Universidade Federal de Juiz de Fora

Faculdade de Engenharia

Programa de Pós Graduação em Engenharia Elétrica

Fabrício de Oliveira Coelho

Missões Autônomas em Robôs Móveis com Tração Diferencial: Planejamento de Caminhos, Localização e Mapeamento

Juiz de Fora

2018

Fabrício de Oliveira Coelho

Missões Autônomas em Robôs Móveis com Tração Diferencial: Planejamento de Caminhos, Localização e Mapeamento

Dissertação apresentada ao Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora, na área de concentração em Energia , como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: André Luís Marques Marcato

Juiz de Fora

2018

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Coelho, Fabrício de Oliveira.
Missões Autônomas em Robôs Móveis com Tração Diferencial: Planejamento de Caminhos, Localização e Mapeamento / Fabrício de Oliveira Coelho. – 2018.
159 f. : il.

Orientador: André Luís Marques Marcato
Dissertação (mestrado acadêmico) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Programa de Pós Graduação em Engenharia Elétrica, 2018.

1. Robótica Móvel. 2. Missões Autônomas. 3. Inteligência Artificial. I. Marcato, André Luís Marques, orient. II. Título.

Fabrício de Oliveira Coelho

Missões Autônomas em Robôs Móveis com Tração Diferencial: Planejamento de Caminhos, Localização e Mapeamento

Dissertação apresentada ao Programa de Pós Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora, na área de concentração em Energia , como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. André Luís Marques Marcato - Orientador
Universidade Federal de Juiz de Fora

Professor Dr. Bruno Henrique Groenner Barbosa
Universidade Federal de Lavras

Professor Dr. Leonardo de Mello Honório
Universidade Federal de Juiz de Fora

AGRADECIMENTOS

A Deus, por se mostrar presente e me dar forças em todos os momentos da minha vida, principalmente nestes anos de Pós-Graduação. Aos meus fiéis intercessores, Nossa Senhor Aparecida, Nossa Senhora de Fátima, Santa Edwiges e Santo Antônio.

Aos meus pais, Marco Antônio e Leandra, por todo suporte e auxilio para que eu pudesse completar meus estudos e minhas pesquisas. A minha irmã. Aos meus avós que sempre me deram ânimo. Enfim, a todos os tios, tias e primos.

Agradeço à minha namorada, Maria Emília, por todas as vezes que me deu apoio e dicas para que esse trabalho pudesse ser terminado

A todos os professores e demais funcionários que dedicam-se ao setor de educação que passaram por minha vida contribuindo de alguma forma para a composição de tudo que aprendi.

Ao meu orientador professor André Marcato, pela confiança depositada, pela orientação e por todas as oportunidades que me deu. Agradeço também à banca dessa dissertação, professores Leonardo Honório e Bruno Barbosa pela dedicação e atenção em avaliar esse trabalho.

Agradeço às agências que fomentaram meus estudos e pesquisas na UFJF, a CAPES, CNPQ e FAPEMIG.

Aos amigos João Pedro, Guilherme Marins, Milena Faria por toda parceria nas pequisas em robótica ao longo desses anos.

Por fim, agradeço a todos os companheiros de Robótica, em especial aos membros dos Laboratórios GRIn e Litel pela experiência que pude adquirir com vocês.

‘A ciência humana de maneira nenhuma nega a existência de Deus. Quando considero quantas e quão maravilhosas coisas o homem compreende, pesquisa e consegue realizar, então reconheço claramente que o espírito humano é obra de Deus, e a mais notável.”

(Galileu Galilei)

RESUMO

Esse trabalho apresenta uma metodologia para a concepção de missões autônomas utilizando robôs móveis com tração diferencial em ambientes internos. As missões consistem em deslocar o robô até uma posição objetivo partindo de uma pose inicial. Para que as missões ocorram com sucesso, são implementados algoritmos de localização e planejamento de caminhos. Para localização, foi utilizado Filtro de Kalman Estendido (do inglês, *Extended Kalman Filter* EKF) para fundir a odometria com visão computacional. A visão é responsável por encontrar marcadores artificiais conhecidos como *Ar codes* que são alocados no ambiente. O planejamento é caracterizado por uma forma híbrida que corresponde a união de um método deliberativo e reativo. No Planejamento deliberativo, foi proposto e utilizado o método Direct-DRRT* cuja base é oriunda do RRT (*Rapidly Exploring Random Tree*). Além do RRT, esse planejador também apresenta características de dois outros métodos já presentes na literatura: RRT* e DRRT. O planejador deliberativo enviará para o reativo um conjunto de sub-objetivos que conecta a posição em que o robô se encontra até o objetivo final. O planejamento reativo é aplicado durante a missão e é o responsável pelo desvio de obstáculos dinâmicos que não foram mapeados. No Reativo, é utilizado o método dos Campos Potenciais Artificiais (CPA), que também se comporta como o controlador do robô durante a navegação. Para encontrar os obstáculos, utilizou-se o sensor de profundidade *Asus Xtion*, pois, a partir das imagens geradas por esse sensor, é possível encontrar as distâncias que os bloqueios se encontram. As informações desse sensor também será de grande valia na atualização do mapa. O sistema é integrado através da *framework* ROS (*Robot Operating System*). Todos os algoritmos foram implementados por meio da linguagem de programação *Python*. Os resultados do trabalho foram apresentados por meio do simulador *Gazebo* e testes práticos a partir da plataforma P3DX. Foram analisados o comportamento do robô em alguns problemas que podem ocorrer durante a navegação, como o sequestro e aparecimento mínimos locais. Ao final desse trabalho, apresentou-se a melhoria nos resultados do planejador de caminhos Direct-DRRT*, onde foi possível constatar a queda no tempo para obter um caminho, a quantidade de iterações, de nós e do comprimento do caminho em comparação aos outros métodos. No que tange à localização, essa dissertação obteve significativas melhoras comparado com o método que utiliza somente a odometria. Além desse resultado, esse trabalho também obteve sucesso em apresentar uma solução para a implementação de missões autônomas.

Palavras-chave: Robótica Móvel. Missões Autônomas. Planejamento de Caminhos. Localização. Mapeamento.

ABSTRACT

This work presents a design methodology for autonomous missions using mobile robots with differential traction in indoor environments. The missions consist of moving the robot to a goal position starting from an initial pose. For missions success, it is necessary to implement localization and path planning algorithms. For localization, Extended Kalman Filters (EKF) used to fuse odometry with computational vision. The view is responsible for finding artificial markers known as *Ar codes* that are presented in the environment. The planning is characterized by a hybrid form that corresponds to the union of a deliberative and reactive methods. In the Deliberative Planning, the Direct-DRRT * method, whose base is derived from the Rapidly Exploring Random Tree (RRT), is proposed and used. In addition to the RRT, this planner also presents characteristics of two other methods already presented in the literature, i.e. RRT * and DRRT. The deliberative planner sends to the reactive a set of sub-objectives that connects the initial position to the final goal. Reactive planning is applied during the mission and it is responsible for the dynamic obstacles avoidance that have not been mapped. In the reactive, the Artificial Potential Fields (APF) method is used, which also behaves as the robot controller during navigation. To find the obstacles, we use the sensor *Asus Xtion*, due to the possibility to find the distances that the locks are in the images generated by this sensor. All the algorithms were implemented through the programming language Python. The results of the work were presented through the simulator Gazebo and practical tests using the P3DX platform. We analyzed the robot behavior in some problems that may occur during navigation, such as kidnapped and local minimum appearance. At the end, the improvement in the results of the Direct-DRRT * path planner is also presented. It is possible to verify the decrease in the time to obtain a path, the number of iterations of nodes and the path length in comparison to other methods. Regarding the location, this dissertation has obtained significant improvements when compared to the methods that use only odometry. Besides, the work was also successful in presenting a solution for the autonomous missions implementation.

Key-words: Autonomous Mission. Localization. Mapping. Mobile Robots. Path Planning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CPA	Campos Potenciais Artificiais
FKE	Filtro de Kalman Estendido
EKF	*Extended Kalman Filter*
FK	Filtro de Kalman
RRT	*Rapidly Exploring Random Tree*
OSRF	*Open Source Robotics Foundation*
PRM	*Probabilistic Road Maps*
ROS	*Robot Operating System*
DARPA	*Defense Advanced Research Projects Agency*
PPEE	Programa de Pós-Graduação em Engenharia Elétrica
UFJF	Universidade Federal de Juiz de Fora
AR Codes	*Augmented Reality Codes*
QR Codes	*Quick Response Codes*
RGB	Acrônimo para as cores em inglês *Red*, *Green* e *Blue*
STAIR	*Stanford Artificial Intelligence Robots*
PR	*Personal Robots*

SUMÁRIO

1 INTRODUÇÃO

A versatilidade da robótica em atuar em distintos campos aponta o foco de pesquisas e inovações contemporâneas para esta área. Uma definição simples e geral para esses dispositivos é definida por [1] como sistemas programáveis que desempenham diversas funções e tarefas. As contribuições dadas pela informática e sistemas de informação, em particular, beneficiam o ser humano com processos automatizados ou semi-automatizados em suas tarefas. Entretanto, algumas atividades exigem uma interação física com o ambiente, como, por exemplo, a movimentação por uma região, o deslocamento de objetos, tarefas que demandam tempo, perícia e repetitividade para a sua execução. As mesmas, estão além dos avanços das áreas supracitadas, porém, a informática avançada, atrelada aos dispositivos robóticos para a automatização, tornam viáveis essas tarefas [2].

A robótica é uma área multidisciplinar, integrando técnicas oriundas das engenharias mecânica, elétrica, eletrônica e a computacional. Dentre os sistemas robóticos pode-se citar os dispositivos: manipuladores robóticos, veículos aéreos não tripulados (VANT) e a robótica móvel.

Os manipuladores robóticos são encontrados comumente nas linhas de produções industriais, podendo ser definidos como um dispositivo mecânico destinado a posicionar e orientar no espaço o seu efetuador, por exemplo, uma garra ou ferramenta [3]. O uso destes robôs no meio industrial impactam em fatores técnicos, econômicos e sociológicos. O primeiro robô industrial que realizava movimentos autônomos foi utilizado pela *General Motors* em 1961 e projetado por Joseph Engelberger e George Devol [4] sendo nomeado de Unimate (Figura 1 (b)) cuja ideia nasceu da patente [5] escrita por Devol anos antes.

Os VANTs foram concebidos a partir dos paradigmas dos veículos aéreos tripulados. Segundo [6], eles foram primeiramente utilizados no setor militar, porém, atualmente áreas como monitoramento, segurança, agricultura, proteção, inspeções de linhas elétricas e filmagens em geral estão acessíveis à camada civil. No Brasil, o ministério da defesa percebeu a relevância destas aeronaves no aspecto militar e humanitário e passou a produzir alguns modelos de VANTs, como, por exemplo, o Horus FT-100 (Figura 1 (c)) utilizado no monitoramento dos jogos Olímpicos de 2016 [7].

Por fim, têm-se a robótica móvel que visa a locomoção por ambientes internos e externos. Possuem aplicações em operações militares, explorações espaciais, locomoção no interior de minas, agricultura entre outros [8]. Ainda segundo [8] esta classe de robôs pode operar de maneira tele-operada, necessitando de um controlador humano e de modo semi ou completamente autônomos, devendo ser implementados algoritmos que simulem a inteligência humana nos sistemas robóticos [9]. Um famoso exemplo é o carro autônomo Stanley (Figura 1 (a)), que venceu o *Grand Challenge* em 2005 ao cruzar 210 Km pelo deserto de Monjave sem quaisquer intervenções humanas. Não obstante, modelos híbridos

denominados manipuladores móveis também são objetos de estudos. Em [10] é apresentado o modelo Kuka YouBot (Figura 1 (d)), uma plataforma voltada para pesquisa e educação. Outros robôs com essas mesmas características são abordados em [11, 12, 13].

Figura 1 – Exemplos de Robôs: (a) Robô Móvel Stanley (b) Manipulador Robótico Unimate (c) VANT ARP Caçador (d) Manipulador Móvel Kuka YouBot

(a)

(b)

(c)

(d)

Um dos desafios da robótica consiste na concepção de dispositivos que cade vez mais não necessitem da intervenção humana [14] e, também, possam interagir com segurança com os seres humanos. A definição de navegação autônoma, fundamenta-se na realização de tarefas com segurança e inteligência sem essas interferência [15]. As missões autônomas que utilizam robôs móveis podem ser aplicadas em ambientes denominados internos (*indoors*) ou externos (*outdoors*). Nos ambientes internos citam-se a capacidade dos robôs em executar tarefas domésticas, comerciais e até no meio industrial. Para que ações autônomas sejam possíveis é necessário que um processador esteja programado para a tomada de decisões pelo robô [16], geralmente esses processadores são limitados, neste sentindo, algoritmos com custos computacionais elevados são preteridos. A seguir serão descritas algumas tarefas e características que alguns robôs móveis com capacidade autônomas realizam em ambientes internos.

Em [17] apresentou-se um trabalho para robôs autônomos visando segurança. O dispositivo robótico teria o serviço de vigiar o ambiente conforme a profissão de um

vigilante. O trabalho apresentou-se conceitos para a detecção de pessoas intrusas, ou seja, indivíduos que não constem em um banco de dados autorizadas a estarem naquele ambiente. Seguindo esta mesma linha, pesquisadores da universidade de Birmingham desenvolveram um vigia noturno autônomo [18], cujo sistema tem capacidade de identificar modificações no ambiente entre vistorias periódicas realizadas no escritório da empresa de segurança *GS4*. Além desta tarefa, o "Bob", como é conhecido, também analisa objetos deixados em cima das mesas pelos funcionários e se portas destinadas à saídas de incêndio estão abertas. Por se tratar de um ambiente interno e conhecido, o robô utiliza a planta da empresa para planejar suas missões de locomoção.

Uma outra aplicação é o uso dos robôs móveis em ambientes domésticos que auxiliam em tarefas diárias como limpar, lavar e manuseio de diversos objetos. Em [19] apresenta-se o robô "Jhonny", desenvolvido na Alemanha capacitado para desenvolver tarefas de maneira autônoma. O robô interpreta algumas cenas que acontecem ao seu redor, manipula objetos, e realiza navegação e missões específicas como seguir determinado indivíduo no ambiente. Outro robô com enfoque doméstico é o também Alemão "Caesar"[20], o mesmo utiliza diversos algoritmos que possibilitam o reconhecimento facial e as peculiaridades apresentadas pelo robô "Jhonny". Estes grandes projetos possuem gigantescos investimentos que permitem equipar os dispositivos robóticos com sensores e equipamentos de ponta como citados em [18]. Foi dado a esses robôs a capacidade de adaptação e aprendizado no decorrer do tempo, essa característica consiste no fato de que índices de insucesso em diversas tarefas foram diminuindo ao longo do tempo.

Os sistemas robóticos que atuam em ambientes internos apresentados até aqui, pertencem aos seus respectivos laboratórios de pesquisas e ainda não são comercializados. Em contrapartida, robôs desenvolvidos para tarefas domésticas mais simples, voltados para limpeza de ambientes *indoors* de maneira autônoma, como a linha Roomba, da empresa iRobot® podem ser adquiridos comercialmente.

Atualmente o meio industrial também conta com robôs móveis atuando no *chão de fábrica*. De acordo a entrevista dada pela proprietária da empresa *Asti* à *Bloomberg* [21], ainda hoje a maioria da produção e pesquisa é voltada para carros autônomos que se locomovem em ambientes externos sendo que veículos *indoors* recebem menos atenção. Neste cenário, os robôs produzidos pela empresa tentam ganhar o chão de fábrica investindo em dispositivos que se desloquem por esses ambientes, como empilhadeiras que conseguem carregar desde grandes quantidades de alimentos, até peças de aviões de 30 T. Esse tipo de robô contribui para o aumento da produção no setor industrial, já que as receitas da empresa quintuplicaram de 2012 a 2016.

As tarefas desempenhadas pelos robôs envolvem um alto grau de complexidade para sua concepção. Para interagir com o ambiente os robôs devem extrair as informações pertinentes para que suas missões possam ser executadas com sucesso. Os dispositivos

responsáveis pela percepção são denominados sensores [22]. Em [23] divide-se os dispositivos de acordo com o seus respectivos propósitos: posição e orientação, obstáculos, contato, deslocamento, velocidade e comunicação. Os responsáveis por realizar as ações do robô pelo ambiente são os atuadores que também de acordo com [23] é possível dividi-los conforme suas características: rodas, esteirais, juntas, articulações, propulsão, hélices e turbinas. Os custos que envolvem um robô estão atrelados principalmente aos sensores e atuadores empregados em seu sistema. Nesse sentido, quando se projeta um robô é necessário a análise dos custos dos elementos que irão formar o sistema robótico, como, por exemplo, os dispositivos de percepção voltados para distância como sonares e câmeras possuem um custo bem inferior se comparados a *lasers scans*.

Missões de deslocamento em um ambiente tornam-se complexas à medida que maiores graus de autonomia são exigidos. O planejamento da trajetória que o robô deverá realizar para atingir determinada posição objetivo é um paradigma fundamental na robótica autônoma. No que se refere ao planejamento, 3 abordagens são definidas: deliberativa ou *offline*, reativa ou *online* e híbrida [24].

O modo deliberativo necessita do mapa do ambiente *a priori*, neste sentido o planejador irá buscar um conjunto de ações que levem o robô da posição inicial até o objetivo, ressalta-se que os planejadores *offline* possibilitam a otimização das trajetórias concebidas [25]. Entretanto, alguns objetos ou pessoas podem surgir no decorrer da missão, podendo ocasionar colisões, a esses indivíduos é dado o nome de obstáculos ou bloqueios dinâmicos. Nesse sentindo existe a abordagem *online* que é caracterizada por estímulos nos atuadores do sistema robótico em tempo real de acordo com as informações extraídas do ambiente pelos sensores [26]. A abordagem híbrida é identificada a partir da fusão dos dois tipos de planejadores concedendo maior robustez ao sistema.

Um outro problema que os robôs enfrentam é obter uma localização satisfatória de sua posição, pois, segundo [27] a mesma é de vital importância para qualquer missão que o robô venha executar, podendo ser considerado um problema fundamental na robótica. Durante as missões os robôs se deparam com diversos problemas como imprecisão e ruídos sensoriais, desconhecimento da posição inicial e o problema do sequestro dos robôs [28]. Neste sentido, o robô deverá obter estimativas da posição ao longo do caminho gerado pelos planejadores supracitados.

A sessão 1.1 realiza uma abordagem técnica dos trabalhos cujo foco principal envolve os aspectos de missões de deslocamento para robôs móveis autônomos e as respectivas propostas para solucionar os problemas de planejamento de caminhos e localização dos sistemas robóticos recentes. Primeiramente serão apresentados trabalhos que possuem como tema principal as missões autônomas. Entretanto, devido à grande quantidade de algoritmos necessários para realizar essas missões, serão abordados trabalhos cujos temas serão apenas partes de uma missão autônoma. A busca concentrou-se em robôs que

realizassem as missões e tarefas em ambientes internos.

1.1 REVISÃO BIBLIOGRÁFICA

No artigo produzido por Zvi Shiller [29], é realizada a execução de missões autônomas em ambientes simulados cujo o foco principal é a comparação entre os planejadores *online* e *offlines*. Para planejamentos *offlines* utilizou-se o RRT (Rapdly-Exploring Random Tree) [30] e para a abordagem *online*, as equações de Hamilton Jacobi Bellman (HJB) [31]. O RRT se mostrou com maior custo computacional, entretanto, o caminho percorrido foi mais seguro se comparado às manobras feitas muito próxima aos obstáculos. Outro ponto importante é que planejador *online* forneceu caminhos mais próximos à trajetória ótima ao desviar dos obstáculos. Do ponto de vista temporal, o planejador *online* possui menores velocidades ao longo do caminho, levando tempo maior na execução das missões. Neste sentido, fica claro que uma boa opção é o planejador de caminhos híbrido que mescla as vantagens das duas vertentes. Nesse trabalho não foram abordados técnicas de localização, ou seja, admite-se o conhecimento da posição do robô a todo momento.

Tendo em vista as melhorias encontradas na fusão dos planejadores, [32] propõe a abordagem deliberativa em ambientes simulados baseado na modificação do algoritmo RRT, o RRT*, que são adaptados para serem sensíveis à obstáculos que surjam durante a missão e que não esteja mapeados. O autor defende uma técnica inteligente de replanejamento da missão quando obstáculos que obstruíssem o caminho fossem detectados. Porém, a exigência computacional no replanejamento da missão pode se tornar altamente elevada. Mesmo tendo alcançado o objetivo final replanejando a missão, o robô se manteria estático quando estivesse buscando uma nova trajetória livres de obstáculos. Portando para um número grande de obstáculos dinâmicos o robô permaneceria um longo tempo parado. Os autores assumem que o robô saiba sua localização a todo instante e também não há informações de quais sensores foram utilizados na identificação dos obstáculos.

Em [33] foram propostas missões autônomas de robôs móveis para operações em elevadas profundidades marítimas com foco na exploração de petróleo. Primeiramente a unidade de controle cria uma rota de maneira deliberativa utilizando a técnica bio-inspirada denominada algoritmos genéticos, e inicializa a missão. Entretanto, possíveis obstáculos surgem de acordo com a navegação, e uma abordagem *online* baseada nas curvas de Bézier é responsável por evitar possíveis colisões. Para a localização do robô, os autores utilizaram os sensores *encoders* (odometria) que forneciam a posição do robô, entretanto, erros inerentes ao deslocamento das rodas vão sendo acumulados ao longo da missão. Foram alcançados resultados práticos em um robô com tração diferencial em um ambiente previamente conhecido.

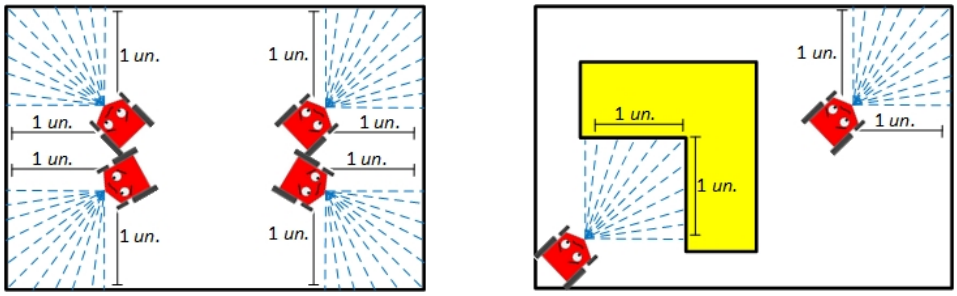
Uma boa alternativa para o problema do erro acumulativo, é utilizar a união de informações providas de vários sensores. Esta prática é tão eficiente que chega a ser tema

específico em diversos trabalhos de localização. Uma pesquisa relacionada a esse tema foi apresentada em [34], os autores propuseram a localização fundindo as informações oriundas de um sonar e um sensor odométrico a partir de técnicas baseadas no Filtro de Kalman (FK). Os testes foram realizados em um ambiente interno estático através de um robô com tração diferencial. O trabalho desenvolvido em [28] apresenta bons resultados na localização de um robô móvel. O sistema robótico reconhecia no mapa os marcadores artificiais caracterizados por cores distintas que foram distribuídos pelo ambiente simulado. Fundiu-se o sensor odométrico com o detector dos objetos cilíndricos artificiais contidos no mapa (o trabalho não detalhou as informações que caracterizassem esse sensor) também com técnicas baseadas no FK. Os marcadores foram armazenados em um mapa baseado em marcos para diminuir o custo computacional.

Em [35] foi proposta uma abordagem híbrida no planejamento de caminhos para missões autônomas em múltiplos robôs móveis de maneira simulada. O planejamento *offline* é executado no início da missão utilizando a biblioteca OPML. Durante a navegação utilizou o algoritmo dos campos potenciais artificiais (CPA) para desviar o robô dos obstáculos e dos outros robôs. Os robôs tinham a informação externa da real localização dos mesmos. Ao final, as missões de deslocamento puderam ser realizadas, entretanto, os robôs enfrentaram pequenos problemas ao desviar dos obstáculos. O autor acredita que para melhoria de seu trabalho, é necessário substituir o planejamento *offline* dado pela biblioteca OPML por outros tal qual alguns baseados nas RRTs. O planejador *online* utilizado apresenta problemas de mínimos locais e movimentos oscilatórios em passagens estreitas.

O projeto do Robô "Caesar"conta com sensores do tipo *laser scan* para se localizar pelo ambiente. O robô conhece o mapa em que está localizado e a partir disto planeja a sua operação de maneira *offline* através do algoritmo A* [36]. Este planejador encontra o caminho ótimo a partir da discretização do mapa apresentado ao robô. O planejador reativo entra em ação quando obstáculos forem encontrados. Dessa maneira, é criado um mapa local a partir das leituras do *laser scan*. O algoritmo A* é executado novamente e encontra uma rota que desvia das possíveis zonas de colisão. Mais detalhes dessa abordagem foram expostas em [37]. A localização é realizada através de uma adaptação do método de Monte Carlo [38] e a interpretação do ambiente é realizada a partir das leituras do *laser scan*. Este tipo de localização não apresenta erros acumulativos tal qual o uso de *encoders*, entretanto, ao extrair as informações do ambiente a partir do *lasers scan* problemas de locais ambíguos ou simétricos podem surgir, pois, esse tipo de sensor extrai características carentes do ambiente. Alguns exemplos que retratam esse problema são ilustrados na Figura 2 que correspondem às posições ambíguas que um ambiente qualquer pode oferecer. Esses cantos são bem comuns em ambientes *indoors* proporcionando maiores localidades que podem ser consideradas simétricas. Já na Figura 2 (b) representa-se também a possibilidade de ambiguidades a partir de obstáculos que possam surgir durante a missão.

Figura 2 – Exemplos de Ambiguidades: (a) Ambiguidades naturais (b) Ambiguidades ocasionadas por objetos.



(a) (b)

Em contrapartida, a visão computacional surge como uma proposta de extração de informações do ambiente de maneira rica se comparada a sonares e *lasers scans*. Porém o custo computacional na análise das imagens é superior em relação à manipulação das informações providas de outros sensores de distância. A capacidade das câmeras e seus baixos custos econômicos tornam esse tema convidativo a ser explorado [39] nos estudos que envolvem robótica em geral. Em [40] propõe-se a navegação autônoma de robôs móveis em ambientes internos utilizando apenas visão computacional. A primeira fase consiste num mapeamento do ambiente. Depois, o robô está capacitado a encontrar uma trajetória para atingir a posição objetivo utilizando um planejador global baseado em buscas de *Dijkistra's*. Caso algum obstáculo inesperado surja durante a missão, um algoritmo que converte a imagem em um conjunto de distâncias de forma similar a um *laser scan* é acionado. O sistema robótico cria mapas locais (em grades de ocupação ou *occupacy grids*) com a leitura destas distâncias e define as localidades que o robô poderá se dirigir para que o obstáculo seja contornado e o caminho gerado pelo planejador global seja retomado. A localização durante as missões é obtida utilizando o Algoritmo Adaptativo de Monte Carlo (AMCL) [41] que também é sensível aos problemas de simetria. No trabalho [39] é proposto um meio de localização para robôs autônomos baseada em visão computacional. O robô de tração diferencial explora um ambiente interno de maneira tele-operada encontrando algumas marcas naturais, após isso, o sistema robótico estará capacitado a se localizar ao encontrar os marcadores definidos anteriormente através de técnicas de processamento de imagens *SURF*.

Trabalhos que fundem câmeras com outros sensores com propósito de navegação de robôs móveis também são focos de estudo. Em [42, 43] é apresentada uma proposta de localização utilizando informações de sensores odométricos e de uma câmera. A câmera reconhece o conjunto de linhas que formam as arestas dos pisos pertencentes ao ambiente. A variedade nas características de marcadores naturais de um ambiente influenciam diretamente nos resultados obtidos, já que os mesmos erros de simetria apresentados pelo algoritmo de Monte Carlo podem ocorrer. Nesse sentido ao reconhecer os objetos é importante que os mesmos tenham características distintas.

Um sensor que vem sendo difundido no meio robótico é o *Kinect*, desenvolvido pela *Microsoft* e é caracterizado por usar visão computacional e sensores infra-vermelhos embutidos. A partir do processamento de imagem desse sensor, é possível extrair distâncias tal qual a sonares e *lasers scan*. Em [44] desloca-se por um ambiente um robô dotado do sensor *Kinect* com intuito de mapear o ambiente. A localização é dada pela comparação das imagens armazenadas em um banco de dados capturadas pelo *kinect* na fase de mapeamento com as imagens geradas do local onde o robô se encontra. Entretanto, o sucesso da localização é fortemente dependente da quantidade de imagens geradas no processo de mapeamento, sendo que em alguns casos esta localização não foi possível de ser obtida. Além disso, o trabalho utilizou o planejador *offline* RRT* para obter uma trajetória ao realizar missões de deslocamento pelo mapa gerado. Não considerou-se obstáculos dinâmicos ao longo das missões, ou seja, caso algum obstáculos surja ao longo da missão o robô poderia colidir. Os algoritmos baseados em RRT levam em conta as restrições de movimento e dinâmicas impostas que o sistema robótico possui. O método consegue se comportar de maneira satisfatória em espaço de buscas elevados devido ao seu aspecto fundamentado em amostras conforme abordado em [45], cuja proposta principal é o movimento de um robô autônomo por um ambiente controlado e estático. A localização ficou a cargo do sensor *Kinect* que é responsável por localizar o sistema robótico em diversas tarefas bem como a percepção de sinais controlados espalhados pelo mapa. Variações no mapa ao longo da missão de deslocamento podem causar colisões no robô. O RRT* que foi utilizado no trabalho [44], é um planejador apresentado em [46] cujo o foco principal é obter caminhos de maneira assintoticamente ótima, isto é, a medida que o número de amostragens cresce, as rotas encontradas tenderão ao ótimo. Entretanto, o custo computacional é elevado quando comparado aos RRT clássico.

Planejadores que utilizam RRTs em missões autônomas também são encontrados em [47], cujo foco principal é a concepção de um algoritmo com base nas RRTs denominado DT-RRT que encontre melhores trajetórias em um ambiente simulado. Os autores não apresentaram resultados que tornassem possível o desvio de obstáculos (planejamento *online*). Utilizou-se o AMCL para a localização do robô durante as missões com o auxílio de um sensor *laser scan*.

Dada a complexidade de missões autônomas, muitos trabalhos abordam somente um único algoritmo que será empregado em uma missão autônoma. Nos parágrafos a seguir serão retratados publicações cujo foco é apresentar apenas um planejador de caminhos ou um método de localização.

Trabalhos cujo o foco das pesquisas são somente os planejadores deliberativos são comuns na literatura. Em [48] os autores propuseram uma técnica chamada Informed-RRT* cujo o objetivo era melhorar o caminho gerado pela RRT. Após um caminho ser encontrado até a posição objetivo, uma elipse seria traçada ao redor do mesmo fazendo

com que a amostragem ficasse restrita à parte interna da elipse. No decorrer das iterações, a distância de seus eixos iriam diminuindo fazendo com que houvesse uma melhora nos caminhos. No trabalho [49] os autores detectaram um problema com a dispersão dos nós que levavam a informações ambíguas. Por esse motivo, foi proposto o método DRRT cujo o foco principal era aplicar uma regra na amostragem, gerando representações no espaço mais homogêneas com um número menor de nós quando comparado ao RRT. Outro método probabilístico tal qual o RRT foi o denominado Espumas Probabilísticas [50]. São amostradas bolhas em formato de circunferências cujo o centro é um estado possível que o robô poderá se localizar. Essas bolhas poderão ter os raios variados conforme a feição do ambiente presentes no mapa. Ao final é formado um conjunto de bolhas denominado rosário cujos centros representam um arranjo de estados que o robô deverá seguir para chegar ao objetivo. Porém, o trabalho poderá enfrentar problemas em respeitar as restrições cinemáticas do robô. Os trabalhos apresentados nesse parágrafo apenas solucionam o problema do planejamento *offline*, porém é necessário a inclusão de algoritmos para desviar de obstáculos e métodos localizadores para que uma missão autônoma possa ser implementada com sucesso.

Trabalhos cujo foco principal são planejamentos *online* estão presentes [51] e [52]. O primeiro propõe uma solução para os problemas de mínimos locais e movimentos oscilatórios utilizando um campo potencial adaptativo. O campo potencial artificial que será encontrado pelo robô deverá ser proporcional á distância e posição angular que o obstáculo e a posição objetivo se encontram em relação ao robô. O problema de mínimos locais poderá existir caso os obstáculos que prendam o robô for do tipo convexo. Já o segundo, conta com um método para correção dos problemas de mínimos locais e oscilações ao se deslocar em corredores estreitos. Os autores definem uma forma de acumular cargas nas regiões problemáticas até que o robô saia dessas situações. Esse trabalho é capaz de solucionar o problema de mínimos até em obstáculos convexos. Entretanto, o sucesso do método está diretamente associado ao emprego de um *laser scan* cujo valor econômico é alto.

Ainda sobre o problema de localização durante as missões autônomas têm-se resultados satisfatórios na literatura a cerca do uso de câmeras. Uma proposta para a localização em ambientes internos é utilizar marcadores artificiais únicos em um ambiente. Como supracitado em [28] os autores propõem objetos distinguíveis por cores, ainda assim, a medida que novos objetos fossem necessitados a detecção da variedade das cores poderia se tornar um empecilho. O uso de *Ar codes* (*Quick Response Code*) vem se tornando um aliado nos projetos de robôs autônomos no que tange à localização. Esse tipo de marcador consegue armazenar informações, sendo assim, os marcadores são dispersos pelo ambiente com uma assinatura única. Em [53] os marcadores são alocados no teto do ambiente interno, e assim, o robô executa as missões de deslocamento pelas salas se localizando apenas com os *Ar codes*, esse marcador corresponde a um tipo de específico de

QR codes. Todo o trabalho foi desenvolvido utilizando a *framework* ROS (*Robot Operating System*). Para que a missão fosse executada de maneira satisfatória seria necessário que a câmera enxergasse um marcador a todo momento, além disso, não se preocupou com ruídos sensoriais. Em [54] o *QR Code* auxiliou nas missões em um ambiente *indoor*, cada um dos marcadores armazenava a informação de que ação o robô deveria tomar para atingir uma determinada posição objetivo. Nesse trabalho, o robô podia desviar de obstáculos utilizando sonares. Já no trabalho implementado em [55] propõe-se a execução de trajetórias em formato de "L"de um robô em um ambiente interno. A fim de melhorias na localização, os autores propõem a fusão entre a câmera e um sensor inercial. Devido à complexidade das missões autônomas, em alguns trabalhos, não foram abordados meios de se planejar caminhos, dessa maneira, esses trabalhos sempre serão partes que irão ser agregadas à outros algoritmos cuja união formará as missões autônomas.

1.2 OBJETIVOS E JUSTIFICATIVAS

Dada a contextualização apresentada na Seção 1.1, esse trabalho propõe a implementação de missões autônomas de deslocamento utilizando um robô móvel com tração diferencial em ambientes internos. Tendo em vista a vasta gama de situações nas quais os robôs podem ser empregados neste tipo de ambiente (museus, escritórios, escolas, indústrias e residências, etc) é interessante apresentar e desenvolver um trabalho voltado para situações genéricas. O robô tem a finalidade de atingir uma posição final, para isto, ele terá que planejar trajetórias, mapear e se localizar durante a missão. Caso algum obstáculo dinâmico surja, é necessário que o dispositivo desvie sem que ocorra colisões.

O planejamento *offline* será realizado por algoritmos que são baseados em RRTs, já que o custo computacional se comparado à outros planejadores são inferiores. Além disso, a característica amostral permite aplicações em ambientes com grandes dimensões e não apresentam problemas com mínimos locais. Mesmo com seu aspecto amostral, os resultados alcançados nos trabalhos mencionados na revisão bibliográfica foram satisfatórios quando comparados à trajetórias ótimas. A planta do local é repassada ao robô para que o algoritmo encontre um caminho que ligue a posição inicial à final. Primeiramente realizou-se a fusão de duas técnicas já conhecidas na literatura a DRRT com a RRT* visando uma melhoria da RRT. Em paralelo, este trabalho propõe uma nova heurística com intuito de melhorar também a RRT. Essa nova proposta foi dado o nome Direct-RRT que quando unido às outras técnicas será nomeada de Direct-DRRT*. Ao final serão analisados quais são os planejadores *offline* que serão mais viáveis para aplicação na missão.

Considerando um ambiente realista, alguns objetos ou pessoas que possam causar colisões, surgem de maneira inesperada. Esses objetos são considerados dinâmicos, pois, de acordo com sua mobilidade, podem surgir em qualquer local do ambiente sem previsão. Sendo assim, o planejador *online* denominado campos potenciais artificiais tem a função

de desviar os robôs dos devidos empecilhos. Os sensores responsáveis por identificar os objetos são sonares e o sensor de profundidade *Asus Xtion Pro Live* que se comporta de maneira idêntica ao *Kinect*. A grande vantagem do método CPA é que além da característica planejadora, o controlador do robô também é implementado de maneira conjunta. O *Asus Xtion* será o responsável por atualizar o mapa do ambiente. Dessa maneira, se algum obstáculo intervier em uma trajetória gerada pelo planejador *offline* de maneira que o dispositivo robótico fique preso em mínimos locais, um caminho alternativo é concebido. Do ponto de vista econômico, os valores comerciais do *Asus Xtion* são bem inferiores aos *lasers scans*. Pode-se citar o *laser* linha LMS100-10000 da marca *Sick* para ambientes internos cujo preço é aproximadamente 4200€ [56]. Em contrapartida, uma versão mais recente do sensor de profundidade da Asus, conhecido como *Asus Xtion 2* custa aproximadamente 220€ [57]. A atuação dos dois planejadores *online/offline* forma a característica híbrida na elaboração de caminhos para navegação autônoma.

Como visto anteriormente, durante a missão o sistema robótico necessita de sua localização para que a trajetória encontrada pelos planejadores possam ser executadas. Optou-se por utilizar uma câmera para identificar os marcadores artificiais do tipo *Ar codes* (uma variante dos *Ar codes*) que é repassado para o robô através de um mapa com os marcadores. Junto à câmera, utilizou-se sensores odométricos de maneira fundida para que nos momentos em que os marcadores artificiais não fossem visíveis, uma estimativa da posição utilizando a odometria seria possível. Os sensores possuem ruídos que podem ser prejudiciais à missão, sendo assim, o trabalho proposto utilizou-se do Filtro de Kalman Estendido para realizar a filtragem e a fusão da câmera com o sensor odométrico.

O sistema autônomo será gerido pelo *framework* ROS. Os resultados das missões serão obtidos a partir de situações práticas e simuladas.

De maneira resumida essa dissertação apresenta os seguintes objetivos gerais:

- Realizar missões autônomas em robôs móveis com tração diferencial.

Dentro do escopo do objetivo geral, surgem específicos que são:

- Implementar um método para localizar o robô durante a missão;
- Implementar um método para planejar o caminho de maneira *offline*;
- Implementar um método para desvio de obstáculos (planejador *online*);
- Implementar um método para realizar o mapeamento para detectar modificações no ambiente;
- Montar um sistema robótico capaz de realizar as Missões Autônomas.

1.3 TRABALHOS PUBLICADOS

Em decorrência das pesquisas relacionadas a esse trabalhou publicou-se dois artigos no Simpósio Brasileiro de Automação Inteligente (SBAI) 2017. O primeiro artigo apresentou uma metodologia que auxiliasse uma cadeira de rodas a se deslocar em passagens estreitas. A metodologia fez uso de um controle não linear que guiasse o robô nessas situações. Um Filtro de Kalman Estendido foi necessário para localizar o robô em relação aos extremos da passagem. O título do trabalho foi Planejamento e Controle Não Linear Para Passagens Estreitas em Robótica Móvel Assistiva [58].

O segundo trabalho abordou a metodologia de localização apresentada nessa dissertação, com uma pequena variação no modelo matemático da câmera. A proposta principal é localizar o robô em um ambiente interno e conhecido. Utilizou-se um Filtro de Kalman Estendido para fundir as informações de uma câmera e a odometria do robô. Além disso, o filtro também auxiliaria no tratamento dos ruídos sensoriais. A câmera seria responsável por detectar marcadores do tipo *Ar codes* para que o robô pudesse ser localizado com eficiência. Os resultados foram obtidos através de simulação em *Matlab*.

1.4 ORGANIZAÇÃO DO TRABALHO

A presente dissertação está organizada conforme mostrado a seguir:

- **Capítulo 2:** O capítulo destina-se a apresentar a fundamentação teórica dos métodos que serão utilizados por esse trabalho. Também será apresentado o sistema robótico que é composto por um robô diferencial e os sensores que os compõe.
- **Capítulo 3:** Nesse capítulo há a exposição da metodologia adotada para a obtenção dos resultados. Serão mostrados como os métodos da fundamentação teórica poderão ser aplicados para a realização da navegação autônoma. Junto à metodologia também serão apresentados algumas contribuições para a melhoria dos resultados.
- **Capítulo 4:** O capítulo expõe os resultados da navegação autônoma. Serão divididos em três seções: a primeira aborda os resultados do planejador *offline* proposto (Direct-DRRT*); a segunda apresenta os resultados por meio de simulação para a navegação autônoma; a terceira serão apresentados resultados práticos para a navegação autônoma.
- **Capítulo 5:** O capítulo 5 é destinado às conclusões do trabalho bem como as propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ROBÔ COM TRAÇÃO DIFERENCIAL SOB DUAS RODAS

O sistema robótico utilizado por esse trabalho é o *Pionner* P3DX (Figura 3) produzido e comercializado pela empresa *Mobile Robots*. De acordo com a fabricante seus dispositivos são indicados para área de educação e pesquisas envolvendo mapeamento, tele-operação, localização, monitoramento, visão, cooperação entre robôs, manipuladores e navegação autônoma. O produto é comercializado com sensores que auxiliam nas missões robóticas, citam-se 8 sonares dianteiros e *encoders* em cada uma das rodas. O sistema robótico oferece suporte para até 3 baterias de 12 V com capacidade de 7.2 Ah [59].

Figura 3 – Sistema Robótico *Pionner* P3DX

Fonte: [59]

A *Mobile Robots* também comercializa um vasto pacote de *softwares* que estão integrados com o robô. Além disso, acessórios como *laser Scan*, sonar traseiro, comunicação serial *wireless* para operações remotas, manipuladores robóticos e câmeras também são facilmente integradas ao P3DX, tudo isso manipulado por um controlador de baixo nível que oferece interface com computadores externos a partir da camada serial aumentando ainda mais a quantidade de sensores que poderão atuar junto ao P3DX. O robô conta com 2 motores DC para cada uma das rodas que de acordo com [59] proporciona velocidade linear máxima de $1,6m/s$ e angular de $300^o/s$.

Assumindo que suas rodas não deslizam no contato com o chão, esse tipo de robô possui restrições não-holonômicas que não permitem o deslocamento lateral [60]. Entretanto sua formação diferencial permite que o mesmo gire sobre sua base em qualquer direção. Para realizar a missões autônomas com sucesso é conveniente estudar o modelo cinemático do robô.

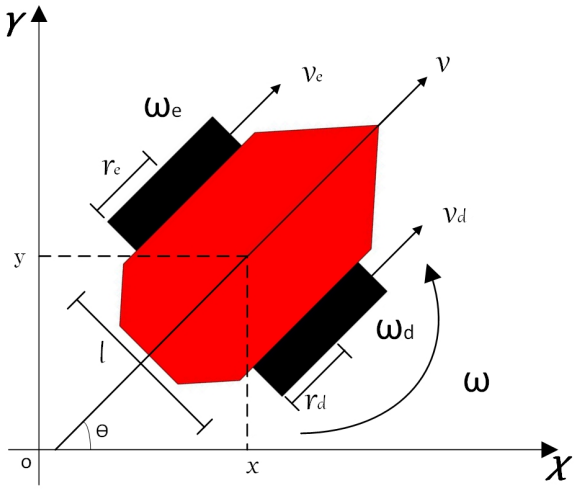
O modelo cinemático representa o conjunto de equações que definem características do movimento bem como suas restrições. A partir de relações algébricas, o modelo cinemático direto visa obter a posição do robô dada as unidades que controlam os atuadores. A Figura 4 corresponde à um estado de uma configuração qualquer do robô representado

pela Equação 2.1:

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.1}$$

em que x e y é a posição do centro do robô e θ a orientação em relação ao referencial global do ambiente de trabalho. Conforme abordado por [60] o modelo cinemático pode ser compreendido de duas maneiras: relação entre as velocidades impostas ao veículo e a relação entre os esforços nos diferentes referenciais (forças e torques oriundas dos motores). Devido à presença do controlador presente no P3DX optou-se por abordar apenas a relação entre as velocidades. As equações descritas nesta seção não se restringem ao modelo comercial P3DX e sim a robôs caracterizados por tração diferencial com duas rodas.

Figura 4 – Configuração do Robô



Fonte: Adaptado de [60]

Relação entre as velocidades

A nomenclatura utilizada na relação entre as diferentes velocidades são definidas da seguinte maneira:

- l é a largura do robô;
- r_d e r_e são os raios das rodas direita e esquerda, respectivamente;
- ω_d e ω_e as velocidades angulares das rodas direita e esquerda, respectivamente;
- v_d e v_e são as velocidades lineares das rodas direitas e esquerdas, respectivamente;
- v e ω são as velocidades lineares e angulares respectivamente no referencial do robô.

A velocidade linear em cada uma das rodas é definida a partir das velocidades angulares das mesmas conforme apresentado na Equação 2.2.

$$v_d = \omega_d r_d \qquad\qquad v_e = \omega_e r_e \tag{2.2}$$

A velocidade em cada uma das rodas obtidas a partir de v e ω encontrada no centro de massa do robô é mostrado pela Equação 2.3.

$$v_d = v + \left(\frac{l}{2}\right)\omega \qquad\qquad v_e = v - \left(\frac{l}{2}\right)\omega \tag{2.3}$$

Manipulando as Equações 2.2 e 2.3 pode-se chegar em:

$$v = \omega_d \frac{r_d}{2} + \omega_e \frac{r_e}{2} \qquad\qquad \omega = \omega_d \frac{r_d}{d} - \omega_e \frac{r_e}{d} \tag{2.4}$$

Em [60] esta análise foi escrita na forma matricial de duas maneiras, conforme mostrado nas Equações 2.5 e 2.6.

$$\mathbf{U} = {}^{u}T_{\omega}\mathbf{W} \tag{2.5}$$

$$\mathbf{W} = ({}^{\mathbf{U}}T_{\mathbf{W}})^{-1}\mathbf{U} \tag{2.6}$$

onde,

$$\mathbf{U} = \begin{bmatrix} v \\ \omega \end{bmatrix}, \qquad {}^{\mathbf{U}}T_{\mathbf{W}} = \begin{bmatrix} r_d/2 & r_e/2 \\ r_d/l & -r_e/l \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} \omega_d \\ \omega_e \end{bmatrix}, \text{ e} \qquad {}^{\mathbf{W}}T_{\mathbf{U}} = ({}^{\mathbf{U}}T_{\mathbf{W}})^{-1} = \begin{bmatrix} 1/r_d & l/2r_d \\ 1/r_e & -l/2r_e \end{bmatrix}$$

Em que $\mathbf{U}$ é vetor formado pelas velocidades no referencial do robô e $\mathbf{W}$ corresponde às velocidades angulares em cada uma das rodas (espaço dos atuadores).

De acordo com a Figura 5, a relação entre as velocidades no referencial local do robô (R) e do global (I) é obtida a partir da multiplicação da matriz de rotação ortogonal $R(\theta)$

com as velocidades nos referenciais global (Υ_I) e local do robô (Υ_R) conforme apresentado pelas Equações 2.7 e 2.8:

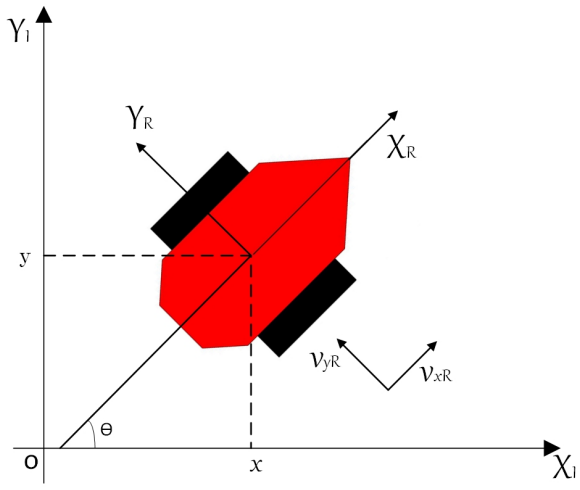
$$\Upsilon_{\mathbf{R}} = R(\theta)\Upsilon_{\mathbf{I}} = \begin{bmatrix} \cos(\theta) & \text{sen}(\theta) & 0 \\ -\text{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \Upsilon_{\mathbf{I}} \tag{2.7}$$

$$\Upsilon_{\mathbf{I}} = R(\theta)^{-1}\Upsilon_{\mathbf{R}} = \begin{bmatrix} \cos(\theta) & -\text{sen}(\theta) & 0 \\ \text{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \Upsilon_{\mathbf{R}} \tag{2.8}$$

onde $\Upsilon_R = [v_{xR}\ v_{yR}\ \omega_R]^T$ e $\Upsilon_I = [v_{xI}\ v_{yI}\ \omega_I]^T$ são os vetores das velocidades nas coordenadas de seus respectivos referenciais. Devido ao aspecto construtivo do robô diferencial, no referencial local, somente são válidas velocidades ao longo do eixo X_R, nesse sentido v_{yR} possui valor nulo obedecendo as restrições de movimento. Dessa maneira, adota-se $v_{xR} = v$ e $\omega_R = \omega$ resultando em uma simplificação na matriz de rotação ortogonal conforme mostrado pela Equação 2.9:

$$\Upsilon_{\mathbf{I}} = {}^{\mathbf{q}}T_{\mathbf{U}}\mathbf{U} = \begin{bmatrix} \cos(\theta) & 0 \\ \text{sen}(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.9}$$

Figura 5 – Referenciais Adotado



Fonte: Elaborado pelo autor

O modelo cinemático incremental direto pode estabelecer a posição $\dot{\mathbf{q}} = [\dot{x}\ \dot{y}\ \dot{\theta}]$ quando o robô é submetido à ações de controle por um período de tempo Δt, esta abordagem é aconselhável nas implementações computacionais. O método define os

incrementos lineares (Δx e Δy) e angulares ($\Delta\theta$) que serão somados à posição atual $\mathbf{q}$ para obter $\dot{\mathbf{q}}$ conforme a Equação 2.10.

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{2.10}$$

Os incrimentos podem ser computados em intervalos de tempo Δt e obtidos de acordo com as velocidades impostas ao robô utilizando relações algébricas.

A relação cinemática entre as velocidades em cada um dos referenciais adotados até aqui possuem restrições matemáticas que devem ser definidas. Essas restrições foram abordadas em [61] e dependem do tipo de roda que está inserida no sistema robótico, para esse caso, adotou-se a análise da roda padrão fixa que não permite movimentos verticais e seu ângulo no eixo de rotação é fixo em relação ao chassi do robô (no P3DX o ângulo é de 0^o). A primeira restrição (já supracita) consiste na definição de que o robô deverá ter velocidades em y no referencial local nula (não-holonômica). A segunda restrição aborda o deslizamento das rodas que pode ser analisada a partir do conjunto de Equações presentes em 2.3. Percebe-se a combinação das velocidades lineares do robô resultando nas respectivas velocidades lineares em cada uma das rodas, dessa maneira, a velocidade que o robô se encontra depende apenas da rotação das rodas. A parcela $\frac{l}{2}\omega$ representa a velocidade linear oriunda do movimento rotacional realizado pelo sistema robótico. Caso alguma das rodas deslizasse, haveria a modificação na velocidade do robô, assim, a equação não poderia ser igualada somente aos deslizamentos das rodas.

O modelo cinemático não considera as características inerciais, deformações na estrutura e forças que se originam com o deslocamento (atrito, escorregamento, etc), para tal, é necessário definir um modelo dinâmico do sistema. Em geral, os trabalhos semelhantes a esse não abordam a modelagem dinâmica, pois, deslocamentos a baixas velocidades em terrenos planos e horizontais com contato suficiente para que não ocorra escorregamentos são suficientes para representar a locomoção do robô [62].

