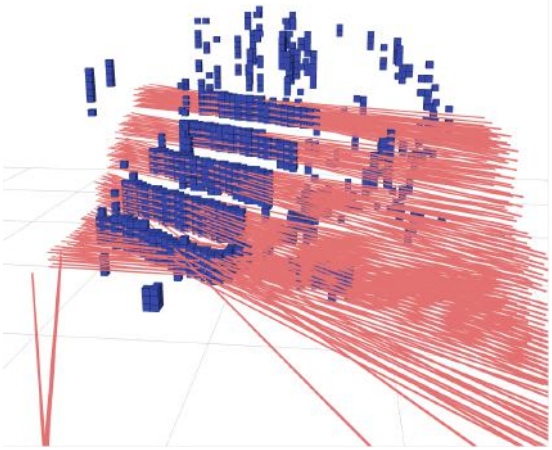


Figura 3.11: Resultado da Detecção de Retas com a Transformada de Hough

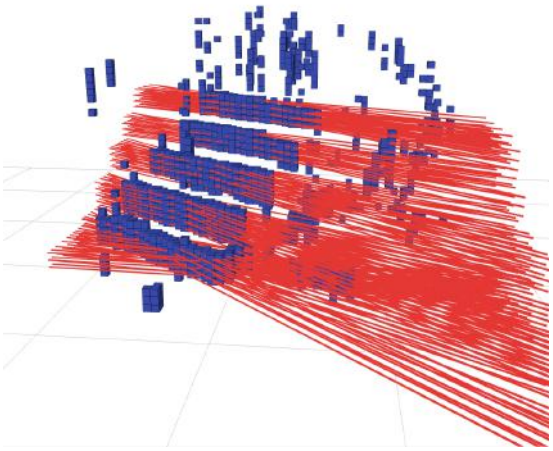


Figura 3.12: Resultado da filtragem das retas durante o Sequenciamento inicial

3.4.4 Sequenciamento

No início do processo de sequenciamento, as retas detectadas, que possuem o mesmo par de parâmetros polares (ρ, θ), são agrupadas. Cada um dos conjuntos resultantes de retas pode ser reconhecido indiretamente como o plano vertical que contém as retas do conjunto.

Destes conjuntos, aqueles que possuem um número de retas menor do que $n_{minlinhas}$ ou maior do que $n_{maxlinhas}$ são descartadas. Estes parâmetros são definidos pelo usuário e determinam indiretamente as alturas mínima e máxima dos degraus que podem ser detectados pelo algoritmo. Desta forma, evita-se que os conjuntos de retas, nos quais não serão encontrados degraus, continuem a ser considerados para o processamento. Na seção "Sequenciamento"da figura 3.8, os conjuntos filtrados são representados pelos círculos brancos. O resultado deste filtro pode ser visto na figura 3.12.

Após o processo de filtragem anterior, o par de parâmetros (ρ, θ) de cada conjunto é armazenado em uma nova estrutura de dados, assim como os valores mínimo e máximo da coordenada Z das linhas desse conjunto. Essa estrutura de dados pode ser entendida como o segmento do plano vertical, limitado somente na coordenada Z, que contém as retas contidas no conjunto.

A etapa final do processo de sequenciamento consiste em buscar sequências de conjuntos de retas que possam ser identificados como os planos frontais de uma escada. Desta maneira, para cada valor de θ no qual foram identificados esses conjuntos, identifica-se as sequências que sejam igualmente espaçadas em ρ por um valor $\delta_{min} \leq \delta_\rho \leq \delta_{max}$, onde δ_{min} e δ_{max} correspondem às profundidades mínima (15cm) e máxima (50cm) padrões de degraus de escadas. Cada sequência detectada nesta etapa terá um único valor de intervalo δ_ρ, um mínimo de três conjuntos e representará os planos verticais de um candidato à escada.

Este procedimento reduz bastante a quantidade de dados a serem analisados nas próximas etapas do algoritmo. Deve-se perceber que, se não existirem escadas a serem detectadas nos dados em nuvem de pontos iniciais, o algoritmo iria terminar o processamento de dados nesta etapa, o que fica evidente na seção "Sequenciamento"da figura 3.8.

3.4.5 Segmentação e Re-sequenciamento

Após obter as sequências, busca-se no conjunto de dados da *octree* inicial, os pontos que pertencem a cada segmento de plano contido em uma sequência válida. Considera-se que um ponto pertence a um plano se a sua menor distância para este plano é menor do que um valor limite. Este procedimento permitirá que o algoritmo segmente os planos de forma a diferenciar faces de objetos distintos.

A lógica de segmentação é aplicada para cada plano e inicia-se ordenando os mesmos por sua coordenada X. Estes pontos são então avaliados em ordem ascendente. Se a distância entre dois pontos subsequentes for maior do que um valor limite, δ_{seg}, ocorre uma segmentação para este plano. O segmento de plano original armazenará todos os pontos avaliados antes da segmentação e um novo segmento de plano é criado para armazenar os pontos restantes que ainda não foram analisados.

Cada novo segmento de plano criado é armazenado na mesma sequência do segmento de plano original e o procedimento é aplicado recursivamente até que o último ponto seja analisado. Este procedimento pode ser visualizado na seção "Segmentação & Re-sequenciamento"da figura 3.8.

Após aplicar a segmentação para todos os segmentos de planos de uma sequência, o algoritmo verifica se dentro desta sequência existe alguma subsequência que possa ser identificada como uma escada. Dado um eixo X' obtido ao rotacionar o eixo X por $(\theta - 90°)$, uma subsequência é válida se os seus segmentos de planos compartilham um domínio similar no eixo X', como mostrado na seção "Segmentação & Re-sequenciamento"da figura 3.8. Isto é verificado projetando-se os pontos dos segmentos de plano no eixo X'. Esta subsequência também deve cumprir os requisitos impostos para sequências na etapa de "Sequenciamento"do algoritmo.

Para cada subsequência encontrada, uma lógica de aglutinação de segmentos de planos é aplicada para juntar aqueles que possuem parâmetros ρ muito próximos, podendo ser considerados o mesmo segmento de plano. Neste procedimento, dois segmentos serão considerados o mesmo segmento se

$$|\rho_1 - \rho_2| \leq \delta_{res} \tag{3.3}$$

onde ρ_1 e ρ_2 correspondem às distâncias da origem até o primeiro e o segundo plano, respectivamente. Se a condição da equação 3.3 for satisfeita, um novo plano é criado para substituir os dois originais e o seu parâmetro ρ é dado por

$$\rho_3 = \frac{n_{p1}\rho_1 + n_{p2}\rho_2}{n_{p1} + n_{p2}} \tag{3.4}$$

onde n_{p1} e n_{p2} correspondem ao número de pontos contidos no primeiro e no segundo plano, respectivamente. Este procedimento reduz o número de estruturas que representam o mesmo degrau dentro da escada. Após este procedimento, cada subsequência é submetida novamente ao filtro aplicado na etapa de "Sequenciamento"do algoritmo e toda subsequência que passar nas verificações realizadas é considerada uma escada. O resultado deste processo pode ser visualizado em 3.13.

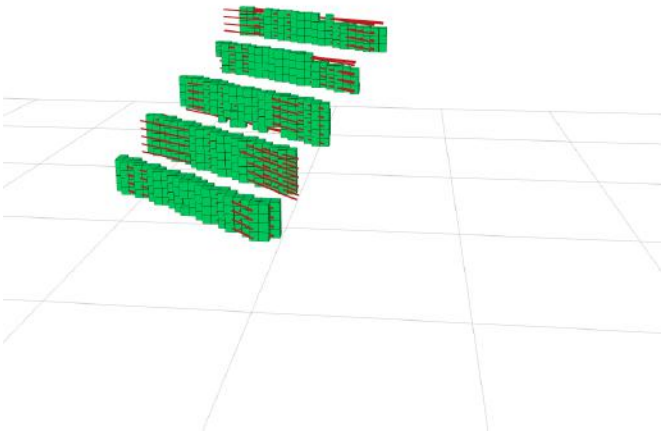


Figura 3.13: Resultado da segmentação com os pontos detectados

3.4.6 Pós-Processamento

O objetivo principal da etapa de pós-processamento é melhorar os valores dos pares de parâmetros (ρ, θ) obtidos para cada degrau das escadas detectadas. Deste modo, aplicou-se o método de mínimos quadrados em cada degrau das escadas detectadas, obtendo-se a melhor reta definida pelos pontos de cada degrau. Uma ilustração do processo realizado para cada degrau pode ser encontrado na seção "Pós-Processamento"da figura 3.8.

