
UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

César Bastos da Silva

# EKF e FastSLAM com base em landmarks naturais e artificiais, aplicados a robôs de serviço

Campinas

2023

UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

César Bastos da Silva

# EKF e FastSLAM com base em landmarks naturais e artificiais, aplicados a robôs de serviço

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Orientador: Prof. Dr. Eric Rohmer

Este exemplar corresponde à versão final da dissertação defendida pelo aluno César Bastos da Silva, e orientada pelo Prof. Dr. Eric Rohmer

Campinas

2023

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Elizangela Aparecida dos Santos Souza - CRB 8/8098

Si38e

Silva, César Bastos da, 1996-
EKF e FastSLAM com base em landmarks naturais e artificiais, aplicados a robôs de serviço / César Bastos da Silva. – Campinas, SP : [s.n.], 2023.

Orientador: Eric Rohmer.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Localização. 2. Mapeamento. 3. Robôs autônomos. 4. Filtragem de Kalman. 5. Processo estocastico. I. Rohmer, Eric, 1974-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações Complementares

**Título em outro idioma:** EKF and FasSLAM based on natural and artficial landmarks, applied to service robots
**Palavras-chave em inglês:**
Localization
Mapping
Autonomous robot
Kalman Filtering
Stochastic process
**Área de concentração:** Engenharia de Computação
**Titulação:** Mestre em Engenharia Elétrica
**Banca examinadora:**
Eric Rohmer [Orientador]
Eleri Cardozo
Leonardo Rocha Olivi
**Data de defesa:** 07-07-2023
**Programa de Pós-Graduação:** Engenharia Elétrica

**Identificação e informações acadêmicas do(a) aluno(a)**
- ORCID do autor: https://orcid.org/0000-0002-8332-6756
- Currículo Lattes do autor: http://lattes.cnpq.br/8664632946321635

**COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO**

**Candidato**: César Bastos da Silva RA: 264521

**Data da Defesa**: 07 de julho de 2023

**Título da Tese**: “EKF e FastSLAM com base em landmarks naturais e artificiais, aplicados a robôs de serviço”.

Prof. Dr. Eric Rohmer
Prof. Dr. Leonardo Rocha Olivi
Prof. Dr. Eleri Cardozo

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de PósGraduação da Faculdade de Engenharia Elétrica e de Computação.

*Dedico esta dissertação a todos que me apoiaram neste caminho.*

# Agradecimentos

Gostaria de agradecer primeiramente à minha famılia, em especial a minha Mãe Carla Rosi Azevedo Bastos por todo suporte, mesmo com toda a saudades por estar na minha cidade, tão longe. Agradecer tambem ao meu orientador Eric Rohmer, por ter me auxiliado na resolução e produção deste trabalho, mesmo com esse período pós-pandêmico, algo que alterou totalmente o cronograma. Agradeço também aos membros da banca, por terem aceito o convite e por terem prestigiado o presente trabalho. Por fim gostaria de agradecer a outra inúmeras pessoas que estiveram comigo ao longo do ano e do mestrado, que fizeram parte desta caminhada, dando suporte com muita paciência, amizade e alegria. Por último, mas não menos importante, ao meu suporte diário e companheiro, que esteve comigo nesse último ano, o meu amor Park Nahyun.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

# Resumo

Atualmente, o número de robôs inseridos no dia a dia das pessoas vem crescendo exponencialmente, desde assistentes digitais até robôs aspiradores, entre outros. Para as diferentes tarefas diárias, centros de pesquisa estão desenvolvendo soluções robóticas e autônomas. Esses robôs normalmente possuem a capacidade de se locomover pelo ambiente, e para que eles tenham autonomia para realizar essas tarefas sozinhos, é necessário o desenvolvimento de um sistema de localização e mapeamento (SLAM) confiável e preciso. A resolução do problema de SLAM começa com a escolha dos sensores adequados, levando em consideração a capacidade computacional que o robô pode proporcionar. Portanto, é possível escolher diferentes abordagens para solucionar esse problema, sendo uma delas o método baseado em pontos de interesse (*landmarks*), onde pontos normalmente fixos, artificiais ou naturais são encontrados e auxiliam na estimação da localização do robô. Entre os principais algoritmos, podemos destacar algoritmos probabilísticos baseados no filtro de Bayes, o Extended Kalman Filter (EKF) SLAM e o FastSLAM, que são dois algoritmos baseados em Filtros de Kalman, estimando a localização como processos estocásticos. O EKF-SLAM utiliza a distribuição de probabilidade esperada, estado anterior e informações do sensor para definir a distribuição seguinte. Já o FastSLAM fatora sua solução, utilizando o filtro de partículas para definir sua próxima estimação e filtros de Kalman individuais para cada ponto de interesse, gerando menor custo computacional. Portanto, este trabalho propõe aplicar os métodos de EKF-SLAM e FastSLAM baseados em *landmarks* naturais e artificiais, visando reduzir a influência dos ruídos inerentes do sensores, que são o problema da localização robótica. Como validação, será utilizado um método baseado em marcadores ArUco como ponto de referência para comparação, juntamente com a localização prévia dos *landmarks*. Os algoritmos serão avaliados de maneira qualitativa, análise da trajetória realizada, elipses de erro e mapa gerado, e quantitativamente relacionando o erro ao número de *landmarks*.

**Palavras-chaves**: Localização; Mapeamento; Robótica Móvel.

# Abstract

Currently, the number of robots integrated into people's daily lives is growing exponentially, from digital assistants to vacuuming robots, among others. For various daily tasks, research centers are developing robotic and autonomous solutions. These robots typically have the ability to move around the environment, and in order for them to have the autonomy to perform these tasks on their own, a reliable and accurate Simultaneous Localization and Mapping (SLAM) system needs to be developed. Solving the SLAM problem starts with choosing suitable sensors, taking into account the computational capacity the robot can provide. Therefore, it is possible to choose different approaches to solve this problem, one of which is the method based on landmarks, where fixed, artificial, or natural points of interest are found and aid in estimating the robot's location. Among the main algorithms, we can highlight probabilistic algorithms based on the Bayes filter, such as Extended Kalman Filter (EKF) SLAM and FastSLAM, which are two algorithms based on Kalman Filters, estimating the location as stochastic processes. EKF-SLAM uses the expected probability distribution, the previous state, and sensor information to define the subsequent distribution. On the other hand, FastSLAM factors its solution using a particle filter to determine its next estimation and individual Kalman filters for each landmark, resulting in lower computational cost. Therefore, this work proposes to apply the methods of EKF-SLAM and FastSLAM based on natural and artificial landmarks, aiming to solve the problem of robotic localization. As validation, a method based on ArUco markers will be used as a reference for comparison, along with the previous localization of the landmarks. The algorithms will be evaluated qualitatively by analyzing the performed trajectory, error ellipses, and generated map, and quantitatively by relating the error to the number of landmarks.

**Keywords**: Localization; Mapping; Mobile Robotics.

# Lista de ilustrações

# Lista de tabelas

# Sumário

# 1 Introdução

Nas últimas décadas observa-se o crescimento do uso de sistemas robóticos, expandido sua aplicação para além do meio industrial. Fato que resulta no aumento da incorporação de robôs para auxiliar o cotidiano das pessoas, desde assistentes digitais como Alexa (AMAZON, 2023) e Siri(APPLE, 2023), até robôs que realizam interações com público, já entrando na classe de robôs de serviço (BELANCHE et al., 2020).

Robôs de serviço segundo a definição da Federação Internacional de Robótica (IFR, 2023) são aqueles que exercem atividade úteis para os humanos, excluindo atividades industriais (BELANCHE et al., 2020). Dentre algumas atividades, é possível destacar robôs de entrega (XU et al., 2019), assistente em casa (MIŠEIKIS et al., 2020), cuidador de idosos (PORTUGAL et al., 2019) e assistente em hospitais (DOBREV; GULDEN; VOSSIEK, 2018). Em muitas destas tarefas há interação dos robôs junto ao meio humano, desta forma ter uma sistema de localização e mapeamento (SLAM) confiável e preciso se torna um pré–requisito essencial para dar autonomia ao robô e por consequência, uma aceitação maior da tecnologia (LEE; KIM; CHO, 2018).

O problema do SLAM começa ao ter que definir qual sensor a ser utilizado e então, qual algoritmo a ser implementado. Um dos fatores que possui mais peso, é o poder computacional da plataforma e quando a questão é robôs de serviço, há claramente essa limitação, por ser um processamento na borda e por não o único algoritmo ativo. Diante desse problema, é interessante utilizar sensores que consigam extrair o máximo de informação com menor custo computacional, destacando–se então os Lidares 2D e câmeras monoculares (LEE; KIM; CHO, 2018; FILIPENKO; AFANASYEV, 2018).

Há outros sensores também bastante utilizados, como câmera stereo e Lidar 3D não serão utilizados devido aos seus altos custos computacionais, como é possível ver em (EGGER et al., 2018; CHILIAN; HIRSCHMÜLLER, 2009). Desta forma então foram selecionados os sensores mais populares e eficientes, câmera monocular e lidar, que a indústria da robótica está utilizando para robôs limpadores de casa, sua proposta possui o mesmo foco do presente trabalho, onde se busca uma precisão na localização e *mapping* com baixo custo, tanto computacional como financeiro.

Após as compreensões anteriores, há a necessidade de encontrar um algoritmo que consiga mapear o ambiente e localizar o robô de maneira eficiente. Inúmeras novas técnicas nas últimas décadas foram propostas, entretanto os métodos baseados em conceitos probabilísticos tem sempre um grande destaque, sendo eles o EKF–SLAM e FastSLAM. Estes métodos se caracterizam por se utilizar *features* extraídas a partir das informações provindas dos sensores. Esses pontos de informação podem ser classificados como naturais,

por já serem parte do ambiente ou artificiais, adicionados pelo homem, podendo também entregar informações mais topológicas sobre o ambiente. A combinação destes dois tipos, só tem a acrescentar para com o SLAM (MUÑOZ-SALINAS; MEDINA-CARNICER, 2020).

Portanto, se faz necessário verificar o comportamento do EKF-SLAM e FastSLAM aplicados a robôs de serviço, a partir da detecção de landmarks naturais e artificiais.

## 1.1 Publicações

Durante o tempo de pesquisa, o autor publicou os seguintes trabalhos:

- DA SILVA C., et al. "EKF–slam baseado em landmarks naturais e artificias a ser aplicado em uma cadeira de rodas inteligente."Iberdiscap 2023 .
- DA SILVA, César Bastos, et al. "FastSLAM 1.0 aplicado em um cadeira de rodas inteligente."EADCA 2022.

## 1.2 Objetivo Geral

O presente trabalho tem como objetivo a implementação dos métodos clássicos de localização e mapeamento, EKF–SLAM e Fast SLAM e utilizar tipos de *landmarks* para ser implementado em robôs de serviço.

Para atender o objetivo principal, é possível definir os seguintes objetivos específicos:

- Utilizar o *framework Robot Operating System* (ROS) como plataforma de desenvolvimento, o que possibilita a integração de métodos e algoritmos consolidados na área de Robótica;
- Utilizar informações provenientes de hodometria e IMU;
- Implementar a detecção em marcadores ArUco;
- Implementar a detecção de *features* do lidar;
- Implementar algoritmos EKF–SLAM e FastSLAM;
- Realizar testes em simulação;
- Realizar testes em uma bancada experimental;
- Implementar um sistema de *ground truth* para comparar a performance dos métodos implementados;
- Utilizar métricas de comparação para apresentar quantitativamente os resultados.

## 1.3 Organização

O presente trabalho está organizado na seguinte forma: no Capítulo 2 será apresentada uma visão geral do estado–da–arte de robôs de serviço, dando ênfase nos tipos de sensores utilizados por eles e também uma visão sobre o estado–da–arte sobre SLAM e *feature detection*. No capítulo 3 serão apresentados os materiais e métodos utilizados para elaborar testes das técnicas de SLAM estudadas. No capítulo 4 será apresentado como foram organizados os experimentos, explicitando já parte de como foi executado. No capítulo 5, os resultados das técnicas tanto em simulação quanto em teste prático serão apresentados. Ao final da dissertação serão apresentas as conclusões deste trabalho.

# 2 Estado da Arte

Este capítulo apresenta uma revisão sobre modelos de robôs de serviços disponíveis na literatura, desde os já comercializados até robôs ainda em estado de laboratório, dando ênfase nos seus meios de localização. Também serão revisados algoritmos de localização da robótica em geral com diferentes sensores.

## 2.1 Robôs de serviço

Robôs de serviço em uma definição bem inicial, são robôs que são utilizados fora do meio industrial. Com o rápido crescimento desta indústria e novas funcionalidades a estes robôs vão sendo introduzidas, novas definições foram sendo criadas. A Federação Internacional de Robótica (IFR)(IFR, 2023) os define como aqueles que exercem atividade úteis para os humanos, excluindo atividades indústria. Já (WIRTZ et al., 2018), os define como "sistemas autônomos e interfaces adaptáveis que interagem, comunicam e entregam um serviço a uma organização de clientes". A partir destas duas novas definições é possível identificar que novas características vão sendo adicionadas ao robô, não somente de entregar os serviços com a tecnologia, mas também o fato da interação com humanos (LEE, 2021).

A área de robôs de serviço abrange diversas atividades, visando então classificar onde pode ser aplicado, a IFR classificou a utilização de robôs de serviço em dois grandes grupos (Figura 2.1), de uso profissional e pessoal. No meio profissional, é possível enfatizar tarefas de limpeza, na área médica, na área de segurança e inspeção. Sua utilização visa aumentar a produtividade, prover um serviço de qualidade e até diminuir custo com pessoal (GONZALEZ-AGUIRRE et al., 2021).

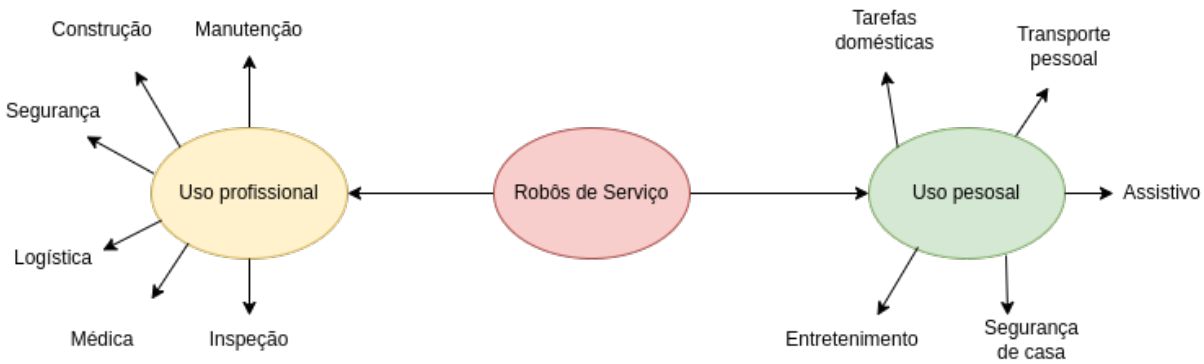



Figura 2.1 – Taxonomia de robôs de serviço. Adaptado de (GONZALEZ-AGUIRRE et al., 2021)

Já para uso pessoal, tarefas de entretenimento, assistência a pessoas com dificuldades, tarefas domésticas e até segurança de casa são algumas das aplicações. Neste

trabalho não há um foco de uma área específica para os robôs de serviço, desta forma serão pontuados exemplos para ambos os segmentos, desde que este tenha mobilidade com rodas como uma de suas *features*.

### 2.1.1 Robôs de serviço na pesquisa

Robôs de serviço profissionais são aqueles que fornecem serviços as pessoas, podendo ou não ter interação social. (LUO; LAI, 2011) propõe um robô de serviço que tenha a capacidade identificar símbolos e sinais que são comuns, construindo um mapa rico de informações. Para construir o mapa é utilizada fusão sensorial a partir da intersecção de covariância (CI). O robô é composto por MESA SwissRanger, Stereo Camera, SICK LMS-100, sensores ultrassônicos e sistema de hodometria nas rodas, conforme Figura 2.2. O método implementado foi comparado com o EKF–SLAM e Graph–SLAM provou que o método consegue construir um mapa preciso e rico em informações.

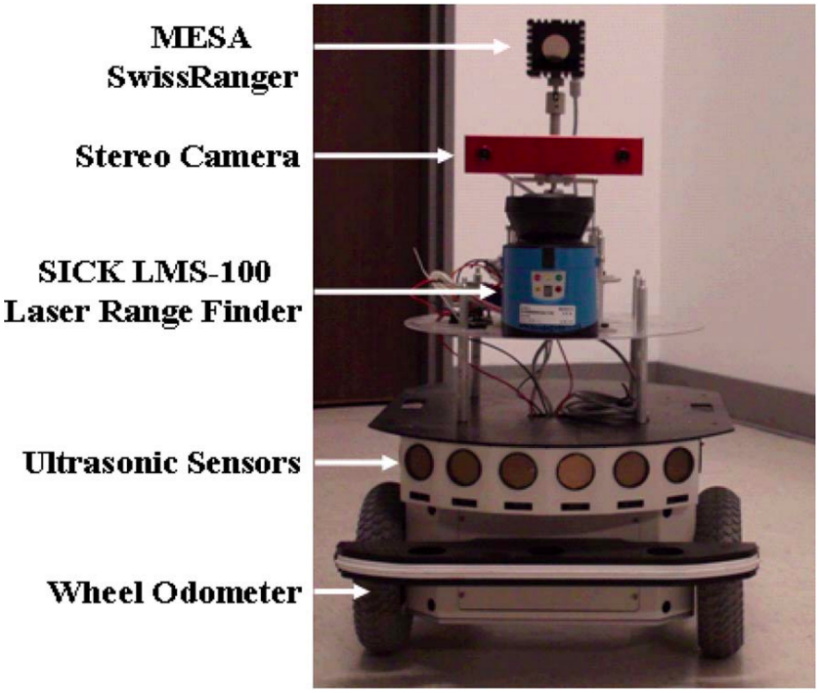



Figura 2.2 – Robô desenvolvido por (LUO; LAI, 2011)

(BEHAN; O'KEEFFE, 2008) propõe "Lucas", um robô para ser assistente em uma biblioteca, desenvolvido para ter uma interação com humanos. Seu sistema de localização realiza a fusão sensorial via EKF de três diferentes fontes de informação, imagem, hodometria e sonar. Gerando um mapa 2D, retirando pontos de interesse, linhas verticais, devido ao padrão das estantes de livros, conseguindo resolver o problema da localização.

Os robôs pessoais, são aqueles mais para usos domésticos, um exemplo é (VALLIVAARA et al., 2011) que trata de robôs para limpeza de chão, resolvendo o

problema de localização e mapeamento através anomalias no campo magnético interno e medindo a qualidade do mapa adquirido. O autor apresentou resultados que o método aplicado conseguiu corrigir a hodometria, tendo como principal contribuição que o uso deste tipo de sensor pode ser útil em situações onde o *laser*, muito utilizado possua problemas. No trabalho de (PYO et al., 2015) aborda a ideia de um robô que consiga reconhecer o seu redor, desde obstáculos a emoções humanas. Em um ambiente tão rico, fica complicada a utilização de apenas uma fonte de informações, criando um sistema distribuído de sensores utilizando o ROS–TMS. O sistema é composto por *Laser Range Finde*r, RGB-D camera, USB câmera, *Velodyne Lidar senso*r e um *Vicon*, mostrando que quanto mais variedade de informação mais rico será o mapa criado.

(KHALIQ; SAFFIOTTI, 2015) apresenta um método baseado em marcadores RFID, mas não utilizados diretamente para se localizar, utilizando–os como uma navegação orientada a objetivos, onde o robô se movimentará sempre em direção ao marcador artificial, não necessitando aplicar técnicas de localização. Já (MIŠEIKIS et al., 2020) (2.3) propõe um robô assistente visando tarefas de interação e cuidados, relacionando as informações visuais, laser, ultrassom, áudio e mecânicos o robô consegue realizar uma navegação segura. Em relação a localização o robô utiliza informações provindas de dois *Lidars* e utiliza a biblioteca GMapping e o método probabilístico Monte Carlo Adaptativo. Além disso o robô possui duas câmeras de profundidade, além de também salvar informações topológicas sobre os locais. Já (DOBREV; GULDEN; VOSSIEK, 2018) prefere a utilização de um radar *wireless*, pois em ambiente hospitalar há muitas superfícies transparentes, fazendo com que o comportamento do lidar 2D seja afetado. O algoritmo utilizado foi o EKF–SLAM e em comparação com o AMCL com lidar, obteve resultados praticamente idênticos.

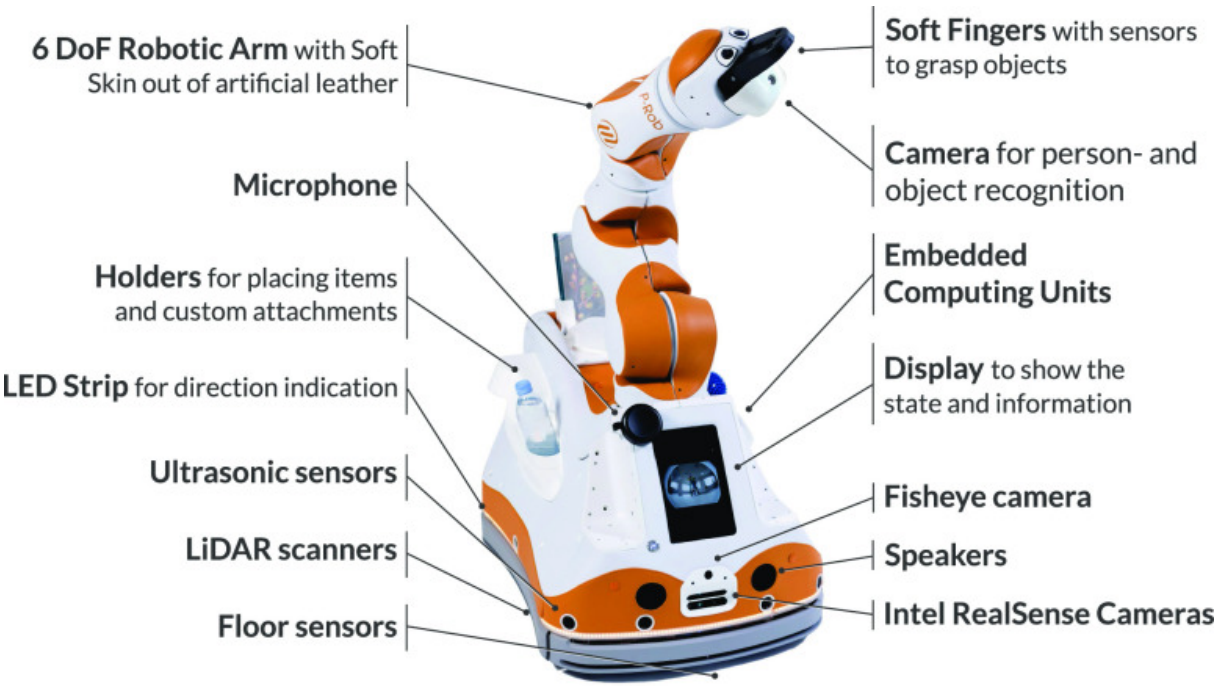



Figura 2.3 – Robô desenvolvido por (MIŠEIKIS et al., 2020)

(RAMALINGAM et al., 2020) também utiliza o AMCL para o sistema de localização e para mapeamento o HectorSLAM. Vale ressaltar, pela disponibilidade de ambos junto ao ROS, sua aplicação fica mais facilitada. Por fim, (LEE; KIM; CHO, 2018), propõe a utilização somente de uma câmera monocular, ao detectar linhas e pontos–de–fuga, corrigindo a posição através combinação de *frames* para fechar o ciclo, Obtendo resultados em relação a velocidade e precisão melhores que o ORB–SLAM.

É possível encontrar diferentes abordagens para a solucionar o problema do SLAM, mas o que é possível encontrar na maioria dos trabalhos é construir um robô que possua diferentes fontes de informação, ou seja, não somente visão por exemplo. Mesmo assim, ainda há uma dificuldade em soluções para o SLAM, visto que muitas soluções são em dados gravados e por um curto período de tempo, estando em aberto a real solução (SHI et al., 2020).

### 2.1.2 Robôs de serviço comerciais

Os exemplos apresentados são todos sendo desenvolvidos em centros de pesquisa, entretanto é possível encontrar alguns exemplares que já comercializados. Visando entender melhor, o que normalmente compõe o robô de serviço até para o meio comercial serão apresentados alguns exemplos.