O controlador de baixo nível presente no P3DX coordena as unidades de controles em cada um dos atuadores, isto é, o torque necessário para que cada roda gire à determinada velocidade angular. Essa característica facilita o controle do operador, já que o controlador necessita das informações relacionadas às velocidades lineares e angulares no referencial ($\mathbf{U}$) do robô no decorrer das missões.

2.2 TRANSFORMAÇÕES HOMOGÊNEAS

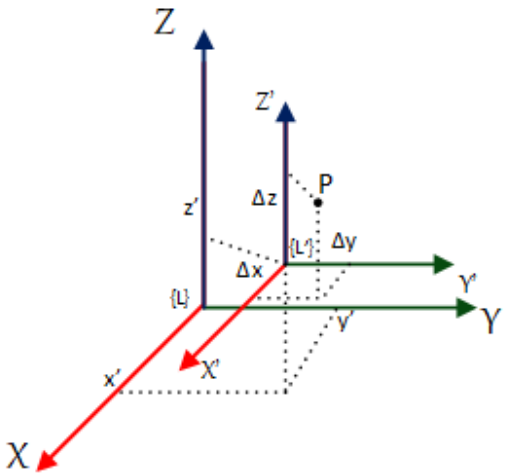
Ao longo das missões autônomas, os robôs deverão interagir com outros corpos que estão envolvidos no ambiente. Cada um desses, possui seu próprio conjunto referencial

denominado *frame*. Um exemplo dessa interação foi mostrada no modelo cinemático, onde o robô precisa encontrar sua posição global no ambiente dada as velocidades que foram impostas ao mesmo. Mostrou-se que em relação ao *frame* do robô, as velocidades lineares se encontram somente no eixo x, dessa maneira, deve haver uma transformação homogênea para que a posição em relação à um referencial global seja obtida, nesse caso, houve uma rotação ortogonal.

Existem diversos meios de se representar um conjunto de coordenadas [63]. Nesse trabalho optou-se por utilizar o denominado ângulos fixos X-Y-Z. Deve-se ressaltar que uma orientação só faz sentido em relação à outro referencial, dessa maneira, pode-se definir como transformação homogênea o conjunto de relações algébricas necessárias para transformar um referencial em outro, por meio de rotações e translações.

Na Figura 6 apresenta-se o *frame* global {L}, onde é possível notar que um novo conjunto de referenciais é formado a partir dos deslocamentos lineares nas coordenadas x, y e z denominado {L'}. Nota-se que para atingir o novo conjunto, não foi necessário nenhum movimento angular.

Figura 6 – Translação entre Dois *Frames*



Fonte: Elaborado pelo Autor

Analisando a Figura 7 é interessante encontrar as coordenadas do ponto P no referencial global. A origem do *frame* {L'} possui coordenadas em relação ao global x', y' e z' sendo denominadas de coordenadas generalizadas. O ponto **P** possui deslocamentos em relação ao novo referencial ({L'}) Δx, Δy e Δz. O ponto **P** pode ser encontrado realizando uma transformação homogênea **T** que leve a origem {L'} a {L} conforme descrito na

equação 2.11:

$$P = {}^{\{\mathbf{L'}\}}_{\{\mathbf{L}\}}\mathbf{T}\mathbf{X'} \tag{2.11}$$

onde a matriz de transformação homogênea $\mathbf{T}$ está presente na Equação 2.12:

$${}^{\{\mathbf{L'}\}}_{\{\mathbf{L}\}}\mathbf{T}(\Delta x, \Delta y, \Delta z) = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

e a relação de {L} com {L'} em 2.13:

$$\mathbf{X'} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \tag{2.13}$$

Além da translação, os eixos podem apresentar rotações entre um referencial e outro. Cada um desses movimentos recebe um nome de maneira convencional. As rotações em X, Y e Z são denominadas de *roll*, *pitch* e *yaw* respectivamente. Para cada rotação de um eixo, existirá uma matriz de transformação distinta. A Figura 7 corresponde a uma rotação em um ângulo γ, β e α em X, Y e Z no *frame* {L}. Dessa rotação é originado o novo referencial denominado {L'}.

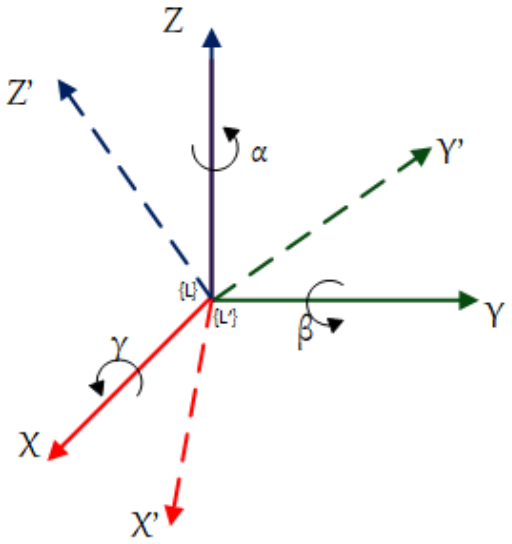
As matrizes de rotações poderão ser vistas nas Equações 2.14, 2.15 e 2.16:

$$\mathbf{R_x}(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha & 0 \\ 0 & \operatorname{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

$$\mathbf{R_y}(\beta) = \begin{bmatrix} \cos\beta & 0 & \operatorname{sen}\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

$$\mathbf{R_z}(\gamma) = \begin{bmatrix} \cos\gamma & -\operatorname{sen}\gamma & 0 & 0 \\ \operatorname{sen}\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

Figura 7 – Rotações em Um *Frame*



Fonte: Elaborado pelo Autor

Para compreender de maneira mais fácil é interessante apresentar um exemplo. Um referencial $\{\mathbf{U}\}$ é obtido após realizar uma rotação do global ($\{\mathbf{L}\}$) na seguinte ordem: uma rotação de γ graus em $\mathbf{X}$; uma rotação de β graus em $\mathbf{Y}$; uma rotação de α graus em $\mathbf{Z}$. A transformação homogênea que representará a relação do referencial $\mathbf{L}$ com $\mathbf{U}$ é calculada conforme a Equação 2.17.

$$R = R_Z(\alpha)R_Y(\beta)R_Z(\gamma) \tag{2.17}$$

Ressalta-se que a multiplicação das matrizes de rotação deve acontecer de maneira invertida à ordem de rotações realizadas no referencial global. Outro ponto que merece destaque é que a ordem deverá ser obedecida, pois as multiplicações de matrizes não possuem características comutativas. Para os casos em que o referencial global for também transladado, a matriz de translação deverá vir multiplicando logo após a última rotação.

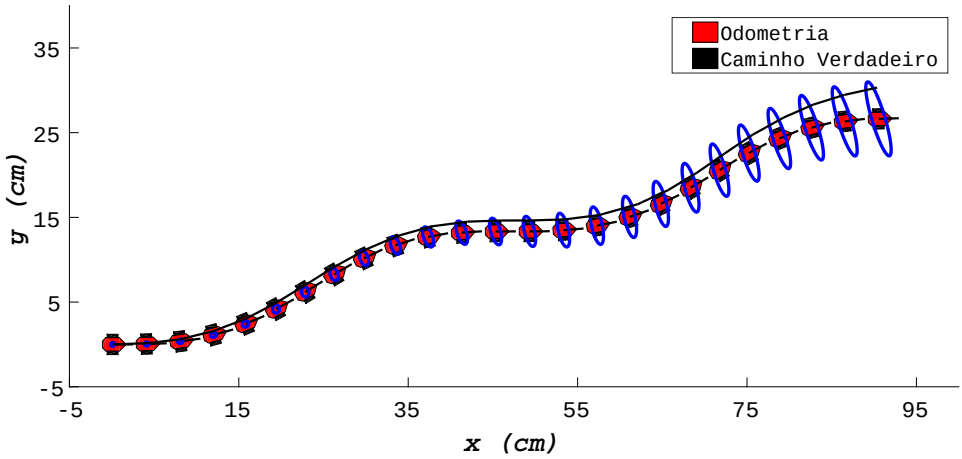
2.3 SENSOR *ENCODER*

Os *encoders* são definidos como dispositivos eletromecânicos que possuem capacidade de reproduzir pulsos a partir da rotação do eixo no qual está acoplado. Na robótica, os *encoders* podem ser utilizados para medir velocidades ou estimar a posição do robô. O P3DX conta com um *encoder* incremental baseado em interrupções ópticas em cada uma das rodas realizando uma contagem de 500 pulsos em uma única rotação [64], isto é, considerando que o raio das rodas possuem comprimentos de $97,5mm$ é possível perceber

deslocamentos de $1,225mm$ a cada pulso gerado por esse sensor.

O controlador interno do P3DX realiza a integração dos dados do *encoder* proporcionando a estimativa da posição que é denominada odometria. Esse tipo de localização é dita local ou relativa, já que *encoders* extraem apenas informações internas do robô. A posição é estimada de maneira integrativa (somatório) utilizando a posição relativa anterior, dessa maneira, os erros inerentes aos sensores tal como os ruídos vão se acumulando ao longo do processo conforme apresentado pela Figura 8. Quando a posição é estimada tendo como base a velocidade em que as rodas do robô se encontram, diz-se que o método é do tipo *dead reckoning* [65].

Figura 8 – Incertezas Relacionadas à Odometria.



Fonte: Elaborado pelo Autor

Em [66], as incertezas relacionadas à localização quando considerado que ruídos gaussianos interfiram na estimativa, são representadas a partir de elipses que crescem com o aumento da distância percorrida. Percebe-se o afastamento da posição dada pelo *encoder* com a posição real do robô bem como o aumento das elipses. Desta maneira, percebe-se a necessidade de utilizar outros sensores em concomitância aos *encoders* para que as incertezas diminuam ao longo das missões.

A informação do sensor *encoder* pode substituir os incrementos definidos no modelo cinemático. Inicialmente mostrou-se que os deslocamentos lineares e angulares que eram somados, foram obtidos a partir das velocidades e frações de tempo. Entretanto, esse modelo tem incertezas nas leituras das velocidades e na medição do tempo. Dessa maneira, [67] define o modelo odométrico que visa obter os incrementos dada duas posições coletadas de maneira consecutiva do sensor *encoder*.

Os incrementos são extraídos da Figura 9 e dividem o movimento entre dois pontos distintos ($\mathbf{X_{t-1}}$ e $\mathbf{X_t}$): há uma primeira rotação (δ_{rot1}) até que a posição em $t-1$ esteja

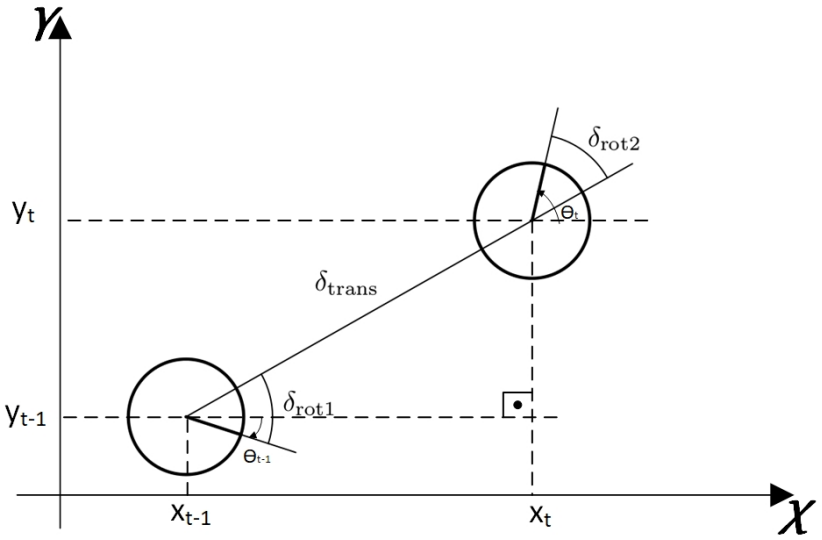
apontada para a $\mathbf{X_t}$, depois desloca-se (δ_{trans}) o ponto $\mathbf{X_{t-1}}$ até $\mathbf{X_t}$ e por fim é realizada uma última rotação (δ_{rot2}). Essa caracterização possibilita a correção da posição que será apresentada na metodologia dessa dissertação, e as Equações que representam esses deslocamentos estão presentes em 2.18, 2.19 e 2.20:

$$\delta_{rot1} = \text{tg}^{-1}\left(\frac{y_t - y_{t-1}}{x_t - x_{t-1}}\right) - \theta_{t-1} \tag{2.18}$$

$$\delta_{trans} = \sqrt{(x_{t-1} - x_t)^2 + (y_{t-1} - y_t)^2} \tag{2.19}$$

$$\delta_{rot2} = \theta_t - \theta_{t-1} - \delta_{rot1} \tag{2.20}$$

Figura 9 – Modelo Odométrico



Fonte: Adaptado de [67]

Dessa maneira, é definido os incrementos da Equação 2.10 conforme as Equações 2.21, 2.22 e 2.23. Deve-se ressaltar que os valores dos deslocamentos encontrados estão corrompidos com ruídos provenientes do sensor *encoder*. Os erros odométricos são dependentes de vários fatores tais como, o chão que o robô se locomove, a velocidade e a taxa de amostragem dos sensores. Para este trabalho modelou-se de maneira simplificada um ruído gaussiano com média zero que é somado aos incrementos Δx, Δy e $\Delta\theta$ a cada iteração.

$$\Delta x = \delta_{trans}\cos(\theta_{t-1} + \delta_{rot1}) \tag{2.21}$$

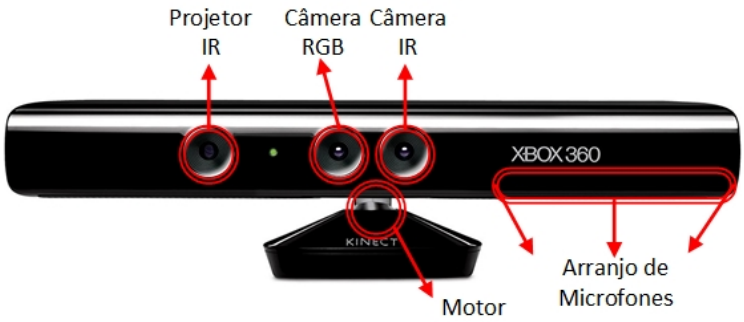
$$\Delta y = \delta_{trans} \operatorname{sen}(\theta_{t-1} + \delta_{rot1}) \tag{2.22}$$

$$\Delta\theta = \delta_{rot1} + \delta_{rot2} \tag{2.23}$$

2.4 SENSOR KINECT

Em novembro de 2010, a empresa *Microsoft* anunciou um acessório destinado a reconhecer o movimento dos jogadores do console *XBOX 360* (Figura 10). O novo dispositivo, nomeado de *kinect*, quebrou algumas barreiras no que tange às interações entre homem e vídeo-game: antes era necessário o contato físico com algum controle, agora com o novo acessório, seria possível se divertir realizando movimentos corporais e falas [68]. A novidade da *Microsoft* foi sinônimo de sucesso, vendendo mais de 8 milhões de aparelhos nos dois primeiros meses, rendendo o título de "*fatest selling consumer electronics device*"pelo Guiness World Records no ano de 2011 [69].

Figura 10 – Dispositivo *Kinect*.



Fonte: Adaptado de [70]

O *Kinect* pode ser considerado um sensor de profundidade que reúne diversos outros sensores tornando possíveis as interações dos usuários com o console. Devido a esses aspectos, o acessório passou a ser objeto de pesquisa para outras aplicações como, por exemplo: plataforma educacional [71], plataformas fisioterápicas [72, 73], assistência a idosos [74], robótica em geral, etc.

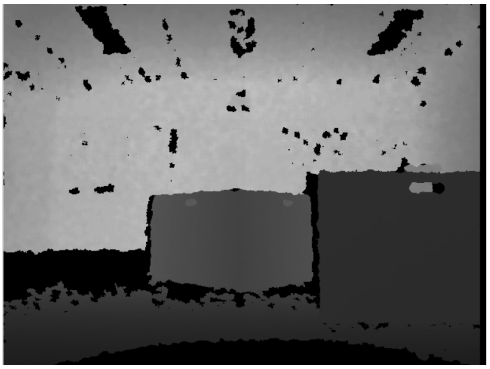
Um conjunto de 4 microfones são os responsáveis por captar o áudio do ambiente no qual o *kinect* está inserido. Para captar imagens é usada uma câmera RGB de 8 *bits* com resolução de 640×480 *pixels* com uma taxa de atualização máxima de 30 Hz [44] (Figura 11 (b)). Além destes sensores, o dispositivo pode mapear o ambiente em formato 3D a partir de sensores profundidade. Uma câmera monocromática infravermelha (IR) CMOS de resolução 320×240 *pixels* e também taxa de atualização de 30 Hz se une a sensores projetores infravermelhos para reconhecer o ambiente com precisão [75]. A partir

do IR, o dispositivo gera uma imagem em escalas de cinza, isto é, quanto mais próximo o objeto se encontra do *kinect* a cor tenderá ao preto, caso contrário, quanto mais distante a cor tenderá ao branco (Figura 11 (c)). A imagem em escala de cinza é sensível à distâncias que variam de $0,4m$ a $4m$, fora destes limites, a escala de cinza é caracterizada por *pixels* pretos. A Figura 11 apresenta os registros feitos pelo *kinect* quando obstáculos se localizam a frente dos mesmos. Propositalmente coloca-se o obstáculo mais próximo do sensor se comparado ao quadrado, nesse sentido é possível perceber que a tonalidade do bloqueio mais próximo corresponde à uma imagem na escala de cinzas mais escura se comparado ao outro obstáculo. A partir do processamento da imagem em escala de cinzas é possível encontrar as distâncias que os obstáculos se encontram ao longo campo de visão do *Kinect*.

Figura 11 – Exemplo do Comportamento de um sensor de Profundidade: (a) Disposição do dispositivo *kinect* e obstáculos (b) Imagem registrada pela câmera RGB (c) Imagem gerada em escala de cinza.

(a)

(b)



(c)

Fonte: Elaborado pelo autor

O *kinect* possui um ângulo de abertura horizontal de aproximadamente 57^o e

vertical de 43^o. Para aumentar a versatilidade do produto, a *Microsoft* acoplou a base do acessório um motor capaz de fornecer um deslocamento angular máximo de $\pm 27^o$.

O dispositivo *kinect* conta um processador e um *software* que é responsável pelo reconhecimento facial, filtragem dos sinais de áudio, discriminação de vozes distintas e criar a imagem de profundidade. De acordo com [75] as grandes inovações do *kinect* é o poder de reconhecer o individuo em inúmeras posições distintas bem como suas expressões faciais.

A única característica que não se encontra no dispositivo *Asus Xtion* presente no robô em estudo deste trabalho é o motor na base, entretanto, a mesma angulação pode ser realizada de maneira manual.

2.5 SENSOR SONAR

Sensores baseados em sonares utilizam a propagação do som para encontrar obstáculos. Esse princípio pode ser estendido até a vida animal onde os morcegos, golfinhos e baleias por exemplo, utilizam da ecolocalização para se mover pelo ambiente no qual vivem e também no encontro presas [76]. Uma das grandes vantagens dos dispositivos sonares é a simplicidade de seu princípio de funcionamento que o torna menos custoso economicamente se comparado a outros sensores *rangefinders* tal qual os *Lasers Scan*.

Os dispositivos sonares contam com dois canais, um emite um sinal de som e o outro, denominado receptor, fica a espera desse sinal. O robô P3DX conta com um cinturão de 8 sonares localizados em sua parte frontal, espaçados em 20^o cobrindo uma faixa de 180^o.

Os sonares podem ser considerados sensores ativos, ou seja, retiram informações dos obstáculos a partir da emissão de sua própria energia no ambiente [77]. Essa característica pode ser prejudicial quando vários sonares estão interagindo no mesmo ambiente, pois, as ondas emitidas podem colidir umas com as outras modificando a informação de distância. Uma possível solução é realizar as medições de cada um dos sonares em instantes de tempo distintos.

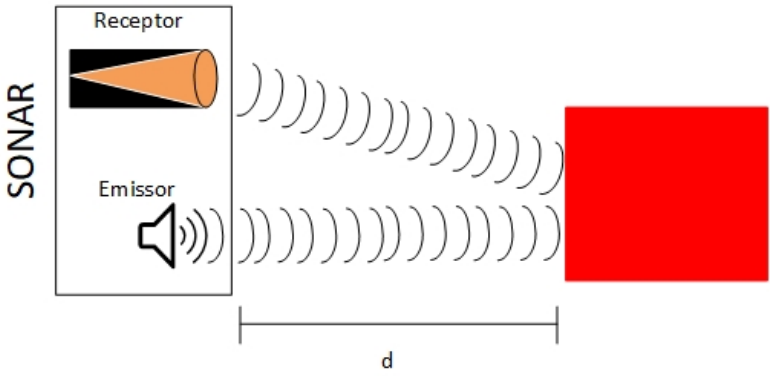
O princípio de funcionamento do sonar consiste em medir o tempo de voo (do inglês, *time of flight*) que o sinal emitido pelo canal emissor leva até chegar ao receptor. Sabendo a velocidade que a onda é emitida, pode-se calcular a distância que o obstáculo em que a onda colidiu se encontra do canal emissor conforme a Equação 2.24 e ilustrado pela Figura 12:

$$d = \frac{t}{2} \times v_s \tag{2.24}$$

onde d é a distância dada pelo sonar, v_s a velocidade do som e t o tempo de ida e volta

da onda emitida. Ressalta-se que o meio impacta diretamente no tempo de voo já que a velocidade do som é diferente e cada ambiente. Outro fator que pode modificar os resultados são os objetos em que as ondas sonoras incidem, as distâncias são mais precisas quando as ondas sonoras chegam até o objeto de maneira perpendicular em relação ao canal emissor. Além disso, há superfícies que absorvem as ondas podendo impedir a reflexão do som.

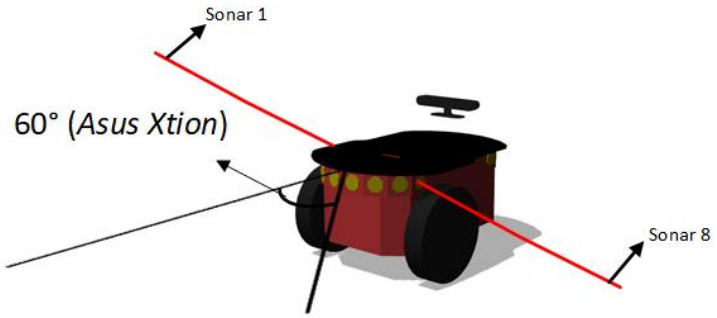
Figura 12 – Funcionamento do Sonar



Fonte: Elaborado pelo Autor

A necessidade do uso do sonar nesse trabalho, foi devido ao pequeno campo de visão do sensor de profundidade *Asus Xtion* (aproximadamente 60º, Figura 13). Nesse sentido utilizaram-se os sonares que se encontravam nos extremos do cinturão, assim, a movimentação teria mais segurança já que as hipóteses de encontrar obstáculos fora do campo de 60º do *Asus Xtion* seria maior.

Figura 13 – Arranjo Entre os Sensores Sonar e *Asus Xtion*



Fonte: Elaborado pelo Autor

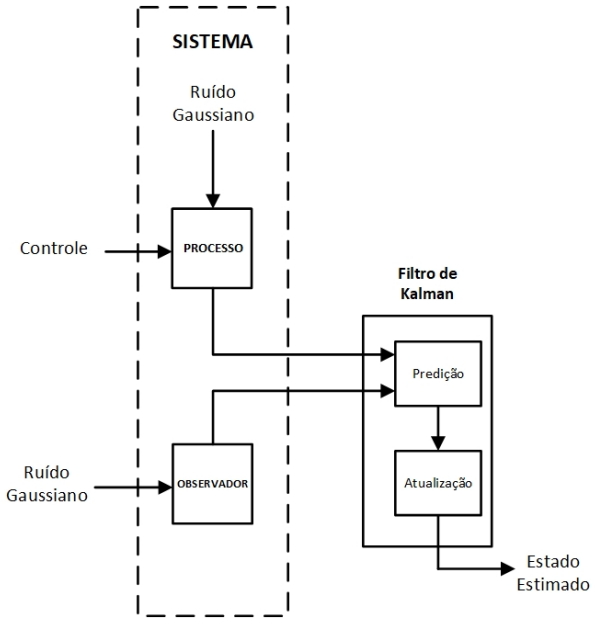
2.6 FILTRO DE KALMAN

Em 1960 Rudolf Kalman propôs um artigo [78] para solucionar problemas relacionados com as áreas de controle e comunicação para dados discretos de natureza estatística, tais como predição de sinais estocásticos, remoção de ruídos, além de detecção de sinais conhecidos (pulsos e senoides) que sejam afetados por ruídos brancos. A esta nova abordagem foi dada o nome de Filtro de Kalman (FK).

O Filtro de Kalman aplica-se em sistemas dinâmicos e lineares que estejam modelados através de espaço de estados. De acordo com [79] a eficiência do FK está atrelada à capacidade de minimização do erro quadrático do sistema de maneira recursiva. Conforme apresentado por Rudolf Kalman, FK é um tipo filtro Bayesiano que utiliza distribuições gaussianas na representação de suas variáveis, neste sentido, manipulações como soma e multiplicação podem ser resolvidas de maneira fácil e analíticas tornando esse fator como propriedade fundamental do filtro e a razão para o custo computacional reduzido [80].

O sistema decomposto em variáveis de estado pode ser descrito conforme as Equações 2.25 e 2.26. Em linhas gerais, as unidades de controle atuam no processo, entretanto, ruídos gaussianos inerentes podem gerar estados errôneos. Então, observações deste processo são realizadas para que os estados errôneos possam ser corrigidos. Entretanto, os dispositivos de observação também são corrompidos por ruídos gaussianos. Neste contexto o Filtro de Kalman atua como um estimador ótimo para o estado conforme apresentado pela Figura 14.

Figura 14 – Atuação do Filtro de Kalman em um Sistema



Fonte: Elaborado pelo autor

$$\mathbf{x_k} = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k \tag{2.25}$$

$$\mathbf{z}_k = \mathbf{C}_k\mathbf{x}_k + \mathbf{v}_k \tag{2.26}$$

onde $\mathbf{x}_k \in \mathbb{R}^n$ e corresponde ao estado que o Filtro de Kalman deve estimar nos instantes k, $\mathbf{F}_k \in \mathbb{R}^{n\times n}$ é denominada matriz de transição de estados e reflete o comportamento do estado na passagem de $k-1$ para k, $\mathbf{u}_k \in \mathbb{R}^l$ é o vetor das entradas de controle, $\mathbf{B}_k \in \mathbb{R}^{n\times l}$ representa, o modelo para a entrada de controle, $\mathbf{z_k} \in \mathbb{R}^m$ é o vetor de observações, $\mathbf{C}_k \in \mathbb{R}^{m\times n}$ a matriz de observação. As variáveis $\mathbf{v}_k \in \mathbb{R}^n$ e $\mathbf{w}_k \in \mathbb{R}^m$ são os ruídos brancos, gaussianos, descorrelacionados, com média zero para o processo e observação respectivamente. A matriz de covariância para o ruído do processo é $\mathbf{Q}_k = \mathbb{E}[w_k w_k^T]$ e a do ruído de observação $\mathbf{R}_k = \mathbb{E}[v_k v_k^T]$.

O FK é dividido em duas etapas: predição (Equações 2.27 e 2.28) e atualização ou correção (Equações 2.29, 2.30, 2.31 e 2.32). A primeira utiliza a equação do processo para predizer o valor do estado e de sua incerteza no instante k a partir do estado anterior $(k-1)$ dada as ações de controle. A fase de atualização corrige o estado predito na primeira etapa dada às observações realizadas, nesta etapa também há atualização da incerteza.

$$\hat{\mathbf{x}}_{\mathbf{k|k-1}} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k \tag{2.27}$$

$$\mathbf{\Sigma}_{k|k-1} = \mathbf{F}_k\mathbf{\Sigma}_{k-1|k-1}\mathbf{F}_k^T + \mathbf{Q}_k \tag{2.28}$$

Onde $\mathbf{\Sigma}_{k|k-1}$ é a covariância do erro no momento k. Nota-se que para inicializar o Filtro de Kalman é necessário um estado inicial e uma covariância do erro diferente de zero para que o filtro funcione. A análise das equações da fase de predição confirma a necessidade de apenas do estado e erros da covariância anteriores, além do conhecimento da dinâmica do sistema.

A fase de atualização inicializa definindo a inovação (Equação 2.29) do Filtro de Kalman, que pode ser interpretada como a diferença entre o modelo observado e o matemático da observação no estado obtido na fase de predição. Quando o valor de $\tilde{\mathbf{y}}$ é baixo significa que o estado estimado na fase de predição está próximo do modelo observado.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{C}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.29}$$

O próximo passo é calcular o ganho de Kalman (K) conforme definido pela Equação 2.30. A característica de estimador de mínimo erro quadrático aparece na equação do

ganho cujo valor obtido é ótimo. O ganho de Kalman pode ser compreendido como os valores observados devem impactar na estimativa do estado atual [22].

$$K_k = \mathbf{\Sigma}_{k|k-1}\mathbf{C}_k^T(\mathbf{C}_k\Sigma_{k|k-1}\mathbf{C}_k^T + \mathbf{R}_k)^{-1} \tag{2.30}$$

Após o ganho de Kalman ser obtido pode-se atualizar o estado estimado na fase de predição $\hat{x}_{k|k-1}$ para $\hat{x}_{k|k}$ denominado estado *a posteriori* mostrado pela Equação 2.31:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k\tilde{\mathbf{y}} \tag{2.31}$$

da mesma maneira a covariância do erro também deverá ser atualizada proporcionalmente ao ganho de Kalman (Equação 2.32) dando origem a um erro *a posteriori* ($\mathbf{\Sigma}_{k|k}$).

$$\mathbf{\Sigma}_{k|k} = (I - K_k\mathbf{C}_k)\mathbf{\Sigma}_{k|k-1} \tag{2.32}$$

O método do Filtro de Kalman pode ser descrito conforme apresentado pelo Algoritmo 1.

Algorithm 1 Algoritmo Filtro de Kalman (FK).

$\hat{\mathbf{x}}_{k|k}$ ▷ Define um estado inicial
$\mathbf{\Sigma}_{k|k}$ ▷ Define uma covariância do erro inicial, não nula
enquanto True **faça**
    $\mathbf{x}_{k-1|k-1} \leftarrow \hat{\mathbf{x}}_{k|k}$
    $\mathbf{\Sigma}_{k-1|k-1} \leftarrow \mathbf{\Sigma}_{k|k}$
    $\mathbf{u_k}$ ▷ Define o vetor variáveis de controle
    $\mathbf{z}_k$ ▷ Realiza uma observação
    função ESTIMAESTADO($\mathbf{x}_{k-1|k-1}, \mathbf{\Sigma}_{k-1|k-1}, \mathbf{u}_k, \mathbf{z}_k$)
        $\hat{\mathbf{x}}_{\mathbf{k|k-1}} \leftarrow \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k$
        $\mathbf{\Sigma}_{k|k-1} \leftarrow \mathbf{F}_k\mathbf{\Sigma}_{k-1|k-1}\mathbf{F}_k^T + \mathbf{R}_k$
        $\tilde{\mathbf{y}}_k \leftarrow \mathbf{z}_k - \mathbf{C}_k\hat{\mathbf{x}}_{k|k-1}$
        $\mathbf{K}_k \leftarrow \mathbf{\Sigma}_{k|k-1}\mathbf{C}_k^T(\mathbf{C}_k\Sigma_{k|k-1}\mathbf{C}_k^T + \mathbf{Q}_k)^{-1}$
        $\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_{k|k-1} + \mathbf{K_k}\tilde{\mathbf{y}}$
        $\mathbf{\Sigma}_{k|k} = (I - \mathbf{K}_k\mathbf{C}_k)\mathbf{\Sigma}_{k|k-1}$
        retorna $\hat{\mathbf{x}}_{k|k}, \mathbf{\Sigma}_{k|k}$
    fim função
fim enquanto

As equações apresentadas até aqui envolvendo FK somente poderão ser empregadas em casos que os modelos que compõe o sistema forem lineares. De acordo com [80] o método Sequencial de Monte Carlo ou Filtro de Partículas consegue realizar a estimativa do estado mesmo nos casos em que o modelo do sistema é não linear. O método gera um grande número de estados aleatórios, entretanto, o processamento se torna mais lento quando comparado aos algoritmos baseados em Filtro de Kalman. Além disso, erros

de simetria poderão ser observados nessas situações. Uma solução para a filtragem em modelos não lineares é o uso do Filtro de Kalman Estendido que será apresentado na próxima seção.

2.6.1 Filtro de Kalman Estendido

O Filtro de Kalman Estendido é uma variação do FK aplicável em sistemas não lineares. Uma característica relevante do Filtro de Kalman é a manutenção de uma Função Densidade de Probabilidade (FDP) nas equações que descrevem o sistema, entretanto, se as variáveis $\mathbf{F}$ e $\mathbf{B}$ são não lineares a manutenção da FDP ficaria comprometida bem como a afirmação de que a Equação 2.28 representaria a covariância do erro [81].

A saída encontrada foi realizar uma aproximação das funções não lineares através de uma função linear tangente à média de sua gaussiana que pode ser obtida expandido os modelos por séries de Taylor [82]. Substituem-se as expressões $\mathbf{F}$ e $\mathbf{B}$ por $g(\mathbf{x}, \mathbf{u})$ e $\mathbf{C}$ por $h(\mathbf{x})$ que são não lineares. O novo sistema pode ser expresso conforme as Equações 2.33 e 2.34.

$$\mathbf{x_k} = g(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{2.33}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{2.34}$$

As Equações do Filtro de Kalman Estendido são mostrados em 2.35, 2.36, 2.37, 2.38, 2.39 e 2.40. Tal qual o Filtro de Kalman, é necessário um estado ($\hat{\mathbf{x}}_{k-1|k-1}$) e uma covariância do erro ($\mathbf{\Sigma}_{k-1|k-1}$) iniciais,

$$\hat{\mathbf{x}}_{\mathbf{k|k-1}} = g(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{2.35}$$

$$\mathbf{\Sigma}_{k|k-1} = \mathbf{G}_k \mathbf{\Sigma}_{k-1|k-1} \mathbf{G}_k^T + \mathbf{R}_k \tag{2.36}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{2.37}$$

$$K_k = \mathbf{\Sigma}_{k|k-1} \mathbf{H}_k^T (\mathbf{H}_k \Sigma_{k|k-1} \mathbf{H}_k^T + \mathbf{Q}_k)^{-1} \tag{2.38}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + K_k \tilde{\mathbf{y}} \tag{2.39}$$

$$\mathbf{\Sigma}_{k|k} = (I - K_k \mathbf{H}_k) \mathbf{\Sigma}_{k|k-1} \tag{2.40}$$

onde $\mathbf{G} \in \mathbb{R}^{n \times n}$ e $\mathbf{H} \in \mathbb{R}^{m \times n}$ são definidos como:

$$\mathbf{G} = \left.\frac{\partial g}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}} \quad \text{e} \quad \mathbf{H} = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}$$

O algoritmo (Algoritmo 2) do EKF também possui a mesma característica de recursividade do FK.

Algorithm 2 Algoritmo Filtro de Kalman Extendido.

$\hat{\mathbf{x}}_{k|k}$ ▷ Define um estado inicial
$\mathbf{\Sigma}_{k|k}$ ▷ Define uma covariância do erro inicial, não nulo
enquanto True **faça**
  $\mathbf{x}_{k-1|k-1} \leftarrow \hat{\mathbf{x}}_{k|k}$
  $\mathbf{\Sigma}_{k-1|k-1} \leftarrow \mathbf{\Sigma}_{k|k}$
  $\mathbf{u_k}$ ▷ Define o vetor variáveis de controle
  $\mathbf{z}_k$ ▷ Realiza uma observação
  função ESTIMAESTADO($\mathbf{x}_{k-1|k-1}, \mathbf{\Sigma}_{k-1|k-1}, \mathbf{u}_k, \mathbf{z}_k$)
    $\hat{\mathbf{x}}_{\mathbf{k|k-1}} \leftarrow g(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k)$
    $\mathbf{\Sigma}_{k|k-1} \leftarrow \mathbf{G}_k \mathbf{\Sigma}_{k-1|k-1} \mathbf{G}_k^T + \mathbf{R}_k$
    $\tilde{\mathbf{y}}_k \leftarrow \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})$
    $\mathbf{K}_k \leftarrow \mathbf{\Sigma}_{k|k-1} \mathbf{H}_k^T (\mathbf{H}_k \Sigma_{k|k-1} \mathbf{H}_k^T + \mathbf{Q}_k)^{-1}$
    $\hat{\mathbf{x}}_{k|k} \leftarrow \hat{\mathbf{x}}_{k|k-1} + \mathbf{K_k} \tilde{\mathbf{y}}$
    $\mathbf{\Sigma}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{\Sigma}_{k|k-1}$
    retorna $\hat{\mathbf{x}}_{k|k}, \mathbf{\Sigma}_{k|k}$
  fim função
fim enquanto

2.7 PLANEJAMENTO DE TRAJETÓRIAS

Para o sucesso das missões autônomas de robôs móveis é necessário um planejamento de caminhos eficiente que ligue o robô de um estado inicial até o estado objetivo de maneira segura. Esta seção possui o propósito de abordar os aspectos dos planejadores utilizados por esse trabalho.

2.7.1 **Espaço de Configurações**

Nos estudos pioneiros no planejamento de caminhos foi definido o termo Espaço de Configurações (C_{space}) para representar matematicamente todas as características da navegação e as possíveis posições que o robô pode atingir [83]. Este conceito permite retratar o robô como um ponto inserido dentro deste espaço facilitando o planejamento de trajetórias.

Quando o robô é simplificado em características pontuais surge o que é denominado planejamento de caminhos (do inglês, *path planning*). De maneira resumida esse problema

é caraterizado por uma posição inicial e uma sequência contínua finita com posições do robô que termina na posição de destino [25].

Os espaços de configurações podem ser determinados a partir de 5 definições cujos conceitos foram retirados de [25] e expostos da seguinte maneira:

- **Definição 1**: O espaço de configurações C_{space} corresponde ao conjunto de configurações que um robô (A) pode se encontrar no ambiente;
- **Definição 2**: A configuração do robô (q) no espaço de configurações possui todas as características necessárias para representar a localização do sistema robótico;
- **Definição 3**: Um caminho (τ) definido para uma missão de deslocamento é caracterizado por possuir uma configuração inicial q_{ini} e uma configuração final q_{fim} conforme a Equação 2.41 de maneira finita;

$$\tau[0:1] \rightarrow C_{space} \mid \tau(0) = q_{ini} \; e \; \tau(1) = q_{fim} \tag{2.41}$$

- **Definição 4**: Os obstáculos C_{obs} em C_{space} é agrupado no conjunto B conforme a Equação 2.42;

$$C_{obs} = \{q \in C_{space} | A(q) \cap B \neq 0\} \tag{2.42}$$

- **Definição 5**: Retirando as regiões de obstáculos do espaço de configurações obtêm-se as regiões navegáveis C_{livre} conforme apresentado pela Equação 2.43;

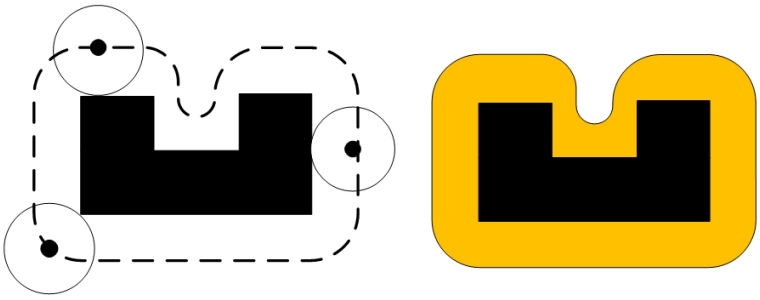
$$C_{livre} = C_{space} \setminus C_{obs} \tag{2.43}$$

A simplificação ocorrida na transformação das dimensões do robô em um ponto, poderá apresentar problemas com colisões em obstáculos caso a modelagem do espaço de configurações não seja realizada de maneira correta. Isso acontece porque a representação pontual é dada pelo centro do sistema robótico. Dessa maneira, determinadas partes de seu corpo poderá se encontrar sob uma região com obstáculo [84]. A Figura 15 apresenta a transformação que deverá ocorrer em um obstáculo (região preta), cujo aumento é proporcional ao formato e tamanho do robô (circunferência). A união das duas regiões serão os novos obstáculos espalhados pelo ambiente. Dessa maneira, percebe-se que os obstáculos foram inchados para que a navegação possa ocorrer de forma segura.

2.7.2 RRTs

As árvores Aleatórias de Exploração Rápida, do inglês, *Rapidly Exploring Random Tree* (RRT) é um método [30] baseado em amostragens, desenvolvido por LaValle em

Figura 15 – Região Segura em Torno de Um Obstáculo



Fonte: Adaptado de [85]

1998 com foco no planejamento de caminhos em ambientes de elevadas dimensões. O RRT com o Mapa de Rotas Probabilístico (do inglês, *Probabilistic Road Map* - PRM) [86] apresentado em 1996 correspondem aos métodos probabilísticos mais utilizados, já que a maioria dos algoritmos não amostrais enfrentam empecilhos com a dimensionalidade dos problemas [25].

O algoritmo PRM é dividido em duas fases cuja primeira denomina-se aprendizagem e consiste em amostrar o espaço de configurações por um determinado período de tempo. Se a configuração sorteada for um local livre de obstáculos ela é mantida, caso contrário é descartada. As rotas são geradas a partir das conexões entre amostras vizinhas para que na próxima fase, uma consulta seja realizada nessas rotas para encontrar um caminho. Entretanto, o custo computacional desse método esta relacionado à quantidade de conexões criadas na fase de aprendizagem podendo chegar a ser inviável em alguns ambientes a busca por rotas. Outros problemas encontrados no uso do PRM são os obstáculos com muitos vértices que também aumentam o custo computacional [25], além desses, o PRM depende que o ambiente permaneça estático.

Como proposta para acelerar o algoritmo surgiu o RRT, que planejaria o caminho de maneira probabilística sem a necessidade de explorar todo o ambiente. Uma árvore cresce a partir de sua raiz que é um nó e se expande pelo ambiente conectando novos nós. Cada um dos novos nós, surgem a partir das amostragens aleatórias que são configurações pertencentes ao espaço C. Além de rapidez, LaValle defendeu que se o crescimento da árvore obedecesse ao modelo cinemático do robô a trajetória encontrada seria factível de ser percorrida por sistemas robóticos com características não-holonômicas já que as restrições de movimento estão presentes no próprio modelo cinemático. A solução do problema é encontrar um conjunto de nós que são conectados por arestas que liguem a raiz até o ponto objetivo.

Um nó pode ser definido como uma estrutura de dados que contém basicamente três informações conforme listado abaixo:

- **Estado:** conjunto de coordenadas no espaço de configurações;
- **Parentesco:** informa qual o nó mais próximo;
- **Custo até o nó raiz:** informa os custos que esse nó possui para chegar até a posição inicial.

Um conceito para avaliação de um planejador é a completude, ou seja, se uma solução existir o método conseguirá encontra-la. Entretanto, por ser baseado em amostragens, o RRT não pode assumir essa completude e possui um conceito mais fraco [84]. O planejador é considerado probabilisticamente completo como provado em [87], isto é, caso exista uma solução e a quantidade de amostras do problema tenda ao infinito a RRT irá encontra-la.

O trabalho de LaValle aponta algumas características que o RRT possui: expansão da árvore em direção a regiões ainda não exploradas; é completo probabilisticamente sob diversas condições; a implementação do algoritmo é simples e a análise de suas desempenho são relativamente fáceis; todos os nós conectados à árvore se ligam até o nó raiz, desta maneira, a trajetória obtida terá a menor quantidade de nós possível; o método não é suscetível a mínimos locais.

Para facilitar o entendimento do método, tabelou-se as principais variáveis de configurações e suas respectivas funções que aparecerão ao longo do algoritmo na Tabela 1.

Tabela 1 – Principais variáveis do RRT

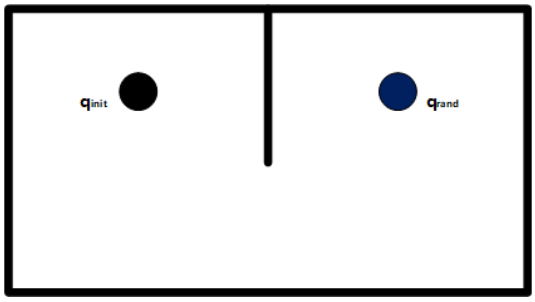
Variável	Função
q_{ini}	Estado inicial da raiz da árvore
q_{rand}	Estado amostrado
q_{near}	Estado do nó mais próximo do amostrado
q_{new}	Estado do nó que será inserido na árvore
q_{fim}	Estado final ou ponto objetivo

O caminho é encontrado analisando os nós pertencente à árvore ao final da expansão. Cabe ressaltar que um nó pode ser indicado como parente de vários outros nós, entretanto, apenas um nó poderá ser indicado como parente de outro nó.

O algoritmo inicializa definindo que o estado do nó raiz é q_{ini} com parente ele mesmo cujo o custo para atingir o ponto inicial deverá ser zero. A partir disso, começa a fase de expansão por meio das amostragens de estados (q_{rand}) conforme apresentado pela Figura 16.

A partir da configuração amostrada verifica-se qual dos nós presentes na árvore é o mais próximo do estado amostrado de acordo com a métrica do problema (distância euclidiana), esse nó, terá seu estado denominado como q_{near}. No exemplo apresentado

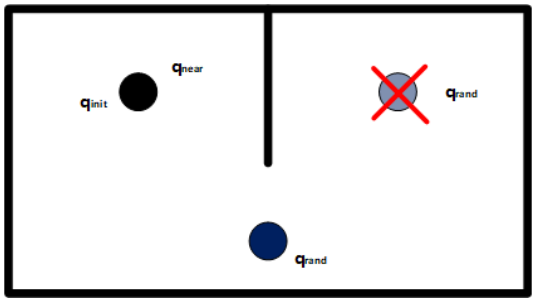
Figura 16 – Inicio do algoritmo RRT.



Fonte: Elaborado pelo Autor

pela Figura 16 existe apenas o nó inicial contido na árvore, nesse sentido, q_{init} deverá ser q_{near}. Entretanto, deve-se verificar se é possível ligar q_{near} a q_{rand} por uma linha reta sem que ocorra colisões com os obstáculos definidos por C_{obs}. Nesse ponto, pode-se perceber a característica *offline* do planejador, já que o conjunto de obstáculos é pré-definido no início em um mapa. Caso exista um obstáculo entre os pontos q_{near} e q_{rand} uma nova iteração começa amostrando um novo q_{rand} (Figura 17).

Figura 17 – Amostragens do algoritmo RRT.

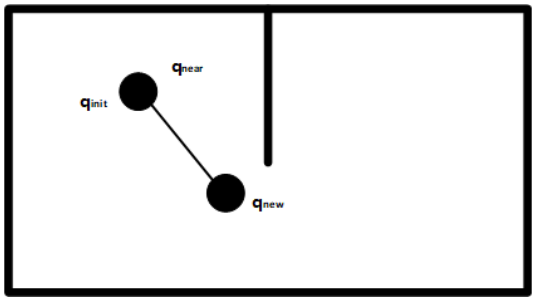


Fonte: Elaborado pelo Autor

Na Figura 17 percebe-se a ausência de obstáculos para o novo q_{rand} amostrado, então, o modelo cinemático do robô em que está se planejando a missão é executado. Será simulado um deslocamento fictício da seguinte maneira: o ponto de partida é q_{near}, o robô seguirá na direção de q_{rand} com unidades de controle u por um período de tempo Δt. Adota-se que estado q_{near} esteja com posição angular apontado para q_{rand}. De maneira proposital, o valor de Δt não é alto o suficiente para que o robô ultrapasse q_{rand} conforme o exemplo apresentado pela Figura 18. A esse novo estado foi dado o nome de q_{new} e é verificado novamente se há obstáculos entre q_{new} e q_{near}, caso não haja pode-se incorporar um novo nó à arvore cujo estado será q_{new}.

O novo nó terá o estado definido por q_{new} cujo parente é q_{near}. O custo para chegar até a posição objetivo é a soma do custo do nó que possui q_{near} como configuração com a distância euclidiana de q_{near} a q_{new}. O final da fase de expansão consiste em encontrar

Figura 18 – Obtendo o novo nó



Fonte: Elaborado pelo Autor

um q_{new} possível de ser incorporado à árvore cuja a distância euclidiana de seu estado até que q_{fim} seja menor ou igual à um valor ϕ pré-definido sem que exista obstáculos entre os mesmos. Um outro modo seria definir o término do algoritmo através de saturação, ou seja, quando o número de iterações for superior a uma constante K. A Figura 19 apresenta um exemplo complexo de expansão do algoritmo RRT em 4 fases, cujo o valor máximo de iterações foi de $K = 5000$, entretanto, q_{fim} pôde ser encontrado a partir da distância euclidiana.

A segunda fase inicializa adicionando o nó referente ao objetivo na árvore, cujo parentesco é o nó que possui o estado mais próximo ao da posição objetivo. O retrocesso que parte do nó objetivo analisará os parentes definidos na fase de expansão. Desta maneira, considerando a disposição dos nós e seus parentescos definidos, o caminho encontrado será o menor dentre os possíveis obtidos com a árvore na fase de expansão. O caminho para o exemplo da Figura 19 é mostrado pela Figura 20 também em 4 fases. A ligação dos nós que estão sob a linha vermelha formará o caminho que liga q_{ini} a q_{fim}. Vale ressaltar que o caminho encontrado é um conjunto dos estados formados pelos nós e a síntese das duas fases do planejador estão descritos no Algoritmo 3. Um resumo da utilidade que cada uma das funções desempenham estão presentes no apêndice A.

O algoritmo RRT pode ser considerado um método de consulta única, ou seja, a árvore gerada na fase de expansão só é válida para o mapa definido no início do algoritmo. Caso ocorra mudanças, uma nova árvore deverá ser elaborada com o mapa novo. Essa seria uma desvantagem do método RRT em relação ao algoritmo PRM que são considerados métodos multi-consulta [25].