Após o método ser aplicado para todos os degraus de uma escada, o par de parâmetros (ρ, θ) otimizado para a mesma é obtido ao calcular-se a média aritmética destes parâmetros. Este método garante que os parâmetros, obtidos para cada escada, estão otimizados de acordo com os pontos existentes nela.

3.5 Modelagem das escadas

O processo de modelagem das escadas detectadas apresentado neste trabalho considera que as mesmas são retas, regulares e que possuem somente um lance, ou seja, não apresentam níveis intermediários entre o piso inferior e superior.

O método consiste em descobrir as propriedades das escadas detectadas no algoritmo de detecção apresentado, tais como a profundidade total (d_{total}), a largura total (w_{total}), a altura total (h_{total}), a profundidade média dos degraus (d_{mean}) e a altura média dos degraus (h_{mean}). Para facilitar a notação, define-se P_k como o conjunto de todos os pontos $p = [p_x \ p_y \ p_z]^{\mathrm{T}}$ contidos em um degrau da escada, e $P := \bigcup P_k$, sendo P o conjunto de todos os pontos existentes na escada.

3.5.1 Cálculo das propriedades das escadas

Para cada escada detectada na etapa anterior, os pontos contidos nos degraus da mesma são extraídos e armazenados em P, que serão utilizados no cálculo de suas propriedades. Define-se também n_{stairs} como o número de degraus detectados na escada, onde considera-se que cada segmento de plano do conjunto final corresponde à um degrau.

A altura total, h_{total}, é calculada como a diferença entre os valores máximo e mínimo da coordenada Z dos pontos $p \in P$, sendo dada por:

$$h_{total} = \max(p_{1z}) - \min(p_{2z}),\ \forall p_1 \in P_1, p_2 \in P_{n_{stairs}} \tag{3.5}$$

A altura média dos degraus, h_{mean}, é obtida ao dividir a altura total da escada pelo número de degraus da mesma, logo:

$$h_{mean} = h_{total}/n_{stairs} \tag{3.6}$$

Para obter as propriedades de profundidade da escada, é calculada a distância, *dist*, do centróide do primeiro segmento de plano ao último plano. A profundidade média dos degraus, d_{mean}, é obtida ao dividir essa distância pelo número de degraus da escada com exceção do último. Desse modo:

$$d_{mean} = dist/(n-1) \tag{3.7}$$

A profundidade total da escada, d_{total}, é dada pelo resultado da multiplicação da profundidade média dos degraus pelo número total de degraus, logo:

$$d_{total} = d_{mean}\ n_{stairs} \tag{3.8}$$

Para realizar o cálculo da largura total da escada, w_{total}, utilizou-se os seguintes três passos. O primeiro passo consiste em identificar os pontos $p_1 \in P_1$ e $p_2 \in P_{n_{stairs}}$ que fazem com que $||p_1 - p_2||$ seja maximizado. É importante notar que estes dois pontos estarão localizados em lados opostos da escada. Em seguida, estes dois pontos são projetados no plano da base da escada, e os pontos resultantes da projeção são projetados no plano vertical que passa pelo centróide do primeiro degrau. A largura total da escada, w_{total}, é considerada como a distância entre os dois pontos resultantes deste procedimento.

Os dois pontos resultantes projetados também representam os pontos limitantes da aresta da escada conectada ao chão. Portanto, estes dois pontos são utilizados como os pontos da aresta inicial para a modelagem da mesma. O próximo par de pontos é obtido ao mover o par inicial na direção do eixo Z pelo valor h_{mean}. Em

seguida, a aresta que une o primeiro degrau ao segundo pode ser obtida ao transladar a anterior na direção de ρ pelo valor d_{mean}. Este processo é repetido até que todos os degraus da escada sejam modelados. O resultado deste procedimento pode ser visualizado nas figuras 3.14 e 3.15.

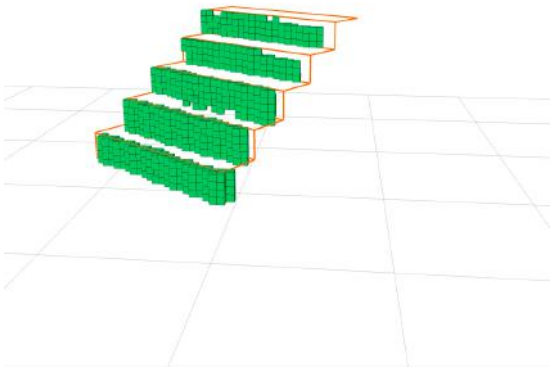


Figura 3.14: Escada modelada com os pontos pertencentes à escada

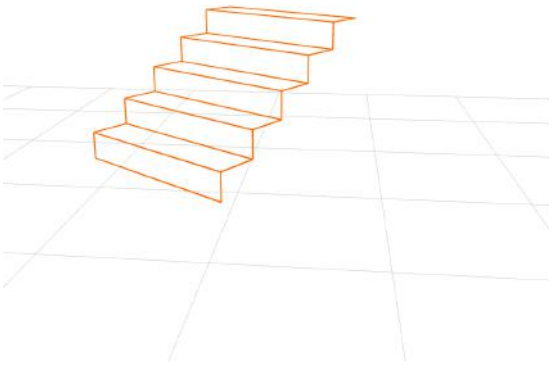


Figura 3.15: Escada modelada

Capítulo 4

Implementação do algoritmo

Conforme citado anteriormente, o algoritmo de detecção e modelagem de escadas foi implementado em *C++* e baseado em ROS, tendo sido criado um pacote de ROS para armazenar as funções desenvolvidas, chamado *diane_octomap*. Com este pacote, busca-se facilitar o consumo da funcionalidade de detecção e de modelagem apresentada neste trabalho, além de torná-lo modular para reutilização em outros projetos que necessitem da mesma.

Neste capítulo são apresentados detalhes sobre o pacote implementado, assim como sua utilização em uma interface gráfica Web utilizada na operação do sistema robótico DIANE.

4.1 Pacote *diane_octomap*

O pacote *diane_octomap* concentra as informações e metodologias referentes à detecção e modelagem de escadas à partir de dados presentes em uma nuvem de pontos. Esta nuvem de pontos é obtida através de um sensor *Kinect* e de um sensor *laser scan Velodyne VLP-16*.

A utilização deste pacote permite a obtenção de propriedades das escadas existentes em um ambiente mapeado, podendo ser aplicado em projetos robóticos que necessitem desta funcionalidade. Em Silva Silva (2017), o modelo de escada obtido utilizando o pacote *diane_octomap* é usado durante o controle semiautônomo de subida de escadas definido para o robô DIANE. É importante lembrar que sua utilização deve ocorrer em conjunto com um processo do *octomap_server*, caso contrário não é possível obter os dados do ambiente armazenados em uma *octree* para a execução do algoritmo.

Apresenta-se a seguir os parâmetros de inicialização, tópicos e serviços existentes no *diane_octomap*, necessários para a utilização deste pacote de forma efetiva.

4.1.1 Parâmetros de inicialização

A iniciliazação do nó do *diane_octomap* permite a configuração dos parâmetros utilizados nos passos do algoritmo de detecção e modelagem de escadas. Isto permite que o usuário deste pacote adapte sua utilização para detectar escadas retas e regulares que não estejam dentro padrão.

A seguir é apresentada a lista de parâmetros configuráveis durante a inicialização do nó *diane_octomap*.

1. **Parâmetros de inicialização:**
   (a) **Ordenação por Z e Filtragem:**
      - **ColumnMinSize**: Quantidade mínima de pontos em uma coluna de pontos para não ser considerada um ruído.
      - **ColumnMaxSize**: Quantidade máxima de pontos em uma coluna de pontos para não ser considerada uma parede.

   (b) **Detecção de Retas:**
      - **RhoStep**: Passo de ρ, em metros, utilizado para a definição do Espaço de Hough.
      - **ThetaStep**: Passo de θ, em graus, utilizado para a definição do Espaço de Hough.
      - **AccumulateDistTolerance**: Distância máxima para que o ponto avaliado seja considerado no processo de votação da Transformada de Hough.
      - **MinAmountVotes**: Quantidade mínima de votos que uma célula do Acumulador deve ter para que seja considerada uma reta detectada.

