Trabalho de Conclusão de Curso

SLAM com associação de dados conhecidos: Uma aplicação usando EKF e mapa com grade de ocupação

Alan Pereira da Silva
aps@ic.ufal.br

Orientador:

Prof. Dr. Leonardo Viana Pereira

Maceió, 21 de Setembro de 2021

Alan Pereira da Silva

SLAM com associação de dados conhecidos: Uma aplicação usando EKF e mapa com grade de ocupação

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Prof. Dr. Leonardo Viana Pereira

Maceió, 21 de Setembro de 2021

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Lívia Silva dos Santos – CRB-4 – 1670

S586s Silva, Alan Pereira da.
SLAM com associação de dados conhecidos: uma aplicação usando EKF e mapa com grade de ocupação / Alan Pereira da Silva. – 2021.
48 f.:il.

Orientador: Carmem. Leonardo Viana Pereira.
Monografia (Trabalho de Conclusão de Curso em Engenharia da Computação) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2021. Maceió, 2021.

Bibliografia: f. 46-48

1. Slam. 2. Python (linguagem de programação). 3. Visão computacional. 4. Extended Kalman Filter (EKF). 5. |Robótica móvel. I. Título.

CDU: 004.421.021

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Instituto de Computação – IC
Campus A. C. Simões - Av. Lourival de Melo Mota, BL 12
Tabuleiro do Martins, Maceió/AL - CEP: 57.072-970
Telefone: (082) 3214-1401

Trabalho de Conclusão de Curso - TCC

Formulário de Avaliação

Nome do Aluno

A	l	a	n		P	e	r	e	i	r	a		d	a		S	i	l	v
a																			

Nº de Matrícula

1	4	2	1	1	2	4	2					-	

Título do TCC (Tema)

SLAM com associação de dados conhecidos: Uma aplicação usando EKF e mapa com grade de ocupação

Banca Examinadora

Prof. Dr. Leonardo Viana Pereira
Orientador
Assinatura

Prof. Dr. André Luiz Lins de Aquino
Avaliador
Assinatura

Dr. Heitor Judiss Savino
Avaliador
gov.br Documento assinado digitalmente
Heitor Judiss Savino
Data: 21/09/2021 17:11:17-0300
Verifique em https://verificador.iti.br
Assinatura

Data da Defesa

21/ 09 / 2021

Nota Obtida

9,5 (nove e meio)

Observações

Coordenador do Curso
De Acordo

Assinatura

Agradecimentos

Primeiramente gostaria de agradecer a Deus por ter me dado saúde e resiliência para superar todos os desafios. A esta universidade, seu corpo docente, direção e administração que demonstraram estar comprometidos com a qualidade e excelência no ensino. Ao meu orientador Prof. Dr. Leonardo Viana, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos. Aos meus pais, pelo amor, incentivo e apoio incondicional. E a todos que direta ou indiretamente fizeram parte de minha formação, o meu muito obrigado.

Dedico esse trabalho ao meu pai, que de maneira tão repentina veio a falecer poucos meses antes de poder me ver realizar esse sonho.

– Ivan da Silva, *In memoriam*

Resumo

Localização e mapeamento simultâneos (SLAM) tem sido um tópico de muita pesquisa nas últimas duas décadas nas comunidades de visão computacional e robótica, e recentemente recebeu a atenção de empresas de alta tecnologia. As técnicas de SLAM criam um mapa de um ambiente desconhecido e localizam o robô nesse mapa, com um forte foco na operação em tempo real. Entre as diferentes modalidades de sensores, as câmeras são baratas e fornecem informações ricas do ambiente que permitem um reconhecimento local robusto e preciso. Portanto, as soluções visuais SLAM, nas quais o sensor principal é uma câmera, são de grande interesse. Este trabalho apresenta uma implementação de localização e mapeamento simultâneo (SLAM) baseado no filtro de Kalman estendido (EKF). Para a implementação foi utilizada a linguagem de programação *python*, o conjunto de bibliotecas e ferramentas do *robot operating system* (ROS) e a biblioteca de visão computacional OpenCV. O trabalho foi desenvolvido no simulador de código aberto Gazebo, onde criamos um ambiente para o robô, simulamos o robô nesse ambiente, e então construímos um mapa desse ambiente. O resultado foi comparado com o mapa original e considerado relativamente preciso. O mapa resultante mostra algumas imprecisões, mas a forma geral é aceitável.