Primeiramente, um dos exemplos mais famosos, o robô Pepper (Figura 2.4) (JONG et al., 2018). O robô humanoide desenvolvido para interações social para todos os públicos. Ele é composto por um *sonar*, um sensor 3D, duas câmeras RGB, sensores infravermelho, módulos de *laser* e *bumpers*. Pepper utiliza o pacote de navegação da NAOqi, o qual possui limitações e baixa acurácia, visto isso grupos de pesquisa também tem trabalhados junto ao robô comercial, visando propor métodos que entreguem eficiência na navegação para o robô Pepper (ALHMIEDAT et al., 2023).

Devido a utilização do NAOqi, o Pepper possui interação com o ROS *framework*, o que facilita a implementação de diferentes técnicas de SLAM. (ALHMIEDAT et al., 2023) propõe a utilização de *Adaptive Monte Carlo Localization* (AMCL), conseguindo construir um mapa, se localizar e navegar de maneira eficiente. Entretanto, o autor enfatiza que visa inserir informações visuais para melhorar o sistema. Já (GÓMEZ et al., 2019), utiliza um sistema de navegação visual baseado no ORB–SLAM, a partir da imagem de uma câmera RGB junto com a informação odométrica do robô. O foco do trabalho foi adaptar os sensores do Pepper, visando trabalhar em grandes ambientes, mesmo que haja limitação de distância na medição do *Lidar* e RGB-D câmera.

Um trabalho que apresenta uma grande contribuição, é apresentado por (GROOT et al., 2018), no qual é realizada a comparação entre três algoritmos de localização, aplicados ao Pepper, *slam_gmapping*, *hector_slam* e *Cartographer*. O que ele

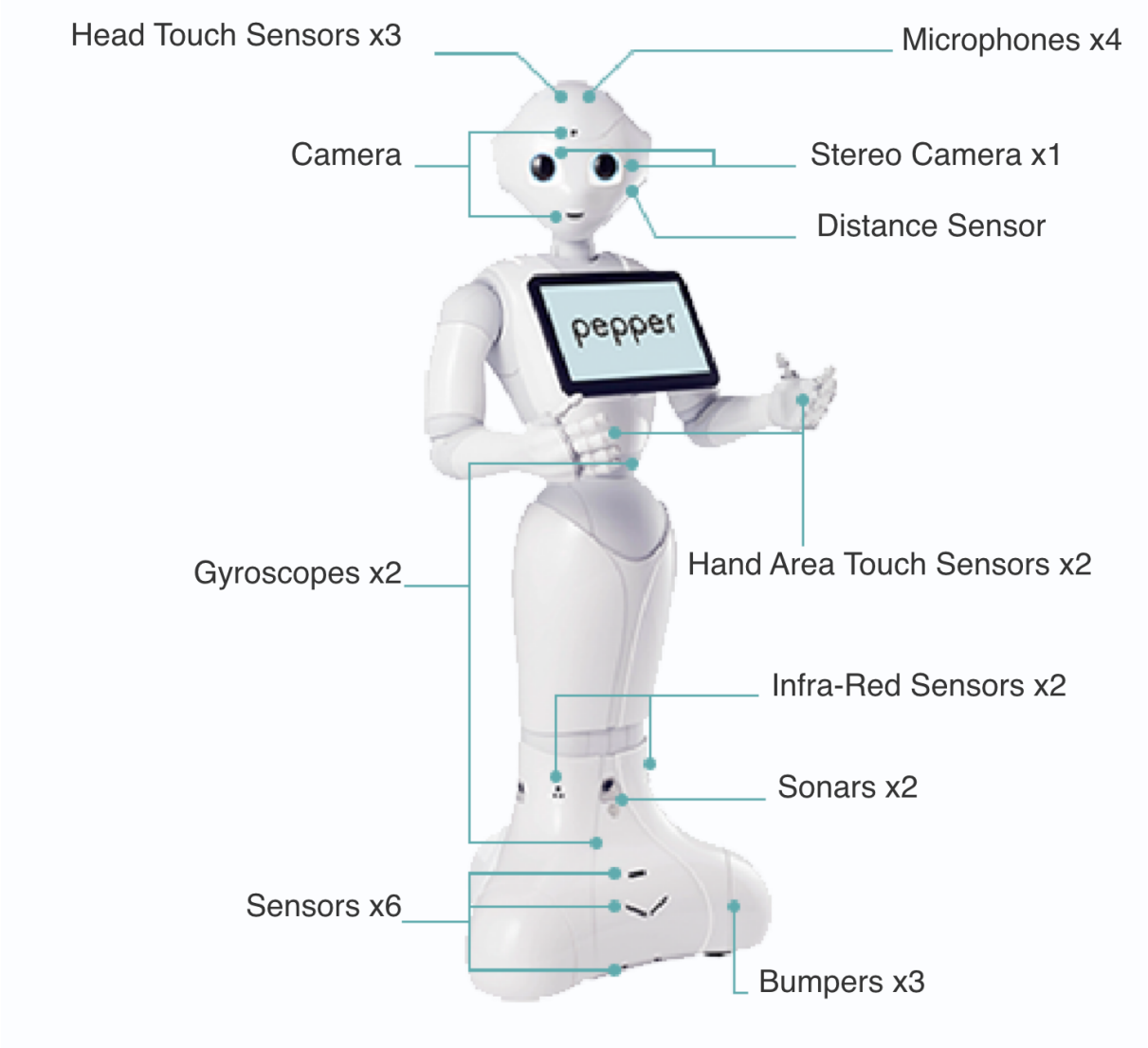



Figura 2.4 – Robô Pepper e suas especificações. Fonte: SoftBank

possui em comum aos outros, é o fato de estarem disponíveis no ROS, mostrando que utilizar o ROS como *framework* facilita a implementação dos métodos em diferentes robôs. Foram utilizadas as informações de profundidade das câmeras, junto a hodometria para realizar o mapeamento, onde o *Gmapping* possui uma performance melhor.

Uma alternativa encontrada ao Pepper, é o robô TIAGo (Figura 2.5 (PAGES; MARCHIONNI; FERRO, 2016), uma arquitetura modular de *hardware* e *software*. Ele é composto por uma câmera RGB-D, um *laser range finder* e um sonar. Ele foi desenvolvido para realizar a localização e o mapeamento com o lidar2D por motivos de segurança. Mas da mesma forma que as aplicações utilizadas no Pepper, TIAGo também disponibiliza essa plataforma compatível com ROS, o que incentiva a utilizar técnicas de localização da robótica ao robô.

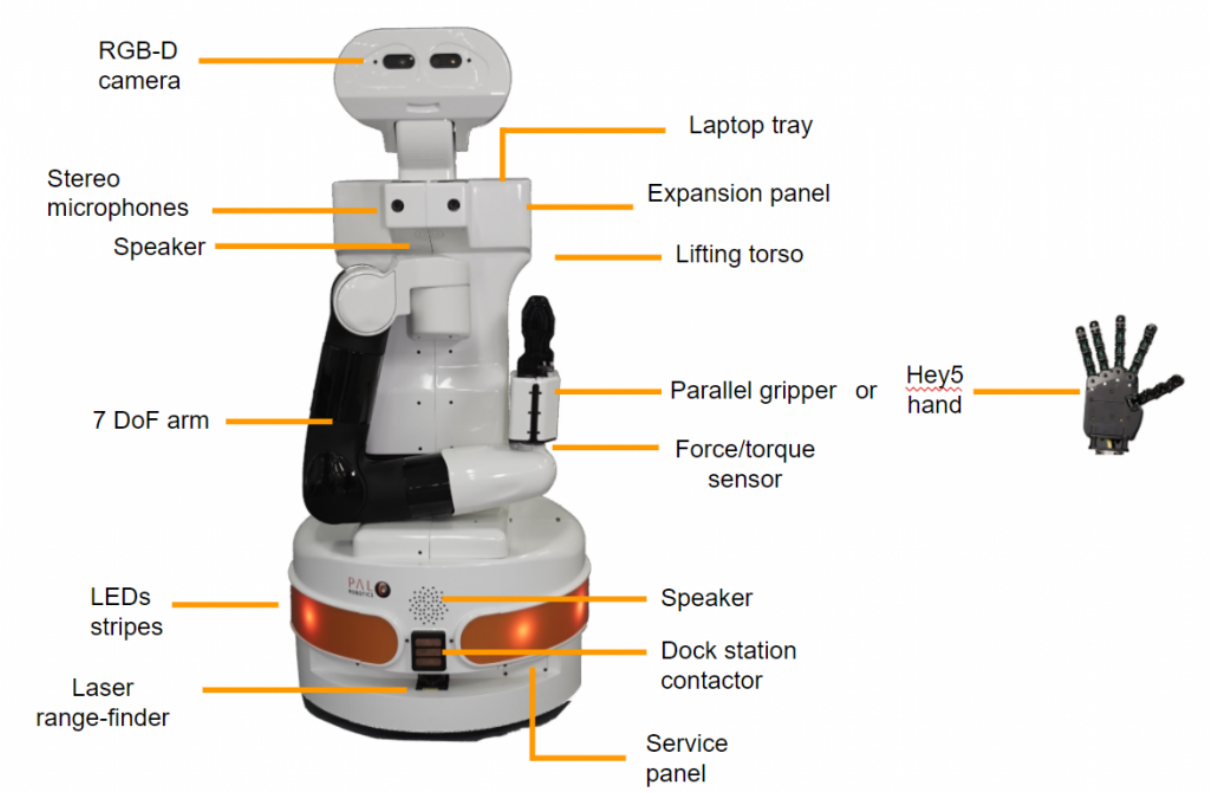



Figura 2.5 – Robô TIago e suas especificações. Fonte: PAL–Robotics

## 2.2 Sensores

Nesta Seção serão apresentados alguns dos principais métodos de detecção de *Landmarks* encontrados na literatura. Neste trabalho serão enfatizados métodos para os sensores listados abaixo:

- *Lidar* 2D
- Câmera monocular

### 2.2.1 *Lidar* 2D

O *Lidar* 2D é um sensor que realiza a medição da distância entre ele e um determinado objeto em duas dimensões dentro do seu campo de visão. Sua utilização dentro da área de localização foi um divisor de águas para que muitos robôs de serviço sejam lançados, devido as técnicas implementadas liderarem em fatores como confiabilidade e eficiência computacional para localização nas últimas décadas. Seu custo financeiro pode ser muitas vezes exorbitante, entretanto existem soluções em que o sensor é de mais baixo custo, como o RP *Lidar*, mas possuem maiores restrições em termos de alcance, alguns exemplos podem ser vistos na Figura 2.6 (FILIPENKO; AFANASYEV, 2018).


(a) RP *Lidar* A1


(b) Sick LMS111

Figura 2.6 – Exemplos de *Lidar* 2D comerciais.

### 2.2.2 Câmera Monocular

A câmera monocular é um dos principais sensores, visto que com a explosão do uso de câmeras no mercado de celulares, a relação entre performance e preço aumentou exponencialmente. De fato, câmeras monoculares, se tornaram o melhor custo beneficio, mas ainda são sensíveis a condições de luz extremas e podem necessitar um grande poder computacional para lidar com a quantidade grande de dados brutos que elas gerenciam, alguns exemplos podem ser vistos na Figura 2.7.

Em comparação a outros sensores visuais, quando o aspecto é hodometria visual, a câmera monocular ainda é a melhor escolha para sistemas que possuem baixo poder de processamento, visto que não possui limitações de profundidade e por possuir um processo de calibração simples. Os métodos existentes na literatura referentes ao SLAM monocular são classificados em duas categorias: método baseado em *features* e métodos diretos (BRASCH et al., 2018; XU; RONG; WU, 2021).


(a) Kinect (b) WebCam tradicional

Figura 2.7 – Câmeras monoculares

## 2.3 *Simultaneous Localization and Mapping (SLAM)*

A base da robótica móvel e autônoma se baseia em criar mapas precisos do ambiente, mas para que isso possa ser alcançado também há a necessidade de estimar a pose do robô de maneira precisa. Da mesma forma, para obter uma estimação da pose precisa é necessário ter um mapa preciso. Na robótica, tais técnicas são conhecidas como SLAM (*Simultanous Localization and Mapping*), as quais ao longo dos anos diversas técnicas foram propostas e desenvolvidas. O problema do SLAM na robótica é definido um problema tridimensional, onde o robô é capaz de se movimentar no plano, correspondendo a um mapa bidimensional (GAILIS, 2015).

O problema do SLAM não é considerado algo apenas da robótica, é um problema geral e que está presente em várias áreas, podendo ser aplicados a uma gama de objetos em movimento em uma grande variedade de ambientes nas quais as diversas técnicas desenvolvidas ao longos dos anos podem ser aplicadas. Um exemplo bastante claro disso, é a utilização de SLAM para cirurgia invasiva, onde utiliza–se a técnica para mapear a cena da cirurgia dentro do corpo humano (MOUNTNEY et al., 2006), conhecido como FootSLAM. Esta técnica é baseada na teoria Bayesiana, estimando a pose de um pedestre que caminha em um ambiente desconhecido a partir de informações inerciais.

Para a robótica, qualquer sensor que possa ser acoplado a um dispositivo móvel pode ser utilizado como base do SLAM, tendo só o algoritmo adequado para sua aplicação. Devido ao foco deste trabalho ser na utilização de sensores como *Lidar* e câmera monocular, aplicados a algoritmos probabilísticos, o estudo apresentado terá maior enfoque nesta área. Entretanto alguns exemplos extras serão disponibilizados visando apresentar que o problema pode ser solucionado de outras formas.

Diversas técnicas foram apresentadas ao longo dos anos, mas as mais consolidadas são aquelas baseadas como um processo de Markov, onde considera-se o robô em um ambiente com estimativas estocásticas. Sendo possível estimar a pose seguinte

utilizando apenas a localização atual e as ações executadas pelo robô a partir deste local. Os métodos fundamentados nesse tipo de processo são considerados probabilísticos, dentre eles é possível destacar os baseados em filtro de Kalman e filtro de partículas (THRUN et al., 2005).

Existem variações filtro de Kalman na literatura, mas dentre elas é possível destacar o *Extended Kalman Filter* (EKF) e o *Unscented Kalman Filter* (UKF) SLAM. Ambos têm a vantagem de serem aplicados a sistemas que possuem não linearidades. Seus algoritmos se diferenciam ao tratar a não–linearidade, enquanto o EKF lineariza, o UKF utiliza Signma–Points. Mas no resultado entregue, há um vetor de estado, onde ficam salvos a pose do robô e a localização dos *landmarks* observados, então a partir do sinal de controle do veículo e da medição dos sensores é possível calcular o ganho de Kalman e consequentemente atualizar a pose estimada do robô e dos *landmarks* (THRUN et al., 2005; BAILEY et al., 2006). A sua utilização não é limitada a um tipo de sensor, podendo ser utilizada tanto com câmeras e/ou *lidars*. O UKF possui uma maior precisão, mas maior custo computacional, favorecendo a escolha do EKF.

Ao longo da literatura é possível encontrar muitos exemplos de aplicação de filtros de Kalman, que possuem a mesma metodologia, alterando apenas qual *feature* está sendo detectada. Um exemplo é o trabalho de (LV et al., 2014) que utiliza *lidar* 2D, aplicando o algoritmo *Split and Merge* e o EKF–SLAM . Utilizando apenas câmera, como em (SMITH; REID; DAVISON, 2006) e (GEE; MAYOL-CUEVAS, 2006), ambos extraem linhas retas como informação, diferindo entre eles por utilizar EKF e UKF, respectivamente. E também é possível encontrar trabalhos onde foram utilizado dois métodos de detecção, como em (SUN et al., 2010), que utilizou a fusão via EKF entre as *features* do lidar com retas verticais extraídas da imagem.

Já o filtro de partículas, que tem como premissa principal ter várias possíveis estimações a partir de amostras aleatórias. A parte chave está na reamostragem dessas partículas, onde aquelas as quais possuem uma estimação mais fidedigna possuem mais chance de serem mantidas. Os métodos mais conhecidos da aplicação de filtro de partículas são o AMCL (*Adaptive Monte Carlo Localization* (AMCL)) (CHUNG; LIN, 2021) e o *Rao-Blackwellized Particle Filtering* (RBPF) (STACHNISS; GRISETTI, 2007).

Por fim, tem–se o *Factored Solution to the Simultaneous Localization and Mapping* (FASTSlam), que mistura as vantagens do filtro de partículas e do filtro de Kalman. Sua principal vantagem é não ter um grande vetor de estados para o filtro de Kalman e sim, um pequeno EKF para cada *landmark*. Computacionalmente é algo interessante, a medida que o número de *features* vai aumentando (MONTEMERLO et al., 2002). Da mesma forma como o EKF–SLAM, este método pode utilizar diferentes sensores, com lidar no trabalho original, com SURF *features* (GIUBILATO; PERTILE; DEBEI, 2016) e com marcadores artificiais (PROCOPIO, 2020).

### 2.3.1 Algoritmos de referência na literatura

Existem diferentes algoritmos *open–source* que já estão consolidados, dentre eles é possível destacar alguns para cada tipo de sensor. Ao utilizar o *Lidar* 2D como sensor único, é possível dar ênfase em dois principais algoritmos, o *Gmapping*, um método baseado em filtro de partículas, onde cada partícula carrega um mapa individual do ambiente (STACHNISS; GRISETTI, 2007). Sua desvantagem é o fato de se basear bastante na hodometria, algo que tem que ser refinado até ser confiável. O outro, é um algoritmo bastante consolidado, o Hector SLAM, baseado em *scan matching* para atualizar o mapa e para atualizar a pose utiliza a fusão da IMU, com o erro do *scan matching*, dentre outros sensores se presentes, em um EKF. Um fator positivo, é que ao se utilizar da distância entre ponto a ponto para obter maior precisão, a hodometria não é utilizada (KOHLBRECHER et al., 2013).

Quando a questão é o SLAM monocular, o algoritmo que representa o mais alto nível do estado da arte é o ORB–SLAM3, seu método é descrito em (MUR-ARTAL; MONTIEL; TARDOS, 2015), no qual ORB *features* são retiradas da imagem e utilizadas para rastreamento, mapeamento, relocalização e *loop–closing*. Todas as etapas mencionadas rodam em paralelo, tornando o sistema mais eficiente e confiável.

Todos os métodos acima, tiveram inicialmente o foco em utilizar inicialmente *landmarks* naturais, em contraponto SPM–SLAM e o UcoSLAM propõe métodos onde *landmakrs* artificiais são pontos chave (MUNOZ-SALINAS; MARÍN-JIMENEZ; MEDINA-CARNICER, 2019; MUÑOZ-SALINAS; MEDINA-CARNICER, 2020). Ambos os trabalhos são dos mesmos autores, inicialmente propondo o SPM–SLAM, que em um ambiente com vários marcadores ArUcos espalhados gera o mapa a medida que vai lendo os marcadores, resolvendo o problema de ambiguidade dos marcadores através do uso dos *frames* anteriores, sem necessitar de informações inerciais, diferente do apresentado em (NEUNERT; BLOESCH; BUCHLI, 2016).

Por fim, um dos mais novos trabalhos da área, o UcoSLAM, combina as qualidades do SPM–SLAM em conjunto com *landmarks* naturais retirados diretamente do *frame* da câmera. Seus resultados quando comparados a métodos puramente monoculares como o ORB–SLAM2 e o LDSO em questões como velocidade e precisão, são melhores principalmente em ambientes com situações ambíguas, como escritórios e fábricas. Tais métodos devido a sua complexidade computacional, podem não ser eficientes quando realizam a etapa de mapeamento e localização simultaneamente. Visto isso, é possível que os métodos mais clássicos como EKF e FastSLAM, que se utilizam da alta precisão da detecção dos marcadores, diminuam a covariância da pose do veículo e dos *landmarks* (NEUNERT; BLOESCH; BUCHLI, 2016; JÚNIOR et al., 2021).

## 2.4 SLAM monocular direto

As abordagens baseadas em *features* tem tido mais destaque na literatura, porém os recentes avanços no campo do SLAM visual tem mostrado que tais métodos perdem sua robustez quando na imagem há poucas *features*. Visto isso, os métodos diretos realizam a estimativa do movimento em conjunto ao minimizar o erro fotométrico no alinhamento direto da imagem. Em especial, destaca–se o LDSO (GAO et al., 2018), o qual apresenta todas as vantagens do método DSO (ENGEL; KOLTUN; CREMERS, 2017) e adiciona um sistema de fechamento de ciclo, o que auxilia em remover o *drift* acumulado em translação global, rotação e escala, o que tornava a longo prazo a trajetória da câmera e o mapa imprecisos, limitando as aplicações do método para curto prazo.

Entretanto, sistemas monoculares possuem pontos fracos, conhecidos como ambiguidade e similaridade. Em resumo, significa que se o SLAM for realizado apenas com as imagens monoculares, não é possível determinar a estrutura 3D, movimento da câmera e uma informação métrica absoluta, não podendo definir se a reconstrução realizada é real ou apenas artificial (HARTLEY; ZISSERMAN, 2003). Para resolver esse problema, há a necessidade de informações adicionais sobre o ambiente, como por exemplo, altura da câmera e tamanho do objeto, ou também informações providas de outros sensores, como uma IMU ou GPS (BRASCH et al., 2018; XU; RONG; WU, 2021; ZHOU; DAI; LI, 2019).

O próprio ORB–SLAM3 já apresenta na sua atual versão, o método monocular inercial, visando a eliminação de tal problema (MERZLYAKOV; MACENSKI, 2021). Além disso, alguns trabalhos mostram que é possível adicionar informações com filtros de Kalman, gerando um ganho na velocidade de processamento (ALATISE; HANCKE, 2017; HARTMANN; HUANG; KLETTE, 2013). Outra solução encontrada na literatura, foi adicionar a informação de profundidade, com lidar 2D ou 3D, gerando resultados mais robustos a iluminação e em ambiente com baixa quantidade de *features* (ZHANG; SINGH, 2015; XU; OU; XU, 2018; GRAETER; WILCZYNSKI; LAUER, 2018).

## 2.5 Detecção de *Features*

### 2.5.1 Métodos de Detecção de *Landmarks* com o *Lidar*

O estudo de detecção de *Landmarks* a partir da leitura de um *Lidar* 2D é bastante consolidado, sendo possível encontrar inúmeros trabalhos de comparação entre os melhores métodos. Em (NGUYEN et al., 2007) é feita a análise entre os seis algoritmos mais famosos. Dentre eles é possível destacar o *Split and Merge* (SaM), apresentado em (BORGES; ALDON, 2004), incremental (FORSYTH; PONCE, 2002) e a regressão linear (ARRAS; SIEGWART, 1998). Os 3 foram os melhores em questão de velocidade e precisão. Mas, o qual possuí maior destaque é o SaM, possuindo uma velocidade bastante superior

aos demais, com uma baixa taxa de falsos positivos.

### 2.5.2 Métodos de Detecção de *landmarks* com câmera monocular

Na literatura é possível encontrar inúmeros algoritmos que retiram informações a partir da imagem, pois a informação provinda da câmera possui uma maior complexidade quando a comparada a sistemas mais simples como o *Lidar* (CAMPOS et al., 2021). De forma análoga, há como retirar retas e pontos que conseguem descrever o ambiente, a partir de arestas como em (SMITH; REID; DAVISON, 2006; GEE; MAYOL-CUEVAS, 2006) ou linhas do chão (ZHANG; SUH, 2011) ou também linhas verticais que representem cantos de paredes (SUN et al., 2010).

Detectar retas em uma figura, pode gerar *features* não padronizadas, desta forma existem algoritmos descritores de *features*. Pode–se destacar os três principais, *Scale Invariant Feature Transform* (SIFT), *Speeded Up Robust Feature* (SURF) e *Oriented Fast and Rotated BRIEF* (ORB). SIFT extrai *features* não variáveis com rotação e escala da imagem e robustas a distorção e mudança de iluminação. Sua principal desvantagem é o seu custo computacional, compensado com resultados mais precisos. (LOWE, 2004). SURF é proposto por (BAY; TUYTELAARS; GOOL, 2006) como um substituto do SIFT, apresenta menor custo computacional e maior eficiência, reduzindo a complexidade do vetor de *features*. Por fim, o que mais se destaca, a ORB *feature*. O algoritmo é baseado em outros dois descritores, o FAST e o BRIEF, tornando–os mais eficiente e preciso, por utilizar as vantagens de cada algoritmo. Assim, o ORB se sobressai quando comparado a SIFT e SURF (RUBLEE et al., 2011).

