Universidade Federal da Paraíba

Centro de Informática

Programa de Pós-Graduação em Informática

Uma Nova Abordagem para Identificação e Reconhecimento de Marcos Naturais Utilizando Sensores RGB-D

André Luiz Figueiredo de Castro

João Pessoa, Paraíba, Brasil

André Luiz Figueiredo de Castro

Uma Nova Abordagem para Identificação e Reconhecimento de Marcos Naturais Utilizando Sensores RGB-D

Dissertação submetida à Cordenação do Curso de Pós-Graduação em Informática do Centro de Informática, da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sinais, Sistemas e Gráficos

Orientador: Tiago Pereira do Nascimento

João Pessoa, Fevereiro de 2017

C355n Castro, André Luiz Figueiredo de.
Uma nova abordagem para identificação e reconhecimento de marcos naturais utilizando sensores RGB-D / André Luiz Figueiredo de Castro.- João Pessoa, 2017.
82 f. : il. -

Orientador: Tiago Pereira do Nascimento.
Dissertação (Mestrado) – UFPB/PPGI

1. Sensores 3D – RGB-D. 2. Robôs Móveis. 3. Visão Robótica. I. Título.

UFPB/BC CDU: 004(043)

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

PPGI
INFORMÁTICA • UFPB

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de ANDRÉ LUIZ FIGUEIREDO DE CASTRO, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 17 de fevereiro de 2017.

Aos dezessete dias do mês de fevereiro, do ano de dois mil e dezessete, às dez horas, no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. André Luiz Figueiredo De Castro, vinculado a esta Universidade sob a matrícula nº 2015105011, candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa *"Sinais, sistemas digitais e gráficos"*, do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Tiago Pereira Do Nascimento (PPGI-UFPB), Orientador e Presidente da Banca, Leonardo Vidal Batista (PPGI-UFPB), Examinador Interno ao Programa, e Luiz Marcos Garcia Gonçalves (UFRN), Examinador externo à instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado "Uma nova abordagem para identificação e reconhecimento de marcos naturais utilizando sensores RGB-D.". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Clauirton de Albuquerque Siebra, Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 17 de fevereiro de 2017.

Prof. Dr. Clauirton de Albuquerque Siebra

Prof. Dr. Tiago Pereira Do Nascimento
Orientador (PPGI-UFPB)

Prof. Dr. Leonardo Vidal Batista
Examinador interno (PPGI-UFPB)

Prof. Dr. Luiz Marcos Garcia Gonçalves
Examinador externo à instituição (UFRN)

Centro de Informática

Universidade Federal da Paraíba

Programa de Pós-Gradução em Informática

Dissertação do Curso de Pós-Graduação em Informática intitulado ***Uma Nova Abordagem para Identificação e Reconhecimento de Marcos Naturais Utilizando Sensores RGB-D*** de autoria de André Luiz Figueiredo de Castro, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Tiago Pereira do Nascimento (Orientador)

Universidade Federal da Paraiba

Prof. Dr. Leonardo Vidal Batista

Universidade Federal da Paraiba

Prof. Dr. Luiz Marcos Garcia Goncalves

Universidade Federal do Rio Grande do Norte

Coordenador(a) do Programa

Prof. Dr. Clauirton de Albuquerque Siebra CI/UFPB

João Pessoa, 05 de Fevereiro de 2017.

Centro de Informática, Universidade Federal da Paraíba

Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600

Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

"A lost battle is a battle one think one has lost."

DEDICATÓRIA

Dedico este trabalho à minha mãe, que tanto me apoiou na minha jornada.

AGRADECIMENTOS

Muitas pessoas fizeram parte comigo durante o período de pesquisa, e muitas outras contribuíram para o desenvolvimento desse trabalho, no entanto gostaria de evidenciar algumas. Em primeiro lugar, gostaria de agradecer à minha família, em especial meu profundo agradecimento à minha mãe, pelo amor e sacrifício constante para que nada me faltasse nessa atribulada viagem. Gostaria de agradecer ao meus irmãos, Caio, Davi e Henrique, pelo companheirismo, pelo apoio dado e pela força e motivação. Gostaria também de agradecer à todos os meus colegas do LaSER, que me apoiavam na fase de implementação e testes, em especial A Yôoh, Leo, Carlos Eduardo, Viviano, Luís Felipe e Abraão. Agradeço também a todos os professores que contribuiram direta e indiretamente, em especial à Professora Thais Gaudêncio e ao Professor Alisson Brito. Por último e não menos importante, gostaria de agradecer ao meu orientador Prof. Doutor Tiago Nascimento, por ter me apresentado a proposta desse trabalho e ter me motivado e passado instruções sempre que necessário.

Resumo

Com o avanço na pesquisa de algoritmos de localização de robôs móveis, a necessidade de identificação e reconhecimento de pontos de referência naturais aumentou. A detecção de marcos naturais é uma tarefa desafiadora, porque a sua aparência pode ser diferente em forma e design e, também, eles sofrem influência da iluminação do ambiente. Como um exemplo, um algoritmo de reconhecimento de objeto 2D típico pode não ser capaz de lidar com a grande variedade óptica de portas e escadas em corredores grandes. Em outra direção, as melhorias recentes em sensores 3D de baixo custo (do tipo RGB-D) permitem aos robôs perceber o ambiente como uma estrutura espacial 3D. Assim, usando esta nova tecnologia, um algoritmo para identificação e reconhecimento de marco natural baseado em imagens adquiridas a partir de uma câmera RGB-D é proposto. Basicamente, durante a fase de identificação que é um primeiro passo para trabalhar com marcos, o algoritmo explora o conhecimento estrutural básico sobre os pontos de referência, extraindo suas bordas e criando uma nuvem de pontos de borda. No próxima, a fase de reconhecimento, as arestas são usadas com um algoritmo de reconhecimento não supervisionado proposto *on-the-fly* para demonstrar a eficácia da abordagem no reconhecimento de portas e escadarias. Dois métodos de Reconhecimento foram propostos e resultados mostram que a eficiência geral dos dois métodos passa dos 96% de Precisão de reconhecimento. Abordagens futuras propõem-se a fusão dos dois métodos para melhores resultados no reconhecimento, bem como inclusão de novos objetos como bebedouros, lixeiras e comparar essa abordagem modificada com outras abordagens que necessitam de treinamento, como K-Neighbouring mais próximo, Bayes e redes neurais.

Palavras-chave: Reconhecimento de Marcos Naturais, Robô Móvel, Localização baseada em Visão, Visão Robótica.

Abstract

With the advance in the research of mobile robots localization algorithms, the need for natural landmark identification and recognition has increased. The detection of natural landmarks is a challenging task because their appearance can be different in shape and design and, as well, they suffer influence of the environment illumination. As an example, a typical 2D object recognition algorithm may not be able to handle the large optical variety of doors and staircases in large corridors. On another direction, recent improvements in low-cost 3D sensors (of the type RGB-D) enable robots to perceive the environment as a 3D spatial structure. Thus, using this new technology, an algorithm for natural landmark identification and recognition based on images acquired from an RGB-D camera is proposed. Basically, during the identification phase that is a first step for working with landmarks, the algorithm exploits the basic structural knowledge about the landmarks by extracting their edges and creating a cloud of edge points. In the next, the recognition phase, the edges are used with a proposed *on-the-fly* unsupervised recognition algorithm in order to demonstrate the effectiveness of the approach in recognizing doors and staircases. Two methods of recognition have been proposed and results show that a general technique of the two methods passes from the 96 of accuracy. Future approaches propose a mix of these two methods for better results of recognition, as well as inclusion of new objects such as drinking fountains, dumps and compare this modified approach with other approaches that require training, such as nearest K-neighbors, Bayes and neural networks .

Keywords: Natural Landmarks Recognition, Mobile Robot, Vision Based Localization, Robot Vision

Conteúdo

Lista de Símbolos

UFPB : *Universidade Federal da Paraiba*

LaSER : *Laboratório de Sistemas Embarcados e Robótica*

SLAM : *Simultaneous Localization and Mapping*

PCL : *Point Cloud Library*

ROS : *Robotic Operation System*

RANSAC : *Random Sample Consensus*

SL0 : *Smoothed L0*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Neste capítulo inicial, é apresentada uma introdução geral da área pesquisada no trabalho, contextualizando-o em uma subárea e identificando a motivação para o desenvolvimento desta dissertação. Os objetivos e contribuições feitas pelo autor também serão definidas e exploradas. A estrutura deste trabalho será apresentada ao final.

1.1 Motivação

A ideia de uma máquina autônoma é muito antiga na história da humanidade, sendo as primeiras configurações chamadas de autômatos, máquinas geralmente construídas com engrenagens que possuíam alguma função específica. No entanto, a palavra robô foi primeiramente introduzida em 1920 por Karel Capek em sua peça "R.U.R, Rossum's Universal Robots"[1]. Nessa peça, a palavra tinha um significado relacionado a servo, ou aquele que faz trabalho obrigado. Nela, os robôs eram um tipo de androide produzidos em massa, destinados a trabalhar para os humanos. Desde os autômatos os robôs evoluíram bastante, passando de simples aparatos estáticos e não autônomos para algo com mais mobilidade e autonomia. Sendo visto como auxiliadores de tarefa, as ações realizadas por robôs neste universo contemporâneo têm a intenção de resolver e facilitar problemas globais e privados.

Nos primeiros anos da robótica, o uso principal de robôs foi focado em arranjos industriais na forma de manipuladores [2], geralmente utilizados em linhas de montagem para trabalhos repetitivos de movimentação limitada. As novas tecnologias trouxeram mobilidade para as plataformas robóticas, tornando-as cada vez mais aplicáveis em outras áreas com novos

paradigmas e mudanças de necessidades. Hoje os robôs podem ser divididos em duas principais esferas: os robôs manipuladores e os robôs móveis. As aplicações envolvendo robôs móveis e autônomos cresceram significativamente nos últimos anos, sendo encontrados em uma larga escala de variedade, passando por veículos terrestres, aeroespaciais, aquáticos e até humanoides. Uma subcaegoria amplamente utilizada dos veículos terrestres são as plataformas móveis com rodas, alvo de pesquisa e estudo desse trabalho.

Dentre as aplicações com robôs móveis, podem ser citadas a direção autônoma, transporte de cargas, limpeza, manutenção, segurança, entretenimento, patrulhamento, exploração espacial e de ambientes perigosos ou danosos ao ser humano [3][4][5] (Ver Figura 1.1). Uma área moderna e emergente onde os robôs podem atuar é o setor de serviços, cooperando com pessoas como atendentes automatizados, cuidadores, entregadores, construtores, dentre outra gama de trabalhos. Em cada uma das atividades mencionadas, o robô requer ótima mobilidade e autonomia, sendo capaz de mover-se pelo mundo de forma otimizada, sem ou com o mínimo possível de intervenção humana.

Figura 1.1: Exemplos de robôs móveis e suas aplicações

Dada essa mudança de um cenário de confinamento para ambientes mais amplos como salas de escritórios, recintos domésticos, grandes salões e até mesmo cenários ao ar livre, os espaços de trabalho dos sistemas robóticos aumentaram proporcionalmente, surgindo a necessidade de novos dispositivos para corrigir as inconsistências e erros que venham a cometer. Imagine uma pessoa andando numa rua com vários obstáculos, procurando chegar a um ponto no espaço, seguindo um caminho previamente conhecido, mas de olhos fechados. Essa pessoa sentirá enorme dificuldade de chegar ao seu destino, pois os movimentos são imprecisos e acumulam ao longo do tempo. Mesmo que uma pessoa ache que estará andando em linha reta, ela poderá fazer curvas sem perceber. Felizmente, existe a visão e outros sentidos que constantemente corrigem as imperfeições dos movimentos, obtendo-se

ideia da localização no espaço.

Robôs que circulam livremente pelo mundo possuem os mesmos problemas que os humanos quando locomovem-se. Se apenas movem-se sem manter um registro das suas ações, perdem-se por causa das imperfeições dos mecanismos de movimento quando atuam no ambiente. Por conseguinte, devem ser equipados com dispositivos capazes de perceber o mundo circundante. Como os seres humanos usando os olhos e outros sentidos, os robôs podem detectar o mundo pelo uso de sensores distribuídos, reparando essas imperfeições e tendo uma ideia melhor onde eles estão localizados.

Um robô móvel pode localizar-se por meio de duas classes de sensores, os *Propriocepti-vos* e os *Exteroceptivos* [2][6][7]. Os sensores de tipo proprioceptivos percebem parâmetros internos do robô, como velocidade das rodas e número de voltas que a roda realizou. Podem ser encontrados na forma de encoders, odômetros e sensores inerciais. Os de tipo exteroceptivos medem parâmetros externos ao robô, tais como medidas de distância, luminosidade, força de recebimento de um tipo de sinal, etc. Deste último tipo podem ser citados sistemas de câmera, de luz infravermelha, de ultrassom, LIDAR, GPS, etc. As informações que os robôs obtêm usando esses sensores precisam ser combinadas para determinar com precisão o estado atual do próprio robô e do ambiente. Desse modo, a fusão de sensores é o problema de combinar dados de diferentes sensores em uma única perspectiva. Mais a frente no **Capítulo 3**, serão abordados trabalhos envolvendo diferentes tipos de sensores e suas características.

Dessa forma, os métodos de localização podem ser classificados em duas hierarquias: métodos de localização relativa e métodos de localização absoluta [8]. Os métodos de localização relativa operam com as localizações medidas em instantes anteriores de forma incremental para estimar-se a posição atual do robô. Distintivamente, os métodos de localização absoluta operam com informações no instante atual dos sensores em relação a um referencial fixo absoluto. Como exemplo de um referencial pode ser citado um marco, objeto ou característica distinguível presente no ambiente, disposto em uma localização conhecida. Os marcos podem ser artificiais, quando introduzidos no ambiente com a finalidade de auxiliar na navegação do robô, ou naturais quando já encontravam-se no ambiente antes do surgimento do problema de localização.

1.2 Definição do Problema

A localização de um robô móvel baseado em sensores, que permite que um robô adquira sua pose (posição e orientação), foi abordado como um dos problemas fundamentais no campo da robótica móvel [9]. Algumas soluções para o problema de localização têm sido desenvolvidas e tratadas por diversas técnicas desde a última década, culminando em uma gama de opções para o problema de localização. Deve-se levar em conta que para ambientes negados por GPS, isto é, dentro de prédios, a precisão é um fator chave para localizar um robô em corredores e salas. Por essa razão, o desenvolvimento de um método de localização acurado é um aspecto de extrema importância para o bom funcionamento dos robôs, resultando na realização das suas ações com êxito.

O desenvolvimento de uma técnica que seja mais próxima de como os humanos se localizam nos ambiente inseridos é de suma importância. Um dos fatos que podem ser citados para tal necessidade é a facilidade de compreensão da técnica, pois como os humanos utilizando a visão, o robô também usaria a visão computacional para realizar a localização. Outro fato importante seria a facilidade de aplicação em ambientes genéricos, pois uma técnica pode ser empregada em vários ambientes sem necessidade de alteração prévia dos mesmos. Por isso, a técnica de Identificação de Marcos Naturais proposta aqui, irá auxiliar em todos estes casos.

1.3 Objetivo

Diante deste cenário, nessa dissertação é realizado um estudo dos trabalhos de alguns dos trabalhos bastante relevantes e recentes na área de localização de robôs móveis e identificação de marcos naturais, culminando no desenvolvimento de uma técnica na detecção de marcos naturais para localização de robôs móveis. Com isso, o trabalho teve como objetivo principal o desenvolvimento de uma técnica diferenciada de detecção e reconhecimento de marcos naturais para auxiliar na localização de robôs móveis, utilizando a visão como sensor principal e a geometria do ambiente observado pelo robô. Assim, objetivos específicos desse trabalho foram:

- A partir das técnicas existentes na literatura, propor uma nova técnica melhorada de

Identificação de Marcos naturais para ser aplicada na Localização de Robôs móveis.

- Implementar essa técnica em forma de algoritmo.
- Testar essa técnica em um ambiente físico real não simulado.
- Comparar com outras técnicas já existentes na literatura.

1.4 Contribuições

A principal contribuição desse trabalho foi o desenvolvimento de uma nova técnica de identificação e reconhecimento de marcos naturais para auxiliar um robô móvel no processo de localização. Essa nova técnica pode ser implantada em qualquer sistema robótico que possua uma câmera RGBD acoplada e atue em ambientes internos, pois localiza-se a partir de características como portas e escadas.

Além disso, durante o desenvolvimento do mesmo e como prova de conceito foi publicado um artigo na ICARSC 2016, intitulado *"A Novel Approach for Natural Landmarks Identification Using RGB-D Sensors"*. Bem como a última versão do trabalho está em processo de aceitação para o Journal *Robótica*.

1.5 Estrutura da Dissertação

O **Capítulo 1** introduziu brevemente a problemática da navegação e localização de robôs móveis, expondo alguns conceitos simples, bem como buscou se estabelecer os objetivos dessa dissertação.O **Capítulo 2** tratá uma discussão dos principais temas relacionados a área de estudos, possibilitando o leitor a ter um entendimento mais embasado da literatura por trás da pesquisa. O **Capítulo 3** mostrará diversos trabalhos da área, contextualizando o leitor adentro dos temas pesquisados durante a pesquisa. Já no **Capítulo 4** será abordado o processo da criação do algoritmo proposto, onde serão apresentados gráficos e textos explicativos sobre o mesmo. Os resultados dos experimentos serão abordados no **Capítulo 5**, trazendo uma análise comparativa entre os algoritmos propostos. Finalmente no **Capítulo 6** serão apresentadas as considerações finais e os trabalhos futuros a serem desenvolvidos a partir do altual.

Capítulo 2

Fundamentação Teórica

No presente Capítulo são apresentados alguns conceitos básicos sobre Robótica Móvel, necessários para compreensão do trabalho proposto. É feita uma análise passando desde conceitos simples como *Mobilidade e Autonomia*, avançando para conceitos de *Localização e Métodos de Localização* e encerrando com alguns tipo de *Sensores e Frameworks* usados no processo.

2.1 Mobilidade e Autonomia

Conforme brevemente iniciado no **Capítulo 1**, os primeiros usos práticos de robôs foram em ambientes industriais com os chamados robôs manipuladores. Estes robôs têm a tarefa de fabricar produtos como peças de automóveis e são programados de tal forma que eles repetem a mesma sequência de ações repetidas vezes, mais rápido, mais barato e mais precisos do que seres humanos. Tipicamente eles consistem de um braço móvel fixo a um ponto no chão, que pode montar ou fabricar diferentes objetos. Além de moverem-se em torno deste ponto fixo, esses tipos de robôs não são capazes moverem-se livremente e, portanto, não são muito móveis. Com o passar do tempo, robôs mudaram de aparatos estáticos e vieram a ter mais mobilidade.

A *Mobilidade* de uma plataforma robótica é definida como o grau de liberdade no qual ela é permitida mover-se ao redor do domínio em que está inserida. Dessa maneira, com propósitos de realizar suas ações, os robôs precisam de certos graus de *Autonomia*, a qual depende da amplitude de conhecimento prévio ou informação de um ambiente que o robô

tenha para realizar suas tarefas, ou seja, é o nível de capacidade de governar-se pelos próprios meios. Assim, podem ser classificados em três classes de Autonomia: *Não Autônomos*, *Semi Autônomos* e *Totalmente Autônomos*.

- *Não Autônomos*: São robôs completamente controlados por humanos, não possuem nenhuma autonomia. A única inteligência consiste em interpretar os comandos recebidos remotamente por controladores humanos.

- *Semi Autônomos*: Possuem uma programação básica, podem navegar sozinhos ou serem controlados.

- *Totalmente Autônomos*: São dirigidos apenas pela própria programação, não requerendo interação humana para cumprir suas tarefas. São capazes de movimento inteligente, sem exigir orientação externa para controlá-los.

O grau de autonomia desejada sempre irá depender da situação em que o robô será aplicado. Para robôs manipuladores em ambientes industriais a não autonomia não é um problema, pois um produtor provavelmente irá querer controlar a produção, sem deixar tudo automatizado. Por outro lado, quando se quer chegar a um resultado sem especificar cada ação a ser tomada um robô semi ou totalmente autônomo entra em aplicação.

As aplicações de robôs autônomos cresceram bastante, podem ser citados como exemplo as iniciativas de exploração espacial [10][11]. Outra área em que robôs autônomos podem ser usados na forma de veículos subaquáticos, fazendo exploração ao seguir recifes ou tubulações e estudando criaturas marinhas obtendo grandes quantidades de novos dados científicos [8]. Também em terra e no ar, aplicações autônomas de robôs móveis podem ser encontradas. Os robôs podem, por exemplo, realizar reparos e manutenção remotamente, como no caso de [12], onde um robô percorre tubulações de água para detectar possíveis vazamentos ou defeitos. Também em [13] os robôs aéreos, chamados de drones, são utilizados para fazer exploração de plantações, ou em [14] que são utilizados em para a tarefa de SLAM em ambientes internos. Ainda os drones são utilizados para tarefas de busca e salvamento como mostra o trabalho de Sousa et al. [15].

Uma outra área de aplicação está no setor de entretenimento, onde robôs podem agir como guias de museus para atrair pessoas e mostrá-las ao redor, tanto fisicamente através de

interfaces [16]. Finalmente, uma área potencialmente interessante onde veículos autônomos podem ser usados é o setor de serviços. As cadeiras de rodas e os aspiradores melhoram a qualidade de vida de muitas pessoas. Robôs médicos ou Entregadores de alimentos podem auxiliar pessoas que trabalham na área de cuidados. Em cada uma das áreas de aplicação mencionadas o robô requer autonomia e mobilidade.