Palavras-chave: SLAM, EKF, ROS.

Abstract

Simultaneous location and mapping (SLAM) has been a topic of much research over the past two decades in the computer vision and robotics communities, and has recently received the attention of high-tech companies. SLAM techniques create a map of an unknown environment and locate the robot on that map, with a strong focus on real-time operation. Among the different sensor modalities, cameras are inexpensive and provide rich information of the environment that allows for robust and accurate local recognition. Therefore, visual SLAM solutions, in which the main sensor is a camera, are of great interest. This work presents an implementation of simultaneous location and mapping (SLAM) based on the extended Kalman filter (EKF). For the implementation, the programming language *python*, the set of libraries and tools of the *robot operating system* (ROS) and the computer vision library OpenCV were used. The work was developed in the open source simulator Gazebo, where we created an environment for the robot, simulated the robot in that environment, and then built a map of that environment. The result was compared with the original map and found to be relatively accurate. The resulting map shows some inaccuracies, but the overall shape is acceptable.

Keywords: SLAM, EKF, ROS.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

IC	Instituto de Computação.
SLAM	Simultaneous Localization and Mapping
KF	Kalman Filter
EKF	Extended Kalman Filter
AR	Augmented reality
ROS	Robot Operating System

Lista de Símbolos

μ	Média
Σ	Covariância
x_t	Vetor transposto representando a posição do robô no tempo t. $[x, y, \theta]^T$
u_t	Vetor transposto representando o controle do robô no tempo t. $[v, \omega]^T$
z_t^k	k-ésima medição do sensor no tempo t
z_t	vetor com todas as medições $\{z_t^1, z_t^2, ..., z_t^k\}$
l_n	Posição do n-ésimo *landmark*
L	Lista de todas as n posições dos *landmarks* $\{l_1, l_2, ..., l_n\}$
m	Mapa, matriz representando os espaços livres e ocupados.
c_t	Associação de dados de observação no tempo t
c^t	Lista com todas as associações de dados $\{c_1, c_2, ..., c_t\}$

Sumário

1

Introdução

1.1 Motivação

Nos últimos anos, a indústria automotiva investiu muito do seu tempo e recursos pesquisando a possibilidade de produzir carros autônomos. A pesquisa avançou a tal ponto que atualmente existem carros semiautônomos (nível 3) disponíveis no mercado, e acredita-se que, em algum momento, quase todos os veículos serão autônomos (nível 5). Para conseguir isso, os carros deverão ter sensores mais precisos que se comuniquem instantaneamente, e algoritmos avançados que sempre tomem decisões assertivas. Além disso, é necessário também construir mapas com alta qualidade e precisão que estejam atualizados com a dinamicidade do ambiente, e o veículo precisa estar ciente de sua posição atual em relação a este ambiente. Isso é conhecido como o problema de localização e mapeamento simultâneos, comumente abreviado de SLAM.

O SLAM têm recebido cada vez mais atenção nos últimos anos, tendo se tornado um dos assuntos mais importantes no campo da robótica móvel no século XXI [17]. O SLAM é formado por duas abordagens principais: SLAM por Filtragem e Visual SLAM (vSLAM). A abordagem de SLAM por filtragem, foi inicialmente proposta por Smith [28] em 1986. Essa abordagem modela o ambiente usando Filtros de Kalman Estendidos, e estima a posição relativa entre o robô e os *landmarks*, filtrando o ruído gaussiano nos dados de observação. Diversos algoritmo desenvolvidos ao longo dos anos é baseada no filtro de kalman. Já a abordagem de SLAM usando câmeras, é conhecida como Visual SLAM, porque é baseada apenas em informações visuais. Algoritmos de vSLAM foram amplamente propostos no campo da visão computacional, robótica e AR [2]. Eles são adequados para estimativa da posição de câmera em sistemas AR porque a configuração dos sistemas pode ser feita de maneira simples em tablets ou smartphones acoplados com câmeras. Comparado com o SLAM baseado em filtragem, o vSLAM tem vantagens em robustez, consistência e precisão. No entanto, quando o robô está em um ambiente altamente incerto, o SLAM baseado em filtragem pode ser melhor do que o vSLAM. Assim, para