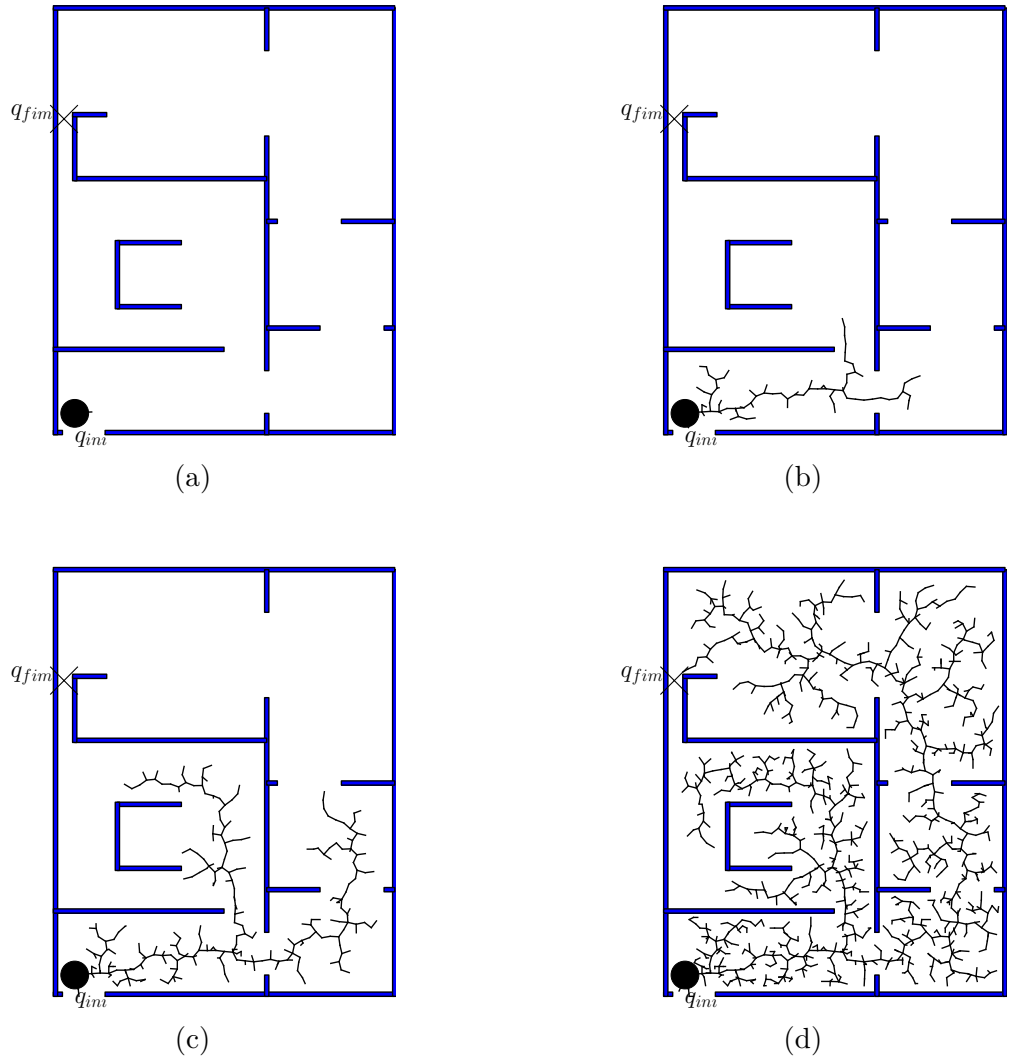
2.7.2.1 RRT*

O algoritmo clássico do RRT apresentado na sessão anterior pode gerar um caminho, porém, a qualidade dos mesmos pode ser considerada um ponto negativo do método. O precursor do RRT afirmou em um trabalho [88] seguinte, que é possível aumentar o sucesso do método a partir de heurísticas acopladas ao algoritmo bem como ajustes dos parâmetros

Algorithm 3 Algoritmo *Rapidly Exploring Random Tree (RRT).*

função PLANJEADORRRT($q_{ini}, q_{fim}, K, u, \Delta t, \phi$)
  função EXANDERRT($q_{ini}, q_{fim}, K, u, \Delta t$) ▷ Primeira Fase
    $No_{ini}.Estado \leftarrow q_{ini}$
    $No_{ini}.Parente \leftarrow 0$
    $No_{ini}.Custo \leftarrow 0$
    $RRT.AddNo(No_{ini})$ ▷ Adiciona à árvore o nó inicial
    $k = 1$
    enquanto $k < K$ *ou* $D < \phi$ **faça**
      $q_{rand} \leftarrow Amostra\_Estado()$
      $No_{near} \leftarrow Busca\_No\_Mais\_Proximo(q_{rand}, RRT)$
      se $Nao\_Ha\_Obst(q_{rand}, No_{near}.Estado)$ **então**
        $q_{new} \leftarrow Usa\_Modelo\_Cinematico(No_{near}.Estado, u, \Delta t)$
        se $Nao\_Ha\_Obst(No_{near}.Estado, q_{new})$ **então**
          $custo \leftarrow Obtem\_Distancia(q_{new}, No_{near}.Estado) + No_{near}.custo$
          $No_{new}.Estado \leftarrow q_{new}$
          $No_{new}.Parente \leftarrow No_{near}$
          $No_{new}.Custo \leftarrow custo$
          $RRT.AddNo(No_{near})$
          $D \leftarrow Obtem\_Distancia(q_{fim}, q_{new}, RRT)$
        fim se
      fim se
      k=k+1
    fim enquanto
    retorna RRT
  fim função
  função GERACAMINHORRT(q_{fim}, RRT) ▷ Segunda Fase
    $No_{near} \leftarrow Busca\_No\_Mais\_Proximo(q_{fim}, RRT)$ ▷ Parente do nó final
    $custo \leftarrow Obtem\_Distancia(q_{fim}, No_{near}.Estado) + No_{near}.custo$
    $No_{fim}.Estado \leftarrow q_{fim}$
    $No_{fim}.Parente \leftarrow No_{near}$
    $No_{fim}.Custo \leftarrow custo$
    $RRT.AddNo(No_{fim})$ ▷ Adiciona à árvore o nó final
    $No \leftarrow RRT(Ultimo\_No\_Inserido)$
    $G.AdicionaEstado(No.Estado)$
    enquanto $No.Parents \neq 0$ **faça**
      $P \leftarrow No.Parente$
      $No \leftarrow RRT(No.Parente = P)$
      $G.AdicionaEstado(No.Estado)$
    fim enquanto
    retorna G ▷ Conjunto de estados
  fim função
fim função

Figura 19 – Comportamento do algoritmo RRT em 4 estágios: (a) Fase 1 (b) Fase 2 (c) Fase 3 (d) Fase 4.

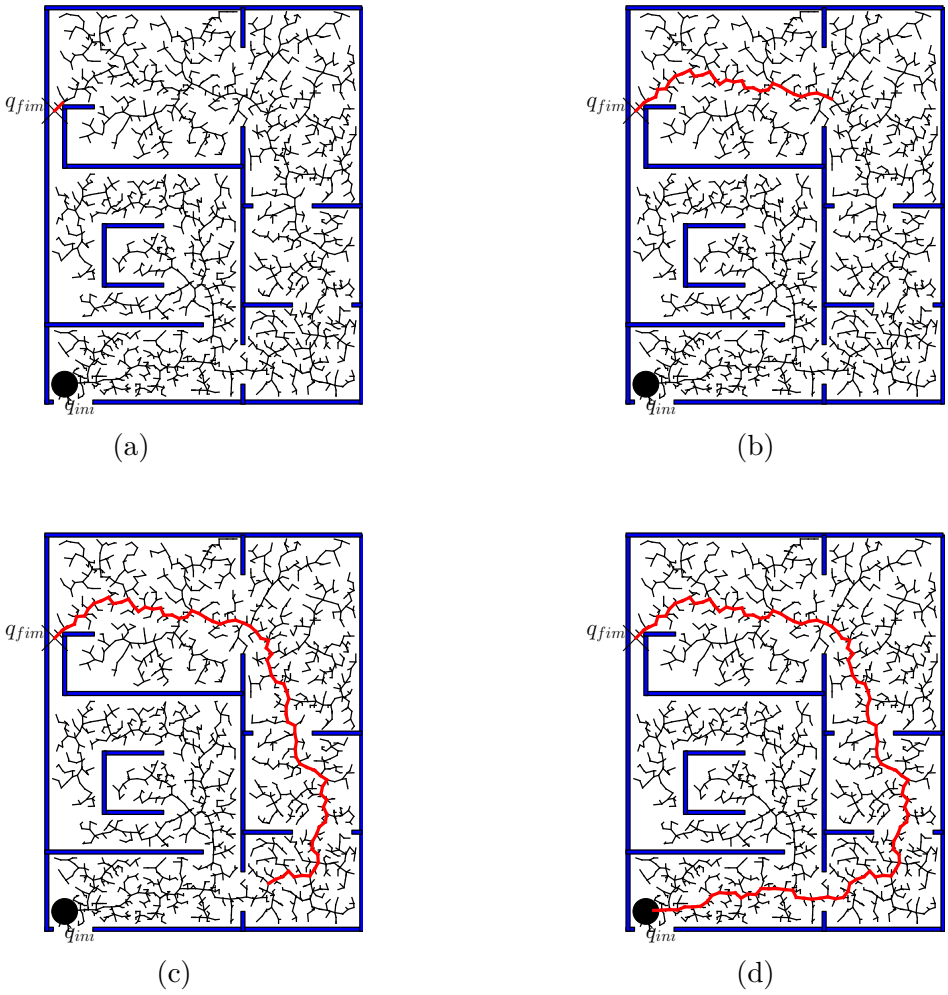


básicos tornando peça fundamental do planejador. Sendo assim, em 2011 Sertac Karaman publicou trabalho [46] onde o caminho obtido tenderia ao ótimo. Mesmo havendo as chances do RRT encontrar um caminho ótimo, os autores mostraram que essas hipóteses são praticamente zero de ocorrerem.

O novo algoritmo ficou conhecido como RRT*, e foi implementado a partir da detecação de um problema na etapa em que é selecionado qual o nó deverá ser o parente da configuração q_{new}. De acordo com [44] o RRT* utiliza da rapidez do RRT e do fato de ser probabilisticamente completo para garantir que um caminho ótimo seja obtido de maneira assintótica. Além dessas características, o planejador mantém a expansão da árvore baseada nas restrições cinemáticas que o modelo do robô possui.

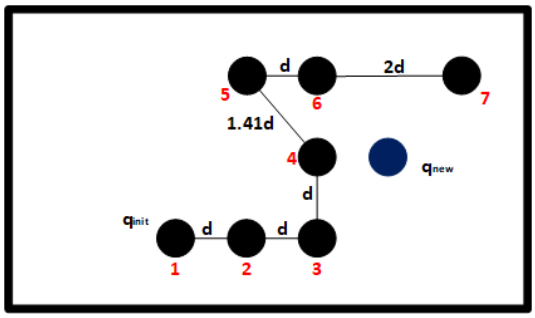
Inicialmente deve-se mostrar o problema no método do RRT a partir do exemplo apresentado pela Figura 21. Os círculos pretos são as configurações dos nós que já estão contidos na árvore e todas as arestas que os ligam possuem tamanho proporcionais a d

Figura 20 – Comportamento do caminho obtido na RRT em 4 Estágios: (a) Estágio 1 (b) Estágio 2 (c) Estágio 3 (d) Estágio 4



para facilitar o entendimento. O estado novo q_{new}, que será incorporado a árvore também possui o mesmo tamanho de aresta e seu parente é o nó **4** (de acordo com o método RRT).

Figura 21 – Expansão da árvore via RRT

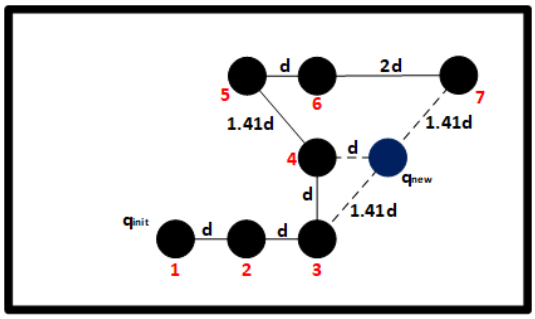


Fonte: Elaborado pelo Autor

Considerando a situação do método base RRT, q_{new} seria ligado ao nó **4** com um custo de d para o nó **4**, o nó 3 possuiria um custo maior no valor de $1,41d$. Contudo, uma

simples análise de custos utilizando a distância euclidiana percebe-se a existência de um custo no valor de $3,41d$ ligando q_{new} ao nó **1** conforme apresentado pela Figura 22. Se o RRT clássico fosse utilizado, q_{new} seria ligado ao nó **4**, e consequentemente, haveria um custo de $4d$ para atingir a posição objetivo. Nesse sentido conclui-se que é mais vantajoso ligar o novo nó à **3** do que **4**.

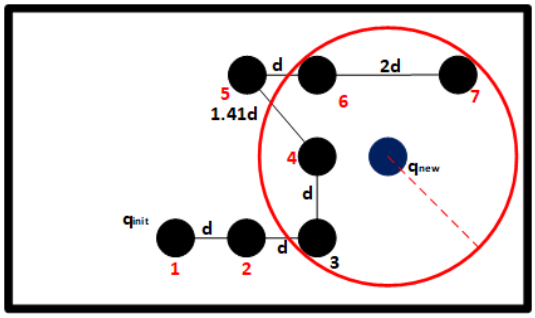
Figura 22 – Custos da expansão



Fonte: Elaborado pelo Autor

Tendo em vista o problema supracitado, uma nova heurística foi proposta com o objetivo de encontrar caminhos menos custosos. Essa heurística visa restabelecer como os parentes são definidos na inserção do nó à árvore. Em um primeiro momento seleciona-se os nós que estão inseridos em um círculo de raio r com o centro no estado q_{new} conforme a Figura 23.

Figura 23 – Escolha do Nó Parente

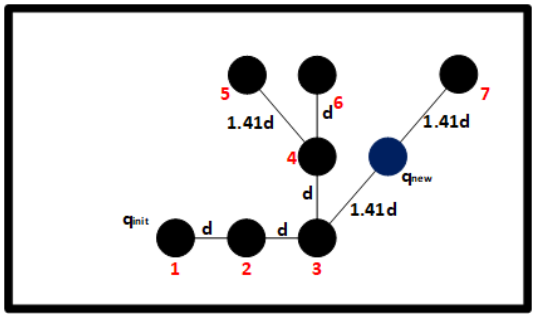


Fonte: Elaborado pelo Autor

Dentre os nós selecionados pelo círculo, busca-se aquele com menor custo que ligue a posição objetivo ao estado q_{new}, para o exemplo acima, essa análise definiria o nó **3** como parente q_{new}, ou seja, **3** passaria a ser q_{near}. Para que as restrições de movimento continuem valendo o modelo cinemático é aplicado partindo do novo estado q_{near} em direção à q_{new}. Essa ação deve ser considerada, pois, a conexão entre o novo q_{near} e q_{new} pode não atender as restrições cinemáticas.

Além dessa característica, o RRT* propõe a reconexão de todos os nós presentes no círculo de seleção baseado no parentesco. Para cada reconexão haverá o uso do modelo cinemático para manter a expansão respeitando as restrições. Essa ação é importante, já que novos caminhos, com menores custos podem ser formados passando pela nova ligação de q_{new}. Para o exemplo analisado, o nó **7** poderá receber como parente q_{new} com custos menores. Como todos os nós são analisados, o **6**, também receberá um novo parentesco, ao invés de se conectar ao **5**, passará a ser ligado ao nó **4** com menores custos. A Figura 24 representa os resultados de uma religação.

Figura 24 – Escolha do Nó Parente



Fonte: Elaborado pelo Autor

A modificação do algoritmo base do RRT (Algoritmo 3) é realizada inserindo uma função abaixo da linha 31, onde a mesma definirá o parentesco através da função *Escolhe_Parente_e_Religa* que é representada pelo Algoritmo 4. Com o aumento dos estados amostrados, a árvore seria sempre atualizada ao longo do tempo convergindo para a solução ótima de maneira assintótica [89]. Entretanto, o obtenção de um caminho ótimo esta condicionado ao aumento do custo computacional, já que a busca pelos vizinhos, o cálculo de custos e a religação da árvore envolvem um alto peso no processamento [89].

2.7.2.2 Análise na Dispersão dos Nós na RRT (Método DRRT)

O gasto de memória dos algoritmos baseados na RRT é proporcional à quantidade de nós inseridos na árvore. O algoritmo RRT* propõe uma solução para o problema de caminhos não ótimos apresentados pelo RRT, entretanto, o custo computacional atrelado a esse novo algoritmo pode ser considerado um problema na concepção de trajetórias, já que quanto maior o número de nós, mais esforço computacional o RRT* terá.

Em [49] os autores demonstraram o problema da dispersão dos nós inseridos em uma árvore, onde porções pequenas do espaço contavam com uma grande quantidade de nós conforme mostrado na Figura 25.

A Figura 25 conta com 5845 nós e na porção destacada, 37 nós em uma área de apenas 1 m^2 obtidos ao longo de 8000 iterações. Esse fato torna as informações dos nós que

Algorithm 4 Função Escolhe Parente e Religa Árvore.

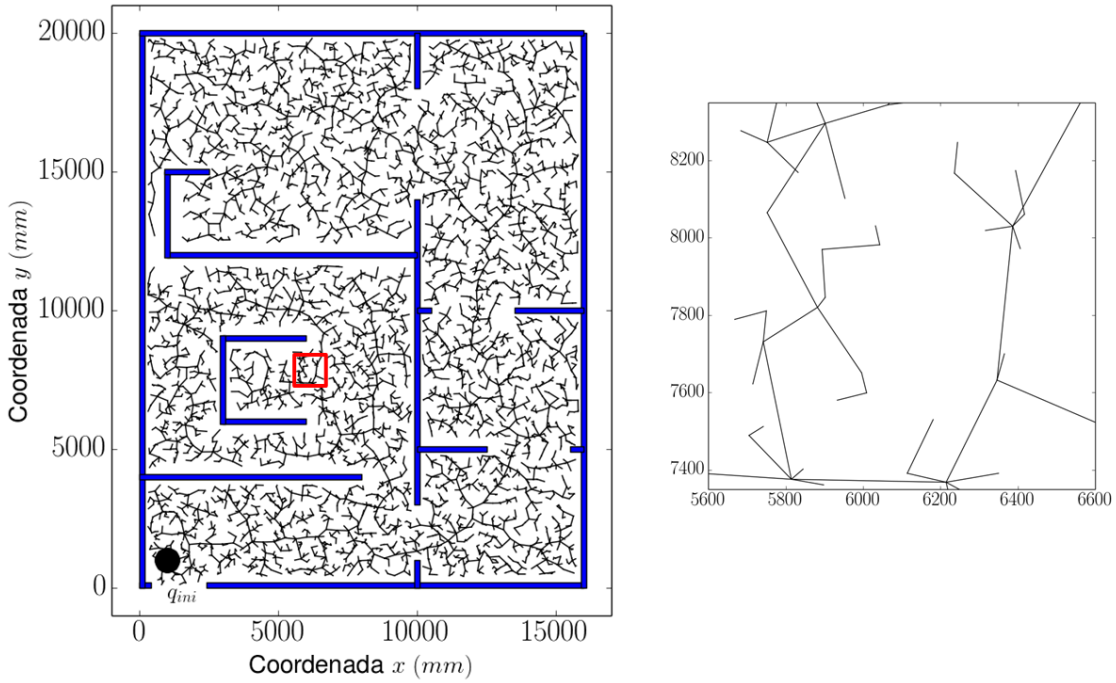
```
1: função ESCOLHE_PARENTE_E_RELIGA(q_new, RRT, r, No)
2:     Z ← Busca_Nos_Proximos(RRT, q_new, r)
3:     para z ∈ Z faça
4:         q_n ← Usa_Modelo_Cinematico(z.Estado, u, Δt)
5:         se Nao_Ha_Obst(q_n, z.Estado) então
6:             custo ← Obtem_Distancia(q_n, z.Estado) + z.custo
7:             se custo < No.Custo então
8:                 No.Custo ← custo
9:                 No.Parente ← z
10:            fim se
11:        fim se
12:    fim para
13:    return No
14:    função RELIGA_ARVORE(Z, RRT, No)
15:        para z ∈ Z \ No faça
16:            q_n ← Usa_Modelo_Cinematico(z.Estado, u, Δt)
17:            custo ← Obtem_Distancia(q_n, z.Estado) + z.custo
18:            se Nao_Ha_Obst(q_n, z.Estado) e No.Custo + custo < z.Custo então
19:                RRT ← Reconecta(No, z, RRT)
20:            fim se
21:        fim para
22:    fim função
23: fim função
```

estão próximos uns aos outros redundantes, ou seja, representam gastos computacionais sem nenhum ganho para o planejador.

Para contornar esse problema foi proposta uma heurística no sorteio das novas amostras em [49], cuja finalidade era evitar que estados próximos aos nós já existentes na árvore não fossem inseridos. A ideia básica era inserir uma zona segura circular de raio R ao redor do estado de um novo nó, essa região foi denominada de zona de descarte. No momento da amostragem de um estado, é verificado se o mesmo esta dentro da área de descarte. Se essa situação correr, haverá o descarte da configuração tal qual os estados amostrados que geram colisões com obstáculos, caso contrário, o método prosseguiria da maneira normal. Quando os novos nós são adicionados à árvore as regiões vão se unindo formando áreas de restrições ainda maior tal qual a Figura 26. A esse método foi dado o nome de DRRT devido a análise na dispersão. Além desse tratamento nos nós, os autores concluíram que o método é favorável a encontrar caminhos que passem por passagens estreitas de maneira mais rápida quando comparados ao RRT.

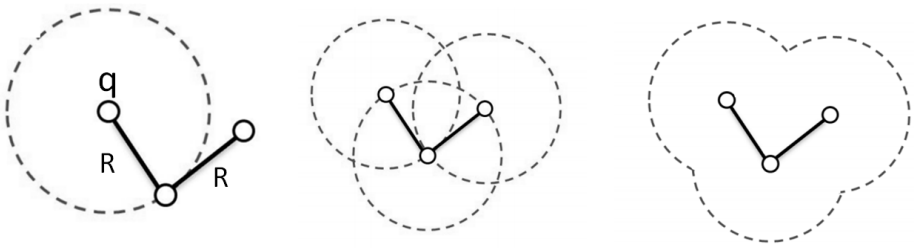
O método DRRT também dividiu o mapa do ambiente em grades de ocupação para análise da distribuição da árvore no ambiente. Uma matriz seria responsável por representar esse mapa e indicaria se a grade estaria ocupada ou em cima de um obstáculo, e livre. Os resultados mostraram que o número de nós necessários para ocupar um

Figura 25 – Análise da Dispersão dos Nós.



Fonte: Elaborado pelo Autor

Figura 26 – Região Segura ao Redor dos Estados.



Fonte: Retirado de [25].

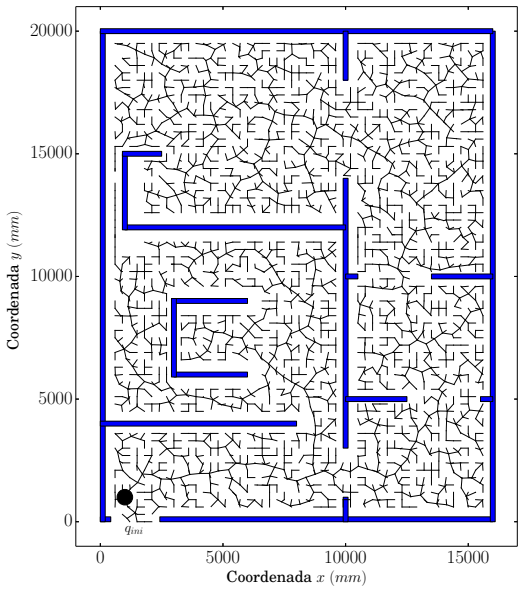
ambiente seria menor enquanto que o número de iterações aumentaria, pois haveria um maior número de descartes. Entretanto, a DRRT define que o estado de um novo nó adicionado terá uma distância fixa ao seu nó parente (devido a circunferência), esse fato pode apresentar incompatibilidade na expansão da árvore quando for necessário utilizar modelos cinemáticos mais rígidos, pois o crescimento nem sempre poderá ser fixo. Além disso, a verificação se um nó está inserido ou não na região de descarte pode aumentar o tempo computacional [25], ao passo que a memória gasta será menor.

Dado os problemas optou-se por utilizar somente as grades de ocupação, ou seja, as zonas de descartes seriam as próprias grades de ocupação. Quando um novo nó fosse adicionado, sua posição na matriz que representa o ambiente seria modificado e na próxima

iteração essa configuração não seria possível de ser sorteada. O uso das grades de ocupação favorece na busca por regiões livres, pois por uma consulta rápida à matriz é possível extrair quais as regiões passíveis de serem sorteadas além de compensar o tamanho do robô já que o mesmo é tratado como um ponto. Dessa maneira, o número de iterações também tende a ser menor para obter um caminho. De acordo com [90] as grades de ocupação são mais indicadas para os planejadores *offline*, pois, um mapa dividido assim é fácil de construir e de se atualizar [91].

As grades utilizadas neste trabalho possuem tamanhos fixos de $30cm$. Para as mesmas 8000 iterações obteve um número de 2922 nós que podem ser vistos na Figura 27. É possível perceber que os nós espalharam-se pelo mapa de maneira homogêneas se comparado ao RRT clássico mesmo com uma menor quantidade de nós.

Figura 27 – Expansão de uma árvore baseada no método DRRT.



Fonte: Elaborado pelo Autor

2.7.3 **Campos Potenciais Artificiais (CPA)**

Campos potenciais artificiais é um método originado na mecânica dos fluídos [92] e importado para a robótica por Khatib em 1978 [93] com o propósito no planejamento de movimentos em manipuladores robóticos. O método foi proposto para o desvio de obstáculos dinâmicos e continuou sendo explorado e aprimorado pelo autor em outros trabalhos [94], [95] e [96], sendo que este último é o trabalho em que foi consolidado o método campos potenciais artificiais (do inglês, *Artificials Potential Fields*).

O robô deverá ser guiado até a posição objetivo através de um campo potencial formado por funções potenciais. Para o método, não é necessário a definição de um mapa prévio, já que seu comportamento é reativo de acordo com a leitura realizada pelos sensores do ambiente. Bem difundido na comunidade acadêmica, estudos em robótica envolvendo esse método, podem ser observados em diversos trabalhos recentes através de vários tipos de robôs: móveis [52], aéreos [97] e móveis aquáticos [98]. Além desses, é possível encontrar aplicações dos conceitos na robótica assistiva, pode-se citar o trabalho [99] que controla uma cadeira de rodas de maneira compartilhada com o usuário. De acordo com [100] o planejamento é realizado de maneira local apenas com as informações retiradas do ambiente naquele instante de tempo e então gera os sinais de controle do robô para desviar dos obstáculos, desta maneira o método se comporta como planejador e um controlador aumentando os motivos para seu uso.

A função potencial deve ser diferenciável, real e é definida como $U(q) : U^2 \rightarrow U$ em que q é um estado que pertence ao conjunto de configurações C. Comumente faz-se a analogia com partículas elétricas, em que o robô se comporta como carga positiva, os obstáculos são considerados também partículas positivas para que exista a repulsão e o robô não venha colidir. Por outro lado, o estado final ($q_{fim} = (x_f, y_f)$) assume o papel de carga negativa para que aja atração mútua entre robô e objetivo. O único objeto que pode se mover no ambiente é o robô, dessa maneira, as interações entre obstáculos e posições objetivos não geram nenhum tipo de movimento. A função potencial resultante corresponde a soma das parcelas de repulsão e atração conforme apresentado pela Equação 2.44:

$$U(q)_{tot} = U_{att} + U_{rep} \tag{2.44}$$

e a ilustração que modela esse problema é feita pela Figura 28.

Figura 28 – Ilustração Campos Potenciais Artificiais.

Fonte: Elaborado pelo Autor

Onde U_{att} e U_{rep} correspondem as funções de potenciais atrativos e repulsivos respectivamente. A força que deverá ser aplicada ao robô para que ele execute os movimentos, pode ser dada de acordo com o gradiente da função potencial, ou seja, de maneira análoga,

a função potencial seria a energia e seu gradiente a força [85]. Matematicamente o conceito de gradiente consiste na generalização da derivada para as funções multivariáveis e seu vetor denota qual é a direção de máximo crescimento da função conforme a Equação 2.45:

$$\nabla U(q) = \left[\frac{\partial U}{\partial q_x}(q),\ \frac{\partial U}{\partial q_y}(q)\right]^T \tag{2.45}$$

Caso seja considerado um sistema de primeira ordem, ignorando a dinâmica, os gradientes podem ser compreendidos como um vetor de velocidade [85]. Na função potencial o robô deverá seguir do máximo para o mínimo, neste sentido, necessita-se adotar o gradiente descendente, que é o vetor de velocidades que aponta para a região de mínimo da função U conforme apresentado na Equação 2.46:

$$-\nabla U(q) = \left[-\frac{\partial U}{\partial q_x}(q),\ -\frac{\partial U}{\partial q_y}(q)]\right]^T \tag{2.46}$$

O robô cessará os movimentos no momento em que $-\nabla U(q^*) = 0$, sendo que q^* é um estado nomeado de ponto crítico de U. De acordo com [85], além do ponto de mínimo (objetivo), q^* pode corresponder ao máximo ou ponto de sela da função potencial. Entretanto, a lei de controle que é imposta ao robô através do gradiente descendente não guiará o robô a um ponto de sela ou ao máximo dessa função, esse fato só ocorreria se o ponto inicial fosse o próprio máximo ou ponto de sela. Porém, esses pontos são instáveis o que implicaria que qualquer perturbação levaria o robô a se movimentar levando-o em direção à posição objetivo.

São diversas funções que podem ser consideradas possíveis funções potenciais para o CPA, pois, atendendo à alguns requisitos, elas poderão ser implementadas no algoritmo. Nas seções 2.7.3.1 e 2.7.3.2 são apresentadas esses requisitos, seus comportamentos e o motivo de suas escolhas.

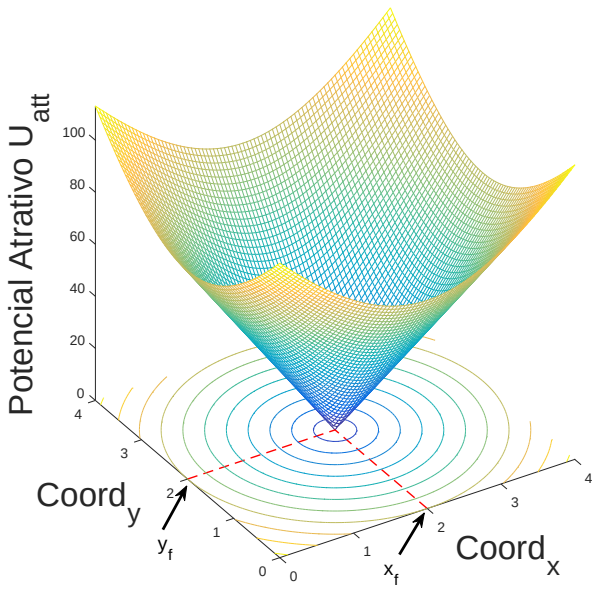
2.7.3.1 Potencial Atrativo

A primeira característica que o potencial atrativo deve apresentar é um comportamento monotonicamente crescente de acordo com a distância até o objetivo, ou seja, a função deve manter o padrão de crescimento a medida que a distância aumenta. Outro ponto importante é que $U_{att}(q)$ deve ser maior que zero para todo $q \in C$ e quando $q = q_{fim}$ $U_{att} = 0$. Assim, defini-se a função mais simples denominada potencial cônico conforme a Equação 2.47

$$U_{att}(x, y) = \xi\sqrt{(x - x_f)^2 + (y - y_f)^2} \tag{2.47}$$

cuja curva pode ser vista na Figura 29. Onde a variável ξ corresponde a uma constante que controla o crescimento do campo atrativo. Entretanto, quando o gradiente descendente da função relativa ao potencial cônico fosse calculado, pode-se perceber que a função não é definida no ponto objetivo conforme visto na Equação 2.48, já que nesse ponto o denominador será 0, a esse problema foi dado o nome de *chattering*.

Figura 29 – Potencial Atrativo Cônico.



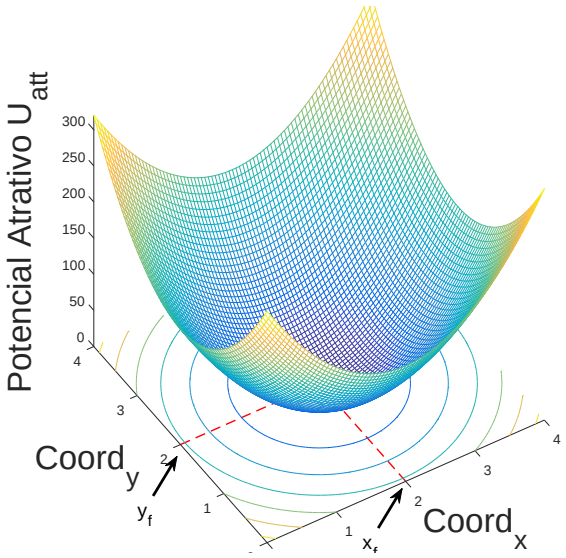
Fonte: Elaborado pelo Autor

$$-\nabla U_{att}(x,y) = \frac{-\xi}{\sqrt{(x-x_f)^2+(y-y_f)^2}}[x-x_f, y-y_f]^T \tag{2.48}$$

Nesse caso, percebe-se outra característica que a função deve apresentar, ser diferenciável em todos os estados contidos no espaço de configurações. Uma saída para esse problema seria utilizar a função quadrática, que se comporta de forma similar a cônica. A mesma diminui a magnitude do potencial de atração a medida que a distância até o objetivo também diminui, porém, de maneira quadrática, cuja Equação é apresentada em 2.49 e a curva na Figura em 30.

$$U_{att}(x,y) = \frac{1}{2}\xi(\sqrt{(x-x_f)^2+(y-y_f)^2})^2 = \frac{1}{2}\xi((x-x_f)^2+(y-y_f)^2) \tag{2.49}$$

Figura 30 – Potencial Atrativo Quadrático.



Fonte: Elaborado pelo Autor

Esse potencial possui a derivada descrita pela Equação 2.50 que mostra a possibilidade de diferenciação em todos os estados pertencentes ao espaço de configuração.

$$-\nabla U_{att}(x,y) = -\xi\left[x - x_f, y - y_f\right]^T \tag{2.50}$$

Contudo, o potencial atrativo quadrático pode apresentar um problema para o controlador quando o estado inicial (q_{ini}) estiver muito longe do ponto objetivo, pois, a velocidade que será imposta ao robô poderá ser alta, causando uma variação abrupta no controlador devido ao seu aspecto quadrático. A solução apresentada por [85] para esse entrave, é mesclar o potencial cônico com o quadrático, ou seja, acima de uma distância limítrofe (d^*_{fim}) utiliza-se o potencial cônico, caso contrário faz-se o uso do quadrático. As Equações 2.51 e 2.52 correspondem a essa função híbrida e sua derivada respectivamente:

$$U_{att}(x,y) = \begin{cases} \frac{1}{2}\xi((x-x_f)^2 + (y-y_f)^2), & d([x,y],[x_f,y_f]) \leq d^*_{fim} \\ d^*_{fim}\xi\sqrt{(x-x_f)^2 + (y-y_f)^2} - \frac{\xi}{2}(d^*_{fim})^2, & d([x,y],[x_f,y_f]) > d^*_{fim} \end{cases} \tag{2.51}$$

$$-\nabla U_{att}(x,y) = \begin{cases} -\xi\left[x - x_f, y - y_f\right]^T, & d([x,y],[x_f,y_f]) \leq d^*_{fim} \\ \dfrac{-d^*_{fim}\xi\left[x - x_f, y - y_f\right]^T}{d([x,y],[x_f,y_f])}, & d([x,y],[x_f,y_f]) > d^*_{fim} \end{cases} \tag{2.52}$$

onde $d([x,y],[x_f,y_f]) = \sqrt{(x-x_f)^2+(y-y_f)^2}$, ou seja, é a função que calcula a distância entre dois estados. Os termos adicionais presente na Equação 2.51 são constantes e tem por objetivo manter a continuidade do potencial cônico.

2.7.3.2 Potencial Repulsivo

O potencial Repulsivo atua quando obstáculos forem detectados pelos sensores incorporados ao robô. Seu comportamento deve ser monotonicamente crescente a medida que a distância para um objeto pertencente à C_{obs} e estado $[x_o, y_o]$ diminua. Além dessas características nesse potencial há uma constante ϵ que define uma região ao redor do robô onde serão considerados os obstáculos, denominado horizonte de eventos, sendo assim, somente os que estiverem dentro dessa região entrarão nos cálculos da função potencial repulsivo, caso contrário esse potencial será considerado nulo conforme a Equação 2.53.

$$U_{rep,i}(x,y) = \begin{cases} \dfrac{\eta}{2}\left(\dfrac{1}{d([x,y],[x_o,y_o]_i)} - \dfrac{1}{\epsilon}\right), & d([x,y],[x_o,y_o]) \leq \epsilon \\ 0, & d([x,y],[x_f,y_f]) > \epsilon \end{cases} \tag{2.53}$$

onde η corresponde à constante relacionada ao ganho do campo repulsivo. O potencial apresentado pela Equação 2.53 corresponde ao *i-ésimo* obstáculo, desta maneira, o potencial repulsivo total é a soma de todos os n potenciais detectados pelos sensores do robô conforme a Equação 2.54:

$$U_{rep}(x,y) = \sum_{i=0}^{n} U_{rep,i}(x,y) \tag{2.54}$$

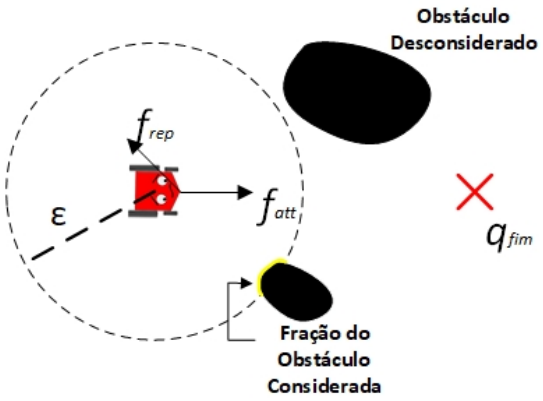
O horizonte de eventos serve para impedir que obstáculos distantes do robô e dentro do raio de visão dos sensores contribuam para o cálculo da força repulsiva, seu comportamento pode ser ilustrado pela Figura 31. Os sensores responsáveis por mapear localmente os obstáculos que se encontram ao redor do robô é o dispositivo *Asus Xtion* e os sonares do P3DX.

Deve-se obter a direção no qual o potencial repulsivo decresce, nesse sentido, utiliza-se também o gradiente descendente (Equação 2.55) para obter esse vetor.

$$-\nabla U_{rep,i}(x,y) = \begin{cases} -\eta\left(\dfrac{1}{\epsilon} - \dfrac{1}{e_i}\right)\dfrac{1}{e_i^3}[x-x_{o,i}, y-y_{o,i}]^T, & d([x,y],[x_o,y_o]) \leq \epsilon \\ 0, & d([x,y],[x_f,y_f]) > \epsilon \end{cases} \tag{2.55}$$

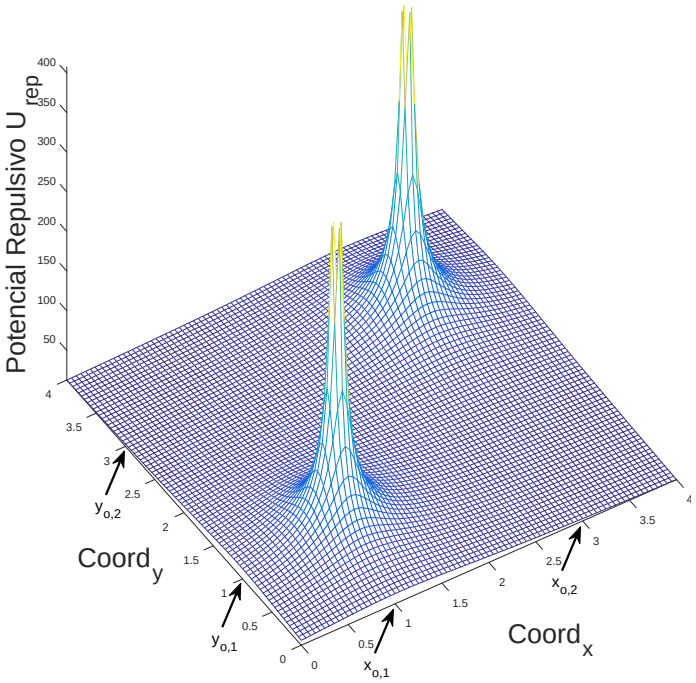
onde $\epsilon_i = d([x,y],[x_o,y_o]_i)$. Como exemplo, têm-se a Figura 32 representando o potencial repulsivo de 2 objetos ($n = 2$) nos estados $o_1 = [x_{o1}, y_{o1}]$ e $o_2 = [x_{o2}, y_{o2}]$.

Figura 31 – Horizonte de Eventos no Potencial Repulsivo.



Fonte: Elaborado pelo Autor

Figura 32 – Potencial Repulsivo.



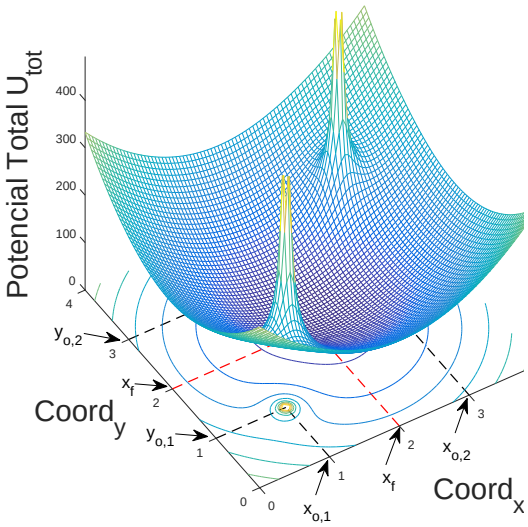
Fonte: Elaborado pelo Autor

2.7.3.3 Potencial Total

O potencial total que deverá atuar no robô é dado pela soma da parte atrativa com a repulsiva conforme a Equação 2.56 e é representado pela Figura 33.

$$U_{tot} = U_{att} + U_{rep} \tag{2.56}$$

Figura 33 – Potencial Total.



Fonte: Elaborado pelo Autor

2.7.3.4 Controlador dos Campos Potenciais Artificiais

Além de planejar trajetórias, o método dos campos potenciais artificiais pode ser considerado um controlador que atua nas velocidades linear (v) e angular do robô (ω). Uma força proporcional ao potencial total é definida para que o sistema robótico se movimente. O vetor que corresponde à força atrativa e repulsiva estão nas Equações 2.57 2.58:

$$F_{att}(x, y) = -\nabla U_{att}(x, y) \tag{2.57}$$

$$F_{rep}(x, y) = -\nabla U_{rep}(x, y) \tag{2.58}$$

A força total (F_{tot}) é dada pela soma das Equações 2.57 e 2.58. O controlador do robô P3DX deste trabalho permite que seja enviado v e ω diretamente para robô, já que esse pode ajustar de maneira autônoma as velocidades em cada uma das rodas. Sabendo

que o gradiente descendente da função potencial total tende a apontar para o objetivo e para o lado oposto aos obstáculos, [101] utiliza as velocidades linear e angular baseadas em CPA conforme as Equações 2.59 e 2.60:

$$v = ||F_{tot}|| \tag{2.59}$$

$$\omega = K_\omega \arctan2(F_{tot}^{\{y\}}, F_{tot}^{\{x\}}) - \theta_R \tag{2.60}$$

onde K_ω corresponde ao ganho responsável por ajustar o movimento angular do robô para que o mesmo não fique oscilando durante a movimentação e a variável θ_R representa à orientação atual do robô. A definição da função arctan2() é definida na Equação 2.61:

$$\arctan2(F_{tot}^{\{y\}}, F_{tot}^{\{x\}}) = \begin{cases} \arctan(\frac{F_{tot}^{\{y\}}}{F_{tot}^{\{x\}}}), & F_{tot}^{\{x\}} \geq 0 \\ \frac{\pi}{2} - \arctan(\frac{F_{tot}^{\{x\}}}{F_{tot}^{\{y\}}}), & F_{tot}^{\{y\}} \geq 0 \\ -\frac{\pi}{2} - \arctan(\frac{F_{tot}^{\{x\}}}{F_{tot}^{\{y\}}}), & F_{tot}^{\{y\}} \leq 0 \\ \arctan(\frac{F_{tot}^{\{y\}}}{F_{tot}^{\{x\}}}) \pm \pi, & F_{tot}^{\{x\}} \leq 0 \\ indefinido, & F_{tot}^{\{x\}} = 0 \ e \ F_{tot}^{\{y\}} = 0 \end{cases} \tag{2.61}$$

2.7.3.5 Algoritmo dos Campos Potenciais Artificiais

Todo processo demonstrado para o método dos campos Potenciais Artificiais pode ser apresentado conforme o Algoritmo 5. Para inicializar, o CPA necessita de uma posição objetivo definido por q_{fim}, as constantes que se encontram nas funções potenciais K_{att}, K_{rep}, K_ω e a distância mínima (δ), que o robô deve se encontrar do objetivo para que o algoritmo seja finalizado.

No Apêndice A há um resumo das funcionalidades de cada uma das funções desse algoritmo.

2.7.3.6 Problema dos Campos Potenciais Artificiais

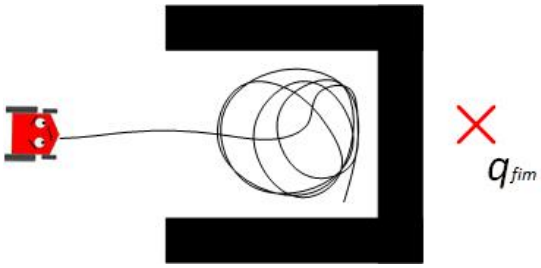
O planejamento *online* baseado nos CPA possui problemas relacionados a mínimos locais que podem inviabilizar a chegada do robô até o objetivo [85]. O gradiente descendente apontará para o mínimo normalmente, entretanto, obstáculos convexos que se situam próximos aos objetivos podem deixar o robô preso dentro desses bloqueios conforme apresentado pela Figura 34.

A força repulsiva gerada pelos obstáculos ao redor do robô cresce, ficando maior que a atrativa e faz com que o mesmo realize um movimento na direção oposta ao objetivo,

Algorithm 5 Algoritmo Campos Potenciais Artificiais (CPA).

função CPA($\delta, K_{att}, K_{rep}, K_{\omega}, q_{fim}$)
  enquanto $\rho > \delta$ **faça**
    $(x_R, y_R, \theta_R) \leftarrow$ `getPosition()`
    $\rho \leftarrow$ `getDistance`(x_R, y_R, x_f, y_f)
    $F_{att} \leftarrow$ `getFatt`(K_{att}, ρ)
    $F_{rep} \leftarrow$ `getFrep`(ε, K_{rep})
    se $F_{rep} \geq 0$ **então**
      se `verifyObs()` **então**
        Break
      fim se
    fim se
    $F_{tot} = F_{att} + F_{rep}$
    $v = \| F_{tot} \|$
    $\omega = K_{\omega} \arctan2(F_{tot}^{\{y\}}, F_{tot}^{\{x\}}) - \theta_R$
    `atuaVel`(v, ω)
  fim enquanto
  return 0
fim função

Figura 34 – Problemas de Mínimos Locais.



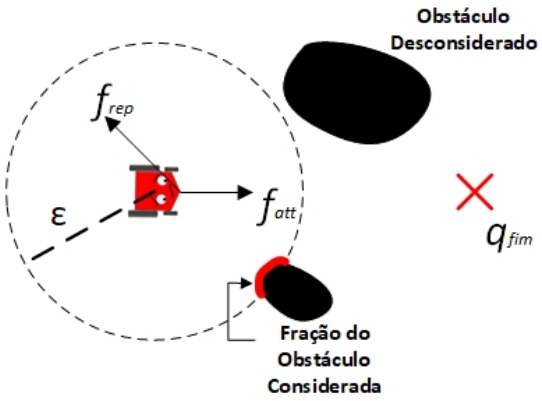
Fonte: Elaborado pelo Autor

entretanto, à medida que o sistema robótico se afasta do objetivo a força atrativa volta a se tornar maior que a repulsiva fazendo com que o mesmo volte para dentro do obstáculo. Esse movimento se repete indefinidamente.

Além desses mínimos locais [85] também aponta para o problema com passagens estreitas, que no pior dos casos, o robô pode ficar impossibilitado de atingir o objetivo conforme representado pela Figura 35. Assim, a força repulsiva pode crescer tanto que o robô se movimentará na direção oposta ao objetivo.

Em outros casos o robô poderá chegar ao objetivo em passagens estreitas, entretanto,

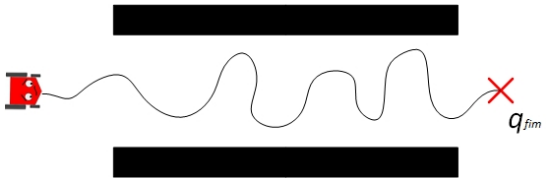
Figura 35 – Problemas de Mínimos Locais em Passagens Estreitas



Fonte: Adaptado de [85]

movimentos oscilatórios diante desses tipo de obstáculos podem surgir [58]. Esse problema pode ser melhorado a partir de uma calibração dos ganhos do método, entretanto, deve-se ressaltar que não é um procedimento simples, e que ao ajustar os ganhos para uma determinada passagem estreita, não há a garantia que o movimento oscilatório deixe de existir em outra passagem.

Figura 36 – Comportamento Oscilatório em Passagens Estreitas



Fonte: Elaborado pelo Autor

2.8 ROBOT OPERATING SYSTEM (ROS)

As aplicações desenvolvidas na área da robótica podem ser incorporadas à diversos tipos de robôs. São várias as plataformas de *Hardwares*, sensores com comportamentos distintos e linguagens de programação que variam de acordo com a opção do desenvolvedor, são elementos que tornavam os pesquisadores na área da robótica distantes. As ferramentas eram desenvolvidas geralmente, no tratamento de determinados tipos de robôs, não podendo ser utilizada de maneira genérica [102]. Desse modo, em maio de 2007 pesquisadores roboticistas iniciaram os estudos de um *framework* para unir os elementos

que são necessários para que robôs desenvolvam suas missões algum tempo depois esse sistema se tornaria a base do que é denominado ROS. O sistema operacional indicado pelos desenvolvedores para o uso do ROS é *Ubuntu Linux*, entretanto, é possível instala-lo em *Mac OS X* e *Windows* de maneira experimental.

O ROS é uma iniciativa das instituições *Stanford Artificial Intelligence Robots* (STAIR) com a *Personal Robots* (PR), e hoje é controlado pela *Open Source Robotics Foundation* (OSRF). O ROS pode ser definido como um conjunto de ferramentas, bibliotecas e convenções utilizados para desenvolver aplicações na área da robótica [103]. No artigo de apresentação do ROS [104] os autores citam como pontos positivos do sistema a leveza de processamento, *framework* baseado em ferramentas, multi-linguagem, gratuito, código aberto e provê comunicações ponto-a-ponto. O ROS não age como um sistema operacional comum realizando escalonamento e organizando tarefas, sua função é manter uma comunicação estruturada acima de um sistema operacional [105]. Versões do ROS são lançadas anualmente para que ocorra a melhoria do sistema e a inserção de novas características. Neste trabalho utilizou-se a versão estável nomeada de *Indigo*.

Esse *framework* provê facilidades em abstração de *hardwares* como sensores, comunicação em baixo nível com microcontroladores bem como em *drivers* de motores e o transporte de mensagens entre os diversos processos que estarão presentes na aplicação [105].

As tarefas que o robô deverá desempenhar poderão ser implementadas em diferentes linguagens de programação *C++*, *Python*, *C*, LISP, *Octave* ou outra qualquer que o desenvolvedor tenha experiência [106]. Recentemente, foi incorporada ao *Matlab* para ser adquirida, a ferramenta denominada *Robotic System Toolbox*, que possibilita o uso de vários algoritmos, além de realizar o interfaceamento com *hardwares* para serem aplicados em manipuladores, veículos robóticos e humanoides utilizando o ROS [107]. A desvantagem da ferramenta é o elevado preço comercial. Através de mensagens padronizadas as tarefas implementadas nas diversas linguagens poderão trocar informações. Como exemplo, há mensagens do tipo *laser Scan*, odometria, velocidades, imagens, sonares, etc.

De acordo com [108] o ROS pode ser dividido em três níveis de conceitos:

- **Sistema de Arquivos:** Conjunto de arquivos estruturados em pastas que são básicos para que o ROS posso operar;
- **Computação em Grafos:** Arquitetura baseada em nós (*nodes*) que permite a execução de maneira independente e estabelece a conexão entre eles de modo ponto-a-ponto. Poderá ser melhor compreendida na Seção 2.8.1.
- **Comunidade:** Nesse nível é onde se encontra a maior fonte de informações do ROS, além dessa característica, há um repositório onde é possível obter algoritmos e

códigos de qualquer desenvolvedor.

No Sistema de Arquivos destaca-se o elemento denominado pacote (*packages*), que carrega conteúdo suficiente para criar programas dentro do ROS. Esse possui processos para a execução de programas, arquivos de configuração e de compilação, bibliotecas, e o executável dos programas (nós). De acordo com [103] os pacotes devem fornecer funcionalidades suficientes para ser prático e leve. Os pacotes que foram utilizados por esse trabalho serão apresentados nas Seções 3.1.1, 3.1.2, 3.1.3 e 3.1.4.