   (c) **Sequenciamento:**
      - **MinNumberLines**: Quantidade mínima de retas que o conjunto de retas agrupadas por (ρ, θ) deve ter para que seja considerado válidas.
      - **MaxNumberLines**: Quantidade máxima de retas que o conjunto de retas agrupadas por (ρ, θ) deve ter para que seja considerado válidas.

   (d) **Segmentação e Re-sequenciamento**
      - **SegmentationTol**: Distância máxima em X tolerada para que os pontos de um plano sejam considerados do mesmo segmento.
      - **IntervalTol**: Distância máxima tolerada nos domínios em X' para que os segmentos de planos sejam considerados do mesmo conjunto.
      - **MinNumSteps**: Número mínimo de segmentos de planos em um conjunto para que seja considerado uma escada.

- **MergeDeltaRhoTol**: Diferença mínima entre os valores do parâmetro ρ de dois segmentos de retas para que não sejam considerados o mesmo segmento.
- **SequenceMinSegmentDist**: Distância mínima entre dois segmentos de planos de um conjunto para que este seja considerado um candidato à escada.
- **SequenceMaxSegmentDist**: Distância máxima entre dois segmentos de planos de um conjunto para que este seja considerado um candidato à escada.

(e) **Modelagem das escadas:**

- **ModellingMinStepHeight**: Altura mínima do degrau para que o modelo de escada seja considerado válido.
- **ModellingMaxStepHeight**: Altura máxima do degrau para que o modelo de escada seja considerado válido.
- **ModellingMinStepDepth**: Profundidade mínima do degrau para que o modelo de escada seja considerado válido.
- **ModellingMaxStepDepth**: Profundidade máxima do degrau para que o modelo de escada seja considerado válido.

4.1.2 Tópicos

Apresenta-se nesta seção uma lista dos tópicos do pacote *diane_octomap*, para fins de análise de sua funcionalidade:

1. **Tópicos:**

   (a) **Publicados:**

   - **diane_octomap/All_Modeled_Stairs_Info**: Publica informações dos modelos de escadas obtidos pela execução do algoritmo de detecção e modelagem.
   - **diane_octomap/Filtered_Hough_Lines**: Publica as informações visuais das retas detectadas e que passaram pelo filtro de aplicado na etapa de Sequenciamento.
   - **diane_octomap/First_Filtered_Points**: Publica as informações visuais dos pontos que passaram pelo filtro de quantidade de pontos na coluna.
   - **diane_octomap/Hough_Lines**: Publica as informações visuais das retas detectadas pela etapa de Detecção de Retas.

- **diane_octomap/Modeled_Stairs_Visualization_Markers**: Publica informações visuais dos modelos de escadas obtidos pela execução do algoritmo de detecção e modelagem.
- **diane_octomap/Sequenced_Plane_Segments**: Publica informações visuais dos conjuntos de segmentos de planos sequenciados obtidos na etapa de Segmentação e Re-sequenciamento.
- **diane_octomap/Stair_Model_Points**: Publica informações visuais dos pontos pertencentes aos modelos de escadas obtidos pela execução do algoritmo de detecção e modelagem.

(b) **Subscritos:**

- **diane_octomap/Start_Visualization_Publishes**: Inicializa as publicações das informações visuais.

4.1.3 Serviços

Nesta seção, são apresentados uma lista dos serviços deste pacote. A utilização do algoritmo de detecção e modelagem de escada é feito através dos serviços disponibilizados por este pacote. Utilizando os serviços, também é possível configurar e consultar os valores dos parâmetros utilizados pelo algoritmo de detecção e modelagem de escada.

Conforme citado anteriormente, é necessário que exista um processo ativo de um *octomap_server* ao se utilizar o pacote *diane_octomap*, caso contrário não será possível obter os dados do ambiente no formato de uma *octree*.

1. **Serviços:**

(a) **Servidor:**

- **diane_octomap/Detect_Stairs_From_Server**: Inicializa a execução do algoritmo de detecção e modelagem de escadas, obtendo a octree existente no *octomap_server* no momento da requisição.
- **diane_octomap/Get_Detect_Stairs_Parameters**: Consulta os valores atuais de todos os parâmetros de inicialização utilizados pelo algoritmo de detecção e modelagem de escadas.
- **diane_octomap/Set_Detect_Stairs_Parameters**: Configura os valores de todos os parâmetros de inicialização utilizados pelo algoritmo de detecção e modelagem de escadas.

(b) **Cliente:**

- **octomap_full**: Serviço do pacote *octomap_server* que disponibiliza a *octree* existente neste nó contendo os dados do ambiente, que foram obtidos pelos sensores.

Ao ser realizada uma requisição para o serviço *diane_octomap/Detect_Stairs_From_Server*, o nó do *diane_octomap* requisita ao nó do *octomap_server* a *octree* contendo as informações do ambiente. Caso existam informações na *octree* recebida, esta é utilizada durante todo o processo de detecção e modelagem de escadas. No final desta execução, todas as informações dos modelos das escadas detectadas são retornadas ao processo que realizou a requisição.

4.1.4 Utilização do pacote

A inicialização do nó de ROS do *diane_octomap* pode ser realizada utilizando um arquivo de *launch* chamado *diane_octomap.launch*, no qual é possível realizar as configurações dos parâmetros de inicialização apresentados anteriormente. A execução do arquivo de *launch diane_octomap.launch* para inicialização do nó do *diane_octomap* pode ser feita através da execução do comando no código 4.1. O conteúdo deste arquivo de *launch* pode ser visto no código 4.4.

Código 4.1: Execução do arquivo de launch *diane_octomap.launch*

```
$ roslaunch diane_octomap diane_octomap.launch
```

Conforme citado anteriormente, cada parâmetro de inicialização influencia em uma determinada etapa do algorítmo. Desta forma, os efeitos da alteração de um parâmetro podem ser visualizados através dos tópicos de visualização deste mesmo pacote. É importante lembrar que na versão atual deste pacote, não é possível configurar a resolução da *octree* utilizada pelo algoritmo, sendo esta resolução fixada em 5 centímetros. Por este motivo, os parâmetros de inicialização devem ser configurados de acordo com esta resolução.

Como citado anteriormente, a utilização deste pacote deve ser feita em conjunto com um processo do *octomap_server*, que, por sua vez, deve estar recebendo as informações obtidas pelos sensores. A inicialização do nó do *octomap_server* pode ser feita utilizando outro arquivo de *launch* chamado de *octomap_server_main.launch*, no qual alguns parâmetros de inicialização deste nó podem ser definidos. A execução do arquivo de *launch octomap_server_main.launch* pode ser feita através da execução do comando no código 4.2. O conteúdo deste arquivo de *launch* pode ser visto no código 4.5. As definições dos parâmetros utilizados para a inicialização do nó do *octomap_server* podem ser encontradas na documentação oficial deste pacote em *http://wiki.ros.org/octomap_server*.

Código 4.2: Execução do arquivo de launch *octomap_server_main.launch*

```
$ roslaunch diane_octomap octomap_server_main.launch
```

Neste arquivo, também são definidas as inicializações da transformação do referencial do *Velodyne VLP-16* para o referencial do *Kinect* e da transformação do referencial do *Kinect* para o referencial utilizado pelo algoritmo, localizado no chão. É importante ressaltar que os parâmetros utilizados para essas transformações devem ser configuradas de acordo com as condições do experimento, levando-se em conta as posições e orientações dos sensores utilizados. Os parâmetros das transformações presentes no arquivo foram determinados para uma situação específica, sendo necessário alterar seus valores de acordo com o experimento para a obtenção de um mapa correto.

Após realizar as inicializações dos nós de ROS do *diane_octomap*, do *octomap_server*, do *Kinect*, do *Velodyne VLP-16* e das transformações de referencial, pode-se utilizar as funcionalidades implementadas no pacote *diane_octomap*. Para executar a detecção e modelagem das escadas presentes no ambiente detectado pelos sensores, deve-se realizar uma requisição ao serviço *diane_octomap/Detect_Stairs_From_Server*, que por sua vez irá retornar para o requisitante as informações e propriedades de todas as escadas identificadas e modeladas. A execução desta requisição ao serviço *diane_octomap/Detect_Stairs_From_Server* pode ser feita através da execução do comando no código 4.3.