### 2.5.3 *Features* artificias

A utilização de apenas *landmarks* naturais torna o sistema não tão eficiente, principalmente em ambientes com baixa quantidade de *features*. Com isso trabalhos vem propondo a adição de *landmarks* artificiais, a fim de auxiliar a localização do veículo. No meio da visão computacional, *Fiducial Markers* se destacam, pois consegue entregar um ponte de referência ou medição a partir da imagem. Existem diversos tipos, como por exemplo ARTag, AprilTag, ArUco e Stag (KALAITZAKIS et al., 2021), na Figura 2.8 é possível ver exemplos de cada marcadores.

O grande diferencial de marcadores fiduciais é o fato de que o padrão adotado pelos marcadores é altamente distinguível, com grandes características visuais, tendo em muitos casos uma característica específica que funciona contra erros de detecção. Tendo o conhecimento do tamanho do marcador e a forma esperada, é possível identificar e localizar o marcador, sempre tendo padrões geométricos para codificar os múltiplos bits que são utilizados para identificação (KALAITZAKIS et al., 2021).

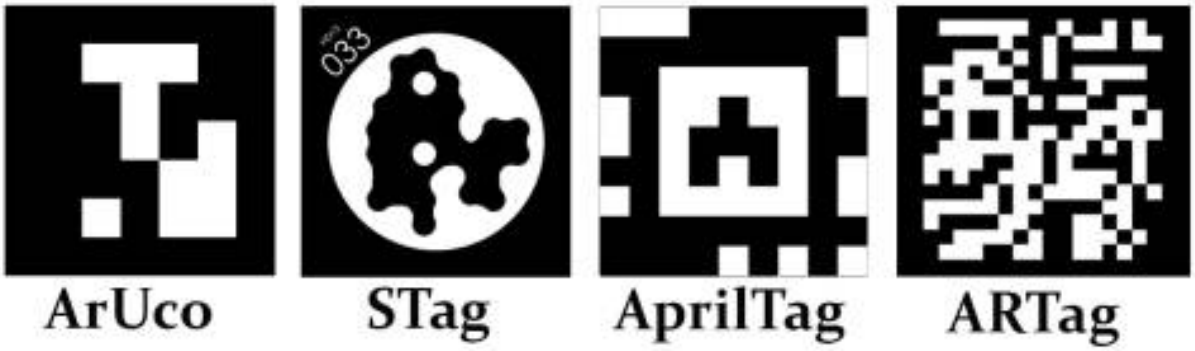



Figura 2.8 – Exemplos de marcadores fiduciais

A maioria de padrões quadrados é baseado no ARTtoolkit, que foi inicialmente criado para realidade aumentada em video conferência. Para determinar a posição do marcador, são identificadas regiões a partir de quatro segmentos de linha. É possível através do ARTtoolkit determinar padrões ou formas, que representaram identificações para cada marcador, utilizando uma correlação de imagens para identificar os diferentes marcadores (KALAITZAKIS et al., 2021).

O ARTag é o mais utilizado, devido a ser baseado na teoria da codificação digital para criar o padrão interno do marcador. Seu desenvolvimento visa garantir que a distância de Hemming seja mínima entre os pontos do marcador. Já o AprilTag, utiliza um algoritmo de segmentação de imagem que analisa os padrões do gradiente da imagem, que de maneira precisa podem estimar as linhas do marcador, entregando uma grande robustez (KALAITZAKIS et al., 2021).

Seguindo na linha de marcadores quadrados, pode–se apresentar o tipo ArUco, bastante popular por possuir como maior benefício o fato de um marcador já conseguir passar a informação necessária para definir a pose da câmera, assim como informações mais topológicas e semânticas para uso além da localização. Outro importante fator é que sua codificação binária o torna robusto, gerando a possibilidade de detecção e correção de erro. Outro fator importante, é o caso de facilitar o problema da relocalização e remover a ambiguidade visual, presente em alguns ambientes (MUÑOZ-SALINAS; MEDINA-CARNICER, 2020; NEUNERT; BLOESCH; BUCHLI, 2016). Devido a ser possível a configuração e haver disponível, para o trabalho em questão será utilizado o ArUco. Um marcador bastante interessante, é o STag, possuindo uma borda quadrada assim como outros exemplos, entretanto, no seu interior inclui um padrão circular. Detectando primeiramente, o padrão quadrado e ao encontrar o círculo no meio, é inicializando o processo de identificação do marcador, entregando uma melhor localização comparada aos outros.

# 3 Metodologia

Neste capítulo serão apresentados os materiais e métodos necessários para a definição de um sistema de localização robusto a ser aplicado em um robô de serviço. Para o desenvolvimento da proposta do projeto em localização robótica, é preciso definir e compreender as imposições dos sensores e algoritmos escolhidos. Após, assimilar e interpretar a modelagem dos métodos escolhidos para aplicar na finalidade do projeto. Com essas definições, a princípio será definido um ambiente de simulação para testes, onde seja possível utilizar um *framework* que possa ser exportado para um ambiente real e seja possível definir um veículo que comporte os sensores escolhidos para a implementação. Assim, são definidas as seguintes etapas metodológicas:

- Robô Móvel
- Sensores a serem utilizados
- Aquisição de *Features*
- Filtro de Kalman
- Filtro de Partículas
- EKF–SLAM
- FastSLAM
- Localização por EKF
- Métricas de avaliação

## 3.1 Modelagem cinemática do Robô

O Pioneer 3–DX, conforme apresenta na Figura 3.1, é um veículo leve, com dois motores, controle diferencial e bastante utilizado em laboratórios. Por ser versátil e adaptável é possível utilizá–lo como base para simulações de robótica, por esta razão será utilizado como veículo genérico, simulando um robô de serviço (MOBILEROBOTS, 2011).

O robô diferencial possui restrições de movimento, devido a possui limitações não-holonômicas (assume-se que as rodas giram sem deslizar). É um sistema sub-atuado, pois possui duas entradas (velocidade das rodas esquerda e direita) e três estados a serem controlados (x, y e $\theta$) (Chwa, 2010). A partir do controle das velocidades de ambas as rodas propicia o controle do robô.

O modelo do robô é utilizado para estimação da posição, como será feito neste trabalho. Para tal, verifica-se as velocidades de cada roda em cada iteração do processo, definindo desta maneira a velocidade linear e angular do robô, de acordo com as Equações 3.1 e 3.2.

$$V = \frac{V_R + V_L}{2} \tag{3.1}$$

$$\omega = \frac{V_R - V_L}{2b} \tag{3.2}$$

Onde:

$V_R$ : Velocidade linear da roda direita;

$V_L$ : Velocidade linear da roda esquerda;

$V$ : Velocidade linear do robô;

$b$ : Distância do centro de rotação até a roda.

Dividindo então as velocidades linear e angular pelo tempo de amostragem (discretização pelo método de Euler), é possível definir o quanto o veículo se movimentou ($\Delta s$) e qual a sua variação angular ($\Delta\theta$). Com isso encontra-se a função apresentada abaixo, que descreve o movimento do robô aproximado de $[x \;\; y \;\; \theta]^T$ para $[x' \;\; y' \;\; \theta']^T$:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta s sen(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix} \tag{3.3}$$

## 3.2 Sensores a serem utilizados

Conforme apresentado no Referencial Bibliográfico, há uma vasta quantidade de algoritmos e trabalhos baseado em *features* que propõe a utilização de câmeras e/ou lidares, obtendo resultados satisfatórios. A câmera monocular possui como principais desvantagens, a visibilidade limitada, escala ambígua e detecção de erro. Mas seu uso junto a um robô de serviço é essencial, pois com ela é possível obter mais informações topológicas e uma melhor interação com o ambiente (STUEDE et al., 2021; CHEBOTAREVA et al., 2021).

Visando então auxiliar e compensar os erros provenientes da câmera, a combinação junto a um *Lidar* 2D é recomendada. O *Lidar* pode entregar informações de profundidade de alta precisão, com uma larga área de detecção, mas não apresenta informações de textura, sendo então complementar a câmera monocular (KHURANA; NAGLA,

2021). O robô utilizado possui um Kinect com 640x480 de resolução e campo de visão de 57º e 43º, horizontal e vertical, respectivamente. O *Lidar* 2D utilizado, é o SICK LMS111-10100, com abertura de 270º, resolução de 0,5º e uma faixa de trabalho de 0,5m a 20m, na Figura 3.1 é possível ver o robô com ambos sensores.

Figura 3.1 – Pioneer 3–DX.

## 3.3 *Split and Merge*

Em concordância ao que foi apresentado no Capítulo 2, o método a ser utilizado é o *Split and merge*, pois seu estudo já está consolidado e sua performance é bastante superior, utilizando apenas de geometria básica.

Para passar os dados diretamente ao algoritmo será realizado um pré–processamento dos dados, com a finalidade de entregar um conjunto de dados contínuos para o algoritmo. Desta forma, pontos cegos, resultados de cantos sobressaídos e pontos que estejam fora do alcance do sensor não seja um problema. O processo foi feito como a seguir:

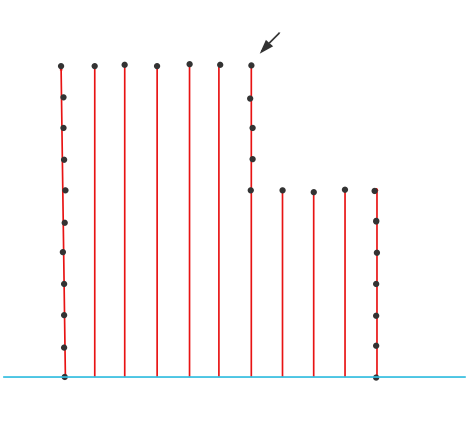
- Quando um dos pontos for acima da medição máxima do equipamento, haverá uma divisão dos dados naquele ponto, criando duas nuvens de pontos independentes, até eliminar todos esses pontos sem informação;
- Em caso de pontos oclusos, resultados de uma quina, será realizado a verificação da distância euclideana entre os pontos do laser, caso apresente uma distância maior que $\epsilon$, serão criados nuvens de pontos distintas, o processo será repetido até criar blocos de distâncias contínuas, visando não apresentar falsos positivos.

O algoritmo de *Split and Merge* é dividido em duas etapas, *SPLIT* e *MERGE*.

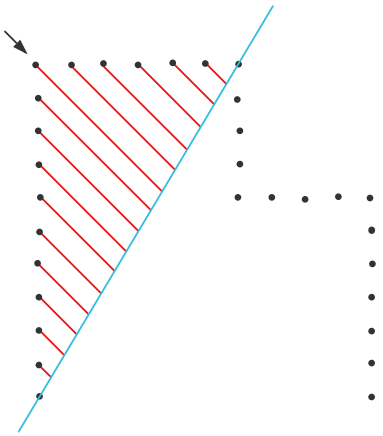
## Fase *SPLIT*

A partir de um conjunto de dados, é traçada uma reta entre o primeiro e o último ponto, conforme a Figura 3.2(a). A distância de cada ponto até a reta é calculada e caso o ponto mais distante seja maior que um $\epsilon$ pequeno, começa a próxima fase do algoritmo. É traçada uma reta entre o primeiro ponto e ponto mais distante da reta anterior (Figura 3.2(b)), desprezando os pontos momentaneamente, repetindo o processo anterior. Repete–se até que o ponto mais distante de uma reta não seja maior que $\epsilon$ e encontre-se a primeira reta (Figura 3.2(c)).

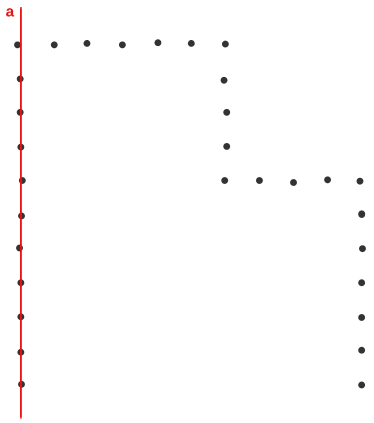
Então, os pontos que restaram são utilizados novamente no mesmo algoritmo, até que todas retas seja encontradas, conforme exemplificado na Figura 3.2(d).



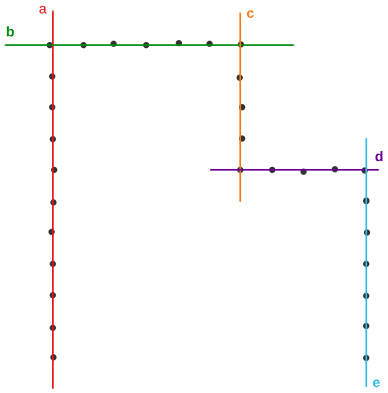
((a)) Reta entre os pontos extremos do conjunto inicial



((b)) Reta entre o ponto inicial e o ponto mais distante




((c)) Primeira reta é adquirida

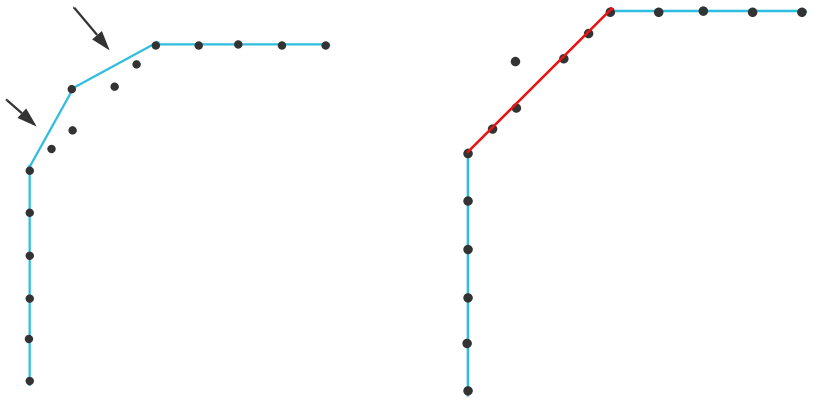



((d)) Processo é repetido, para encontrar todas retas presentes

Figura 3.2 – Fase SPLIT

### Fase *MERGE*

Esta fase é realizada após todas retas serem descritas, com a finalidade de eliminar possíveis retas próximas ou colineares. Ao encontrar retas muito próximas, uma linha comum entre elas é gerada, um exemplo é a Figura 3.3(a), onde as duas retas apontadas são bastante próximas, gerando uma nova reta em vermelho conforme apresentado em 3.3(b).



((a)) Algoritmo encontra retas quase colineares

((b)) Gera uma reta resultante

Figura 3.3 – Fase MERGE

## 3.4 Marcadores do tipo ArUco

Para auxiliar na estimação da pose e ao mesmo tempo proporcionar informações topológicas sobre o ambiente, serão utilizados marcadores do tipo ArUco. Sua grande vantagem, é que apenas um marcador consegue passar informação suficiente para obter a pose da câmera. Além disso, sua codificação torna o sistema robusto, abrindo a possibilidade de detectar erros e implementar técnicas de correção. Essencialmente, um marcador ArUco é um quadrado com borda preta e com uma matriz binária que determinar seu identificador, o que proporciona a informação topológica, um exemplo é apresentado na Figura 3.4 (GARRIDO-JURADO et al., 2014).

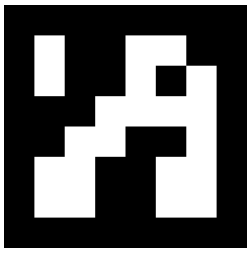


Figura 3.4 – Marcador Aruco

No processo de detecção, são adquiridas as posições dos quatro cantos da imagem e o identificador do marcador. Para detectá–lo basta seguir as seguintes etapas:

- A imagem é analisada, procurando por possíveis pontos na imagem, que sejam candidatos a ser marcadores;
- Realiza–se a verificação se os candidatos são realmente marcadores, a partir da codificação interna de cada um.

## 3.5 SLAM baseado em *Landmarks*

Esta seção tem um enfoque em apresentar todo o embasamento necessário para entender a metodologia por trás dos métodos probabilísticos baseados em *Landmarks*. Como comentado anteriormente, existem várias soluções que visam resolver o problema do SLAM nas três dimensões a serem resolvidas via métodos probabilísticos, como EKF–SLAM, AMCL e FastSLAM. A ideia principal por trás de todas essas técnicas é de representar explicitamente a incerteza utilizando os cálculos de incerteza, ou seja, não trabalhando apenas com a melhor escolha, mas com a distribuição que descreve a estimativa, representado um espaço de possíveis hipóteses. Desta forma é necessário entender, que não haverá uma certeza absoluta de onde o robô está, mas sim uma ideia que há a chance dele estar ali (THRUN et al., 2005). O robô e o seu ambiente são descritos através de alguns itens, sendo eles:

- **Estado:** Esse item engloba todas as informações tanto do robô como do ambiente que podem impactar o futuro, mas não sendo necessariamente fixo. Alguns exemplos são, a localização do robô, atuadores do robô, de objetos de interesse, paredes etc.
- **Interação com o ambiente:** A interação com o ambiente pode ser dado de duas maneiras, ou com os atuadores do robô ou através da leitura de sensores. O primeiro, considera–se como ações de controle, onde o robô altera o estado, seja sua posição, ou manipulando objetos. Já na parte do sensoriamento está a relação entre como o robô pode obter informações do estado do seu ambiente, por câmeras, lidars, etc.

O estado junto com as medições são governados por regras probabilísticas. O estado $x_t$ será gerado estocasticamente, sendo necessário especificar a distribuição de probabilidade do qual ele foi gerado. Desta forma, segundo (THRUN et al., 2005), a evolução do estado é dado pela distirbuição de probabilidade da seguinte maneira:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{3.4}$$

Onde considera–se $x$ o estado, $z$ as leituras de sensores e $u$ a ação de controle. O estado de $x$ completo no instante de tempo anterior consegue representar todos outros passos, desta forma é possível simplificar a representação. A partir do ponto de vista das medições, onde agora é necessário modelar a geração da medição, encontra–se:

$$p(z_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|z_t, z_t) \tag{3.5}$$

A simplificação é resultado de que o estado $x$ pode predizer todos os possíveis ruídos na medição. Estas duas distribuições conseguem descrever como os estados evoluem ao longo do tempo, a partir do controle exercido no robô $u$. A Eq. 3.4 representa a distribuição de probabilidade na transição de estados e a Eq. 3.5 representa a distribuição de probabilidade da medição, que não necessariamente depende do tempo $t$.

### 3.5.1 *Belief*

O conceito de *Belief*(crença), tenta expressar o conceito do conhecimento que o robô vai ter com relação sobre o estado do ambiente. O estado não pode ser medido diretamente, nem mesmo a pose do robô, sendo calculado através de inferências, que não representam a "verdade absoluta". Na robótica probabilística, os estados são representados como distribuições de probabilidade em relação ao estado verdadeiro

### 3.5.2 Filtro de Bayes

Os algoritmos de SLAM abordados nesse trabalho são implementações do Filtro de Bayes, desta forma apresentar o seu conceito é importante. Seu algoritmo é a implementação mais geral quando é visado o cálculo da crença de uma distribuição a partir de uma medição de sensores e um controle. A crença do no tempo $t$ é calculado a partir da crença do tempo anterior, com o valor de controle atual e a medição dos sensores (THRUN et al., 2005).

Sua execução pode ser dividida em dois grandes passos, primeiro é realizado o processamento do sinal de controle, junto a crença do estado anterior. A relação entre a distribuição de probabilidade do estado anterior junto a do controle induz uma transição para o estado atual, sendo chamado de predição. O segundo passo é conhecido como a atualização pela medição, na qual a distribuição é observado para cada hipótese a posteriori, gerando a nova crença (THRUN et al., 2005).

#### Suposição de Markov

Outro conceito bastante importante a ser entendido, que também exerce um papel fundamental no entendimento das técnicas probabilísticas, é a suposição de Markov.

Esta suposição afirma que o estado consegue representar tudo que aconteceu no passado, ou seja, ter o conhecimento da crença do estado é suficiente para representar a história do robô. Um ponto é que, existem inúmeros fatores que violam essa hipótese de Markov, tais como:

- Dinâmicas não modeladas;
- Falta de precisão na modelagem;
- Aproximação de erros;

Mesmo assim, na prática o filtro de Bayes se mostrou bastante robusto quanto a essa violações, entretanto esses aspectos não são modelados podem muitas vezes terem efeitos completamente aleatórios.

## 3.6 Filtro de Kalman

O Filtro de Kalman tem esse nome devido a ter tido seu desenvolvimento por Rudolf Kalman (KALMAN, 1960), o qual baseou seu trabalho em Thorvald Thiele e Peter Swerling, tendo contribuições posteriores de Richard Bucy. Basicamente, o Filtro de Kalman resolve o problema de estimação dos estados em sistemas lineares ou linearizados. Sendo a mais utilizada técnica, mesmo possuindo algumas deficiências bastante conhecidas, como por exemplo ter a necessidade de ser um sistema linear para garantir a propagação gaussiana.

A Figura 3.5 ilustra um sistema utilizando Kalman, no qual o estado do sistema é baseado no modelo dinâmico e a leitura dos sensores, onde ambos sofrem influência de erros não sistemáticos. O Filtro de Kalman implementa a computação por crença para estados discretos, a cada tempo $t$ terá uma crença representa com uma média $\mu$ e uma covariância $\Sigma_t$.

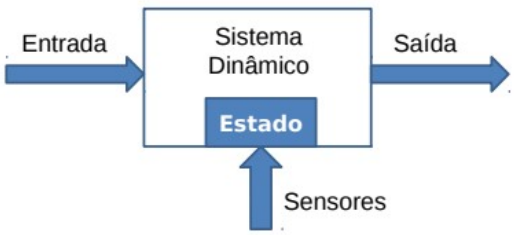



Figura 3.5 – Sistema dinâmico. Fonte: Eleri Cardozo

Em alguns casos, o Filtro de Kalman consegue ser definido como um estimador ótimo, ou seja, consegue minimizar a variância do estado estimado, sendo eles:

1. O sistema tem dinâmica linear;
2. O sistema é completamente observável (todos os estados podem ser estimados a partir do sensoriamento das saídas do sistema);
3. As leituras dos sensores são funções lineares do estado;
4. Os erros que afetam a transição de estado e a leitura dos sensores são independentes e possuem distribuições normais com média zero.

Para conseguir entender como o Filtro de Kalman funciona, um sistema linear e discreto é utilizado, descrito pelas Equações 3.6 e 3.7 .

$$x_k = Ax_{k-1} + Bu_k + q_k \tag{3.6}$$

$$z_k = Cx_k + r_k \tag{3.7}$$

Sendo as variáveis a serem observadas, os estados ($x_k$), controle de entrada ($u_k$) e as matrizes os parâmetros lineares que afetam as equações. Os vetores $q_k$ e $r_k$ são ruídos gaussianos brancos não correlacionados. O objetivo então é obter os estados descritos por $x$, com observações de $z$, ao considerar a matriz $C$ como uma matriz identidade, a Equação 3.7 é rescrita como:

$$x_k = z_k - r_k \tag{3.8}$$

O problema agora claramente é que o ruído $r_k$ não é conhecido, visto que é um ruido branco e aleatório por definição. Visando resolver esse problema, Kalman percebeu que é possível estimar o estado atual se levar em consideração tanto a observação atual como a crença do estado atual. Representando então o estado estimado por $\hat{x}_k$. o valor resultado pelo modelo $x$, pode-se expressá–lo uma relação com o estado estimado anterior junto à atual observação através da Equação 3.9, sendo $G_k$ uma matriz de ganhos.

$$\hat{x}_k = \hat{x}_{k-1} + G_k(z_k - Cx_k). \tag{3.9}$$

Para entender melhor a influência do ganho do Kalman, aplica-se os dois casos extremos. Primeiramente, caso o ganho for zero, obtém-se:

$$\hat{x}_k = \hat{x}_{k-1} + 0(z_k - C\hat{x}_{k-1}) = \hat{x}_{k-1} \tag{3.10}$$

Portanto, quando o ganho é zero, a observação atual não possui nenhuma influência no estado e o estado atual é igual ao anterior. Quando o ganho e a matriz $C$ são identidades, obtém-se:

$$\hat{x}_k = \hat{x}_{k-1} + I(z_k - I\hat{x}_{k-1}) = \hat{x}_{k-1} + z_k - \hat{x}_{k-1} = z_k \tag{3.11}$$

O estado atual da crença, estimando desta forma apenas utilizando a atual medição para a estimação do estado atual (BECKER et al., 2019).