2.8.1 **Computação em Grafos**

Para o uso do ROS é necessário o entendimento dos conceitos fundamentais: nós (*nodes*), mensagens (*messages*), tópicos (*tópics*), recipientes (*bags*) e serviços (*services*). A computação em Grafos consiste em formar uma *interface* onde todos os processos serão conectados, a partir desse conceito, pode-se afirmar que qualquer elemento poderá ter acesso aos outros pertencentes ao sistema. Abaixo haverá uma breve descrição dos elementos que são observados na Computação em Grafos:

- **Nós:** são definidos como os programas que serão executados no ROS. A característica modular do ROS é propícia a criação da aplicação através de diversos nós. De acordo com [109] o uso dos nós, simplifica o sistema já que pode-se separar as funcionalidades e os códigos em nós distintos. A modularidade também possibilita o processamento da aplicação de maneira paralela, ou seja, vários nós poderão atuar ao mesmo tempo, na execução da missão. Um nó deverá ter um nome único no sistema, já que essa característica é a distinção entre os nós. Ressalta-se que o termo nós quando se tratar de ROS, será diferente dos nós apresentados nos algoritmos baseados nas RRTs.

- **Nó *Master*:** o Nó *Master* é um nó padrão criado logo após a inicialização do ROS que se comporta como um servidor, onde é armazenado o registro e informações de todos os outros nós, serviços, tópicos e mensagens do sistema. Dessa maneira, quando algum nó quer trocar informações com outro ou simplesmente publicar em um tópico, primeiramente, é acessado o nó *Master* para que as informações sejam repassadas de modo correto (Figura 37 (a)). Quando dois nós querem se conectar haverá a consulta de informações no nó *Master*, porém, quando a conexão for efetivada, será de maneira ponto-a-ponto, ou seja, sem a necessidade do Nó *Master*.

- **Mensagens:** consiste na informação que circula entre os diversos nós do sistema e são caracterizadas pelo tipo de informação que a mesma carrega. Atualmente, o sistema conta com uma grande quantidade de tipos de mensagens, entretanto, é

possível criar outros tipos conforme a necessidade do desenvolvedor. As mensagens podem ser tanto uma variável quanto um conjunto de dados estruturados.

- **Tópicos:** podem ser definidos como os canais que trafegam as informações denominadas mensagens que caracterizam o tipo do tópico. Essas mensagens utilizam o padrão *publish - subscribe* (publicadores - assinantes), em que o publicador é o responsável pelo envio da mensagem e o assinante o responsável por recebê-la. Um tópico pode ter vários *subscribers*, entretanto, apenas um publicador poderá enviar informações pelo canal. Esse padrão de comunicação foi definido por [110] e emula um *container* onde as informações publicadas são armazenadas, nesse sentido, os assinantes interessados nessas mensagens se conectarão a esse *container*. O uso desse meio possibilita a publicação sem que haja *subscribers* para assinar.

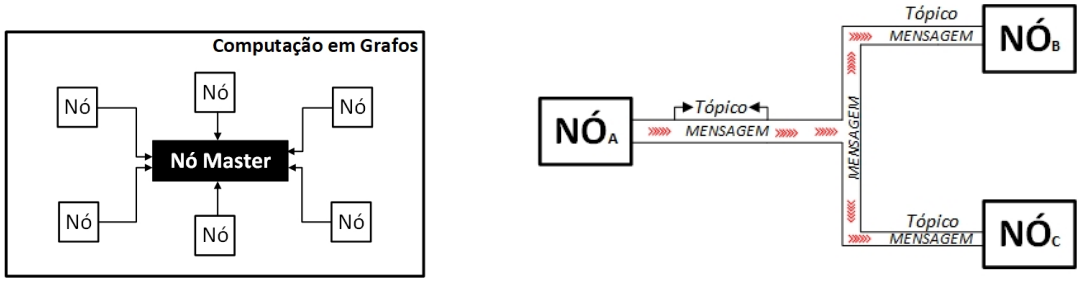
- **Servidor de Parâmetros:** quando for necessário a alteração do funcionamento ou de alguma configuração de um nó durante a execução do sistema, utiliza-se o Servidor de Parâmetros. Por esse elemento é possível que seja armazenado informações em um dicionário multivariado de maneira compartilhada.

- **Serviços:** consiste em uma variante do tipo de comunicação entre dois nós. É utilizado quando for necessário que a comunicação seja do tipo requisição/resposta, ou seja, um nó publicará em um tópico caso seja requisitado.

- **Recipientes (*bags*):** são arquivos gravados de um sistema onde determinados tipos de mensagens já foram reproduzidas. A intenção no uso das *bags* é dar capacidade ao desenvolvedor de analisar, reutilizar, interagir e editar as aplicações sem a necessidade de realizar novamente o experimento [105].

A Figura 37 (b) representa um exemplo geral da topologia da troca de informações entre o Nó **A** (publicador) com os Nós **B** e **C** (assinantes). Destaca-se a característica unidirecional da mensagem que circula pelo tópico.

Figura 37 – Arquitetura ROS: (a) Interação dos Nós com o Nó *Master* (b) Troca de informações entre Nós.



(a) (b)

3 METODOLOGIA PROPOSTA

Nesse capítulo será apresentado como foi implementada a missão autônoma proposta neste trabalho bem como as contribuições realizadas. Utilizou-se o *framework* ROS para o controle e leitura dos sensores do sistema robótico cujos pacotes serão apresentados nessa seção. Posteriormente será apresentado como o Filtro de Kalman Estendido possibilita a localização do robô. E por fim, serão mostrados como serão realizados o planejamento de caminhos durante as missões.

3.1 PACOTES DO ROS UTILIZADOS NA MISSÃO AUTÔNOMA

3.1.1 Pacote *RosAria*

O pacote *RosAria* destina-se a integrar o robô P3DX ao sistema ROS. O desenvolvedor desse robô liberou a *framework* Aria para prover o controle e a leitura dos sensores acoplados ao robô [111], dessa maneira, criou-se o pacote para ROS que faz uso da biblioteca Aria para a aplicação em diversos robôs da fabricante.

Através do *RosAria* é possível ter acesso ao baixo nível do robô aplicando velocidades e também o acesso às leituras dos sensores que estão presentes no P3DX. O pacote cria o nó *RosAria* no momento da execução que por sua vez, produz vários tópicos conforme mostrado a seguir:

- **cmd_vel**: o nó *RosAria* se inscreve como assinante desse tópico a espera do sinais de velocidades lineares e angulares que irão controlar o robô. Esses sinais serão provenientes do controlador definido no método dos campos potenciais artificiais explicados na Seção 2.7.3

- **pose**: o nó *RosAria* se inscreve como publicador desse tópico. Por esse tópico trafegam a posição estimada pelos sensores *encoders*. O controlador interno do P3DX é o responsável por integrar as rotações do sensor e estimar a posição.

- **bumper_state**: o nó *RosAria* se inscreve como publicador desse tópico. A informação que trafega está relacionada a um sensor de contato que se encontra logo à frente do P3DX, sendo assim, é possível verificar quando o robô colide frontalmente com um obstáculo.

- **sonar**: o nó *RosAria* se inscreve como publicador desse tópico. A partir desse, são retiradas as 8 leituras dos sonares que se encontram no P3DX. Essa informação será utilizada no planejador *online* para que o robô se comporte de maneira reativa conforme os obstáculos apareçam pelo ambiente.

- **battery_state_of_charge**: o nó *RosAria* se inscreve como publicador desse tópico que circula a informação se as baterias que alimentam o P3DX necessitam ser carregadas ou não.

- **battery_voltage**: o nó *RosAria* se inscreve como publicador desse tópico que informa qual é a tensão presente nos polos das baterias do robô.

- **motor_state**: o nó *RosAria* se inscreve como publicador desse tópico. É possível verificar a partir desse tópico se os motores estão habilitados ou não.

3.1.2 Pacote Openni2

O pacote Openni2 provê os *drivers* necessários para que o dispositivo de visão seja integrado ao ROS. É importante ressaltar que optou-se utilizar um dispositivo idêntico ao *Kinect* que foi desenvolvido pela *Microsoft* por ser mais robusto fisicamente e utilizar alimentação de energia no próprio cabo USB que trafega as informações da imagem. O dispositivo em questão foi desenvolvido pela empresa *Asus* com foco no uso em computadores e o modelo utilizado foi o *Xtion PRO Live*.

A partir desse pacote é possível ter acesso às principais imagens geradas pela câmera RGB, pelo sensor Infravermelho e em escalas de cinza.

Ao ser executado, é criado o nó *camera* que inicializa vários tópicos cujos principais serão descritos a seguir:

- **rgb/image_raw**: o nó *camera* se inscreve como publicador desse tópico e envia a imagem RGB provida pela câmera RGB no dispositivo *Asus Xtion*. Esse tópico será assinado pelo pacote *ArTrack Alvar* que será explicado na Seção 3.1.4. O tipo de mensagem é A *image*.

- **depth/image_raw**: o nó *camera* se inscreve como publicador desse tópico que circula a imagem em escalas de cinza. Esse tópico será assinado pelo pacote *Pacote DepthImage to Laser Scan* para extrair as distâncias entre os obstáculos e a câmera. Por esse tópico também circula mensagens do tipo *image*.

- **rgb/camera_info e depth/camera_info**: o nó *camera* se inscreve como publicador desse tópico. Esse tópico trafega as informações referentes às câmeras que geram as respectivas imagens.

- **driver/parameter_descriptions**: o nó *camera* se inscreve como publicador desse tópico. Esse tópico trafega as mensagens com os parâmetros de configuração de *drivers* do pacote *Openni2*.

- **driver/parameter_updates**: o nó *camera* se inscreve como assinante desse tópico. Por esse tópico é possível publicar mensagens que alterarão as configurações de *drivers* do pacote.

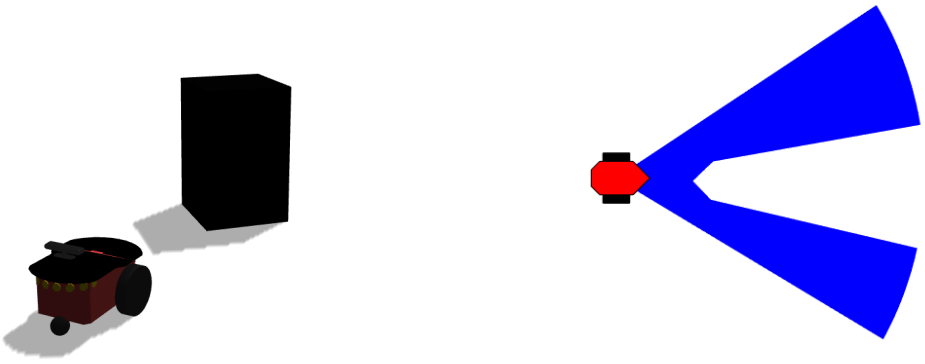
Por ser um pacote que geraM tópicos que outros pacotes irão assinar, é necessário que sua execução seja realizada primeiramente.

3.1.3 Pacote *DepthImage to Laser Scan*

Como mencionado na seção 2.4, o sensor *Asus Xtion* consegue gerar imagens em escala de cinzas proporcional à distância em que os objetos se encontram da câmera, então convém obter essas distâncias tal qual o comportamento dos sensores *lasers scans*. Entretanto, o ângulo de cobertura da câmera seria apenas de 60^o, menor que a maioria dos sensores *Lasers*, além disso, a distância de detecção de objetos seria uma faixa de de $0,4m$ à $5m$. Contudo, o preço do *Asus Xtion* é bem inferior quando comparado à sensores *lasers scan*, o que torna o experimento mais atrativo do ponto de vista comercial.

O pacote *DepthImage to Laser Scan* interpreta a imagem em escalas de cinzas e extrai uma faixa com 640 medidas em um ângulo de aproximadamente 60^o como mostrado pela Figura 38, gerando uma resolução de 1 medida a cada $0,0935^o$.

Figura 38 – Comportamento do Pacote *DepthImage to Laser Scan*: (a) Situação Analisada (b) Conversão da Imagem em *Lasers scan*.



(a) (b)

A execução do pacote cria o nó *DepethImage_To_LaserScan* que por sua vez cria alguns tópicos como pode ser visto a seguir.

- **scan**: o nó *DepethImage_To_LaserScan* se inscreve como publicador desse tópico que trafega mensagens do tipo *LaserScans*. Essas mensagens são listas que contem todas as 640 medidas de distâncias encontradas a partir da análise da imagem em escalas de cinzas. Essas leituras serão utilizadas posteriormente no mapeamento e pelos campos potenciais para o planejamento *online* da missão. As medidas que se

encontram abaixo de $0,40m$ e acima de $5m$ são definidas como *nan* (do inglês, *not a number*) e simplesmente são descartadas.

- **parameter_descriptions**: o nó *DepethImage_To_LaserScan* se inscreve como publicador desse tópico. A partir desse tópico é possível encontrar informações relacionadas às configurações do algoritmo que interpreta a imagem em escala de cinza e converte para *Laser Scan*.

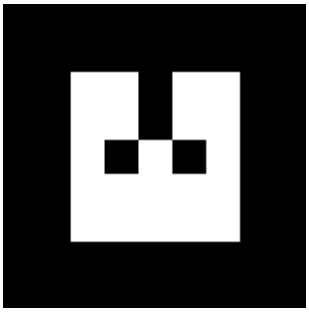
- **parameter_update**: o nó *DepethImage_To_LaserScan* se inscreve como assinante desse tópico. Caso seja necessário modificar alguma configuração do pacote, será utilizado esse tópico.

3.1.4 Pacote ArTrack Alvar

O pacote denominado *ArTrack Alvar* é capaz de detectar padrões denominados *Ar codes* (*Augmented Reality Codes*) como o exemplo apresentado pela Figura 39. De acordo com [112], esses padrões podem ser considerados códigos de barras em duas dimensões. Os *Ar codes* são baseados na tecnologia criada em 1994 pela empresa *Denso-Wave* com propósitos na catalogação de peças de uma linha de montagem automobilística [113], cujos benefícios logo fizeram com essa novidade fosse nomeada como *QR Codes* (*Quick Response Codes*).

O pacote é implementado através da linguagem C++ e pertence ao conjunto de bibliotecas *VTT ALVAR Tracking Library* [114]. Os *Ar codes* que são reconhecidos por esse pacote, pertencem ao conjunto de padrões cadastrados na biblioteca Alvar. Nesse trabalho, os padrões corresponderão aos marcadores que serão espalhados pelo ambiente de modo a auxiliar o robô no problema de localização.

Figura 39 – Exemplo de Padrão Utilizado.



Fonte: Elaborado pelo Autor

O uso dos *Ar codes* são impulsionados por vários motivos, como a simples confecção, fácil detecção por câmeras convencionais, não obstrui passagens e ainda armazena

informações [112]. Essas características contribuem para o uso em diferentes aplicações, especialmente na área da robótica como apresentado nos trabalhos [115], [112], [116].

Esse pacote consegue entregar a posição relativa do marcador em relação à câmera. O aspecto de profundidade é dado pelo tamanho do marcador, ou seja, inicialmente deve-se passar para o pacote as suas dimensões. A partir da análise do tamanho do *Ar Code* encontrado em uma determinada imagem, é possível obter as coordenadas relativas, já que o tamanho real do marcador é de conhecimento do pacote. Além dessas dimensões, são necessários valores de *threshold*, que ao serem diminuídos podem aumentar a quantidade de detecção de falsos positivos obtidos pelo *Ar Track Alvar*.

As informações são armazenadas na imagem utilizando uma codificação binária representadas por quadrados (células) pretos e brancos com as mesmas dimensões. A quantidade de informações é proporcional ao número de células que estão contidos no interior da faixa preta externa da Figura 39. A faixa tem o objetivo de realizar um contraste em relação aos outros elementos presentes na imagem. Nesse trabalho os *Ar codes* possuem um total de 25 células dispostos em fileiras com 5.

O nó *Ar_Track_Alvar* é criado quando o pacote é executado. Para o funcionamento de maneira correta, é necessário que a execução aconteça após o funcionamento do *Openni2*. Esse nó se inscreve para ser assinante do tópico em que a imagem RGB que foi gerada pelo sensor *Asus Xtion* circula (/camera/rgb/image_raw). Além desse tópico, é necessário que esse pacote tome conhecimento das informações de calibração da câmera presentes no tópico */camera/rgb/camera_info*. Também é necessária a interação com o tópico onde há as informações do *frame* que a câmera se encontra para que as coordenadas dos *Ar codes* sejam obtidas de maneira correta.

Os algoritmos desse tipo poderão enfrentar problemas em encontrar marcadores inclinados em relação à posição da câmera ou em ambientes com pouca luminosidade, pois, essa característica impacta diretamente na diferença de contraste entre as partes pretas e brancas [103].

Um breve resumo dos tópicos que o pacote cria ao ser inicializado é visto a seguir:

- **ar_pose_marker**: o nó *Ar_Track_Alvar* se inscreve como publicador desse tópico que trafega mensagens do tipo *AlvarMarkers*. Essas mensagens são listas que contem todas as posições relativas dos marcadores vistos em relação à câmera.
- **enable_detection**: o nó *Ar_Track_Alvar* se inscreve como assinante desse tópico. Caso seja necessário desativar ou ativar a detecção de *Ar codes* realiza-se por esse tópico.
- **parameter_descriptions**: o nó *Ar_Track_Alvar* se inscreve como publicador desse tópico. A partir desse canal, é possível encontrar informações relacionadas às

configurações do algoritmo que está sendo utilizado.

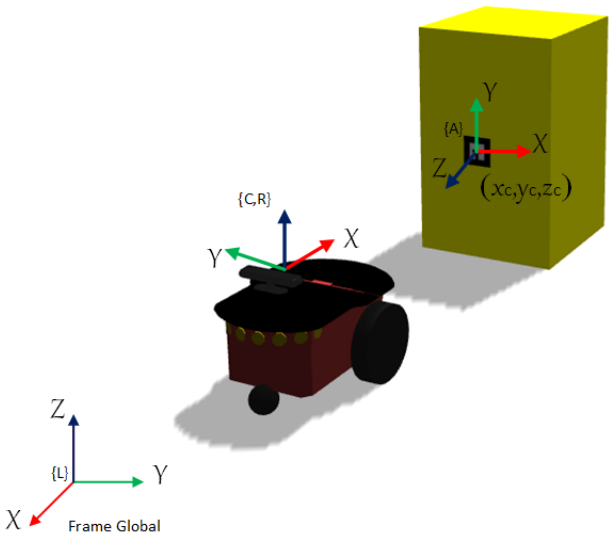
- **parameter_updates**: o nó *Ar_Track_Alvar* se inscreve como assinante desse tópico. Caso seja necessário atualizar alguma informação após a execução do pacote, utiliza-se esse tópico.

- **visualization_marker**: o nó *Ar_Track_Alvar* se inscreve como publicador desse tópico que trafega mensagens do tipo *Markers* semelhantes aos *AlvarMarkers*, entretanto, há algumas informações a mais sobre os *Ar codes* além das posições relativas.

3.1.4.1 Localização do Robô em Relação ao *Frame* Global Via *Asus Xtion*

O objetivo do uso desses marcadores é obter a posição em que o robô se encontra em relação ao referencial global do sistema. Entretanto, cada um dos elementos que irão compor a missão também possui seu próprio conjunto de referenciais (marcadores, câmera e robô). A Figura 40 exemplifica o sistema proposto.

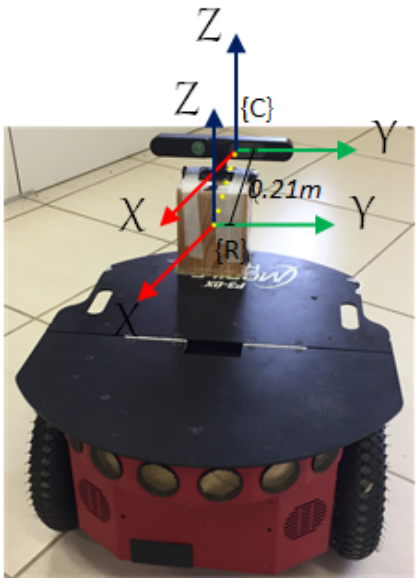
Figura 40 – Disposição dos *Frames* Utilizados.



Fonte: Elaborado pelo Autor

Na Figura 40 simplificou-se o referencial da câmera $\{\mathbf{C}\}$ com o do robô $\{\mathbf{R}\}$, entretanto, devido à dificuldade do sensor *Asus Xtion* em encontrar obstáculos a distâncias menores que $0,4m$, o mesmo foi deslocado para a parte traseira do P3DX. Dessa maneira, a interação entre os 2 *frames* consiste em um deslocamento apenas na direção $\mathbf{X}$ conforme apresentado na Figura 41.

Figura 41 – Interação entre o *Frame* da Câmera e do P3DX.



Fonte: Elaborado pelo Autor

Quando um marcador é encontrado, o tópico *ar_pose_marker* publica mensagens contendo a posição e a orientação do mesmo em relação ao referencial da câmera. Entretanto, a posição do marcador e a orientação em relação ao referencial global estão armazenados em um mapa de marcadores, de onde será possível encontrar a posição da câmera de maneira global, e consequentemente, a posição do robô. Ressalta-se que os *Ar codes* espalhados pelo ambiente serão únicos.

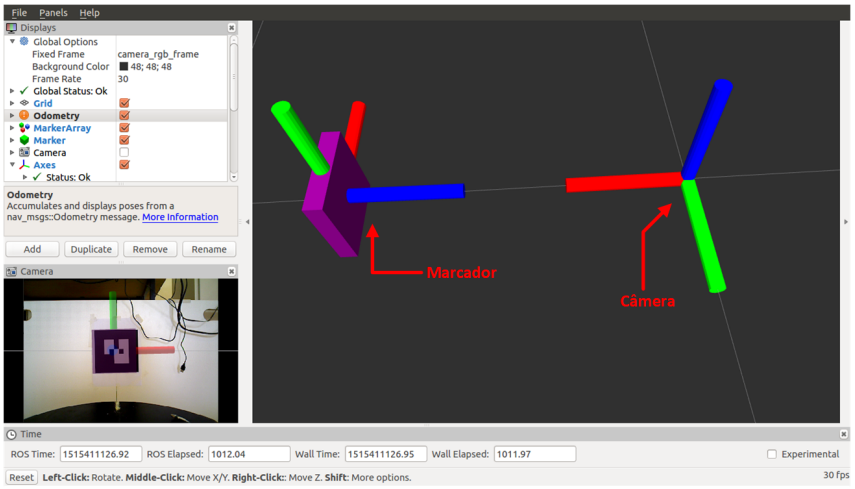
A posição de um marcador é dada a partir das posições x_c, y_c e z_c em relação ao *frame* da câmera $\{\mathbf{C}\}$ (Figura 40), e a orientação é dada pelos *quaternions* q_x, q_y, q_z e q_ω. Propositalmente colocou-se os centros dos *Ar codes* para coincidir com o plano $Z = 0$ do referencial global, assim, não haveria resultados no eixo Z. Quando a câmera avista um marcador, deve-se analisar como estão se comportando os referenciais. A partir da Figura 42 é possível notar que o eixo Z do *frame* que corresponde ao marcador está na direção contrária ao eixo X da câmera. Para esse trabalho, considerou-se que os eixos X,Y e Z da câmera/robô deverá ter as mesmas orientações do *frame* global.

A primeira informação que poderá ser encontrada é a orientação do robô. As informações angulares de um marcador é dada por *quaternions*, nesse sentido, será necessário convertê-los para ângulos fixos conforme a Equação 3.1:

$$\phi = \arctan \frac{2(q_\omega q_z + q_x q_y)}{1 - 2(q_y^2 + q_z^2)} \tag{3.1}$$

Analisando a disposição dos *frames* na Figura 42, é possível traçar relação de ϕ

Figura 42 – Interação entre o *Frame* da Câmera e um Marcador Encontrado.



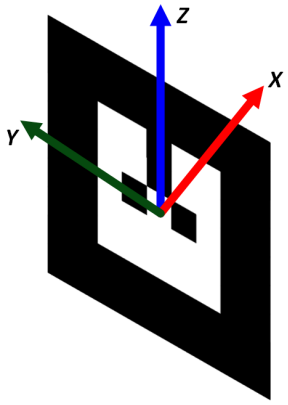
Fonte: Elaborado pelo Autor

para obter o ângulo do robô (θ) no *frame* global conforme a Equação 3.2:

$$\theta = -\phi - \frac{\pi}{2} + \theta_M \tag{3.2}$$

onde θ_M corresponde à orientação do marcador em relação ao referencial global. Convencionalmente, adotou-se que a orientação do marcador globalmente seria conforme a Figura 43. Ressalta-se a diferença na orientação no marcador armazenado no mapa, com os eixos captados pela câmera. Desse modo, ao longo do processo de localização, a orientação dos eixos coordenados deverá ser corrigida.

Figura 43 – Convenção Adotada ao Armazenar um Marcador no Mapa.



Fonte: Elaborado pelo Autor

Para obter a posição do robô no ambiente, foi necessário realizar as transformações

homogêneas apresentadas na fundamentação teórica deste trabalho a parti das informações dadas pelo *ar_pose_marker*. Deve-se realizar o processo inverso partindo da câmera indo até o marcador, que por sua vez chega até o referencial global. Como o marcador está orientado em um ângulo $90 + \phi$ em relação a câmera, e o mesmo é fixo no ambiente, é necessário fazer uma rotação $-(90 + \phi)$ em **Z**. Essa fase inicial pode ser observada pela Equação 3.3:

$$P_T = (Rot_Z(\frac{-\pi}{2})Rot_X(\frac{\pi}{2}))^{-1}Rot_Z(-(90 + \phi))P \tag{3.3}$$

as informações dada pelo *Ar Code*, posiciona a câmera em relação ao marcador, dessa maneira, deve-se inverter a matriz de transformação para obter a relação correta. Primeiramente, deve-se rotacionar em torno de **Z** um ângulo de $\frac{-\phi}{2}$ e posteriormente $\frac{\phi}{2}$ ao redor de **X** conforme a Equação 3.3.

Para que P_T chegue à orientação dos eixos coordenados globais, é necessário realizar outras rotações para que os eixos coincidam. Assim, rotacionou-se **X** em um ângulo de $\frac{-\phi}{2}$ e depois **Z** $\frac{\phi}{2}$ conforme a Equação 3.4:

$$P_T = Rot_Z(\frac{\pi}{2})Rot_X(\frac{-\pi}{2})P_T \tag{3.4}$$

Por fim, deverá haver uma rotação para que o ângulo de orientação do marcador coincida com o *frame* global. É necessário levar em consideração a defasagem de π dos *Ar codes* (Figura 43) em relação ao eixo global, dessa maneira, defini-se $\delta = \theta_M + \pi$. Por fim, há uma última rotação em torno de **Z** com um ângulo de δ. Como os eixos estão coincidindo com o *frame* global, pode-se somar a posição dos marcadores que foram armazenados no mapa para obter a posição global da câmera. Esse último passo pode ser descrito a partir da Equação 3.5:

$$P_{global,C} = Rot_Z(\delta)P_T + [m_{j,x}, m_{j,y}, m_{j,z}, 1]^T \tag{3.5}$$

onde o vetor $[m_{j,x}, m_{j,y}, m_{j,z}, 1]$ corresponde à posição do marcador j em relação ao *frame* global para os eixos **X**, **Y** e **Z**. O último elemento do vetor é igual a um e tem o objetivo de garantir a consistência matricial dos cálculos. Deve-se ressaltar que todas as componentes no eixo **Z** são iguais a 0.

A posição global obtida corresponde localização do *Asus Xtion*, entretanto, o dispositivo de visão se encontra em um *frame* diferente, conforme mostrado pela Figura 41. Assim, deve-se realizar uma rotação em **Z** com o mesmo ângulo de orientação do robô (θ) e uma translação (T) de $\Delta x = 0,21m$ para encontrar a posição global conforme apresentado

na Equação 3.6:

$$P_{global,R} = Rot_Z(\theta)T(\Delta x, 0, 0)P_{global,C} \tag{3.6}$$

Unindo as informações de posição definidas em $P_{global,R}$ com a orientação θ é possível criar o vetor de observação $\hat{\mathbf{Z}}$ com a resposta do sensor *Asus Xtion* conforme apresentado pela Equação 3.7

$$\hat{\mathbf{Z}}_{\mathbf{t}} = \begin{pmatrix} x_s \\ y_s \\ \theta_s \end{pmatrix} \tag{3.7}$$

onde $\hat{\mathbf{Z}}$ é o vetor que contém as coordenadas x_s, y_s e θ_s dada pela localização do robô através da visão de *Ar codes*, no momento t ao encontrar o marcador j alocado no mapa. Cada uma das componentes de $\hat{\mathbf{Z}}$ estarão sujeitas à ruídos gaussianos. Os índices s ao lado das componentes de $\hat{\mathbf{Z}}$ remetem sensor. Deve-se frisar que uma modelagem mais robusta dos ruídos teriam outros fatores que impactariam nesses valores, como a distância que a câmera se encontra do marcador e luminosidade do ambiente.

3.2 LOCALIZAÇÃO DE ROBÔS MÓVEIS

A localização é um importante conceito que deverá ser utilizado nas missões autônomas. Optou-se por utilizar o Filtro de Kalman, pois, os ruídos sensoriais e problemas como deslizamento de rodas e o sequestro dos robôs poderão ser tratados através da união de sensores. A modelagem matemática da odometria e da câmera apresentaram aspectos não lineares que impossibilitam o uso do Filtro de Kalman. Por esse motivo utilizou-se o Filtro de Kalman Estendido (EKF).

Essa seção destina-se a apresentar como foi implementado o algoritmo de localização de maneira detalhada. Como descrito na fundamentação teórica, o EKF é dividido em duas fases predição e atualização que foram apresentadas nas seções 3.2.1 e 3.2.2 respectivamente.

O sistema definido para esse trabalho consiste obter a posição atual $\mathbf{X_t}$ a partir da posição anterior $\mathbf{X_{r-1}}$ e as unidades de controle $\mathbf{u_t} = [v\ \omega]$. O vetor $\mathbf{X_t} = [x_t\ y_t\ \theta_t]$ é formado pela posição do robô em relação ao *frame* global e sua orientação com a adição de ruídos. O vetor $\mathbf{X_{r-1}} = [x_r\ y_r\ \theta_r]$ corresponde a posição real que o robô se encontra ao iniciar o deslocamento com as unidades de controle $\mathbf{u_t}$. A resposta da odometria é dependente dos vetores $\mathbf{X_{r-1}}$ e $\mathbf{u_t}$ podendo ser incorporados em uma função g conforme apresentado pela Equação 3.8. A posição atual é estimada a partir da odometria cujos

ruídos poderão afetar essas medidas. Na Equação 3.8 os ruídos correspondem ao vetor $\mathbf{E_o}$.

$$\mathbf{X_r} = g(\mathbf{X_{r-1}}, \mathbf{u}_t) + \mathbf{E_o} \tag{3.8}$$

A observação do sistema ajuda a estimar a posição atual e será realizada pelas câmeras do sensor *Asus Xtion* que retornará um vetor com a posição relativa dos *Ar codes* em relação à $\mathbf{X_t}$. A matriz de observação corresponde ao vetor $\hat{\mathbf{Z}}$ da Equação 3.7, cuja equação é descrita em 3.9. O vetor $\mathbf{E_c}$ corresponde aos ruídos gaussianos provenientes do sensor *Asus Xtion*:

$$\hat{\mathbf{Z}}_\mathbf{t} = h(\mathbf{X_t}) + \mathbf{E_c} \tag{3.9}$$

onde $h(\mathbf{X_r})$ corresponde à função que que descreve a observação de maneira ideal. Essa função é apresentada pela Equação 3.10:

$$\mathbf{h}(\mathbf{X_r}) = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix} \tag{3.10}$$

Ressalta-se que a posição real do robô não é possível obter, somente a posição dada pelo sensor que é corrompida pelo ruído é possível estimar.

3.2.1 **Fase de Predição**

A fase de predição será a responsável por realizar estimativa da posição $\mathbf{X_t}$ para o momento atual t. A partir de duas leituras em sequência do sensor odométrico é possível dividir o movimento em três partes (δ_{rot1}, δ_{trans} e δ_{rot2}) como apresentado na Seção 2.3.

Nessa fase será utilizado apenas o sensor odométrico, que estima a posição atual a partir da anterior. Soma-se incrementos lineares (Δx e Δy) e angulares ($\Delta\theta$) à posição anterior para obter o estado atual. A obtenção desses incrementos também foram registradas na Seção 2.3 da fundamentação teórica.

A nova posição é obtida através da função de deslocamento $o(\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ que é dependente da divisão em três movimentos do modelo odométrico. Pode-se chegar aos três movimentos utilizando duas posições captadas de maneira consecutivas pelo sensor odométrico. Quando há essa divisão, os erros acumulados ao longo da missão poderão não são serem propagados quando existir a fase de atualização. Deve-se ressaltar que a resposta do *encoder* leva em consideração a posição $\mathbf{X_{t-1}}$ e as unidades de controle (u).

Dessa maneira, a função o (Equação 3.11) pode ser considerada uma boa predição para o estado do sistema, tomando o lugar da função g.

$$o(\delta_{rot1}, \delta_{trans}, \delta_{rot2}) = \mathbf{X_t} = \mathbf{X_{t-1}} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{pmatrix} \tag{3.11}$$

Assim a posição atual do robô será estimada, entretanto, a mesma apresentará erros devido aos ruídos presentes em cada um dos três movimentos. A continuação do EKF consiste em encontrar a atriz de covariância do erro (Σ) para a iteração atual. A abordagem utilizada por Thrun em [27] sugere que Σ (Equação 3.12) seja utilizado da seguinte maneira:

$$\mathbf{\Sigma_t} = \mathbf{G_t \Sigma_{t-1} G_t^T} + \mathbf{M_t} \tag{3.12}$$

onde a variável $\mathbf{G_t}$ representa o Jacobiano da função de deslocamento em relação às coordenadas do robô considerando δ_{rot1}, δ_{trans} e δ_{rot2} constantes. A matriz $\mathbf{\Sigma_{t-1}}$ é a covariância do erro obtido na iteração anterior. $\mathbf{M_t}$ é a matriz diagonal com as covariâncias dos ruídos gaussianos do sensor *encoder* que interferem na posição. Essas matrizes são mostradas nas Equações 3.13 e 3.14.

$$\mathbf{G_t} = \begin{bmatrix} \frac{\partial o_1}{\partial x_t} & \frac{\partial o_1}{\partial y_t} & \frac{\partial o_1}{\partial \theta_t} \\ \frac{\partial o_2}{\partial x_t} & \frac{\partial o_2}{\partial y_t} & \frac{\partial o_2}{\partial \theta_t} \\ \frac{\partial o_3}{\partial x_t} & \frac{\partial o_3}{\partial y_t} & \frac{\partial o_3}{\partial \theta_t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta y \\ 0 & 1 & \Delta x \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

$$\mathbf{M_t} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{3.14}$$

Onde as variáveis σ_x^2, σ_y^2 e σ_θ^2 correspondem às covariâncias oriundas do *encoder* para cada uma das coordenadas que compõe uma pose do robô.

3.2.2 **Fase de Atualização**

A fase de atualização tem o objetivo de corrigir a posição estimada na fase de predição. Essa fase não acontece em todas as iterações, pois, em alguns momentos o sensor *Asus Xtion* não encontrará *Ar codes* no seu raio de visão. Quando esse fato ocorre, a predição realizada é a melhor estimativa para a posição atual. Entretanto, se a câmera captar um ou mais *Ar codes* entrará na fase de atualização.

Seguindo a ordem apresentada para o Filtro de Kalman Estendido na Seção 2.6.1, a fase de atualização inicializa calculando a inovação que as observações estão fazendo no momento. Essa inovação compara a posição global $\hat{\mathbf{Z}}_\mathbf{t}$ dada pela manipulação do pacote *ArTrack Alvar* com a posição global encontrada na fase de predição $\mathbf{X_t}$ (Equação 3.7). A inovação $\mathbf{N}$ pode ser calculada conforme a Equação 3.15:

$$\mathbf{N} = \mathbf{X_t} - \hat{\mathbf{Z}}_\mathbf{t} \tag{3.15}$$

Como $\mathbf{X_t}$ é a posição estimada para o robô na fase de predição, a inovação será um importante argumento para a avaliação do quão longe está a posição estimada da captada pela câmera. Essa informação pôde ser utilizada desse modo justamente pelo fato do *ArTrack Alvar* informar qual é o *Ar code* que está no campo de visão da câmera e os mesmos serem únicos no ambiente. Por esse motivo os *Ar codes* foram escolhidos para auxiliar a localização, por não apresentarem problemas de ambiguidades. Ao ser descoberto qual é o *Ar code* visto, será possível consultar no mapa sua respectiva posição. Dessa maneira, calcula-se a posição global conforme Equação 3.7. Então, se a posição do robô estiver distante da posição real o valor de $\mathbf{N}$ será alto, impactando diretamente na atualização da posição. Caso fosse utilizado outra característica que não fosse distinguível, a eficiência do EKF para localização estaria comprometida, já que o vetor $\hat{\mathbf{Z}}_\mathbf{t}$ poderia retornar a posição de um outro *Ar code*.

O próximo passo do EKF é encontrar o ganho de Kalman necessário para atualizar a posição predita calculado pela Equação 3.16:

$$K_t = \mathbf{\Sigma}_t \mathbf{H}_t^T (\mathbf{H}_t \Sigma_t \mathbf{H}_t^T + \mathbf{Q}_t)^{-1} \tag{3.16}$$

onde K_t é o ganho de Kalman, $\mathbf{H}_t$ é o Jacobiano do modelo da câmera (Equação 3.10) em relação ao vetor de posição $\mathbf{X}$. A matriz $\mathbf{Q_t}$ é a matriz diagonal com as variâncias referentes ao ruído gaussiano da câmera para cada uma das coordenadas do vetor de posição. O Jacobiano $\mathbf{H}_t$ é mostrado na Equação 3.17 e a matriz $\mathbf{Q_t}$ na Equação 3.18.

$$\mathbf{H_t} = \begin{bmatrix} \frac{\partial h_1^i}{\partial \bar{x}_r} & \frac{\partial h_1^i}{\partial \bar{y}_r} & \frac{\partial h_1^i}{\partial \bar{\theta}_r} \\ \frac{\partial h_2^i}{\partial \bar{x}_r} & \frac{\partial h_2^i}{\partial \bar{y}_r} & \frac{\partial h_2^i}{\partial \bar{\theta}_r} \\ \frac{\partial h_3^i}{\partial \bar{x}_r} & \frac{\partial h_3^i}{\partial \bar{y}_r} & \frac{\partial h_3^i}{\partial \bar{\theta}_r} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$\mathbf{Q_t} = \begin{bmatrix} \sigma_{x,s}^2 & 0 & 0 \\ 0 & \sigma_{y,s}^2 & 0 \\ 0 & 0 & \sigma_{\theta,s} \end{bmatrix} \tag{3.18}$$

Após a aplicação do jacobiano, a matriz $\mathbf{H_t}$ corresponde à forma diagonal. Esse resultado foi utilizado em [118] e mostrou-se que o mesmo ocorre devido à modelagem do sensor. A partir da visão dos *Ar codes*, é possível encontrar a posição global, então pode-se dizer esse sensor retorna o estado completo (do inglês, *Full State Sensor*) [118].

Por fim deve-se atualizar a posição atual e a matriz de covariância encontrado na primeira fase do EKF apresentados nas Equações 3.19 e 3.20, respectivamente.

$$\mathbf{X_t} = \mathbf{X_t} + K_t\mathbf{N} \tag{3.19}$$

$$\mathbf{\Sigma_t} = (\mathbf{I} - K_t\mathbf{H_t})\mathbf{\Sigma_t} \tag{3.20}$$

Onde a matriz $\mathbf{I}$ corresponde a matriz de identidade.

As variâncias relacionadas aos ruídos do sensor odométrico e da câmera foram obtidas de maneira empírica. O ruído do sensor odométrico deve receber uma atenção maior, já que o impacto do mesmo na estimativa da posição vai se acumulando ao longo do tempo. Além disso, deve-se atentar ao fato de que o ruído a ser considerado também depende das velocidades das rodas que estão impostas no robô. Devido à complexidade de definir um modelo que considere todos os fatores que impactam diretamente nos resultados do *encoder* optou-se por uma abordagem mais simples. Considerou ruídos independentes entre si nas coordenadas x, y e θ das posições. Essa abordagem simplória do ruído da odometria foi utilizado em vários trabalhos [28], [58] e [118] com sucesso na solução dos problemas relacionados à localização.

3.2.3 O Algoritmo de Localização

Para que a localização se comporte da maneira correta, é necessário passar para o algoritmo uma estimativa inicial e erro de predição inicial não nulos. Pois, a fase de predição necessita das estimativas no momento $t-1$ para predizer os valores na posição t. Para garantir a recursividade do algoritmo, é necessário definir os termos no instante $t-1$ tal como as Equações 3.21 e 3.22 ao final de toda iteração do FKE.

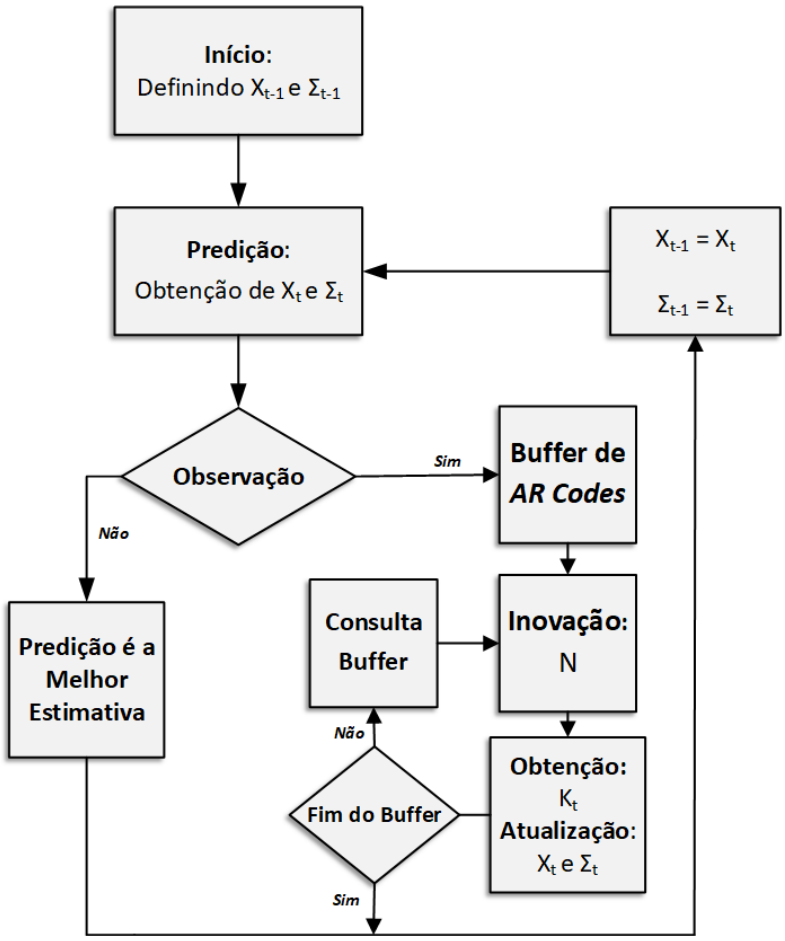
$$\mathbf{X_{t-1}} = \mathbf{X_t} \tag{3.21}$$

$$\mathbf{\Sigma_{t-1}} = \mathbf{\Sigma_t} \tag{3.22}$$

O algoritmo também executa a fase de atualização em uma iteração n vezes conforme apresentado o resumo do método no Fluxograma de Figura 44. Em que n é a quantidade de *Ar codes* encontrados pelo *ARTrack Alvar*. Desse modo, é possível

aumentar o desempenho da localização dos robôs móveis nesse trabalho. Todos os *Ar codes* observados armazenam a informação em um *buffer*, para que todos sejam considerados na estimativa da posição. A fase de observação terminará quando todos *Ar codes* forem utilizados.

Figura 44 – Fluxograma da Localização via FKE.



Fonte: Elaborado pelo Autor

3.2.4 Topologia do Algoritmo no ROS

A localização através do Filtro de Kalman Estendido terá que interagir com alguns tópicos criados por outros nós. Quando o código do EKF proposto for executado, será criado um novo nó denominado *EKF*.

Quando o nó for inicializado, o tópico *Pose_EKF* é criado tendo como publicador o próprio nó *EKF*. Esse tópico circulará as posições corrigidas a partir do método FKE. O intuito desse tópico é divulgar para os outros algoritmos que compõe o sistema de navegação uma estimativa da posição melhor quando comparados à informação odométrica.

O EKF também irá se inscrever como assinante no tópico *pose* que foi criado pelo pacote *RosAria* para que a predição no algoritmo EKF seja realizada. Além dessa, também há a assinatura do tópico *ar_pose_marker* gerado pelo pacote *ArTrack Alvar* que passa as posições relativas dos *Ar codes* no campo de visão da câmera.

3.3 PLANEJAMENTO DE CAMINHOS

Essa seção destina-se a apresentar como os planejadores descritos na fundamentação teórica deste trabalho auxiliam nas missões autônomas. O planejamento será responsável por guiar o robô por uma trajetória que ligue a posição inicial ao objetivo em que o mesmo se encontra. Ressalta-se que o robô deverá desempenhar uma missão de navegação de maneira autônoma deslocando-se até uma posição objetivo.

A metodologia do planejamento é divida em duas etapas. A primeira executa um planejamento de maneira deliberativa ou *offline* baseado nas RTTs. A partir das coordenadas geradas pela etapa inicial o algoritmo dos campos potenciais irá planejar o caminho de maneira reativa ou *online* para segui-las. A atuação dos CPA será a segunda etapa dos planejadores.

3.3.1 **Planejador Deliberativo ou *Offline***

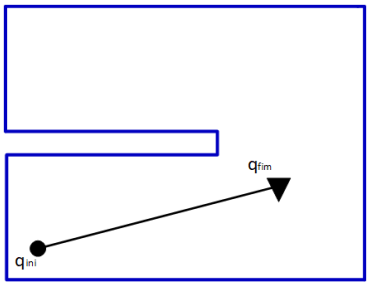
3.3.1.1 Direct-RRT

Devido à experiência obtida a partir da implementação e estudos dos métodos baseados em RRTs (já presentes na literatura, RRT, RRT* e DRRT) detectou-se a possibilidade de melhoria a partir de uma situação distinta. Esse trabalho propõe um método baseado na RRT para o aumento da eficiência do planejador *offline*. Para demonstrar a melhoria ao utilizar o método, serão analisados o tempo para obter um caminho, a quantidade de nós, iterações e o tamanho do caminho gerado na seção referente aos resultados.

A primeira etapa do algoritmo, verifica se há algum obstáculo entre a configuração inicial (q_{ini}) e o objetivo (q_{fim}). Se não houver, a amostragem da RRT será fixa na configuração objetivo. Dessa maneira, o caminho gerado pelo método será ótimo, conforme mostrado pela Figura 45. Caso fosse utilizado o RRT, o caminho não seria ótimo e amostragens desnecessárias seriam realizadas. Vale lembrar que o estado amostrado não será a configuração de um novo nó que será conectado à árvore, ele apenas ajudará a criar o estado q_{new} (oriundo da aplicação do modelo cinemático), que corresponderá ao estado de um novo nó.

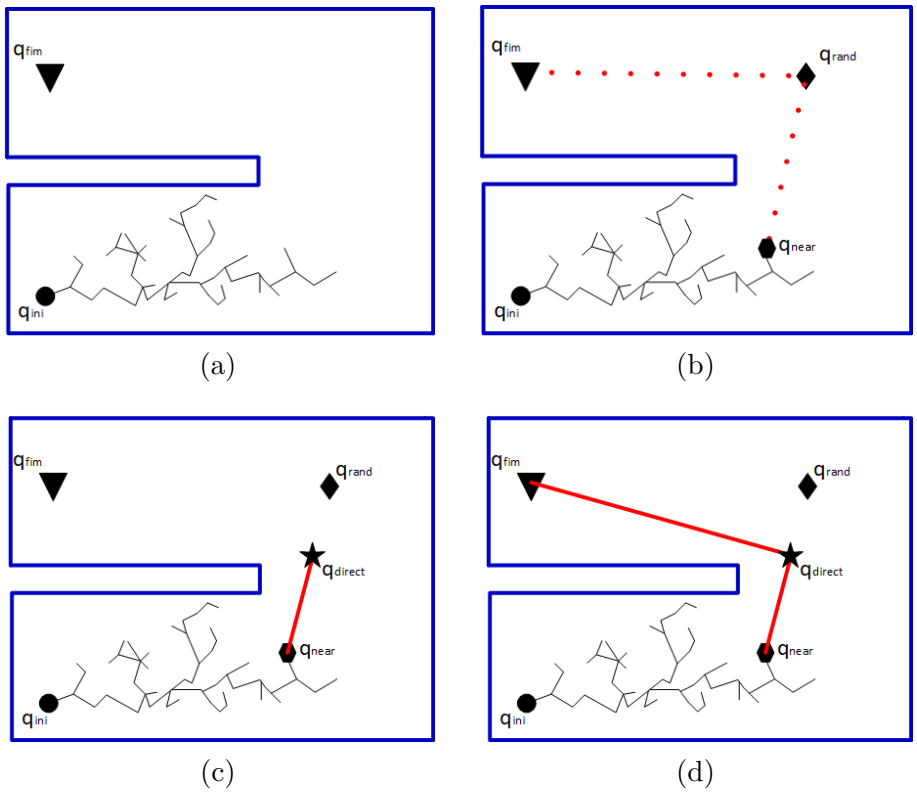
Se houver obstáculos entre a posição inicial e o objetivo, a árvore se expande tal qual a RRT clássica (Figura 46 (a)). Essa expansão ocorre até que a situação de ativação da heurística do método aconteça.

Figura 45 – Resultado do Direct-RRT com Caminho Ótimo.



Fonte: Elaborado pelo Autor

Figura 46 – Expansão do algoritmo Direct-RRT: (a) Expansão Tal Qual uma RRT; (b) Ativação da Heurística; (c) Expansão através do método Direct-RRT; (d) Resultado Final do Algoritmo Direct-RRT.



Em toda iteração é verificada duas características. A primeira analisa se as configurações q_{near} podem ser conectada à configuração amostrada q_{rand} naquela iteração sem a interferência de obstáculos. A segunda análise, verifica se o mesmo q_{rand} pode ser conectado à posição objetivo q_{fim} também sem a presença de obstáculos. Quando essas duas situações forem satisfeitas a heurística é ativada (Figura 46 (b)).

A heurística inicia-se construindo uma nova árvore cujo nó raiz é aquele que possui a configuração q_{near} no momento em que a heurística for ativada. A expansão da árvore é dividida em duas etapas. A primeira realiza uma amostragem fixa na posição q_{rand} que ativou a heurística.

Dessa maneira, a árvore cresce em direção ao q_{rand}. Sempre que um novo nó é adicionado à nova árvore é verificado se sua configuração pode ser ligada ao objetivo sem a colisão com obstáculos (Figura 46 (c)). Quando ocorrer essa situação a segunda etapa da expansão do método Direct-RRT é iniciada. Caso não fosse realizada essa a verificação de colisão em todos os novos nós a expansão seguiria até a configuração q_{rand} aumentando a distância do caminho gerado pelo método de maneira desnecessária.

A segunda fase define que a amostragem seja fixa na configuração q_{fim}. Assim a árvore seria expandida de q_{direct} até o q_{fim}. O caminho gerado pela nova árvore é mostrado na Figura 46 (d) por uma linha sólida vermelha. Ressalta-se que todos nós adicionados à nova árvore também são incorporoados à árvore inicial para que o caminho que conecte q_{ini} à q_{fim} seja encontrado da mesma maneira definida no RRT. A amostragem fixa faz com que a árvore do método proposto se conecte ao objetivo de maneira direta, por esse motivo o método foi denominado Direct-RRT.

Após o q_{new} chegar até q_{fim}, o método realiza o retrocesso, partindo do último nó até o inicial, analisando o parentesco tal qual ao RRT clássico para que o conjunto de estados que definem o caminho seja obtido.

3.3.1.2 Planejador Deliberativo Implementado

O planejador deliberativo é constituído da união dos algoritmos RRT [30], RRT* [46], DRRT [49] e Direct-RRT (proposto nesse trabalho). O RRT compõe a base desse planejador e os motivos do seu uso foram justificados na fundamentação teórica.