aproveitar ao máximo as vantagens de ambos métodos, diversas aplicações têm usado uma combinação das duas abordagens [17].

1.2 Trabalhos Relacionados

Robot Operating System(ROS)[22] foi desenvolvido como um *middleware* da robótica para dar suporte ao projeto Stanford AI Robot STAIR em 2007. Ele fornece muita conveniência para projetos de cluster de computador heterogêneo, como abstração de hardware, troca de mensagens entre processos, controle de dispositivos de baixo nível e gerenciamento de pacotes. Com a forma de comunicação usando nós, os submódulos do algoritmo de navegação podem ser convenientemente integrados em um único sistema. Diversas implementações de SLAM tem sido desenvolvidas ao longo do tempo, boa parte dessas implementações são pacotes de códigos aberto e compatíveis com o ROS. Nesta seção iremos relatar alguns métodos de implementação de SLAM mais utilizados pela comunidade ROS.

1.2.1 HectorSLAM

Este pacote constrói mapas de grade de ocupação 2D e pode funcionar com ou sem informações de odometria [15]. O Algoritmo combina uma abordagem de escaneamento robusto usando o sistema LIDAR com um sistema de estimativa de atitude 3D baseado em sensoriamento inercial [16]. Os recursos de localização e mapeamento são confiáveis devido ao uso de uma aproximação rápida de gradientes de mapa e uma grade de resolução múltipla. De acordo com os autores, o HectorSLAM requer poucos recursos computacionais para funcionar.

1.2.2 EKF SLAM

Mobile Robot Programming Toolkit (MRPT) [4] é um conjunto de ferramentas de código aberto que fornece muitos aplicativos e bibliotecas portáteis e bem testadas, abrangendo estruturas de dados e algoritmos empregados nas áreas de robótica móvel. O MRPT torna possível adicionar funções em um sistema de navegação de robô móvel rapidamente. Dentre esse conjunto de ferramentas, podemos encontrar o **mrpt_ekf_slam_2d** [3] que se trata de um pacote que implementa o filtro de Kalman Estendido com sensores de alcance, odometria e *landmark* 2D. A complexidade do algoritmo deste pacote é $O(N^3)$ que em grande parte dependente do tamanho do mapa [31]. O pacote **mrpt_ekf_slam_2d** possui uma baixa robustez, pois seu desempenho depende da associação de dados que faz com o sensor de alcance, por isso é mais adequado para mapas menores contendo menos *landmarks* devido à grande carga computacional.

1.2.3 Gmapping

Gmapping é um pacote ROS que fornece SLAM baseado em laser e pode criar um mapa de grade de ocupação 2D [13]. Este algoritmo é baseado no Filtro de Partículas Rao-Blackwellized para construir o mapas de grade. Esta abordagem foi desenvolvida por [14], e leva em consideração a observação mais recente e o movimento do robô para realizar a distribuição proposta. Assim, o algoritmo considera uma abordagem para a realização das operações de reamostragem para reduzir o número de partículas e para prevenir o problema da destruição de partículas. O **gmapping** é um dos pacotes mais famosos do ROS. Esse pacote é mais robusto do que o **mrpt_ekf_slam_2d** devido ao fato de que a associação de dados é baseada em partículas, o que reduz o impacto de uma associação incorreta de dados, sua complexidade é de $O(MLog(N))$, onde M é o número de partículas e N o número de *landmarks*.