Código 4.3: Requisição ao serviço *diane_octomap/Detect_Stairs_From_Server*

```
$ rosservice call /diane_octomap/Detect_Stairs_From_Server "{}"
```

Código 4.4: diane_octomap.launch

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>

    <arg name="manager" default="diane_manager"/>
    <arg name="namespace" default="r"/>

    <node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="$(arg manager)"
    args="manager"/>


    <node ns="$(arg namespace)" pkg="nodelet" type="nodelet" name="diane_octomap"
    args="load diane_octomap/DianeOctomapNodelet $(arg manager)">

    <!--Z-Sorting And Filtering's Parameters-->
        <param name="ColumnMinSize" type="int" value="3" />
        <param name="ColumnMaxSize" type="int" value="6" />


    <!--Line Detection's Parameters-->
        <param name="RhoStep" type="double" value="0.05" />
        <param name="ThetaStep" type="double" value="5" />

        <param name="AccumulateDistTolerance" type="double" value="0.025" />
        <param name="MinAmountVotes" type="int" value="10" />


    <!--Sequencing Parameters-->
        <param name="MinNumberLines" type="int" value="3" />
        <param name="MaxNumberLines" type="int" value="6" />


    <!--Segmentation and Resequencing's Parameters-->
        <param name="SegmentationTol" type="double" value="0.20" />

        <param name="IntervalTol" type="double" value="0.50" />

        <param name="MinNumSteps" type="int" value="3" />

        <param name="MergeDeltaRhoTol" type="double" value="0.11" />

        <param name="SequenceMinSegmentDist" type="double" value="0.23" />
        <param name="SequenceMaxSegmentDist" type="double" value="0.38" />


    <!--Stair Modelling's Parameters-->
        <param name="ModellingMinStepHeight" type="double" value="0" />
        <param name="ModellingMaxStepHeight" type="double" value="0.23" />
        <param name="ModellingMinStepDepth" type="double" value="0.24" />
        <param name="ModellingMaxStepDepth" type="double" value="0.36" />

    </node>

</launch>
```

Código 4.5: octomap_server_main.launch

```
<?xml version="1.0" encoding="utf-8"?>
<launch>

    <!--LASER-KINECT TRANSFORM-->

    <node pkg="tf" type="static_transform_publisher"
    name="simple_tf_laser_broadcaster_node" args="0.05 -0.35 0.05
    1.57079632679 0 3.1415 velodyne camera_depth_optical_frame 100" />

    <!--KINECT-MAP TRANSFORM-->

    <node pkg="tf" type="static_transform_publisher"
    name="simple_tf_kinect_broadcaster_node"
    args="0 0.38 0 0 0 1.57079632679 camera_depth_optical_frame map 100" />

    <!--OCTOMAP SERVER-->

    <node pkg="octomap_server" type="octomap_server_node"
    name="octomap_server_node_laser" output='screen'>
        <param name="frame_id" type="string" value="/map" />
        <param name="resolution" value="0.01" />
        <param name="height_map" type="bool" value="true" />
        <param name="sensor_model/max_range" value="-1" />
        <param name="latch" type="bool"  value="false" />
        <param name="sensor_model/max_range" value="100.0" />
        <param name="height_map" value="false" />
        <param name="color/r" value="0.62" />
        <param name="color/g" value="0.66" />
        <param name="color/b" value="0.85" />
        <param name="color/a" value="1" />