Entretanto, as amostragens aleatórias do algoritmo RRT geram caminhos não-ótimos. Por esse motivo utilizou-se o método RRT* para obter caminhos que se aproxime dos ótimos. Contudo, o esforço computacional será maior e o tempo para gerar um caminho também aumentará.

Para tentar diminuir os impactos negativos no processamento que o RRT* traz, utilizou-se o DRRT. Como mostrado na fundamentação teórica o RRT* analisa os nós dentro de uma circunferência. O tempo computacional tende a aumentar a medida que a quantidade de nós também aumentam. Para ajudar a diminuir o peso computacional do RRT*, utilizou-se o DRRT para evitar que nós com estados ambíguos sejam alocados na árvore.

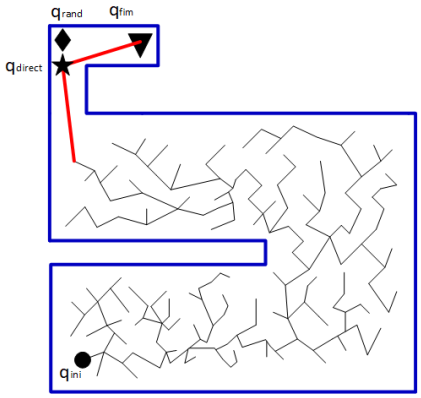
O DRRT analisará a dispersão dos nós evitando que sejam incorporados à árvore nós com estados que estejam próximos uns aos outros para não haver redundâncias. Quando os nós são filtrados, a tendência é diminuir o número de iterações e nós, consequentemente, haverá a diminuição no tempo para obter um caminho. A fusão entre o RRT* e DRRT é relativamente fácil, pois, a atuação dos mesmos acontecem em etapas distintas do algoritmo RRT. Enquanto o DRRT atua na amostragem do método, o RRT* atua na definição dos

parentescos dos nós.

Além do RRT* e do DRRT, adicionou-se ao RRT básico, o método proposto por este trabalho denominado Direct-RRT. Essa heurística tem o propósito em diminuir a quantidade de nós e iterações, o tempo computacional e o tamanho do caminho gerado. A união destes métodos foi dado o nome de Direct-DRRT*.

O uso do DRRT e RRT* junto ao método Direct-RRT foi necessário, pois, o sucesso do último está diretamente relacionado à situação que ativa a heurística. Portanto, antes de sua inicialização, o método deverá preocupar-se com caminhos curtos através do RRT* e da quantidade de nós e iterações DRRT. A Figura 47 representa um desses casos em que a ativação da heurística demoraria acontecer.

Figura 47 – Posição Objetivo de Difícil Acesso.



Fonte: Elaborado pelo Autor

A Figura 47 mostra os casos em que o objetivo se encontra em uma área de difícil acesso. A expansão da árvore antes da ativação da heurística deve receber uma atenção especial, pois, até que a ativação da heurística ocorra, a árvore expandida pode apresentar problemas com redundância dos nós (motivação DRRT) e definição de parentescos ruins (motivação RRT*).

Priorizou-se obter caminhos em um tempo mínimo, ou seja, o primeiro caminho encontrado pelo algoritmo seria o planejamento da missão. O resultado do planejador Direct-DRRT* é um conjunto de coordenadas ordenadas que poderão ser consideradas sub-objetivos que os campos potenciais deverão seguir.

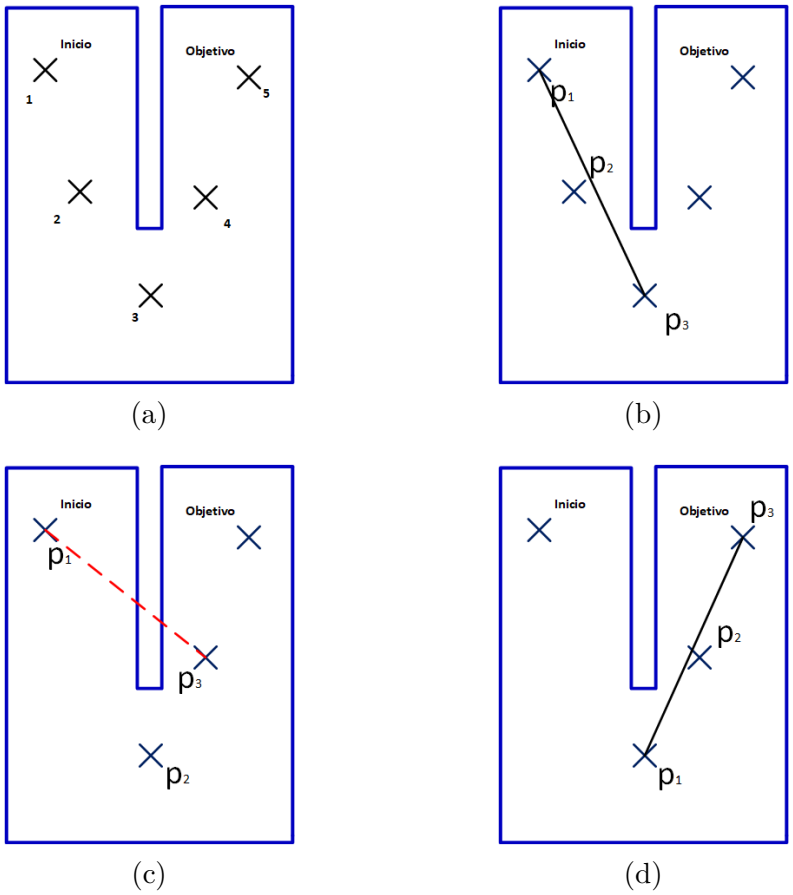
A vantagem do método deliberativo em questão, é a impossibilidade de existir problemas com mínimos locais. Então, dado o mapa do ambiente com os obstáculo, será possível encontrar um caminho entre o início e objetivo, caso exista (completude probabilística).

3.3.1.3 Processo de Suavização

A resposta dos algoritmos RRTs é um conjunto de coordenadas ordenadas. Caso sejam repassadas diretamente para os campos potenciais artificiais, o robô poderia se comportar de maneira não satisfatória. Essa característica ocorre porque duas coordenadas consecutivas se encontram próximas umas das outras, e as leis de controle poderiam tornar o processo instável. Como apresentado na fundamentação teórica, a velocidade linear dada pelo controlador dos CPA é dependente da distância entre o robô e o objetivo (ou sub-objetivo). Então, o controlador poderia enviar velocidades pequenas, podendo deixar a missão inviável. Devido à esses problemas, foi necessária a suavização do conjunto de coordenadas.

O método de suavização foi o mesmo abordado em [119] e mostrou resultados satisfatórios. A suavização utilizada é mostrada na Figura 48 em 4 estágios. Imagina-se que o planejamento deliberativo tenha resultado em um caminho com 5 coordenadas conforme apresentado na Figura 48 (a).

Figura 48 – Processo de Suavização de um resultado do Planejador Deliberativo: (a) Estágio 1 (b) Estágio 2 (c) Estágio 3 (d) Estágio 4.

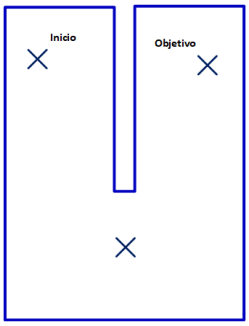


Defini-se o primeiro ponto do conjunto de Coordenadas como p_1, o seguinte como p_2 e o terceiro como p_3. Dessa maneira, é verificado se existe algum obstáculo entre p_1 e p_3 (Figura 48 (b)). Caso não haja, a coordenada em que p_2 foi associada é excluída. O

algoritmo segue definindo como p_2 o terceiro ponto do conjunto de coordenadas e p_3 o quarto ponto.

Entretanto, se houver algum obstáculo entre p_1 e p_3 o ponto p_2 é mantido no conjunto de coordenadas (Figura 48 (c)). Para o exemplo retratado, os pontos p_1, p_2 e p_3 deslocam-se para suas respectivas coordenadas seguintes. O final da suavização ocorre após a verificação da existência de obstáculos quando p_3 coincidir com a posição objetivo da missão (Figura 48 (d)). O resultado final desse exemplo, houve um decréscimo de 2 coordenadas em relação à situação inicial (Figura 49).

Figura 49 – Algoritmo Final de Suavização.



Fonte: Elaborado pelo Autor

Entretanto, esse algoritmo poderia resultar em problemas quando o robô fosse percorrer os sub-objetivos utilizando o CPA. Em situações quando a distância entre dois sub-objetivos fossem distantes, um objeto que representasse hipóteses de colisão poderia facilmente colocar o robô em situações com mínimos locais. Esse fato ocorre porque a força de atração poderia ser bem inferior à repulsiva nessas situações. Nesse sentido, deve-se suavizar o conjunto de coordenadas de maneira com que a distância entre dois sub-objetivos não sejam nem tão distantes e nem tão próximas.

Para contornar essa situação, colocou-se um limitador para que p_2 seja excluído. Antes que p_2 seja excluída, verifica-se a distância entre p_1 e p_3. Caso esse valor seja maior que um limiar, p_2 será mantido e p_1, p_2 e p_3 deslocam-se para suas coordenadas seguintes.

3.3.1.4 Planejador Deliberativo no ROS

O planejador deliberativo criará um *Nó* do ROS denominado *Path_Delib* que por sua vez também criará um tópico *Coordenadas*. Esse nó se comporta como o publicador do tópico que trafega o conjunto de coordenadas gerado pelo método Direct-DRRT*.

3.3.2 **Planejador Reativo ou *Online***

O planejador Reativo consiste na segunda etapa do processo de planejamento do caminho para que o robô atinja a posição objetivo. Essa etapa consiste na aplicação dos campos potenciais artificiais nos sub-objetivos (coordenadas) gerados pelo método Direct-DRRT* implementado na primeira fase.

O principal motivo para a implementação dessa segunda etapa é a possibilidade de surgir obstáculos dinâmicos que poderão estar no caminho das coordenadas geradas pelo Direct-DRRT*. Assim, o robô deverá se comportar de maneira reativa diante dos obstáculos para que um desvio possa ser realizado. Caso um obstáculo esteja muito próximo a um sub-objetivo o robô passa para o seguinte de maneira autônoma. Essa verificação só foi possível pela concepção de um algoritmo de localização.

Como mostrado na fundamentação teórica, as velocidades que serão impostas ao robô é proporcional a quantidade de obstáculos encontrado pelo sensor *Asus Xtion* e da posição objetivo. De acordo com as equações apresentadas pelo potencial reativo surgiria um problema quando o robô estivesse se aproximando de um sub-objetivo. As velocidades iriam diminuindo de acordo com a aproximação, podendo fazer com que o robô nunca chegasse ao objetivo. Por esse motivo, foi definida uma velocidade mínima a ser utilizada pelo robô. Será aplicada velocidade nula ao robô somente quando o mesmo se encontra dentro de uma região ao redor do último sub-objetivo, ou seja, no objetivo final. As constantes que deverão ser definidas no método foram escolhidas de maneira empírica. Utilizou-se uma heurística em que era verificado se a força repulsiva oriunda do sensor *Asus Xtion* era superior à atrativa em um fator de n vezes, nessas situações o robô receberia uma velocidade negativa, ou seja, o deslocamento terá velocidade negativa em relação ao referencial do robô. Essa característica foi imposta para que o robô evite realizar movimentos muito próximo a um obstáculo.

Quando o método reativo for utilizado em conjunto com o deliberativo tal qual exposto neste trabalho é formado um planejador híbrido.

O planejamento *offline* resulta em um caminho livre de mínimos locais, porém, os campos potenciais apresentam esses problemas. Então, para obstáculos de pequeno porte o sistema robótico conseguirá navegar até a posição objetivo com sucesso. Entretanto, para obstáculos de grande porte a característica reativa poderá prender o robô em mínimos locais.

Nessas situações o sistema deverá replanejar o caminho. Porém, os novos obstáculos representam uma importante região que deverá ser representada no mapa para que um caminho alternativo seja obtido. Para que seja possível obter um conjunto de coordenadas alternativo um mapeamento do ambiente deverá ser realizado.

3.3.2.1 Planejador Reativo no ROS

O planejador *online* criará um nó *Path_Reat* e se inscreverá em outros tópicos já existentes. No tópico *Coordenadas* o nó irá se inscrever como assinante para que o mesmo tenha acesso ao conjunto de coordenadas oriundas do planejador *offline*.

O controlador dos campos potenciais artificias terá que publicar no tópico */cmd_vel* que foi criado pelo pelo pacote *RosAria*.

A percepção dos obstáculos é dada pelas leituras do Sensor *Asus Xtion* convertida em distâncias através do pacote *DepethImage to Laser Scan*. O nó *Path_Reat* se inscreve como assinante do tópico */scan* para encontrar os obstáculos.

O método dos CPA também necessita da informação da posição em que o robô se encontra para no cálculo das funções repulsivas e atrativas. Assim, o *Path_Reat* deverá se inscrever como assinante no tópico *Pose_EKF* para obter as informações de localização.

3.4 MAPEAMENTO

O mapeamento será necessário quando alguns obstáculos irão modificar o ambiente a ponto de que as coordenadas geradas pelo Direct-DRRT* prendam o robô em regiões de mínimos locais. Quando essas situações ocorrem, é necessário que um mapeamento do ambiente seja realizado.

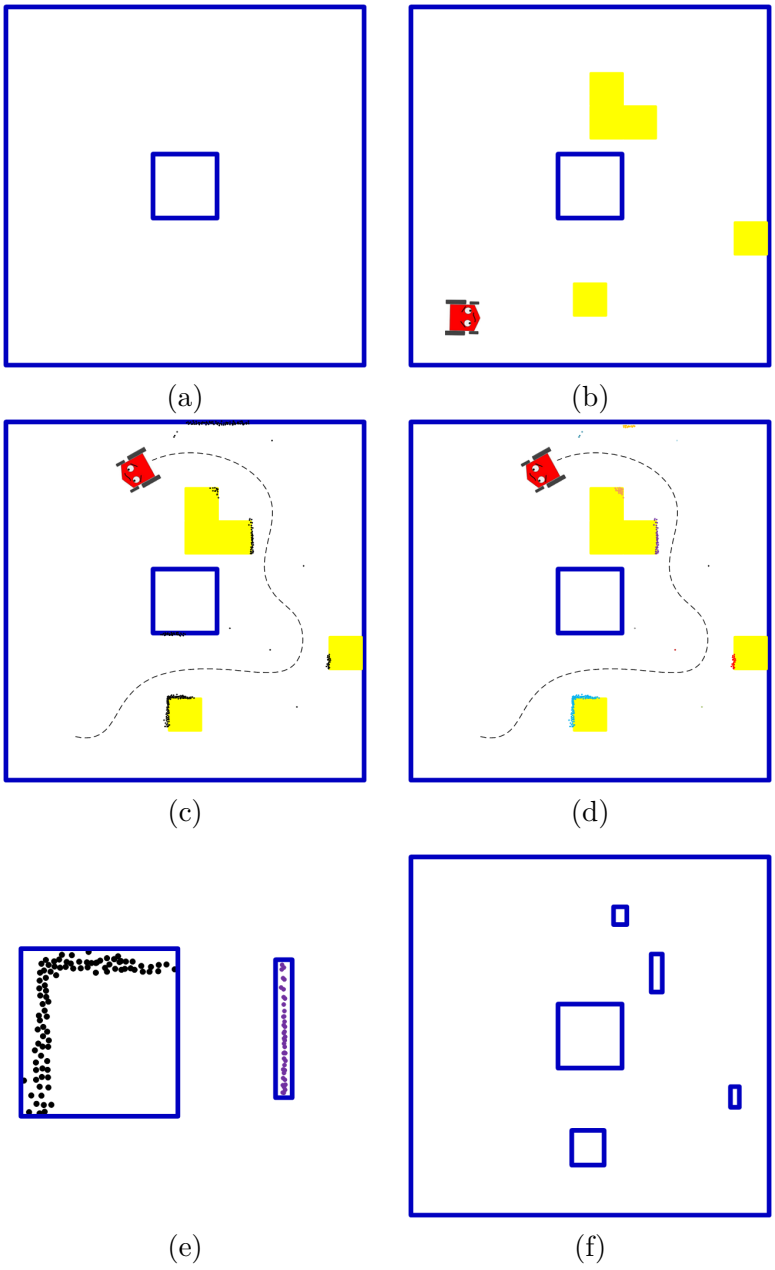
O mapeamento é realizado a partir das leituras extraídas do pacote *DepethImage to Laser Scan*. A medida que o robô vai se deslocando pelo ambiente os obstáculos detectados vão sendo armazenados em um vetor que futuramente atualizará o mapa. Essa situação ocorre somente se for necessário, isto é, se o robô ficar preso em um mínimo local. Considerou-se que o robô estivesse preso em mínimos locais, quando o deslocamento entre duas coordenadas ultrapasse um tempo pré-definido.

No mapeamento, muitos pontos que correspondem aos mesmos obstáculos vão sendo alocados próximos uns aos outros podendo haver um gasto de memória desnecessário. O sensor *Asus Xtion* obtém 640 medidas em um ângulo de aproximadamente 60^o que resulta em uma resolução menor que $0,1^o$. Nesse sentido, utilizou-se um conjunto de 20 medidas pertencente ao pacote *DepethImage to Laser Scan*, para realizar o mapeamento.

Para mostrar a metodologia do mapeamento, utiliza-se o mapa da Figura 50 (a). As arestas azuis são retângulos que possuem seus posicionamentos armazenados em um mapa. A Figura 50 (b) corresponde ao ambiente real, isto é, é apresentado os obstáculos dinâmicos (bloqueios amarelos) que por sua vez, não foram representados no mapa. O robô deverá deslocar-se pelo ambiente para extrair os obstáculos. Na Figura 50 (c) estão presente os obstáculos que foram armazenados no vetor durante a movimentação.

A localização dos obstáculos de maneira correta está atrelada ao bom funcionamento

Figura 50 – Processo de Mapeamento: (a) Mapa do Ambiente (b) Mapa do ambiente com os Obstáculos dinâmicos (c) Movimento e Mapeamento Feito pelo Robô (d) Processo de Clusterização (e) Transformação em Retângulos (f) Mapa Atualizado.



do algoritmo de localização que corrige a pose do robô durante a navegação. A ausência de um método de localização levaria o robô a encontrar posições de obstáculos erradas, ou seja, a atualização do mapa receberia posições equivocadas podendo comprometer a missão do sistema robótico.

O sensor *Asus Xtion* também detecta os obstáculos que já estão presentes no mapa, não sendo necessário considera-los novamente. Nesse sentido, os pontos pertencente ao vetor deverão ser analisados. Essa análise consistirá em verificar se o ponto do obstáculo percebido pertence ao mapa já existente, caso pertença, o mesmo será desconsiderado.

Após a exclusão dos pontos que já pertencem ao mapa, o restante deverão ser separados em obstáculos independentes. Para separa-los, utilizou-se um método de clusterização que avaliasse a proximidade dos pontos uns aos outros a partir da distância euclidiana. A clusterização é realizada pela função *FClusterData* da linguagem de programação *Python* pertencente à biblioteca *Scipy Cluster Hierarchy*. A clusterização foi retratada na Figura 50 (d) com cores distintas em cada um dos conjuntos de pontos obtidos.

Após a clusterização, haverá uma análise para descartar os *clusters* que possuam poucos pontos associados. Essa situação significa que foram realizadas poucas observações desse possível obstáculo ou erros encontrados pelo pacote *DepethImage_To_LaserScan*. Por esse motivo os mesmos poderão serem desconsiderados.

Por segurança os pontos são transformados em obstáculos no novo mapa por meio de um processo de retangularização sobre os conjuntos de pontos gerado pela clusterização. Esse processo foi representado na Figura 50 (e). O resultado final do mapa é representado na Figura 50 (f). Percebe-se que apenas algumas características dos obstáculos dinâmicos foram captadas, entretanto, a fidelização dessas obstruções com a realidade está diretamente relacionado com a movimentação do robô pelo ambiente.

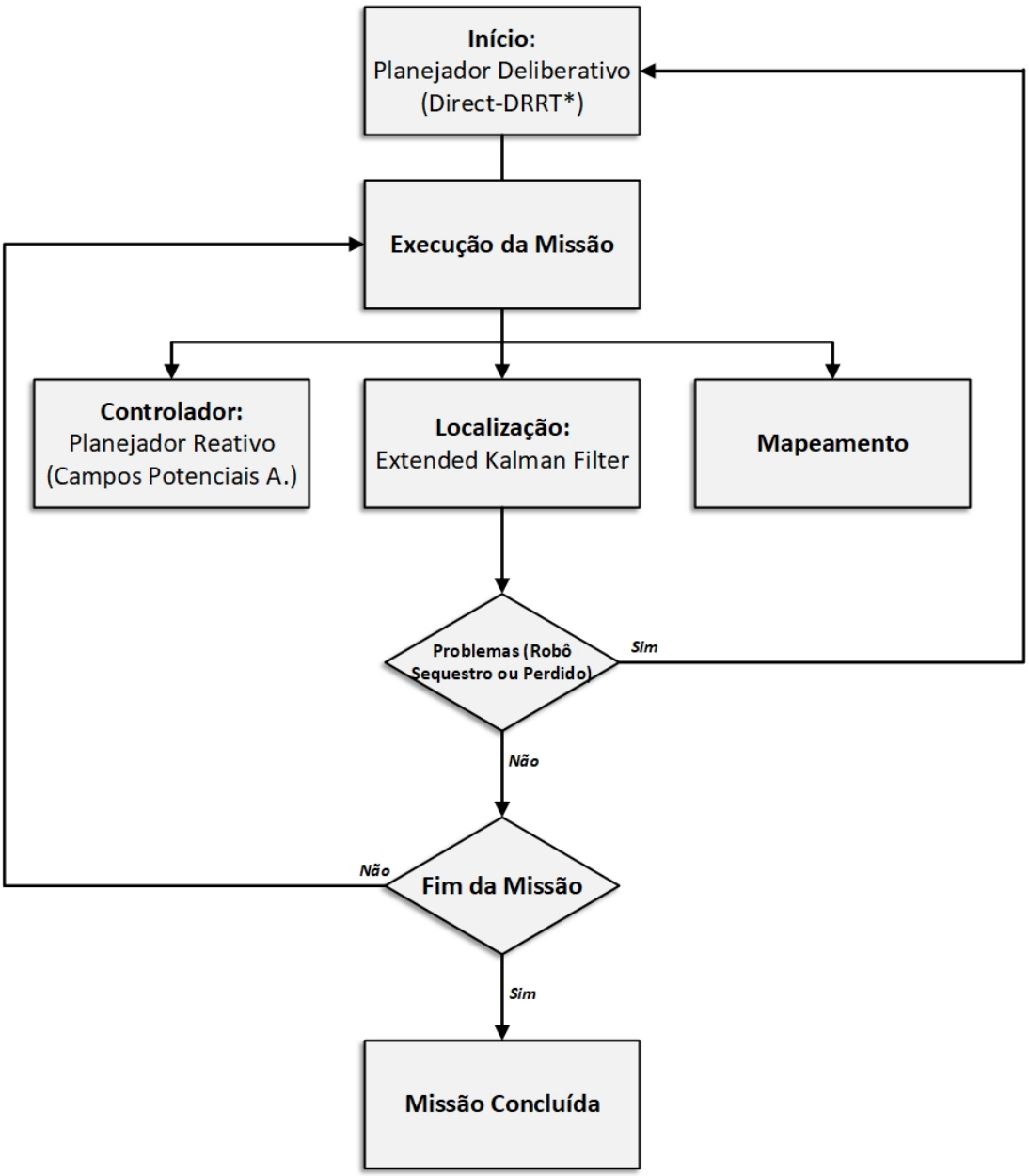
Após ser detectado que o robô está preso em um mínimo local é necessário paralisar a missão atual e gerar uma nova trajetória utilizando o Direct-DRRT*. Assim, o vetor de pontos detectados pelo *Asus Xtion* é analisado, e o mapa é atualizado. Deve-se ressaltar que os novos obstáculos deverão ser ampliados pelos mesmos motivos apresentados na Seção 2.7.1 para serem utilizados no planejador deliberativo. O planejamento *offline* é executado novamente e um novo caminho que passe por um local alternativo para atingir o objetivo deverá ser encontrado.

Devido à grande dependência da localização, o mapeamento é implementado no nó responsável por estimar a posição. Quando fosse requisitado (em mínimos locais) o tópico nomeado de *mapping* disponibilizaria o mapa atualizado. Desse modo, o nó *EKF* se inscreve como o publicador desse tópico.

3.5 Resumo da Missão Autônoma

A partir da metodologia proposta, é possível representar a missão autônoma através de um Fluxograma conforme apresentado na Figura 51.

Figura 51 – Fluxograma da Missão Autônoma.



Fonte: Elaborado pelo Autor

4 RESULTADOS

Este capítulo destina-as a apresentar os resultados obtidos para a concepção da missão autônoma de uma plataforma robótica diferencial. O computador utilizado neste trabalho possui 4 GB de RAM, o processador é um dispositivo da Intel modelo i3-3227U com 4 núcleos que trabalham com frequências de 1,90 GHz. Todos os códigos foram implementados na linguagem *Python*.

Os resultados foram divididos em três partes. O primeiro consiste em apresentar os resultados do planejador deliberativo Direct-DRRT* proposto, comparando com outros métodos presentes na literatura. A avaliação será feita nos parâmetros número de nós, de iterações, do tamanho dos caminhos e do tempo necessário para encontrar um caminho.

A segunda parte destina-se a apresentar os resultados encontrados para a simulação da missão autônoma. A missão consistirá no planejamento deliberativo, reativo e localização conforme apresentado na metodologia. Serão abordados três casos distintos. O primeiro compreende uma missão autônoma simples, o segundo, consiste em analisar o comportamento do sistema robótico quando o mesmo se encontra preso em mínimos locais, e por fim, a análise para o caso do sequestro de robôs. Os limites do ambiente em uma simulação pode ser considerado bem superiores se comparado a cenários reais, por esse motivo, é interessante haver a simulação prévia das missões.

O terceiro conjunto de resultados avalia o comportamento de um robô real. Será utilizada a plataforma P3DX e um dispositivo *Asus Xtion* para realizar as missões do robô.

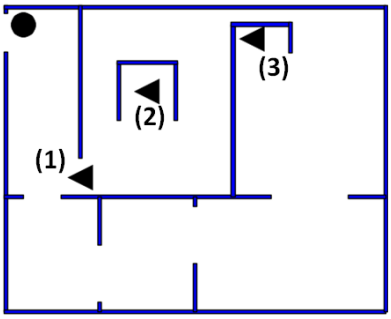
O objetivo desse trabalho é implementar em um sistema robótico com tração diferencial uma missão autônoma. O robô deverá partir de uma pose inicial e seguir até uma posição objetivo. Inicialmente deverá ser feito um planejamento *offline* a partir de um mapa base. Após obter um conjunto de sub-objetivos o deslocamento é feito pelo planejador *online*. O sistema deverá desviar dos obstáculos que não foram considerados pelo planejador *offline*. Além dessa característica, caso algum obstáculo prenda o robô em mínimos locais, o mesmo deverá conseguir mapear e replanejar um caminho que conecte a posição atual do sistema robótico até o objetivo. Além do problema dos mínimos locais, também será avaliado o comportamento do robô nos momentos em que for realizado o sequestro do dispositivo. A única condição necessária é que exista um caminho factível entre a pose do robô até a posição objetivo.

4.1 RESULTADOS DO PLANEJAMENTO *OFFLINE*

Os primeiros resultados que serão expostos por este trabalho estão relacionados com o planejador deliberativo apresentado na metodologia. Serão avaliados: o tempo

computacional gasto para encontrar um caminho; a quantidade de nós e iterações; o tamanho do caminho gerado. O ambiente retratado é apresentado pela Figura 52. Como mencionado na fundamentação teórica, o robô foi considerado com dimensões pontuais, nesse sentido, os obstáculos do mapa foram ampliados para aumentar a segurança nas rotas que serão obtidas.

Figura 52 – Ambiente Retratado.



Fonte: Elaborado pelo Autor

Na Figura 52 a circunferência representa a configuração inicial, ou seja, será a raiz da árvore. Os triângulos representam as posições objetivas cujo caminho serão obtidos a partir de vários métodos baseados no RRT.

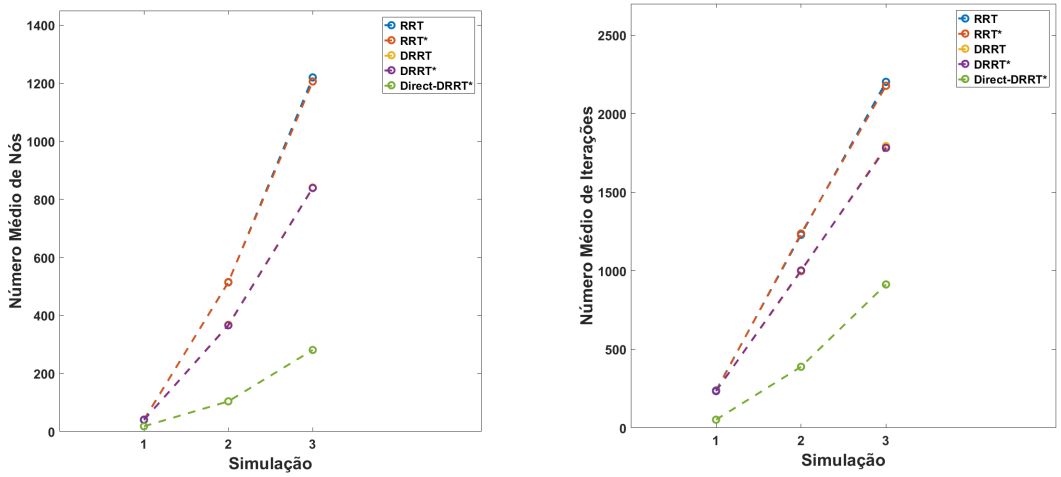
Para cada um dos objetivos serão realizadas 1000 execuções em sequência do planejador. Para que o método proposto seja avaliado é necessário uma comparação entre os métodos já existentes na literatura. A primeira análise está presente na Tabela 2 com os números médios de iterações e nós. Os dados também foram representados por meio de gráficos conforme apresentado pela Figura 53.

Tabela 2 – Número Médio de Nós e Iterações.

Simulação	**RRT**		**RRT***		**DRRT**		**DRRT***		**Direct-DRRT***	
	Nós	**Ite.**	**Nós**	**Ite.**	**Nós**	**Ite.**	**Nós**	**Ite.**	**Nós**	**Ite.**
1	41	238	40	234	40	235	40	234	18	52
2	514	1229	515	1237	368	996	366	1002	104	389
3	1220	2203	1206	2178	841	1794	864	1832	281	913

O método DRRT se comportou da maneira esperada, diminuiu o número de iterações e de nós quando comparado ao RRT. Por esse motivo observou uma queda nesses parâmetros no método DRRT* em relação ao RRT. Como o RRT* não tem propósitos na redução da quantidade de nós e iterações, não houve queda nesses índices. Observou-se que o melhor resultado considerando esses parâmetros foi apresentado pelo método Direct-DRRT*. Ao utilizar-se a fusão DRRT* foi possível encontrar melhorias nas características analisadas, entretanto, o método proposto (Direct-DRRT*) obteve

Figura 53 – Resultados dos Planejadores Deliberativos: (a) Número Médio de Nós (b) Número Médio de Iterações.



(a) (b)

uma significativa melhora, podendo ser considerado a melhor escolha para esses atributos. Outro ponto que merece destaque é a proximidade nos valores encontrado para a simulação 1, por existir poucos obstáculos entre o ponto final e o ponto inicial as heurísticas RRT* e DRRT não foram tão efetivas.

Por apresentarem as mesmas características quanto a quantidade de nós e de iterações, os gráficos do RRT e RRT* ficaram sobrepostos tal qual ao DRRT e DRRT*. conforme apresentado na Figura 53. Essa característica mostra a consistência dos testes realizados.

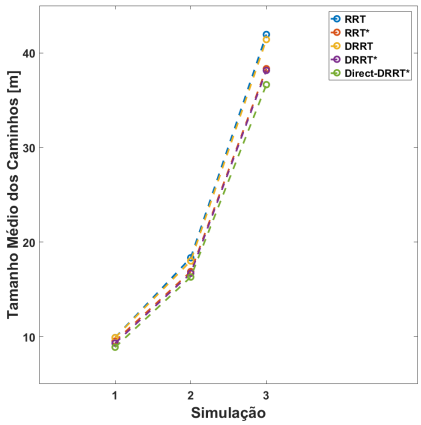
A Tabela 3 apresenta o valor médio dos tamanhos dos caminhos obtidos pelos planejadores. O gráfico para esse teste foi apresentado na Figura 54.

Tabela 3 – Tamanho dos Caminhos Médios Gerado pelos Planejadores.

Simulação	**RRT** $[m]$	**RRT*** $[m]$	**DRRT** $[m]$	**DRRT*** $[m]$	**Direct-DRRT*** $[m]$
1	9.91	9,53	9,90	9,32	8,89
2	18,35	16,89	18,01	16,71	16,31
3	41,96	38,34	41,42	38,23	36,65

Analisando a Tabela 3 é possível notar que o RRT* cumpre o seu papel, pois, o tamanho dos caminhos são menores se comparados ao RRT. O método DRRT não interfere no tamanho dos caminhos já que seu foco é a redução do número de nós e iterações. Nesse sentido pode-se encontrar um resultado próximo no tamanho das rotas geradas pelo RRT e DRRT. Entretanto, para caminhos longos é possível notar uma redução no tamanho das rotas. A presença do RRT* na fusão que originou o método DRRT* contribuiu para a queda no tamanho desses caminhos. Novamente, o método Direct-DRRT* apresentou-se como a melhor solução dentre os casos avaliados, já que o tamanho dos caminhos

Figura 54 – Resultado do Planejador Deliberativo: Tamanho Médio dos Caminhos.



Fonte: Elaborado pelo Autor

encontrados foi o menor.

Ressalta-se que o tamanho médio do caminho encontrado pelo Direct-DRRT* é melhor que o RRT* para o primeiro caminho encontrado entre todas as 3 Simulações. O modo de trabalho do RRT* continua a amostrar mesmo após o primeiro caminho ser encontrado com o objetivo de melhora-lo. Contudo, caso as amostragens continuassem o tempo computacional aumentaria fugindo dos objetivos deste trabalho. Além disso, o objetivo dos planejadores implementados aqui, é encontrar o primeiro caminho e posteriormente inicializar a missão.

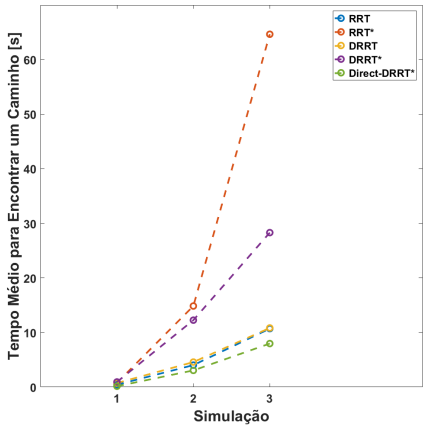
É importante observar o tempo necessário para encontrar os resultados. Pois, as melhorias no método podem estar associadas a um alto custo computacional, ocasionando maiores medidas de tempo para a concepção de um caminho. A Tabela 4 refere-se ao tempo médio para encontrar uma rota através dos planejadores deliberativos. Os gráficos referente a esses resultados foram apresentados na Figura 55.

Tabela 4 – Tempo Médio.

Simulação	**RRT** $[s]$	**RRT*** $[s]$	**DRRT** $[s]$	**DRRT*** $[s]$	**Direct-DRRT*** $[s]$
1	0,43	0,79	0,74	0,94	0,15
2	4,02	14,85	4,58	12,26	3,05
3	10,71	64,65	10,83	39,05	7,97

Ao analisar a Tabela 4 é possível perceber o alto custo do método RRT* para encontrar caminhos melhores. Conseguiu-se uma queda no tempo para obter um caminho durante a fusão DRRT* em comparação com os tempos obtidos pelo RRT*. Além disso, percebe-se que a adição da heurística DRRT aumenta o tempo para obter caminhos quando comparado ao algoritmo clássico RRT. Por fim, a técnica proposta Direct-DRRT* obteve

Figura 55 – Resultado do Planejador Deliberativo: Tempo Médio para Obter um caminho.



Fonte: Elaborado pelo Autor

o menor índice de tempo médio para as três simulações distintas podendo ser considerada a mais interessante ao analisar o aspecto tempo.

Os resultados apresentados pelas tabelas anteriores mostram que o Direct-DRRT* seria a melhor escolha entre os métodos estudados. Obteve-se uma menor quantidade de nós e de iterações para obter caminhos mais curtos em um menor espaço de tempo.

A Figura 56 representa os resultados para cada um dos planejadores implementados onde é possível notar algumas de suas características. Foi adotado a Simulação 3 para extrair as Figuras. As linhas contínuas vermelhas correspondem a conexão entre os nós que pertencem aos caminhos encontrados.

Na Figura 57 é possível perceber a quantidade de amostras (pontos verdes) que foram necessárias para obter um caminho nos 5 planejadores deliberativos testados. Nota-se que com a inserção do DRRT a quantidade de amostras são menores quando comparados ao RRT e RRT*. Entretanto, o método Direct-DRRT* apresenta a menor quantidade de amostragens necessárias para se chegar em um caminho.

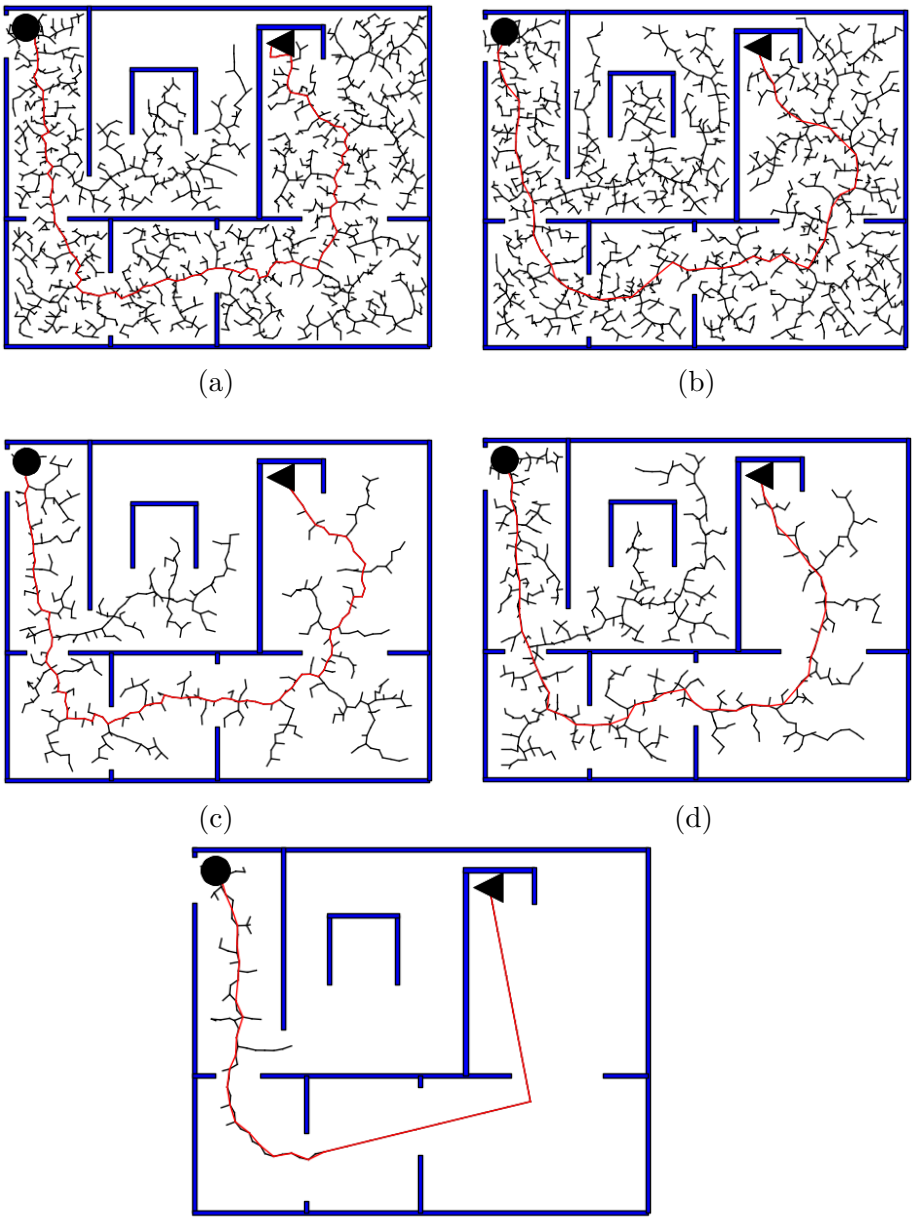
A análise da dispersão estatística dos testes realizados para os planejadores deliberativos poderão ser vistos a partir de histogramas apresentados no Anexo A.

Entre todas as figuras nota-se a característica direta do Direct-DRRT* sendo importante para obter um caminho. Na imagem que corresponde ao planejamento através do RRT nota-se uma grande quantidade de nós para encontrar um caminho.

4.2 RESULTADOS DAS MISSÕES AUTÔNOMAS SIMULADAS

Os primeiros resultados obtidos para a missão autônoma foram encontrados a partir de simulações. Utilizou-se a plataforma de Simulação Gazebo 7 [120] que provê uma

Figura 56 – Resultados dos Planejadores Implementados: (a) Resultado Relativo ao RRT (b) Resultado Relativo ao RRT* (c) Resultado Relativo ao DRRT (d) Resultado Relativo ao DRRT* (e) Resultado Relativo ao Direct-DRRT*

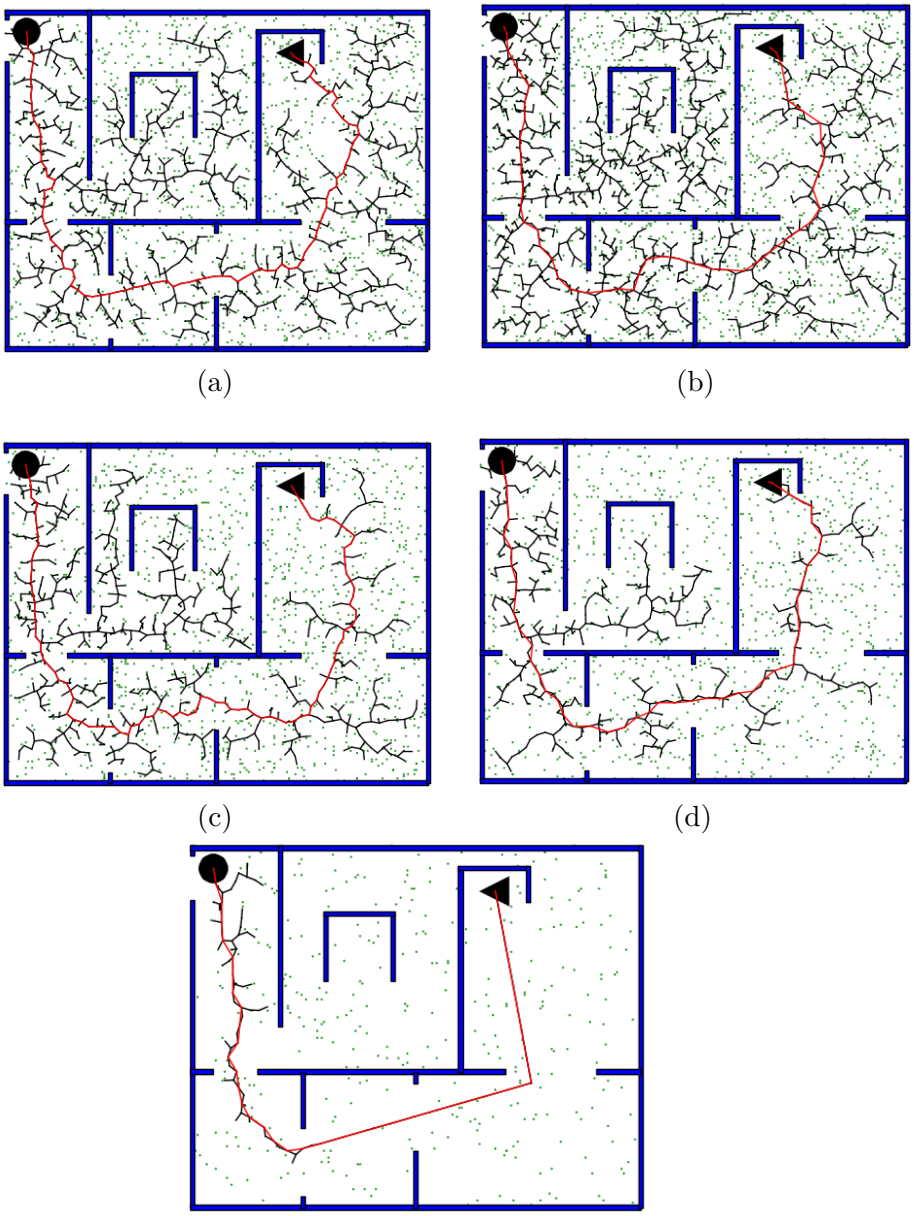


(a) (b) (c) (d) (e)

grande quantidades de ferramentas que aproximam as situações simuladas da realidade. O Gazebo tem uma *interface* que possibilita a conexão com o ROS que torna interessante em diversos trabalhos relacionados à robótica.

Em 2002 Andrew Howard e Nate Koenig iniciaram o desenvolvimento do Gazebo na USC (*University of Southern California*). A intenção era criar um ambiente de simulação com alta fidelidade com a realidade para utilizar em robótica [121]. No ano de 2009 o simulador recebeu a comunicação com o ROS e em 2011 o *Willow Garage* passou a financiar o projeto. Em 2013 o desafio DARPA utilizou o ambiente simulado no Gazebo

Figura 57 – Resultados dos Planejadores Implementados: (a) Resultado Relativo ao RRT (b) Resultado Relativo ao RRT* (c) Resultado Relativo ao DRRT (d) Resultado Relativo ao DRRT* (e) Resultado Relativo ao Direct-DRRT*



(a) (b) (c) (d) (e)

para realizar suas competições. Atualmente o Gazebo é um simulador *open source* em desenvolvimento pela OSRF, que é a mesma entidade que cuida do *framework* ROS [122].

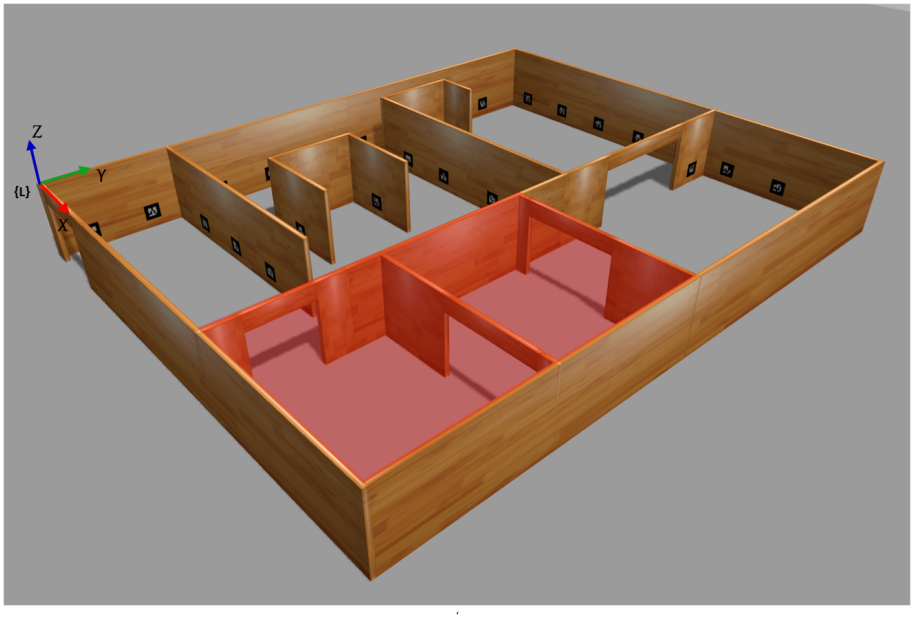
Dentre os elementos principais que compõe o Gazebo destacam-se os *Plugins*. Esses componentes são os códigos que emulam os sensores reais. A partir desses elementos é possível controlar ruídos e os parâmetros que cercam os sensores. O propósito geral dos *Plugins* é a reutilização dos mesmos, já que um mesmo sensor pode ser utilizado em diversos robôs. Por exemplo, cita-se o *Plugin* do sonar onde é possível configurar os parâmetros de distâncias mínimas e máximas de percepção de obstáculos, ruídos, etc.

Nesse trabalho foi utilizado os seguintes *Plugins*: odometria, sonar e o *Openni2*.

A simulação em trabalhos de robótica é de fundamental importância para o sucesso do projeto. A mesma deve estar bem implementada para que as pesquisas possam avançar para trabalhos práticos. Nas simulações, é possível realizar testes e modificações em larga escala com curtos espaços de tempo, fato que não seria possível ao se realizar experimentos práticos. Um simulador que aproxime o máximo possível da realidade tal qual o Gazebo contribui para uma análise mais detalhada das missões autônomas.

Foi construído para esse trabalho, um ambiente interno de dimensões $20m \times 16m$ dividido em cômodos. Por esse ambiente foram espalhados 59 *Ar codes* distintos de formatos quadrados de lado de $50cm$. Para mostrar o comportamento da técnica de localização separou dois cômodos onde não haveria nenhum marcador. Nesse sentido, ao mostrar os resultados, espera-se que a diferença entre a posição real e a estimada amplie, pois, somente a odometria seria a responsável por encontrar a posição. Nos simuladores é possível obter o *Ground Truth* do robô ao longo da missão e compara-lo com a localização dada pela odometria e pelo FKE. Esse termo em robótica corresponde à real posição que o robô ocupa no ambiente dado um referencial global. O computador utilizado foi o mesmo dos testes envolvendo os planejadores deliberativos.

Figura 58 – Ambiente Retratado no Simulador Gazebo.



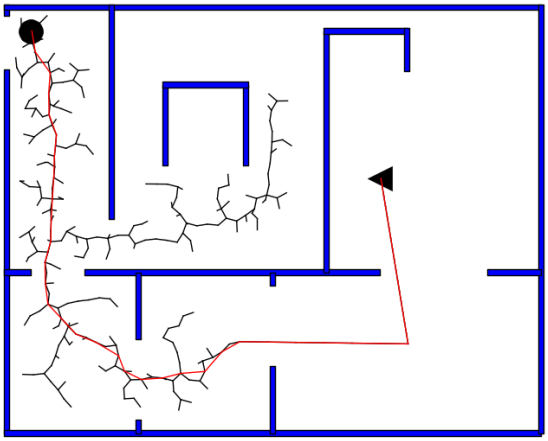
Fonte: Elaborado pelo Autor

As regiões hachuradas de vermelho no ambiente correspondem àquelas que não possuem nenhum *Ar codes* espalhado. A origem do sistema e a orientação dos eixos cartesianos são marcados pela letra **{L}**.

4.2.1 **Missão Autônoma Simulada**

O primeiro passo para realizar a missão autônoma é planejar um caminho de maneira deliberativa. Conforme apresentado na Seção 4.1 o método mais vantajo dos simulados é o Direct-DRRT*. A posição objetivo é definida como $\mathbf{X}_{obj} = [6,5\ 14]$. Esse trabalho não leva em consideração posições angulares nos objetivos. A localização inicial do robô será na posição $\mathbf{X}_{ini} = [1\ 1]$. Vale ressaltar que para manter a realidade na simulação, incorporou-se o arquivo gerado na calibração no *Asus Xtion* do gazebo. A metodologia utilizada para calibrar a câmera é apresentada no Anexo B deste trabalho. O caminho definido pelo Direct-DRRT* é apresentado pela Figura 59.

Figura 59 – Caminho Obtido pelo Direct-DRRT*.