2.2 Navegação

Para se alcançar um bom nível de autonomia, um aspecto importante é que o robô seja capaz se locomover-se de forma inteligente dentro do seu local de trabalho. Este movimento é conhecido como *Navegação*. A navegação não é uma tarefa trivial, sendo um dos principais problemas na robótica móvel. Está diretamente associada ao problema de localização, o qual será abordado mais a frente. A navegação de um robô autônomo pode ser descrita como o processo utilizado pelo mesmo para mover-se em seu ambiente de trabalho, geralmente povoado com obstáculos, de uma posição e orientação inicial para uma final. O problema geral da navegação pode ser formulado em três questões [8][17]:

- *Onde estou?*: O robô precisa saber onde está para realizar decisões úteis. Perceber e entender os arredores está relacionada a Localização.
- *Onde estou indo?*: Para cumprir alguma meta o robô precisa saber onde se está indo, precisa ter um objetivo. Esse problema está relacionado com o Reconhecimento de Objetivos.
- *Como chegar lá?*: Uma vez que o robô está localizado e sabe onde tem que ir, precisa agora saber como chegar lá. Encontrar um caminho para o objetivo está relacionado com Planejamento de Trajetórias.

Para que a navegação seja possível, é necessário que o robô seja projetado com base em uma arquitetura baseada em uma sequencia de primitivas: Sentir $\rightarrow$ Planejar $\rightarrow$ Agir. Na Figura 2.1 pode ser visto um exemplo desse tipo de arquitetura. O processo de navegação é dividido em cinco níveis hierárquicos: Mapeamento, Localização, Geração de caminho, Geração de trajetória e Execução de Trajetória [18].

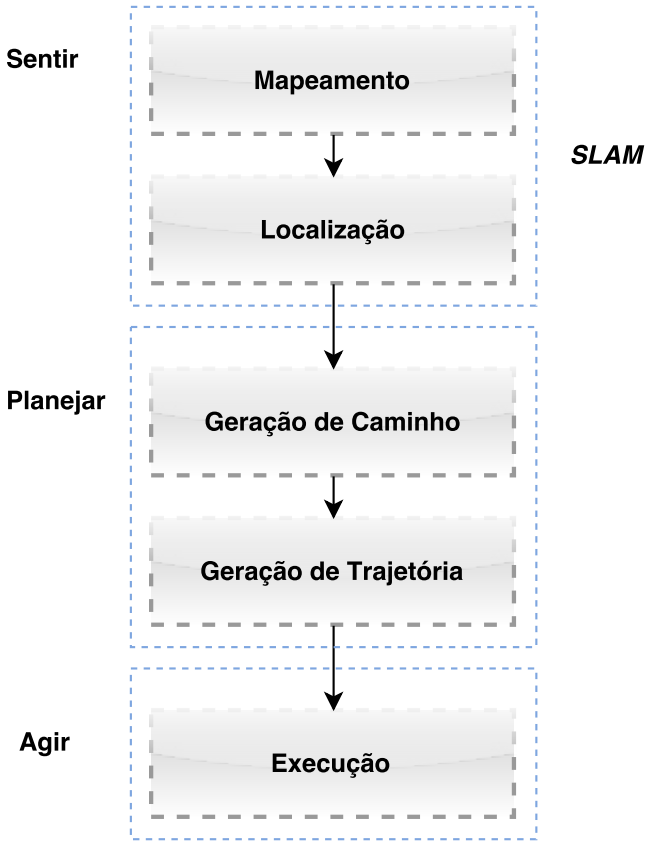

Figura 2.1: Arquitetura.

- **Mapeamento:** O robô utiliza sensores para capturar informações e criar uma modelagem contendo as principais características estruturais do ambiente, incluindo possíveis obstáculos.

- **Localização:** Com base no modelo obtido na fase de mapeamento, é feita a determinação da posição e orientaçao do robô no espaço. Aplicações que realizam as tarefas de Mapeamento e Localização simultaneamente são denominadas SLAM [19][20], visto como um dos problemas mais difíceis da robótica [21].

- **Geração de caminho:** O cálculo de um caminho que leve o robô de sua posição inicial até uma posição final é feito, minimizando a colisão com obstáculos.

- **Geração de trajetória:** O caminho gerado no nível anterior sofre adaptações levando-se em conta às restrições temporais, com isso são calculadas as velocidades que devem ser submetidas ao robô em cada instante de tempo para que o mesmo possa percorrer o caminho no intervalo de tempo especificado.

- **Execução de trajetória:** Os atuadores do robô são controlados de forma a executarem a trajetória gerada o mais fielmente possível. A localização deve ser monitorada constantemente para que desvios em realação à trajetória possam ser detectados e corrigidos.

Através desses exemplos fica claro que a Localização é uma etapa crucial no processo de localização do robô. Durante toda a execução da trajetória é de suma importância que seja conhecida constantemente a localização do robô, para que se possa realizar um acompanhamento da trajetória percorrida.

2.3 Localização

O problema de localização consiste em constantemente responder a pergunta *Onde estou?* do ponto de vista do robô. Localizar um robô fundamenta-se em determinar a sua *pose* (posição e orientação) em um determinado instante de tempo, ou seja, a sua coordenada x e y e direção no sistema de coordenadas global. O problema de localização é um problema importante e componente chave em muitos sistemas robóticos autônomos bem sucedidos [22]. Se um robô não sabe onde ele está em relação ao ambiente, é difícil decidir o que fazer. O robô provavelmente precisará ter pelo menos alguma ideia de onde está para ser capaz de operar e agir com sucesso [8]. Para alguns autores o problema tem sido apontado como o "mais fundamental capaz de fornecer reais capacidades autônomas"[23]. Os métodos de localização existentes podem ser classificados em três categorias: *Localização Relativa* , *Localização Absoluta* e *Fusão de Multi-Sensores (Localização Relativa + Absoluta)*.

2.3.1 Localização Relativa

O método de *Localização Relativa*, também conhecido como *Dead Reckoning*, vem regularmente sendo utilizado desde muito cedo. Foi um dos primeiros métodos de localização devido a sua simplicidade, onde a posição atual é baseada nas posições estimadas em instantes anteriores. Originalmente, este é o processo de estimar a posição de um avião ou um navio, apenas com base na velocidade e direção da viagem e no tempo passado desde a última posição conhecida, porém percebe-se que os erros de medição aumentar com o pas-

sar do tempo. Nas aplicações robóticas, as medições da posição relativa são adquiridas por *Odometria* ou *Navegação Inercial*.

A *Odometria* determina a localização do robô por medidas incrementais do movimento das rodas com o passar do tempo. Ao se utilizar *encoders* (sensores de rotação) para contar o número de revoluções que cada roda do robô realiza, o sistema interno do mesmo calcula a distância linear percorrida e orientação. Sistemas que usam Odometria são muito utilizados, porque ela é boa em precisão a curto prazo, é fácil de se implementar e permite taxas de amostragem muito altas [8]. Contudo, a desvantagem de tal método é o acúmulo de erros que aumenta significamente e proporcionalmente com à distância percorrida. Devido ao escorregamento das rodas, o método pode levar a medidas erradas tanto na distância como orientação. Outra desvantagem é a alta sensibilidade ao terreno, se a superfície que o robô está andando não for suave, pode resultar isso pode resultar em erros de posição consideráveis, uma vez que o sistema de odometria não corrige as irregularidades do terreno. Embora a odometria cause erro crescente na estimativa de localização, é a forma mais fácil de acessar da informação de posição e, portanto, é uma importante fonte de informação para localização. No próximo capítulo serão apresentados alguns trabalhos relacionados com odometria.

Técnicas que usam *Navegação Inercial* se baseiam no uso de *Giroscópios* e *Acelerômetros* para medir a rotação e aceleração do robô. Enquanto os primeiros detectam pequenas variações na orientação, os Acelerômetros detectam variações no eixo x ou y. Como na técnica passada, estimativas de posição com Navegação Inercial são adquiridas integrando as informações obtidas de sensores, sem dependência de informação externa. No entanto como as medidas são feitas por integração, sofrem do mesmo problema do acúmulo de erro com o passar do tempo, aumentando sem limites.

2.3.2 Localização Absoluta

As técnicas de *Localização Absoluta* fornecem informações sobre a localização independente de medidas feitas anteriormente, isto é, a localização não é derivada da integração de uma sequencia de medidas, mas de uma única medida. A grande vantagem desse método é o não acúmulo de erro com o tempo. Os métodos para obter Medições Absolutas podem ser divididos em métodos com base na utilização de *Marcos* ou *Balizas* e métodos baseados na utilização *Mapas*.

Na Localização baseada em Marcos, o sistema robótico se baseia em características detectáveis do ambiente para se localizar. As informações dos marcos detectáveis são combinadas com outras previamente conhecidas para determinar a posição do robô. Marcos podem ser divididos em *Passivos* e *Ativos*.

Marcos Ativos, também chamados de *Balizas*, enviam ativamente informações de localização, a título de exemplo ondas de rádio ou sinais de satélite. O robô que captura esses sinais consegue se localizar a partir do seu processamento. Diversos equipamentos empregam o uso desse tipo de técnica de localização, tal como os que utilizam rádio [24], ultrasom [25] e o GPS (Global System Position) [26][27]. Apesar dos sistemas de GPS serem bastante utilizados em localização, a precisão não é tão grande para ambientes internos, além do seu uso em ambientes onde não se consegue captar os sinais de satélite, ser limitado.

Já or *Marcos Passivos* não transmitem sinais ativamente, o robô precisa enxergá-los no ambiente através de sensores para receber alguma informação de posição. Os tipos de Marcos identificados estão diretamente ligados aos tipos de sensores utilizados, por exemplo, ao se detectar marcos em imagens com um sistema de visão, sensores ópticos são capazes utilizados com técnicas de processamento de imagem. Marcos passivos podem tanto ser do tipo *Artificiais* quanto do tipo *Naturais*.

Marcos Artificiais são construídos especificamente para que o robô faça uma fácil identificação dos mesmos. São postos em locais conhecidos de antemão e que são bem visíveis aos sensores do robô. Exemplos de marcos Artificiais são códigos de barras, figuras geométricas coloridas, como quadrados e círculos ou contendo algum padrão. Uma das desvantagens da utilização de Marcos Naturais é que quanto mais longe um robô de um marco, menos precisa é a sua estimativa de posição [8]. Em um sistema que utiliza visão, quando mais longe um robô se localiza de um marco , menor o marco será nas imagens do sistema e mais difícil se torna a base de uma medida de posição precisa. Se a distância for menor, as medições de posição serão mais precisas. Além dessa desvantagem, as medições podem ser imprecisas, incorretas ou mesmo ausentes por em diferentes condições de iluminação ou visibilidade parcial [8]. Contudo, os marcos artificiais podem ser usados quando o ambiente é bem estruturado e não muda muito, como em ambientes internos, escritórios e salas. Além disso, comparada com Marcos ativos, a detecção de marcos artificiais requer maior poder de processamento, além dos marcos e o ambiente precisam ser projetados para fazer o trabalho

de reconhecimento. Apesar das desvantagens, para ambientes internos os Marcos Artificiais são amplamentes utilizados, podendo ser vistos em aplicações em [28][29][30].

Marcos Naturais já fazem parte da estrutura onde robô irá operar, ou seja, são nativos do ambiente. Em ambiente internos exemplos de marcos naturais são portas, janelas, lâmpadas, enquanto em ambientes externos são estradas, árvores, placas, etc. As mesmas desvantagens dos Marcos Artificiais podem ser aplicadas nos Marcos Naturais, exceto que o ambiente não precisa ser projetado. A complexidade computacional e confiabilidade do reconhecimento é menor comparada com a de Marcos Artificiais [8], especialmente ao ar livre, quando não existe ajuda de uma geometria estruturada para se fazer um processamento de imagem e reconhecimento eficaz.

Uma vez que técnicas que usam Marcos não são especificamente projetados para fins de localização, o robô pode precisar de um número de observações antes que ele possa determinar a sua posição. Por exemplo, várias portas em uma fileira podem se parecer todas a mesma e o robô precisa de um número de observações para descobrir em qual porta ele está realmente de frente. Este problema pode ser superado ao se incluir marcos naturais que determinam de forma única uma localização, mas novamente irá aumentar a complexidade computacional. Um equilíbrio entre complexidade computacional e quantidade de marcos naturais a serem usados que pode aprendido com redes neurais.

Outro grupo de técnica de localização é a baseada em *Mapas*. Essa abordagem usa características geométricas do ambiente como as linhas que descrevem paredes em corredores ou salas. Saídas de sensores como por exemplo sonares, câmeras RGBDS, Laser Scanners, são então combinadas com estas características para localizar o robô. Há situações em que o mapa do ambiente está disponível ao robô. Logo, as medições sensoriais são adquiridas e comparadas (matching) com as informações do mapa. A partir da comparação a posição absoluta do robô é deduzida. A representação de mapas pode ser Geométrica ou Topológica. Os mapas geométricos contêm o ambiente em uma sistema de coordenada global, uma vez que os mapas topológicos contêm o ambiente sob a forma de uma rede, onde nós representam lugares no ambiente e arcos entre os nós representam ações que relacionam um local com outro. Ao se usar técnicas de correspondência de modelos para determinar a posição absoluta de um robô tem-se a desvantagem de que precisa haver informações sensoriais suficientes para serem combinadas com o mapa a fim de se determinar uma posição. Além

disso, Técnicas de correspondência de dados de sensores com mapas exigem alto poder de processamento e detecção [8].

2.3.3 Fusão de Multi-Sensores

Apesar da possibilidade de se usar as abordagens Absoluta e Relativa de localização separadamente, a partir da integração de informações de múltiplos sensores, pode-se obter uma estimativa de localização mais aprimorada. Essa abordagem se chama *Fusão de Multi-sensores*. A fusão de informação é importante uma vez que, juntas, passam a ser mais precissas, em especial quando os sensores capturam diferentes informações. Algumas características do ambiente podem estar oclusas a um sensor e não a outro, logo, a fusão pode promover uma visão mais aguçada do cenário em questão. Métodos de fusão dependem de abordagens probabilísticas, onde noções de incerteza e confiabilidade são terminologias comuns. Geralmente a aplicação de algoritmos de filtros são empregados, como os filtros de Kalman e de Partículas [8].

2.4 Sensores

Como brevemente mencionado no **Capítulo 1**, uma grande gama de sensores foram empregados para auxiliar na tarefa de localização de robôs móveis. Os do tipo *Exteroceptivos*, que capturam parâmetros externos ao robô, são largamente utilizados em conjunto com os *Proprioceptivos*, capturando informações inerente a plataforma robótica. Exemplos de sensores do *Proprioceptivos* são sistemas de Odometria, já explicados anteriormente, e bastante discutidos em trabalhos como [31][32][33]. No entanto, esses últimos tipos de sensores não são ideais para percepção de fatores externos ao robô, limitando a localização a um cenário menos abrangente. Para se ter uma visão geral da localização, o uso de sensores *Exteroceptivos* resolvem o problema.

2.4.1 Câmeras RGB-D

O foco do trabalho tem como utilização câmeras RGB-D, que são tipos de sensores *Exteroceptivos* capazes de obter informação espacial. Pode ser citado como exemplo a câmera

Kinect da Microsoft [34]. Equipado com uma câmera RGB com um emissor de raios infravermelho (IR) e com um sensor de profundidade, o Kinect foi planejado, inicialmente, para atuar como um controlador de jogos para o console Xbox 360 [34], permitindo que os usuários controlassem e interagissem com ele a partir de gestos e comandos de voz. Com esse sensor da nova geração, a comunidade que trabalha com sistemas de visão começou a desenvolver aplicações para tarefas de percepção com baixo custo.

O Kinect é um sensor RGB-D capaz de prover dados à uma frequência de até 30 Hz, combinando informação de imagem RGB e profundidade (Depth). Com esse último tipo de aquisição de dados, é possível criar um tipo de representação do mundo real chamado de Nuvem de Pontos [35], que pode ser entendida como uma coleção de pontos no sistema tridimensional de coordenadas (xyz), representando a área de captura do sensor. Este sensor é constituído pelos seguintes componentes:

- Uma câmera RGB, que armazena dados em três canais com resolução de 1280x960, permitindo a captura de cor.

- Um emissor de Raios Infravermelhos (IR) e um sensor IR de profundidade. O primeiro é responsável por emitir feixes de luz infravermelha, enquanto o segundo captura os feixes refletidos de volta para o sensor. Os feixes refletidos são convertidos em informação da profundidade, medindo a distância entre os objetos e o sensor. Isso faz com que a captura de uma imagem de profundidade seja possível.

- Um microfone multi-array, contendo quatro microfones para captar o som. Uma vez que existam quatro microfones, é possível gravar áudio, bem como encontrar a localização da fonte de som e a direção da onda sonora.

- Um acelerômetro de 3 eixos configurado para a faixa de 2G, na qual G é a aceleração da gravidade. É possível utilizar o acelerômetro para determinar a orientação do Kinect.

- Um motor tilt na base, que permite ajustar o sensor segundo o eixo horizontal.

2.5 PCL e algoritmos

Para que as Nuvens de Pontos pudessem ser aplicadas em códigos, trabalhando em tempo real, tornou-se necessária a criação de ferramentas capazes de realizar processamento dos pontos, destacando a biblioteca Point Cloud Library (PCL) [35]. De código Open Source, essa biblioteca contém uma coleção de algoritmos e ferramentas no estado da arte para processamento de dados 3D.

O uso da PCL na Visão é vasto, um grande exemplo do seu uso são os sistemas que fazem mapeamento SLAM (Simultaneaous localization and mapping) [20][36][37]. Algoritmos de SLAM são capazes de construir e atualizar mapas de ambientes, previamente, desconhecidos, como cenas de ambientes internos, devido a área reduzida e bastante controlável.

Outra grande aplicação da PCL são os algoritmos de reconhecimento de objetos para tarefas domésticas, como mostrado na Figura 2.2. Observa-se a crescente popularidade de robôs em ambientes internos, realizando tarefas junto com humanos. Por isso surgiu a necessidade de bons algoritmos de detecção e reconhecimento de objetos. Nesse ramo de pesquisa, as Nuvens de Pontos têm sido utilizadas com sucesso para tais tarefas. Trabalhos como [38][39][40] expõem um vasto uso de suas funcionalidades .

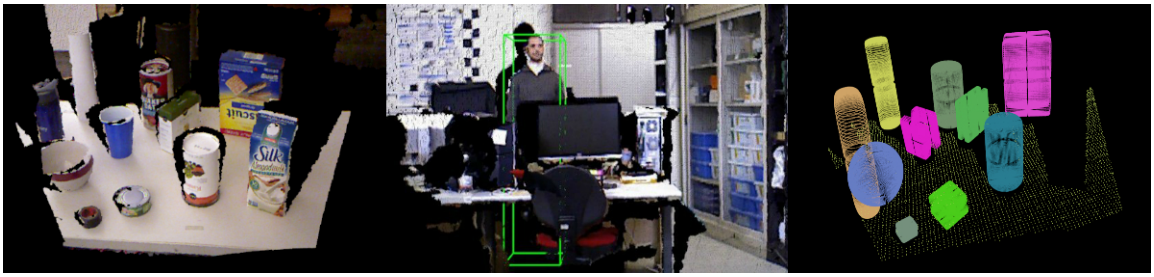

Figura 2.2: Exemplos de nuvens de pontos.

Capítulo 3

Trabalhos Relacionados

No presente capítulo serão apresentados os trabalhos relacionados. Serão explorados trabalhos relacionados à localização de Robôs Móveis, usando-se vários tipos de sensores. Serão feitas análises dos trabalhos em questão, explorando as vantagens e desvantagens de cada algoritmo.

3.1 Trabalhos do Estado-da-Arte

Diferentes tipos de sensores foram introduzidos recentemente para serem usados no problema de localização. Entre algumas abordagens aplicadas para cenários internos Chao et al. [41] propõem um sistema de localização magnética onde bobinas transmissoras magnéticas são colocadas no robô, enquanto se fixam bobinas sensoriais em um local espacial na sala interna. Mais adiante, Jung et al. [42] melhoram a localização magnética com auxílio de sinais de rádio. Em seu trabalho eles apresentam um método para resolver tanto SLAM e problema de realocação (robô sequestrado).

Um método de localização baseado em rádio e em redes de área local sem fio (WLANs) é proposto por Wu e Jen [43]. A localização do robô é estimada dinamicamente usando o filtro de partículas de busca local adaptativo (adaptive local search particle filter - ALSPF) baseado na intensidade do sinal recebido (receive signal strength - RSS). Estendendo-se para múltiplas tecnologias sem fio, Rodrigues et al. [44] adotou o uso de dispositivos compatíveis com ZigBee e Wi-Fi, analisando os resultados obtidos pela fusão de dados de odometria de robôs com estimativas de distância entre o robô e balizas sem fio com filtro de kalman

estendido (extended kalman filter - EKF). Estas distâncias são estimadas com base no RSS obtido a partir dos sinais de radiofrequência (RF) recebidos. Usando a tecnologia ZigBee para múltiplas redes sem fio, MacDougall e Tewolde [45] propõem um método de localização para um robô guia. Estes dispositivos ZigBees são usados como nós de referência com locais conhecidos, ou pontos de ancoragem, para fornecer informações de referência para auxiliar a localização do robô móvel.