Para implementação do filtro de Kalman em fusão sensorial, separa-se o filtro em duas etapas, predição e atualização, onde a primeira etapa realiza uma predição sobre a dinâmica do modelo e no segunda passo uma correção, atuando diretamente na covariância do erro. Desta forma, o filtro de Kalman funciona como um estimador otimizador do estado $x_k$ com a medição de $z_k$ e informações obtidas no instante atual, além de estimar também a atual covariância do do erro entre o sistema e sua crença (OLIVEIRA; GONÇALVES, 2017).

Contextualizando para o problema de um robô móvel, o modelo descrito em 3.1 é aplicado na fase de predição, para gerar uma crença sobre o sistema, já $z_k$ é a informação sobre os arredores do robô, sendo essa informação composta por pontos de características, que identificam objetos de interesse, sendo as informações providas pelo *lidar* e da câmera monocular. Visto isso, é possível detalhar a função de cada etapa do algoritmo do Filtro de Kalman:

**A etapa de predição é composta por:**

1. Calcular a média das variáveis de estado no instante atual $t(\mu_t)$ com base no estado obtido no instante anterior $(x_{t-1})$ e do controle imposto ao sistema no instante atual $(u_t)$;
2. Calcular a covariância no estado atual $\Sigma_t$ a partir da covariância no estado anterior $\Sigma_{t-1}$ (propagação de erros).

**E a etapa de atualização é composta por:**

1. Calcular o ganho do filtro $K_t$;
2. Efetuar uma medida dos sensores $(z_t)$;
3. Calcular a inovação $z_t - C_t\mu_t$;
4. Corrigir a média calculada ($\mu_t$ e a covariância $(\Sigma_t)$ a partir do ganho do filtro e da inovação;
5. Voltar para o início da etapa de predição.

### 3.6.1 Filtro de Kalman Estendido - EKF

Conforme foi descrito na Seção 3.6, o filtro de Kalman resolve o problema de estimar o estado de um sistema em tempo discreto, no qual é governado por uma equação diferencial estocástica e linear. Entretanto, quando o processo a ser estimado possui relações não lineares atreladas a ele é necessária a linearização da crença em média com o intuito de obter as matrizes para o cálculo da covariância, sendo referido então como Filtro de Kalman Estendido (WELCH; BISHOP et al., 1995).

O EKF teve sua primeira aplicação na missão Apollo para a lua e na volta. Desde então vem sendo bastante utilizando para solucionar problemas não–lineares, tendo seu sucesso dependente de ser um problema *quasilinear*, onde os erros gerados em linearizações não modeladas são significantemente menores que os erros da modelagem devido a incerteza dinâmica,e ruídos de medição (THRUN; BURGARD; FOX, 2000).

O conceito de aplicação da linearização é aplicado para que as propriedades Gaussianas da distribuição sejam mantidas ao propagá–la, pois as transições de estado e medição precisam ser lineares. Caso a probabilidade de estimação dos estados não for uma Gaussiana, este não poderá ser representado por sua média e covariância. Segundo (GREWAL; ANDREWS, 2014), filtros não lineares não podem ser implementados apenas utilizando informação de média e covariância, portanto na prática, tais filtros precisam de técnicas de aproximação. Vale ressaltar que, a linearização só é necessária em momentos que utilizam a distribuição, desta forma as equações não-lineares que representam o modelo podem ser utilizados para estimação e ao computar saídas de sensores (THRUN; BURGARD; FOX, 2000).

Seu algoritmo é bastante similar ao Filtro de Kalman, as predições lineares são trocadas pelas equações não lineares. Também, ao invés de utilizar as matrizes de estado A, B e C, são utilizados os jacobianos do modelo, $G_t$ e $H_t$. $G_t$ corresponde as matriz A e B e o jacobiano $H_t$ corresponde a C (THRUN; BURGARD; FOX, 2000).

## 3.7 Filtro de Partículas

Uma opção a filtros Gaussianos são os filtros não paramétricos. Diferente dos modelos Gaussianos, os filtros não paramétricos não focam em representar a posteriori por um curva e sim por um número finito de valores. Neste trabalho será analisado a utilização de filtros que aproximam o espaço de estado em amostras aleatórias a partir da distribuição a posteriori (THRUN et al., 2005).

O filtro de partículas se baseia em descrever a função a posteriori através de amostras aleatórias do estado a partir desta função, ou seja, representa a distribuição esperada com finitas amostras desta distribuição. Essa representação ela é definida como

uma aproximação, mas por não ser paramétrica, consegue representar uma maior gama do espaço de distribuições (THRUN et al., 2005). Cada amostra realizada da distribuição é definida como partícula e são definidas como:

$$\chi_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]} \tag{3.12}$$

Cada partícula é considerada uma hipótese de como representar uma estimativa do estado do sistema no tempo t. $M$ é definido como o número total de partículas, normalmente sendo um valor alto, para representar mais hipóteses com a intenção de melhor aproximar a estimativa através das partículas. Em um mundo ideal a hipótese de cada partícula deve ser proporcional a estimativa do filtro de Bayes, ou seja:

$$x_t^{[m]} \; p(x_t | z_{1:t}, u_{1:t}) \tag{3.13}$$

Desta forma, quanto mais populada de amostras uma região do espaço de estados for, haverá uma maior probabilidade do estado verdadeiro estar ali. Para construir e definir a estimativa a posteriori, assim como outros algoritmos Bayesianos, o algoritmo age recursivamente a partir do passo anterior. Como a distribuição em todo passo é definida por amostras, logo a distribuição no tempo $t$ é definida recursivamente de $t-1$ (THRUN et al., 2005).

O algoritmo de um filtro de partícula é realizado em três passos:

1. Para gerar uma hipótese para o tempo $t$, utiliza–se a partícula do tempo anterior junto com o controle $u_t$;
2. Para definir o quão correta aquela partícula possa estar, é calculado um fator de importância, a partir de medições de *landmarks* ($z_t$). O agregado do peso de todas as partículas irá representar a aproximação a estimativa do filtro de Bayes a posteriori;
3. O grande diferencial do filtro partículas está nesse último passo, a reamostragem. As partículas que possuem um maior peso, tendem a ser mantidas na distribuição, enquanto partículas com menores pesos, tendem a ser trocadas.

O passo de reamostragem possui uma função importante em fazer com que as partículas sigam a estimativa a posteriori,. É possível utilizar um filtro de partículas sem esse passo, mas desta forma muitas partículas iriam tender a regiões de baixa probabilidade, necessitando então de mais partículas para manter uma melhor aproximação da distribuição a posteriori(THRUN et al., 2005).

## 3.8 *Extendend Kalman Filter SLAM* (EKF–SLAM)

Esta técnica resolve o problema do SLAM ao explorar a utilização de um filtro de Kalman Estendido. O filtro de Kalman soluciona a questão de estimar o estado de um sistema em tempo discreto, no qual é governado por uma equação diferencial estocástica e linear. Entretanto, para aplicar a sistemas não–lineares, aplica–se o filtro de Kalman Estendido, onde é realizada a linearização da média para o álculo da covariância e do ganho de Kalman.

No EKF–SLAM, descreve–se em um vetor de estados, Equação 3.14, a pose do veículo e a pose dos *landmarks* encontrados ao longo do caminho. Sendo pose, um termo que representa posição e orientação de um sistema ou ponto de interesse. Na Equação 3.14, $x, t$ e $\theta$ representam a estimativa da pose do robô e $m_{i,x}$ e $m_{i,y}$ a estimativa da posição espacial de cada *landmark* $i$.

$$x_t = (x \quad y \quad \theta \quad m_{1,x} \quad m_{1,y} \quad \dots \quad m_{i,x} \quad m_{i,x})^T \tag{3.14}$$

O tipo de *landmark* afeta a sua precisão:

1. Se ao detectar um *landmark* é possível sua identificação com precisão, um identificador de um marcador ArUco, por exemplo;
2. *Landmarks* desconhecidos, ou seja, que não possuam rótulos, que possam ser identificados e portanto tendo de utilizar técnicas como distância mínima

O algoritmo do EKF–SLAM, é composto de três etapas, de predição, atualização e inovação. A etapa de predição, atualiza a pose do robô através do seu modelo cinemático e calcula a covariância do modelo. A etapa é descrita por duas principais Equações, 3.15 e 3.16.

$$\overline{\mu}_t = g(u_t, \mu_{t-1}, m) \tag{3.15}$$

$$\overline{\Sigma}_t = G_T \Sigma_{t-1} G_t^T + R_t \tag{3.16}$$

A etapa atualização é responsável pela identificação de novos *landmarks* a serem adicionados ao vetor de estados e para a contribuição dos *landmarks* já identificados para atualizar a pose do robô e a localização dos outros *landmarks* presentes no vetor de estados. Ou seja, cada *landmark* que está sendo visualizado novamente, irá contribuir para correção de todo o vetor de estados. O primeiro passo da etapa de atualização é computar a estimação da posição dos sensores em função da pose do robô e da posição dos *landmarks* presentes do vetor de estados.

$$\delta = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = \begin{bmatrix} \overline{\mu}_{k,x} - \overline{\mu}_{t,x} \\ \overline{\mu}_{k,y} - \overline{\mu}_{t,y} \end{bmatrix} \tag{3.17}$$

Onde $\overline{\mu}_k$ é a posição do *landmark* k e $\overline{\mu}_t$ a pose atual do robô. Desta forma calcula-se:

$$q = \delta^T \delta \tag{3.18}$$

Podendo desta maneira determinar a estimação da distância e angulação de cada *landmark* a partir do vetor estado.

$$\hat{z}_t^k = \begin{bmatrix} \sqrt{q} \\ arctan2(\delta_y, \delta_x) - \overline{\mu}_{t,\theta} \end{bmatrix} \tag{3.19}$$

A parte seguinte consiste em determinar o Jacobiano do modelo de observação (função h):

$$H_t^k = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial \theta} \\ \frac{\partial}{\partial m_x} \\ \frac{\partial}{\partial m_y} \end{bmatrix} \begin{bmatrix} z_t^1 \dots z_t^n \end{bmatrix} \tag{3.20}$$

$$H_t^k = \frac{1}{q} \begin{bmatrix} -\delta_x\sqrt{q} & -\delta_y\sqrt{q} & 0 & \delta_x\sqrt{q} & \delta_y\sqrt{q} \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix} F_{x,y} \tag{3.21}$$

Onde, $F_{x,y}$ é uma máscara utilizada afim de apenas utilizar o *landmark* em questão para o cálculo. Para finalizar esta etapa, basta calcular o ganho do filtro. Vale ressaltar, que esta etapa é executada para cada *landmark*, já presente no vetor de estados.

$$K_t^k = \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1} \tag{3.22}$$

Sendo, o $K_t^k$, conforme mencionado, a contribuição do *landmark* k para atualizar a pose do robô, a sua própria localização e a dos demais *landmarks*. $Q_t$ é definido como a variância do sensor em relação a distância e angulação, conforme pode ser visto em 3.23.

$$Q_t = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix} \tag{3.23}$$

Na etapa de inovação, o novo *landmark* não presente no vetor de estados, é adicionado, calculado a sua localização através da Equação 3.24.

$$\begin{bmatrix} m_{xk} \\ m_{yk} \end{bmatrix} = \begin{bmatrix} \overline{\mu}_{t,x} \\ \overline{\mu}_{t,y} \end{bmatrix} + \begin{bmatrix} r_t^k cos(\phi_t^k + \overline{\mu}_{t,\theta}) \\ r_t^k sin(\phi_t^k + \overline{\mu}_{t,\theta}) \end{bmatrix} \tag{3.24}$$

Com o ganho calculado e novos *landmarks* adicionados, basta então recalcular os valores do vetor de estados e a nova covariância encontrada, a partir de:

$$\mu_t = \overline{\mu}_t + K_t^k(z_t^k - \hat{z}_t^k) \tag{3.25}$$

$$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t \tag{3.26}$$

## 3.9 FastSLAM

O FastSLAM foi apresentado em (MONTEMERLO et al., 2002), tendo como principal premissa superar a principal limitação do EKF–SLAM que é a sua relação entre a complexidade computacional e o número de *landmarks* quadrática, mesmo que apenas um *landmark* esteja sendo visualizado. O problema do SLAM então é decomposto, em um problema de localização e em N problemas de estimação de cada *landmark*, que são condicionados pela pose do robô.

O FastSLAM é composto então de um filtro de partículas modificado para estimar a posteriori sobre o caminho do robô. Cada partícula é composta de N filtros de Kalman que estimam as N localizações dos *landmarks* condicionados pela estimação do caminho. Como resultado, foi obtido um filtro de partículas Rao–Blacwellized, gerando um tempo de processamento de $M \log N$, onde M é o número de partículas e N, número de *landmarks*.

A primeira versão do FastSLAM apresentou uma independência do cálculo da pose do robô, dependendo apenas do valor vindo da hodometria. Sua convergência ficou afetada, caso o erro hodométrico fosse alto. Visto isso, (MONTEMERLO et al., 2003) adiciona à estimação da pose, a observação atual do robô, propondo o FastSLAM 2.0, sendo o algoritmo que será seguido.

O algoritmo começa com a inicialização das partículas, nelas estão presentes, a sua pose, cada *landmark* observado por ela e seus respectivos filtros de Kalman. Na primeira etapa, é realizada a atualização da pose da partícula, a partir de atualização de estilo do EKF, gerando uma distribuição Gaussiana, com os seguintes parâmetros a serem calculados:

$$\Sigma_{s_t}^{[m]} = \left[G_s^T Q_t^{[m]-1} G_s + P_t^{-1}\right]^{-1} \tag{3.27}$$

$$\mu_{st}^{[m]} = \Sigma_{s_t}^{[m]} G_s^T Q_t^{[m]-1}(z_t - \hat{z}_t^{[m]}) + \hat{s}_t^{[m]} \tag{3.28}$$

onde,
$\hat{z}_t^{[m]}$, a estimação da medição;
$\hat{s}_t^{[m]}$, a predição da pose do robô;
$G_\theta$, o jacobiano em relação a $\theta$;
$G_s$, o jacobiano com relação a s;
$z_t$, a medição atual;

Por fim, o $Q_t^{[m]}$ é definido como:

$$Q_t^{[m]} = R_t + G_\theta \Sigma_{n_t,t-1}^{[m]} G_\theta^T \tag{3.29}$$

A próxima etapa é a atualização da estimação do *landmark* observado, seguindo o mesmo padrão apresentado na atualização do EKF–SLAM. Ressaltando o ponto de que cada *landmark* terá o seu próprio EKF 2x2. Por fim, uma etapa importante para todo algoritmo baseado em filtro de partículas é a reamostragem, onde inicialmente calcula–se o peso de cada partícula, conforme:

$$L^{[k]} = G_s P_t G_s^T + G_\theta \Sigma_{n_t} G_\theta^T + R_t \tag{3.30}$$

$$w^{[m]} = \sum_1^M \frac{1}{\sqrt{|2L|}} exp\left\{\frac{1}{2}(z_t - \hat{z}_t^{[m]})^T L^{-1}(z_t - \hat{z}_t^{[m]})\right\} \tag{3.31}$$

Então, implementa–se a amostragem estocástica universal, que pode ser exemplificada como uma roleta, onde cada seção é proporcional ao peso da partícula e tem o número de setas correspondente ao número de partículas, gerando mais escolhas e um sistema com pouca variância.

## 3.10 Localização por EKF com Landmarks conhecidos

Visando adicionar a informação dos marcadores artificiais, será adicionado um passo de atualização da pose e sua covariância a partir da detecção dos marcadores do tipo ArUco, gerando desta forma mais um Gaussiana que auxilia na estimativa da pose a posteriori. Seu algoritmo em essência segue os mesmo passos apresentados do EKF–SLAM, a grande diferença é que os pontos de interesse já são conhecidos, desta forma não há a necessidade de estimá–los e corrigir sua posição a cada iteração.

A etapa de predição é compartilhada junto ao EKF–SLAM e como não é necessário atualizar a localizações dos pontos de interesse, a etapa de atualização calcula diretamente o erro da medição com o valor real e então atualiza a pose e covariância. Nesta etapa então, de maneira análoga ao explicado no EKF–SLAM é utilizada a leitura dos sensores com o intuito de diminuir a incerteza em $x_t$. A grande vantagem de adicionar esse método é que a posição dos pontos de interesse é conhecida, o que aumenta a confiabilidade desta correção.

Não dependendo também da precisão da estimativa do *landmark*, é possível então definir as equações da etapa de atualização, iniciando pelo cálculo do ganho do filtro de Kalman:

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \tag{3.32}$$

$H_t$ é determinado como o jacobiano da função $h(x_t)$, definido pela Equação 3.33.

$$h(x_t) = \begin{bmatrix} \sqrt{(L_x - x_t)^2 + (L_y - y_t)^2} \\ arctan2(L_y - y_t, L_x - x_t) - \theta_t \end{bmatrix} \tag{3.33}$$

$L_x$ e $L_y$ representam a localização real de um *landmark* obtido do mapa. Calculando o jacobiano encontra-se:

$$H_t = \begin{bmatrix} -\frac{L_x - x_t}{\sqrt{q}} & -\frac{L_y - y_t}{\sqrt{q}} & 0 \\ \frac{L_y - y_t}{q} & -\frac{L_x - x_t}{q} & 1 \end{bmatrix} \tag{3.34}$$

Onde:

$$q = (L_x - x_t)^2 + (L_y - y_t)^2 \tag{3.35}$$

$x_t$, $y_t$ representam a pose atual do robô.

$Q_t$ é a parcela que representa os desvios padrões dos sensores, ou seja a incerteza dos mesmos. O tamanho de $Q_t$ e $H_t$ irá depender do número de *landmarks* encontrados. O $\bar{\Sigma}_t$ não precisa ser calculado, visto que, fora previamente calculado no estágio de predição, desta forma o ganho $K_t$ para o filtro é definido.

$$Q_t = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_{th}^2 \end{bmatrix}, \tag{3.36}$$

$\sigma_d$ e $\sigma_{th}$ são os desvios padrão do sensor para distância e angulação.

Então agora é a etapa que irá corrigir a média calculada, a partir da leitura dos sensores e o estado anterior, representada pela Equação 3.37.

$$\mu_t = \bar{\mu}_t + K_t(zsensor_t - zreal_t) \tag{3.37}$$

$zsensor_t - zreal_t$ é a relação entre a posição encontrada pelo algoritmo de detecção com a real posição do *landmark*. Para determinar $zsensor_t$, basta:

$$zsensor_t = \begin{bmatrix} zrange \\ zbearing \end{bmatrix} \tag{3.38}$$

onde,

$$zrange = \sqrt{(l_x - x_t)^2 + (l_y - y_t)^2}, \tag{3.39}$$

$$zbearing = atan2((l_y - y_t), (l_x - x_t)) - \theta_t. \tag{3.40}$$

Sendo então $l_x$ e $l_y$ os valores que o algoritmo de *feature detection* retorna para a posição dos *landmarks*. De maneira bastante similar, determina-se $zreal_t$:

$$zreal_t = \begin{bmatrix} Lrange \\ Lbearing \end{bmatrix} \tag{3.41}$$

onde,

$$Lrange = \sqrt{(L_x - x_t)^2 + (L_y - y_t)^2}, \tag{3.42}$$

$$Lbearing = atan2((L_y - y_t), (L_x - x_t)) - \theta_t. \tag{3.43}$$

$L_x$ e $L_y$ também são retornados pelo *feature detection*, sendo as posições reais dos *landmarks*. Com a média calculado corigida, basta corrigir a covariância através da Equação 3.44.

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t \tag{3.44}$$

Devido ao FastSLAM utilizar o filtro de partículas, o método de adicionar marcadores via EKF não é possível realizar, pois a pose estimada do robô não apresenta robô e sim com as n partículas representa a covariância. Desta forma, a adição dos marcadores artificiais será dada apenas no EKF–SLAM.

## 3.11 Localização por ArUco

Para avaliar e comparar os métodos aplicados, se faz necessário a implementação de uma medição de um método externo, o qual já foi validado e possua alta confiabilidade. Para isto trabalhos utilizam métodos baseados em marcadores, podendo ser ArUco (VáVRA, 2019) ou *Fiducial Markers* (EDWARDS; HAYES; GREEN, 2016). Então um marcador do tipo ArUco é colocado em cima do robô e uma câmera é dedicada somente para observar a movimentação do robô, guardando valores de x e y. O sistema de detecção é composto por uma AXIS 214 PTZ Network Camera, com uma resolução de 640x480 e 30 FPS, seu posicionamento no sistema pode ser visto na Figura 3.6.

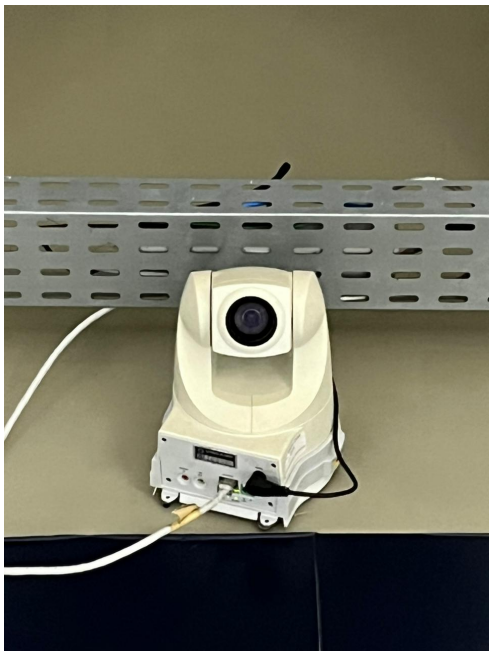


Figura 3.6 – Sistema para detecção da pose via ArUco. Fonte: Autor

Como o sistema de detecção foi construído com o intuito de entregar o melhor resultado possível, o marcador tem uma grande borda branca, o que facilita sua detecção e estabilidade. Na Figura é possível observar a visão entregue pela câmera e a detecção do marcador.

A detecção do marcador foi desenvolvida de maneira análoga ao utilizado na detecção dos pontos de interesse artificiais, dando foco apenas no marcador do robô. Para aquisição da imagem da câmera Axis foi utilizado o driver para ROS[1]. O driver retorna a informação da câmera, que é manualmente adicionada, e a imagem de *streaming* em sua versão comprimida, em menor qualidade. Utilizando o nodo do pacote "image_view"é possível retornar a qualidade inicial e então aplica-se ao algoritmo de detecção de pose do ArUco através do OpenCV. Na Figura 3.7 é possível encontrar o gráfico de nós do sistema é baseado na detecção do ArUco.

[1] Disponível em: https://github.com/ros-drivers/axis_camera

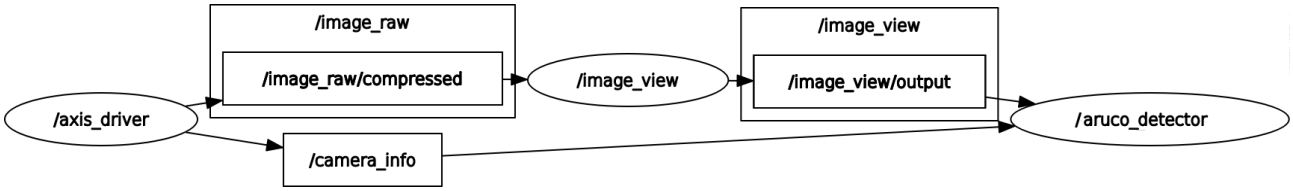



Figura 3.7 – Gráfico de nós no ROS para detecção do ArUco. Fonte: Autor

.

## 3.12 Métricas de avaliação

Para realizar uma avaliação diretamente objetiva, serão realizadas algumas métricas que visa apresentar tanto o funcionamento do algoritmo, no qual é possível perceber o controle da qualidade da estimação, assim como uma comparação junto ao método de detecção da pose via ArUco.

### Elipses de Erro

Para representar graficamente a propagação de erro ao longo do processo, foi determinada a utilização da elipse de erro, tal método consiste em apresentar uma elipse que representa o erro em x, y e $\theta$ ao longo do tempo em torno de um ponto, conforme apresentado pela Figura 3.8.