1.2.4 Graph SLAM

Por último, mas não menos importante, o **slam_karto** [12] é baseado no GraphSLAM [31], possui uma complexidade de $O(N^2)$ e uma robustez também maior do que o **mrpt_ekf_slam_2d**, em parte devido ao fato de que associações de dados mais antigas podem ser reexaminadas se uma associação de dados incorreta tiver sido realizada. Basicamente, isso reduz o risco de produzir associações errôneas de dados futuros, aumentando assim a robustez.

Em [24], foi realizado um estudo de diversas técnicas de SLAM 2D baseadas em laser disponíveis em ROS, os algoritmos utilizados foram HectorSLAM, KartoSLAM, CoreSLAM, LagoSLAM e Gmapping, os experimentos foram feitos em simulações e no mundo real. Os melhores resultados foram obtidos por Gmapping, KartoSLAM e HectorSLAM com baixa carga de processos na CPU.

No nosso trabalho iremos fazer uma implementação do algortimo EKF usando a Odometria e tag ArUco para diminuir o erro da associação de dados e aumentar a robustez do nosso pacote.

1.3 Objetivo

Nosso trabalho tem por objetivo fazer uma implementação do Filtro de Kalman Estendido baseado em SLAM, usando as informações da odometria e tags ArUco como *landmark* artificial. A nossa abordagem deverá ter um resultado melhor em ambientes simétricos do que outras implementações de SLAM usando nuvem de pontos para associação de dados ou baseado em laser, pois esse tipo implementação gera uma grande incerteza na localização do robô. A principal aplicação desse tipo de implementação é *indoor*, ou seja, dentro de ambientes fechados como por exemplo fábricas, onde existem um alto fluxo de pessoas trabalhando e o

robô precisa lidar com as incertezas do ambiente e de sua localização.

1.4 Estrutura do texto

A estrutura do texto segue com o Capítulo 2 discutindo sobre conceitos fundamentais envolvidos no trabalho apresentado. No Capítulo 3 temos a apresentação da nossa metodologia e posteriormente a apresentação dos resultados no Capítulo 4. Por fim, o Capítulo 5 contém a conclusão e considerações finais.

2

Fundamentação Teórica

Nesta seção apresentaremos alguns conceitos fundamentais que utilizaremos ao longo deste trabalho. Em sua maioria, os conceitos apresentados aqui remete-se a teoria de probabilidade, controle e estimação recursiva bayesiana.

2.1 Probabilidade

2.1.1 Probabilidade Condicional

A probabilidade condicional de um evento B, é a probabilidade que o evento irá ocorrer dado o conhecimento que um evento A já ocorreu. Está probabilidade é escrita como $P(B|A)$, notação da probabilidade de B dado A. Em casos onde o evento A e B são independentes (onde o evento A não tem efeito sobre a probabilidade do evento B), a probabilidade condicional do evento B dado o evento A é simplesmente a probabilidade do evento B, que é $P(B)$. Se o evento A e B não são independentes, então a probabilidade da intersecção de A e B (a probabilidade que ambos os eventos ocorram) é definido por $P(A \cap B) = P(A)P(B|A)$. Desta definição a probabilidade condicional $P(B|A)$ é facilmente obtida dividindo por $P(A)$ [7]:

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \tag{2.1}$$

2.1.2 Lei da Probabilidade Total

A lei da probabilidade total ou soma das probabilidade, expressa a probabilidade total de um resultado que pode ser realizado através de vários eventos distintos [7].

$$P(A) = \sum_{n} P(A \cap B_n) \tag{2.2}$$

Este resultado reveste-se de grande importância por permitir calcular a probabilidade de um evento A quando se dispõem das probabilidades de A condicionadas a eventos B_n e das probabilidades destes eventos condicionais.