Sensores ultrassônicos também são usados em sistemas de localização como o trabalho apresentado por Kim et al. [46]. Os autores desenvolveram um sistema de localização híbrida ultrassônico composto por vários transmissores ultrassônicos acoplados ao teto em posições conhecidas e vários receptores ultrassônicos situados equilateralmente no topo de um robô móvel. De modo semelhante, Qian et al. [47] introduz um algoritmo de localização de triângulos e um filtro de média para robôs móveis em ambiente internos usando rede ultrassônica. Estas abordagens mencionadas acima podem apresentar alguns desafios, em relação ao ruído, atenuação e desvanecimento da intensidade do sinal, devido às paredes, objetos, pessoas e objetos ferromagnéticos. Outra situação desafiadora é a precisão na determinação da localização, dependendo da técnica a covariação pode variar de vários centímetros.

Em relação à confiança do sensor e aos erros de medição, os sensores baseados em câmera são, de longe, os mais precisos. Além disso, os sensores visuais são capazes de recuperar as características ambientais, permitindo a computação de informações para se obter uma localização robusta do robô. Várias técnicas foram propostas como o sistema que mistura visão e odometria [33][48] por meio de marcos artificiais. Outra dessas técnicas usadas para localizar robôs é denominada visão global e é abordada por Lee et al. [49] e também por Peipei et al. [50]. Em ambas as abordagens, as câmeras montadas no teto proporcionam uma visão ampla do cenário, favorecendo o avistamento das plataformas robóticas dispostas no ambiente. No entanto, esta abordagem tem graves limitações ao ser empregada em corredores estreitos. Além disso, sensores de câmera têm a vantagem de ser de baixo custo, intuitivo para a interação humana, e capaz de gerar informações abundantes. Nas técnicas propostas em [32], [33], [48] e [51], algoritmos alternativos são usados em conjunto com a visão, como o filtro de partículas modificado [51], o filtro de Kalman [32] eo filtro de Kalman estendido mais o filtro de partículas [33]. Essas pesquisas foram realizadas para corrigir os erros que a odometria do robô pode ter acumulado ao longo do tempo.

Focando separadamente no robô, câmeras de diversos tipos podem aparecer, como as omnidirecionais [52], olho de peixe [53] ou os dispositivos RGB-D mais recentes [54]. Na abordagem baseada em marcos, a localização é realizada extraindo pontos de interesse visual no ambiente, chamados marcos, e podem ser artificiais ou naturais. Os pontos de referência são usados para atualizar a pose do robô quando detectado por seu sistema visual. Lee et al. [55] e Huletski et al. [28] usam um conjunto de marcos artificiais baseados em QR-code que são colocados em locais estratégicos. Chen et al. [29] deram preferência a utilização de um padrão impresso em um cartão como marco. A localização baseada em marco artificial também é usada em ambientes industriais [56]. Em relação aos marcos naturais, o trabalho de Sun et al. [57] encontra lâmpadas circulares no teto como os marcos e, similarmente, Alves et al. [58] usam luminárias como marcos naturais estimando a localização do robô com um Filtro de Partículas e uma abordagem de Filtro de Kalman. A detecção de pontos de referência é comumente realizada usando descritores para recuperar características visuais, ou pontos-chave, como SIFT no trabalho de Salem et al. [59] e SURF que é usado por Chang et al. [54].

Além disso, uma vez que a identificação de marcos naturais é geralmente interligada com outras classes de robótica móvel (por exemplo, localização, mapeamento e, portanto, SLAM), se deve ter a preocupação com isso ao listar os documentos que poderiam potencialmente contribuir para a classificação e notoriedade do trabalho ao máximo. Dado que o uso de câmeras de baixo custo RGB-D na identificação de pontos de referência é algo relativamente novo na literatura, a maioria das técnicas nos últimos anos apresentam soluções para o problema de localização de robôs dependentes de sensores montados nos robôs. É preciso também levar em conta que para ambientes negados por GPS, isto é, dentro de edifícios, a precisão é um fator chave para localizar um robô, como em quartos e corredores.

Liu et al. [60] propõem uma abordagem para a extração de características pertencentes a marcos naturais a partir dos dados capturados por uma varredura a laser 2D. O algoritmo extrai pontos de referência (isto é, pontos relativos a bordas, segmentos curvos de centro, entre outros) que poderiam ser alguns marcos naturais. O processo é basicamente feito através do que os autores chamam de nova abordagem de identificação geométrica de escala de curvatura. Enquanto isso, Rous et al. [61] propor uma abordagem para a navegação de um robô móvel com base na observação de marcos naturais contidos em uma determinada cena

usando uma câmera monocular anexada ao robô como o sensor principal. No algoritmo, eles aplicam uma análise na detecção de estruturas e escolhem a mais relevante para realizar a tarefa de navegação do robô. No entanto, esta detecção depende do conhecimento prévio sobre a estrutura que seria encontrada. Exemplos de tais estruturas seriam portas ou objetos que os autores escolherem. O processo começa com uma orientação seletiva baseada na Hough-transform (OHT). Este processo gera uma rede de polígonos convexos. Os polígonos que são homogêneos são segmentados e fundidos na região de referência e, finalmente, uma extração é feita a fim de descobrir regiões de objetos que têm seus comportamentos anteriormente conhecidos.

Xiong e Choi [62] apresentam um algoritmo para corrigir a posição estimada da plataforma robótica com base na comparação entre os marcos naturais e as referências atribuídas no cenário. A abordagem usa uma câmera de olho de peixe e o processo de correção de erro faz uso de uma estimativa sobre a relação entre a identificação do marco natural juntamente com a distorção dos dados de calibração do sensor. O comportamento padrão do robô memoriza alguns pontos durante o caminho no cenário em que esses pontos são acreditados estarem relacionados a marcos naturais.

Outra abordagem encontrada na literatura é apresentada por Chai et al. [63], que se caracteriza pela identificação de linhas retas e cores homogêneas dentro do ambiente. A partir dessas características de área bem particulares o algoritmo seria capaz de identificar objetos para usar esses objetos como marcos naturais. O trabalho compara pontos selecionados. O detector de bordas Harris também é aplicado neste processo de pontos de classificação, tendo como orientação para a seleção de marcos naturais uma combinação de linhas retas para formar polígonos convexos. Observa-se que Rous et al. [61] fazem uso de polígonos, bem como acima mencionado. Em contraste, Chai et al. Também demonstram um método para detectar linhas para simplificar a criação de polígonos.

Quando se trata de reconhecimento de portas, o algoritmo proposto por Yimin Zhou [64] et al. consegue trabalhar com imagens de profundidade adquiridas do sensor Kinect. A vantagem de se trabalhar com tais imagens seria a robustez às variações de iluminação. No algoritmo proposto é feita uma análise do Histograma da imagem de profundidade, conseguindo-se extrair as linhas verticais das portas. Já nos trabalhos de Derry e Argall [65], Borgsen et al. [66] e Kuramochi et al.[67] a identificação das portas é feita usando-se o algoritmo *RANSAC*

[68], o qual procura planos dominantes na cena com um método de estimativa de modelos.

O reconhecimento de escadarias é abordado em trabalhos como de Lee et al. [69], onde um óculos de visão 2D estéro é utilizado com um algoritmo que utiliza características HAAR [70] e o algoritmo de aprendizagem Adaboost [71]. No trabalho de Romic et al. [72] e [73] também é utilizada visão 2D, mas no primeiro utiliza-se o algoritmo de detecção de bordas Canny para a detecção dos pisos da escada, já no segundo utiliza-se gradiente de Sobel na imagem em escala de cinza para a mesma tarefa. Observa-se que esses trabalhos tratam somente de identificação em 2 Dimensões, diferente do trabalho proposto por Sinha et al. [74], no qual utiliza-se o sensor Kinect e o framework PCL para a detecção de escadarias em 3 Dimensões. No trabalho é proposta uma nova estrutura chamada Mapa de Colunas baseada no alinhamento do vetor de gravidade e no eixo vertical da nuvem, compondo incrementalmente formações de escadarias.

Observou-se que a maioria dos trabalhos tradicionais encontrados na literatura fornece detecção de pontos de referência considerando um ambiente 2D semiestruturado. Quando se trata de detecções de marcos utilizando estruturas em 3 Dimensões, algoritmos da literatura não combinam abordagens de portas e escadas, bem como não mostram a posição dos marcos no espaço, diferentemente da abordagem proposta. Esta diferença e outros detalhes do algoritmo proposto neste trabalho são explicados no Capítulo a seguir.

Capítulo 4

Metodologia

No presente capítulo será introduzida a abordagem proposta por trás do desenvolvimento do trabalho. Será explorada cada parte do processo da elaboração do mesmo, começando desde a primeira versão até a última implementação da mesma. Diversas modificações e descobertas foram feitas ao longo dessa evolução, a qual será explorada por diagramas, pseudocódigos e equações.

4.1 Formalização do Problema

A grande necessidade da construção de um sistema de localização por marcos naturais se dá pela minimização da alteração do homem no ambiente de trabalho dos robôs. Também tem-se como objetivo a criação de algoritmos mais parecidos com os métodos e ações realizados por humanos na tarefa de se localizarem em ambientes previamete conhecidos ou recém percebidos. Como os humanos, a existência de objetos naturais ao ambiente ajuda ao robô na tarefa de localização. O grande desafio dessa área é encontrar soluções que facilitem o mapeamento e a interpretação de um cenário, permitindo que robôs tenham aptidão para detectar, localizar, reconhecer e reconstruir geometricamente os objetos do meio no qual estão inseridos, bem como na execução de tarefas de navegação e manipulação.

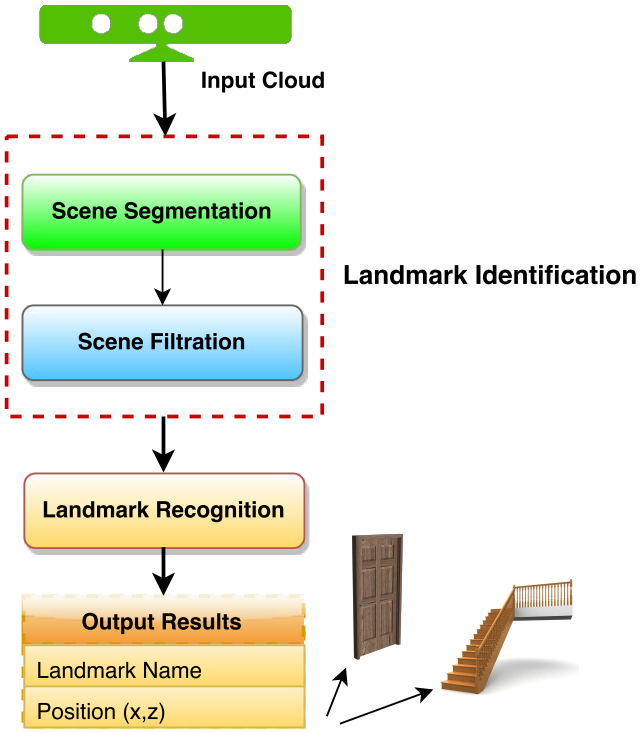

Figura 4.1: Diagrama contendo a estutura geral do algoritmo.

4.2 Algoritmo

A principal característica do algoritmo proposto é o funcionamento pleno para identificação de marcos naturais usando sensores RGBD, diferente de trabalhos como [60][61], onde uma visão de profundidade não era capaz de ser recuperada. Seguindo a Figura 4.1, tem-se como entrada uma imagem recuperada do sensor RGBD Kinect, onde essa imagem é transformada em uma nuvem de pontos a partir da biblioteca *Point Cloud Library*. Os dois módulos seguintes são responsáveis pelo tratamento da nuvem, onde o primeiro, chamado de *Landmark Identification*, será encarregado de segmentar e filtrar a cena, onde serão removidos e filtrados objetos desinteressantes para o processo . Já o segundo módulo, chamado de *Landmark Recognition*, se encarregará de fazer o reconhecimento em si, efetuando cálculos matemáticos com o objetivo de estimar características únicas dos marcos circundantes, distinguindo os marcos uns dos outros. Por fim, o objeto encontrado e reconhecido é retornado, juntamente com sua posição espacial no eixo (x,z).

Uma grande vantagem do algorimo proposto é inexistência de uma fase de treinamento para a realização do Reconhecimento dos Marcos Naturais, todo o processo é feito no mo-

mento de execução. Diante deste avanço, o algoritmo se torna genérico para qualquer ambiente em que possa ser posto em prática, reduzindo tempo e custos consideravelmente.

O algortimo final tem como passos os seguintes enumerados:

1. Receber a Nuvem de Entrada
2. Identificação de Marcos
    (a) Segmentação da Cena
    (b) Filtragem da Cena
3. Reconhecimento de Marcos
4. Retornar objeto e sua Posição Espacial

O passo 3 do algoritmo, referente ao Reconhecimento de Marcos, passou por mudanças durante a pesquisa realizada, obtendo-se melhores resultados ao longo do tempo. Antes se fazia o uso de uma árvore de decisão para apontar qual objeto o sensor havia detectado, agora os passos descritos acima são os utilizados no momento. Todo o processo de implementação e evolução do passo de Reconhecimento será explicado mais abaixo na sua sessão respectiva.

4.3 Identificação

Seguindo o tratamento modular, a primeira parte do algoritmo trata da Identificação dos Marcos Naturais. Para melhor tratamento didático, o cálculo de Identificação foi ainda subdividido em duas etapas principais. A primeira etapa lida com a segmentação da cena recuperada, já a segunda é responsável pela aplicação de subetapas de filtragem, gerando uma nuvem final que seguirá para a fase de reconhecimento (Figura 4.1). Os passos em verde vistos na caixa delimitadora destinam-se a preprocessar a nuvem, retornando apenas pontos de interesse prontos para tratamento e cálculos com fins de identificar Marcos Naturais. Após a segmentação, a nuvem processada é submetida a operações de filtragem (passos em azul) designadas para descartar dados ruidosos. (Ver Figura 4.2)

4.3.1 Segmentação

Os passos de segmentação podem ser vistos na Figura 4.2, onde estão listados em verde. O processo de segmentação inicia com o sensor Kinect da plataforma robótica móvel capturando imagens do ambiente. A PCL (*Point Cloud Library*) é usada para transformar a imagem em uma nuvem de pontos, estrutura essa escolhida para o tratamento de dados no presente trabalho. Juntamente com a nuvem adquirida a partir do sensor, o filtro *Narrow Down* é usado para reduzir o tamanho da nuvem para um retângulo horizontal. Esta redução é realizada para eliminar muitos dados não utilizados e repetidos sobre o eixo vertical, uma vez que paredes, portas, colunas, têm a mesma estrutura de cima para baixo, acelerando assim o processamento. Logo após o estreitamento da visão, o algoritmo passa a procurar planos dominantes na cena com o método de estimativa de modelos *RANSAC* [68]. Após os principais planos da cena serem identificados, eles são movidos para uma nova estrutura, separada da nuvem original, em outras palavras os planos são recortados da cena principal, onde passarão pelo passo que calculará suas bordas. Feito isso, a nova estrutura contendo apenas as bordas dos planos principais da cena geral é passada adiante para os passos em azul da Figura 4.2. Percebe-se que o algoritmo trabalha com percepção de planos, base geral para a identificação de marcos naturais. Sejam portas, janelas ou escadas, todos esses marcos possuem planos que podem ser identificados pelo algoritmo.

Filtro Narrow Down

Como mencionado, o método de identificação de marcos naturais adotado neste trabalho tem seu primeiro módulo de processamento destinado a preparar a imagem adquirida, retornando pontos que têm características específicas de borda, para a detecção do marco. Por esta razão, adotou-se o filtro Narrow Down que visa reduzir os dados, diminuindo assim a visão do robô em uma caixa de retângulo horizontal fino de acordo com o sistema de coordenadas Kinect (Figura 4.3). Por meio da PCL foi utilizada a função *PassThrough* nos eixos X e Y. Os parâmetros utilizados no filtro podem ser vistos na Tabela 4.1.

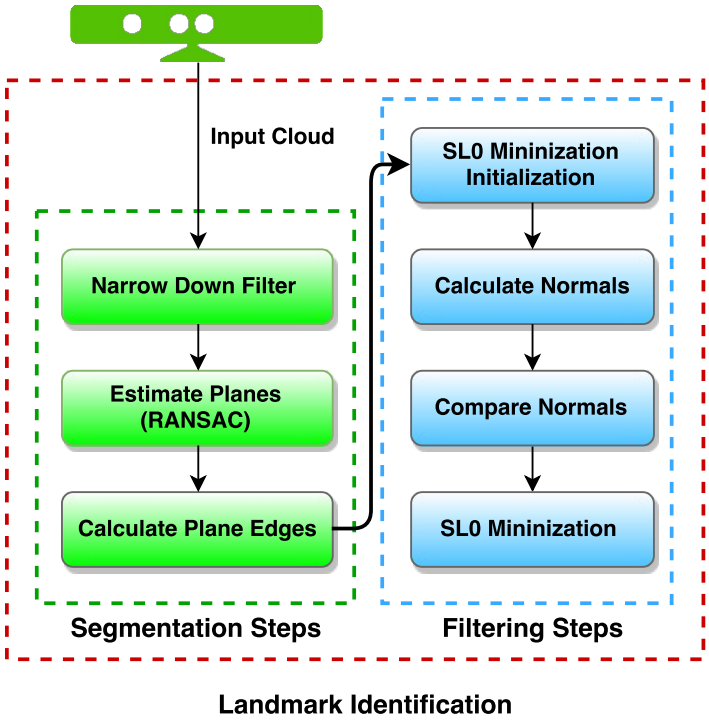

Figura 4.2: Diagrama de Identificação.

Tabela 4.1: Parâmetros do filtro Narrow Down. Onde X e Y são os eixos de observação e os valores correspondem a área que o sensor irá se restringir a observar.

Eixo	**Valores Mínimos**	**Valores Máximos**
Y	-0.6m	0.2m
X	-0.6m	0.6m

Estimativa de Planos

Depois de reduzir a nuvem de pontos para o tamanho desejado, o passo seguinte é identificar planos na estrutura encontrada. Uma vez que o algoritmo está em busca de marcos naturais e esses marcos podem ser entendidos como paredes, corredores, colunas, bordas, etc, é aconselhável procurar alguma geometria regular (ou seja, objetos formados por planos). Assim, uma estimativa de plano predominante é realizada pelo algoritmo *RANSAC* disponível na biblioteca PCL. Foi realizada uma modificação no código proposto para encontrar apenas os planos (maiores) mais adequados disponíveis na cena (o maior número de pontos *inliers*)

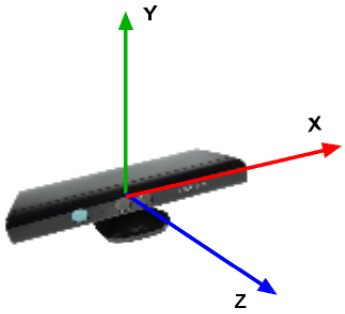

Figura 4.3: Sistema de Coordenadas do Kinect.

dentro do filtro narrow down. Portanto, nesta etapa, pequenos planos encontrados em objetos como cadeiras e caixas são ignorados deixando apenas as características de planos maiores como paredes. A Figura 4.4 demonstra uma imagem filtrada após o filtro narrow down e o algoritmo RANSAC modificado. Depois que os planos dominantes na cena são identificados, são movidos para uma nova nuvem de pontos de modo que, no final da etapa da estimativa dos planos, a saída resulta em uma nuvem contendo somente os planos dominantes da cena original.

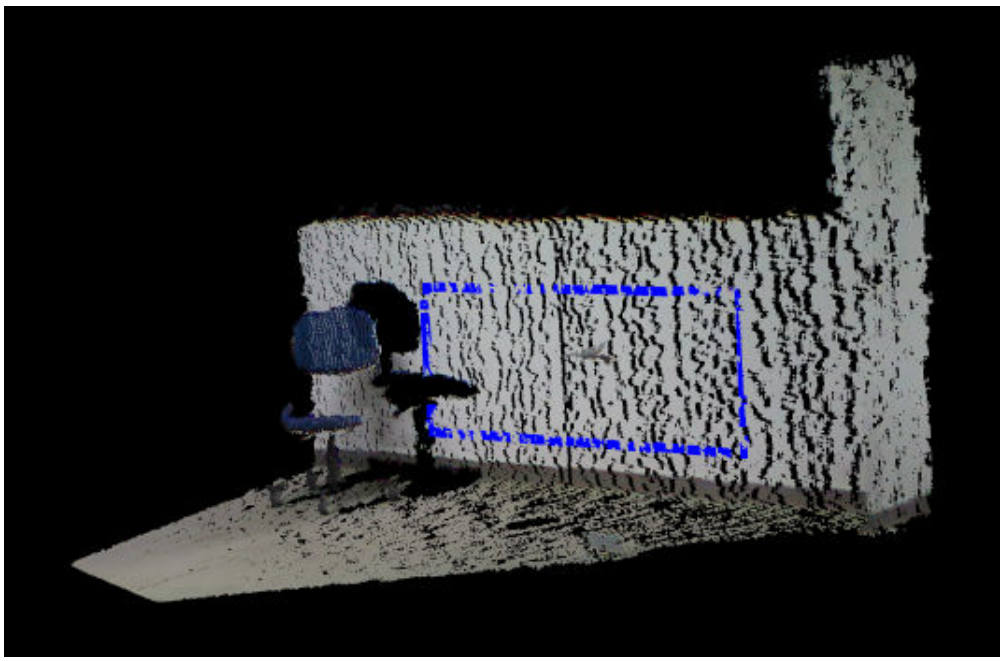

Figura 4.4: Resultado após filtro Narrow Down e RANSAC modificado.