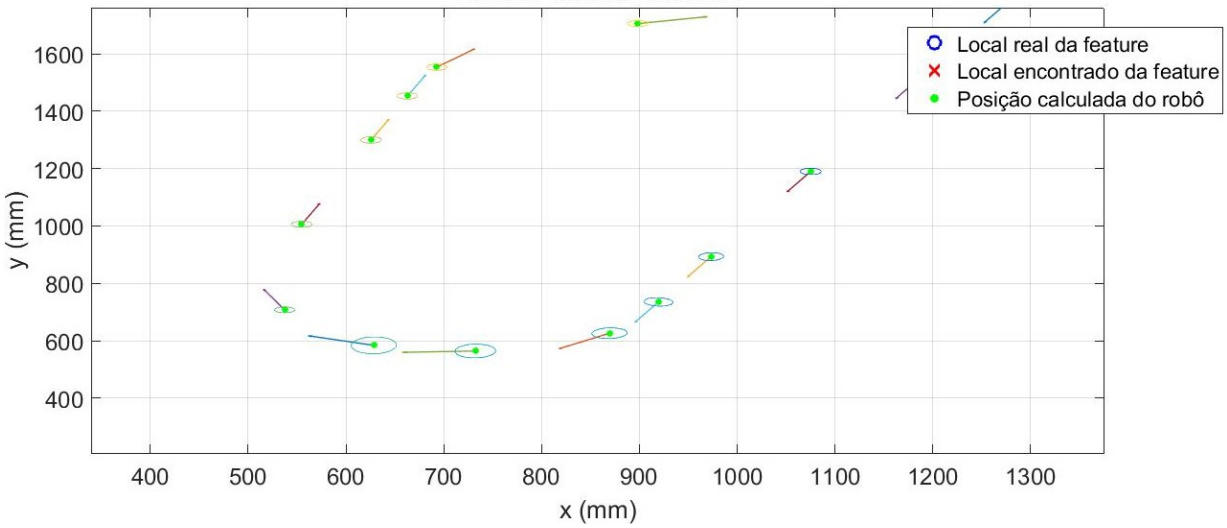



Figura 3.8 – Exemplo do método Elipse de erro.

Para seu cálculo, utiliza-se as Equações 3.45, 3.46, 3.47 e 3.48.

$$\Sigma_{xy} = \begin{bmatrix} \Sigma_x^2 & \Sigma_{yx} \\ \Sigma_{xy} & \Sigma_y^2 \end{bmatrix} \tag{3.45}$$

$$tan(2\beta) = \frac{2\sigma_{xy}}{\sigma_y^2 - \sigma_x^2} \tag{3.46}$$

$$a^2 = \frac{1}{2}[\sigma_x^2 + \sigma_y^2 + \sqrt{(\sigma_y^2 - \sigma_x^2)^2 + 4\sigma_{xy}^2}] \tag{3.47}$$

$$b^2 = \frac{1}{2}[\sigma_x^2 + \sigma_y^2 - \sqrt{(\sigma_y^2 - \sigma_x^2)^2 + 4\sigma_{xy}^2}] \tag{3.48}$$

Resultando na Elipse representada na Figura 3.9, onde a e b são os semi-eixos da elipse e $\beta$ o seu ângulo de inclinação.

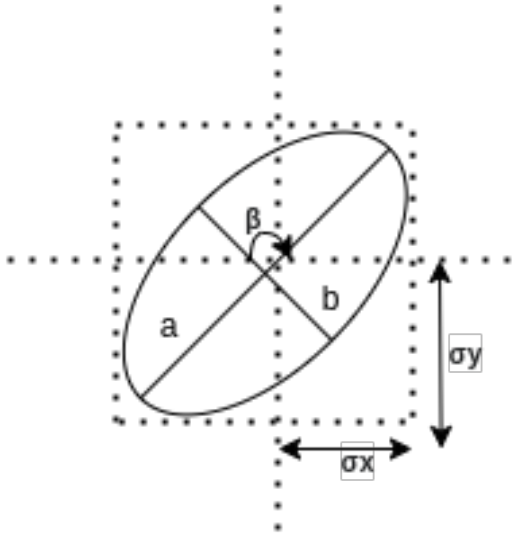



Figura 3.9 – Elipse a ser calculada.

## Erro médio quadrático - EMQ

Com a finalidade de apresentar a atuação dos métodos ao longo do tempo, o cálculo da distância do ponto esperado até o ponto calculado é realizado através do erro médio quadrático. Então, comparando a estimativa dada por cada método, junto ao valor retirado pelo método base, é encontrada uma métrica.

$$EMQ = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(\widetilde{x} - x_{gt})^2 + (\widetilde{y} - y_{gt})^2} \tag{3.49}$$

Sendo $\widetilde{x}$, a estimativa da pose atual e $x_{gt}$ o valor medido pelo sistema considerado como *ground truth*.

## Relação Número de Landmarks e EMQ

Esta métrica visa entender a influência do número de *Landmarks* detectados em relação ao erro para cada iteração, tentando desta forma perceber aspectos que possam

levar a melhoria do algoritmo. Uma imagem com mesma relação temporal será colocada, correlacionando erro e número de *Landmarks*.

# 4 Experimentos

Neste capítulo serão apresentados como os experimentos foram conduzidos e sua implementação. Apresentando os aspectos de como informação foi organizada e distribuída, quais foram os elementos de teste e como foram montados. Foram conduzidos dois experimentos, primeiramente em um ambiente de simulação e após em uma bancada experimental, utilizando o P3–DX.

## 4.1 *Robotic Operation System* (ROS)

O ROS é um *middleware* que fornece bibliotecas e ferramentas que auxiliam os desenvolvedores de *software* a criar aplicações robóticas.Com a presença de algoritmos do estado da arte e poderosas ferramentas de desenvolvimento, sendo livre e de código aberto. Com o ROS, há a abstração de vários recursos, como: controle de acesso a dispositivos, troca de mensagens entre processos, sincronismo e gerenciamento de pacotes, etc.

## 4.2 CoppeliaSim

O simulador de robótica CoppeliaSim (ROHMER; SINGH; FREESE, 2013) é um *framework* que permite uma simulação versátil e escalável, muito utilizado na área da robótica e simulações computacionais e possui uma API remota que permite a interação com o ROS. Desta forma, os primeiros testes foram realizados em simulação, para verificar a viabilidade do algoritmo e para integração do ambiente simulado com o real em trabalhos futuros.

Na Figura 4.1 é possível ver o ambiente de simulação construído no simulador, nela é possível analisar a posição dos marcadores. Serão conduzidos testes com e sem adição de marcadores. Utilizando a API, informações dos sensores do robô são enviados diretamente nos tópicos do ROS e então os algoritmos desenvolvidos conseguem realizar a detecção. Uma diferença em relação ao modelo real, é o *lidar* utilizado. Um *Lidar* 2D com $180^{o}$ de abertura será utilizado, com incremento de $0.4^{o}$. A simulação será conduzida com um tempo de amostragem de 50ms. Nas Figuras 4.2 e 4.3 é possível entender como fica o gráfico de nós de cada algoritmo. Para o EKF–SLAM, foram conduzidos três diferentes aspectos primeiramente somente com detecção do *lidar*, somente com os ArUcos e depois realizando a fusão entre eles. Vale ressaltar que quando utiliza apenas o ArUco trata–se apenas de um problema de localização, visto que os marcadores já são conhecidos.

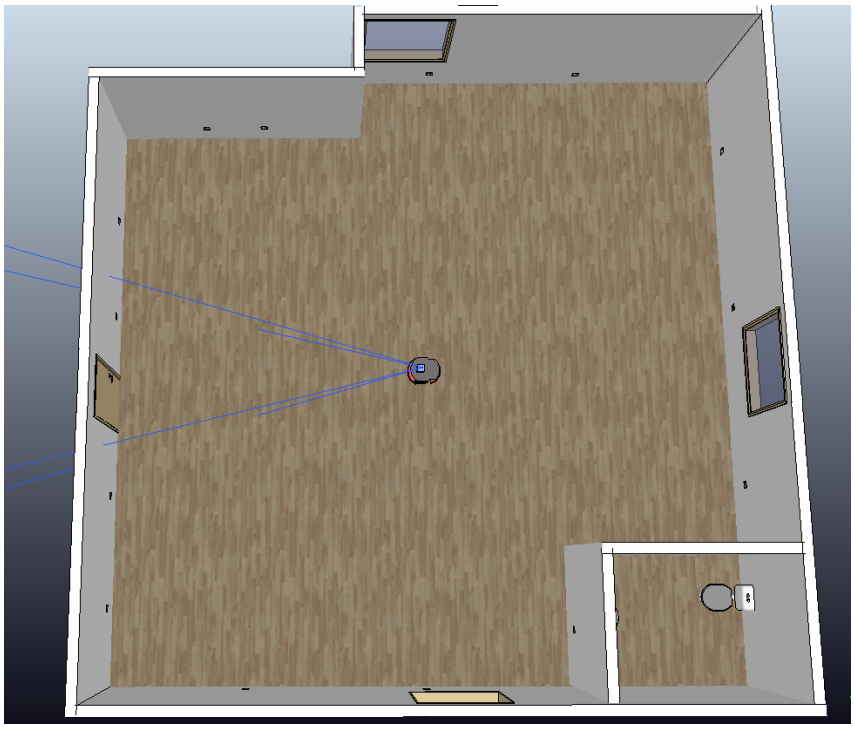


Figura 4.1 – Ambiente de simulação criado.

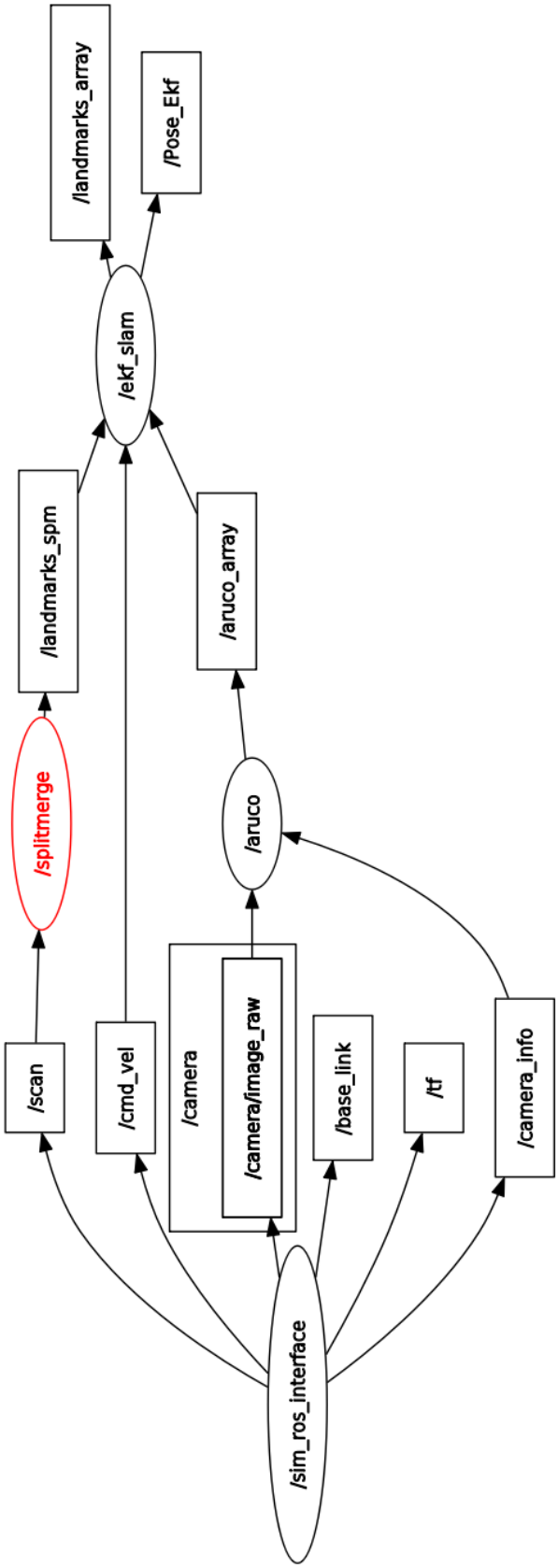



Figura 4.2 – Gráfico de nós gerado no ROS para computar o EKF–SLAM

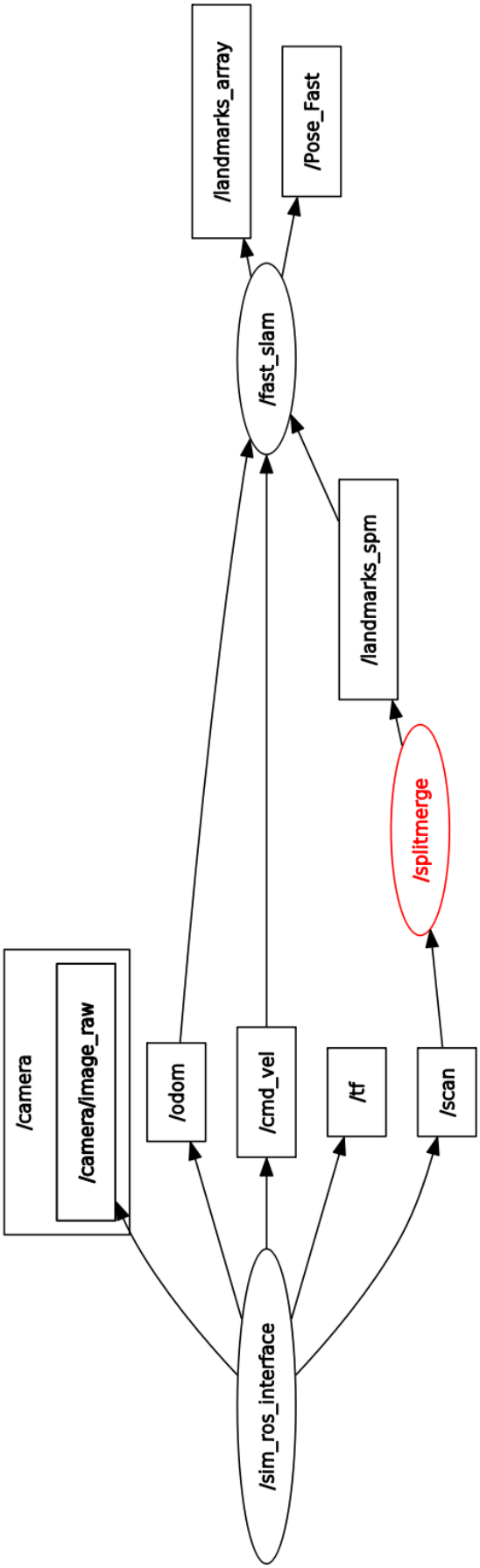



Figura 4.3 – Gráfico de nós gerado no ROS para computar o FastSLAM

## 4.3 Experimentos reais

A bancada de testes experimentais como apresentado na Metodologia, irá utilizar o robô Pioneer P3–DX que possui um *framework* restThru (JÚNIOR, 2016) que via conexão local é possível acessar as informações do robô. Um ambiente controlado, com marcadores foi construído para o robô andar, como é possível ver na Figura 4.4

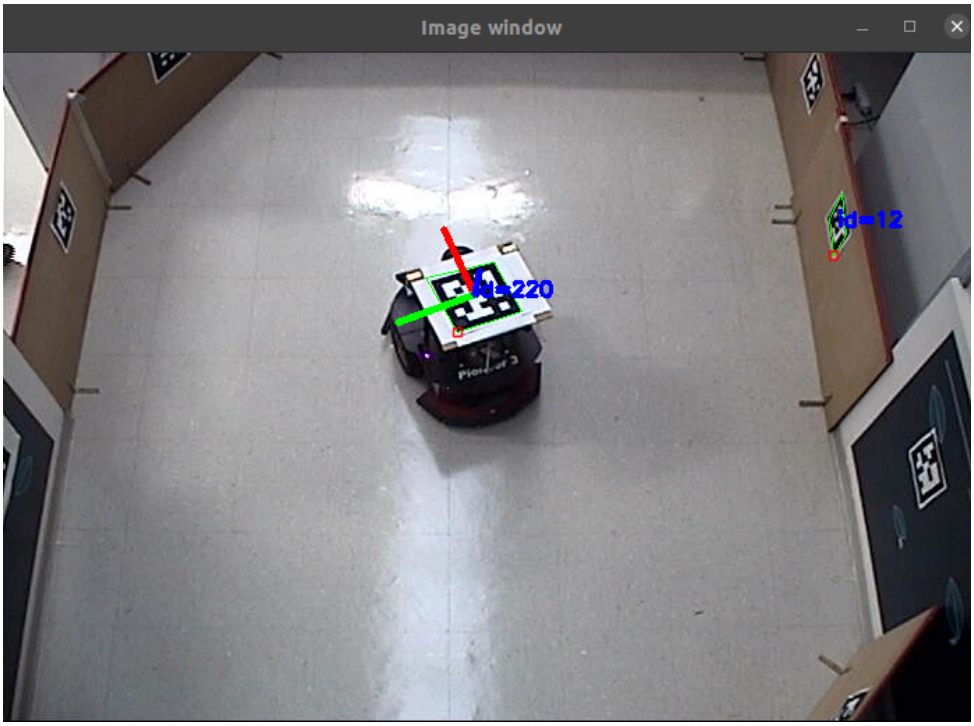



Figura 4.4 – Robô no ambiente real.

### 4.3.1 RestThru

O restThru nessa aplicação funciona como uma camada intermediária entre o controlador de baixo nível, com o computador que realizar o mapeamento, organizando as informações. Ele é uma aplicação RESTfull baseado em protocolos padrões da internet, entregando segurança. Utilizando protocolos HTTP é possível trocar informações entre a aplicação e o robô, não apresentando problemas com *FireWalls*. Sua utilização não limita o processo a não poder utilizar o ROS.

### 4.3.2 Plataforma de execução

O computador que será responsável por gerenciar os algoritmos tem as características apresentadas na tabela abaixo:

Para criar esta leitura dos sensores disponibilizados através do restThru, nodos no ROS foram criados publicando a informações nos tópicos para a aplicação dos algoritmos de SLAM. O veículo foi liberado a andar livremente pelo ambiente construído e então,

Tabela 4.1 – Confiuração do sistema

| Característica | Configuração |
|---|---|
| Processador | AMD® Ryzen 5 5500u with radeon graphics × 12 |
| Memória RAM | 5.6 GB |
| Sist. Op. | Ubuntu 20.04.6 LTS |
| ROS | Noetic |

Fonte: o autor.

uma *bag* do ROS foi salva para testar os algoritmos. Os gráficos gerados no uso com o simulador são bastante similares, mostrando que com a utilização do ROS, foi possível mudar de plataforma sem necessidade de mudanças nos algoritmos gerados. Os mesmo testes realizados na simulação, também foram conduzidos na parte experimental.

# 5 Resultados

Neste Capítulo serão apresentados os resultados encontrados tanto para simulação, quanto para o teste experimental. Primeiramente, serão apresentados os resultados de ambos os detectores de *features* propostos e por fim os algoritmos implementados. Para melhor entendimento e para tentar entender o comportamento dos algoritmos em cada ambiente, os resultados serão divididos em simulação e em experimentos. Ao fim de cada, o cálculo das métricas propostas.

## 5.1 Simulação

O experimento do robô como foi explicitado anterior foi colocado para andar no mapa livremente e ao longo do percurso os valores dos sensores foram adquiridos. Então, aplicou–se os algoritmos de detecção de *features* e de localização e mapeamento com os mesmos dados. A seguir serão apresentados os mapas gerados pelo algoritmos, as trajetórias captadas e por fim erro quadrático médio ao longo do tempo.

### 5.1.1 Mapas gerados

Os mapas gerados são compostos pelas *Features* detectadas pelo algoritmo *Split and Merge*, que já compõe um mapa. O propósito dos experimentos é comparar os resultados entre os dois métodos propostos. Tanto o EKF–SLAM, como o FastSLAM irão construir mapas referentes aos cantos da sala. O EKF–SLAM será testado somente com o detector de cantos, mas também com o ArUcos, comparados junto com o FastSLAM.

A partir da Figura 4.1, o robô era responsável por identificar os cantos da imagem e por consequência conseguir fazer o desenho da sala. Os três algoritmos conseguiram isso de maneira satisfatória ao se locomover pelo mapa. O mapa apresentado pela Figura 5.1, foi encontrado utilizando apenas o *Split and Merge*. Ao adicionar a detecção de ArUcos, encontrou–se o mapa na Figura 5.2, bastante similar, com pequenas mudanças. Por fim, aplicou–se o FastSLAM com 200 partículas, resultando na Figura 5.3. Nos três mapas, é possível perceber dois pontos que não fazem parte de cantos, mas são pontos na simulação onde se encontra um Marcador ArUco que há um relevo e devido ao algoritmo estar sensível detectou, não afetando o resultado final.

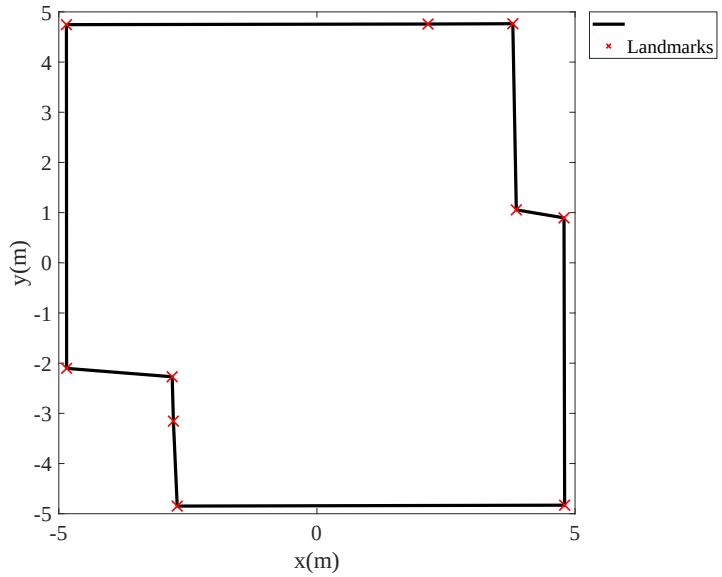



Figura 5.1 – Mapa gerado pelo EKF–SLAM em simulação apenas com *Landmarks* naturais (Simulação).

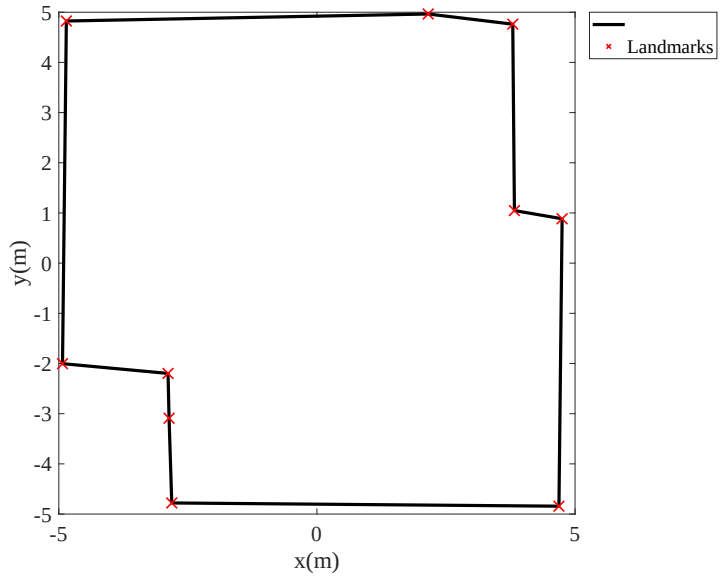



Figura 5.2 – Mapa gerado pelo EKF–SLAM em simulação apenas com adição das marcadores ArUco(Simulação).

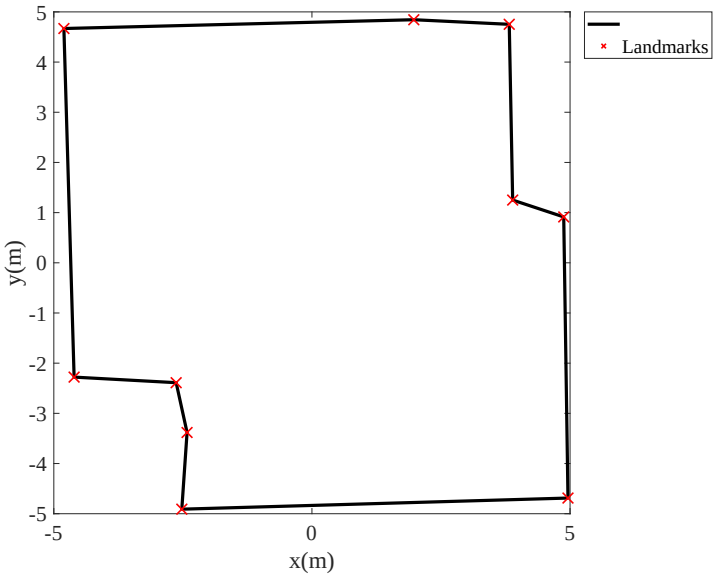



Figura 5.3 – Mapa gerado pelo FastSLAM em simulação com detecção de *Landmarks* naturais (Simulação).