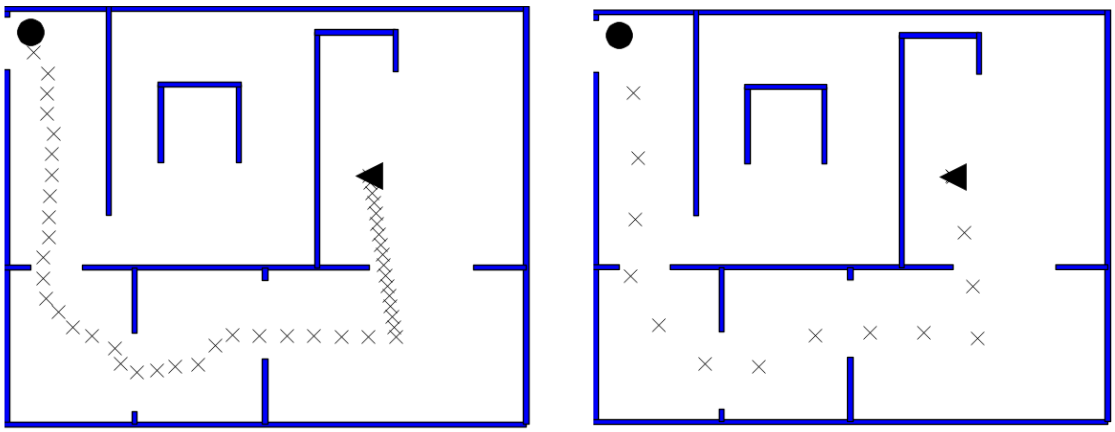
Fonte: Elaborado pelo Autor

Foram necessárias 909 iterações para obter 285 nós ao todo. O caminho encontrado possui aproximadamente $31,75m$ divididos em 46 nós. Após o processo de suavização o caminho que o robô deverá seguir terá apenas 14 nós que corresponderão a 14 sub-objetivos. Os resultados da suavização podem ser vistos na Figura 60.

Em ambientes internos obstáculos surgem de maneira dinâmica com frequência, para representar essa situação, foram espalhadas pelo mapa algumas caixas que pudessem causar colisões com o robô durante a missão. Além de caixas, também foi posicionada uma pessoa com as dimensões e feições de um indivíduo adulto. Então, de maneira proposital, 3 obstáculos dinâmicos foram posicionados exatamente acima de um sub-objetivo gerado pelo planejador. Os obstáculos poderão ser visualizados na Figura 61.

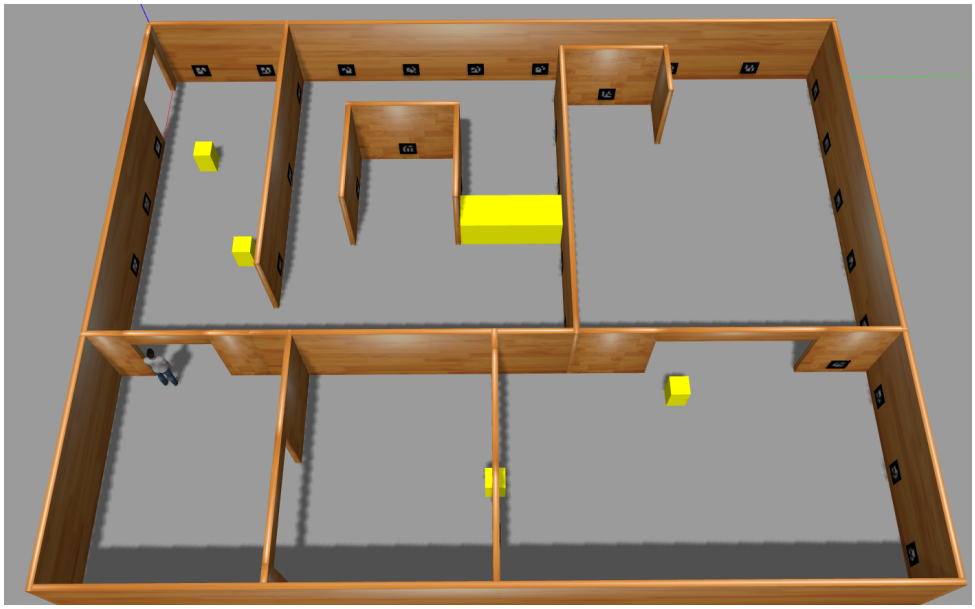
A missão durou aproximadamente 129 segundos totalizando uma distância percorrida de $34,30m$. Se comparar o valor da distância com a que foi obtida pelo Direct-DRRT* percebe-se um aumento, pois, o robô teve a necessidade de desviar de obstáculos aumentando seu percurso. O resultado para a localização via EKF pode ser comparado com a

Figura 60 – Suavização no Resultado do Direct-DRRT*: (a) Caminho Sem Suavização (b) Caminho Suavizado.



(a) (b)

Figura 61 – Obstáculos Dinâmicos pelo Ambiente.



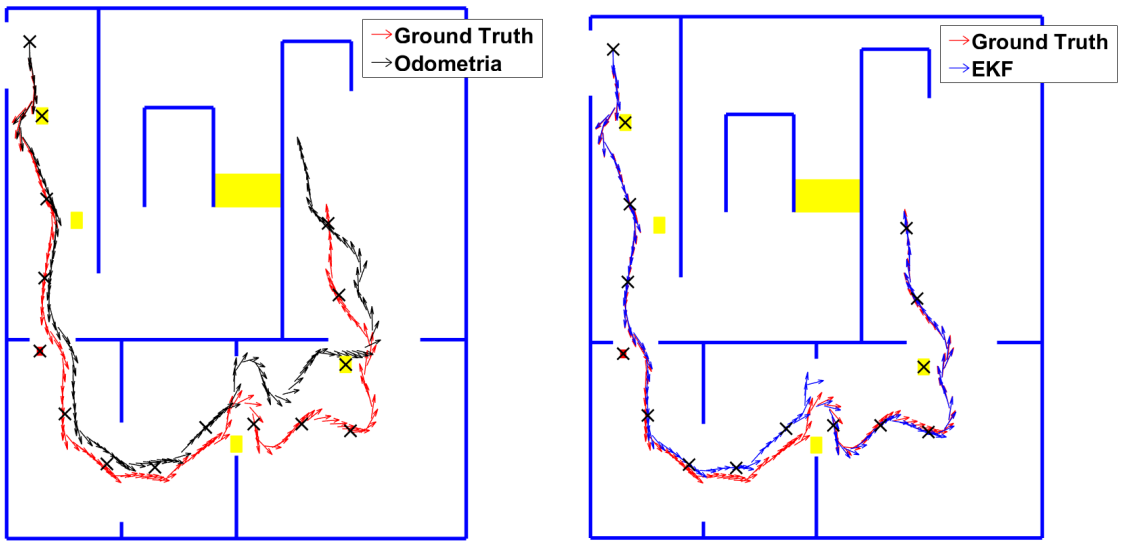
Fonte: Elaborado pelo Autor

posição oriunda da Odometria. Os gráficos com os respectivos resultados são representados pela Figura 62.

A partir dos gráficos é possível perceber o bom comportamento do planejador *online* agindo para desviar dos obstáculos. Além disso, uma localização eficiente permite inferir a posição dos obstáculos dinâmicos espalhados pelo mapa. Dessa maneira, aqueles que estivem exatamente em cima de um sub-objetivo será desconsiderado.

De posse da posição real. é possível encontrar o erro absoluto em x, y e θ para a

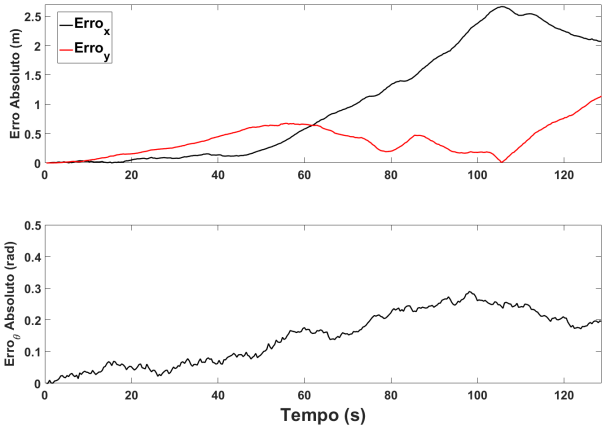
Figura 62 – Localização Durante a Missão Autônoma: (a) Comparação Entre o *Ground Truth* e a Odometria (b) Comparação Entre o *Ground Truth* e o FKE.



(a) (b)

localização via EKF e Odometria. As Figuras 63 e 64 correspondem aos erros de odometria e do FKE, respectivamente.

Figura 63 – Erros Absolutos de Odometria.

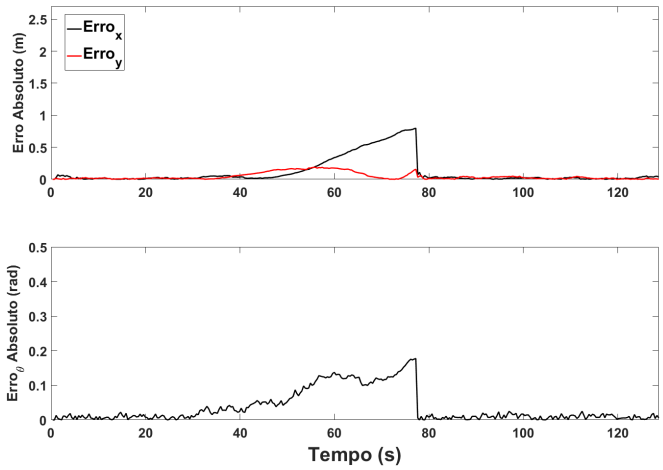


Fonte: Elaborado pelo Autor

Comparando os gráficos dos erros absolutos nota-se que os valores relacionados à odometria são bem superiores aos do Filtro de Kalman Estendido. Para uma melhor avaliação, convém calcular o erro absoluto médio ao longo da missão autônoma. Esses valores médios estão resumidos na Tabela 5.

Analisando a Tabela 5 nota-se a grande diferença do erro médio calculado para a odometria em relação ao FKE. Nesse sentido, mostra-se o quão necessário é a proposta de

Figura 64 – Erros Absolutos do Filtro de Kalman Estendido.



Fonte: Elaborado pelo Autor

Tabela 5 – Erros Absolutos Médios.

Localização	$\mathbf{x_m(m)}$	$\mathbf{y_m(m)}$	$\theta_\mathbf{m}(rad)$
Odometria	1,009	0,381	0,146
Kalman	0,111	0,045	0,036

localização desenvolvida nesse trabalho.

O objetivo final é garantir que o robô chegue a posição desejada, definida para esse problema como $\mathbf{X}_{obj} = [6,5\ 14]$. As posições finais atingidas pelo robô móvel estão sintetizadas na Tabela 6

Tabela 6 – Posições Finais Atingidas.

Localização	$\mathbf{x_f(m)}$	$\mathbf{y_f(m)}$
Ground Truth	6,560	14,010
Odometria	4,780	12,908
Kalman	6,596	14,020

A partir da Tabela 6 é possível afirmar que a estimativa da posição dada pelo Filtro de Kalman Estendido é melhor que a odometria, pois a posição final real é próxima à estimada pelo EKF. Caso fosse utilizado somente a odometria, o robô não obteria êxito em chegar até o objetivo.

Para comprovar a consistência da missão autônoma proposta, realizou-se 30 simulações consecutivas. Para cada um dos 14 sub-objetivos, armazenou a posição estimada pelo Filtro de Kalman Estendido, pela odometria e a posição real. Dessa maneira, calculou-se as distâncias do EKF (Nomeado de Dist.1) e da Odometria (Nomeado de Dist.2) para a posição real (*Ground Truth*). A média das posições obtidas para cada um dos métodos

foram alocados na Tabela 7. Em todas as missões simuladas, houve a convergência do algoritmo.

Tabela 7 – Análise das Médias das Posições Para Múltiplas Missões.

	Pos. Obj.		EKF		*Ground T.*		Odom.		Dist.	Dist.
	x	y	x	y	x	y	x	y	1	2
Início	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,00
Sub. 1	3,25	1,55	-	-	-	-	-	-	-	-
Sub. 2	5.75	1,75	5.63	1,71	5.62	1,69	5.61	1,68	0,05	0,16
Sub. 3	8,15	1,65	8,03	1,66	8,03	1,65	7,98	1,63	0,05	0,23
Sub. 4	10,35	1,45	-	-	-	-		-	-	-
Sub. 5	12,25	2,55	12,13	2,53	12,12	2,51	12,08	2,48	0,14	0,37
Sub. 6	13,75	4,35	13,68	4,24	13,71	4,19	13,65	4,16	0,20	0,46
Sub. 7	13,85	6,45	13,86	6,31	13,94	6,26	13,85	6,23	0,28	0,52
Sub. 8	12,65	8,65	12,72	8,56	12,74	8,54	12,62	8,52	0,09	0,60
Sub. 9	12,55	10,75	12,44	10,62	12,45	10,61	12,29	10,59	0,05	0,73
Sub. 10	12,55	12,85	12,54	12,71	12,55	12,71	12,36	12,69	0,03	0,88
Sub. 11	12,75	14,95	12,73	14,82	12,74	14,82	12,50	14,78	0,04	1,02
Sub. 12	10,75	14,75	-	-	-	-	-	-	-	-
Sub. 13	8,65	14,45	8,74	14,53	8,75	14,54	8,51	14,48	0,04	1,08
Objetivo	6,50	14,00	6,60	14,00	6,60	14,02	6,37	13,97	0,03	1,18

As linhas que apresentam traços ('-') correspondem aos sub-objetivos que possuem obstáculos dispostos sobre os mesmos. Nesse sentindo, o robô opta por se direcionar até o próximo sub-objetivo. É possível notar que os valores do EKF distanciam dos reais nos sub-objetivos 5, 6 e 7. Essa característica surge quando o robô está navegando nas regiões que não existem *Ar codes*. A distância média da odometria em relação ao *Ground Truth* é superior ao FKE, entretanto, ao analisar as posições médias encontradas pela odometria nota-se valores próximos à posição real. Desse modo, é necessário analisar a dispersão dos resultados encontrados para comprovar a eficiência dos métodos. Na Tabela 8 é possível encontrar os resultados dos desvios padrões para as 30 simulações. A unidade de medida para todos os testes sistemáticos é o metro.

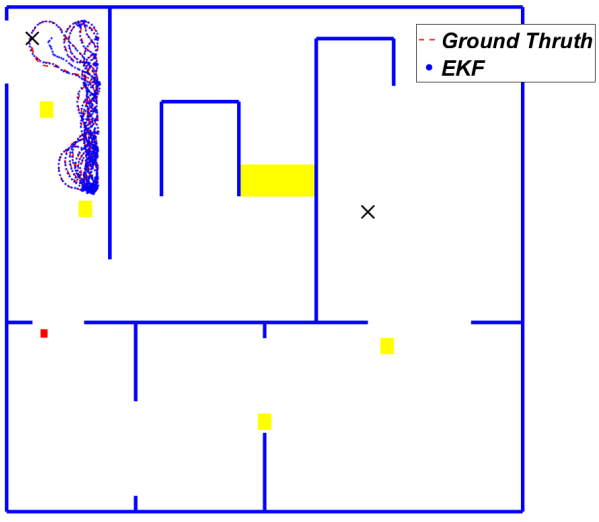
Analisando a Tabela 8, é possível notar que a dispersão das coordenadas dada pela odometria é superior ao método proposto de localização. A dispersão do *Ground Truth* aumenta quando o robô está navegando nas regiões sem *Ar codes*, pois, nesses locais, a localização é dada pela odometria.

Deve-se ressaltar a importância do planejador *offline* para o sucesso da missão. A Figura 65 corresponde à situação em que um planejador deliberativo não é executado previamente. Nesse sentido, definiu-se apenas um único sub-objetivo que ocupa a mesma posição do objetivo final para simular a situação. Ao iniciar a missão o planejador *online* receberá a posição $\mathbf{X_{obj}} = [6,5\ 14]$. Percebe-se que mesmo com uma localização satisfatória o sistema robótico fica preso em uma região bem distante do objetivo.

Tabela 8 – Análise dos Desvios Padrões das Posições Para Múltiplas Missões.

	EKF		*Ground T.*		Odom.		Dist. 1	Dist. 2
	x	y	x	y	x	y		
Início	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Sub. 1	-	-	-	-	-	-	-	-
Sub. 2	0,03	0,03	0,04	0,04	0,14	0,12	0,03	0,09
Sub. 3	0,03	0,01	0,06	0,02	0,17	0,18	0,03	0,16
Sub. 4	-	-	-	-		-	-	-
Sub. 5	0,03	0,02	0,11	0,21	0,36	0,05	0,14	0,37
Sub. 6	0,02	0,03	0,15	0,17	0,27	0,43	0,10	0,32
Sub. 7	0,02	0,03	0,24	0,18	0,37	0,45	0,15	0,35
Sub. 8	0,05	0,04	0,13	0,05	0,54	0,42	0,07	0,38
Sub. 9	0,04	0,03	0,06	0,03	0,72	0,42	0,03	0,46
Sub. 10	0,02	0,02	0,03	0,03	0,92	0,45	0,02	0,57
Sub. 11	0,02	0,03	0,03	0,04	1,16	0,46	0,02	0,72
Sub. 12	-	-	-	-	-	-	-	-
Sub. 13	0,02	0,03	0,03	0,03	1,11	0,56	0,01	0,66
Objetivo	0,03	0,02	0,04	0,03	1,07	0,77	0,02	0,67

Figura 65 – Missão Autônoma sem o Planejamento Deliberativo.



Fonte: Elaborado pelo Autor

No *Link* abaixo encontra-se um video com imagens da missão autônoma simulada:

`https://youtu.be/E74A6NhORgc`

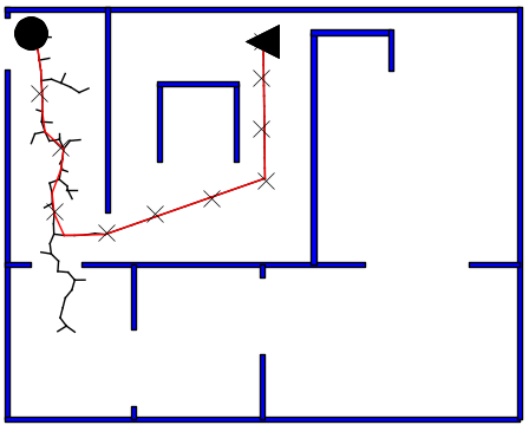
4.2.2 Missão Autônoma Simulada e o Problema de Mínimos Locais

Mesmo com a implementação do planejador deliberativo há hipóteses de ocorrer problemas de mínimos locais. Esses problemas surgem quando os obstáculos dinâmicos

possuem tamanho suficiente para bloquear uma passagem, fazendo com que o atual conjunto de sub-objetivos não seja possível.

Para visualizar essa situação, definiu-se como o ambiente de missão o mesmo mapa apresentado na Figura 52 (Seção 4.2). Os obstáculos dinâmicos estão posicionados nos mesmos locais. A posição inicial foi mantida e o objetivo foi definido em $X_{obj} = [1,35\ 10]$. O resultado para o planejamento deliberativo utilizando o Direct-DRRT* obteve 11 sub-objetivos com um comprimento total de $21,81m$. O resultado do planejamento bem como os sub-objetivos demarcados com **x** são apresentados na Figura 66.

Figura 66 – Caminho Gerado pelo Direct-DRRT*.



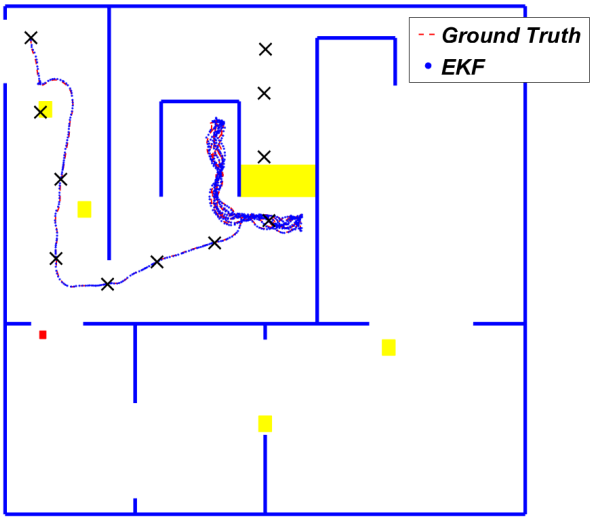
Fonte: Elaborado pelo Autor

Como mencionado no capítulo de metodologia deste trabalho, em alguns casos, um mapeamento será necessário para que os obstáculos que causam os problemas de mínimos loais sejam representados no mapa. Primeiramente realizou-se a missão sem a técnica do mapeamento para demonstrar a necessidade da mesma. A Figura 67 corresponde à esse resultado.

O novo obstáculo junto com a disposição das paredes do ambiente causou o problema de mínimos locais. Dessa maneira, é importante que aja um mapeamento enquanto o robô se desloca pelo ambiente. A partir do mapa atualizado, um novo planejador deliberativo será ativado quando o problema de mínimos locais surgir. Essa questão é facilmente detectada quanto o tempo de deslocamento entre dois sub-objetivos for maior que um valor pré-definido.

Quando o problema de mínimos locais surgir, a missão será paralisada e a atualização do mapa é executada. Antes da atualização, os pontos detectados deverão ser analisados se correspondem à algum obstáculo já pertencente ao mapa. Essa tarefa pode ser considerada uma pré-filtragem e ajuda a eliminar uma grande quantidade pontos, aumentando o desempenho computacional no processo de mapeamento. Depois haverá a clusterização

Figura 67 – Problema de Mínimos Locais.



Fonte: Elaborado pelo Autor

mencionada na metodologia, a filtragem dos grupos gerados pela clusterização, e por fim os pontos remanescentes são transformados em novos obstáculos que irão atualizar o mapa.

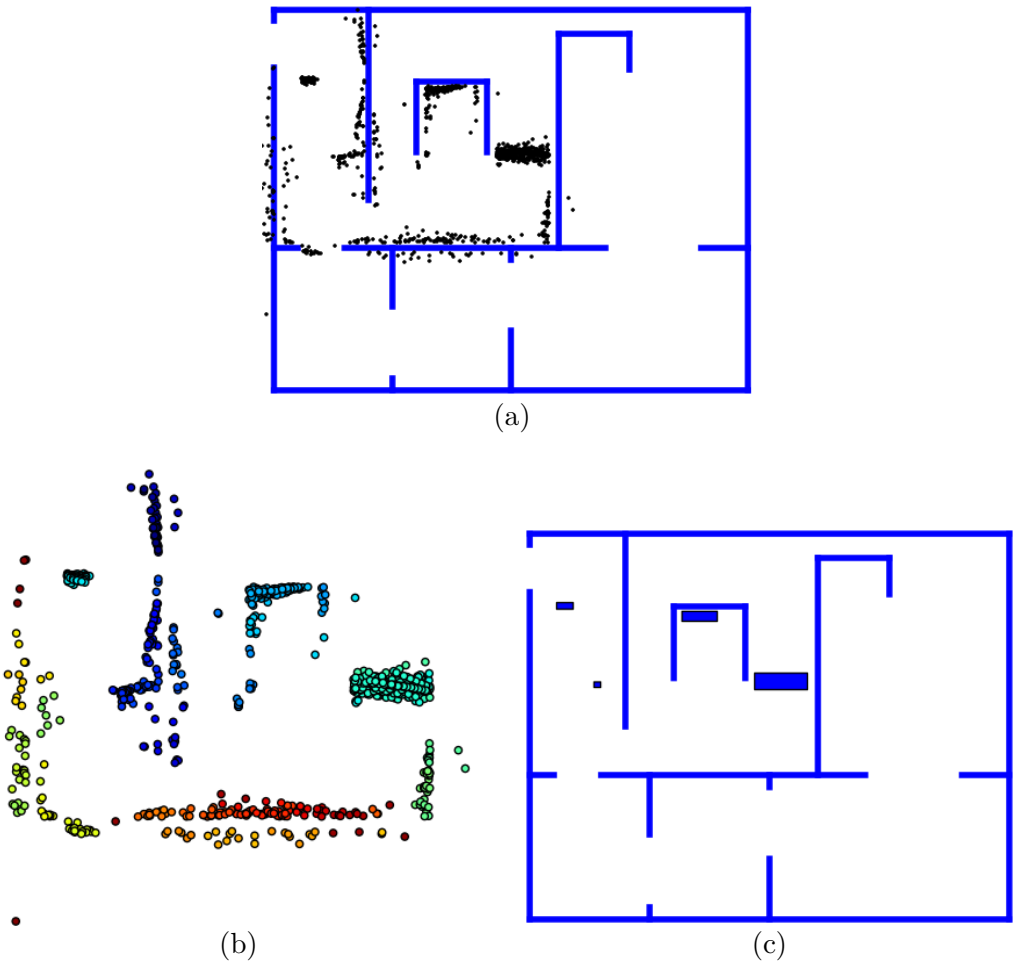
Após o mapa ser atualizado, deve-se executar novamente o planejador Direct-DRRT*. A posição inicial corresponde à última estimada pelo *EKF*, enquanto que a final permanece a mesma.

Primeiramente será mostrado o mapeamento até o momento em que a missão é paralisada dado o problema de mínimos locais cujo resultado está presente na Figura 68.

Analisando a Figura 68 (b), nota-se que alguns conjuntos gerados pela clusterização não gerou obstáculos no mapa atualizado (Figura 68 (c)). Isso se deve ao fato que uma filtragem ocorre nos pontos após o processo de transformação em *clusters*. A *clusterização* separou por cores os conjuntos gerados. Visualmente alguns conglomerados de pontos parecem possuir a mesma cor, entretanto, pode-se afirmar que possuem cores diferentes. A filtragem define como os pontos pertencentes à um obstáculo, aqueles que possuem em seu conjunto de *cluster* mais de 20 observações. Esse valor deverá ser ajustado conforme a rigidez no mapeamento que o projetista queira. Conjuntos com poucos pontos podem ser erros gerados devido à estimativa de posições erradas ou obstáculos pequenos que tiveram observações consideráveis. Por esse motivo vários conjuntos serão excluídos. Assim, pode-se afirmar que um obstáculo entrará no mapa caso o sensor *Asus Xtion* tenha visualizado-o diversas vezes.

Outro ponto que merece destaque é que para o obstáculo ser representado em suas dimensões reais é necessário que observações sejam realizadas em todo o perímetro do obstáculo. Nesse sentido os bloqueios que foram acrescentados ao mapa, não refletem o

Figura 68 – Mapeamento Até a Paralisação da Missão: (a) Obstáculos Detectados (b) Clusterização (c) Mapa Atualizado.



(a) (b) (c)

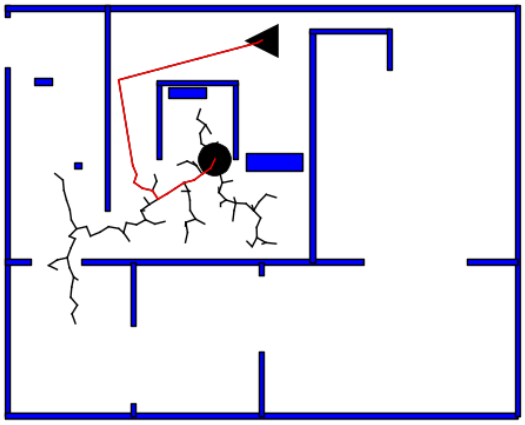
tamanho real, porém, a parte representada é suficiente para que o robô encontre uma nova rota para chegar até o ponto objetivo.

Após o mapeamento é necessário replanejar a missão de maneira deliberativa através do método Direct-DRRT*. Assim será possível encontrar um novo conjunto de sub-objetivos para chegar até o objetivo cuja representação encontra-se na Figura 69.

Em determinados casos, o mapeamento pode definir como obstáculos, alguns pontos que foram definidos graças aos erros na localização. No resultado apresentado pela Figura 69 encontra-se um pequeno obstáculo que não existe na realidade. Esse problema ocorreu por erros no estimativa da localização. Dentro do pequeno ambiente, o robô realizou diversas curvas cujo os efeitos dos ruídos do *encoder* na coordenada θ são maiores dos que na coordenada x e y.

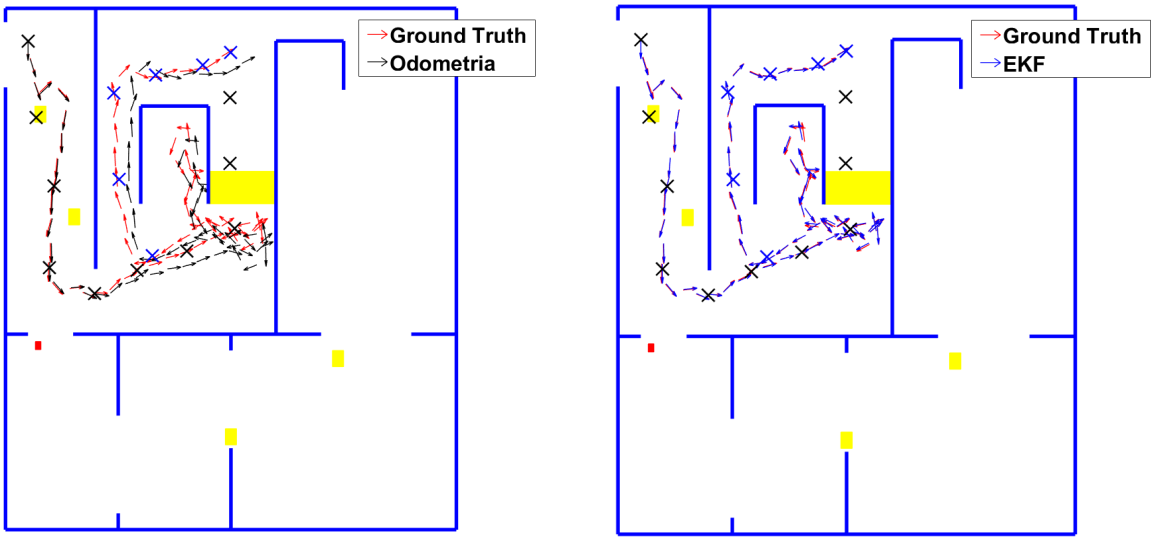
O caminho percorrido bem como a localização dada pela odometria e pelo método proposto baseado no Filtro de Kalman Estendido poderão ser vistos na Figura 70. A missão durou aproximadamente 196 segundos.

Figura 69 – Planejamento Deliberativo Após a Atualização do Mapa.



Fonte: Elaborado pelo Autor

Figura 70 – Localização Durante a Missão Autônoma com Problemas de Mínimos Locais: (a) Comparação entre o *Ground Truth* e a Odometria (b) Comparação entre o *Ground Truth* e o FKE.



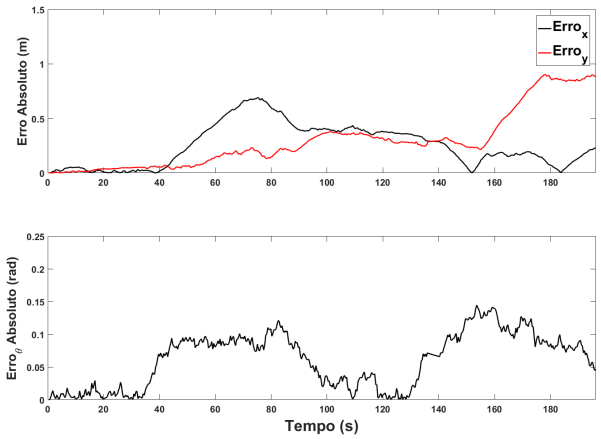
(a) (b)

Os marcadores azuis correspondem aos sub-objetivos encontrados pelo Direct-DRRT* após a atualização do mapa.

Nessa situação também é possível avaliar o comportamento dos erros absolutos entre odometria (Figura 71) e do *EKF* (Figura 72) comparados ao *Ground truth* do Gazebo.

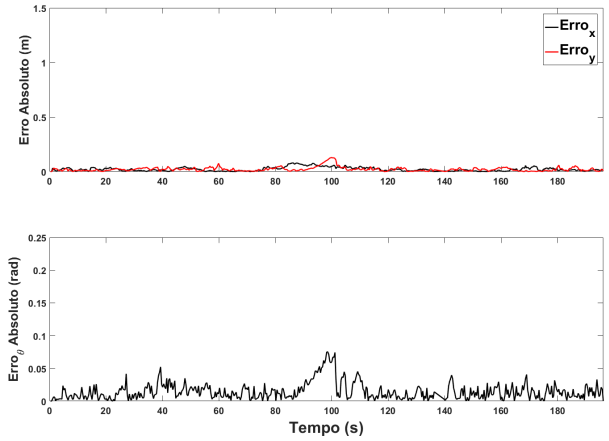
Novamente percebe-se que as curvas relacionadas ao erro absoluto correspondente à localização via odometria atingiram valores superiores quando comparados ao método FKE. A avaliação do erro médio e da posição final dessa missão estão representados nas

Figura 71 – Erros Absolutos de Odometria no Problema de Mínimos Locais.



Fonte: Elaborado pelo Autor

Figura 72 – Erros Absolutos do Filtro de Kalman Estendido no Problema de Mínimos Locais.



Fonte: Elaborado pelo Autor

Tabelas 9 e 10 respectivamente.

Tabela 9 – Erros Absolutos Médios.

Localização	$\mathbf{x_m(m)}$	$\mathbf{y_m(m)}$	$\theta_\mathbf{m}(rad)$
Odometria	0,249	0,291	0,061
Kalman	0,012	0,021	0,015

Os testes sistemáticos para essa avaliação também foram realizados 30 vezes. Entretanto, os resultados relacionados aos sub-objetivos foram suprimidos, pois, como haveria um replanejamento da missão os sub-objetivos iriam modificar. As distância do EKF até o *Ground Truth* é nomeada de Dist. 1, e a distância da odometria para o mesmo *Ground Truth* Dist. 2.

Tabela 10 – Posições Finais Atingidas.

Localização	$\mathbf{x_f(m)}$	$\mathbf{y_f(m)}$
Ground Truth	1,371	9,868
Odometria	1,600	10,759
Kalman	1,382	9,860

Tabela 11 – Análise das Posições Médias Para Múltiplas Missões.

	Pos. Obj.		EKF		*Ground T.*		Odom.		Dist. 1	Dist. 2
	x	y	x	y	x	y	x	y		
Objetivo	1,35	10,00	1,37	9,90	1,37	9,92	1,46	9,51	0,03	1,11

Os desvios padrões relacionados à essas medidas estão presentes na Tabela 12.

Tabela 12 – Análise dos Desvios Padrões das Posições Para Múltiplas Missões.

	EKF		*Ground T.*		Odom.		Dist. 1	Dist. 2
	x	y	x	y	x	y		
Objetivo	0,03	0,03	0,04	0,05	0,64	1,14	0,02	0,82

No *Link* abaixo encontra-se um video contendo imagens da Missão Autônoma e o problema de mínimos locais que ocasiona a perda do robô:

`https://youtu.be/KPqIN9Wm57k`

4.2.3 **Missão Autônoma Simulada e o Sequestro do Robô**

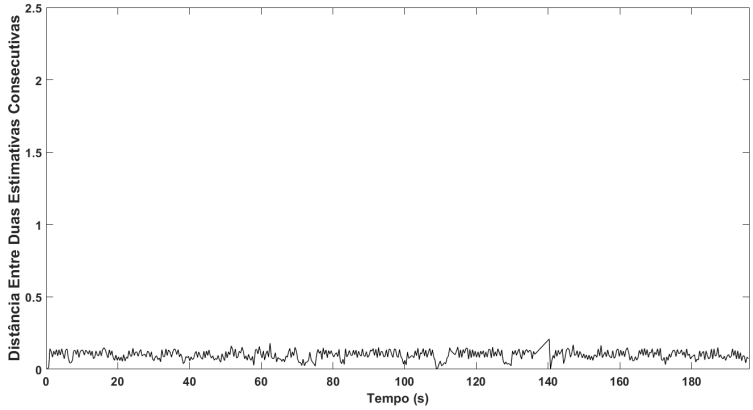
Algumas situações extremas podem surgir ao longo da missão, e podem colocar em risco os objetivos do robô. Um dos problemas é o sequestro do robô, que pode ser definido como o deslocamento do sistema robótico realizado por um agente externo de maneira arbitrária [118]. Além disso, caso houver falhas nos sensores que compõe o sistema robótico, a localização poderá ser diretamente afetada. O *enconder* não é sensível a esses problemas, portanto, o processo de localização será afetado.

Entretanto, o método de localização proposto poderá corrigir a posição estimada a medida que os marcadores espalhados pelo mapa vão sendo observados. Para representar essa situação, definiu-se que o objetivo seria o mesmo da Seção 4.2.1 ($X_{obj} = [6,5\ 14]$). Em uma posição aleatória, o robô será sequestrado para a pose $X_{seq} = [11\ 19\ 0]$. Foi considerado que o conjunto de sub-objetivos gerados pelo planejador deliberativo fosse o mesmo utilizado no resultado da navegação autônoma.

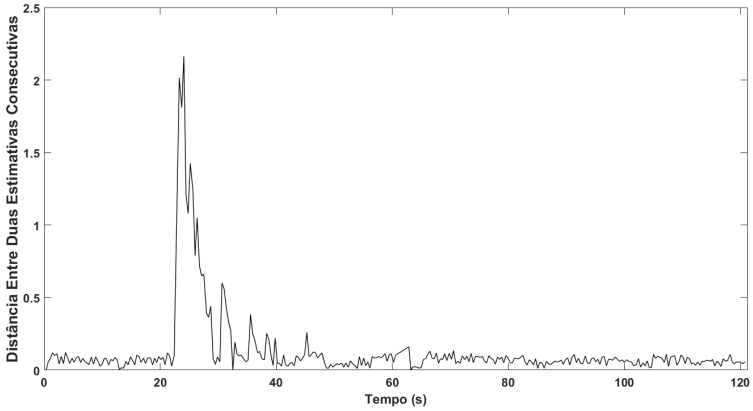
Neste trabalho é possível reconhecer um sequestro quando a estimativa obtida na iteração atual se encontra distante da encontrada pela iteração anterior por um limitante δ. Esse limitador deverá ser definido pelo projetista conforme as suas necessidades. Definindo um δ baixo, correções de odometria poderão ser considerada sequestros, e para valores

alto, os sequestros poderão não ser detectados. Essa situação pode ser vista nas Figuras 73 (a) e (b).

Figura 73 – Distâncias entre duas Posições Estimadas: (a) Missão Sem a Ocorrência do Sequestro (b) Missão com a Ocorrência do Sequestro.



(a)



(b)

Para demonstrar a diferença entre as estimativas durante uma missão normal e uma que ocorre o sequestro serão analisadas as distâncias entre duas medidas consecutivas. Nesse exemplo, os dados relacionados à missão normal foram coletados durante a missão em que o robô ficou preso em um mínimo local. Para a situação da Figura 73 (b), coletou-se as distâncias referentes a missão do sequestro que será apresentado a seguir. Ressalta-se que a capacidade de detectar sequestros está ligada ao EKF, que por sua vez, necessita de visualizar *Ar codes* para melhorar a estimativa da localização.

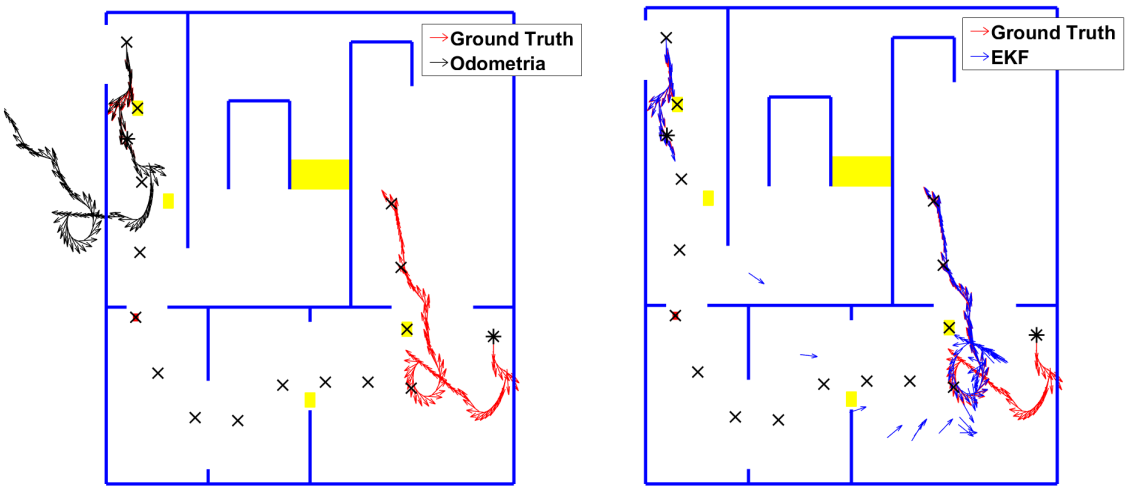
Em uma missão autônoma sem o problema de sequestro (Figura 73 (a)) as distâncias entre duas medidas consecutivas possuem valores próximos. Percebe-se que na Figura 73 (b) as distâncias entre duas posições cresce abruptamente ao longo do sequestro que pode ser visto próximo ao tempo de 20 segundos. Devido às baixas velocidades aplicadas ao robô, a diferença entre duas posições não poderão ter valores altos. Quando situações assim ocorrem podem ter dois significados: o robô está saindo de uma zona onde não há

Ar codes e ficou por muito tempo navegando nela; ou o robô foi sequestrado. Deve-se ressaltar que a primeira situação pode ser considerada como um sequestro e poderá exigir a necessidade da concepção de um novo conjunto de sub-objetivos (replanejamento).

Após o sequestro ser detectado pelo sistema, o mapeamento é desabilitado e uma contagem regressiva é iniciada para que a missão paralise. A contagem tem o objetivo de melhorar a estimativa da posição, já que a localização através do Filtro de Kalman Estendido não corrige a posição instantaneamente. O conjunto inicial de sub-objetivos não deve ser considerado após o sequestro, dessa maneira, um novo planejador deliberativo é executado após ao final da contagem regressiva.

Para o caso do sequestro estudado por esse trabalho, a localização através da odometria e do EKF poderão ser vistas na Figura 74.

Figura 74 – Caminho Percorrido pelo Robô: (a) Comparação entre o *Ground Truth* e a Odometria (b) Comparação entre o *Ground Truth* e o FKE.



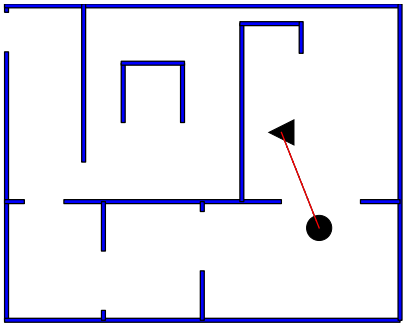
(a) (b)

Percebe-se que a estimativa dada pela odometria não é sensível ao sequestro, nesse sentido, um sistema robótico que utilizasse apenas a odometria teria a missão comprometida. Entretanto, ao utilizar a metodologia de localização proposta, a estimativa da posição é sensível ao sequestro e corrige a localização a ponto de conseguir chegar até a posição final. Marcados por um "X"em preto estão os sub-objetivos do planejamento inicial e em azul estão os que foram gerados pelo re-planejamento da missão que poderá ser visto na Figura 75.

Na Figura 75 percebe-se que o caminho encontrado é o ótimo, pois, conforme apresentado na fundamentação teórica do método Direct-RRT, os casos em que não existam obstáculos mapeados entre a posição inicial e final deverá haver.

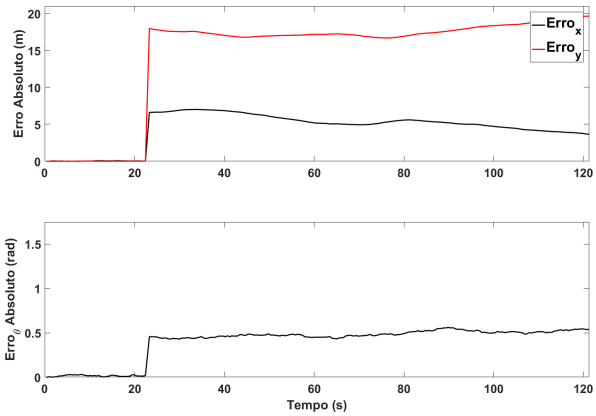
Os erros absolutos para a odometria e a localização através do EKF poderão ser visto nas Figura 76 e 77.

Figura 75 – Replanejamento após o Sequestro através do Direct-DRRT*.



Fonte: Elaborado pelo Autor

Figura 76 – Erros Absolutos de Odometria no Problema de Sequestro de Robôs.



Fonte: Elaborado pelo Autor

Ao analisar os gráficos correspondentes aos erros para o sequestro do robô, percebe-se o crescimento abrupto do erro absoluto no momento da localização. O erro da localização via Filtro de Kalman Estendido consegue reduzir o erro ao longo da missão, diferente para o caso da estimativa da posição através da odometria.

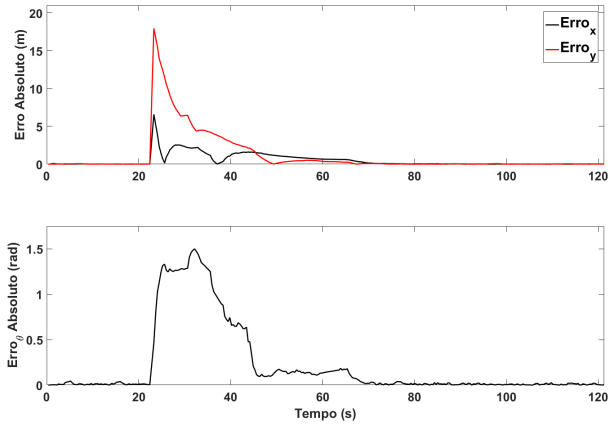
As tabelas referentes ao erro médio absoluto e das posições finais poderão ser encontradas em 13 e 14.

Tabela 13 – Erros Absolutos Médios.

Localização	$\mathbf{x_m(m)}$	$\mathbf{y_m(m)}$	$\theta_\mathbf{m}(rad)$
Odometria	4,406	14,303	0,398
Kalman	0,490	1,139	0,219

O erro médio absoluto para a odometria foi visivelmente superior ao dado pelo FKE. Na Tabela 14, a odometria encontra-se distante da posição final real atingida. O

Figura 77 – Erros Absolutos do Filtro de Kalman Estendido no Problema de Sequestro de Robôs.



Fonte: Elaborado pelo Autor

Tabela 14 – Posições Finais Atingidas.

Localização	$\mathbf{x_f}$**(m)**	$\mathbf{y_f}$**(m)**
Ground Truth	6,597	14,018
Odometria	2,945	-5.616
Kalman	6,585	14,050

EKF conseguiu contornar o problema do sequestro e levar o robô até a posição final previamente definida.

Um ponto importante que deve ser frisado é a necessidade da câmera encontrar *Ar codes* em seu campo de visão após o sequestro. A retomada de uma estimativa de posição satisfatória está diretamente ligada ao fato de existirem marcadores próximos ao local para onde o robô foi sequestrado. Caso o sequestro ocorresse para os cômodos sem *Ar codes* o sucesso da missão estaria comprometido, pois, a única estimativa da posição seria a odometria.

Para esse caso, também realizou-se simulações sistemáticas para comprovar a eficácia do método. Em todas as 30 simulações simuladas a missão autônoma convergiu por encontrar uma estimativa de posição próximo ao objetivo. Também para esse caso, optou-se por analisar somente a posição final em que o robô atingiu devido ao replanejamento que haveria ao longo da missão. Esses resultados poderão ser analisados na Tabela 15.

Tabela 15 – Análise das Posições Médias Para Múltiplas Missões.

	Pos. Obj.		EKF		*Ground T.*		Odom.		Dist. 1	Dist. 2
	x	y	x	y	x	y	x	y		
Objetivo	6,50	14,00	6,56	14,01	6,55	14,02	2,77	-4,52	0,06	18,97

Para esse caso, também é importante analisar como se comporta a dispersão

dos dados apresentados na tabela anterior, cujos desvios padrões foram sintetizados na Tabela 16.

Tabela 16 – Análise dos Desvios Padrões das Posições Para Múltiplas Missões.

	EKF		*Ground T.*		Odom.		Dist. 1	Dist. 2
	x	y	x	y	x	y		
Objetivo	0,06	0,10	0,07	0,11	1,25	1,18	0,04	0,97

No *Link* abaixo encontra-se um video contendo imagens da Missão Autônoma e o problema do sequestro do robô:

`https://youtu.be/KPqIN9Wm57k`

4.3 RESULTADOS DAS MISSÕES AUTÔNOMAS EXPERIMENTAIS

Após realizar as missões autônomas de maneira simulada, foi necessário fazer experimentos práticos para que a metodologia proposta seja consolidada. A prática foi realizada em uma sala de aula do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora (PPEE) de maneira adaptada para receber os testes conforme apresentado pela Figura 78 (a) e (b).

A missão autônoma consiste em deslocar o robô pelo ambiente, partindo de um ponto inicial para atingir um objetivo sem interferência humana. Definiu-se como posição inicial $\mathbf{X}_{ini} = [1, 2, 1, 5]$ e final $\mathbf{X}_{fim} = [5, 6, 1]$ marcados por uma circunferência e um triângulo respectivamente na Figura 78 (b).

O planejamento *offline* da missão foi realizado pelo Direct-DRRT* apresentado anteriormente. Para encontrar um possível caminho para chegar até o objetivo foi necessária 31 iterações para gerar uma árvore com 13 nós. O tamanho do caminho encontrado foi de $6,29m$, cujo tempo necessário para encontra-lo foi de $0,19s$. A rota é um conjunto de 7 sub-objetivos e o resultado do Direct-DRRT* foi apresentado na Figura 79 (a). As três práticas que serão analisadas por esse trabalho terão os mesmos conjuntos de sub-objetivos.

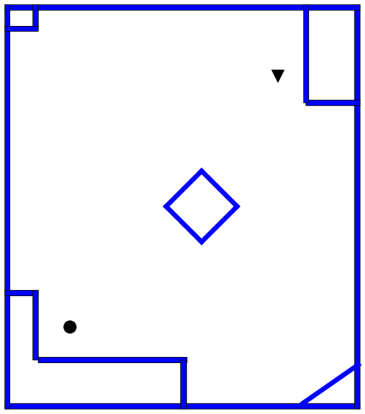
Para comparação realizou-se a mesma simulação utilizando o método DRRT*. Observou-se resultados piores quando comparados ao Direct-DRRT*: 1625 iterações; 145 nós presentes na árvore; Tamanho do caminho de $8,20m$; Tempo necessário de $6,80s$; 8 sub-objetivos. O resultado desse planejador é apresentado na Figura 79 (b). Todos os testes realizados terão o mesmo planejamento deliberativo cujo resultado é apresentado na Figura 79 (a).

4.3.1 **Missão Autônoma**

O primeiro teste prático consiste em deslocar o robô da pose inicial até o ponto objetivo. Colocou-se um obstáculo próximo ao primeiro sub-objetivo para mostrar a

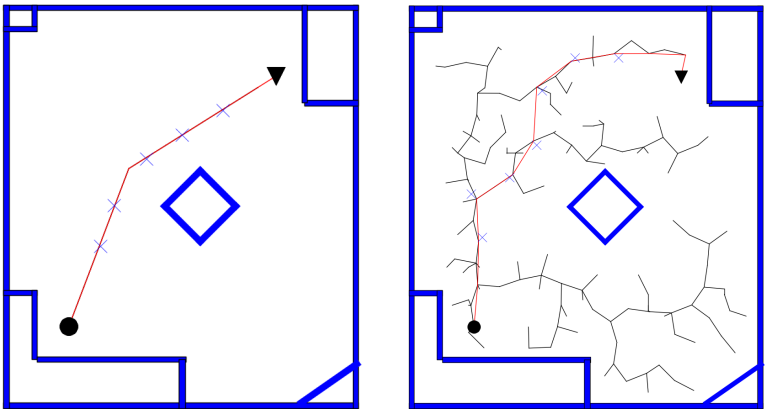
Figura 78 – Ambiente Utilizado no Experimento: (a) Foto Panorâmica do Ambiente (b) Ambiente Mapeado.

(a)



(b)

Figura 79 – Planejamento *Offline*: (a) Método Direct-DRRT* (b) Método DRRT*.