Cálculo das bordas dos Planos

Em seguida, a nuvem contendo apenas os planos estimados é então usada como entrada para o cálculo das bordas. Este passo destina-se a minimizar os pontos que virão a ser trabalhados e, como resultado, as bordas calculadas podem ser vistas na Figura 4.5. O cálculo de bordas dos planos é realizado usando o algoritmo *Concave Hull* disponível na PCL. No entanto, a Figura 4.5 também demonstra que as etapas de segmentação não são robustas, dependendo da orientação do robô, inclinação do objeto e iluminação do ambiente, criando distorção nos planos e pontos externos. Assim, a nuvem precisa ser submetida novamente a outros processos de filtragem.

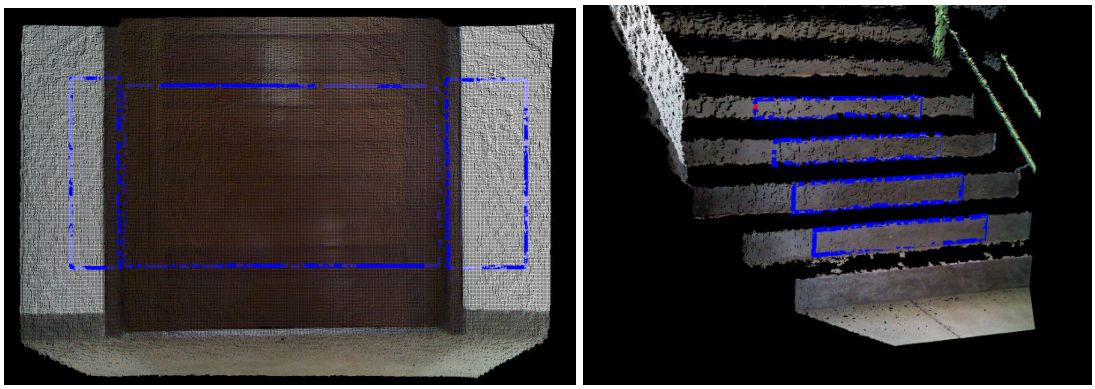

Figura 4.5: Resultados do cálculo de bordas.

4.3.2 Filtragem

Após a segmentação, foi inicializado o algoritmo de minimização de $1$0-norm (SL0), que irá diminuir os pontos remanescentes que não têm as características que permitem classificá-los como pontos de borda e foram selecionados para serem parte da nuvem devido ao sensor ou ruído ambiente. Posteriormente, o vetor normal dos pontos na nuvem de bordas é calculado e comparado com o ponto normal dos pontos circundantes. Pontos onde as normais circundantes não são paralelas às anteriores ou próximas são classificadas como pontos de borda. Finalmente, a nuvem de bordas é minimizada através do algoritmo SL0. Este algoritmo minimiza o desvio padrão nos ângulos entre as normais. As etapas seguintes descrevem o método utilizado para filtrar a nuvem resultante do passo de segmentação, estimando assim as normais de cada parte da trama observada e minimizando os seus outliers. Os passos

detalhados da etapa de filtragem podem ser vistos em azul na Figura 4.2.

Inicialização do Algoritmo SL0

Como mencionado acima, essa parte trata da inicialização do algoritmo de minimização. Essa subetapa tem como objetivo fazer uma filtragem geral por meio da função $pcl :: PassThrough$ da PCL. Novamente o campo de visão é reduzido para se gerar um quadrado e reduzir as nuvens de bordas dos planos somente a linhas desejadas. Observa-se a diferença entre a Figura 4.5 e a Figura 4.6 que o campo de visão foi reduzido, reduzindo-se as linhas de pontos mais externas, porém mantendo as linhas de bordas intactas.

Cálculo da Normal dos Pontos

O passo seguinte no processo de filtragem é percorrer a nuvem de pontos e calcular o vetor normal para cada ponto. Para isso, assume-se que os pontos retornados estão sempre na superfície dos objetos, de modo que seus vetores normais estão apontando para fora da superfície. O cálculo de normais é realizado usando a função $pcl :: computePointNormal$, disponível no framework PCL. Observa-se que o sensor RGB-D contém algumas imprecisões, introduzindo dados ruidosos, às vezes. Portanto, há uma necessidade de uma etapa de minimização para filtrar os pontos e seus valores normais que são considerados outliers.

Um ponto é considerado um outlier se seu ângulo de curvatura entre o plano dominante for diferente de 90 graus, considerando um limiar (tolerância) $sigma\_min = 0.01$, ou seja, procuram-se pontos exatamente ou bem próximos a 90 graus. Por este motivo, o algoritmo que será descrito na próxima secção é utilizado para resolver este problema e estimar as normais dos pontos que são adequados para serem utilizados no passo de identificação do marco.

Comparação de Normais

Depois de se calcular as normais com o algoritmo anterior, uma comparação entre as normais é realizada, ponto a ponto da nuvem resultante. Por iteração através da nuvem de bordas, o algoritmo pode comparar cada par de normais e recuperar o ângulo de diferença entre eles. Duas normais com um ângulos parecidos são consideradas candidatas a ser de pontos de

borda, pois como se aproximam a 90 graus, são pontos do plano. Os pontos-chave resultantes são pontos que se encontram na mesma superfície de um canto de um plano.

Minimização SL0

A próxima subetapa é filtrar os pontos-chave resultantes usando o algoritmo de minimização de 10-norm. Seguindo o trabalho anterior de [75], escolheu-se a definição da norma l_0 de um vetor $\mathbf{s} = [s_1,\ cdots, s_N]$ como o número de componentes não-zero de $\mathbf{s}$. Para calcular isso, primeiro define-se:

$$\nu(s) = \begin{cases} 1 & , s \neq 0 \\ 0 & , s = 0 \end{cases} \tag{4.1}$$

Então a norma l_0 de $\mathbf{s}$ pode ser escrita como:

$$\|\mathbf{s}\|_0 = \sum_{i=1}^{N} \nu(s_i) \tag{4.2}$$

É claro que esta norma é descontínua, uma vez que ν é uma função descontínua. Isso é contornado empregando uma aproximação suavizada de $\nu(s)$. Várias funções poderiam ser usadas para isso. Foi escolhida uma função Gaussiana de média zero, devido à sua diferenciabilidade, definida como:

$$f_\sigma(s) = exp\left(\frac{-s^2}{2\sigma^2}\right), \tag{4.3}$$

tendo:

$$\lim_{\sigma \to 0} f_\sigma(s) = \begin{cases} 0 & , s \neq 0 \\ 1 & , s = 0 \end{cases} \tag{4.4}$$

Consequentemente, $\lim_{\sigma \to 0} f_\sigma(s) = 1 - \nu(s)$, e portanto se definido:

$$F_\sigma(\mathbf{s}) = \sum_{i=1}^{N} f_\sigma(s_i), \tag{4.5}$$

pode-se ter

$$\lim_{\sigma \to 0} F_\sigma(\mathbf{s}) = N - \|\mathbf{s}\|_0. \tag{4.6}$$

Portanto, tomado $[N - F_\sigma(\mathbf{s})]$ como uma aproximação de $\|\mathbf{s}\|_0$:

$$\|\mathbf{s}\|_0 \approx N - F_\sigma(\mathbf{s}), \tag{4.7}$$

onde σ é o balanço entre a precisão e a suavidade da aproximação.

Da equação 4.7, a minimização da norma $\|\mathbf{s}\|_0$ é equivalente a maximização de $F_\sigma(\mathbf{s})$ para um σ suficientemente pequeno. Para simplicidade de notação, a equação do Problema Normal pode ser escrita como $x = A * B$, onde B é o vetor de peso e A é a matriz de medidas. Também é necessário, como ponto de partida, a pseudoinversa da matriz A, que pode ser denotada como A_{pinv}, da qual pode ser encontrada a primeira suposição de B na forma $B = A_{pinv} * x$.

A suposição de B é tomada e a multiplicada por uma fração ponderada de uma aproximação de sua norma l_0, para cada célula do vetor. Aqui, a aproximação é definida pelo desvio padrão de uma função gaussiana de média zero, simplesmente chamada de σ. Então, de B é subtraido um residual encontrado projetando o erro na matriz pseudoinversa. Depois de repetir isto um número satisfatório de vezes, o valor do sigma é diminuído. Todo o procedimento é repetido até o sigma atingir a referência σ, que foi parâmetro para a precisão. O pseudocódigo para isso pode ser visto no Algoritmo 1.

A Figura 4.6 apresenta quatro elementos (três portas e uma escada) com os pontos-chave resultantes das etapas do cálculo de normais, comparação de normais e minimização SL0.

4.4 Reconhecimento

Depois de estimar um conjunto de pontos candidatos, como visto na Figura 4.6, a última parte do algoritmo deve classificar o tipo de marco presente na frente do sensor RGB-D. Os tipos de marcos escolhidos para identificação neste trabalho foram portas e escadarias. Observa-se que estes são marcos comumente vistos em um ambiente interno (isto é, o prédio da faculdade) e não são bem abordados na literatura sobre o reconhecimento de marcos naturais aplicados à localização de robôs. Para realizar a identificação utilizou-se informações de profundidade e de geometria da nuvem resultante. Como mencionado no início do Capítulo, o processo de Reconhecimento passou por modificações ao longo do desenvolvimento. Na primeira instância do algoritmo, se usava uma abordagem mais geométrica e intuitiva. Os dois algoritmos de Reconhecimento, sendo o segundo mais robusto, serão explicados a seguir.

Data: B=SL0(A,x,σ_{min},A_{pinv})

Result: Weight vector B that solves $x = A * B$

Initialization;

$B = A_{pinv} * x$;

$\sigma = 2 * max(abs(B))$;

$\mu_0 = 2$;

$L = 3$;

$\sigma_{decrease\_factor} = 0.5$;

$\sigma_{min} = 0.01$;

while $\sigma < \sigma_{min}$ **do**

  for $i = 1 : L$ **do**

    $\delta = B. * exp(-abs(B).^2/\sigma^2)$;

    $B = B - \mu_0 * \delta$;

    $B = B - A_{pinv} * (A * B - x)$;

  end

  $\sigma = \sigma * \sigma_{decrease\_factor}$;

end

Algorithm 1: Algoritmo SL_0 de minimização de normas

4.4.1 Algoritmo 1

As informações de profundidade e de geometria são inseridas, juntamente com o número de planos, em uma árvore de decisão, responsável por verificar se o marco é uma porta ou uma escada. Conhecendo o número atual de planos visíveis, foram calculados seus pontos de centroide e verificou-se se é uma escada ou uma porta comparando a distância entre os planos visíveis no eixo Z (profundidade). Se os planos são paralelos e a distância entre eles ao longo do eixo Z é distintamente maior, então o marco é uma escada caracterizada por pelo menos dois planos em cima uns dos outros. A diferença de profundidade entre os planos em uma caixa de escada é distintamente maior do que no caso da entrada.

Para analisar os tipos de portas, calculou-se os pontos mais próximos e mais distantes da nuvem resultante. Então, desenhou-se uma linha entre esses pontos e calculou-se a projeção da diagonal formada no chão. Para diferenciar uma porta aberta de uma fechada ou semi-

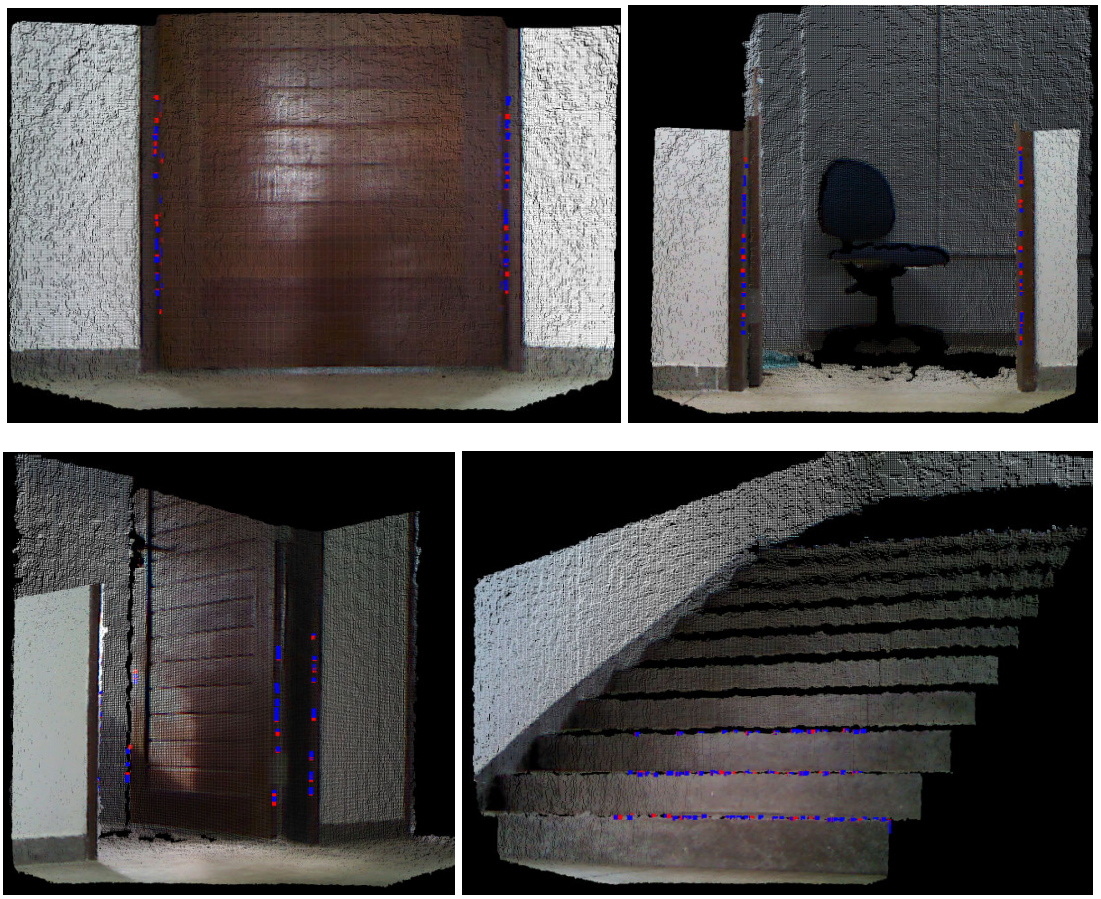

Figura 4.6: Imagens resultantes das etapas do cálculo de normais, comparação de normais e minimização SL0. Os elementos são uma porta fechada, porta aberta, porta aberta com uma visão de $45°$ graus e uma escadaria com uma visão de $45°$ graus.

aberta, apenas verificou-se a distância calculada e o número de planos visíveis. O padrão desses recursos é único e esses marcos têm uma geometria simples de ser analisada.

4.4.2 Algoritmo 2

Como o algoritmo da seção anterior tratava o Reconhecimento de uma maneira simples e intuitiva, modificações foram feitas para que o torna-se mais robusto e embasado matematicamente. A Figura 4.7 mostra os passos do atual algoritmo de Reconhecimento.

O método aqui aplicado consiste em filtrar mais ainda a nuvem resultante para recuperar as características geométricas intrínsecas em relação às portas e escadas. Em primeiro lugar, a nuvem resultante do último passo da Segmentação é projetada no plano XY (plano de

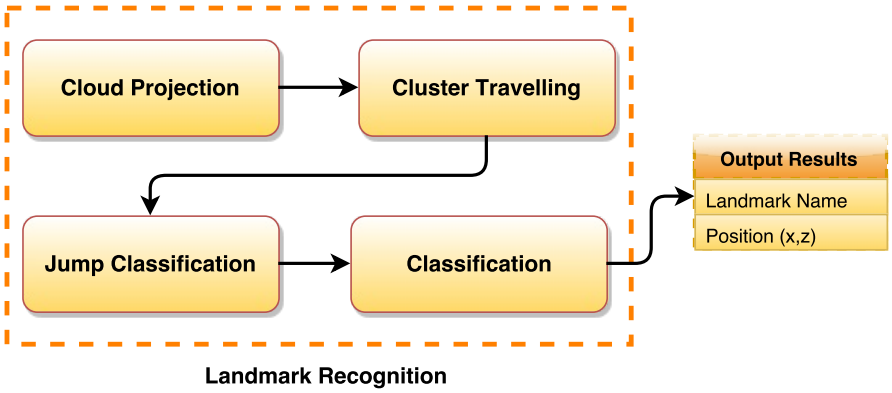

Figura 4.7: Diagrama de Reconhecimento.

chão), criando clusters de pontos. Isso pode ser ilustrado observando-se os pontos-chave na Figura 4.6. Se forem projetados no plano do chão, os pontos da mesma linha vertical irão ser agrupados uns muito próximos aos outros, gerando clusters de pontos. No caso de portas como as quinas são verticais serão criados quatro clusters parecidos, já no caso de escadas, como as quinas são horizontais, será criado um número de clusters de acordo com o número de degraus observados. Esses últimos clusters serão linhas no plano XY. A Figura4.8 ilustra a nuvem após a projeção dos pontos-chave no plano do chão.

Logo após a projeção, todos os pontos da nuvem são percorridos em uma forma iterativa, extraindo sua informação espacial. A informação espacial dos pontos é usada para classificar se eles pertencem a um determinado tipo de marco. As características das portas ou escadas são adquiridas quando a diferença entre os pontos é observada. Esse comportamento foi chamado de *"salto"* de cluster. A partir dessas características o algoritmo classifica os saltos, podendo diferenciar entre escadas ou portas. A parte de reconhecimento proposta tem quatro etapas:

- Projetar a Nuvem no Plano do Chão;
- Percorrer Clusters (Saltos);
- Classificar Saltos (Portas ou Escadarias);
- Retornar objeto mais dominante (Reconhecimento Final);

Figura 4.8: Imagens resultantes da Projeção.

4.4.3 Projeção da Nuvem

A projeção da nuvem resultante do passo de Identificação segue um critério que afirma que um dado ponto de nuvem $P(x', y', z')$ e sua projeção P' determinam uma linha na qual o vetor de direção s coincide com o vetor normal N do Plano de projeção E (Figura 4.9). A projeção do ponto P em p' segue a Matriz:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Observe que todos os pontos de nuvem são projetados no plano XZ, ou seja, pela definição do sensor, um plano é paralelo ao solo se o sensor estiver na posição vertical. Para as características das portas, observa-se que a nuvem resultante após a projeção forma uma

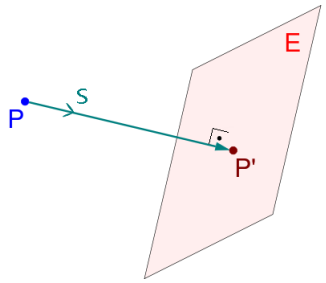

Figura 4.9: Projeção Ortogonal.

região com pontos agrupados (Figura 4.8). No entanto, nas características da escada a nuvem resultante é um conjunto de linhas (Figura 4.8 parte inferior). Estes grupos de pontos e linhas são chamados clusters de porta e de escada, respectivamente. A projeção da nuvem é feita por meio do objeto da PCL *ProjectInliers*.

4.4.4 Percorrer Clusters (Saltos)

Pode-se notar que nas projeções resultantes das portas os pontos são acumulados bem próximos, em aglomerados bastante definidos, dentro de uma circunferência de raio r, ao contrário dos aglomerados de escadas que são linhas bem definidas. Após a projeção, a nuvem é submetida a um processo iterativo no qual cada ponto é visitado e sua posição espacial em XYZ é armazenada. Após a visitação de 15 pontos (este valor foi determinado empiricamente), cada ponto consecutivo é analisado e comparado com o anterior e com os pontos seguintes. Este processo é repetido até que o último ponto seja percorrido. Quando a diferença entre pontos consecutivos corresponde a um certo limiar, é afirmado que a iteração deixou o cluster e o ponto atual pertence a outro cluster. Isso é chamado de **"Salto"** de cluster.

4.4.5 Classificação do Salto e Reconhecimento Final

Como mencionado anteriormente, uma classificação é definida quando ocorre um **salto**. Cada cluster é percorrido e deve ser classificado de acordo. Por exemplo, antes do salto, o ponto P_1 (x_1, z_1) é submetido a uma comparação com um ponto anterior P_0 (x_0, z_0) e ao próximo ponto P_2 (X_2, z_2). Se for observado que o ponto anterior P_0 é semelhante (valores de x e z são proximos em uma escala menor que 1 cm) ao ponto atual P_1 e os valores do próximo ponto P_2 aumentaram (valores de x e z são maiores que 10 cm), o salto é classifi-

cado como um salto de porta, assim como o cluster. Isto é verdadeiro porque os pontos que pertencem aos conjuntos de porta estão concentrados em um espaço confinado. Os saltos de escada são classificados quando P_0 é muito menor (o valor de x é muito menor que 1 cm de diferença) que P_1, P_2 é maior (x e y são maiores que 1cm)que P_1 e há uma grande variação na coordenada de profundidade (z é maior que 25cm) de P_2 de P_1, sugerindo P_2 pertence a outro cluster (Ver o quadro inferior direito da Figura 4.8).