### 5.1.2 Trajetórias Realizadas

O mapeamento e localização são realizados ao mesmo, com uma interdependência entre ambos algoritmos, com um bom mapa se tem uma boa localização e vice versa. Através dos mapas gerados, que possuem o formato esperado, é possível comprovar também a aplicação ao algoritmo *Split and Merge*, logo cabe ao robô se localizar no ambiente. Junto a pose estimada, para os métodos do EKF–SLAM que se baseia na distribuição de probabilidade para descrever a pose, foram adicionadas as elipses de erro para ver a atuação do EKF.

Para entender a trajetória esperada que os algoritmos tentem descrever, na Figura 5.4 é apresentado a trajetória *ground truth*, retirado a partir dos valores do próprio simulador, estes mesmos valores serão utilizados para o cálculo do erro médio quadrático.

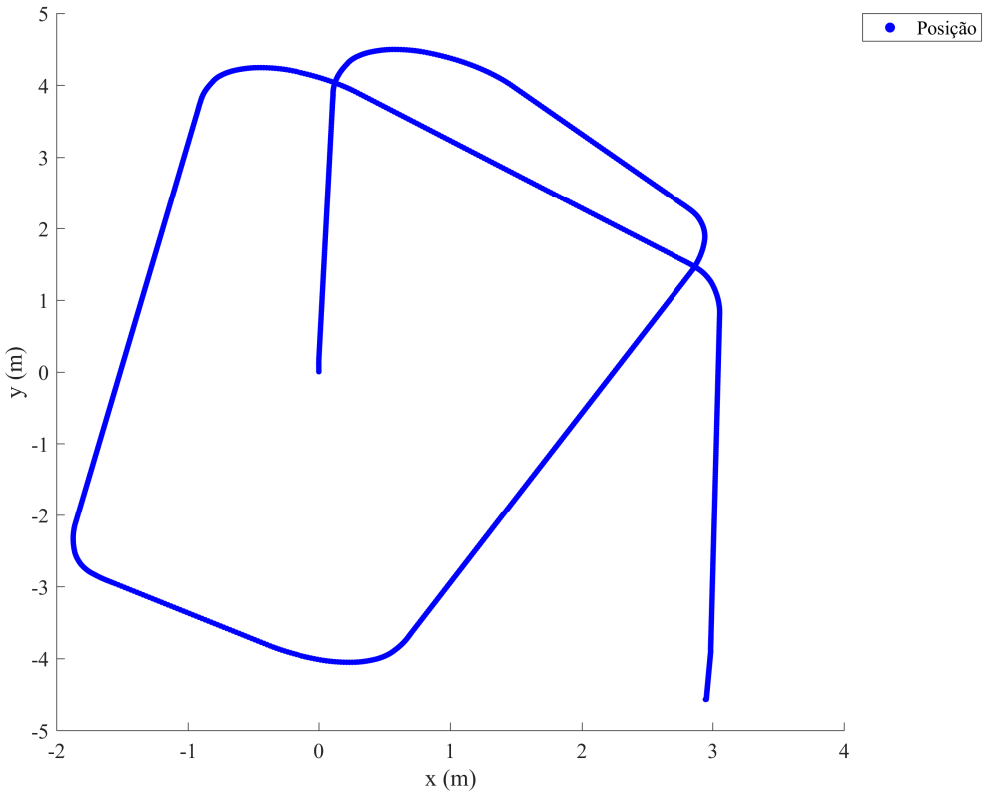



Figura 5.4 – Trajetória gerada pelo simulador, utilizado como *Ground Truth* (Simulação).

Como resultado, aplicou–se as duas variações do EKF–SLAM, só com *landmarks* naturais, combinado com ArUco e localização por EKF, baseada em ArUcos. Junto a eles, também é implementado o FastSLAM. A Figura 5.5 apresenta o resultado encontrado com o EKF–SLAM baseando em *landmarks* naturais. O resultado encontrado apresenta grande similaridade, mostrando que o algoritmo consegue realizar a estimativa da pose.

Na Figura ao redor de cada pose calculada há a elipse que fora previamente calculada, sendo mais perceptível durantes as curvas, onde não há uma uniformidade entre as poses.

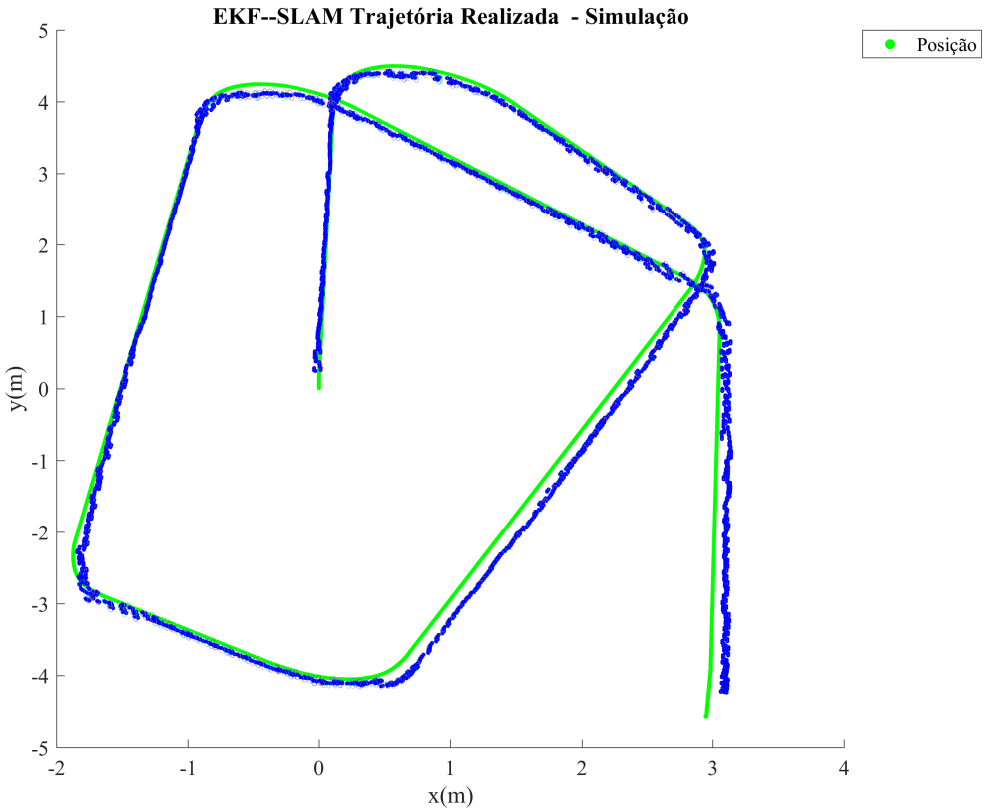



Figura 5.5 – Trajetória gerada pelo EKF–SLAM com *landmarks* naturais (Simulação).

Na Figura 5.6 obteve–se o resultado estimação da pose do robô apenas com o ArUcos, vale ressaltar que este método não realiza o mapeamento, apenas a estimação da localização, visto que a pose dos marcadores é conhecida. O resultado encontrado não se mostra tão satisfatório para se utilizar sozinho, percebe–se que as elipses de erro crescem bastante. Isto está diretamente relacionado a quando há a detecção ou não de marcador, caso não haja, o sistema é puramente baseado na hodometria das rodas, o que gera muito erro. Além disso, o mapa gerado foi bastante amplo, com marcador de 10mm, a detecção distância fica debilitada. Entretanto, percebe–se através das elipses que o algoritmo diminui a covariância da pose, gerando uma melhor estimativa.

Combinando ambos métodos encontra–se como resultado a Figura 5.7, não havendo grande diferença do método baseado marcadores naturais. Como o resultado encontrado utilizando somente o lidar já obteve um resultado bastante satisfatório, visualmente a adição do ArUco pode não adicionar tantos ganhos. Outro fator importante, por ser em simulação sensores de distância não possuem tanto ruído assim como na prática, entregando um alto nível de estimação e no caso da simulação a adição do ArUco gerou

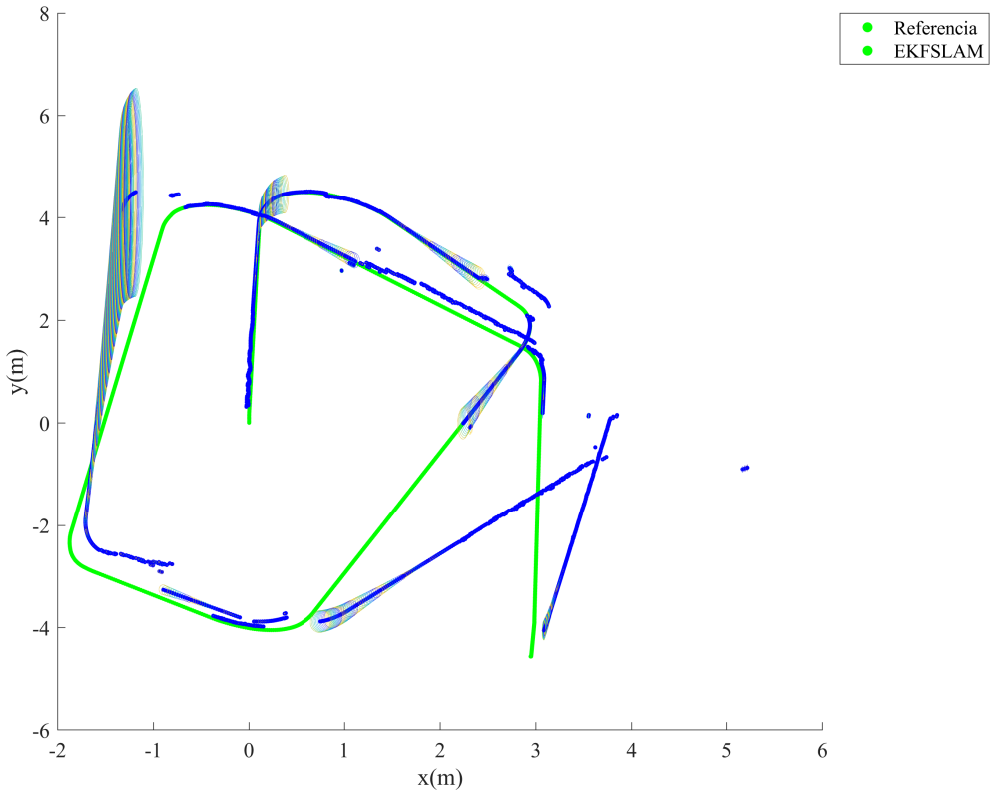



Figura 5.6 – Trajetória gerada pelo EKF–SLAM com *landmarks* artificiais (Simulação).

mais incerteza. As elipses presentes na Figura, quase que imperceptíveis mostram que a estimação é confiável e a covariância controlada.

Por fim, a Figura 5.8 é o resultado da estimação da pose do FastSLAM, percebe–se que diferente do EKF–SLAM, a pose é mais uniforme, entretanto quando comparado visualmente com o *Ground Truth* percebe–se que o algoritmo não alcançou uma boa estimativa dos valores. A cada curva realizada, o algoritmo se afasta da estimativa.

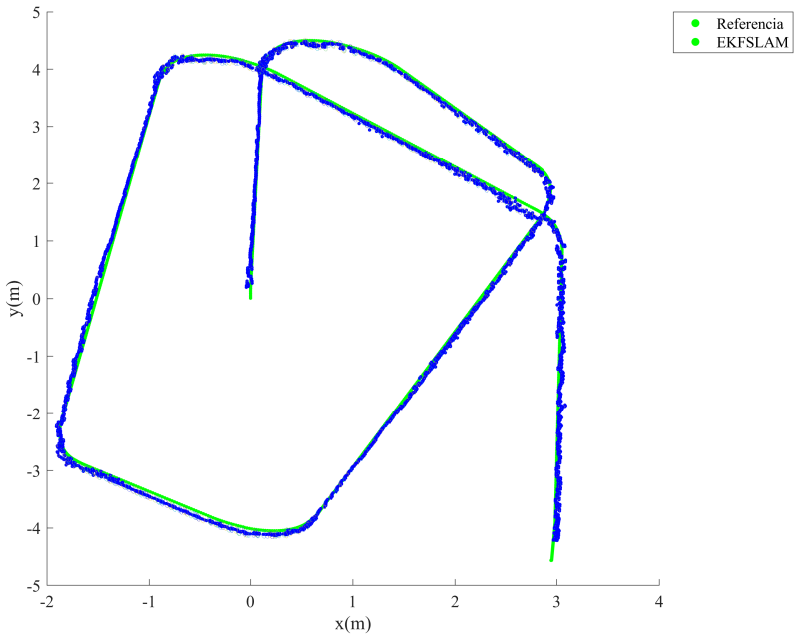



Figura 5.7 – Trajetória gerada pelo EKF–SLAM com *landmarks* com ambos detectores (Simulação).

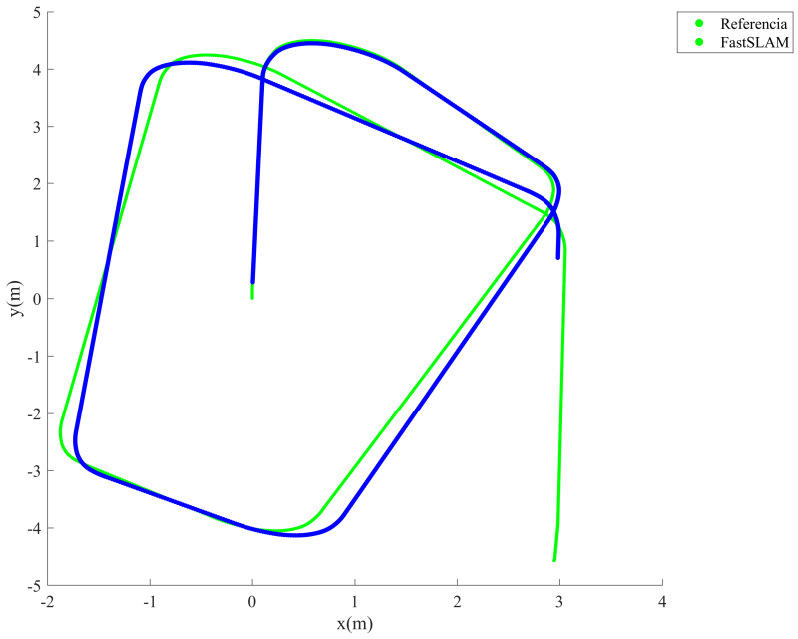



Figura 5.8 – Trajetória gerada pelo FastSLAM com *landmarks* naturais (Simulação).

### 5.1.3 Erro médio quadrático e relação número de cantos e/ou ArUcos

Análise visual é importante, mas é difícil expressar o quão realmente foi boa a estimativa do algoritmo. Desta forma através do erro médio quadrático será mais fácil analisar e comparar dentre os algoritmos implementados. Desta forma, para cada algoritmo implementado foi gerado um gráfico que correlaciona os *landmarks* utilizados e o erro ao longo do tempo. O primeiro a ser apresentado na Figura 5.9, mostrando como foi analisado na Figura 5.5, também é possível perceber que quando o número de *Landmarks* diminui, nos instantes seguintes o erro cresce.

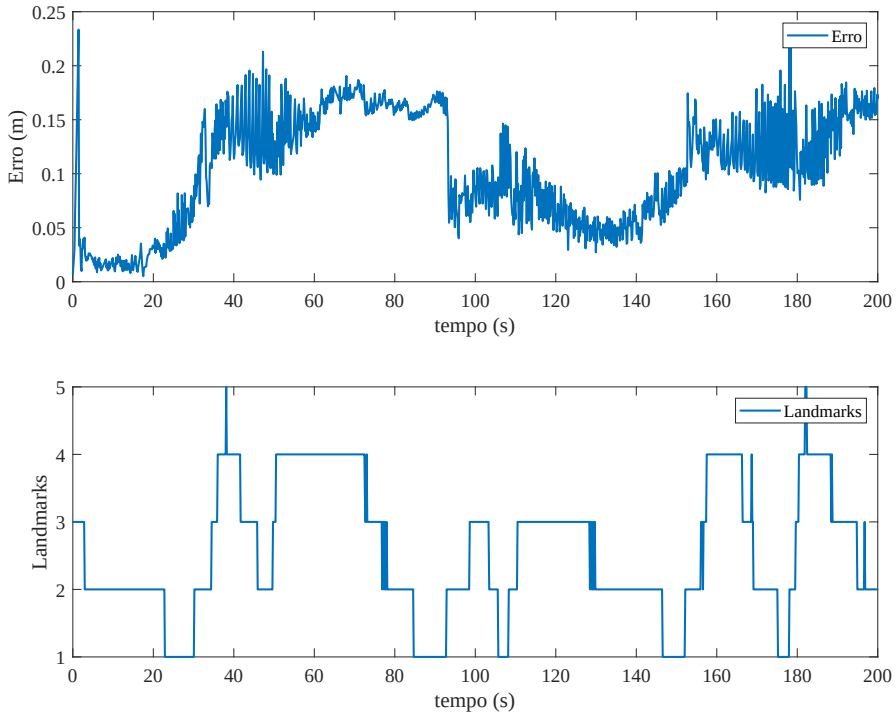



Figura 5.9 – Relação entre o erro do EKF–SLAM ao longo do tempo e e o número de cantos (Simulação).

A Figura 5.10 é relacionada a aplicação da EKF localização apenas com detecção de ArUcos. Diferente da detecção com *lidar*, é possível ver um erro muito mais oscilatório e assumindo valores bem expressivos, chegando a 3m. Ao analisar a trajetória era possível observar que o erro era exacerbado. Mas da mesma forma, que o gráfico anterior, quando o robô percebe marcadores, consegue diminuir o erro.

O resultado da combinação de ambos os *Landmarks* pode ser visto na Figura 5.11 gerando um resultado piorado em relação ao apresentado com o *lidar* somente. Como comentado, isso pode ter uma relação direta por ser em simulação. O ponto interessante a se observar é que o sistema, ao não perceber mais marcadores ArUco, num primeiro momento aumenta o erro drasticamente, até se adaptar novamente.

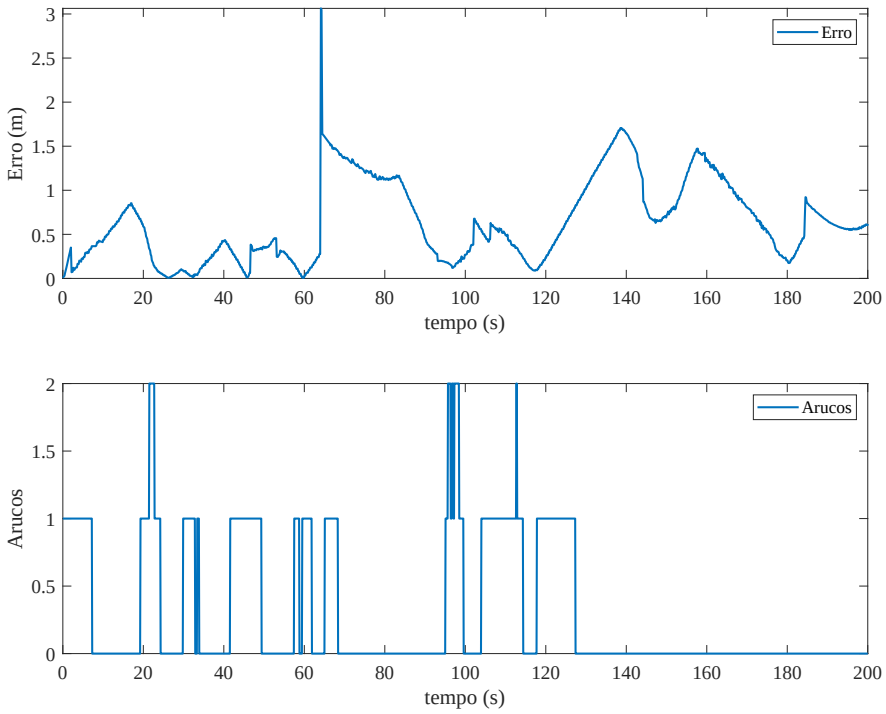



Figura 5.10 – Relação entre o erro do EKF–SLAM ao longo do tempo e o número de ArUcos (Simulação).

Por fim, é apresentado o resultado do FastSLAM na Figura 5.12. Que apenas demonstra o que foi previamente analisado no gráfico da trajetória, por mais que o mapa estivesse bem adaptado, a pose do veículo foi divergindo ao longo do tempo. A relação do número de cantos não consegue apresentar nenhuma explicação do porquê da divergência.

Resultados de simulação são preliminares que indicam a viabilidade de um método na prática, com a finalidade de facilitar quando se é necessário fazer testes. Num primeiro momento o FastSLAM não se mostra utilizável, pois apresenta um erro exponencial e o EKF–SLAM apenas com lidar apresenta o melhor resultado. Na prática muitas premissas podem mudar, por isso agora serão apresentados os resultados do experimento na bancada experimental.

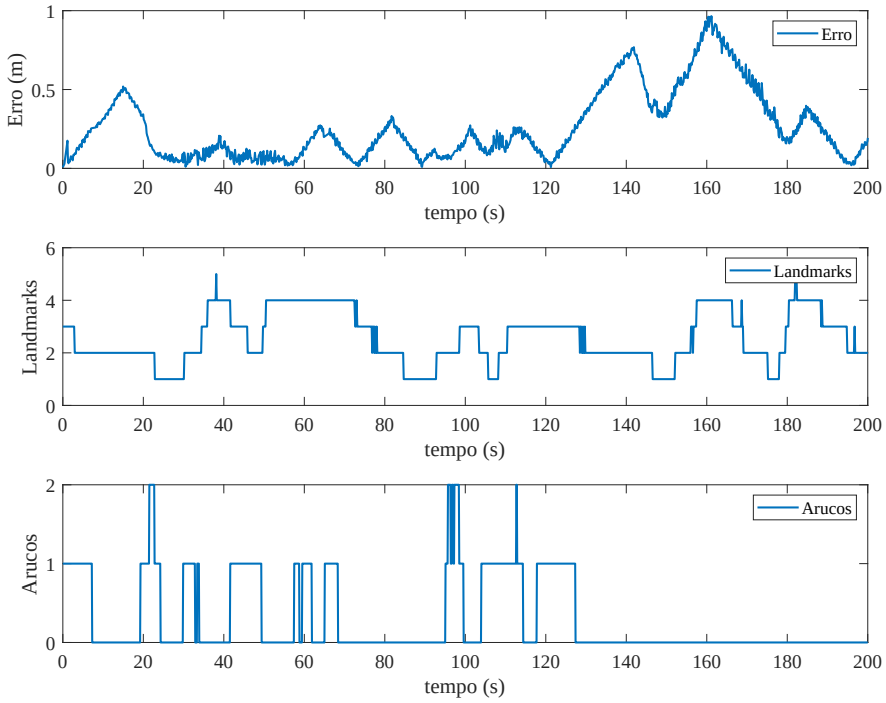



Figura 5.11 – Relação entre o erro do EKF–SLAM ao longo do tempo e o número de ArUcos e cantos (Simulação).

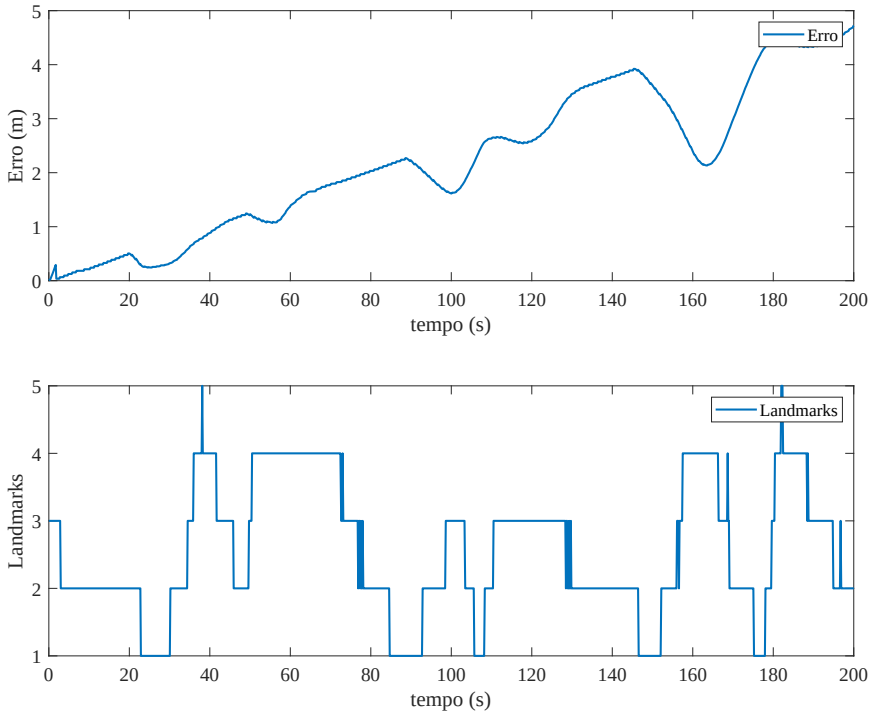



Figura 5.12 – Relação entre o erro do FastSLAM ao longo do tempo e o número cantos (Simulação).