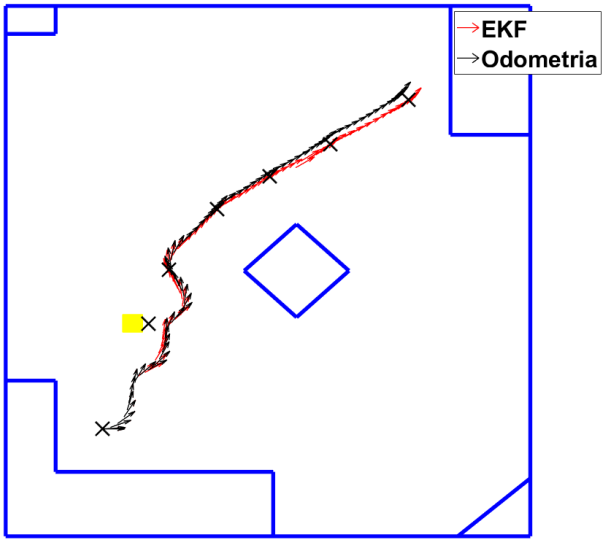


(a) (b)

capacidade do robô desviar de bloqueios que não foram mapeados. Além disso, o robô deverá tomar a decisão que é impossível atingir esse primeiro sub-objetivo, e então deverá prosseguir até o próximo dando continuidade à missão.

As posições estimadas pelo Filtro de Kalman Estendido e a Odometria poderão ser analisados na Figura 80.

Figura 80 – Caminho Prático de Uma Missão Autônoma.



Fonte: Elaborado pelo Autor

Percebe-se que o Filtro de Kalman Estendido leva o robô até uma posição mais próxima do objetivo quando comparado ao resultado da odometria. Na Tabela 17 corresponde as poses finais para cada um dos métodos.

Tabela 17 – Posições Finais Atingidas.

Localização	$\mathbf{x_f(m)}$	$\mathbf{y_f(m)}$
Odometria	4,837	6,147
Kalman	4,973	6,060
Real	4,960	6,100

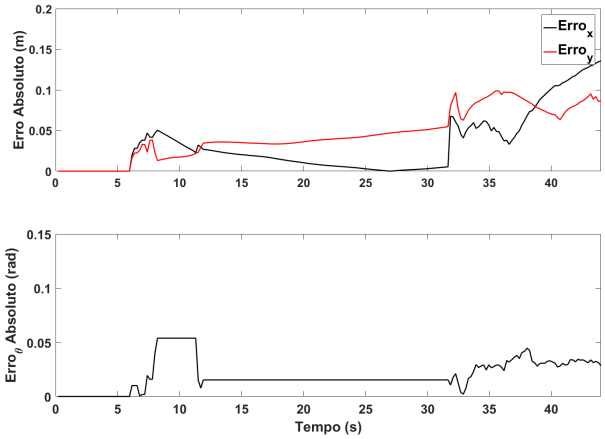
Para os testes práticos realizados, não é possível afirmar com exatidão a posição real em que o robô se encontra ao longo da missão. Somente medir, a posição em que o centro do robô se encontra quando a missão for finalizada. Para confirmar que há uma diferença entre a odometria e a estimada pela EKF é apresentado na Figura 81 o valor absoluto da subtração entre as coordenadas de cada um dos métodos (odometria e EKF).

Mesmo para deslocamentos pequenos é possível observar que há uma diferença entre odometria e o Filtro de Kalman Estendido. Esse resultado seria maior se as missões fossem em ambientes maiores tal qual a simulação realizada.

No *Link* abaixo encontra-se um video contendo imagens do experimento envolvendo a Missão Autônoma:

`https://youtu.be/muZ4PXzHC6I`

Figura 81 – Erro absoluto entre os Métodos EKF e Odometria.



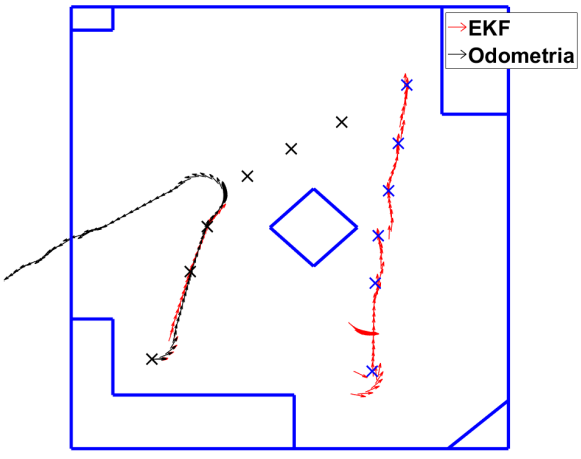
Fonte: Elaborado pelo Autor

4.3.2 Missão Autônoma e o Sequestro do Robô

A segunda missão prática feita, consiste em realizar o sequestro do robô durante a navegação autônoma. Dessa maneira, o robô deverá se localizar e replanejar sua missão. O replanejamento é necessário, pois, o conjunto inicial de sub-objetivos não será ideal para o prosseguimento da missão.

A comparação do resultado obtido a partir do EKF e da Odometria poderão ser analisados na Figura 82. Os "×"correspondem aos sub-objetivos do novo planejamento.

Figura 82 – Caminho Prático de Uma Missão Autônoma e o Sequestro do Robô.



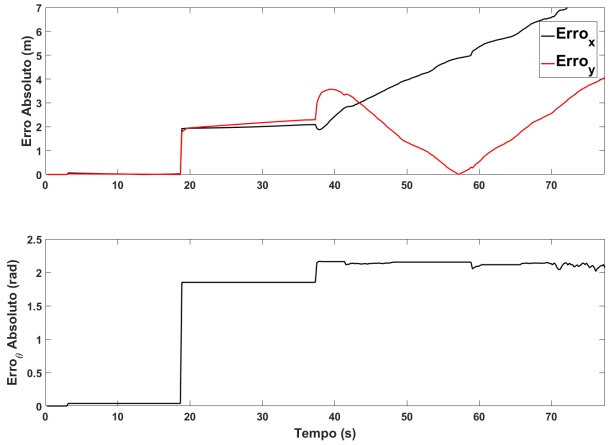
Fonte: Elaborado pelo Autor

Para que o robô não navegue com a localização errada, limitou-se a velocidade máxima linear em $1cm/s$ e a angular em $0,0017rad/s$. Desse modo, a localização seria

corrigida, não havendo riscos para a integridade do robô. Após o replanejamento, a missão seguiria de maneira normal, não existindo mais a limitação da velocidade.

Da mesma maneira que a missão autônoma mostrada anteriormente, não é possível afirmar qual posição real que o robô se encontra. Foi medido apenas a posição final que o robô chegou. Entretanto, é possível mostrar os erros entre a Odometria e o Filtro de Kalman conforme apresentado pela Figura 83.

Figura 83 – Erro absoluto entre os Métodos EKF e Odometria Durante o Sequestro do Robô.



Fonte: Elaborado pelo Autor

Nota-se que a odometria não é sensível ao sequestro, entretanto, o método apresentado detecta esse problema, e guia o robô para uma região próxima ao ponto objetivo a partir de um novo replanejamento. As posições finais atingidas são apresentadas na Tabela 18.

Tabela 18 – Posições Finais Atingidas.

Localização	$\mathbf{x_f(m)}$	$\mathbf{y_f(m)}$
Odometria	-2,68	2,025
Kalman	4,983	6,080
Real	4,950	6,140

O salto no erro é o exato momento em que o Filtro de Kalman tenta corrigir a localização após o sequestro. Dessa maneira, é possível afirmar que o método apresentado consegue detectar o sequestro com êxito. Contudo, a localização exata (*Ground Truth*) do percusso feito com o robô ainda não é possível afirmar.

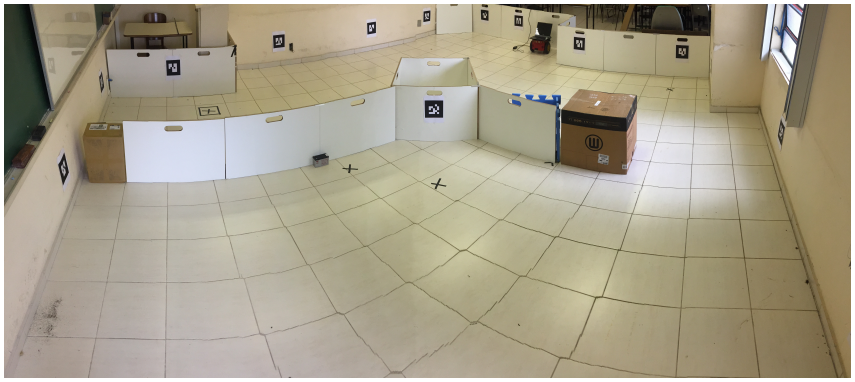
No *Link* abaixo encontra-se um video contendo imagens do experimento envolvendo a missão autônoma e o problema do sequestro de robôs:

`https://youtu.be/lWIRNjyQgaU`

4.3.3 Missão Autônoma e o Problema de Mínimos Locais

O terceiro resultado prático que foi realizado, consiste em alocar obstáculos dinâmicos no mapa para que seja formada uma região de mínimo local. Dessa maneira, o robô deverá verificar o tempo necessário entre um sub-objetivo e outro para que possa ser considerado preso ou perdido. Durante todo o processo, o robô estará mapeando o ambiente para que se for necessário, o mapa seja atualizado.

Figura 84 – Conjunto de Obstáculos Formando um Mínimo Local.



Fonte: Elaborado pelo Autor

Ao ser verificado que o robô está preso em um mínimo local, é realizado a atualização do mapa para que um novo planejamento deliberativo possa acontecer. Um caminho seguro e livre de mínimos locais deverá ser encontrado para que o robô dê prosseguimento com a missão de modo a atingir a posição objetivo.

Da mesma maneira que os outros testes, o caminho percorrido pela Odometria e pelo EKF até que o robô reconheça a necessidade do replanejamento pode ser apresentado Figura 85.

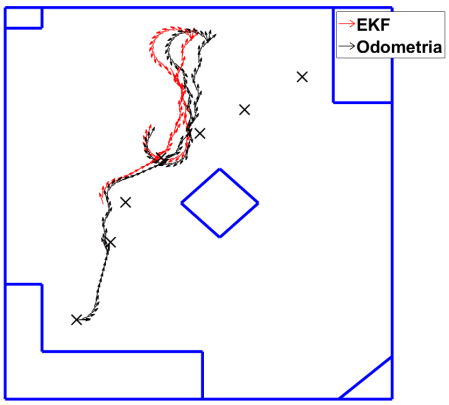
Ao detectar que o robô está perdido, o conjunto de pontos que foram mapeados durante a navegação (Figura 86 (a)) é filtrado e deverá passar por um processo de clusterização (Figura 86 (b)) para que o novo mapa com os novos obstáculos seja definido (Figura 86 (c)).

Com o novo mapa, é possível planejar novamente a missão conforme apresentado pela Figura 87.

Após o novo planejamento, a missão poderá continuar para que o robô se locomova até a posição objetivo. A Figura 88 representa o caminho completo do robô para chegar até a posição objetivo. Os "×"azuis marcados nessa Figura correspondem aos sub-objetivos do novo planejamento.

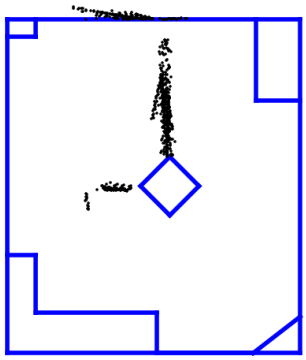
Os erros envolvendo a Odometria e o EKF poderão ser analisados na Figura 89.

Figura 85 – Caminho de Uma Missão Autônoma Prática Parcial e o problema de Mínimos Locais.



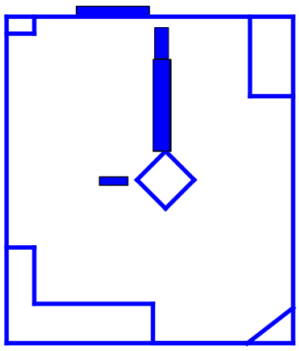
Fonte: Elaborado pelo Autor

Figura 86 – Mapeamento Até a Paralisação da Missão Prática: (a) Obstáculos Detectados (b) Clusterização (c) Mapa Atualizado.



(a)

(b)



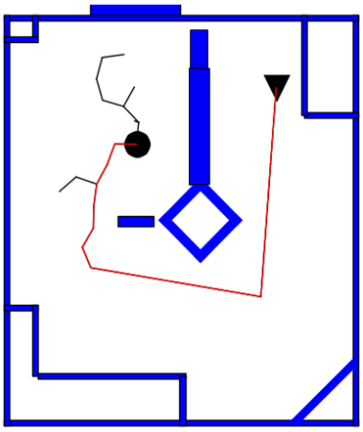
(c)

As posições finais atingidas poderão ser observadas na Tabela 19.

No *Link* abaixo encontra-se um video contendo imagens do experimento envolvendo a missão autônoma e o problema de mínimos locais que ocasiona a perda do robô:

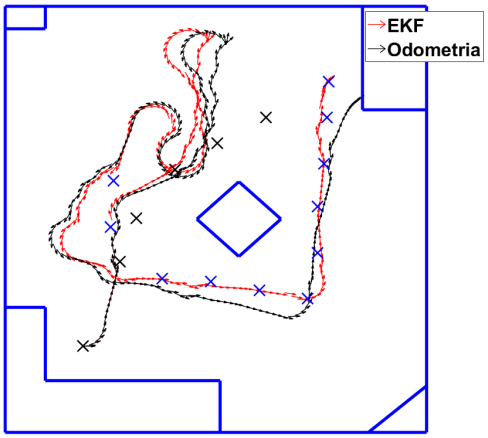
`https://youtu.be/R408gVJGuto`

Figura 87 – Replanejamento da Missão Autônoma.



Fonte: Elaborado pelo Autor

Figura 88 – Missão Autônoma Completa e o problema de Mínimos Locais.



Fonte: Elaborado pelo Autor

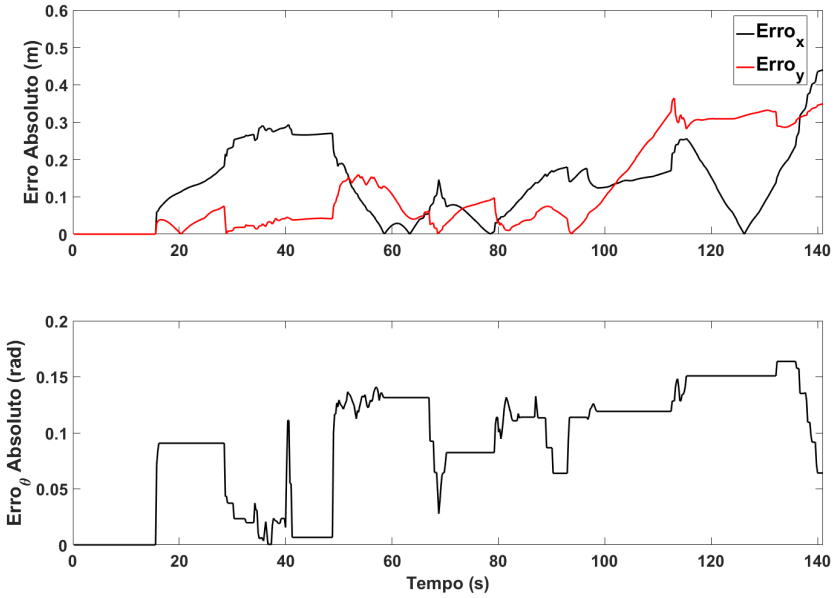
Tabela 19 – Posições Finais Atingidas.

Localização	$\mathbf{x_f(m)}$	$\mathbf{y_f(m)}$
Odometria	5,41	5,73
Kalman	4,97	6,080
Real	5,05	6,220

4.3.4 Problemas Encontrados Durante os Testes Práticos

No decorrer dos testes práticos, foram observados alguns aspectos que poderiam impactar diretamente na localização do sistema robótico. O primeiro deles consiste na adaptação da sala de aula no ambiente de simulação. A posição dos marcadores espalhados no ambiente poderia variar alguns centímetros da posição armazenada no mapa. Essa

Figura 89 – Erros entre Odometria e Filtro de Kalman Estendido.



Fonte: Elaborado pelo Autor

característica observada aconteceu devido aos instrumentos utilizados para medição (Fita Métrica) não serem precisos. Então, a estimativa global do robô com observação do *Asus Xtion* poderia variar.

Os papéis que foram utilizados para imprimir os *Ar codes* também contribuíram para prejudicar as estimativas da posição. Esse material é sensível à temperatura e umidade do ar, dessa maneira, ao longo dos dias em que foram realizados os testes práticos, os papéis irão se deformando. A partir da deformação, a posição angular do marcador em relação à câmera poderia sofrer alterações.

5 Conclusões

Através da metodologia apresentada por este trabalho, foi possível realizar o objetivo principal que corresponde à concepção de missões autônomas. Realizou-se diferentes testes, tanto no ambiente prático quanto no simulado o robô diferencial conseguiu atingir a posição objetivo.

O planejador de caminhos Direct-DRRT* que foi proposto nesta dissertação, obteve resultados melhores que os métodos já consolidados na literatura RRT, RRT* e DRRT. Além desses, o novo planejador conseguiu superar a fusão dos métodos RRT* com o DRRT (DRRT*) para os parâmetros analisados. Dessa modo, optou-se por utiliza-lo para planejar a rota das missões autônomas.

O uso do sonar e do sensor de profundidade *Asus Xtion* aliado ao planejador reativo dos campos potenciais artificiais, foi suficiente para realizar as manobras de contorno em obstáculos dinâmicos. Além de evitar colisões, as medidas encontradas pelo sensor, foram usadas para atualizar o mapa. Dessa maneira, o robô foi capacitado para sair de situações em que o mesmo tivesse perdido. Devido ao campo de abertura da câmera ser pequeno (aproximadamente 60^o), as laterais do sistema robótico ficariam expostas ao contato com obstáculos, por esse motivo os sonares laterais tiveram que ser utilizados.

A visão computacional auxiliou o algoritmo de localização a estimar posições do robô melhores que a odometria. O uso de marcadores artificiais, tornou nulas as hipóteses ambíguas que um ambiente pode oferecer. Sendo assim, o robô recebeu a capacidade de identificar situações de sequestro e consequentemente se relocalizar para dar continuidade à missão. Vale ressaltar que uma boa localização está condicionada a visão dos *Ar codes* que foram espalhados pelo ambiente.

Ao utilizar o sensor de profundidade *Asus Xtion Pro Live*, o trabalho se torna mais viável do ponto de vista econômico ao ser comparado com sensores *Lasers scan*. O dispositivo foi utilizado no planejamento reativo, no mapeamento e na localização das missões autônomas, tendo um papel fundamental para a realização desse trabalho.

O uso da *framework* ROS ajudou o nível organizacional do trabalho a partir da segmentação da missão em nós. Além disso, o ROS possibilitou a implementação paralela da missão autônoma. Pode-se citar a filtragem dos pontos captados pelo sensor de profundidade para a atualização do mapa no decorrer da navegação como tarefas paralelas.

Para a consolidação da metodologia, realizou-se as 3 três situações distintas: missão autônomas, missão autônomas com o problema do sequestro e a missão autônoma com o problema de mínimos locais. Obteve-ve resultados práticos e simulados para cada uma das missões. Em todas, a missão foi concluída com êxito.

Para trabalhos futuros, deve-se direcionar as pesquisas para o controlador do robô.

Verificou-se que em alguns momentos, o movimento apresentava pequenas oscilações. Nesse trabalho, não preocupou-se com o tempo necessário para que o robô cumprisse a missão. Desse modo, estudos envolvendo otimização dos perfis de velocidades aplicados ao robô deverão ser implementados futuramente. Outro modo de auxiliar o controle do robô, é utilizar as velocidades que são impostas ao modelo cinemático no fase de expansão dos planejadores baseados no RRT (*offline*). Assim, antes de iniciar a missão o robô saberá quais as velocidades necessárias para guia-lo até a posição objetivo.

Para o ambiente prático, ressalta-se a necessidade da modificação do material em que os *Ar codes* se encontram, pois, características como temperatura e umidade podem deformar os papéis (utilizados nesse trabalho) interferindo nas medidas de posições.

REFERÊNCIAS

[1] Aníbal Ollero Baturone. *Robótica: manipuladores y robots móviles*. Marcombo, 2005.

[2] Renato Reder Cazangi et al. Sintese de controladores autonomos em robotica movel por meio de computação bio-inspirada. 2008.

[3] Richard D Klafter, Thomas A Chmielewski, and Michael Negin. Robotic engineering: an integrated approach. *Robotic Engineering: An Integrated Approach, by Richard D. Klafter, Thomas A. Chmielewski, and Michael Negin, Englewood Cliffs, New Jersey: Prentice Hall, 1989, 608 p., ISBN 0134687523*, 1989.

[4] Peter Corke. *Introduction*, pages 1–14. Springer International Publishing, Cham, 2017.

[5] J.G.C. Devol. Programmed article transfer, June 13 1961. US Patent 2,988,237.

[6] Rejane Cavalcante Sá. *Construção, modelagem dinâmica e controle PID para estabilidade de um veículo aéreo não tripulado do tipo quadrirotor*. PhD thesis, 2012.

[7] Leandro de Oliveira Silva, Renata Albergaria de Mello Bandeira, Vânia Barcellos Gouvêa Campos, and Jacy Montenegro Magalhães Neto. Tecnologia e aplicações do veículo aéreo não tripulado em ações de logística humanitária.

[8] Jefferson Rodrigo de Souza. *Navegação autônoma para robôs móveis usando aprendizado supervisionado.* PhD thesis, Universidade de São Paulo, 2014.

[9] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.

[10] Rainer Bischoff, Ulrich Huggenberger, and Erwin Prassler. Kuka youbot-a mobile manipulator for research and education. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[11] Rodolfo Rosendo dos Santos and Caio Igor Gonçalves Chinelato. Análise de um manipulador robótico móvel. *REGRASP-Revista para Graduandos/IFSP-Campus São Paulo*, 1(1):28–34, 2016.

[12] Benjamin Jarrett Ebersole. *Skid-steer kinematics for dual-arm mobile manipulator system with dynamic center of gravity*. PhD thesis, 2017.

[13] Shuai Guo, Yi Jin, Sheng Bao, and Feng-Feng Xi. Accuracy analysis of omnidirectional mobile manipulator with mecanum wheels. *Advances in Manufacturing*, 4(4):363–370, 2016.

[14] M Figueiredo. *Navegação Autônoma de Robôs*. Livro da VII ERI (Escola de Informática da SBC), 1999.

[15] Anthony Stentz, John Bares, Thomas Pilarski, and David Stager. The crusher system for autonomous navigation. *AUVSIs Unmanned Systems North America*, 3, 2007.

[16] Denis Fernando Wolf, EV Simões, Fernando S Osório, and Onofre Trindade Junior. Robótica móvel inteligente: Da simulação às aplicações no mundo real. In *Mini-Curso: Jornada de Atualização em Informática (JAI), Congresso da SBC*, page 13, 2009.

[17] Fernando Osório, Denis Wolf, Kalinka Castelo Branco, Jó Ueyama, Gustavo Pessin, Leandro Fernandes, Maurício Dias, Leandro Couto, Daniel Sales, Diogo Correa, et al. Pesquisa e desenvolvimento de robôs táticos para ambientes internos. In *Internal Workshop of INCTSEC, N. I, São Carlos, SP, 2011*, 2011.

[18] Nick Hawes, Chris Burbridge, Ferdian Jovan, Lars Kunze, Bruno Lacerda, Lenka Mudrová, Jay Young, Jeremy Wyatt, Denise Hebesberger, Tobias Körtner, et al. The strands project: Long-term autonomy in everyday environments. *arXiv preprint arXiv:1604.04384*, 2016.

[19] Thomas Breuer, Geovanny R Giorgana Macedo, Ronny Hartanto, Nico Hochgeschwender, Dirk Holz, Frederik Hegger, Zha Jin, Christian Müller, Jan Paulus, Michael Reckhaus, et al. Johnny: An autonomous service robot for domestic environments. *Journal of intelligent & robotic systems*, 66(1-2):245–272, 2012.

[20] Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer. Caesar: an intelligent domestic service robot. *Intelligent Service Robotics*, 5(4):259–273, 2012.

[21] Rodrigo Orihuela. Empresa espanhola fabrica robôs para chão de fábrica. `https://goo.gl/cgkVht`. Acessado em: 01.08.2017.

[22] Guilherme Leal Santos. Localização de robôs móveis autônomos utilizando fusão sensorial de odometria e visão monocular. Master's thesis, Universidade Federal do Rio Grande do Norte, 2010.

[23] G Dudek and M Jenkin. Computational principles of mobile robotics–cambridge univ. press, 2000.

[24] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1):113–126, 2001.

[25] Stephanie Kamarry Alves de Sousa et al. Planejamento de movimento para robôs móveis baseado em uma representação compacta da rapidly-exploring random tree (rrt). 2017.

[26] Ronald C Arkin. *Behavior-based robotics*. MIT press, 1998.

[27] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[28] Daniel Leite, Karla Figueiredo, and Marley Vellasco. *Localização por Kalman Estendido Aplicado a Mapas Baseados em Marcos Com e Sem Correspondência Conhecida*. XII Simpósio Brasileiro de Automação Inteligente (SBAI), 2015.

[29] Zvi Shiller. Off-line and on-line trajectory planning. In *Motion and Operation Planning of Robotic Systems*, pages 29–62. Springer, 2015.

[30] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[31] R Bellman. Dynamic programming. princeton (nj): Princeton univ, 1957.

[32] Devin Connell and Hung Manh La. Dynamic path planning and replanning for mobile robots using rrt. *arXiv preprint arXiv:1704.04585*, 2017.

[33] Caio Júlio César do Vale Silva et al. Planejamento de trajetórias e navegação de robôs móveis. Master's thesis, Universidade Federal do Rio Grande do Norte, 2015.

[34] Ivo Paixao de Medeiros, Luciano Buonocore, and Cairo Lúcio Nascimento Júnior. Localização de robô móvel baseada em filtro de kalman estendido usando um sonar.

[35] Johnathan Fercher da Rosa and Paulo Fernando Ferreira Rosa. Construçao de um framework de planejamento e controle de trajetória em tempo real de múltiplos robôs terrestres. In *III Workshop on MSc Dissertation and PhD Thesis in Robotics, ROBOTICA*, pages 12–01, 2016.

[36] Nils J Nilsson. *Principios de inteligencia artificial*. Morgan Kaufmann, 1980.

[37] Stefan Jacobs, Alexander Ferrein, Stefan Schiffer, Daniel Beck, and Gerhard Lakemeyer. Robust collision avoidance in unknown domestic environments. In *RoboCup*, pages 116–127. Springer, 2009.

[38] Andreas Strack, Alexander Ferrein, and Gerhard Lakemeyer. Laser-based localization with sparse landmarks. In *RoboCup*, pages 569–576. Springer, 2005.

[39] Leandro Nogueira Couto. *Sistema para localização robótica de veículos autônomos baseado em visão computacional por pontos de referência*. PhD thesis, Universidade de São Paulo, 2012.

[40] Feras Dayoub, Timothy Morris, Ben Upcroft, and Peter Corke. Vision-only autonomous navigation using topometric maps. In *Intelligent robots and systems (IROS), 2013 IEEE/RSJ international conference on*, pages 1923–1929. IEEE, 2013.

[41] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2, 1999.

[42] Clauber G Bezerra, P Alsina, and A Medeiros. *Um sistema de localizaçao para um robô móvel baseado em odometria e marcos naturais*. PhD thesis, Master's thesis, UFRN, Natal-RN, 2004.

[43] Clauber Gomes Bezerra. Localização de um robô móvel usando odometria e marcos naturais. Master's thesis, Universidade Federal do Rio Grande do Norte, 2004.

[44] Daniel Fernando Costa Ferreira. *Sistema de visão robótico baseado no sensor kinect para orientação no espaço e contorno de obstáculos*. PhD thesis, 2013.

[45] José Carlos da Costa Gonçalves. *Visão artificial em condução autónoma com câmara Kinect*. PhD thesis, 2013.

[46] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.

[47] Chang-bae Moon and Woojin Chung. Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree. *IEEE Transactions on industrial electronics*, 62(2):1080–1090, 2015.

[48] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2997–3004. IEEE, 2014.

[49] Stephanie Kamarry, Lucas Molina, Elyson Ádan Nunes Carvalho, and Eduardo Oliveira. Drrt: Uma nova abordagem para aumentar a dispersao dos nos na rrt aplicadaa representacao do ambiente e planejamento de caminho. 2015.

[50] Yuri Sarmento Silveira. Um novo método de planejamento de caminho para robôs baseado em espuma probabilística. Master's thesis, Brasil, 2016.

[51] Lei Tang, Songyi Dian, Gangxu Gu, Kunli Zhou, Suihe Wang, and Xinghuan Feng. A novel potential field method for obstacle avoidance and path planning of mobile robot. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 9, pages 633–637. IEEE, 2010.

[52] Milena Pinto Faria, Thiago Mendonça, Leonardo Olivi, and André Marcato. Modified approach using variable charges to solve inherent limitations of potential fields method. In *IEEE/IAS International Conference on Industry Applications (INDUSCON)*, 2014.

[53] Tansuriyavong Suriyon, Higa Keisuke, and Boonmee Choompol. Development of guide robot by using qr code recognition. In *The Second TSME International Conference on Mechanical Engineering*, volume 21, 2011.

[54] Seok Ju Lee, Jongil Lim, Girma Tewolde, and Jaerock Kwon. Autonomous tour guide robot by using ultrasonic range sensors and qr code recognition in indoor environment. In *Electro/Information Technology (EIT), 2014 IEEE International Conference on*, pages 410–415. IEEE, 2014.

[55] Cimini Gionata, Ferracuti Francesco, Freddi Alessandro, Iarlori Sabrina, and Monteriù Andrea. An inertial and qr code landmarks-based navigation system for impaired wheelchair users. In *Ambient Assisted Living*, pages 205–214. Springer, 2014.

[56] Generetion Robots. Sick lms100-10000 laser scanner, indoor version, medium range. `https://goo.gl/HUKQH9`. Acessado em: 07.11.2017.

[57] NewEgg. Asus xtion 2 3d sensor (xtion2). `https://goo.gl/XehQEq`. Acessado em: 07.11.2017.

[58] Guilherme Maciel, Fabrício Coelho, Leonardo Olivi, André Marcato, Ivo Júnior, and Daniel Fernandes. Planejamento e controle n ao-linear para passagens estreitas em robótica móvel assistiva. XIII Simpósio Brasileiro de Automação Inteligente (SBAI), 2017.

[59] Adept Mobile Robots. Pionner 3-dx datasheet. `http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx`, 2011. Acessado em: 09.08.2017.

[60] Frederico Carvalho Vieira. Controle dinâmico de robôs móveis com acionamento diferencial. Master's thesis, Universidade Federal do Rio Grande do Norte, 2006.

[61] Igor F. Okuyama, Marcos R. O. A. Maximo, Anderson L. O. Cavalcanti, and Rubens J. M. Afonso. *Nonlinear Grey-Box Identification of a Differential Drive Mobile Robot*. XIII Simpósio Brasileiro de Automação Inteligente (SBAI), 2017.

[62] Eleri Cardozo, Eric Rhomer, Leonardo Olivi, Ricardo Souza, Fernando Pinho, and Paulo Pinheiro. Ia368n - introdução à robótica móvel. *FEEC, Unicamp*, 2016.

[63] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[64] Felipe Espinosa, Marcelo Salazar, Daniel Pizarro, and Fernando Valdés. Electronics proposal for telerobotics operation of p3-dx units. In *Remote and Telerobotics*. InTech, 2010.

[65] EDSON ROBERTO DE PIERI. Curso de robótica móvel. *Santa Catarina, Florianópolis*, 2002.

[66] Milton Felipe Souza Santos et al. Análise e proposta de arquiteturas de hardware para veículos autônomos. 2013.

[67] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics (intelligent robotics and autonomous agents). 2005.

[68] César Bruno Gomes Rocha. *Monitorização dos modelos de quebra-mares com o sensor Microsoft Kinect V2*. PhD thesis, 2017.

[69] Diana Pagliari and Livio Pinto. Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors. *Sensors*, 15(11):27569–27589, 2015.

[70] Amazon. Official xbox 360 kinect sensor with kinect adventures (xbox 360). `https://goo.gl/w9wyky`. Acessado em: 04.09.2017.

[71] Rodrigo de Sales Alves, Jefferson Oliveira Alves de Araujo, and Francisco Madeiro. Alfabetokinect: Um aplicativo para auxiliar na alfabetização de crianças com o uso do kinect. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 23, 2012.

[72] Lydia R Reither, Matthew H Foreman, Nicole Migotsky, Chase Haddix, and Jack R Engsberg. Upper extremity movement reliability and validity of the kinect version 2. *Disability and Rehabilitation: Assistive Technology*, pages 1–9, 2017.

[73] Fabio Madureira Garcia, Ivo Pedro Gonzalez Junior, Andrea Cardoso Ventura, Karla Souza Caggy Costa da Silva, Fernanda Xavier Ferreira, Jaqueline Ribeiro da Silva, and Wylena Monteiro das Chagas. O uso de novas tecnologias na reabilitação fisioterapêutica: Possibilidades e desafios com o xbox e o kinect. In *14th CONTECSI-International Conference on Information Systems and Technology Management*, 2017.

[74] Pornchai Mongkolnam, Yoottana Booranrom, Bunthit Watanapa, Thammarsat Visutarrom, Jonathan H Chan, and Chakarida Nukoolkit. Smart bedroom for the elderly with gesture and posture analyses using kinect. *Maejo International Journal of Science and Technology*, 11(1):1, 2017.

[75] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[76] João Paulo Ristow et al. Estudo e desenvolvimento de algoritmos de sonar ativo para o mapeamento de áreas submersas. 2015.

[77] Richard P Hodges. *Underwater acoustics: Analysis, design and performance of sonar*. John Wiley & Sons, 2011.

[78] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[79] Fernando Antonio Lucena Aiube, Tara Keshar Nanda Baidya, and Edison Americo Huarsaya Tito. Processos estocásticos dos preços das commodities: uma abordagem através do filtro de partículas. *Revista Brasileira de Economia*, 60(3):215–233, 2006.

[80] Marco Antonio Dalcin Tocchetto. Estimador de estados para robô diferencial. 2017.

[81] Gabriel Lucas Sousa da Silva. *CALIBRAÇÃO DE MODELO CONSTITUTIVO DE INTERFACE POR MEIO DE FILTRO DE KALMAN ESTENDIDO*. PhD thesis, Universidade Federal do Rio de Janeiro, 2016.

[82] André Macêdo Santana. Localização e mapeamento simultâneos de ambientes planos usando visão monocular e representação híbrida do ambiente. 2011.

[83] Tomas Lozano-Perez. Spatial planning: A configuration space approach. *IEEE transactions on computers*, (2):108–120, 1983.

[84] Daniel Alves Barbosa de Oliveira Vaz. *Planejamento de movimento cinemático-dinâmico para robôs móveis com rodas deslizantes*. PhD thesis, Universidade de São Paulo, 2011.

[85] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[86] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[87] J Kuffner and S LaValle RRT-Connect. An efficient approach to single-query path planning ieee international conference on robotics and automation. *San Francisco*, pages 473–479, 2000.

[88] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.

[89] Hiparco Lins Vieira. *Redução do custo computacional do algoritmo RRT através de otimização por eliminação*. PhD thesis, Universidade de São Paulo, 2014.

[90] Jean-Claude Latombe. Robot motion planning (the kluwer international series in engineering and computer science). 1990.

[91] Sebastian Thrun and Arno Bücken. Learning maps for indoor mobile robot navigation. Technical report, DTIC Document, 1996.

[92] Rovilson Mezencio. *Implementação do método de campos potenciais para navegação de robôs móveis baseada em computação reconfigurável.* PhD thesis, Universidade de São Paulo, 2002.

[93] O Khatib and JF Le Maitre. Dynamic controlof manipulators operating in a complex environment. In *On Theory and Practice of Robots and Manipulators, 3rd CISM-IFToMM Symp*, volume 267, 1978.

[94] Oussama Khatib. Commande dynamique dans l'espace opérationnel des robots manipulateurs en présence d'obstacles. *PhD dissertation, Ecole Nationale Superieure de l'Aeronautique et de l'Espace*, 1980.

[95] Oussama Khatib. Dynamic control of manipulator in operational space. In *Proc. 6th IFToMM World Congress on Theory of Machines and Mechanisms*, pages 1128–1131, 1983.

[96] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[97] Alexander C Woods and Hung M La. A novel potential field controller for use on aerial robots. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017.

[98] S Sharma, R Sutton, D Hatton, and Y Singh. Path planning of an autonomous surface vehicle based on artificial potential fields in a real time marine environment. 2017.

[99] Leonardo Rocha Olivi et al. Navegação de robôs móveis assistivos por controle compartilhado baseado em campos vetoriais. 2014.

[100] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[101] Tadeu Abreu Cerqueira, Tito LM Santos, and André GS Conceição. A new approach based in potential fields with obstacles avoidance for mobile robots. In *Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), 2016 XIII Latin American*, pages 229–233. IEEE, 2016.

[102] Leandro Luiz Rezende de Oliveira et al. Controle de trajetória baseado em visão computacional utilizando o framework ros. 2013.

[103] Alexandre Menezes Teixeira et al. Coordenação ótima de múltiplos robôs de serviço em tarefas persistentes. 2015.

[104] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[105] Eduardo Brendler Viecili et al. Exploração robótica ativa usando câmera de profundidade. 2014.

[106] Steve Cousins. Welcome to ros topics [ros topics]. *IEEE Robotics & Automation Magazine*, 17(1):13–14, 2010.

[107] Inc The MathWorks. Robotics system toolbox. `https://goo.gl/QXX4EQ`. Acessado em: 02.12.2017.

[108] Aaron Martinez and Enrique Fernández. *Learning ROS for robotics programming*. Packt Publishing Ltd, 2013.

[109] Ricardo Emerson Julio. Dbml: Uma biblioteca de gerenciamento dinâmico de banda para sistemas multirrobôs baseado em ros. 2015.

[110] John Vlissides, Richard Helm, Ralph Johnson, and Erich Gamma. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49(120):11, 1995.

[111] Omron Mobile Robots Adept. Aria core library. `https://goo.gl/iVG1YP`. Acessado em: 26.10.2017.

[112] Michał Nowicki, Marta Rostkowska, and Piotr Skrzypczyński. Indoor navigation using qr codes and wifi signals with an implementation on mobile platform. In *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), 2016*, pages 156–161. IEEE, 2016.

[113] Natividade Santos and AO MONTEIRO. Qr code nas bibliotecas escolares. *ENCONTRO SOBRE JOGOS E MOBILE LEARNING*, pages 166–174, 2012.

[114] Chen Minhua and Guo Jiangtao. Mobile robotics lab spring 2015 automatic landing on a moving target. 2015.

[115] Fabrício Coelho, João Pedro Souza, Milena Pinto, Guilherme Maciel, and André Marcato. Filtro de kalman estendido baseado em visão computacional e odometria aplicado à localização de robôs móveis. XIII Simpósio Brasileiro de Automação Inteligente (SBAI), 2017.

[116] Joao Pedro Carvalho, Marco Jucá, Alexandre Menezes, André Marcato, Alexandre dos Santos Bessa, and Leonardo Olivi. Landing a uav in a dynamical target using fuzzy control and computer vision. In *CBA2016*, pages 2636–2641, 2016.

[117] OpenCV. Camera calibration with opencv. `https://goo.gl/noCCK2`. Acessado em: 18.12.2017.

[118] N Rudy. *Robot Localization and Kalman Filters On finnding your position in a noisy world*. PhD thesis, Thesis submitted for the degree Master of Science, Utrecht University, 2003.

[119] Sônia Cristina Bastos de Souza. *Planejamento de trajetória para um robô móvel com duas rodas utilizando um algoritmo A-Estrela modificado*. PhD thesis, Dissertação de mestrado, Programa de Engenharia Elétrica, UFRJ/COPPE, Rio de Janeiro, RJ, 2008.

[120] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan, Sep 2004.

[121] Raphael de Abreu Alves Silva. Desenvolvimento de ambiente operacional ros para quad-rotores. 2017.

[122] Gazebo. Gazebo history. https://goo.gl/Yhk4nV. Acessado em: 04.12.2017.

APÊNDICE A – Explicação das Funções e Variáveis Presentes nos Algoritmos

Algoritmo 3: Rapidly Exploring Random Trees

- *No* → Estrutura de Dados que contém as três informações de um nó: Estado, Parente e custo.
- *RRT* → Estrutura de Dados que contém todos os nós no algoritmo, conta com a função *AddNo* responsável por adicionar novos nós.
- *Amostra_Estado()* → Função que gera estados em C de maneira aleatória.
- *Busca_No_Mais_Proximo(E,Arvore)* → Função que realiza a busca em uma árvore de qual Nó de seus nós está mais próximo de um estado.
- *Nao_Ha_Obst(E1,E2)* → Função que verifica se não há obstáculos entre dois estados.
- *Obtem_Distancia(E1,E2)* → Função que calcula a distância euclidiana entre dois estados.
- G → Estrutura de Dados que contem estados que formarão o caminho na RRT.

Algoritmo 4: Função Escolhe Parente e Religa Árvore

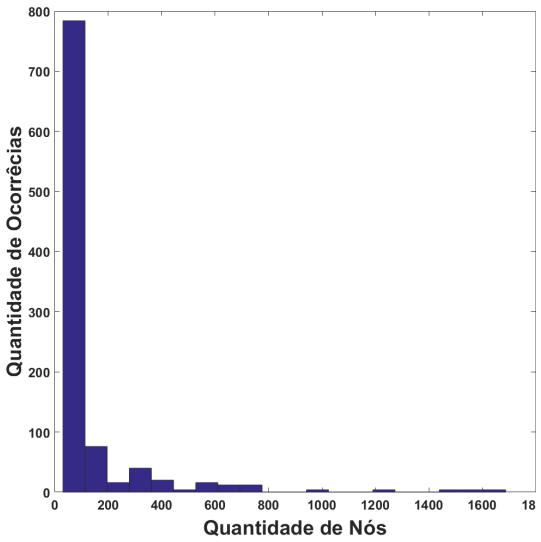
- Z → Conjunto de nós selecionados que estão dentro do círculo.
- *Reconecta* → Realiza as reconexões baseadas no parentesco presentes na árvore.

Algoritmo 5: Campos Potenciais Artificiais

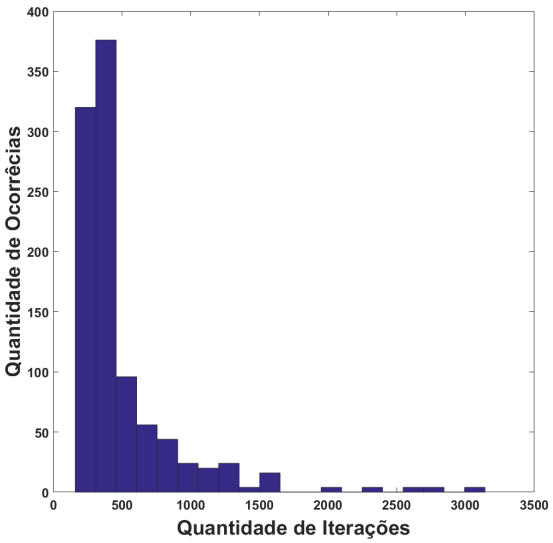
- *getPosition()* → Obtém a leitura da posição atual do Robô.
- *getDistance()* → Obtém a distância da posição atual até o objetivos.
- *getFatt()* → Obtém a Força Atrativa.
- *getFrep()* → Obtém a Força Repulsiva.
- *verifyObs()* → Verifica se os objetos encontrados encontram-se em cima da posição objetivo.
- *atuaVel* → Função que envia ao controlador do robô as velocidades linear e angular.

ANEXO A – Histogramas dos Planejadores Deliberativo

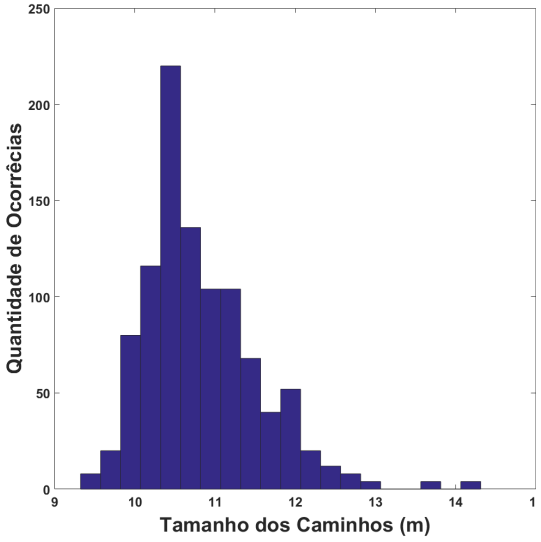
Figura 90 – Histogramas Referentes à Simulação 1 para o Planejador RRT: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo



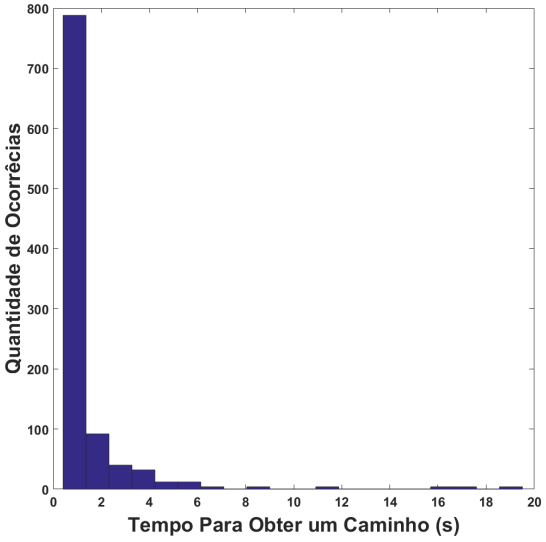
(a)



(b)

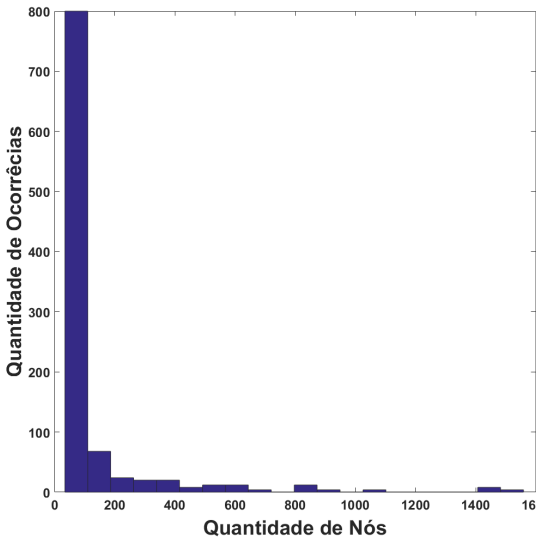


(c)

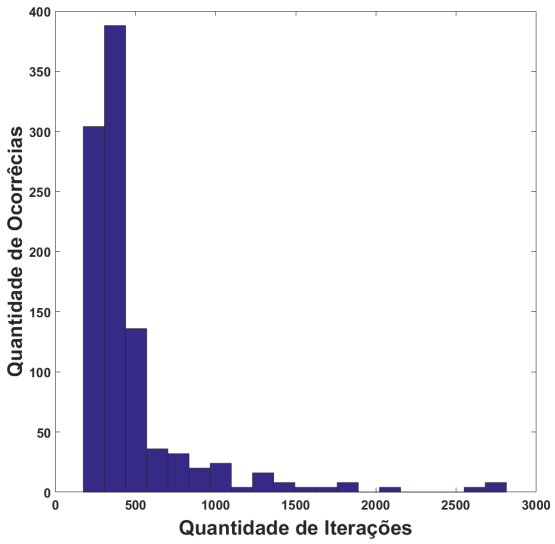


(d)

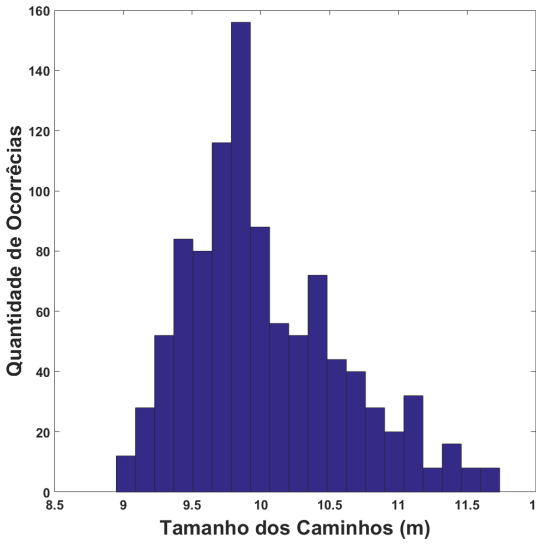
Figura 91 – Histogramas Referentes à Simulação 1 para o Planejador RRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



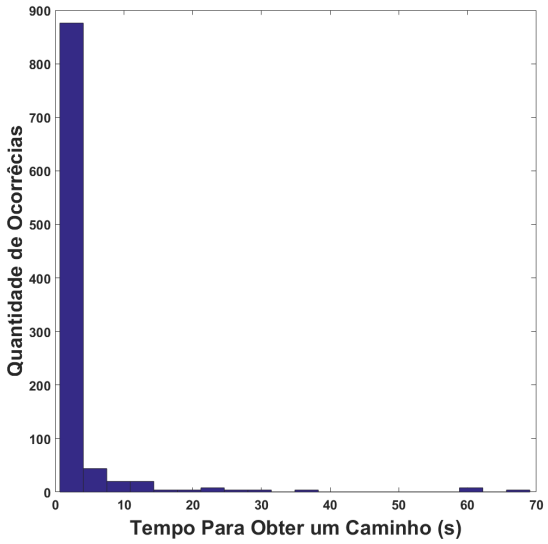
(a)



(b)

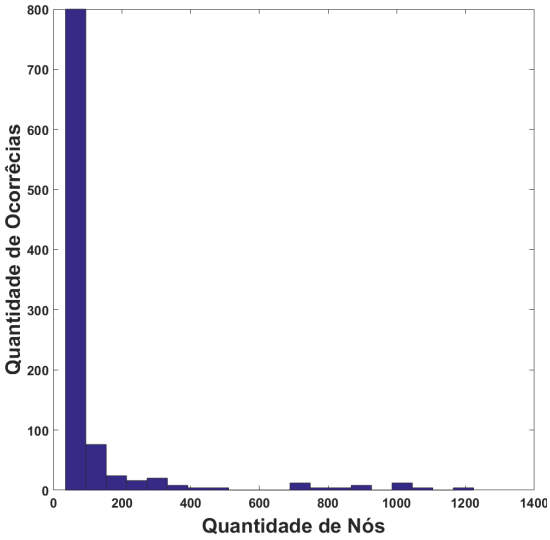


(c)

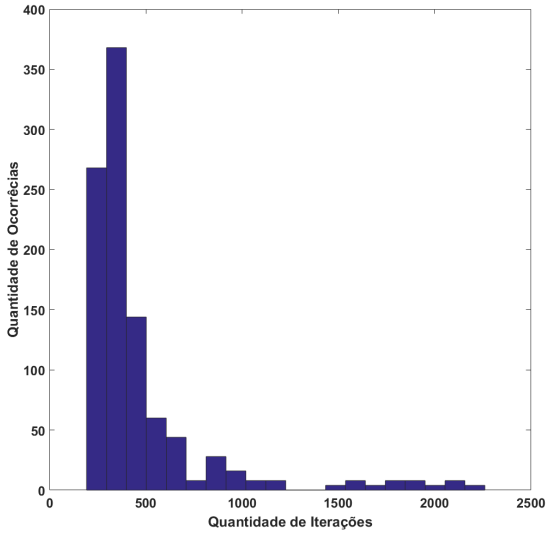


(d)

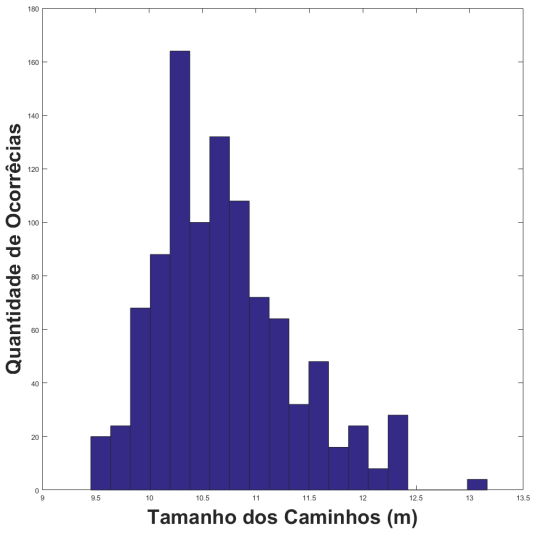
Figura 92 – Histogramas Referentes à Simulação 1 para o Planejador DRRT: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