Capítulo 5

Avaliação Experimental

Este capítulo apresenta os experimentos realizados para avaliar a abordagem proposta, nesta seção foram realizados dezesseis testes do sistema completo, demonstrando a sua efetividade. Os cenários dos testes foram corredores e salas do Laboratório de Sistemas Embarcados e Robótica (LaSER), assim como do Centro de Informática da UFPB. Os experimentos dos sistema geral foram divididos em dois conjuntos, o primeiro relacionado a primeira implementação do algorimo de Reconhecimento, o segundo relacionado a segunda implementação. Cada conjunto de experimento foi analisado separadamente em subsessões, culminando em uma análise das duas no final do capítulo.

Figura 5.1: Turtlebot - LASER/UFPB.

5.1 Configurações dos Experimentos

Os testes foram realizados utilizando-se um Um robô TurtleBot 2 teleoperado com um sensor Kinect 1 (Figura 5.1). O TurtleBot e o sensor Kinect estão conectados a um Notebook Lenovo rodando o ROS (Robotic Operation System) no Ubuntu 14.04: CPU Intel Core i5 de 1,86 GHz, 500 GB de HD e 4 GB de RAM. Nenhum outro tipo de sensor foi utilizado nos experimentos.

5.2 Experimentos de Identificação

A primeira parte dos experimentos foi realizada com o módulo de Identificação. Este experimento consistiu na realização de simulações dos passos em verde e azul da Figura 4.2. Foi visto na seção 4.3 que a divisão entre os passos em verde e azul seria para ajudar na compreensão didática, diferenciando-os em passos de Segmentação e Filtragem, respectivamente. No entanto, nesta seção todos os passos serão tratados como fazendo parte do sistema geral de Identificação.

5.2.1 Filtro Narrow Down

Como primeira amostragem, a imagem capturada pelo sensor Kinect é transformada em uma nuvem de pontos pela PCL e logo é processada com o filtro *pcl::PassThrough*. No Anexo A pode ser observado o código fonte da implementação do Filtro Narrow Down. Os objetos da filtragem são iniciados na linha 8, a filtragem é feita nos eixos X e Y a partir da linha 29. Na Figura 5.2 observam-se a nuvens de pontos originais obtidas pelo sensor. Todas essas nuvens originais são passadas para a variável de nuvem $cloud$, onde são processadas pelos filtros e copiadas para a outra variável de nuvem $filteredCloud1$. Depois de se aplicar o algoritmo de estreitamento as nuvens resultantes podem ser vistas na Figura 5.3.

Observa-se que na realização destes experimentos, independente da distância do robô do marco em questão, o plano do chão sempre será excluído, devido a inclinação da câmera e o seu campo de visão.

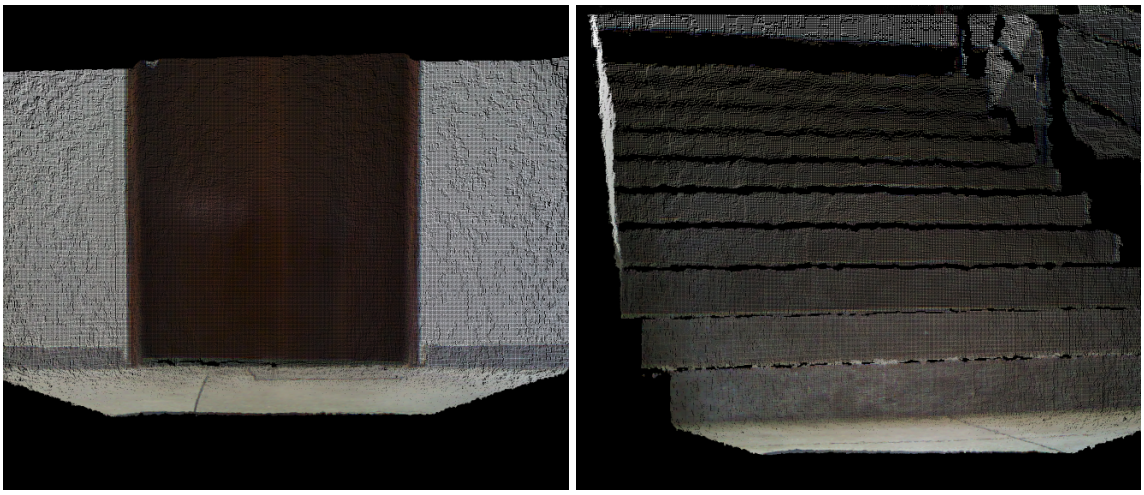

Figura 5.2: Nuvens originais de Porta e Escadaria.

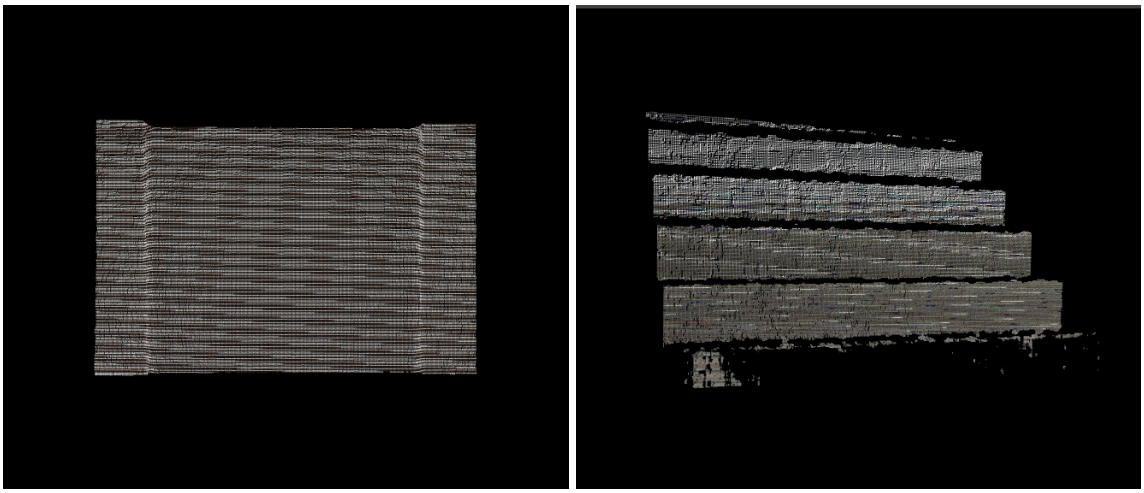

Figura 5.3: Nuvens estreitadas pelo filtro Narrow Down.

5.2.2 Estimativa de Planos

Logo após o estreitamento da nuvem, o próximo passo consiste em calcular um número mínimo de planos na nuvem para realizar a busca por marcos. O início do algortimo de *RANSAC* modificado pode ser visto no Anexo A na linha 50. O algoritmo inicia criando as variáveis para a segmentação. A partir da linha 55 o algoritmo entra em um loop no qual buscará por planos contendo mais do que 15000 pontos, casando com o modelo $pcl :: SACMODEL_PLANE$. A nuvem que será tratada será a nuvem estreitada no passo anterior, observando-se esse passo na linha 59. Caso um número mínimo de planos seja encontrados casando com os critérios anteriores, eles são extraidos da nuvem original para

uma nova nuvem contendo apenas planos chamada $plane$, passos entre as linhas 79 e 85 do algoritmo. A Figura 5.4 mostra as nuvens estreitadas da Figura 5.3 após a estimativa de planos.

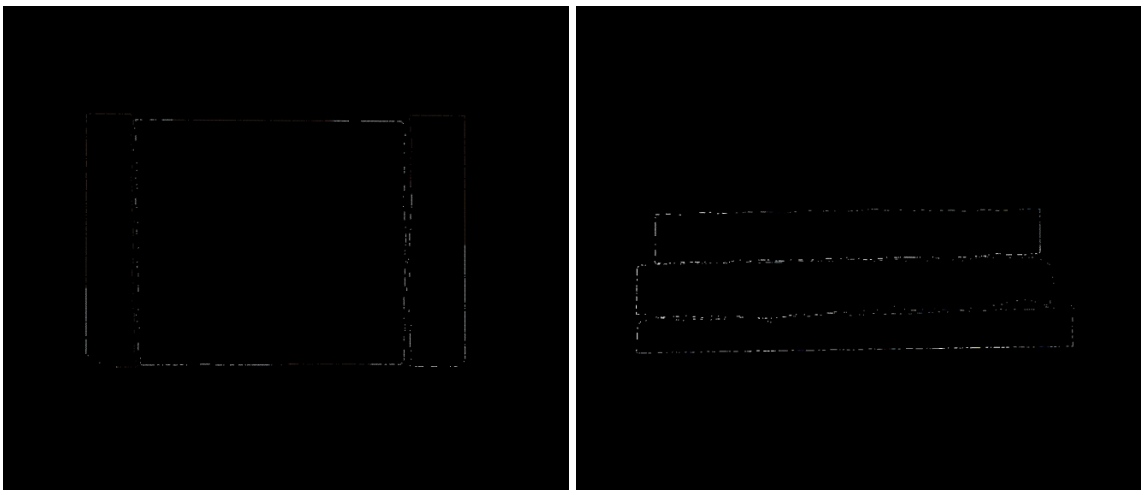

Figura 5.4: Nuvens após a estimativa de planos.

5.2.3 Cálculo das bordas do Planos

O cálculo das bordas é realizado pela função *pcl::ConcaveHull* da PCL, essa parte do algoritmo tem como objetivo restringir os planos encontrados no passo anterior apenas a suas bordas externas, encontrando o seu casco côncavo. As operações são feitas com a nuvem contendo apenas os planos, podendo ser vistas entre linhas 95 e 104 do algoritmo no Anexo A. A nova nuvem chamada $hullCloud$ (linha 104) conterá apenas as bordas dos planos da imagem da nuvem original.

5.2.4 Inicialização do Algoritmo SL0

Com intuito de se reduzir as bordas mais externas da nuvem de bordas, é feita uma filtragem na altura e largura por meio da função $pcl :: PassThrough$. Os resultados da filtragem podem ser comparados observando-se as Figuras 5.5 e 5.6.

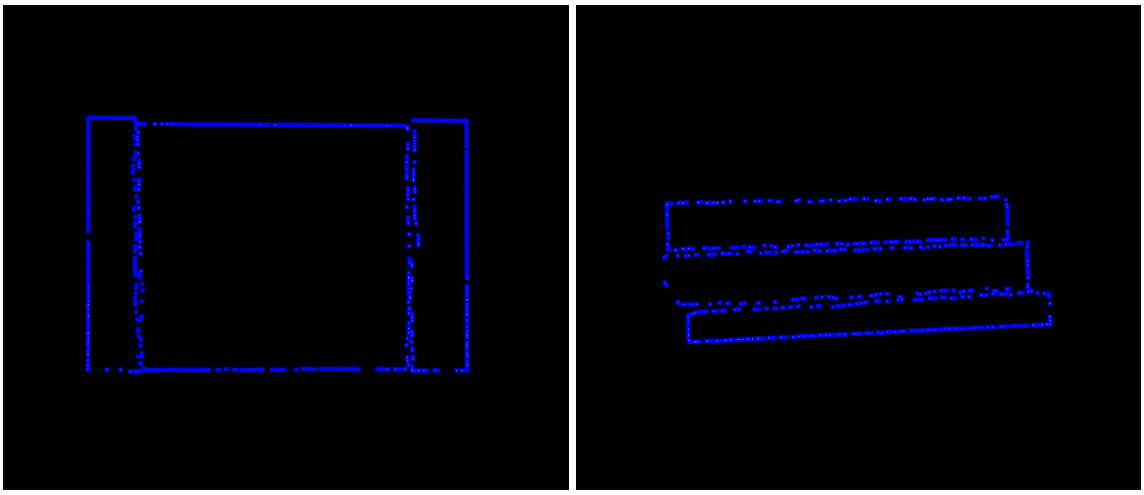

Figura 5.5: Nuvens após serem calculadas as bordas dos planos.

Figura 5.6: Nuvens após a nova filtragem.

5.2.5 Cálculo, Comparação de Normais e Minimização SL0

Com um laço iterador a nuvem do passo anterior é percorrida ponto por ponto, calculando-se o vetor normal do ponto atual e do próximo. A partir da linha 165 do Anexo A pode-se observar todos os passos feitos para a obtenção do vetor normal. A comparação dos vetores é feita e caso o limiar esperado seja atingido, ou seja, os pontos se pareçam, eles são considerados pontos candidatos. O processo pode ser visto a partir da linha 165 do Anexo A. Para a minimização foi usada uma lógica simples, mostrada entre as linhas 192 e 200 do anexo A. O resultado dos pontos candidatos é mostrado na Figura 5.7.

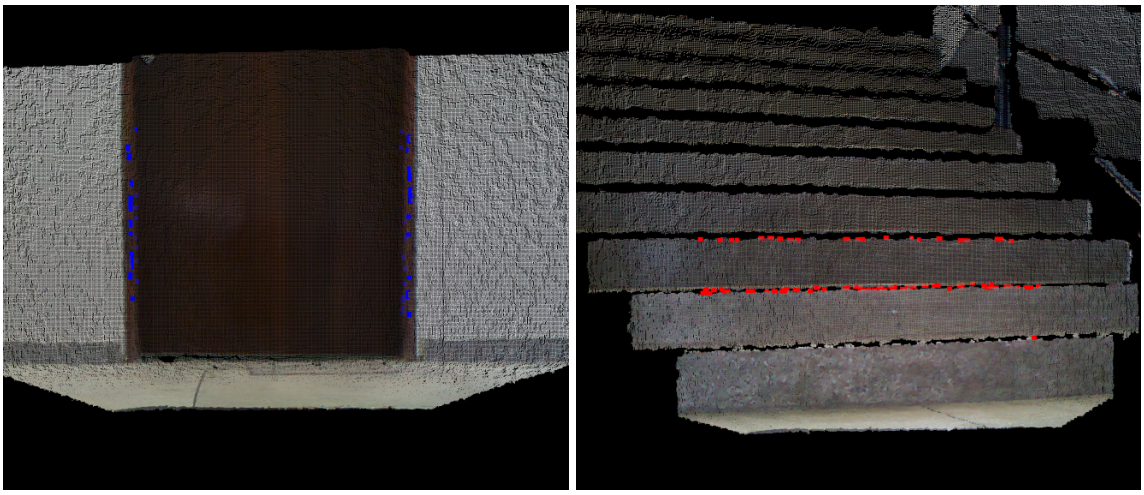

Figura 5.7: Nuvens após serem calculadas os pontos de interesse.

5.3 Experimentos de Reconhecimento

Finalmente, para a validação do sistema de Identificação e Reconhecimento foram feitos dezesseis experimentos dos dois módulos em conjunto, oito com o Algoritmo 1 e mais 8 com o Algoritmo 2. Cada teste de Algoritmo será analisado separadamente, logo em seguida uma comparação dos dois resultados será feita.

5.3.1 Algoritmo 1

Para aplicar a parte de Reconhecimento com o Algoritmo 1 utilizou-se a nuvem minimizada contendo os pontos candidatos. O experimento realizado levou em consideração Portas Abertas, Portas Semiabertas, Portas Fechadas e Escadarias, posicionadas na frente do sensor e avaliadas durante 1000 frames de observação. A diferença entre o estado $Semiaberto$ eo estado $Aberto$ é que no primeiro a porta não está totalmente aberta, no segundo a porta está. Todos os casos foram testados com um ângulo de observação de $0°$ e $45°$ graus, de acordo com o esquema abaixo:

1. Marco: Porta
   (a) Característica: Aberta
      i. Ãngulo de frente ($0°$ graus)
      ii. Ângulo lateral ($45°$ graus)

(b) Característica: Semiaberta

i. Ãngulo de frente (0° graus)

ii. Ângulo lateral (45° graus)

(c) Característica: Fechada

i. Ãngulo de frente (0° graus)

ii. Ângulo lateral (45° graus)

2. Marco: Escada

(a) Ãngulo de frente (0° graus)

(b) Ângulo lateral (45° graus)

A abordagem realizou com sucesso a detecção de portas abertas, portas fechadas, portas semiabertas e escadarias, com relativamente poucas características perdidas ou incorretamente identificadas. A Figura 5.8 apresenta a detecção de todas as quatro características em ambos os ângulos medidos. Durante cada frame observado foi feito um registro do marco corretamente identificado, incorretamente ou perdido, gerando a Tabela 5.1, a qual apresenta as Matrizes de Confusão para os ângulos de 0° e 45° graus de observação.

Tabela 5.1: Matriz de Confusão da Indentificação (Algoritmo 1).

Medidas com 0° graus (Frente)				
Real	**Previsto**			
	Porta Aberta	**Porta Semiaberta**	**Porta Fechada**	**Escada**
Porta Aberta	1000	0	0	0
Porta Semiaberta	0	997	0	3
Porta Fechada	0	0	1000	0
Escada	0	3	1	989
Medidas com 45° graus (Lado)				
Real	**Previsto**			
	Porta Aberta	**Porta Semiaberta**	**Porta Fechada**	**Escada**
Porta Aberta	998	0	0	0
Porta Semiaberta	0	941	0	59
Porta Fechada	0	0	1000	0
Escada	0	2	3	916

Na Tabela 5.1 atingiu-se uma média de 99.5% de verdadeiros positivos no processo de identificação dos marcos, quando observados com um ângulo de $0°$ graus de deslocamento. Quando observados com um ângulo de $45°$ graus , a média de verdadeiros positivos diminuiu para 96.3%. Após uma inspeção adicional, esses falsos positivos eram frequentemente provenientes do ruído no sensor Kinect, causando uma identificação incorreta dos planos do marco.

Tabela 5.2: Média das Distâncias medidas (Algoritmo 1)

	Ângulo (○)	**Dist. Real Centroide Z(m)**	**Dist. Real Centroide X(m)**	**Dist. Medida Centroide Z(m)**	**Dist. Medida Centroide X(m)**
Porta Aberta	0	1.2	0	1.109	0.071
Porta Semiaberta	0	1.6	0	1.578	0.051
Porta Fechada	0	1.4	0	1.405	0.018
Escada	0	1.2	0	1.142	0.078
Porta Aberta	45	1.6	0	1.664	0.066
Porta Semiaberta	45	1.5	0	1.456	0.024
Porta Fechada	45	1.7	0	1.701	0.086
Escada	45	1.4	0	1.293	0.086

Além disso, a Tabela 5.2 apresenta as distâncias reais e medidas do sensor RGB-D para o centroide de cada marco. Esta Tabela apresenta as distâncias nos eixo Z e X entre o sensor RGB-D e o marco detectado. Demonstra-se na Tabela 5.3 que o erro absoluto médio máximo no eixo Z (profundidade) foi de 0,11m, relacionado a Escadaria medida com $45°$ graus de ângulo de observação. Quando observam-se mais ou menos degraus a média do centroide geral muda, sendo o caso no angulo de $45°$ graus, no qual o sensor fica muito sensível as detecções de planos. No eixo X, o erro absoluto médio máximo foi inferior a 0,09m quando observou-se a Porta Fechada e Escadarias, ambos com $45°$ grau de ângulo de deslocamento.

Finalmente, a Tabela 5.4 apresenta os valores de Acurácia e Precisão de ambos os ângulos de observação ($0°$ e $45°$ graus) dos quatro objetos identificados. Podemos ver nesta Tabela que a abordagem proposta no Algoritmo 1 tem uma Acurácia de 97.8% como valor mais baixo, quando feitas medições de Escadarias no ângulo de $45°$ graus. Na Precisão, a aproximação foi de 93.9% de exatidão, sendo o valor de observações de Escadarias no ângulo de $45°$ graus.

Tabela 5.3: Erro Absoluto (Algoritmo 1).

	Ângulo (○)	**Erro Absoluto Z(m)**	**Erro Absoluto X(m)**
Porta Aberta	0	0.091	0.071
Porta Semiaberta	0	0.021	0.051
Porta Fechada	0	0.006	0.018
Escada	0	0.058	0.078
Porta Aberta	45	0.064	0.066
Porta Semiaberta	45	0.044	0.024
Porta Fechada	45	0.001	0.086
Escada	45	0.11	0.086

Tabela 5.4: Valores de Acurácia e Precisão (Algoritmo 1).

Medidas com $0°$ **graus**		
	Precisão %	**Acurácia** %
Porta Aberta	100	100
Porta Semiaberta	100	99.8
Porta Fechada	99.9	99.9
Escada	99.7	99.8
Medidas com $45°$ **graus**		
	Precisão %	**Acurácia** %
Porta Aberta	100	100
Porta Semiaberta	99.7	98.4
Porta Fechada	99.7	99.9
Escada	93.9	97.8

5.3.2 Algoritmo 2

A análise do Algoritmo 2 seguiu a mesma do seu predecessor, dividindo os marcos em Portas Abertas, Portas Semiabertas, Portas Fechadas e Escadarias na mesma esquematização de ângulos (Ver esquema do Algoritmo 1). A diferença principal foi a implementação e o número de amostras realizadas nos testes, sendo dessa vez usados 5000 frames de observação. A Tabela 5.5 apresenta as matrizes de Confusão para ambos os ângulos de observação, $0°$ e $45°$ graus. Na Tabela 5.5 observa-se uma média atingida de 98.9% de Verdadeiros Positivos no Reconhecimento de marcos naturais (sendo portas ou escadas) quando observados com $0°$ graus de ângulo de deslocamento. Quando observados com $45°$ graus de ângulo, a média dos Verdadeiros Positivos diminuiu para 98.04%.

Além disso, a Tabela 5.6 apresenta as distâncias reais e medidas do sensor RGB-D para

Tabela 5.5: Matriz de Confusão da Indentificação (Algoritmo 2).