        <remap from="cloud_in" to="/velodyne_points" />

        <remap from="cloud_in" to="/camera/depth/points" />

    </node>

</launch>
```

4.2 Utilização através da interface gráfica Web

Apresenta-se agora um exemplo de utilização do pacote *octomap_server* desenvolvido neste trabalho. As funcionalidades deste pacote foram integradas à uma interface gráfica Web, utilizada para operar o sistema robótico DIANE, sem a necessidade da instalação do sistema operacional Ubuntu no computador do operador. A interface gráfica Web utilizada pode ser vista na figura 4.1.

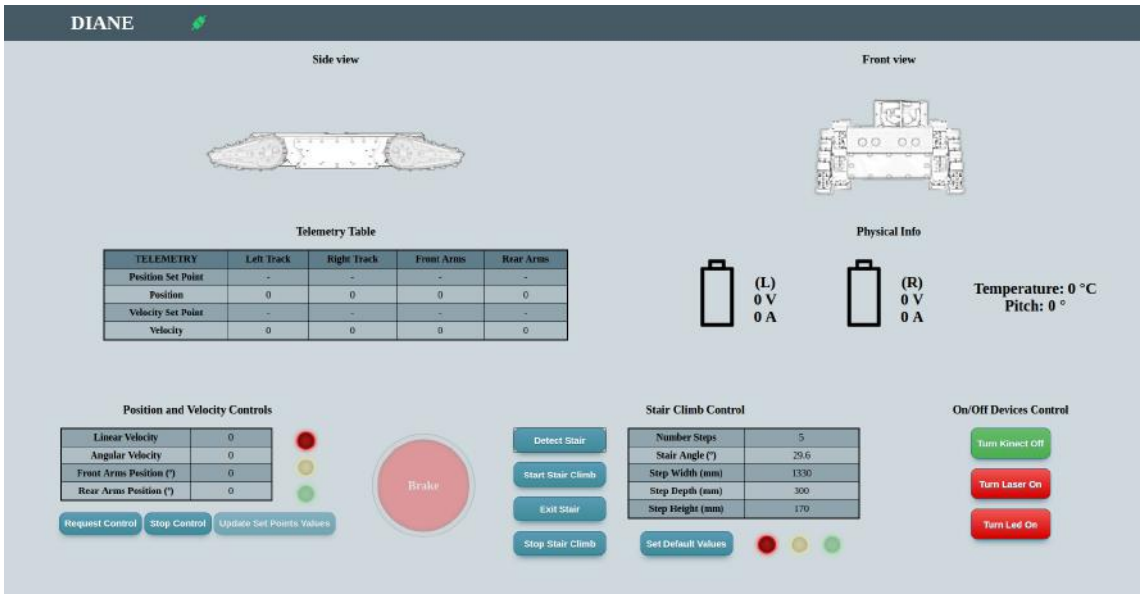


Figura 4.1: Interface Gráfica Web para operação do DIANE

Nesta aplicação, as rotinas de ROS, ou *nós*, do *diane_octomap*, do *octomap_server* e do *Kinect* são executadas no computador embarcado do DIANE, em paralelo aos processos referentes ao *software* de controle deste robô.

A nuvem de pontos contendo as informações do ambiente, obtida utilizando o *Kinect*, é transmitida para o nó do *octomap_server*. Essas informações são processadas e armazenadas em uma *octree*, que, por fim, é utilizada pelo algoritmo de detecção do nó do *diane_octomap*.

Na interface gráfica Web, implementou-se uma região dedicada ao controle de subida de escadas, que pode ser vista com mais detalhes na figura 4.2. Nesta região, encontra-se um botão *"Detect Stair"* e uma tabela de dados com campos referentes às informações de escada modelada. Estas são utilizadas para requisitar a execução do algoritmo de detecção e para mostrar as propriedades da escada modelada ao operador, respectivamente.

Desta forma, quando o operador pressiona o botão "*Detect Stair*", uma requisição ao serviço *diane_octomap/Detect_Stairs_From_Server* é enviada, iniciando a execução da detecção. Ao final do procedimento, as propriedades das escadas modeladas são retornadas para a interface, que repassa essas informações ao operador

através da tabela desta região, como pode ser visto na figura 4.3. As propriedades das escadas modeladas são posteriormente utilizadas no controle de subida de escadas deste robô, que pode ser encontrado em Silva Silva (2017).

Figura 4.2: Sessão de detecção de escadas - Botão para iniciar a detecção

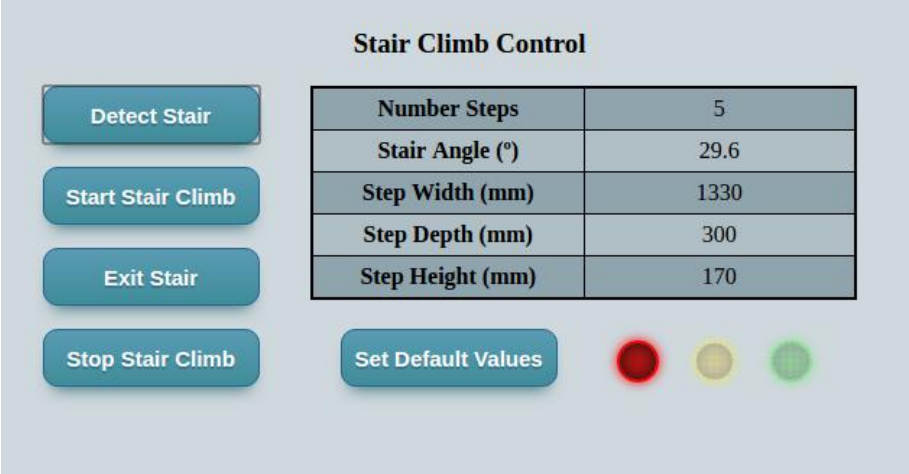


Figura 4.3: Resultado da detecção de escadas

Capítulo 5

Resultados Experimentais da Detecção de Escada

Para validar a robustez e a aplicabilidade das metodologias de detecção e modelagem propostas neste trabalho, foram realizados experimentos em diferentes cenários.

Discute-se, neste capítulo, os resultados experimentais obtidos de 3 escadas mapeadas utilizando o *Kinect*, de duas delas mapeadas utilizando o sensor *Velodyne VLP-16*, que podem ser encontradas na figura 5.1. Por fim, analisa-se também o resultado obtido do mapeamento de uma escada feito utilizando os dois sensores simultaneamente.

5.1 Resultados

Como citado anteriormente, o *Kinect* trabalha a distâncias entre 0,8 m e 4 m. Devido à esta característica e aliada ao fato de as escadas se encontrarem a distâncias superiores a 4 m, observa-se a presença significativa de ruídos nas nuvens de pontos obtidas com este sensor, como mostrado na figura 5.2.

Os dados da nuvem de pontos obtidos dos sensores são transformados para um referencial localizado no chão e os pontos que estão no chão ou abaixo dele são descartados, reduzindo a quantidade de pontos a serem tratados pelo algoritmo. A nuvem de pontos restante é então processada utilizando-se o framework *OctoMap* para produzir uma *octree*.

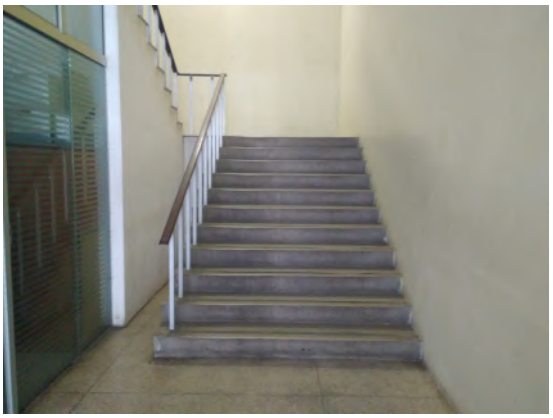
Para a obtenção destes resultados, as transformações de referencial foram definidas experimentalmente, uma vez que não estava disponível uma metodologia de localização dos sensores para um referencial inercial global. Para os resultados obtidos utilizando somente o *Kinect*, mediu-se a distância vertical do sensor até o chão para se determinar os valores da translação necessária para levar o referencial do sensor até o chão; em seguida, aplicou-se uma rotação de forma a deixar o eixo Z sendo

vertical com sentido positivo para cima e o eixo Y apontando para a mesma direção e sentido da frente do sensor. Para cada experimento realizado utilizando somente o *Kinect*, combinou-se a translação e a rotação determinadas para este experimento para que os parâmetros da transformação de referencial fossem definidos.

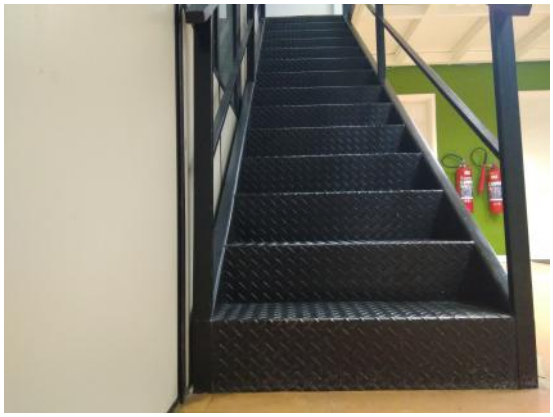
Nos experimentos realizados utilizando somente o *Velodyne VLP-16*, os parâmetros das transformações de referencial foram obtidos aplicando-se a mesma metodologia experimental dos experimentos realizados utilizando somente o *Kinect*. Determinou-se a distância deste sensor *laser* até o chão para obter a translação necessária e, em seguida, aplicou-se uma rotação de forma a deixar o eixo Z vertical com sentido positivo para cima e o eixo Y apontando para a frente do sensor *laser*.