## 5.2 Experimento real

Para a parte prática, foi construído um mapa real previamente apresentado para comparar os mesmos algoritmos que foram utilizados na simulação. De maneira análoga, o robô foi liberado para andar livremente e os dados dos sensores foram adquiridos através do *framework* apresentado. O experimento irá validar tanto os detectores de *features* como também os algoritmos de localização e mapeamento.

### 5.2.1 Mapas gerados

A metodologia utilizada na simulação foi mantida e todos os algoritmos também. A diferença a relação a simulação é que sensores são mais suscetíveis a perturbações e no caso alguns *Landmarks* naturais não são os planejados para o teste, mas o algoritmo tende a se adaptar. O mapa é apresentado na Figura 4.4 e espera–se que os mapas gerados tenham uma similaridade.

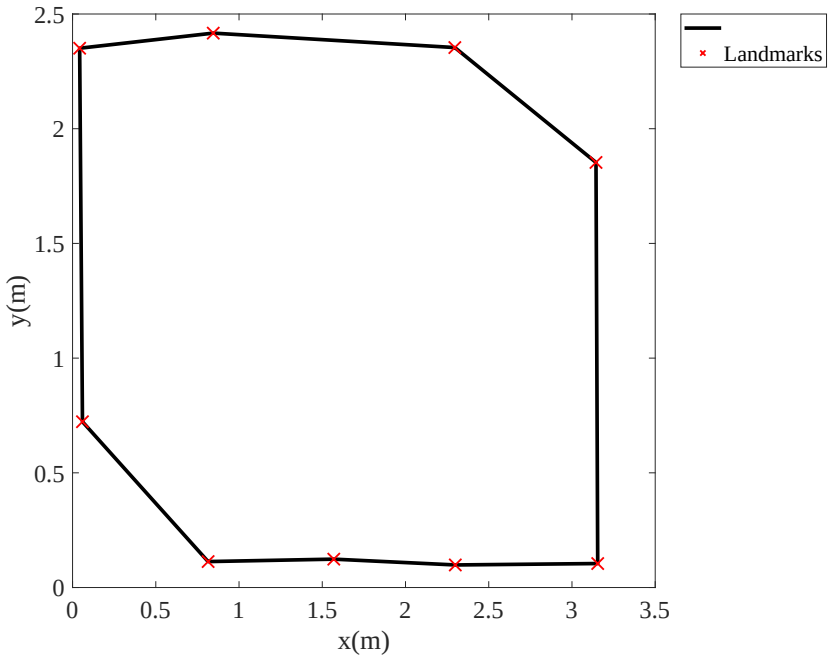



Figura 5.13 – Mapa gerado pelo EKF–SLAM em simulação apenas com *Landmarks* naturais.

A Figura 5.13 apresenta o mapa adquirido via EKF–SLAM utilizando apenas os cantos, o mapa apresenta grande similaridade, mais uma vez o algoritmo nesse tipo de situação apresenta uma boa estimação. Já com o marcadores, Figura 5.14, o mapa adquirido também é bastante similar e também apresenta uma boa representação do mapa. Por fim, o mapa encontrado pelo FastSLAM, Figura 5.15, apresenta algumas similaridades, mas alguns pontos não esperados fazem o mapeamento perder um pouco da qualidade.

Um ponto em comum entre os três resultados, é alguns *Landmarks* no meio das paredes, isso é resultado da conexão entre as paredes e alguns marcadores ArUcos, que tem um breve relevo.

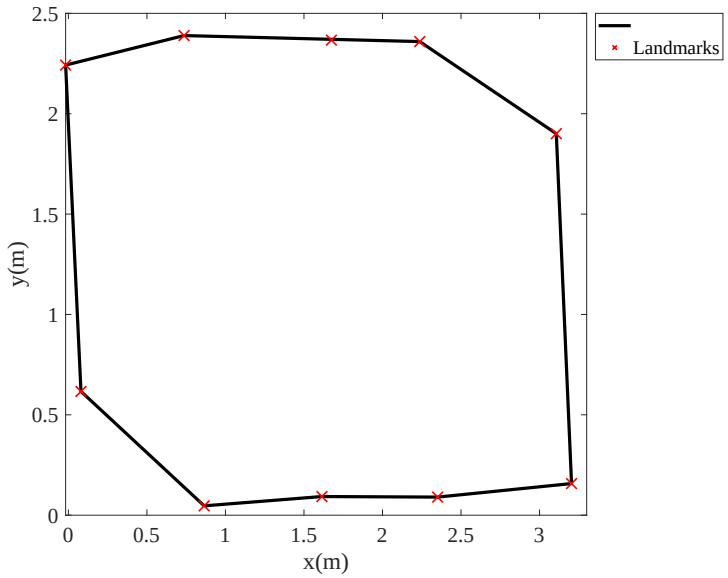



Figura 5.14 – Mapa gerado pelo EKF–SLAM em simulação apenas com adição das marcadores ArUco.

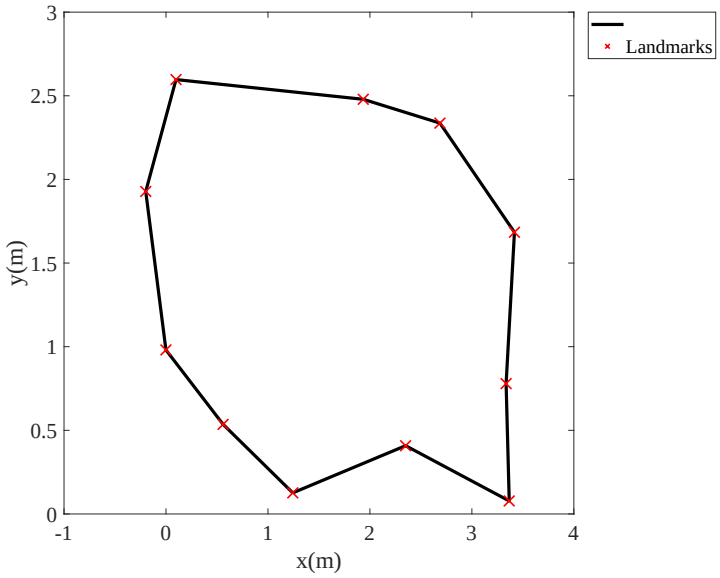



Figura 5.15 – Mapa gerado pelo FastSLAM em simulação com detecção de *Landmarsk* naturais.

### 5.2.2 Trajetórias Realizadas

Para gerar a trajetória de referência, foi captada a informação da câmera e gerado um gráfico de maneira análoga ao apresentado para simulação. A trajetória esperada é apresentada na Figura 5.16 e esses dados serão utilizados para calcular o erro médio quadrático.

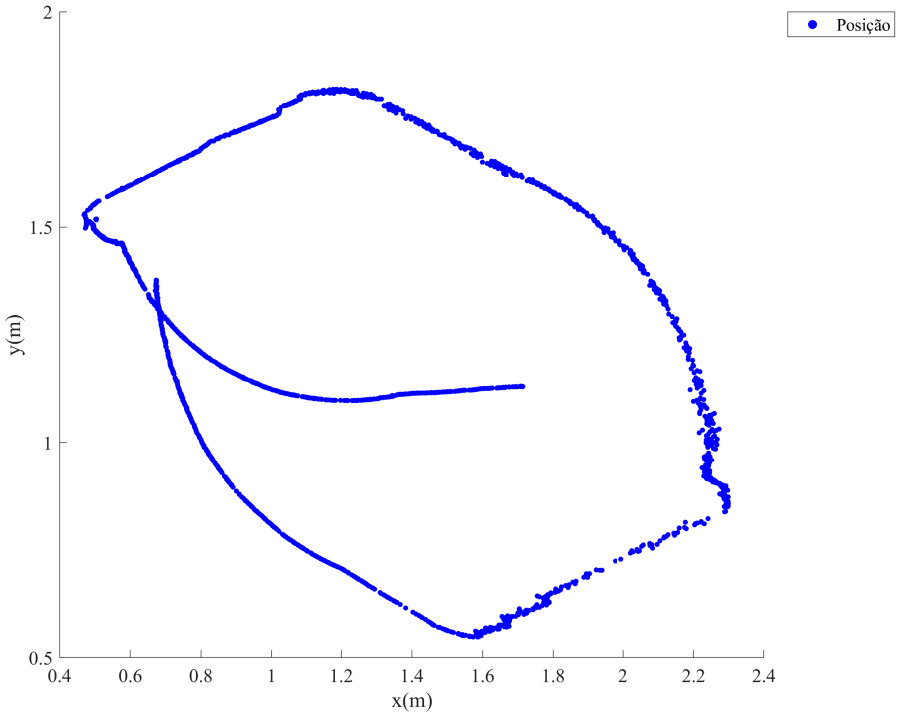



Figura 5.16 – Trajetória referência para o teste experimental.

Na Figura 5.17 é possível observar a trajetória de pontos gerados pelo algoritmo EKF–SLAM utilizando apenas *Landmarks* naturais. É possível perceber uma similaridade em relação ao utilizado como referência. Uma grande diferença que é possível perceber diferente da simulação é a taxa de amostragem e também devido ao fato de ser experimental, as elipses de erros apresentam maiores variações. Após as curvas, as medições se tornam menos precisas, o que é possível ver pela elipse. Ao se movimentar sem uma curva abrupta, a covariância diminui bastante, representando que a estimação é precisa.

Ao utilizar apenas a localização EKF, foi encontrada a Figura 5.18 e foi percebido que de maneira similar ao encontrado via simulação, o algoritmo sozinho, não consegue uma estimação da posição do robô satisfatória. Devido ao mesmo problema anteriormente comentado, devido a pontos onde há falta de detecção de marcadores ArUco, tendo que utilizar somente a hodometria.

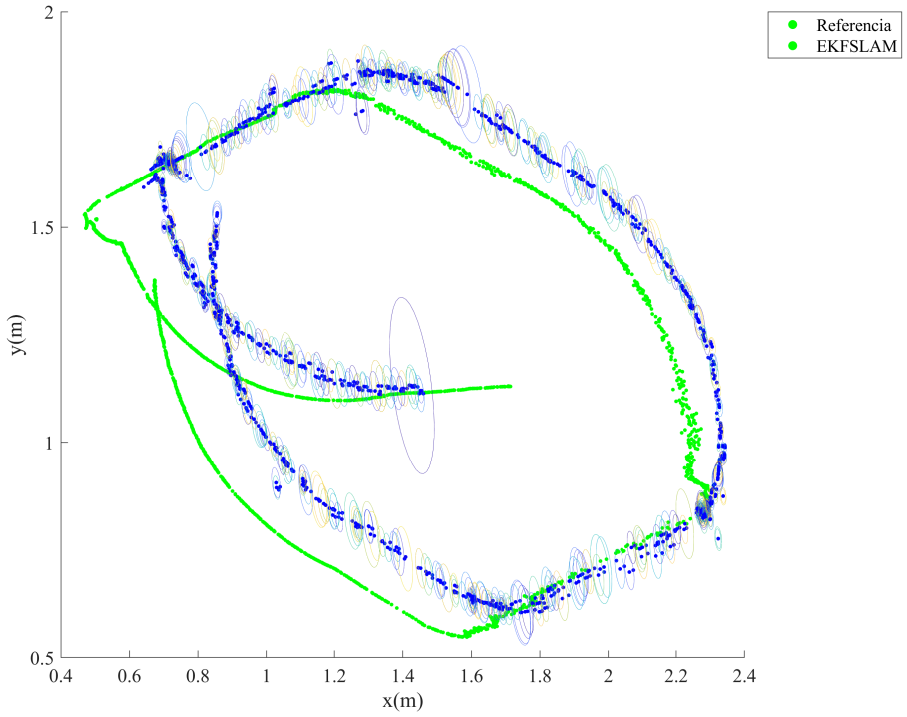



Figura 5.17 – Trajetória gerada pelo EKF–SLAM com *landmarks* naturais.

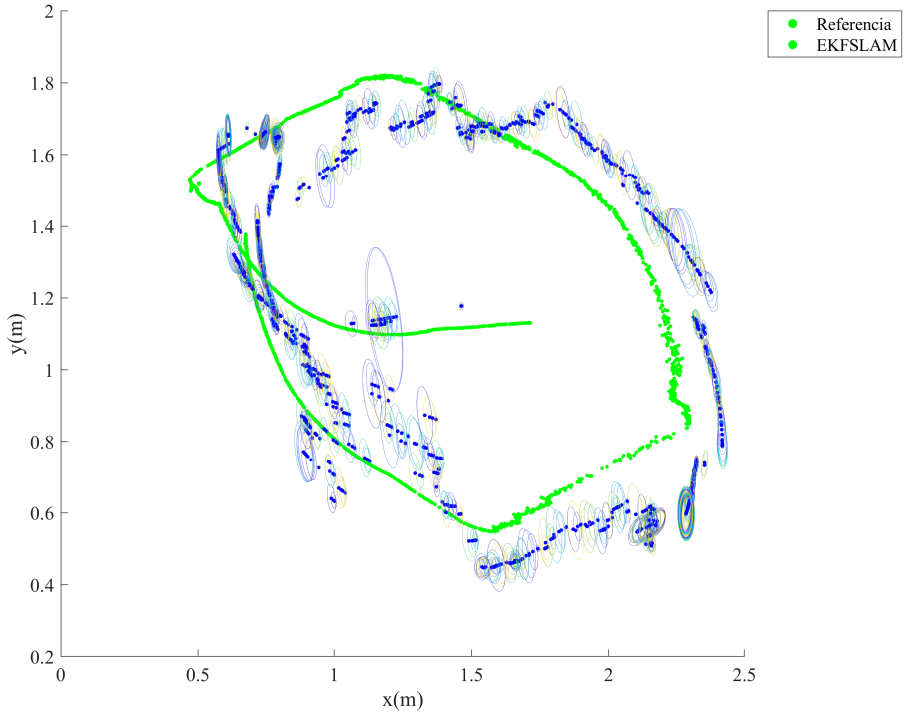



Figura 5.18 – Trajetória gerada pelo EKF–SLAM com *landmarks* artificais.

Então, foram combinados ambos os métodos, adquirindo a Figura 5.19 utilizando somente o *lidar*, mas um ponto importante a se analisar, é a covariância da posição se mantém praticamente constante por toda a trajetória. Até mesmo nas curvas, onde normalmente a incerteza aumentava, a combinação entre os algoritmos conseguiu responder bem.

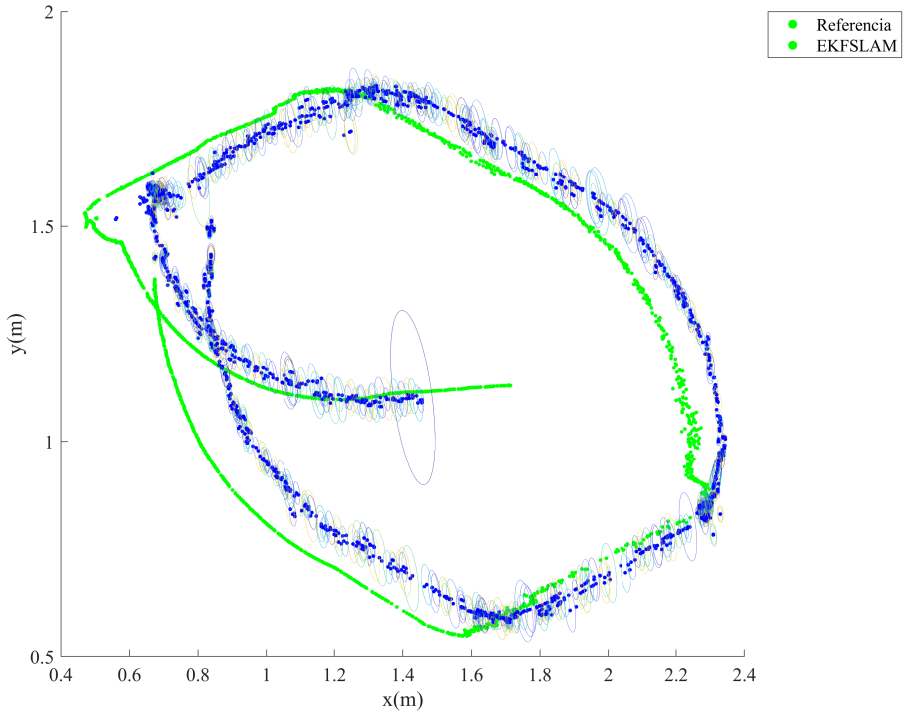



Figura 5.19 – Trajetória gerada pelo EKF–SLAM com *landmarks* com ambos detectores.

Por fim, foi testado o algoritmo do FastSLAM, o que gerou a Figura 5.20, que é possível perceber uma leve semelhança a trajetória referência, pois segue um trajeto similar. Mas é possível perceber que o algoritmo está um pouco atrasado em relação a referência.

### 5.2.3 Erro médio quadrático e relação número de cantos e/ou ArUcos

Utilizando os dados que geraram as trajetórias e comparados com a trajetória referência foram construídos os mesmos gráficos de erro correlacionados com o número de *Landmarks* naturais e/ou artificiais. Começando novamente com o algoritmo EKF–SLAM utilizando somente *lidar*, foi possível encontrar a Figura 5.21, nela é possível ver que o erro a passar um pouco do 0,6m, mas se mantém estável ao longo de toda a trajetória. Também é possível ver a correlação com o número de *Landmarks*, visto que ao aumentar o número, alguns passos seguintes o erro diminui e também que maiores espaços com pouca *feature* gera maiores erros.

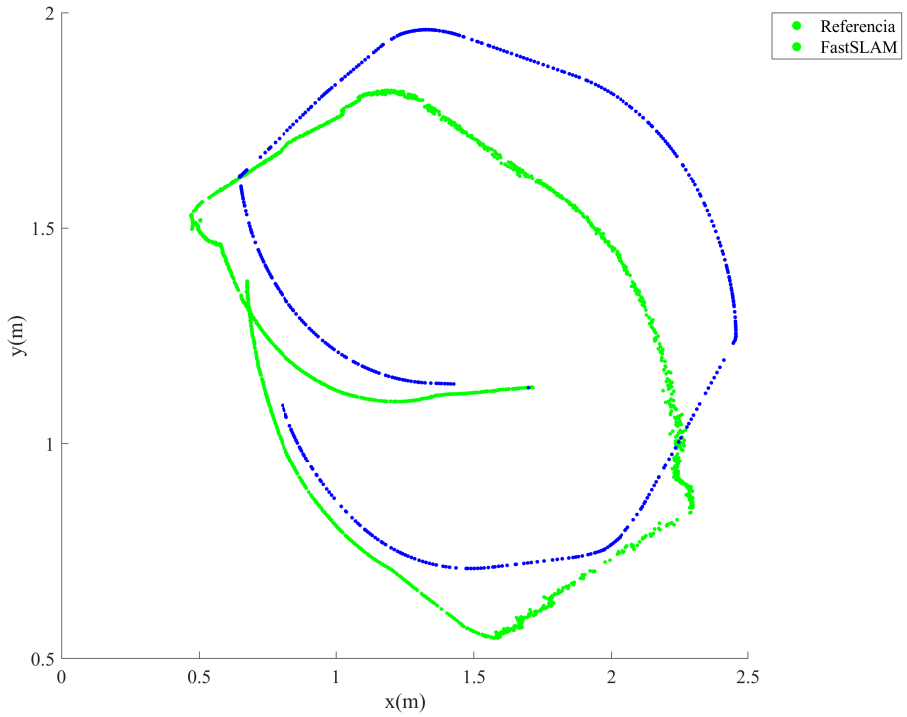



Figura 5.20 – Trajetória gerada pelo FastSLAM com *landmarks* naturais.

Ao verificar a localização por EKF, o resultado apresentado na Figura 5.17 que apresentou um comportamento diferente do que ocorreu na simulação, mais constante. Entretanto, o algoritmo dentre os testados teve o pior rendimento. Quase todo tempo do percurso teve um marcador sendo detectado, o que auxiliou erro a não aumentar.

Ao realizar a combinação de ambos os *Landmarks*, o resultado apresentado na Figura 5.23 foi oposto a simulação. Ou seja, a adição da detecção do ArUco apresentou ganhos, o pico de erro foi menor e o algoritmo apresenta menos variações que anteriormente utilizando apenas o lidar.

Por fim, aplicar o algoritmo do FastSLAM resultou na Figura 5.24 e teve um resultado comparado ao EKF–SLAM na prática bastante diferente do que foi na simulação, não tendo sua estimativa divergindo. O erro apresentado foi, bastante similar ao do EKF–SLAM, entretanto manteve um erro mais alto e constante.

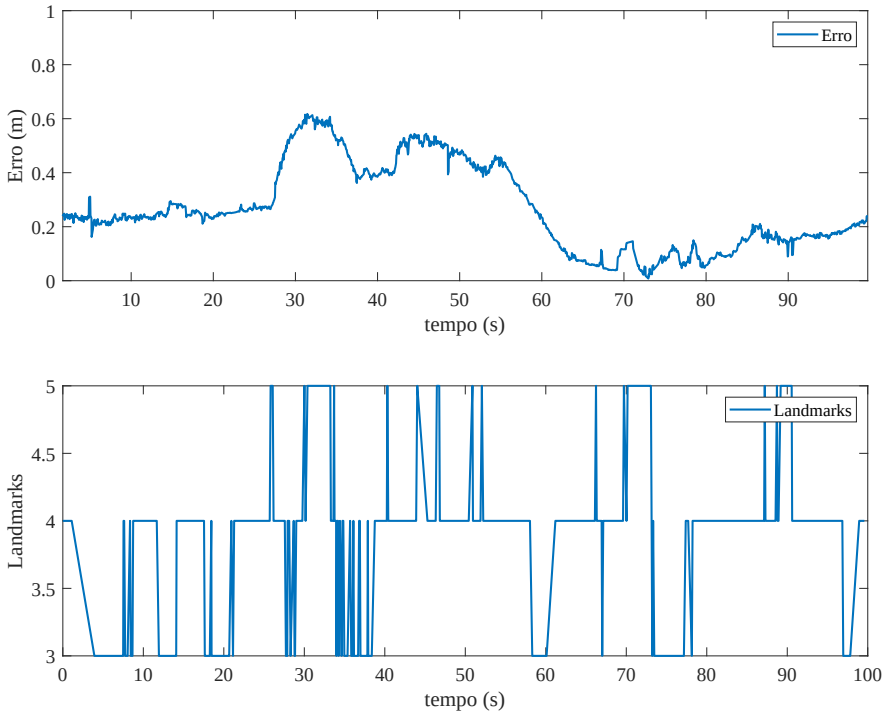



Figura 5.21 – Relação entre o erro do EKF–SLAM ao longo do tempo e o número cantos.