Medidas com 0° graus (Frente)				
Real	**Previsto**			
	Porta Aberta	**Porta Semiaberta**	**Porta Fechada**	**Escada**
Porta Aberta	5000	0	0	0
Porta Semiaberta	0	5000	0	0
Porta Fechada	0	0	5000	0
Escada	0	0	161	4791
Medidas com 45° graus (Lado)				
Real	**Previsto**			
	Porta Aberta	**Porta Semiaberta**	**Porta Fechada**	**Escada**
Porta Aberta	4983	0	0	17
Porta Semiaberta	0	4979	0	21
Porta Fechada	0	0	4999	1
Escada	0	0	167	4647

o centroide de cada marco. Demonstra-se através da Tabela 5.7 que o erro absoluto médio máximo no eixo Z (profundidade) foi 0,02m, relacionado a Escadaria medida com 45° graus de ângulo de observação. No eixo X, o erro absoluto médio máximo foi inferior a 0,04m quando observou-se a Porta Semiaberta com 45° graus de ângulo de deslocamento.

Tabela 5.6: Média das Distâncias medidas (Algoritmo 2)

	Ângulo (○)	**Dist. Real Centroide Z(m)**	**Dist. Real Centroide X(m)**	**Dist. Medida Centroide Z(m)**	**Dist. Medida Centroide X(m)**
Porta Aberta	0	1.4	0	1.41	0.004
Porta Semiaberta	0	1.8	0	1.81	0.004
Porta Fechada	0	1.5	0	1.49	0.008
Escada	0	1.2	0	1.191	0.022
Porta Aberta	45	1.6	0	1.583	0.014
Porta Semiaberta	45	1.5	0	1.482	0.04
Porta Fechada	45	1.5	0	1.493	0.021
Escada	45	1.3	0	1.28	0.007

Finalmente, a Tabela 5.8 apresenta os valores de Precisão e Acurácia de ambos os ângulos de observação (0° e 45° graus) dos quatro objetos identificados. Pode observar-se nesta Tabela a nova abordagem tem uma Acurácia de $98,9\%$ como valor mais baixo, quando feitas medições de Escadarias no ângulo de 45° graus. Na Precisão, a aproximação foi de 96.5% de exatidão, sendo o valor de observações de Escadarias no ângulo de 45° graus.

Tabela 5.7: Erro Absoluto (Algoritmo 2).

	Ângulo (○)	**Erro Absoluto Z(m)**	**Erro Absoluto X(m)**
Porta Aberta	0	0.01	0.004
Porta Semiaberta	0	0.01	0.004
Porta Fechada	0	0.01	0.008
Escada	0	0.009	0.022
Porta Aberta	45	0.017	0.014
Porta Semiaberta	45	0.018	0.04
Porta Fechada	45	0.007	0.021
Escada	45	0.02	0.007

Tabela 5.8: Valores de Acurácia e Precisão (Algoritmo 2).

Medidas com 0° graus		
	Precisão $\%$	**Acurácia** $\%$
Porta Aberta	100	100
Porta Semiaberta	100	100
Porta Fechada	100	100
Escada	96.7	99.1
Medidas com 45° graus		
	Precisão $\%$	**Acurácia** $\%$
Porta Aberta	99.6	99
Porta Semiaberta	99.5	99
Porta Fechada	99.9	99.1
Escada	96.5	98.9

5.3.3 Comparação e Considerações

Após serem feitas as análises dos dois Algoritmos de Reconhecimento, montou-se dois gráficos para comparar as respectivas Precisões e Acurácias. A Figura 5.10 mostra as Precisões de ambos os algoritmos comparadas em ambos os ângulos de 0° e 45° graus. Já a Figura 5.11 mostra as Acurácias.

Na Figura 5.10, os valores menores em Precisão na Escadaria do Algoritmo 2 em relação ao Algoritmo 1, foram percebidos devido ao fato de que alguns pontos no agrupamento de escada podem ser agrupados quando a projeção força duas bordas de etapas de escada juntos, de modo que o algoritmo pode concluir que existem alguns grupos de portas. Observa-se que técnicas de filtragem podem ser aplicadas para minimizar esta situação e garantir melhores resultados.

Na Figura 5.11, os valores de Acurácia ficaram bem próximos, salvo para os observações de Porta Aberta e Porta Fechada nos ângulos de 45° graus. No entanto essas diferenças foram mínimas se consideradas uma Acurácia geral. Para concluir, pode-se afirmar que as duas abordagens geraram resultados satisfatórios, onde na primeira foram obtidos baixos resultados em Precisão para certos marcos, que foram resolvidos na segunda abordagem. Apesar da segunda abordagem ter mais embasamento matemático, onde foi criado um novo algoritmo para definição dos marcos, ela apresentou baixa Precisão em marcos onde a primeira tinha um bom resultado. Desse modo, uma maneira de se resolver tais problemas e aumentar a taxa de Precisão geral seria fundir o melhor dos dois algoritmos em uma única abordagem, podendo ser explorada em trabalhos futuros.

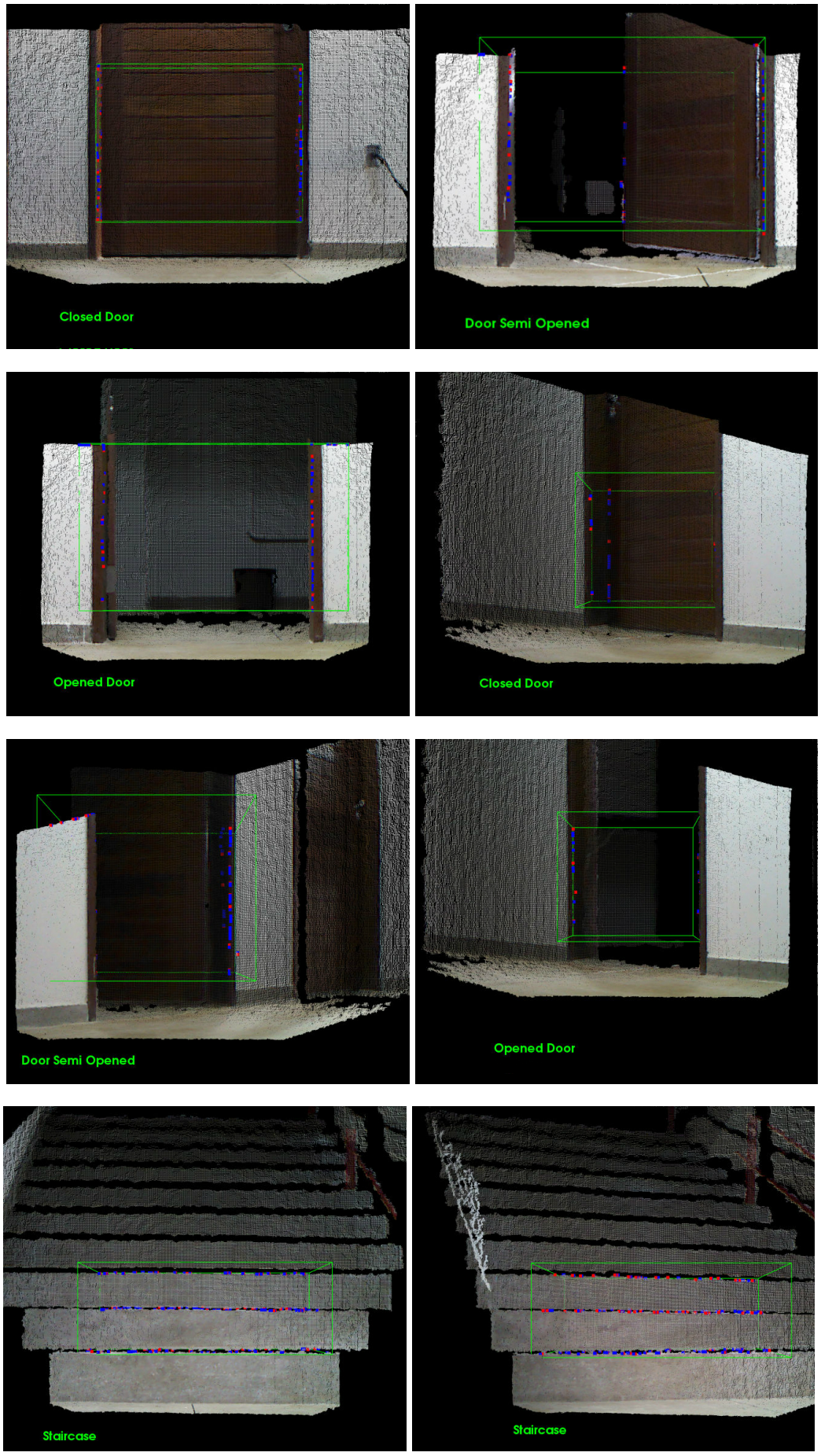

Figura 5.8: Resultados do Reconhecimento com o Algoritmo 1.

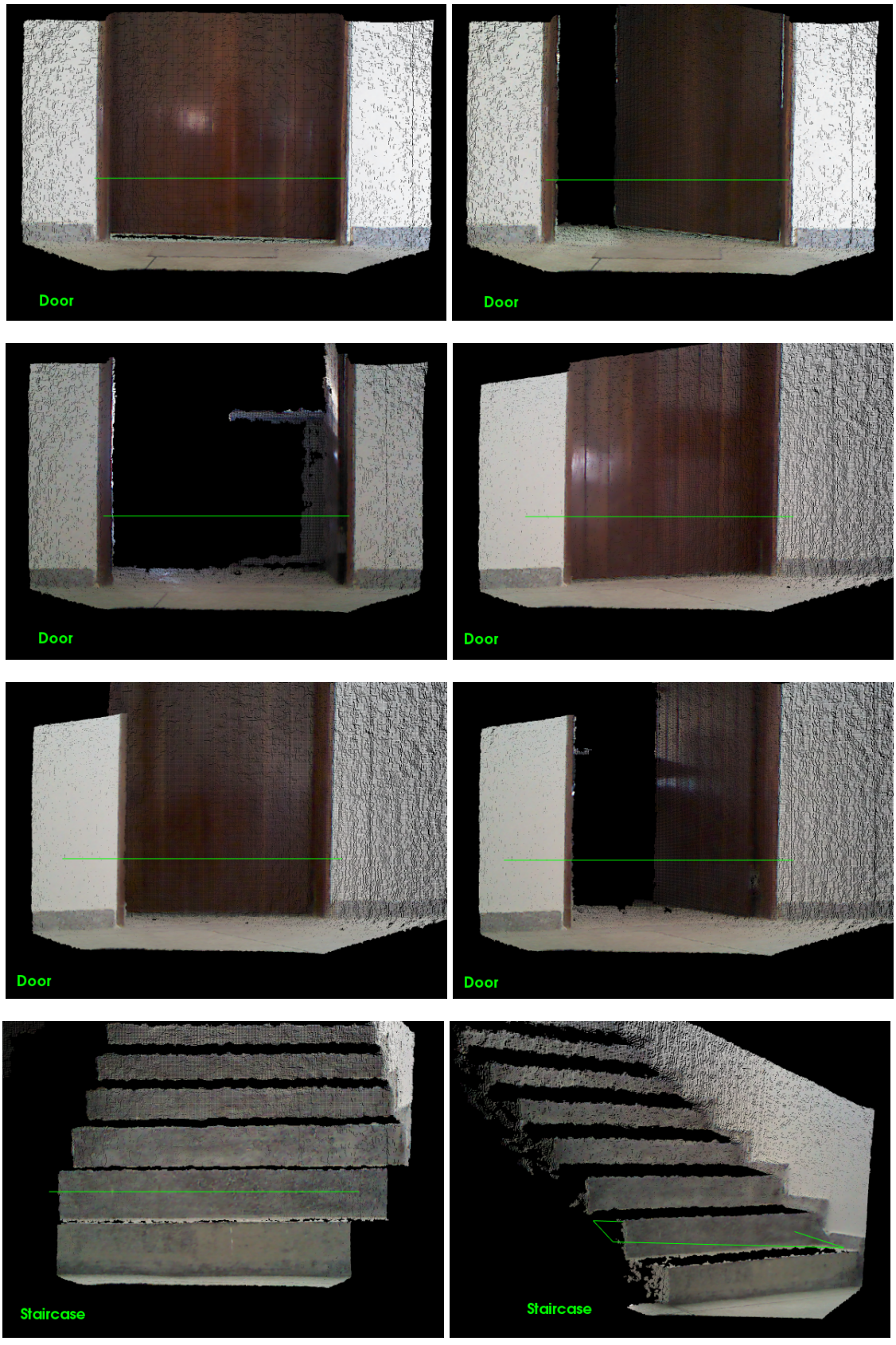

Figura 5.9: Resultados do Reconhecimento com o Algoritmo 2.

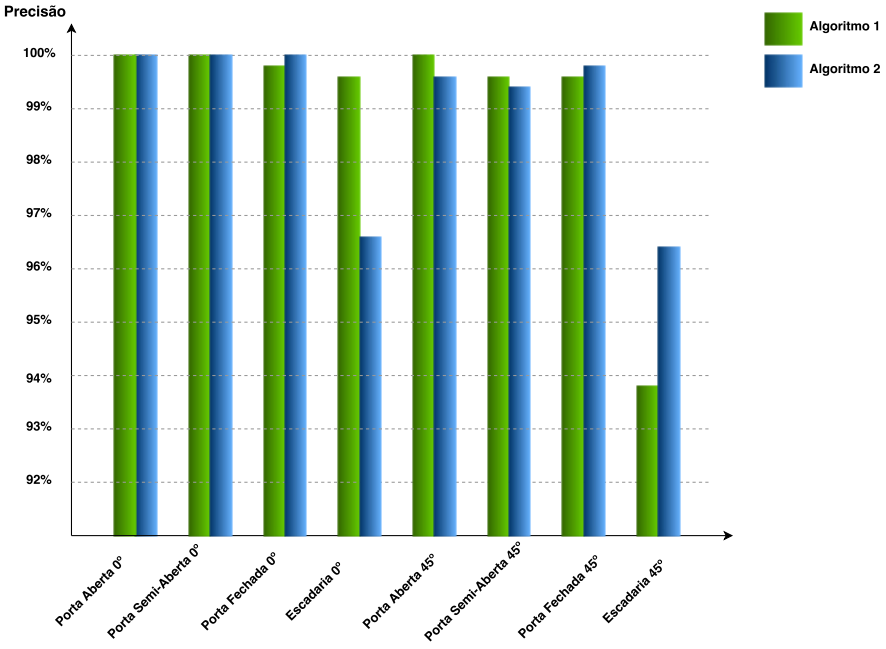

Figura 5.10: Gráfico da comparação entre Precisão do Algoritmo 1 e 2 para ambos os ângulos ($0°$ e $45°$ graus).

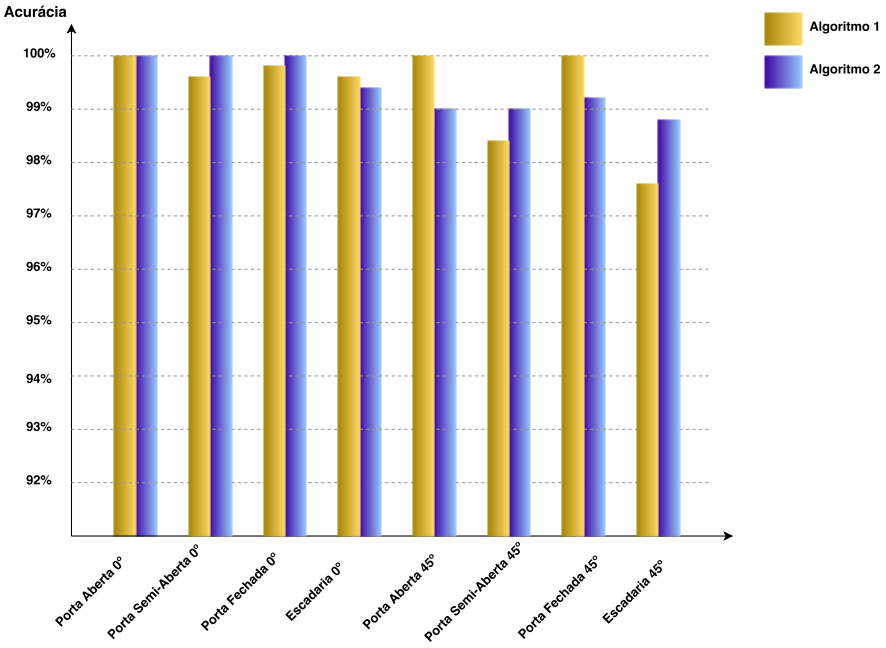

Figura 5.11: Gráfico da comparação entre Acurácia do Algoritmo 1 e 2 para ambos os ângulos ($0°$ e $45°$ graus).

Capítulo 6

Conclusão

Uma representação espacial precisa é primordial para a Localização de Robôs Móveis. Este trabalho propôs um método robusto para detectar portas e escadas usando dados de nuvem de pontos 3D. Foi apresentada aqui uma abordagem genérica de identificação e reconhecimento de marcos naturais, com base no tratamento do processo de agrupamento e aplicação da minimização do ruído. O uso de informações 3D é importante quando a altura, a profundidade e a distância ao solo precisam ser confirmadas se a plataforma do robô estiver carregando carga extra, como no caso de robôs de serviço. Além disso, através da detecção da posição e orientação de recursos, trajetórias personalizadas podem ser planejadas e executadas com eficácia, bem como a localização do marco natural pode ser aplicado.

A abordagem proposta é realizada on-line sem a necessidade de uma etapa de treinamento antes do reconhecimento do marco. As experiências realizadas demonstram que a abordagem teve um sucesso de 100% na classificação de Portas (independente do estado da porta) e um sucesso de $95,8\%$ na classificação de Escadarias quando o robô está na frente do Marco. Quando o robô está em uma orientação de $45°$ graus em relação ao marco, a abordagem atinge uma média de $99,74\%$ de sucesso na classificação de Portas e um sucesso de $92,94\%$ na classificação de Escadarias em um conjunto de 5.000 quadros capturados pelo Câmera do robô.

Além das deficiências relacionadas ao sensor, que estão além do escopo desta pesquisa, existem algumas situações que não poderiam ser tratadas por este algoritmo. Pode-se quebrar essas situações em dois grupos principais: deficiências ambientais e deficiências relacionadas ao algoritmo. As deficiências ambientais são causadas por iluminação e oclusões. Como

o caso de Porta Semiaberta demonstrou, o algoritmo é robusto à oclusão. No entanto, a iluminação ambiental ainda é um problema a ser resolvido. O grupo final de deficiências está relacionado ao algoritmo. Em grande parte, elas derivam das funções utilizadas na abordagem. Estas projeções de linha às vezes podem ser confundidas em casos que ocorre uma porta dupla ou um corredor tão reto como uma porta. Há outro caso peculiar quando há duas portas lado a lado separadas por uma pequena porção de parede. Neste caso, o plano frontal será o plano da parede enquanto os dois planos laterais serão os planos das portas que estão em uma maior profundidade. Neste caso, o algoritmo irá identificar a pequena parede como uma porta fechada. Este caso especial será tratado numa versão posterior do trabalho.

6.1 Trabalhos Futuros

Como trabalho futuro, pretendemos modificar o método de classificação utilizando uma fase de reconhecimento supervisionado de alto nível em paralelo com a parte de reconhecimento atual, a fim de melhorar nossa técnica. Visando o aplicativo de localização, também pretendemos misturar essa abordagem com a identificação de objetos como latas de lixo, bebedouros e outros objetos presentes nos corredores do prédio. Também pretendemos comparar essa abordagem modificada com outras abordagens que necessitam de treinamento, como K-Neighbouring mais próximo, Bayes e redes neurais, além de estar fora do escopo do trabalho atual. Finalmente, pretendemos aplicar o reconhecimento de marcos naturais aos algoritmos SLAM, onde as marcas podem ser usadas para lidar com problemas como robôs sequestrados, mapeamento de bordas, detecção de fechamento de loop e realocação.

Bibliografia

[1] Karel Capek. Rossum's Universal Robots. `http://capek:misto:cz/english/`, 1920. Acessado em: Janeiro, 2017.

[2] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1989.

[3] L.R. Silva. Análise e programação de robôs móveis autônomos da plataforma eyebot. Dissertação, Departamento de Engenharia Elétrica, Universidade Federal de Santa Catarina, 2001.

[4] Samuel T. Pfister. Algorithms for mobile robot localization and mapping, incorporating detailed noise modeling and multi-scale feature extraction. Thesis, California Institute of Technology.

[5] T.P. Nascimento. *Coordinated Multi-Robot Formation Control*. PhD thesis, Faculdade de Engenharia, Universidade do Porto.

[6] Robin R. Murphy. *Introduction to AI Robotics*. The MIT Press.

[7] Nourbakhsh I. R. Scaramuzza D. Siegwart, R. *Introduction to autonomous mobile robots*. Cambridge, Mass: MIT Press.

[8] R. Negenborn. Robot localization and kalman filters. Master thesis, Institute of Information and Computing Sciences, Utrecht University, 2003.

[9] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. Collaborative multi-robot localization. In *Proceedings of the 23rd Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence*, KI '99, pages 255–266, London, UK, UK, 1999. Springer-Verlag.

[10] Nasa. Mars Exploration Rovers.

[11] M. J. Schuster, C. Brand, S. G. Brunner, P. Lehner, J. Reill, S. Riedel, T. Bodenmüller, K. Bussmann, S. Büttner, A. Dömel, W. Friedl, I. Grixa, M. Hellerer, H. Hirschmüller, M. Kassecker, Z. C. Marton, C. Nissler, F. Ruess, M. Suppa, and A. Wedler. The lru rover for autonomous planetary exploration and its success in the spacebotcamp challenge. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 7–14, May 2016.