Para os experimentos realizados utilizando os dois sensores em conjunto, é necessário que os dados obtidos dos sensores possuam coordenadas referentes ao mesmo referencial. Para isto, determinou-se primeiramente a transformação necessária para levar o referencial do *Velodyne VLP-16* para o referencial do *Kinect*. Isto foi feito com o auxilio visual da ferramenta *rviz* do ROS, através da qual foi possível observar o resultado da transformação dos pontos obtidos pelo *laser* e alterar esta transformação até que o resultado coincidisse visualmente com os pontos obtidos pelo *Kinect*, segundo o referencial do mesmo. A obtenção da transformação do referencial do *Kinect* para o referencial localizado no chão foi obtida utilizando a mesma metodologia dos experimentos realizados utilizando somente o *Kinect*. Desta forma, determina-se os parâmetros das duas transformações necessárias para este experimento. Estes parâmetros das transformações são utilizados no arquivo de *launch octomap_server*, que pode ser visualizado em 4.5.

(a) Escada A



(b) Escada B



(c) Escada C

Figura 5.1: Escadas utilizadas para a avaliação do algoritmo proposto, com dados obtidos de diferentes perspectivas.

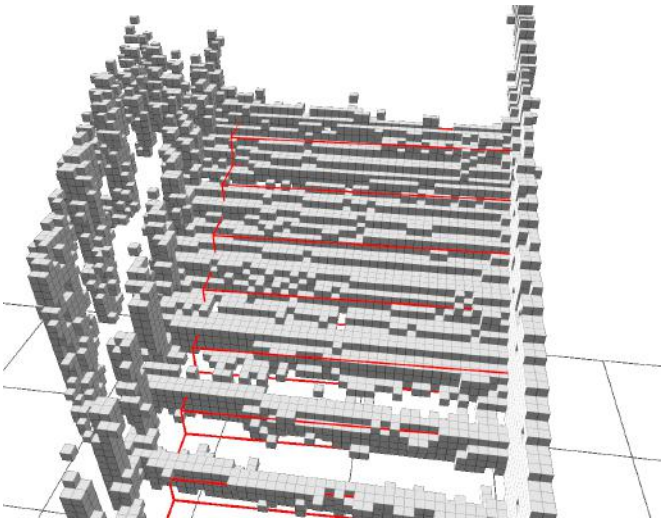


Figura 5.2: Ruído observado ao detectar os últimos 6 degraus da escada B com o *Kinect*

5.1.1 Configuração do algoritmo

Os seguintes valores de configuração foram utilizados nos experimentos. É importante ressaltar que os parâmetros definidos para o algoritmo possuem relação com os parâmetros de inicialização do pacote *diane_octomap*, baseadas na tabela 5.1. Utilizou-se uma *octree* com $\delta_{res} = 5$ cm de resolução para armazenar os dados obtidos do sensoreamento das 3 escadas da figura 5.1. O primeiro filtro foi configurado para descartar colunas de pontos mais baixas do que 15 cm e mais altas do que 30 cm, determinando assim $col_{minsize} = 3$ e $col_{minsize} = 6$. Para a Transformada de Hough, utilizou-se passos em ρ e θ iguais a $\rho_{step} = 5$ cm e $\theta_{step} = 5$ graus. Para a etapa "Detecção de Retas", a distância mínima para que um ponto seja considerado um voto foi definida como $dist_{maxtol} = 2.5$ cm, e a largura mínima de um degrau foi definida como $w_{min} = 25$ cm, o que corresponde a $n_{minvotos} = 5$. As alturas mínima e máxima definidas para um degrau são de $h_{min} = 15$ cm e $h_{max} = 30$ cm, respectivamente. Com esses valores, define-se para a etapa "Sequenciamento"os parâmetros $n_{minlinhas} = 3$, $n_{maxlinhas} = 6$, $\delta_{min} = 15$ cm e $\delta_{max} = 50$ cm. Para a etapa "Segmentação e Re-sequenciamento", utilizou-se uma distância máxima de $\delta_{seg} = 10$ cm para segmentar os planos.

Tabela 5.1: Relacionamento entre os parâmetros de inicialização e os do algoritmo.

Etapa	Parâmetro de iniciliazação	Parâmetro do algoritmo	Valor
Ordenação por Z e Filtragem	ColumnMinSize	$col_{minsize}$	3
	ColumnMaxSize	$col_{maxsize}$	6
Detecção de Retas	RhoStep	ρ_{step}	0.05 m
	ThetaStep	θ_{step}	5 graus
	AccumulateDistTolerance	$dist_{maxtol}$	0.025 m
	MinAmountVotes	$n_{minvotos}$	5
Sequenciamento	MinNumberLines	$n_{minlinhas}$	3
	MaxNumberLines	$n_{maxlinhas}$	6
	SequenceMinSegmentDist	δ_{min}	0.15 m
	SequenceMaxSegmentDist	δ_{max}	0.50 m
Segmentação e	SegmentationTol	δ_{seg}	0.10 m
Re-sequenciamento	IntervalTol	-	0.15 m
Modelagem	ModellingMinStepHeight	h_{min}	0.15 m
	ModellingMaxStepHeight	h_{max}	0.30 m

5.1.2 Discussão

Para tornar este algoritmo eficiente, foi necessário utilizar filtros para remover grande parte dos pontos que não são referentes à escadas. Isto é feito em diversas etapas do método, como a remoção de paredes e objetos altos durante o passo "Sequenciamento". A figura 5.3 mostra os planos filtrados e os que permanecerão no processo na etapa de "Sequenciamento". Os planos em vermelho são descartados porque possuem menos de $n_{minlinhas}$ ou mais do que $n_{maxlinhas}$. Pela figura, também é possível perceber a grande quantidade de dados filtrados. Os pontos remanescentes ainda são sujeitos à lógica de sequenciamento, reduzindo ainda mais a quantidade de dados.

Os modelos de escadas obtidos nos experimentos são encontrados nas figuras 5.4, 5.5 e 5.6, nas quais pode-se verificar que o algoritmo proposto é capaz de encontrar escadas mesmo em regiões ruidosas do conjunto de dados. A tabela 5.2 mostra que, mesmo utilizando uma resolução de 5 cm, o algoritmo é capaz de modelar as escadas com precisão com os dados obtidos pelo *Kinect*, nas quais os erros permanecem limitados pelo valor da resolução, com excessão do erro de largura. A tabela 5.3 mostra que os modelos das escadas A e B se tornam ainda mais precisos ao se utilizar o laser *Velodyne VLP-16* na etapa de mapeamento. Entretanto, percebe-se o aumento do erro de modelagem na largura da escada B, que ocorre devido à falta dos pontos dos extremos do primeiro degrau no mapeamento.