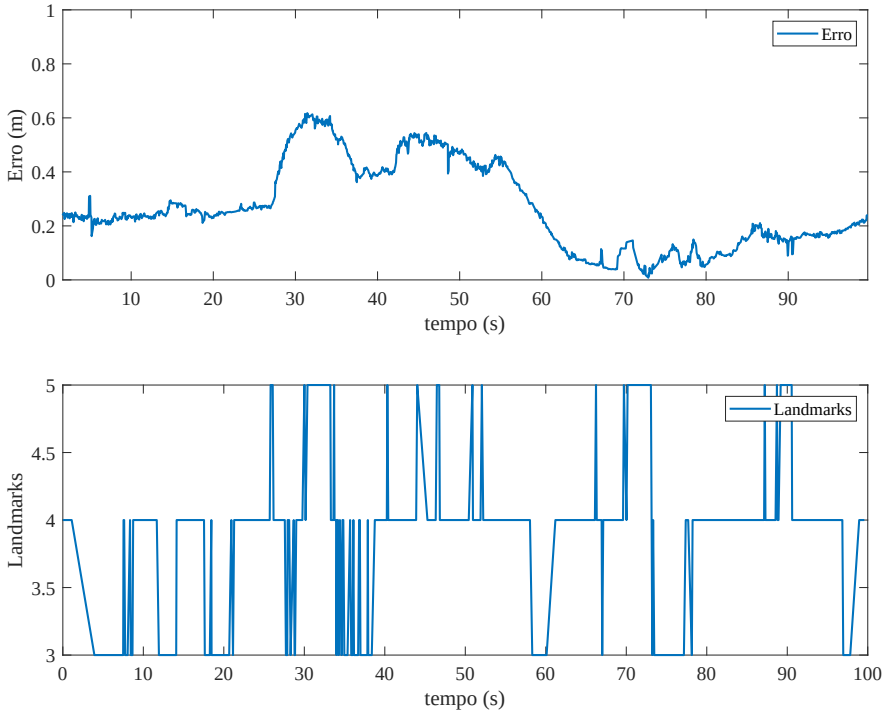



Figura 5.22 – Relação entre o erro da localização por EKF ao longo do tempo e o número de ArUcos.

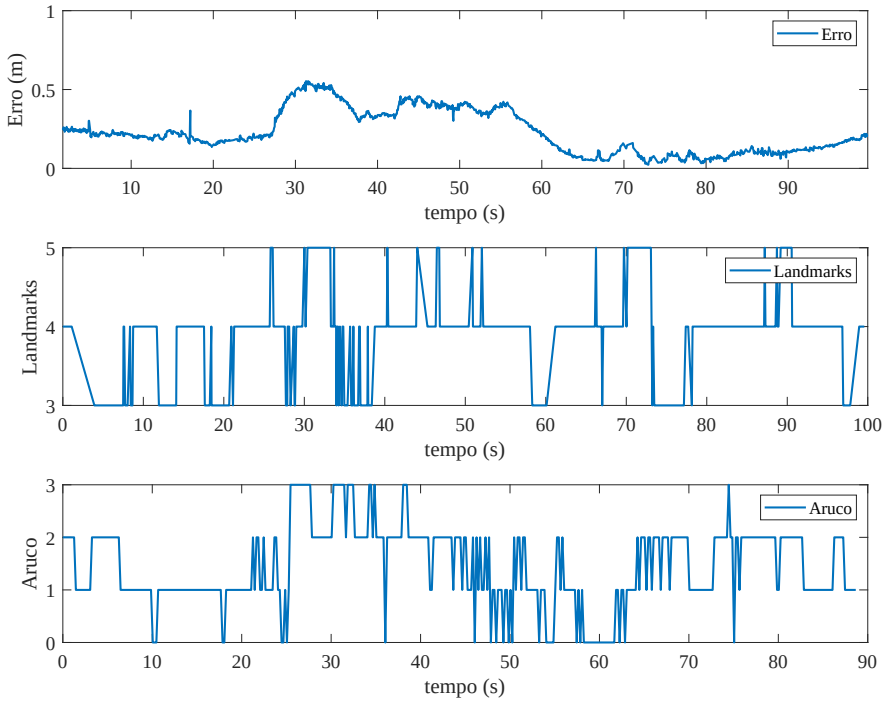



Figura 5.23 – Relação entre o erro do EKF–SLAM ao longo do tempo e o número de ArUc0s e cantos.

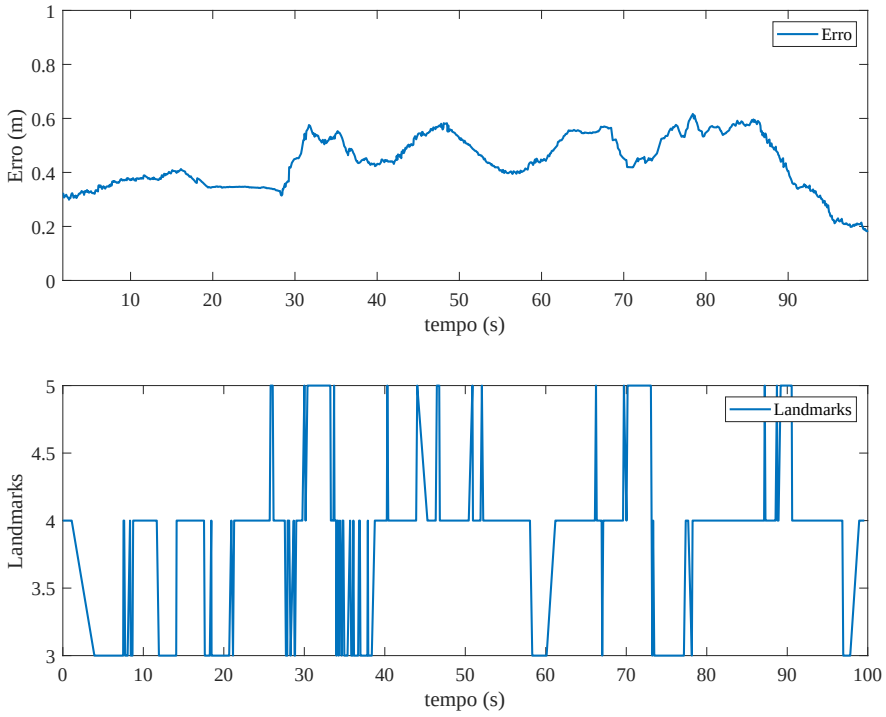



Figura 5.24 – Relação entre o erro do FastSLAM ao longo do tempo e o número cantos.

# Conclusão

Este capítulo finaliza a dissertação e está dividido em duas Seções. Num primeiro momento serão apresentados as considerações finais sobre o trabalho como um todo e por fim, alguns trabalhos futuros que podem ser explorados para avançar ao estudo proposto neste texto.

## 5.3 Considerações Finais

Este trabalho propôs implementar EKF–SLAM e FastSLAM e além disso, explorar tanto o campo de *Landmarks* naturais como também artificiais (marcadores ArUco). Para isso, utilizando tanto a simulação como uma bancada experimental foi possível validar os algoritmos e métodos apresentados. Visto isso, foi possível encontrar vantagens e desvantagens dentre os métodos apresentados. O EKF–SLAM possui uma estimação mais ruidosa quando comparada ao FastSLAM, mas consegue performar a tarefa de localização satisfatoriamente. Já, o FastSLAM, em simulação não teve uma boa performance, mas na prática mostrou melhor resultado. Em relação aos diferentes grandes espaços podem dificultar a aplicação da técnica, tanto pela possível limitação de distância do lidar, como também pelo ArUco ser de difícil detecção. Uma desvantagem do uso de ArUco está relacionado ao seu posicionamento, que adiciona mais incerteza ao processo. Apesar das limitações expressas, as técnicas mostraram uma boa estimação da pose, o que habilita sua utilização. Este trabalho, contribui bastante no estudo da utilização de mais de um sensor para técnicas de localização, algo que vem sendo abordado por grupos de pesquisa.

## 5.4 Trabalhos Futuros

Alguns pontos podem ser trabalhados, tanto para trazer robustez ao sistema, como também versatilidade. Um bom ponto que pode ser estudado é de remover possíveis *outliers* na detecção de marcadores, a aplicação do filtro mahalanobis, é um bom exemplo, pois leva em consideração a covariância da detecção e pode trazer robustez ao sistema. Outro aspecto relevante é na substituição de marcadores artificiais e identificar objetos que possam ser usados de maneira análoga, que normalmente estarão naquele local, deixando o sistema mais versátil, para mais aplicações, não necessitando "marcar"o ambiente, entretanto isso pode trazer um custo computacional indesejado.

# Referências

ALATISE, M. B.; HANCKE, G. P. Pose estimation of a mobile robot based on fusion of imu data and vision data using an extended kalman filter. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 10, p. 2164, 2017. Citado na página 27.

ALHMIEDAT, T. et al. A slam-based localization and navigation system for social robots: The pepper robot case. *Machines*, MDPI, v. 11, n. 2, p. 158, 2023. Citado na página 20.

AMAZON. *Amazon official website*. 2023. Urlhttps://www.amazon.com/. Citado na página 14.

APPLE. *Apple official website*. 2023. Urlhttps://www.apple.com/. Citado na página 14.

ARRAS, K. O.; SIEGWART, R. Y. Feature extraction and scene interpretation for map-based navigation and map building. In: SPIE. *Mobile Robots XII*. [S.l.], 1998. v. 3210, p. 42–53. Citado na página 27.

BAILEY, T. et al. Consistency of the ekf-slam algorithm. In: IEEE. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2006. p. 3562–3568. Citado na página 25.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. In: SPRINGER. *European conference on computer vision*. [S.l.], 2006. p. 404–417. Citado na página 28.

BECKER, M. B. et al. Design of an extended kalman filter for an autonomous sailboat. Florianópolis, SC., 2019. Citado na página 39.

BEHAN, J.; O'KEEFFE, D. T. The development of an autonomous service robot. implementation:"lucas"—the library assistant robot. *Intelligent Service Robotics*, Springer, v. 1, p. 73–89, 2008. Citado na página 18.

BELANCHE, D. et al. Service robot implementation: a theoretical framework and research agenda. *The Service Industries Journal*, Taylor & Francis, v. 40, n. 3-4, p. 203–225, 2020. Citado na página 14.

BORGES, G. A.; ALDON, M.-J. Line extraction in 2d range images for mobile robotics. *Journal of intelligent and Robotic Systems*, Springer, v. 40, n. 3, p. 267–297, 2004. Citado na página 27.

BRASCH, N. et al. Semantic monocular slam for highly dynamic environments. In: IEEE. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2018. p. 393–400. Citado 2 vezes nas páginas 23 e 27.

CAMPOS, C. et al. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, IEEE, v. 37, n. 6, p. 1874–1890, 2021. Citado na página 28.

CHEBOTAREVA, E. et al. Laser rangefinder and monocular camera data fusion for human-following algorithm by pmb-2 mobile robot in simulated gazebo environment. In: SPRINGER. *Proceedings of 15th International Conference on Electromechanics and Robotics"Zavalishin's Readings"*. [S.l.], 2021. p. 357–369. Citado na página 31.

CHILIAN, A.; HIRSCHMÜLLER, H. Stereo camera based navigation of mobile robots on rough terrain. In: IEEE. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. [S.l.], 2009. p. 4571–4576. Citado na página 14.

CHUNG, M.-A.; LIN, C.-W. An improved localization of mobile robotic system based on amcl algorithm. *IEEE Sensors Journal*, IEEE, v. 22, n. 1, p. 900–908, 2021. Citado na página 25.

Chwa, D. Tracking control of differential-drive wheeled mobile robots using a backstepping-like feedback linearization. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, v. 40, n. 6, p. 1285–1295, 2010. Citado na página 30.

DOBREV, Y.; GULDEN, P.; VOSSIEK, M. An indoor positioning system based on wireless range and angle measurements assisted by multi-modal sensor fusion for service robot applications. *IEEE Access*, IEEE, v. 6, p. 69036–69052, 2018. Citado 2 vezes nas páginas 14 e 19.

EDWARDS, M. J.; HAYES, M. P.; GREEN, R. D. High-accuracy fiducial markers for ground truth. In: IEEE. *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. [S.l.], 2016. p. 1–6. Citado na página 48.

EGGER, P. et al. Posemap: Lifelong, multi-environment 3d lidar localization. In: IEEE. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2018. p. 3430–3437. Citado na página 14.

ENGEL, J.; KOLTUN, V.; CREMERS, D. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 40, n. 3, p. 611–625, 2017. Citado na página 27.

FILIPENKO, M.; AFANASYEV, I. Comparison of various slam systems for mobile robot in an indoor environment. In: IEEE. *2018 International Conference on Intelligent Systems (IS)*. [S.l.], 2018. p. 400–407. Citado 2 vezes nas páginas 14 e 22.

FORSYTH, D. A.; PONCE, J. *Computer vision: a modern approach*. [S.l.]: prentice hall professional technical reference, 2002. Citado na página 27.

GAILIS, J. *Towards 6D SLAM with ground truth integration*. 2015. Citado na página 24.

GAO, X. et al. Ldso: Direct sparse odometry with loop closure. In: IEEE. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2018. p. 2198–2204. Citado na página 27.

GARRIDO-JURADO, S. et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, Elsevier, v. 47, n. 6, p. 2280–2292, 2014. Citado na página 34.

GEE, A. P.; MAYOL-CUEVAS, W. Real-time model-based slam using line segments. In: SPRINGER. *International Symposium on Visual Computing*. [S.l.], 2006. p. 354–363. Citado 2 vezes nas páginas 25 e 28.

GIUBILATO, R.; PERTILE, M.; DEBEI, S. A comparison of monocular and stereo visual fastslam implementations. In: IEEE. *2016 IEEE Metrology for Aerospace (MetroAeroSpace)*. [S.l.], 2016. p. 227–232. Citado na página 25.

GÓMEZ, C. et al. Visual slam-based localization and navigation for service robots: The pepper case. In: SPRINGER. *RoboCup 2018: Robot World Cup XXII 22*. [S.l.], 2019. p. 32–44. Citado na página 20.

GONZALEZ-AGUIRRE, J. A. et al. Service robots: Trends and technology. *Applied Sciences*, MDPI, v. 11, n. 22, p. 10702, 2021. Citado 2 vezes nas páginas 8 e 17.

GRAETER, J.; WILCZYNSKI, A.; LAUER, M. Limo: Lidar-monocular visual odometry. In: IEEE. *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. [S.l.], 2018. p. 7872–7879. Citado na página 27.

GREWAL, M. S.; ANDREWS, A. P. *Kalman filtering: Theory and Practice with MATLAB*. [S.l.]: John Wiley & Sons, 2014. Citado na página 40.

GROOT, R. et al. *Autonomous Exploration and Navigation with the Pepper robot*. Dissertação (Mestrado), 2018. Citado na página 20.

HARTLEY, R.; ZISSERMAN, A. *Multiple view geometry in computer vision*. [S.l.]: Cambridge university press, 2003. Citado na página 27.

HARTMANN, G.; HUANG, F.; KLETTE, R. Landmark initialization for unscented kalman filter sensor fusion in monocular camera localization. *International Journal of Fuzzy Logic and Intelligent Systems*, Korean Institute of Intelligent Systems, v. 13, n. 1, p. 1–11, 2013. Citado na página 27.

IFR. *International Federation of Robotics*. 2023. Urlhttps://ifr.org/. Citado 2 vezes nas páginas 14 e 17.

JONG, M. D. et al. Towards a robust interactive and learning social robot. In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. [S.l.: s.n.], 2018. p. 883–891. Citado na página 20.

JÚNIOR, A. d. N. *Robotização de uma cadeira de rodas motorizada: arquitetura, modelos, controle e aplicações*. [S.l.]: Campinas, 2016. Citado na página 56.

JÚNIOR, A. de O. et al. Improving the mobile robots indoor localization system by combining slam with fiducial markers. In: IEEE. *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE)*. [S.l.], 2021. p. 234–239. Citado na página 26.

KALAITZAKIS, M. et al. Fiducial markers for pose estimation: Overview, applications and experimental comparison of the artag, apriltag, aruco and stag markers. *Journal of Intelligent & Robotic Systems*, Springer, v. 101, p. 1–26, 2021. Citado 2 vezes nas páginas 28 e 29.

KALMAN, R. E. A new approach to linear filtering and prediction problems. 1960. Citado na página 37.

KHALIQ, A. A.; SAFFIOTTI, A. Stigmergy at work: Planning and navigation for a service robot on an rfid floor. In: IEEE. *2015 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2015. p. 1085–1092. Citado na página 19.

KHURANA, A.; NAGLA, K. Extrinsic calibration methods for laser range finder and camera: A systematic review. *MAPAN*, Springer, v. 36, n. 3, p. 669–690, 2021. Citado na página 32.

KOHLBRECHER, S. et al. Hector open source modules for autonomous mapping and navigation with rescue robots. In: SPRINGER. *Robot Soccer World Cup*. [S.l.], 2013. p. 624–631. Citado na página 26.

LEE, I. Service robots: a systematic literature review. *Electronics*, MDPI, v. 10, n. 21, p. 2658, 2021. Citado na página 17.

LEE, T.-j.; KIM, C.-h.; CHO, D.-i. D. A monocular vision sensor-based efficient slam method for indoor service robots. *IEEE Transactions on Industrial Electronics*, IEEE, v. 66, n. 1, p. 318–328, 2018. Citado 2 vezes nas páginas 14 e 20.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, Springer, v. 60, n. 2, p. 91–110, 2004. Citado na página 28.

LUO, R. C.; LAI, C. C. Enriched indoor map construction based on multisensor fusion approach for intelligent service robot. *IEEE Transactions on Industrial Electronics*, IEEE, v. 59, n. 8, p. 3135–3145, 2011. Citado 2 vezes nas páginas 8 e 18.

LV, J. et al. Straight line segments extraction and ekf-slam in indoor environment. *Journal of Automation and control Engineering*, v. 2, n. 3, 2014. Citado na página 25.

MERZLYAKOV, A.; MACENSKI, S. A comparison of modern general-purpose visual slam approaches. In: IEEE. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2021. p. 9190–9197. Citado na página 27.

MIŠEIKIS, J. et al. Lio-a personal robot assistant for human-robot interaction and care applications. *IEEE Robotics and Automation Letters*, IEEE, v. 5, n. 4, p. 5339–5346, 2020. Citado 3 vezes nas páginas 8, 14 e 19.

MOBILEROBOTS adept. *Pioneer3 - DX*. 2011. Disponível em: <https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf>. Citado na página 30.

MONTEMERLO, M. et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, v. 593598, 2002. Citado 2 vezes nas páginas 25 e 44.

MONTEMERLO, M. et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: *IJCAI*. [S.l.: s.n.], 2003. v. 3, p. 1151–1156. Citado na página 44.

MOUNTNEY, P. et al. Simultaneous stereoscope localization and soft-tissue mapping for minimal invasive surgery. In: SPRINGER. *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2006: 9th International Conference, Copenhagen, Denmark, October 1-6, 2006. Proceedings, Part I 9*. [S.l.], 2006. p. 347–354. Citado na página 24.

MUNOZ-SALINAS, R.; MARÍN-JIMENEZ, M. J.; MEDINA-CARNICER, R. Spm-slam: Simultaneous localization and mapping with squared planar markers. *Pattern Recognition*, Elsevier, v. 86, p. 156–171, 2019. Citado na página 26.

MUÑOZ-SALINAS, R.; MEDINA-CARNICER, R. Ucoslam: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *Pattern Recognition*, Elsevier, v. 101, p. 107193, 2020. Citado 3 vezes nas páginas 15, 26 e 29.

MUR-ARTAL, R.; MONTIEL, J. M. M.; TARDOS, J. D. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, IEEE, v. 31, n. 5, p. 1147–1163, 2015. Citado na página 26.

NEUNERT, M.; BLOESCH, M.; BUCHLI, J. An open source, fiducial based, visual-inertial motion capture system. In: IEEE. *2016 19th International Conference on Information Fusion (FUSION)*. [S.l.], 2016. p. 1523–1530. Citado 2 vezes nas páginas 26 e 29.

NGUYEN, V. et al. A comparison of line extraction algorithms using 2d range data for indoor mobile robotics. *Autonomous Robots*, Springer, v. 23, n. 2, p. 97–111, 2007. Citado na página 27.

OLIVEIRA, W. dos S.; GONÇALVES, E. N. Implementação em c: filtro de kalman, fusão de sensores para determinação de ângulos. *ForScience*, v. 5, n. 3, 2017. Citado na página 39.

PAGES, J.; MARCHIONNI, L.; FERRO, F. Tiago: the modular robot that adapts to different research needs. In: *International workshop on robot modularity, IROS*. [S.l.: s.n.], 2016. v. 290. Citado na página 21.

PORTUGAL, D. et al. A study on the deployment of a service robot in an elderly care center. *International Journal of Social Robotics*, Springer, v. 11, n. 2, p. 317–341, 2019. Citado na página 14.

PROCOPIO, A. Simulazione ed implementazione di un algoritmo di slam per la guida autonoma su circuito di un veicolo small-scale. 2020. Citado na página 25.

PYO, Y. et al. Service robot system with an informationally structured environment. *Robotics and Autonomous Systems*, Elsevier, v. 74, p. 148–165, 2015. Citado na página 19.

RAMALINGAM, B. et al. A human support robot for the cleaning and maintenance of door handles using a deep-learning framework. *Sensors*, MDPI, v. 20, n. 12, p. 3543, 2020. Citado na página 20.

ROHMER, E.; SINGH, S. P.; FREESE, M. V-rep: A versatile and scalable robot simulation framework. In: IEEE. *2013 IEEE/RSJ international conference on intelligent robots and systems*. [S.l.], 2013. p. 1321–1326. Citado na página 52.

RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. In: IEEE. *2011 International conference on computer vision*. [S.l.], 2011. p. 2564–2571. Citado na página 28.

SHI, X. et al. Are we ready for service robots? the openloris-scene datasets for lifelong slam. In: IEEE. *2020 IEEE international conference on robotics and automation (ICRA)*. [S.l.], 2020. p. 3139–3145. Citado na página 20.

SMITH, P.; REID, I.; DAVISON, A. J. Real-time monocular slam with straight lines. In: *BMVC*. [S.l.: s.n.], 2006. v. 6, p. 17–26. Citado 2 vezes nas páginas 25 e 28.

STACHNISS, C.; GRISETTI, G. Gmapping project at openslam. org. 2007. Citado 2 vezes nas páginas 25 e 26.

STUEDE, M. et al. Sobi: An interactive social service robot for long-term autonomy in open environments. In: IEEE. *2021 European Conference on Mobile Robots (ECMR)*. [S.l.], 2021. p. 1–8. Citado na página 31.

SUN, F. et al. Research on active slam with fusion of monocular vision and laser range data. In: IEEE. *2010 8th World congress on intelligent control and automation*. [S.l.], 2010. p. 6550–6554. Citado 2 vezes nas páginas 25 e 28.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. [S.l.]: MIT press Cambridge, 2000. v. 1. Citado na página 40.

THRUN, S. et al. *Probabilistic robotics, vol. 1*. [S.l.]: MIT press Cambridge, 2005. Citado 5 vezes nas páginas 25, 35, 36, 40 e 41.

VALLIVAARA, I. et al. Magnetic field-based slam method for solving the localization problem in mobile robot floor-cleaning task. In: IEEE. *2011 15th international conference on advanced robotics (ICAR)*. [S.l.], 2011. p. 198–203. Citado na página 18.

VáVRA, P. *ROS FRAMEWORK UTILIZATION FOR AUTONOMOUS MOBILE ROBOT CONTROL SYSTEM*. Dissertação (Mestrado) — Brno University of Technology, Czech Republic, 2019. Citado na página 48.

WELCH, G.; BISHOP, G. et al. *An introduction to the Kalman filter*. [S.l.]: Citeseer, 1995. Citado na página 40.

WIRTZ, J. et al. Brave new world: service robots in the frontline. *Journal of Service Management*, Emerald Publishing Limited, v. 29, n. 5, p. 907–931, 2018. Citado na página 17.

XU, X. et al. E-sbot: A soft service robot for user-centric smart service delivery. In: IEEE. *2019 IEEE World Congress on Services (SERVICES)*. [S.l.], 2019. v. 2642, p. 354–355. Citado na página 14.

XU, Y.; OU, Y.; XU, T. Slam of robot based on the fusion of vision and lidar. In: IEEE. *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. [S.l.], 2018. p. 121–126. Citado na página 27.

XU, Z.; RONG, Z.; WU, Y. A survey: which features are required for dynamic visual simultaneous localization and mapping? *Visual Computing for Industry, Biomedicine, and Art*, SpringerOpen, v. 4, n. 1, p. 1–16, 2021. Citado 2 vezes nas páginas 23 e 27.

ZHANG, G.; SUH, I. H. Building a partial 3d line-based map using a monocular slam. In: IEEE. *2011 IEEE International Conference on Robotics and Automation*. [S.l.], 2011. p. 1497–1502. Citado na página 28.

ZHANG, J.; SINGH, S. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In: IEEE. *2015 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2015. p. 2174–2181. Citado na página 27.

ZHOU, D.; DAI, Y.; LI, H. Ground-plane-based absolute scale estimation for monocular visual odometry. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 21, n. 2, p. 791–802, 2019. Citado na página 27.