[12] T. Seco, C. Rizzo, J. Espelosín, and J. L. Villarroel. A robot localization system based on rf fadings using particle filters inside pipes. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 28–34, May 2016.

[13] T. Pobkrut, T. Eamsa-ard, and T. Kerdcharoen. Sensor drone for aerial odor mapping for agriculture and security services. In *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 1–5, June 2016.

[14] S. García, M. E. López, R. Barea, L. M. Bergasa, A. Gómez, and E. J. Molinos. Indoor slam for micro aerial vehicles control using monocular camera and sensor fusion. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 205–210, May 2016.

[15] P. Sousa, A. Ferreira, M. Moreira, T. Santos, A. Martins, A. Dias, J. Almeida, and E. Silva. Isep/inesc tec aerial robotics team for search and rescue operations at the eurathlon challenge 2015. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 156–161, May 2016.

[16] J. Pérez, F. Caballero, and L. Merino. Integration of monte carlo localization and place recognition for reliable long-term robot localization. In *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 85–91, May 2014.

[17] P. Jensfelt. *Approaches to Mobile Robot Localization in Indoor Environments*. PhD

thesis, Department of Signals, Sensors and Systems Royal Institute of Technology, Stockholm, Sweden, 2001.

[18] C. G. Bezerra. Localização de um robô móvel usando odometria e marcos naturais. Dissertação, Universidade Federal do Rio Grande do Norte, 2004.

[19] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *2012 IEEE International Conference on Robotics and Automation*, pages 1691–1696, May 2012.

[20] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *2012 IEEE International Conference on Robotics and Automation*, pages 1691–1696, May 2012.

[21] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[22] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99 – 141, 2001.

[23] Ingemar J. Cox. *Blanche: Position Estimation for an Autonomous Robot Vehicle*, pages 221–228. Springer New York, New York, NY, 1990.

[24] K. Fujii, W. Wang, Y. Kaneko, Y. Sakamoto, H. Arie, and S. Sugano. Accurate indoor positioning using imes radio. In *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1966–1971, Aug 2015.

[25] J. Zhang and K. Pan. Ultrasound/dr integrated localization and pose tracking for an indoor mobile robot. In *2015 IEEE International Conference on Information and Automation*, pages 847–851, Aug 2015.

[26] Y. Sakamoto, K. Kodaka, T. Ebinuma, K. Fujii, and S. Sugano. Active-localization methods for mobile robots in a coarsely structured environment with floor-embedded rfid tags and indoor gps. In *2012 IEEE International Conference on Mechatronics and Automation*, pages 539–545, Aug 2012.

[27] T. Saito and Y. Kuroda. Mobile robot localization using multiple observations based on place recognition and gps. In *2013 IEEE International Conference on Robotics and Automation*, pages 1548–1553, May 2013.

[28] A. Huletski, D. Kartashov, and K. Krinkin. The artificial landmark design for mobile robots localization and mapping. In *Open Innovations Association (FRUCT), 2015 17TH Conference of*, pages 56–61, April 2015.

[29] Diansheng Chen, Zhaoliang Peng, and Xiao Ling. A low-cost localization system based on artificial landmarks with two degree of freedom platform camera. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 625–630, Dec 2014.

[30] Song Yulong, Lian Baowang, and Li Cha. Indoor integrated navigation for differential wheeled robot using ceiling artificial landmark. In *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, pages 828–831, Dec 2013.

[31] Jingdong Yang, Jinghui Yang, and Zesu Cai. An efficient approach to pose tracking based on odometric error modelling for mobile robots. *Robotica*, 33(6):1231–1249, 007 2015.

[32] B. Bischoff, Duy Nguyen-Tuong, F. Streichert, M. Ewert, and A. Knoll. Fusing vision and odometry for accurate indoor robot localization. In *Control Automation Robotics Vision (ICARCV), 2012 12th International Conference on*, pages 347–352, Dec 2012.

[33] N. Ganganath and H. Leung. Mobile robot localization using odometry and kinect sensor. In *Emerging Signal Processing Applications (ESPA), 2012 IEEE International Conference on*, pages 91–94, Jan 2012.

[34] Kinect. Xbox 360 + Kinect. `http://www.xbox.com/en-US/xbox-360/accessories/kinect`. Acessado em: Janeiro, 2017.

[35] A. Aldoma, Z. C. Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R. B. Rusu, S. Gedikli, and M. Vincze. Tutorial: Point cloud library: Three-dimensional

object recognition and 6 dof pose estimation. *IEEE Robotics Automation Magazine*, 19(3):80–91, Sept 2012.

[36] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly, and A. J. Davison. Dense planar slam. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 157–164, Sept 2014.

[37] L. A. Souto and T. P. Nascimento. Distributed object subtraction 3d mapping. In *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*, pages 7–12, Oct 2015.

[38] Jesus Martínez-Gómez, Vicente Morell, Miguel Cazorla, and Ismael García-Varea. Semantic localization in the {PCL} library. *Robotics and Autonomous Systems*, 75, Part B:641 – 648, 2016.

[39] A. Aldoma, Z. C. Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R. B. Rusu, S. Gedikli, and M. Vincze. Tutorial: Point cloud library: Three-dimensional object recognition and 6 dof pose estimation. *IEEE Robotics Automation Magazine*, 19(3):80–91, Sept 2012.

[40] C. Yin, S. Yang, X. Yi, Z. Wang, Y. Wang, B. Zhang, and Y. Tang. Removing dynamic 3d objects from point clouds of a moving rgb-d camera. In *2015 IEEE International Conference on Information and Automation*, pages 1600–1606, Aug 2015.

[41] Hu Chao, Feng Zhongqing, Ren Yupeng, Chen Yueyue, Lin Haixiang, Wang Kai, Xu Xiaodong, and Bao Jianmeng. An efficient magnetic localization system for indoor planar mobile robot. In *Control Conference (CCC), 2015 34th Chinese*, pages 4899–4904, July 2015.

[42] Jongdae Jung, Seung-Mok Lee, and Hyun Myung. Indoor mobile robot localization and mapping based on ambient magnetic fields and aiding radio sources. *Instrumentation and Measurement, IEEE Transactions on*, 64(7):1922–1934, July 2015.

[43] Bing-Fei Wu and Cheng-Lung Jen. Particle-filter-based radio localization for mobile robots in the environments with low-density wlan aps. *Industrial Electronics, IEEE Transactions on*, 61(12):6860–6870, Dec 2014.

[44] M.L. Rodrigues, L.F.M. Vieira, and M.F.M. Campos. Mobile robot localization in indoor environments using multiple wireless technologies. In *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, pages 79–84, Oct 2012.

[45] J. MacDougall and G.S. Tewolde. Tour guide robot using wireless based localization. In *Electro/Information Technology (EIT), 2013 IEEE International Conference on*, pages 1–6, May 2013.

[46] Seong Jin Kim and Byung Kook Kim. Dynamic ultrasonic hybrid localization system for indoor mobile robots. *Industrial Electronics, IEEE Transactions on*, 60(10):4562–4573, Oct 2013.

[47] Jia Qian, Niu Weiqiang, Wang Mulan, Liu Shuqing, and Ge Jianjing. Design of localization system for indoor mobile robots based on ultrasonic net. In *Control and Decision Conference (CCDC), 2015 27th Chinese*, pages 3248–3252, May 2015.

[48] Wenzheng Chi, Wei Zhang, J. Gu, and Hongliang Ren. A vision-based mobile robot localization method. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 2703–2708, Dec 2013.

[49] SeokJu Lee, Girma S. Tewolde, Jongil Lim, and Jaerock Kwon. Vision based localization for multiple mobile robots using low-cost vision sensor. In *Electro/Information Technology (EIT), 2015 IEEE International Conference on*, pages 280–285, May 2015.

[50] Song Peipei, Li Wenyu, Yang Ningjia, and Duan Feng. An intelligent vision localization system of a service robot nao. In *Control Conference (CCC), 2015 34th Chinese*, pages 5993–5998, July 2015.

[51] Peng Duan, Guohui Tian, and Hao Wu. A multi-sensor-based mobile robot localization framework. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 642–647, Dec 2014.

[52] Maohai Li, Rui Lin, Zhenhua Wang, and Yunbo Hong. Omnidirectional vision based mobile robot topological localization. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1905–1910, Dec 2013.

[53] Zhaopeng Gu, Hong Liu, and Guodong Zhang. Real-time indoor localization of service robots using fisheye camera and laser pointers. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 1410–1414, Dec 2014.

[54] Chang Lin, Bingwei He, and Jianwei Zhang. Study on indoor robot of self-localization based on rgb-d sensor. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, pages 2619–2624, Dec 2014.

[55] Seok-Ju Lee, G. Tewolde, Jongil Lim, and Jaerock Kwon. Qr-code based localization for indoor mobile robot with validation using a 3d optical tracking instrument. In *Advanced Intelligent Mechatronics (AIM), 2015 IEEE International Conference on*, pages 965–970, July 2015.

[56] Huosheng Hu and Dongbing Gu. Landmark-based navigation of industrial mobile robots. *Industrial Robot: An International Journal*, 27(6):458–467, 12 2000.

[57] Yuandong Sun, Ning Ding, Huihuan Qian, and Yangsheng Xu. Real-time monocular visual self-localization approach using natural circular landmarks for indoor navigation. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 495–500, Dec 2012.

[58] P. Alves, H. Costelha, and C. Neves. Localization and navigation of a mobile robot in an office-like environment. In *Autonomous Robot Systems (Robotica), 2013 13th International Conference on*, pages 1–6, April 2013.

[59] M.A. Salem. Multi-stage localization given topological map for autonomous robots. In *Computer Engineering Systems (ICCES), 2012 Seventh International Conference on*, pages 55–60, Nov 2012.

[60] M. Liu, X. Lei, S. Zhang, and B. Mu. Natural landmark extraction in 2d laser data based on local curvature scale for mobile robot navigation. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 525–530, Dec 2010.

[61] M. Rous, H. Lupschen, and K. F. Kraiss. Vision-based indoor scene analysis for natural landmark detection. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4642–4647, April 2005.

[62] X. Xiong and B. J. Choi. Position estimation algorithm based on natural landmark and fish-eyes' lens for indoor mobile robot. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pages 596–600, May 2011.

[63] X. Chai, F. Wen, and K. Yuan. Fast vision-based object segmentation for natural landmark detection on indoor mobile robot. In *2011 IEEE International Conference on Mechatronics and Automation*, pages 2232–2237, Aug 2011.

[64] Y. Zhou, G. Jiang, G. Xu, X. Wu, and L. Krundel. Kinect depth image based door detection for autonomous indoor navigation. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 147–152, Aug 2014.

[65] M. Derry and B. Argall. Automated doorway detection for assistive shared-control wheelchairs. In *2013 IEEE International Conference on Robotics and Automation*, pages 1254–1259, May 2013.

[66] S. Meyer Zu Borgsen, M. Schöpfer, L. Ziegler, and S. Wachsmuth. Automated door detection with a 3d-sensor. In *2014 Canadian Conference on Computer and Robot Vision*, pages 276–282, May 2014.

[67] Y. Kuramochi, H. Takizawa, M. Aoyagi, N. Ezaki, and M. Shinji. Recognition of elevators with the kinect cane system for the visually impaired. In *2014 IEEE/SICE International Symposium on System Integration*, pages 128–131, Dec 2014.

[68] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[69] Y. H. Lee, T. S. Leung, and G. Medioni. Real-time staircase detection from a wearable stereo system. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3770–3773, Nov 2012.

[70] OpenCv. Face Detection using Haar Cascades. `http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html`. Acessado em: Janeiro, 2017.

[71] MathWorks. Ada Boost. `https://www.mathworks.com/discovery/adaboost.html`. Acessado em: Janeiro, 2017.

[72] K. Romić, I. Galić, and T. Galba. Technology assisting the blind - video processing based staircase detection. In *2015 57th International Symposium ELMAR (ELMAR)*, pages 221–224, Sept 2015.

[73] A. Qureshi, R. Hussain, R. Ali, R. Ijaz, N. Rasheed, J. Iqbal, and M. I. Tiwana. Stair case perception using monoscopic vision for a teleoperated vehicle. In *2015 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pages 174–178, Sept 2015.

[74] A. Sinha, P. Papadakis, and M. R. Elara. A staircase detection method for 3d point clouds. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 652–656, Dec 2014.

[75] Hossein Mohimani, Massoud Babaie-Zadeh, and Christian Jutten. A fast approach for overcomplete sparse decomposition based on smoothed L0 norm. *IEEE Transactions on Signal Processing*, 57(1):289–301, 2009.

Apêndice A

Código da Identificação

Código Fonte A.1: Código da Identificação

```
1  // BEGIN ----------------------------------------------------------------------
2          // Make the detection of Natural Landmarks
3
4
5                  //*************** HERE STARTS THE SEGMENTATION AND CLASSIFICATION PART
                       **********************
6
7                  //------ Create PassThrough Filter
8                  pcl::PassThrough<pcl::PointXYZRGBA> filter;
9                  pcl::PassThrough<pcl::PointXYZRGBA> filter2;
10                 pcl::PassThrough<pcl::PointXYZRGBA> filter3;
11                 pcl::PassThrough<pcl::PointXYZRGBA> filter4;
12
13                 //------ Create Object for Voxel Grid
14                 pcl::VoxelGrid<pcl::PointXYZRGBA> sor;
15
16                 //------ Create Object for Normals
17                 pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
18
19                 //------ Create Object for Projection
20                 pcl::ProjectInliers<pcl::PointXYZRGBA> plane_proj;
21                 pcl::ProjectInliers<pcl::PointXYZRGBA> objects_proj;
22
23                 //------ Create Objects for Hulls
24                 pcl::PointCloud<pcl::PointXYZRGBA>::Ptr concaveHull (new pcl::PointCloud<
                       pcl::PointXYZRGBA> ());
25
26         std::vector<int> mapping;
27         pcl::removeNaNFromPointCloud(*cloud,*cloud,mapping);
28
```

```
29              //Filter on X and Y axis
30          filter.setInputCloud(cloud);
31          filter.setFilterFieldName("y");
32          filter.setFilterLimits(-0.6, 0.2);
33          filter.filter(*filteredCloud1);
34          filter2.setInputCloud(filteredCloud1);
35          filter2.setFilterFieldName("x");
36          filter2.setFilterLimits(-0.6, 0.6);
37          filter2.filter(*filteredCloud1);
38
39
40          //EXTRACT N BEST PLANES AND CALCULATE CONCAVE HULL
41          PointCloudT::Ptr hullCloud (new PointCloudT);
42          //bool fewPlanes = false;
43          bool havePlane = true;
44          int planeNumber = 0;
45          std::vector<pcl::ModelCoefficients> planeParameters;
46
47              std::cout <<"\nBegin: "<<std::endl;
48
49              //Variables for Plane Segmentation
50              pcl::ModelCoefficients::Ptr coefficients(new pcl::ModelCoefficients);
51              pcl::PointIndices::Ptr inlierIndices(new pcl::PointIndices);
52          pcl::SACSegmentation<pcl::PointXYZRGBA> segmentation1;
53
54
55          while(havePlane == true)
56          {
57             // Get the plane model, if present.
58
59             segmentation1.setInputCloud(filteredCloud1);
60             segmentation1.setModelType(pcl::SACMODEL_PLANE);
61             segmentation1.setMethodType(pcl::SAC_RANSAC);
62             segmentation1.setDistanceThreshold(0.01);
63             segmentation1.setOptimizeCoefficients(true);
64             segmentation1.segment(*inlierIndices, *coefficients);
65                 //std::cout << "\nAfter Segmentation" << std::endl;
66                 //std::cout << inlierIndices->indices.size() << std::endl;
67
68
69             if(inlierIndices->indices.size() >= 15000)
70             {
71
72                //Get parameters of found and good planes
73                planeParameters.push_back(*coefficients);
74
75                //fewPlanes = false;
```

```
76                  //lastIndices->indices = inlierIndices->indices;
77
78                  //Cloud for storing the plane
79              pcl::PointCloud<pcl::PointXYZRGBA>::Ptr plane(new pcl::PointCloud<pcl::
                    PointXYZRGBA>);
80                  //Project the found plane in a new point cloud
81              plane_proj.setModelType (pcl::SACMODEL_PLANE);
82              plane_proj.setIndices (inlierIndices);
83              plane_proj.setInputCloud (filteredCloud1);
84              plane_proj.setModelCoefficients (coefficients);
85              plane_proj.filter (*plane);
86
87                  //Extract the points of the plane from the original cloud
88                  pcl::ExtractIndices<pcl::PointXYZRGBA> extract1;
89                  extract1.setInputCloud(filteredCloud1);
90                  extract1.setIndices(inlierIndices);
91                  extract1.setNegative(true);
92              extract1.filter(*filteredCloud1);
93
94                  // Object for retrieving the concave hull.
95                  pcl::ConcaveHull<pcl::PointXYZRGBA> hull;
96                  hull.setInputCloud(plane);
97                  // Set alpha, which is the maximum length from a vertex to the center of
                        the voronoi cell
98                  // (the smaller, the greater the resolution of the hull).
99                  hull.setAlpha(0.08);
100             hull.setDimension (2);
101                 hull.reconstruct(*concaveHull);
102
103                 //std::cout << "\nBefore add plane to hull." << std::endl;
104                 *hullCloud += *concaveHull;
105                 //std::cout << "\nAdded a new plane." << std::endl;
106
107                 // Visualize the hull.
108                 //pcl::visualization::CloudViewer viewerPlane("Concave hull");
109                 //viewerPlane.showCloud(concaveHull);
110             planeNumber++;
111             //std::cout << "\nFound " << planeNumber << " planes." <<std::endl;
112             std::cout << inlierIndices->indices.size() << std::endl;
113
114                }//END IF
115                else if ((inlierIndices->indices.size() < 15000) || planeNumber == 5)
116            {
117                   std::cout << "\nCould not find a plane in the scene." << std::endl;
118               havePlane = false;
119               //fewPlanes = false;
120            }
```

```
121
122         }//END WHILE
123
124
125         if(planeNumber >= 1)
126         {
127         float min_y = -0.5;
128         float max_y = 0.1;
129         float min_x = -0.5;
130         float max_x = 0.5;
131
132         filter3.setInputCloud(hullCloud);
133         filter3.setFilterFieldName("y");
134         filter3.setFilterLimits(min_y, max_y);
135         filter3.filter(*hullCloud);
136         filter4.setInputCloud(hullCloud);
137         filter4.setFilterFieldName("x");
138         filter4.setFilterLimits(min_x, max_x);
139         filter4.filter(*hullCloud);
140
141
142         //Key Points to show in resultant cloud
143         pcl::PointCloud<pcl::PointXYZRGBA>::Ptr point (new pcl::PointCloud<pcl::
                PointXYZRGBA>);
144         pcl::PointCloud<pcl::PointXYZRGBA>::Ptr pointred (new pcl::PointCloud<pcl::
                PointXYZRGBA>);
145
146         //Variables for radial search and normal calculation
147         pcl::KdTreeFLANN<PointT> kdtree;
148         kdtree.setInputCloud (cloud);
149         int ka = 15;
150
151         pcl::PointXYZRGBA searchPoint1 = hullCloud->points[0];
152         std::vector<int> pointIdxRadiusSearch1;
153         std::vector<float> pointRadiusSquaredDistance;
154
155         float curvature;
156         Eigen::Vector4f plane_parameters1;
157
158         float s_normal[4][hullCloud->points.size()];
159
160         //COMPARE POINT BY POINT ON THE RUN
161
162         //Variable to hold indices after Sl0
163         std::vector<int> inliers_s10;
164
165         for (size_t iterator = 0; iterator < hullCloud->points.size()-1; iterator++)
```

```
166         {
167             //Compute normal for Point 1
168
169             //Radial search for estimating neighborhood indices
170             kdtree.nearestKSearch(searchPoint1, ka, pointIdxRadiusSearch1,
                    pointRadiusSquaredDistance);
171
172                 //Compute normal of a single point (Point 1)
173                 computePointNormal(*cloud, pointIdxRadiusSearch1, plane_parameters1, curvature)
                        ;
174
175                     //Compute normal for Point 2
176                     pcl::PointXYZRGBA searchPoint2 = hullCloud->points[iterator+1];
177             std::vector<int> pointIdxRadiusSearch2;
178                     //Radial search for estimating neighborhood indices
179             kdtree.nearestKSearch(searchPoint2, ka, pointIdxRadiusSearch2,
                    pointRadiusSquaredDistance);
180
181                     //Compute normal of a single point (Point 2)
182             Eigen::Vector4f plane_parameters2;
183                 computePointNormal(*cloud, pointIdxRadiusSearch2, plane_parameters2, curvature)
                        ;
184
185             double radian = getAngle3D(plane_parameters1 , plane_parameters2);
186
187                 double angle = (radian * 180 / M_PI);
188
189
190                     //THIS WOULD BE THE LOGIC OF SL0 MINIMIZATION APLIED IN NORMALS
191
192             if(angle >= 0 && angle <= 10.0)
193             {
194                //std::cout << angle <<std::endl;
195
196                     point->points.push_back(hullCloud->points[iterator]);
197                     pointred->points.push_back(hullCloud->points[iterator+1]);
198
199                     inliers_sl0.push_back(iterator);
200             }
201
202                         //Change variables
203                         searchPoint1 = hullCloud->points[iterator+1];
204                      //pointIdxRadiusSearch1 = pointIdxRadiusSearch2;
205                         //plane_parameters1 = plane_parameters2;
206
207         }//END FOR
208
```

```
209         //sl0indices->indices = inliers_sl0;
210         copyPointCloud(*hullCloud,inliers_sl0,*filteredCloud2);
211
212
213
214         pcl::visualization::PointCloudColorHandlerCustom<PointT>
                scene_keypoints_color_handler (point, 0, 0, 255);
215         pcl::visualization::PointCloudColorHandlerCustom<PointT>
                scene_keypoints_color_handler1 (point, 255, 0, 0);
216         viewer.removeAllPointClouds();
217         viewer.removeAllShapes();
```