Além disso, conforme pode ser verificado na tabela 5.4, o modelo da escada A obtido pelos sensores *Kinect* e *Velodyne VLP-16* executando em conjunto apresenta a melhor precisão. Neste resultado, o erro da largura é muito menor do que nos modelos obtidos com os sensores separados. Isto acontece porque os pontos do primeiro degrau, que não eram obtidos pelo laser, são detectados pelo *Kinect*; e porque, devido à maior precisão dos dados obtidos pelo laser, ocorre uma redução na quantidade de ruído no último degrau presente na *octree* utilizada pelo algoritmo. Com este resultado, confirma-se também que combinações de sensores podem ser utilizados junto com este algoritmo. Os experimentos demonstram que o algoritmo foi capaz de realizar todo o processo de detecção e modelagem das escadas em menos de $60ms$ em todos os casos, podendo então ser considerado bastante eficiente.

Apesar de apresentar pequenos erros na modelagem da altura e profundidade das escadas, a largura apresenta erros significativos em todos os modelos obtidos utilizando o *Kinect*. Isto ocorreu devido à busca dos pontos do primeiro e último degrau que possuem a maior distância entre si para realizar o cálculo desta propriedade. A fim de evitar este problema, o cálculo desta propriedade pode ser feito descartando as duas maiores distâncias. Apesar de a solução ter mostrado uma redução neste erro, devem ser realizadas mais experiências para validar esta abordagem.

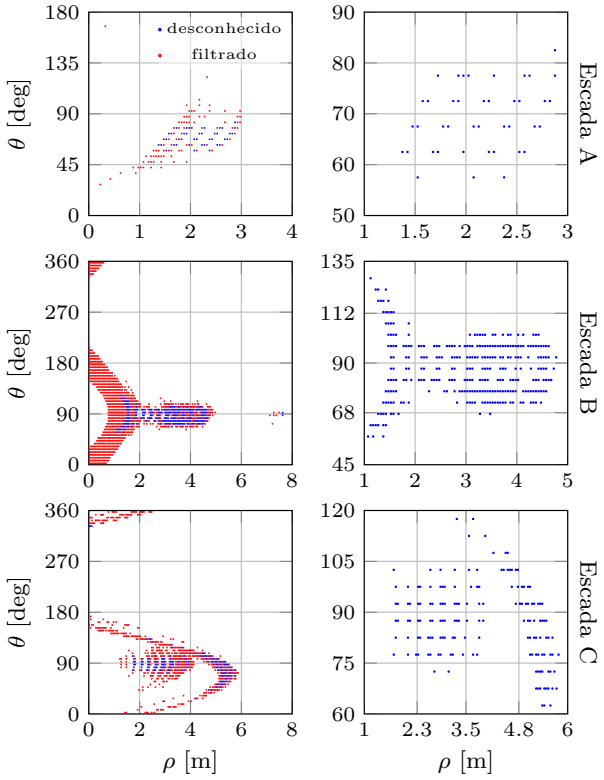


Figura 5.3: Dados experimentais obtidos durante o "Sequenciamento". Elementos em vermelho são descartados e os azuis são submetidos ao sequenciamento.

Nota-se que a detecção é bem sucedida mesmo na presença de ruído de medição, como pode ser observado na figura 5.2. Percebe-se também que os últimos 3 degraus não são completamente observados e mesmo assim eles foram detectados pelo algoritmo.

O algoritmo também não produz resultados falso positivos na ausência de escadas na nuvem de pontos utilizada. Os resultados obtidos neste tipo de experimento mostram que, uma vez que nenhuma sequência válida é detectada durante a etapa "Sequênciamento"e que a maioria dos pontos são descartados por filtros anteriores, a execução do algoritmo termina em menos de $10ms$.

Percebe-se também que a hipótese das escadas serem regulares é necessária para a metodologia de detecção de escada, uma vez que durante o processo de sequenciamento, busca-se uma escada com degraus igualmente espaçados entre si. Para a metodologia de modelagem de escada, esta hipótese também se faz necessária, dado que esta metodologia realiza a modelagem de um degrau médio utilizando todos os pontos detectados para a escada. Caso a modelagem de cada degrau fosse realizada utilizando somente os pontos detectados do degrau, esta hipótese não seria necessária e os degraus seriam modelados com propriedades diferentes.

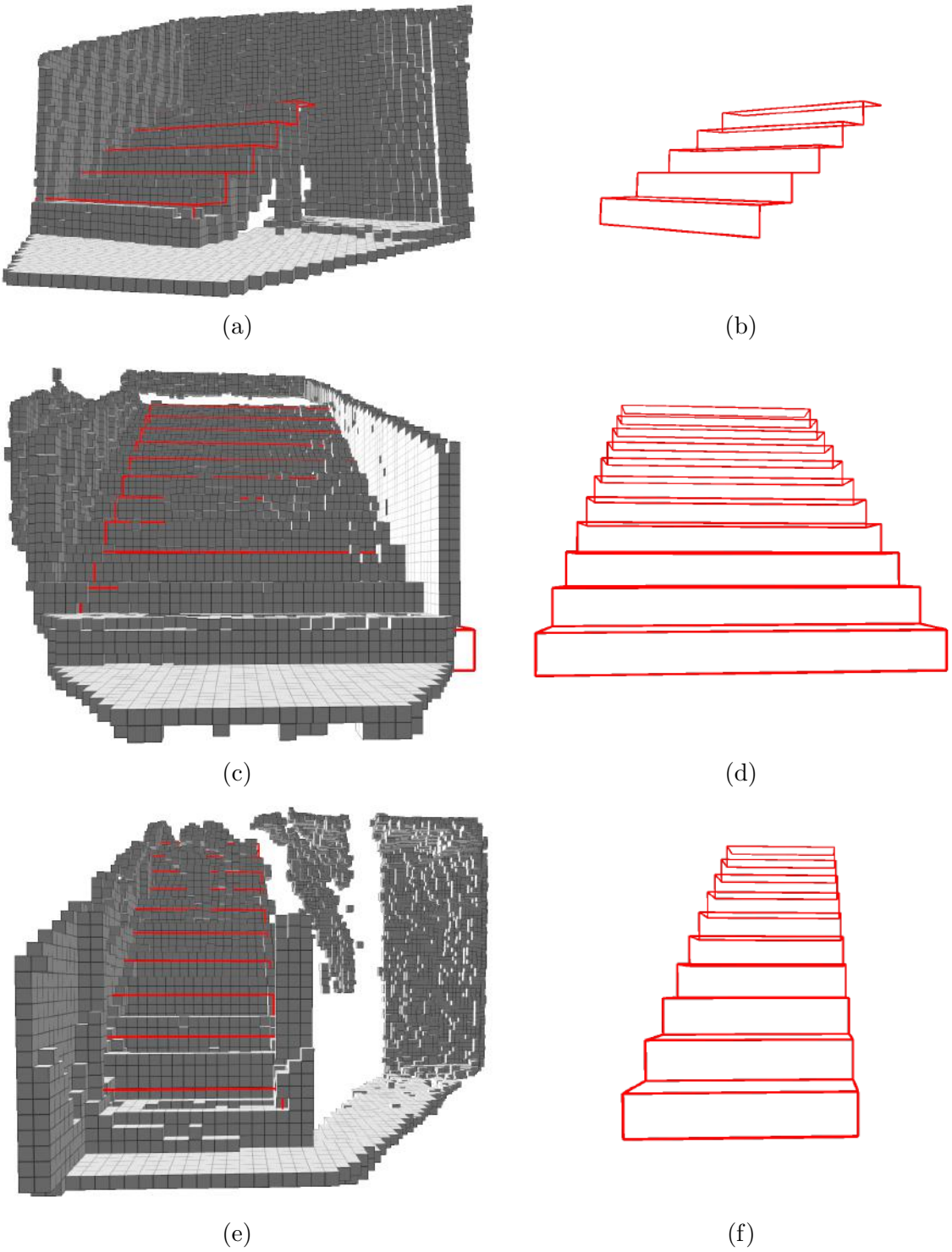


Figura 5.4: Resultado da Detecção à partir de dados obtidos com o *Kinect* da Escada A (a, b), Escada B (c, d) e Escada C (e, f).

Tabela 5.2: Erro de modelagem de escada com o *Kinect*.

Escada	Altura (cm)	Profundidade (cm)	Largura (cm)
A	1×10^{-6}	0.4	11
B	2.6	0.9	17
C	0.5	5	14

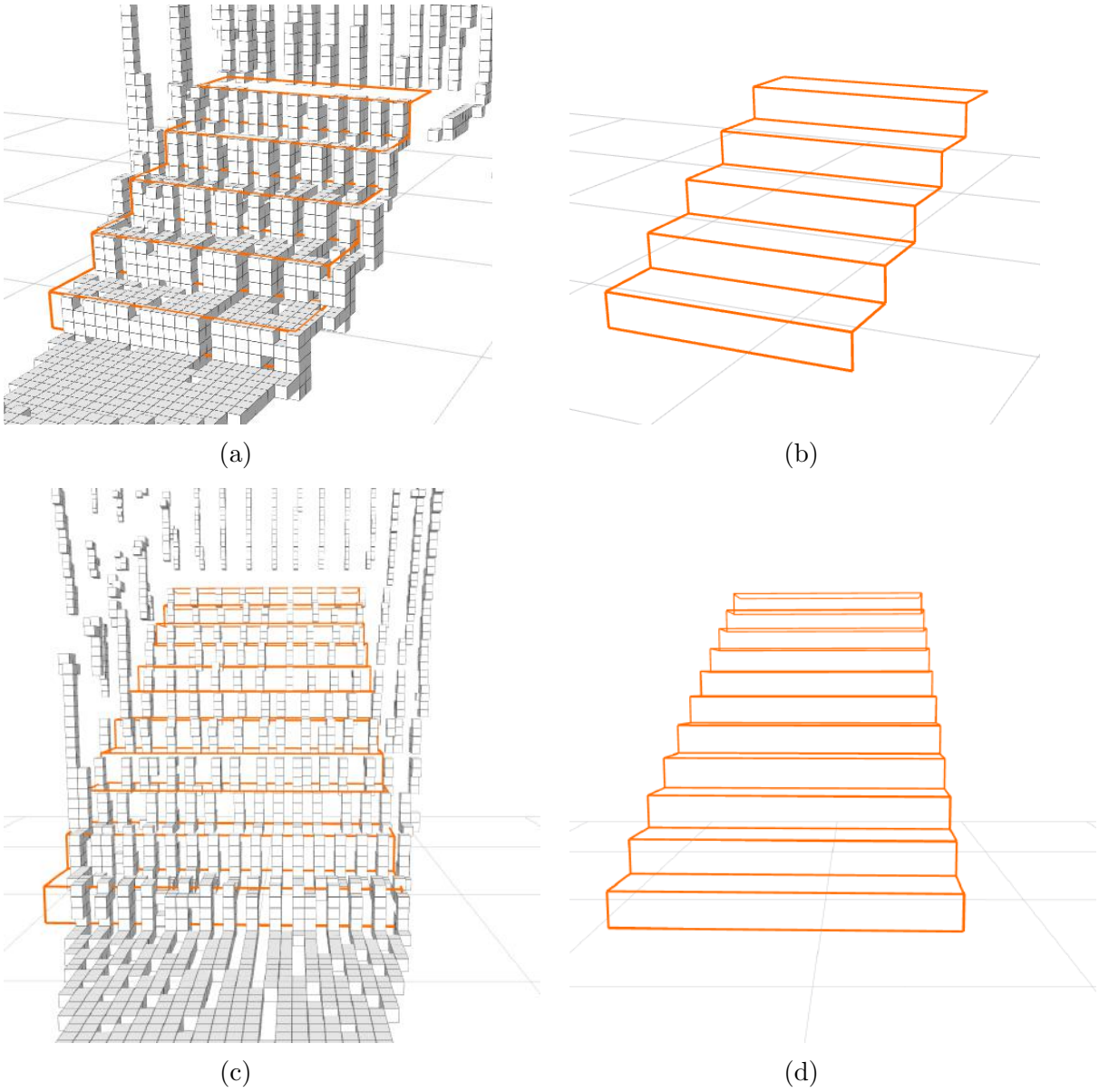


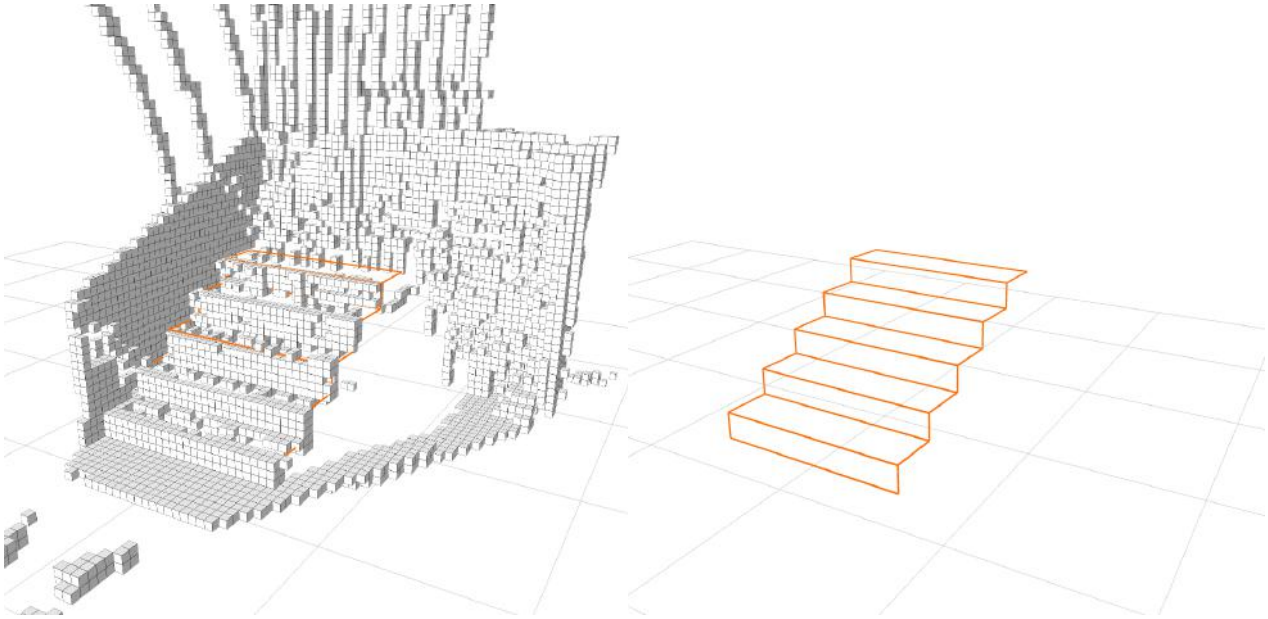
Figura 5.5: Resultado da Detecção à partir de dados obtidos com o sensor laser *Velodyne VLP-16* da Escada A (a, b) e Escada B (c, d).

Tabela 5.3: Erro de modelagem de escada com o *Velodyne VLP-16*.

Escada	Altura (cm)	Profundidade (cm)	Largura (cm)
A	1×10^{-7}	0.01	6
B	1.9	0.2	34

Tabela 5.4: Erro de modelagem de escada com o *Kinect* e *Velodyne VLP-16*.

Escada	Altura (cm)	Profundidade (cm)	Largura (cm)
A	2×10^{-9}	0.04	1.2



(a) (b)

Figura 5.6: Resultado da Detecção à partir de dados obtidos simultaneamente com o *Kinect* e com sensor laser *Velodyne VLP-16* da Escada A.

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho apresentou uma novo algoritmo de detecção e modelagem de escadas computacionalmente eficiente e robusto a ruído de medição. O intuito do projeto é fazer com que esta tarefa de identificação de uma escada seja menos custosa para o processamento e mais rápida, de modo que o poder de processamento possa ser utilizado em tarefas mais críticas.

Apesar de este trabalho ter sido desenvolvido com enfoque na sua utilização durante o controle de subida de um robô móvel, ele foi implementado pensando-se em sua modularização, de maneira que outras aplicações possam reutilizá-lo de uma forma bem simples. Sua utilização depende somente da integração com o *ROS*, da execução do *octomap_server* e de sensores de mapeamento.

Os resultados experimentais mostram que o método de detecção é eficaz, realizando a identificação das escadas utilizadas nos experimentos em menos de 50 *ms*. Esse tempo possibilita seu uso embarcado para aplicações em tempo real. Entretanto, testes mais complexos devem ser feitos para verificar sua eficiência computacional. Os erros encontrados nas modelagens das escadas são aceitáveis, com excessão dos erros observados nas larguras dos modelos, o que pode impactar fortemente nas aplicações que utilizarem este algoritmo.