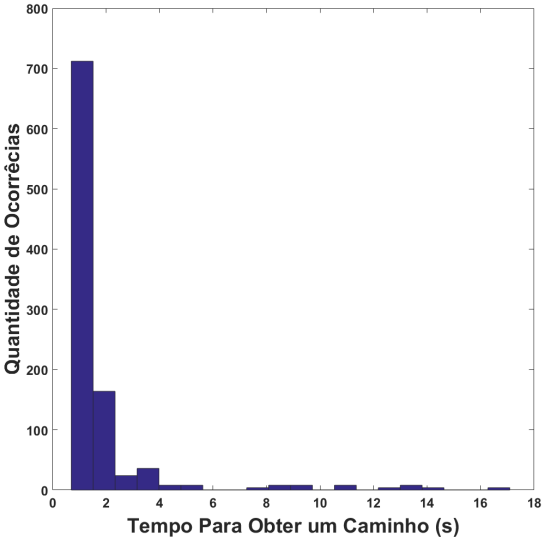
(a)



(b)

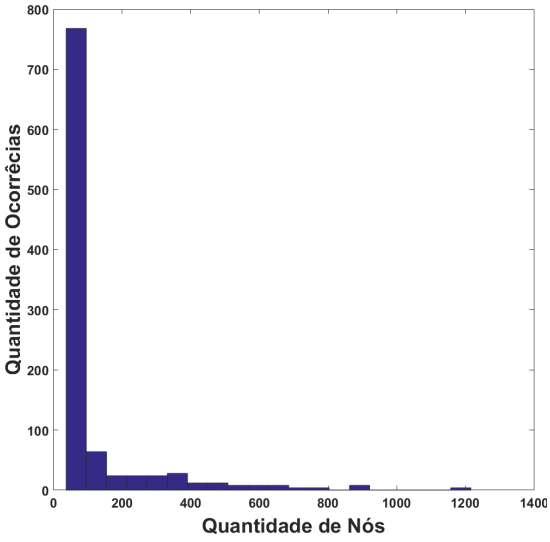


(c)

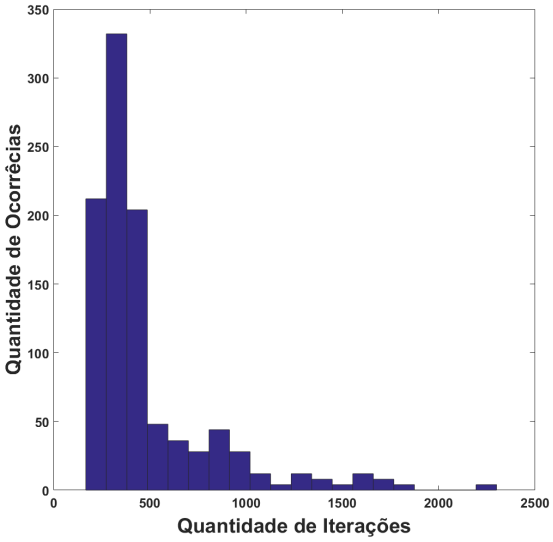


(d)

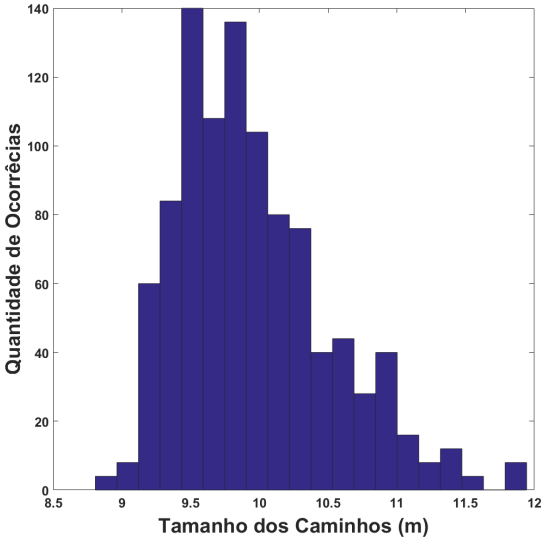
Figura 93 – Histogramas Referentes à Simulação 1 para o Planejador DRRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



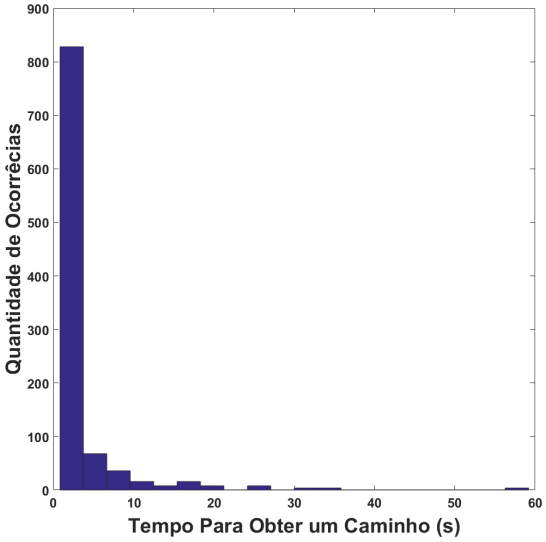
(a)



(b)

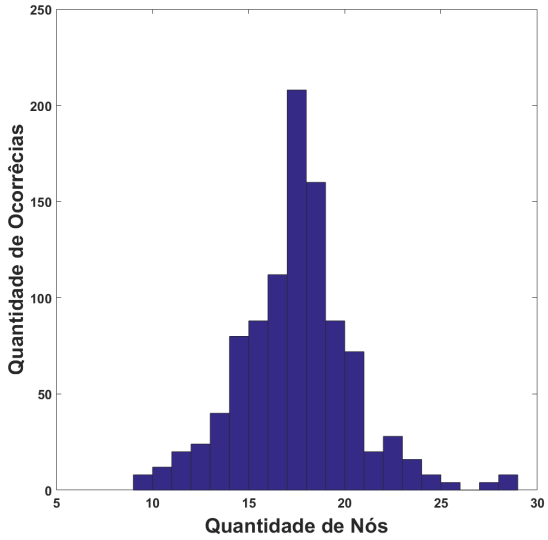


(c)

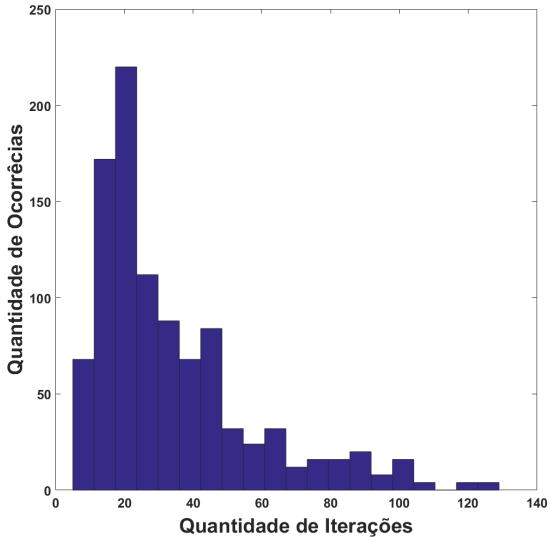


(d)

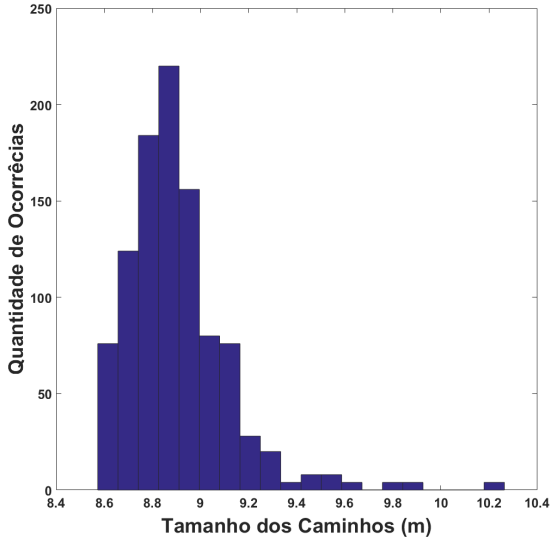
Figura 94 – Histogramas Referentes à Simulação 1 para o Planejador Direct-DRRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



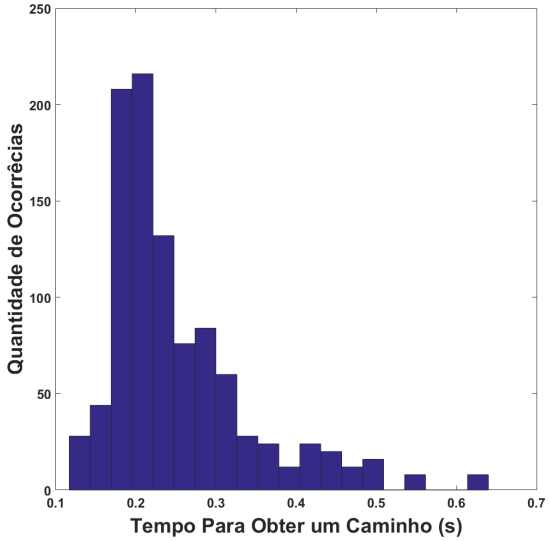
(a)



(b)

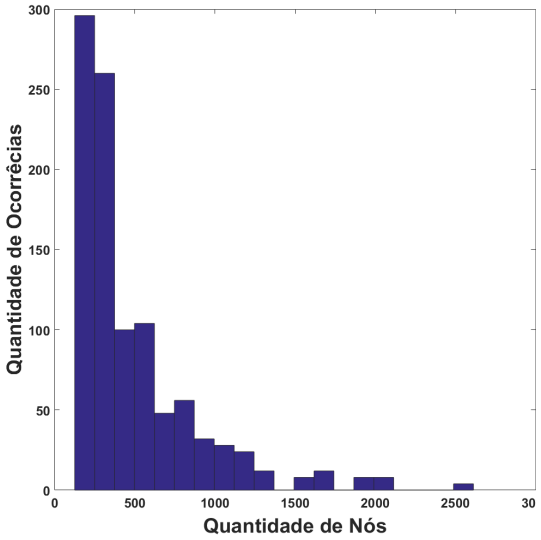


(c)

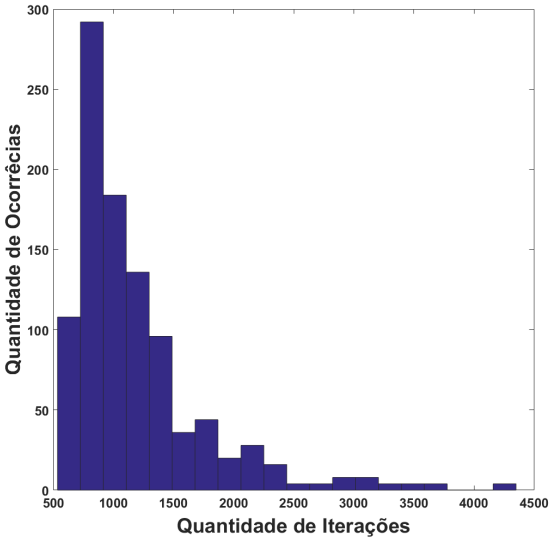


(d)

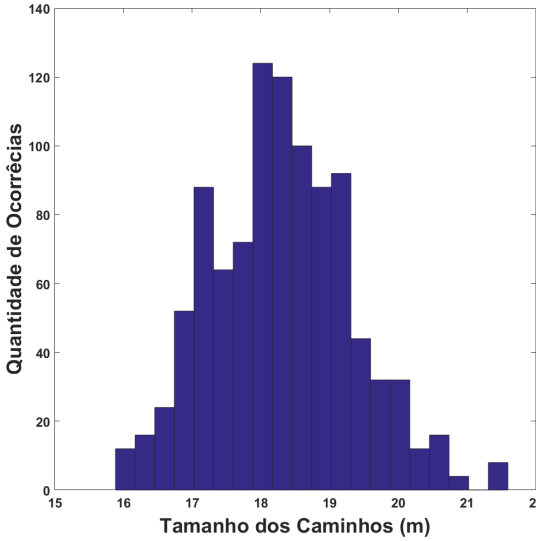
Figura 95 – Histogramas Referentes à Simulação 2 para o Planejador RRT: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo



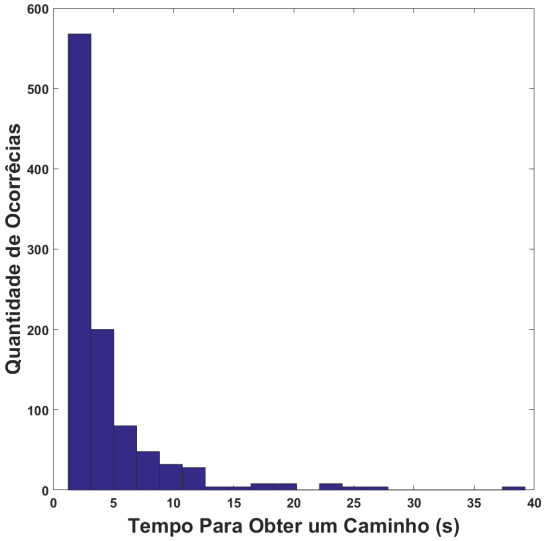
(a)



(b)

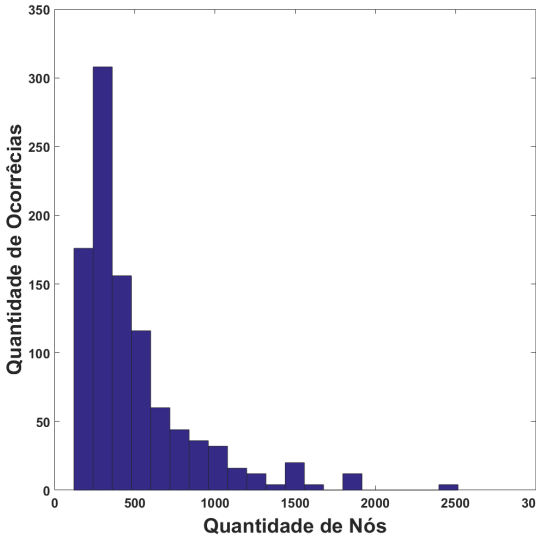


(c)

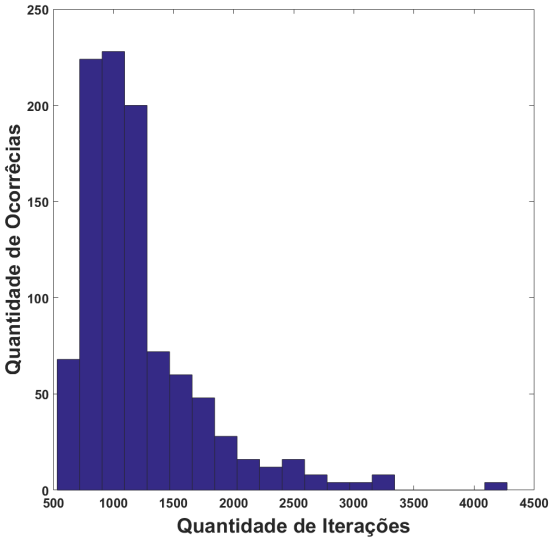


(d)

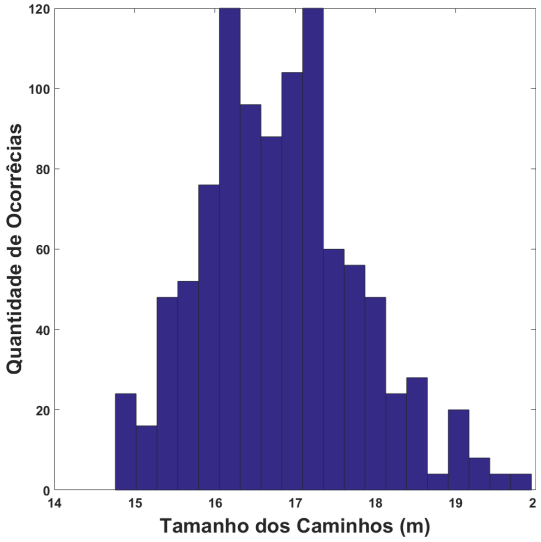
Figura 96 – Histogramas Referentes à Simulação 2 para o Planejador RRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



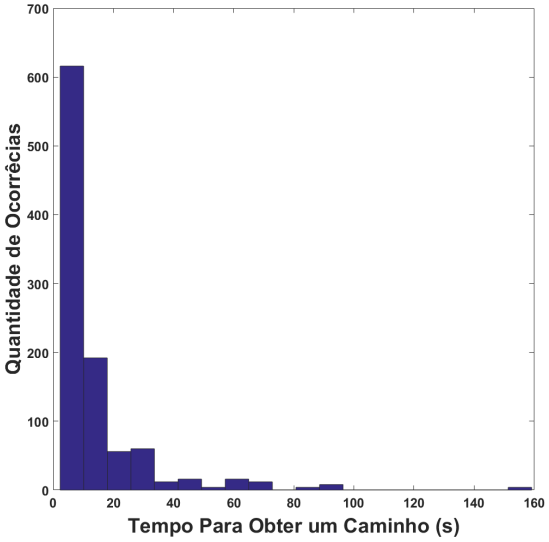
(a)



(b)

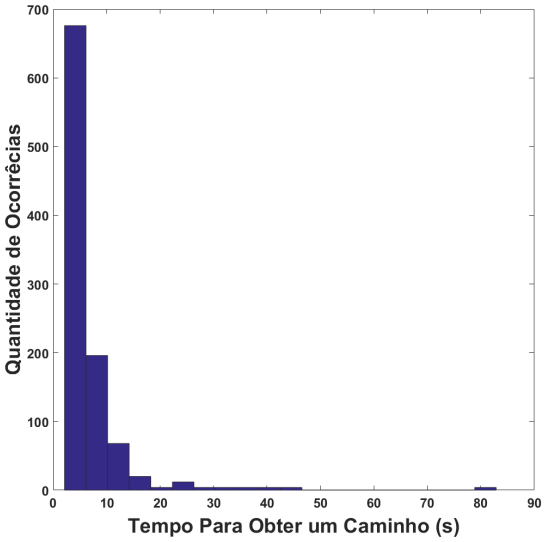


(c)

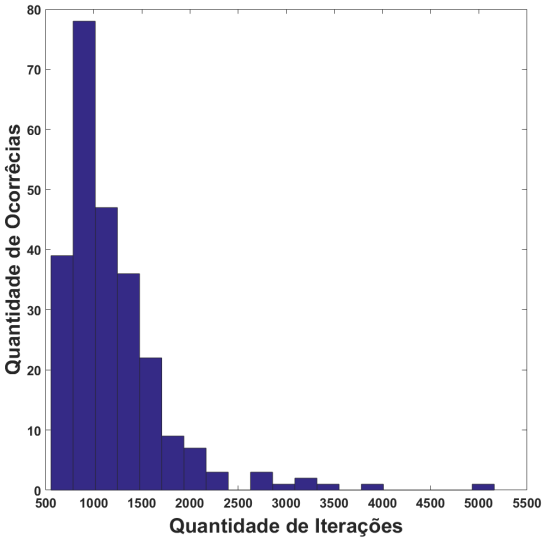


(d)

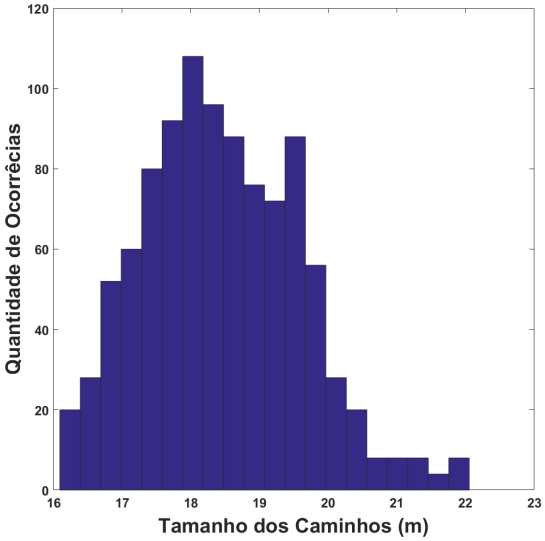
Figura 97 – Histogramas Referentes à Simulação 2 para o Planejador DRRT: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



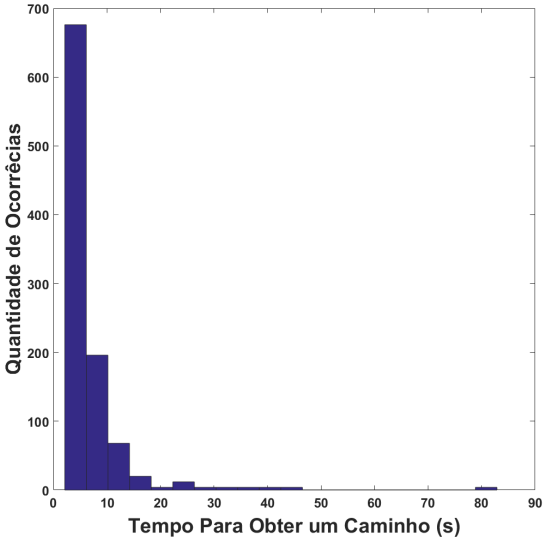
(a)



(b)

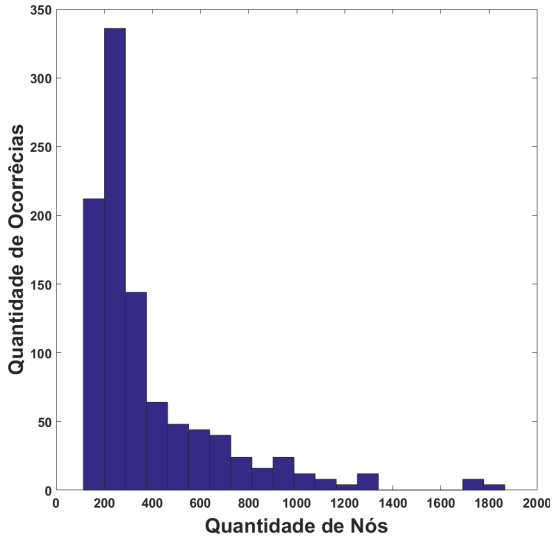


(c)

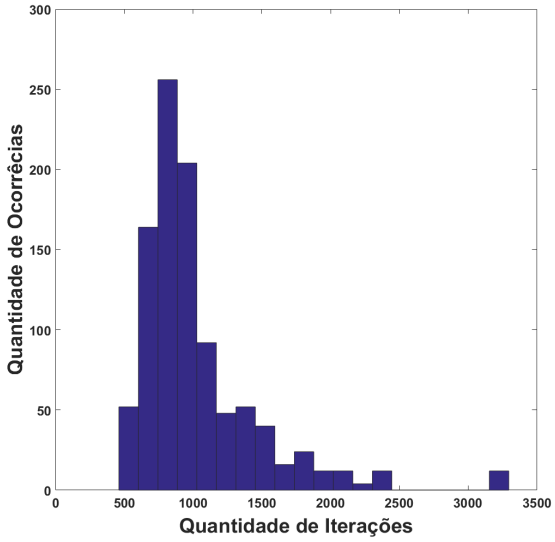


(d)

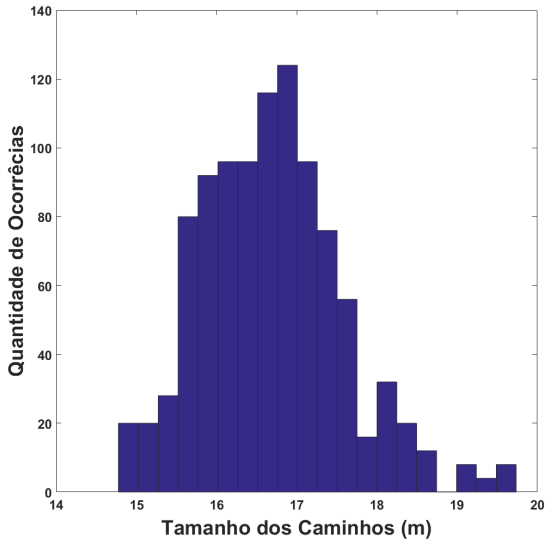
Figura 98 – Histogramas Referentes à Simulação 2 para o Planejador DRRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



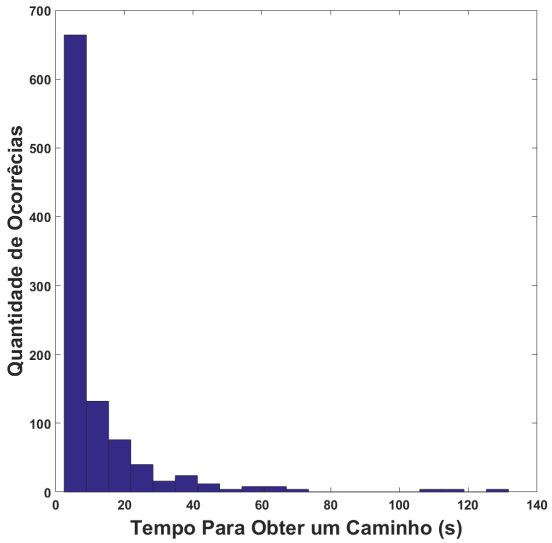
(a)



(b)

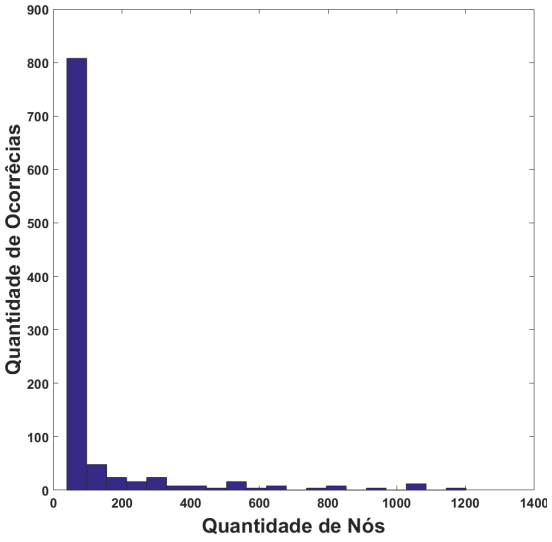


(c)

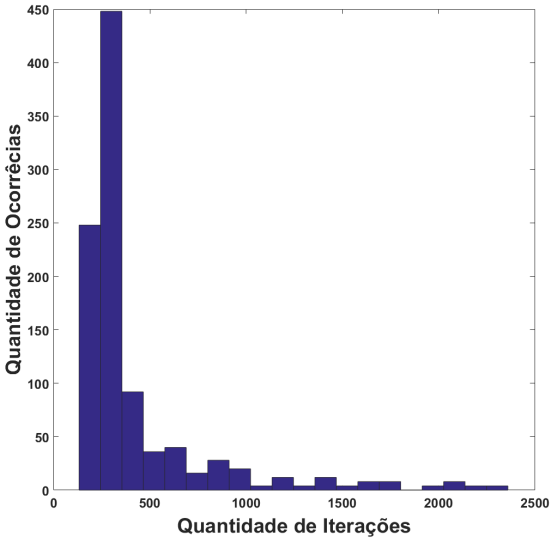


(d)

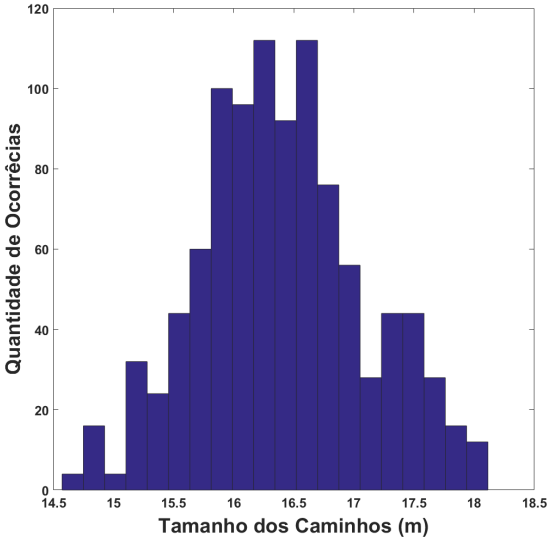
Figura 99 – Histogramas Referentes à Simulação 2 para o Planejador Direct-DRRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



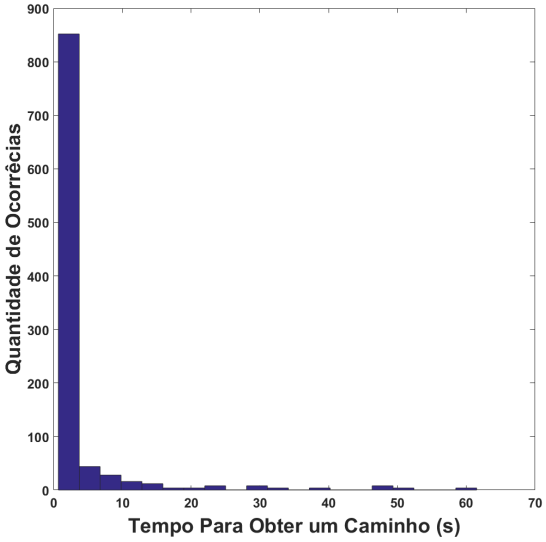
(a)



(b)

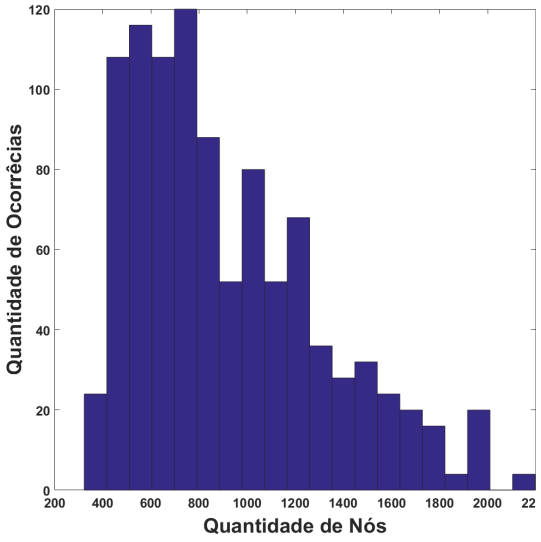


(c)

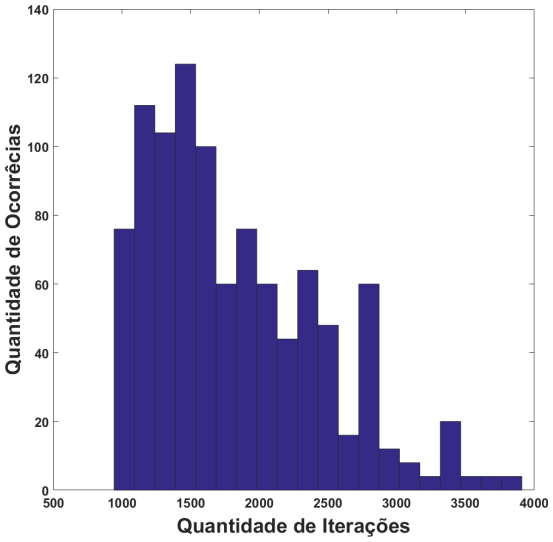


(d)

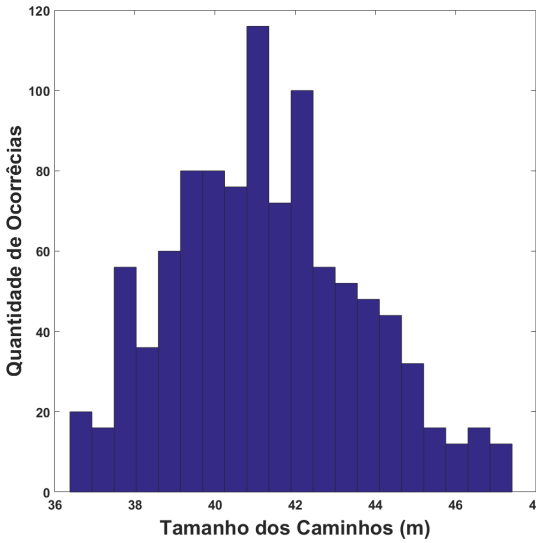
Figura 100 – Histogramas Referentes à Simulação 3 para o Planejador RRT: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo



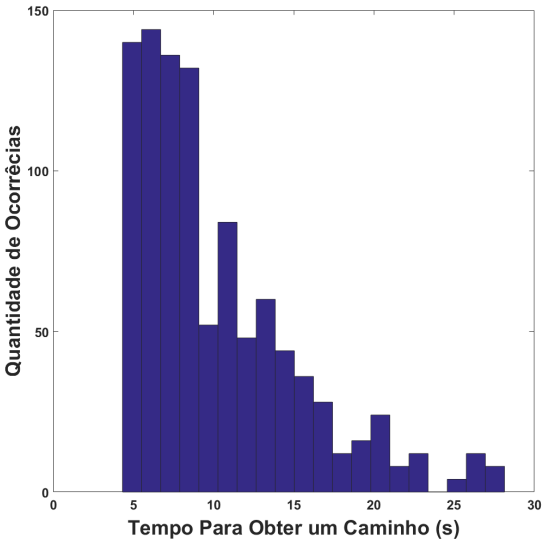
(a)



(b)

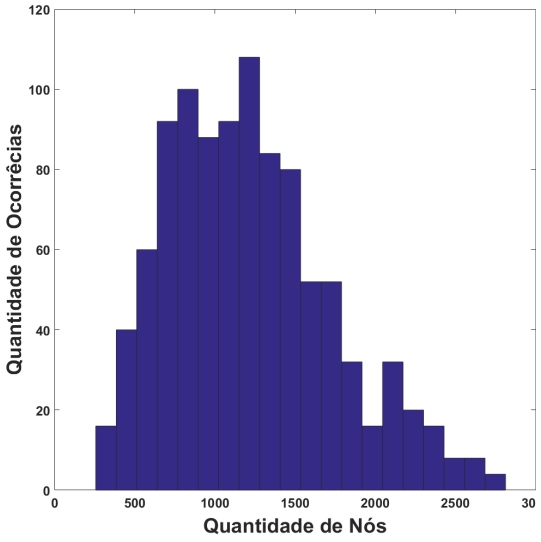


(c)

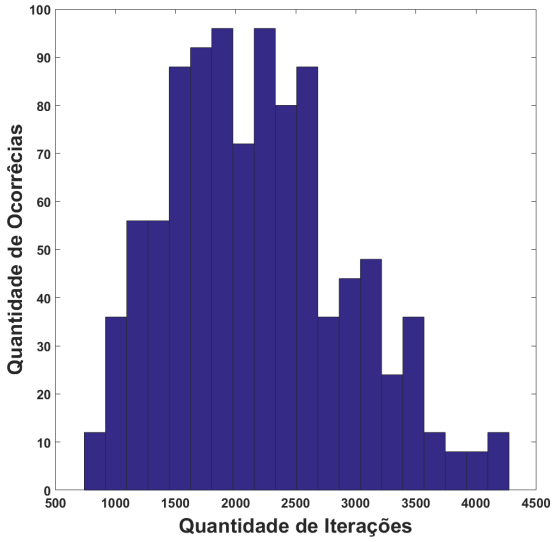


(d)

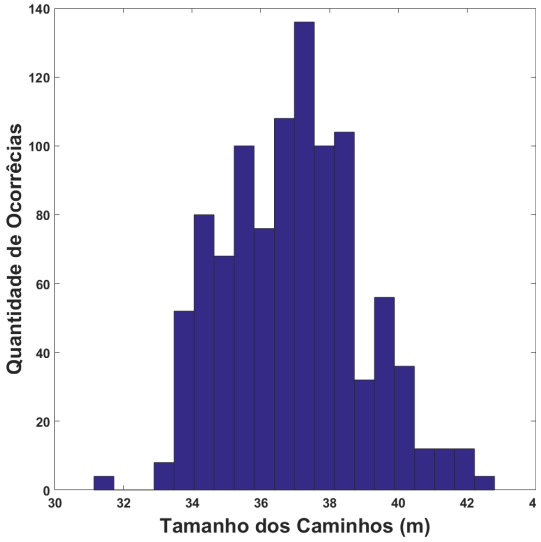
Figura 101 – Histogramas Referentes à Simulação 3 para o Planejador RRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



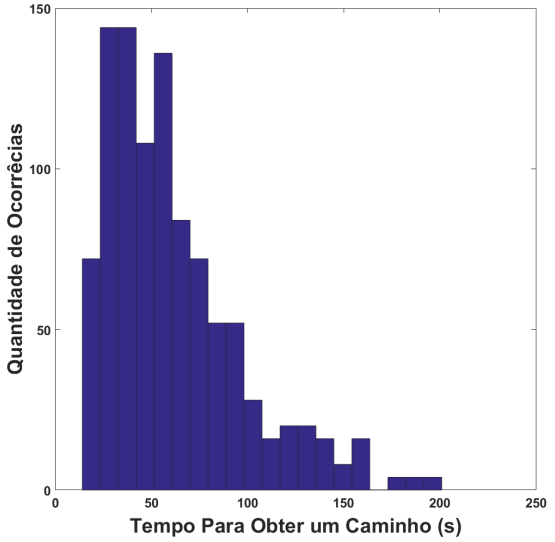
(a)



(b)

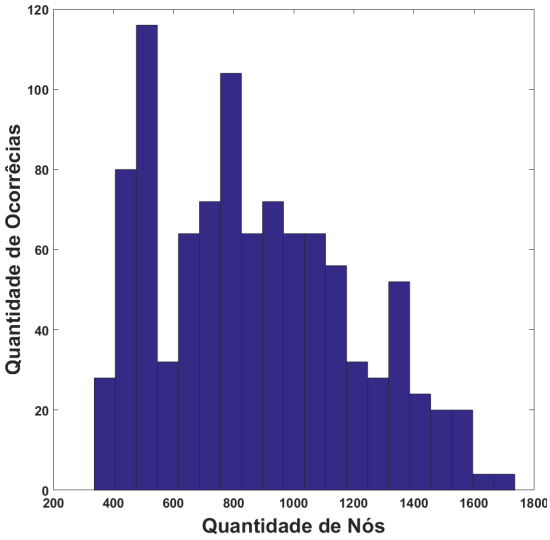


(c)

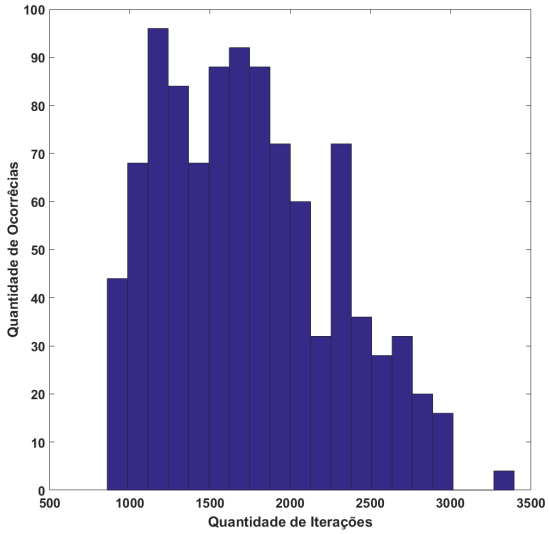


(d)

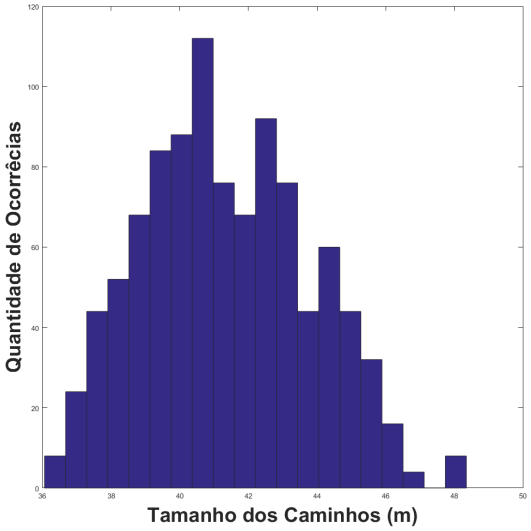
Figura 102 – Histogramas Referentes à Simulação 3 para o Planejador DRRT: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



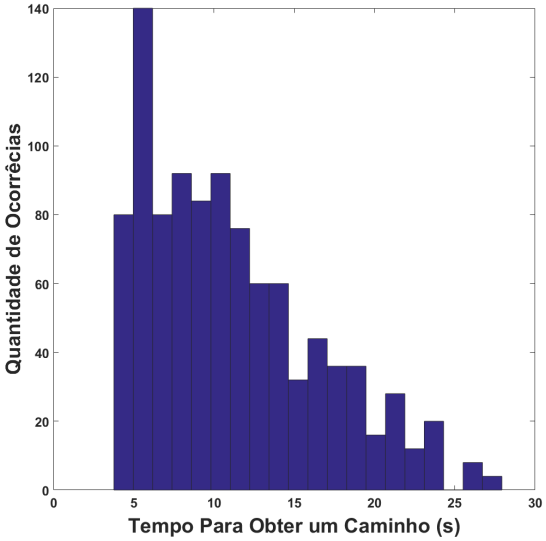
(a)



(b)

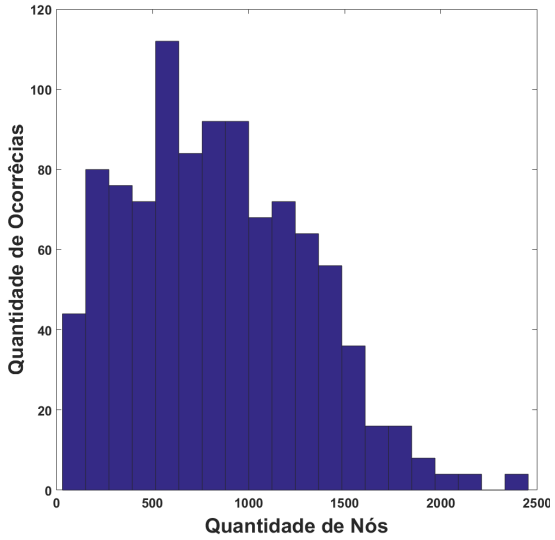


(c)

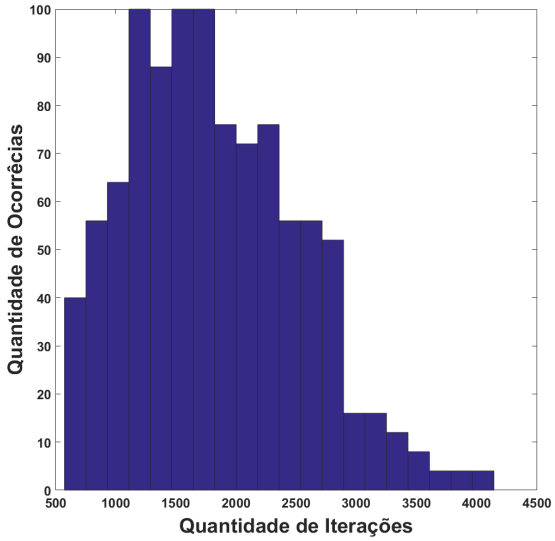


(d)

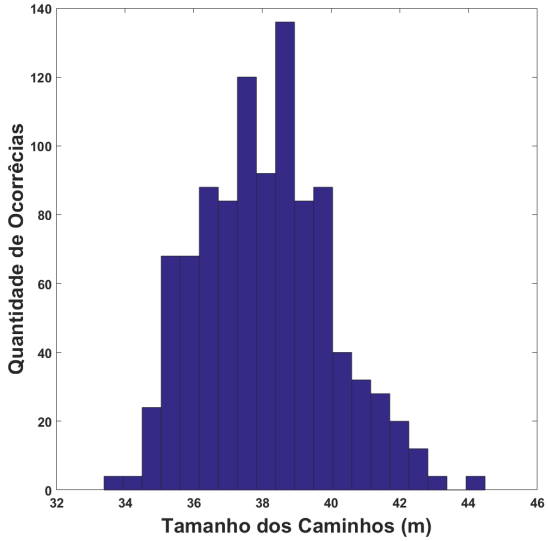
Figura 103 – Histogramas Referentes à Simulação 3 para o Planejador DRRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



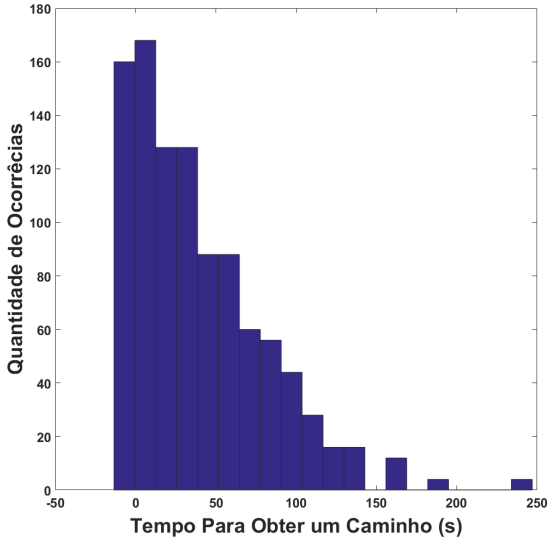
(a)



(b)

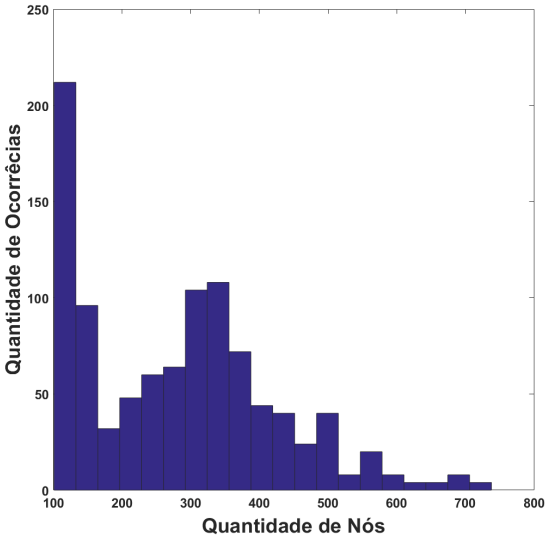


(c)

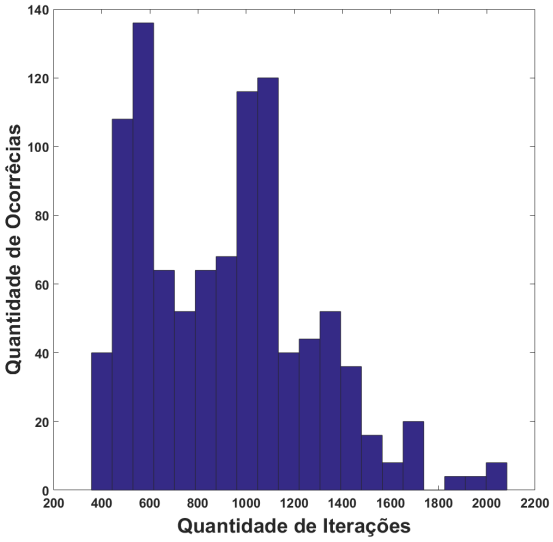


(d)

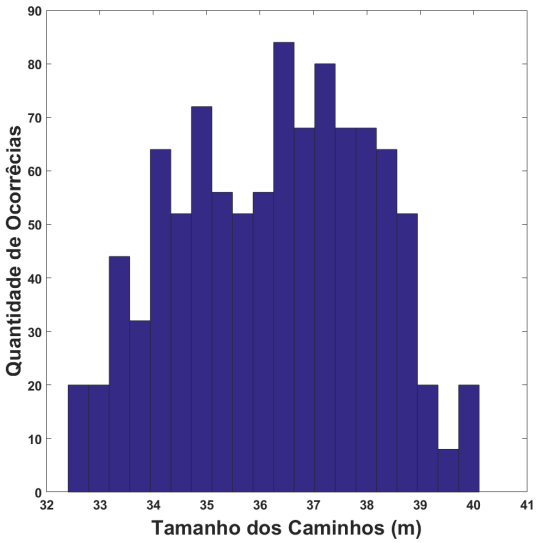
Figura 104 – Histogramas Referentes à Simulação 3 para o Planejador Direct-DRRT*: (a) Número de Nós (b) Número de Iterações (c) Tamanho do Caminho (d) Tempo.



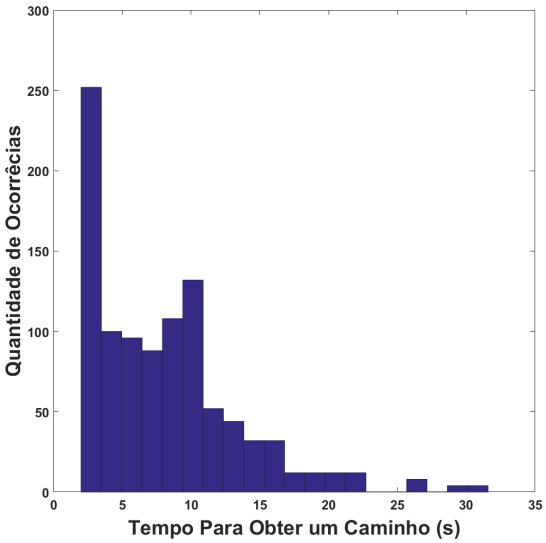
(a)



(b)



(c)



(d)

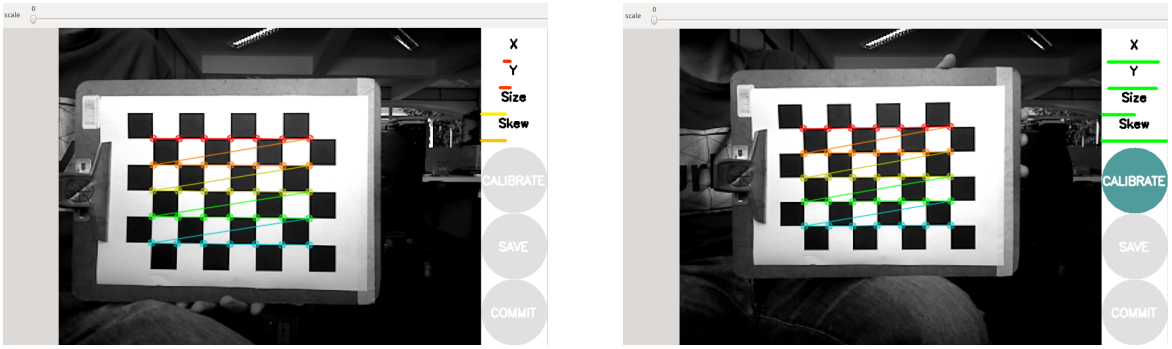
ANEXO B – Metodologia Para Calibração da Câmera

Antes da implementação prática do algoritmo, a câmera RGB que é pertencente ao *Asus Xtion* deverá ser calibrada, para obter os parâmetros que a descrevem. Um arquivo de formato *yaml* será o responsável por armazenar os dados oriundos dessa calibração.

No ROS, há um pacote que realiza a calibração da câmera denominado *camera_calibration*. Sua interface ajuda a realizar as calibrações de maneira intuitiva e rápida. O pacote se inscreve como assinante em dois tópicos criados pelo *Openni2*, o *rgb/image_raw* e o *rgb/camera_info*. A calibração é realizada utilizando a biblioteca *OpenCV* e é demonstrada pela própria empresa em [117].

Basicamente, o responsável por fazer a calibração posiciona um marcador, conhecido como *chessboard*. A interface inicia com as barras dos indicadores de calibração vazias (Figura 105 (a)), nota-se que os botões para gerar a calibração estão anulados. Deve-se realizar movimentos da *chessboard* até que as barras que indicam o progresso da calibração estejam cheias (Figura 105 (b)). Nesse momento, o botão *CALIBRATE* é liberado para que o calibrador possa gerar o arquivo *yaml*.

Figura 105 – Calibração da Câmera: (a) Início da Calibração (b) Final da Calibração.



(a) (b)