Apêndice B

Código de Reconhecimento

Código Fonte B.1: Código do Reconhecimento

```
1  //================== CLASSIFICATION PART BEGIN ==================
2
3          //Create plane to project points
4          // Create a set of planar coefficients with X=0 Y=1 Z=0
5          pcl::ModelCoefficients::Ptr ppcoefficients (new pcl::ModelCoefficients ());
6          ppcoefficients->values.resize (4);
7          ppcoefficients->values[0] = 0;
8          ppcoefficients->values[1] = 1.0;
9          ppcoefficients->values[2] = 0;
10         ppcoefficients->values[3] = 0;
11
12         // Create the filtering object
13         pcl::ProjectInliers<pcl::PointXYZRGBA> proj;
14         proj.setModelType (pcl::SACMODEL_PLANE);
15         proj.setInputCloud (filteredCloud2);
16         proj.setModelCoefficients (ppcoefficients);
17         proj.filter (*cloud_projected);
18
19         std::string object;
20
21         PointT min_pt, max_pt, left_pt, right_pt;
22         getMinMaxXZ(*cloud_projected, min_pt, max_pt, left_pt, right_pt, viewer, object,
               myfile3, frames);
23
24
25         //Calculate Diagonal
26         double a = min_pt.x - max_pt.x;
27         double b = min_pt.z - max_pt.z;
28         double d = sqrt(pow(min_pt.x-max_pt.x,2)+pow(min_pt.z-max_pt.z,2));
29         //Calculate Width
```

```
30          //double width = abs(max_pt.x) + abs(min_pt.x);wer
31          double width = sqrt(pow(min_pt.x-right_pt.x,2)+pow(min_pt.z-right_pt.z,2));
32
33          //Get all Planes parameters
34          Eigen::Vector4f plane1(0,0,0,0);
35          Eigen::Vector4f plane2(0,0,0,0);
36          Eigen::Vector4f plane3(0,0,0,0);
37          Eigen::Vector4f plane4(0,0,0,0);
38          Eigen::Vector4f plane5(0,0,0,0);
39
40          for(int y=0; y<planeNumber; y++)
41          {
42              if(y==0)
43              {
44                  for(int i=0; i<4; i++)
45                  plane1[i] = (planeParameters[y].values[i]);
46              }
47              else if(y==1)
48              {
49                  for(int i=0; i<4; i++)
50                  plane2[i] = (planeParameters[y].values[i]);
51              }
52              else if(y==2)
53              {
54                  for(int i=0; i<4; i++)
55                  plane3[i] = (planeParameters[y].values[i]);
56              }
57              else if(y==3)
58              {
59                  for(int i=0; i<4; i++)
60                  plane4[i] = (planeParameters[y].values[i]);
61              }
62              else if(y==4)
63              {
64                  for(int i=0; i<4; i++)
65                  plane5[i] = (planeParameters[y].values[i]);
66              }
67          }
68
69          //BEGIN Calculate paralelism of planes
            ------------------------------------------------------------------------------

70          int parallelPlanes = 0;
71          double meanDist = 0.0;
72          vector<double> meanDistance;
73          double meanDist1 = 0.0;
74          double meanDist2 = 0.0;
```

```
75          double meanDist3 = 0.0;
76          double meanDist4 = 0.0;
77
78          //std::cout << "\n" << "Plane number: " << planeNumber <<std::endl;
79
80          double radian = getAngle3D(plane1,plane2);
81          double angle1 = (radian * 180 / M_PI);
82          //std::cout << "\n" << "Angle 1: " <<angle1 <<std::endl;
83          if(angle1 < 15.0)
84          {
85              parallelPlanes++;
86              //std::cout << " Plane1[3] "<< plane1[3] <<std::endl;
87              //std::cout << " Plane2[3] "<< plane2[3] <<std::endl;
88              meanDist1 = abs(plane2[3]-plane1[3]);
89              meanDistance.push_back(abs(plane1[3]));
90              meanDistance.push_back(abs(plane2[3]));
91              //std::cout << "Mean Dist 1: "<< meanDist1 <<std::endl;
92          }
93          //std::cout << "Mean Dist 1: "<< meanDist1 <<std::endl;
94          radian = getAngle3D(plane2,plane3);
95          double angle2 = (radian * 180 / M_PI);
96          //std::cout << "Angle 2: " << angle2 <<std::endl;
97          if(angle2 < 15.0)
98          {
99              parallelPlanes++;
100             //std::cout << " Plane2[3] "<< plane2[3] <<std::endl;
101             //std::cout << " Plane3[3] "<< plane3[3] <<std::endl;
102             meanDist2 = abs(plane3[3]-plane2[3]);
103             meanDistance.push_back(abs(plane2[3]));
104             meanDistance.push_back(abs(plane3[3]));
105             //std::cout << "Mean Dist 2: "<< meanDist2 <<std::endl;
106         }
107         //std::cout << "Mean Dist 2: "<< meanDist2 <<std::endl;
108         radian = getAngle3D(plane3,plane4);
109         double angle3 = (radian * 180 / M_PI);
110         //std::cout << "Angle 3: " << angle3 <<std::endl;
111         if(angle3 < 15.0)
112         {
113             parallelPlanes++;
114             meanDist3 = abs(plane4[3]-plane3[3]);
115        meanDistance.push_back(abs(plane3[3]));
116             meanDistance.push_back(abs(plane4[3]));
117             //std::cout << "Mean Dist 3: "<< meanDist3 <<std::endl;
118         }
119         //std::cout << "Mean Dist 3: "<< meanDist3 <<std::endl;
120         radian = getAngle3D(plane4,plane5);
121         double angle4 = (radian * 180 / M_PI);
```

```
122         //std::cout << "Angle 4: " << angle4 <<std::endl;
123         if(angle4 < 15.0)
124         {
125             parallelPlanes++;
126             meanDist4 = abs(plane5[3]-plane4[3]);
127             meanDistance.push_back(abs(plane4[3]));
128             meanDistance.push_back(abs(plane5[3]));
129             //std::cout << "Mean Dist 4: "<< meanDist4 <<std::endl;
130         }
131         //std::cout << "Mean Dist 4: "<< meanDist4 << "\n" <<std::endl;
132
133         //END Calculate paralelism of planes
                ---------------------------------------------------------------------------
134
135         //Sort Plane distances
136         sort(meanDistance.begin(), meanDistance.end());
137
138         if(planeNumber >= 2 && !meanDistance.empty())
139         {
140             meanDist = ((meanDistance.back()) - (meanDistance.front()))/parallelPlanes;
141         }
142         else
143         {
144          meanDist = 0.0;
145         }
146
147         // Print Classificator Attributes
148
149         double pmeanDist = roundf(abs(meanDist) * 10) / 10;
150         double pd = d;
151         double pwidth =          width;
152
153         //std::cout << "\nPlanes number: "<< planeNumber <<std::endl;
154         //std::cout << "Number Parallel planes: "<< parallelPlanes <<std::endl;
155         //std::cout << "Parallel planes Distance Mean: "<< pmeanDist <<std::endl;
156         //std::cout << "D: "<< pd <<std::endl;
157         //std::cout << "Width: "<< pwidth <<std::endl;
158     //std::cout << "Object: "<< object <<std::endl;
159
160
161     // ============================== Statistical Analysis Begin
            ======================================
162
163
164         myfile2 << planeNumber <<" "<< parallelPlanes <<" "<< pmeanDist <<" "<< pd <<" "<<
                pwidth <<"\n";
165
```

```
166
167     if(object.compare("stair") == 0){
168        escada++;
169        mxe += abs(max_pt.x) - abs(min_pt.x);
170        mze += (max_pt.z - min_pt.z)/2 + min_pt.z;
171        //viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z, 0.0,
               1.0, 0.0, "Porta");
172        //viewer.addText("Staircase", 400, 200, 20, 0, 1, 0, "Porta1");
173     }
174     else if (object.compare("door") == 0)
175     {
176        porta++;
177        mxp += abs(max_pt.x) - abs(min_pt.x);
178        mzp += (max_pt.z - min_pt.z)/2 + min_pt.z;
179        //viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z, 0.0,
               1.0, 0.0, "Escada1");
180        //viewer.addText("Door", 400, 200, 20, 0, 1, 0, "Escada");
181     }
182
183         if(frames == 5000)
184     {
185
186        myfile3 << escada << " Deteccoees de Escada - " << "Media x z "<< mxe/escada <<" "<<
               mze/escada <<" "<< "\n";
187        myfile3 << porta << " Deteccoees de Porta - " << "Media x z "<< mxp/porta <<" "<<
               mzp/porta <<" "<< "\n";
188
189        myfile2.close();
190
191        myfile3.close();
192     }
193
194     if(frames == 1001)
195     {
196        myfile << escada << " Deteccoees de Escada - " << "Media x y z "<< mxe/frames <<" "
               << mye/frames <<" "<< mze/frames <<" "<< "\n";
197        myfile << porta << " Deteccoees de Porta - " << "Media x y z "<< mxp/frames <<" "<<
               myp/frames <<" "<< mzp/frames <<" "<< "\n";
198        myfile << coluna << " Deteccoees de Coluna - " << "Media x y z "<< mxc/frames <<" "
               << myc/frames <<" "<< mzc/frames <<" "<< "\n\n";
199        myfile.close();
200
201     }
202
203
204     // ============================== Statistical Analysis End
           ======================================
```

```
205
206
207         //vviewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z, 0.0,
                1.0, 0.0, "Object");de
208
209
210         //Here the algorithm performs the raw classification from the data gathered
211
212
213         if(planeNumber == 2)
214         {
215            if(angle1 < 15.0 && abs(plane1[3]-plane2[3]) > 15)
216            {
217               viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                       0.0, 1.0, 0.0, "Escada");
218               viewer.addText("Staircase", 400, 100, 20, 0, 1, 0, "Escada");
219               if(frames <= 1000)
220               {
221                  mxe += max_pt.x - min_pt.x;
222                  //mye += (max_pt.y - min_pt.y)/2 + min_pt.y;
223                  mze += (max_pt.z - min_pt.z)/2 + min_pt.z;
224                  escada ++;
225               }
226            }
227            else if(d > 0.5 && d < 1.0)
228            {
229               viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                       0.0, 1.0, 0.0, "Door");
230               viewer.addText("Opened Door", 400, 100, 20, 0, 1, 0, "Porta");
231               if(frames <= 1000)
232               {
233                  mxp += max_pt.x - min_pt.x;
234                  //myp += (max_pt.y - min_pt.y)/2 + min_pt.y;
235                  mzp += (max_pt.z - min_pt.z)/2 + min_pt.z;
236                  porta ++;
237               }
238            }
239            else if(d < 0.5)
240            {
241               viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                       0.0, 1.0, 0.0, "Column");
242               viewer.addText("Column", 400, 100, 20, 0, 1, 0, "Coluna");
243               if(frames <= 1000)
244               {
245                  mxc += max_pt.x - min_pt.x;
246                  //myc += (max_pt.y - min_pt.y)/2 + min_pt.y;
247                  mzc += (max_pt.z - min_pt.z)/2 + min_pt.z;
```

```
248                 coluna ++;
249             }
250         }
251     }
252     else if(planeNumber >= 3)
253     {
254         if(angle1 < 15.0 && angle2 < 15.0 && abs(plane1[3]-plane2[3]) > 15)
255         {
256             viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                    0.0, 1.0, 0.0, "Escada");
257             viewer.addText("Staircase 1", 400, 100, 20, 0, 1, 0, "Escada1");
258             if(frames <= 1000)
259             {
260                 mxe += max_pt.x - min_pt.x;
261                 //myp += (max_pt.y - min_pt.y)/2 + min_pt.y;
262                 mze += (max_pt.z - min_pt.z)/2 + min_pt.z;
263                 escada ++;
264             }
265         }
266         else if(planeNumber == 3 && d < 1.0 && d > 0.5)
267         {
268             viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                    0.0, 1.0, 0.0, "Door");
269             viewer.addText("Closed Door", 400, 100, 20, 0, 1, 0, "Porta");
270             if(frames <= 1000)
271             {
272                 mxp += max_pt.x - min_pt.x;
273                 //myp += (max_pt.y - min_pt.y)/2 + min_pt.y;
274                 mzp += (max_pt.z - min_pt.z)/2 + min_pt.z;
275                 porta ++;
276             }
277         }
278         else if(planeNumber == 3 && d > 1.0 && d < 1.5)
279         {
280             viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                    0.0, 1.0, 0.0, "Door Semi open");
281             viewer.addText("Door Semi Opened", 400, 100, 20, 0, 1, 0, "Porta");
282             if(frames <= 1000)
283             {
284                 mxp += max_pt.x - min_pt.x;
285                 //mye += (max_pt.y - min_pt.y)/2 + min_pt.y;
286                 mzp += (max_pt.z - min_pt.z)/2 + min_pt.z;
287                 porta ++;
288             }
289         }
290         else if(d > 1.2)
291         {
```

```
292                 viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z,
                        0.0, 1.0, 0.0, "Staircase");
293                 viewer.addText("Staircase", 400, 100, 20, 0, 1, 0, "Escada");
294                 if(frames <= 1000)
295                 {
296                     mxe += max_pt.x - min_pt.x;
297                     //mye += (max_pt.y - min_pt.y)/2 + min_pt.y;
298                     mze += (max_pt.z - min_pt.z)/2 + min_pt.z;
299                     escada ++;
300                 }
301             }
302         }
303         else if(d > 1.2)
304         {
305             viewer.addCube(min_pt.x, max_pt.x, min_pt.y, max_pt.y, min_pt.z, max_pt.z, 0.0,
                      1.0, 0.0, "Staircase");
306             viewer.addText("Staircase", 400, 100, 20, 0, 1, 0, "Escada");
307             if(frames <= 1000)
308             {
309                 mxe += max_pt.x - min_pt.x;
310                 //mye += (max_pt.y - min_pt.y)/2 + min_pt.y;
311                 mze += (max_pt.z - min_pt.z)/2 + min_pt.z;
312                 escada ++;
313             }
314         }
315
316
317
318         //==========================================  CLASSIFICATION  PART  END
              ==========================================
```

Apêndice C

Função Modificada do Algoritmo 2

Código Fonte C.1: Código da Função Modificada

```
1  template <typename PointT> inline void
2  getMinMaxXZ (const pcl::PointCloud<PointT> &cloud, PointT &min_pt, PointT &max_pt, PointT &
       left_pt, PointT &right_pt,
3                pcl::visualization::PCLVisualizer &viewer, std::string &object, ofstream &
                     myfile3, int frames)
4  {
5
6       Eigen::Array4f min_p, max_p;
7       Eigen::Array4f left_p, right_p;
8
9       Eigen::Array4f now_p, last_p, next_p, old_p;
10      Eigen::Array2f mean_p;
11
12      min_p.setConstant (FLT_MAX);
13      max_p.setConstant (-FLT_MAX);
14      left_p.setConstant (FLT_MAX);
15      right_p.setConstant (-FLT_MAX);
16
17      int door = 0;
18      int stair = 0;
19
20      int divider = 1;
21      double increaser = 0.0;
22
23      std::vector<Eigen::Array2f> points;
24
25      //Fill points X coordinate
26      left_p[0] = 0;
27      right_p[0] = 0;
28
```

```
29      //Fill points X-Z coordinate of SUM - will be the "mean" of the cluster points
30      mean_p[0] = 0.0;
31      mean_p[1] = 0.0;
32
33      double old_var_x = 0.0;
34      double old_var_z = 0.0;
35
36      // If the data is dense, we don't need to check for NaN
37      if (cloud.is_dense)
38      {
39        for (size_t i = 1; i < cloud.points.size (); ++i)
40        {
41          pcl::Array4fMapConst pt = cloud.points[i].getArray4fMap ();
42          min_p = min_p.min (pt);
43          max_p = max_p.max (pt);
44
45          if(pt[0] < left_p[0]) left_p = pt;
46          if(pt[0] > right_p[0]) right_p = pt;
47
48          //Compare current point with next
49
50          if (i < cloud.points.size())
51          {
52             now_p = pt;
53             next_p = cloud.points[i+1].getArray4fMap ();
54
55             //Store variation of 21 points behind
56             if(i > 20)
57             {
58                old_p = cloud.points[i-20].getArray4fMap ();
59                old_var_x = abs(now_p[0] - old_p[0]);
60                old_var_z = abs(now_p[2] - old_p[2]);
61             }
62
63             double var_x = abs(next_p[0] - now_p[0]);
64             double var_z = abs(next_p[2] - now_p[2]);
65
66
67             mean_p[0] = mean_p[0] + now_p[0];
68             mean_p[1] = mean_p[1] + now_p[2];
69
70             //If its on the final point of the cloud, calculate the final median
71             if (i == cloud.points.size()-1)
72             {
73                mean_p[0] = mean_p[0] / divider;
74                mean_p[1] = mean_p[1] / divider;
75
```

```
76                 myfile3 << mean_p << " - final " <<  frames << "\n";
77
78                 points.push_back(mean_p);
79                 mean_p[0] = 0.0;
80                 mean_p[1] = 0.0;
81                 divider = 0;
82             }
83
84             else if(var_x > 0.0 || var_z > 0.0) //Variation must be larger than 0 cm
85             {
86
87               //Calculates mean of past points and compare with actual point
88
89               double mean_x = mean_p[0] / divider;
90               double mean_z = mean_p[1] / divider;
91
92               //double dif_x = abs(mean_x - now_p[0]);
93               //double dif_z = abs(mean_z - now_p[2]);
94
95               double nextdif_x = abs(mean_x - next_p[0]);
96               double nextdif_z = abs(mean_z - next_p[2]);
97
98               //std::count << "old var x: "<< old_var_x <<std::endl;
99
100              //if ((var_z <= 0.2 && var_x > 0.45) || (var_x <= 0.9 && var_z > 0.05 &&
                     var_z < 0.2)) //Door edge (On JUMP)
101              if ((old_var_x < 0.01 && old_var_z < 0.01) && (nextdif_x > 0.1 || nextdif_z >
                      0.1)) //Door edge (On JUMP)
102              {
103                 mean_p[0] = mean_p[0] / divider;
104                 mean_p[1] = mean_p[1] / divider;
105
106
107                 points.push_back(mean_p);
108                 myfile3 << mean_p << " - door " << frames << "\n";
109                 mean_p[0] = 0.0;
110                 mean_p[1] = 0.0;
111                 divider = 0;
112                 door ++;
113              }
114              //else if ((var_z > 0.25 && var_z < 0.35)) //Stair edge (On JUMP)
115              else if ((var_z > 0.25 && var_z < 0.38)) //Stair edge (On JUMP)
116              {
117                 mean_p[0] = mean_p[0] / divider;
118                 mean_p[1] = mean_p[1] / divider;
119
120
```

```
121                 points.push_back(mean_p);
122                 myfile3 << mean_p << " - stair " <<  frames << "\n";
123                 mean_p[0] = 0.0;
124                 mean_p[1] = 0.0;
125                 divider = 0;
126                 stair ++;
127               }
128             }
129
130         }//END IF (i < cloud.points.size)
131
132         last_p = pt;
133
134         divider++;
135
136       }//END FOR
137
138       myfile3 << "\n";
139
140       std::cout << "Door n: "<< door <<std::endl;
141       std::cout << "Stair n: "<< stair <<std::endl;
142
143       if(stair == 0 && door == 0)
144       {
145          object = "miss";
146       }
147
148       else if(stair > 0 || stair > door)
149       {
150          //Found a stair - Return
151          object = "stair";
152          stair = 0;
153       }
154       else if (door > stair)
155       {
156          //Found a Door - Return
157          object = "door";
158          door = 0;
159       }
160
161
162     }//END IF data is dense
163
164     // NaN or Inf values could exist => check for them
165     else
166     {
167       for (size_t i = 0; i < cloud.points.size (); ++i)
```

```
168         {
169           // Check if the point is invalid
170           if (!pcl_isfinite (cloud.points[i].x) ||
171               !pcl_isfinite (cloud.points[i].y) ||
172               !pcl_isfinite (cloud.points[i].z))
173             continue;
174           pcl::Array4fMapConst pt = cloud.points[i].getArray4fMap ();
175           min_p = min_p.min (pt);
176           max_p = max_p.max (pt);
177 
178           if(pt[0] < left_p[0]) left_p = pt;
179           if(pt[0] > right_p[0]) right_p = pt;
180 
181           //points.push_back(pt);
182         }
183       }
184       min_pt.x = min_p[0]; min_pt.y = min_p[1]; min_pt.z = min_p[2];
185       max_pt.x = max_p[0]; max_pt.y = max_p[1]; max_pt.z = max_p[2];
186 
187       left_pt.x = left_p[0];    left_pt.y = left_p[1];    left_pt.z = left_p[2];
188       right_pt.x = right_p[0]; right_pt.y = right_p[1]; right_pt.z = right_p[2];
189 
190       //std::sort (points.begin(), points.end());
191 
192 
193 }
```