6.1 Trabalhos futuros

Este trabalho apresentou resultados bastante satisfatórios em relação a sua eficiência e robustez na detecção e modelagem de escadas. Apesar disto, pode-se propor melhorias para a metodologia e novas linhas de estudo. Nesse sentido, pode-se citar:

- Melhorar o processo de detecção e modelagem de escadas, de forma que a segmentação de planos não seja dependente somente da coordenada X dos pontos do plano, uma vez que esta dependência pode impedir a detecção de escadas caso o sistema de coordenada for diferente; e de forma que o modelo obtido seja mais preciso.

- Desenvolver algoritmos de detecção de escadas descendentes que possam ser utilizados pelo robô DIANE. O algoritmo proposto neste trabalho não é capaz de detectar escadas descendentes. Assim, se faz necessária para a exploração de ambientes urbanos a criação de um método de reconhecimento e modelagem de escadas descendentes.

- Implementar algoritmos de localização para o robô DIANE, permitindo a geração de mapas utilizando os sensores e a movimentação do robô. A existência de um mapa facilitaria a tarefa de reconhecimento de obstáculos, podendo inclusive ser utilizado como fonte de informações para o algoritmo proposto neste trabalho.

Referências Bibliográficas

Azevedo, T. P. (2017), Locomoção de um robô móvel com esteiras em escadas, Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Borrmann, D., Elseberg, J., Lingemann, K. & Nchter, A. (2011), 'The 3D Hough transform for plane detection in point clouds: A review and a new accumulator design', *3D Research*.

Carbonara, S. & Guaragnella, C. (2014), 'Efficient stairs detection algorithm assisted navigation for vision impaired people', *IEEE International Symposium on Innovations in Intelligent Systems and Applications* pp. 313–318.

Chan, D., Monteiro, J., Silva, R. & Lizarralde, F. (2017), 'Stairway detection and modeling for climbing control of mobile robots', *Int. Conf. on Intelligent Robots and Systems (IROS)*.

De Lima, L. C. (2016), Locomoção semiautônoma em escadas para robôs móveis com esteiras, Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Delmerico, J. A., Baran, D., David, P., Ryde, J. & Corso, J. J. (2013), Ascending stairway modeling from dense depth imagery for traversability analysis, *in* 'IEEE Int. Conf. on Robot. and Autom.', pp. 2283–2290.

Eilering, A., Yap, V., Johnson, J. & Hauser, K. (2014), Identifying support surfaces of climbable structures from 3d point clouds, *in* 'IEEE Int. Conf. on Robot. and Autom.', IEEE, pp. 6226–6231.

Harms, H., Rehder, E., Schwarze, T. & Lauer, M. (2015), Detection of ascending stairs using stereo vision, *in* 'IEEE/RSJ Int. Conf. on Intel. Robots and Syst.', pp. 2496–2502.

Hart, P. E. (2009), 'How the Hough transform was invented [dsp history]', *IEEE Sign. Process. Mag.*

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. & Burgard, W. (2013), 'Octomap: An efficient probabilistic 3d mapping framework based on octrees', *Auton. Robots* **34**(3), 189–206.

Hough, P. V. C. (1962), 'Method and means for recognizing complex patterns'. US Patent 3,069,654.

Illingworth, J. & Kittler, J. (1988), 'A survey of the Hough transform', *Comp. Vision, Graph., and Image Process.* **44**(1), 87–116.

Kuo, C.-F. J., Shih, C.-Y. & Lee, J.-Y. (2005), 'Repeat pattern segmentation of printed fabrics by hough transform method', *Textile Research Journal* **75**(11), 779–783.

Loureiro, G. S. M. (2017), Desenvolvimento do software para o posicionamento dinâmico do rov luma, Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Oßwald, S., Gutmann, J.-S., Hornung, A. & Bennewitz, M. (2011), From 3d point clouds to climbing stairs: A comparison of plane segmentation approaches for humanoids, *in* 'IEEE-RAS Humanoid Robots (Humanoids)', pp. 93–98.

Silva, R. K. (2017), Metodologia para subida de escada em robôs móveis com esteiras, Graduate dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Apêndice A

Informações dos sensores

Tabela A.1: Informações do Kinect Fonte: https://msdn.microsoft.com/en-us/library/

Alimentação	12 V (DC)
Campo de visão (Vertical)	43°
Campo de visão (Horizontal)	57°
Alcance de inclinação	±27°
Taxa de quadros	30 FPS
Características do acelerômetro	**3-axial configurado para alcance de 2G**
Acurácia do acelerômetro	1°

Tabela A.2: Informações do laser VLP-16 Informações retiradas do manual

Alimentação	$9 - 32$ V (DC)
Ângulo de varredura (Vertical)	±15°
Resolução ângular (Vertical)	2°
Ângulo de varredura (Horizontal)	360°
Resolução ângular (Horizontal)	$0.1° - 0.4°$
Alcance	Até 100m
Acurácia	±3 cm
Frequência de varredura	$5 - 20$ Hz

Tabela A.3: Informações do laser UST-10LX Informações retiradas do manual

Alimentação	12/24 V (DC)
Ângulo de varredura (Horizontal)	270°
Resolução ângular	0.25°
Alcance	0.06m - 30m
Período de varredura	25 ms