2.1.3 Teorema de Bayes

O teorema de Bayes é baseado na expressão $P(B) = P(B|A)P(A) + P(B|A^c)P(A^c)$ [7], na qual afirma que a probabilidade do evento B é a soma das probabilidade condicionais do evento B dado que o evento A tenha ocorrido ou não. Se A e B forem eventos independentes, então podemos escrever $P(B)P(A) + P(B)P(A^c) = P(B)[P(A) + P(A^c)] = P(B)(1) = P(B)$ [7], uma vez que a probabilidade de um evento e seu complemento deve sempre somar 1. A fórmula de Bayes é definida da seguinte forma:

$$(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.3}$$

2.2 Representação do espaço de estado

Uma representação no espaço de estados, consiste nas equações diferenciais de primeira ordem das quais pode ser obtida a solução para as variáveis de estado e a equação algébrica de saída a partir da qual todas as demais variáveis do sistema podem ser obtidas. Uma representação no espaço de estados não é única, uma vez que uma escolha diferente das variáveis de estado leva a uma representação diferente do mesmo sistema [20]. Em um sistema de tempo contínuo o espaço de estado pode ser escrito como :

$$\dot{x}(t) = f(x(t), u(t)), \tag{2.4a}$$

$$y(t) = g(x(t), u(t)), \tag{2.4b}$$

Onde x é o vetor de estado, u é o vetor do sinal de entrada e y é o vetor do sinal de saída. A derivada do vetor de estado $\dot{x}$ é uma função do estado atual $x(t)$ e a entrada atual $u(t)$. Da mesma forma, a saída do sistema $y(t)$ é uma função do estado atual e da entrada atual. Esse modelo pode ser reescrito na forma matricial abaixo [19].

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.5a}$$

$$y(t) = Cx(t) + Du(t) \tag{2.5b}$$

2.3 Filtros bayesiano

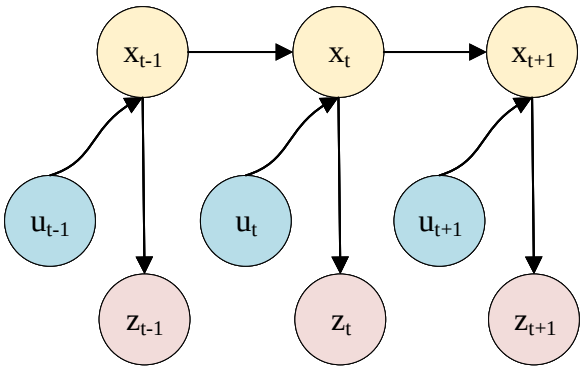
2.3.1 Filtro de Bayes Recursivo

A posição do robô depende do estado atual de x_t. Quando o sistema obtém um novo estado x_{t+1}, que é resultado de algum comando de controle do robô u, a nova posição produzirá uma medição z_{t+1}. A relação entre controle, posição e medição é mostrado na Figura 2.1. Um filtro de Bayes é um algorítimo que calcula a função de densidade de probabilidade para o estado do vetor x_t. Isto é feito em dois passos: predição $\bar{bel}(x_t)$ e a correlação $bel(x_t)$. Na qual o passo de predição é obtido usando apenas o controle u_t e o estado da posição anterior x_{t-1}. Esta predição é alterada no passo de correlação usando os dados do sensor para obter a melhor estimativa [31, p. 26]. O algoritmo do filtro de Bayes é descrito na Tabela 2.1

Tabela 2.1: Algoritmo para filtro de Bayes

1:	**Algorithm Bayes_filter(**$bel(x_{t-1})$**,** u_t**,** z_t**)**
2:	for all x_t do
3:	$\bar{bel}(x_t) = \int p(x_t \mid u_1, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
4:	$bel(x_t) = \eta p(z_t \mid x_t) bel(x_t)$
5:	endfor
6:	return $bel(x_t)$

Figura 2.1: Relação entre a posição, controle e medição



2.3.2 Filtro de Kalman

Filtro de Kalman ou KF (singla proviniente do inglês para Kalman Filter) é um conjunto de equações matemáticas que fornece uma eficiente solução computacional (recursiva) para o método dos mínimos quadrados [33]. Ele implementa um filtro de Bayes, o qual é usado em um

sistemas gaussiano linear [31, p. 40]. Basicamente, o filtro de Kalman calcula a *belief* (crença) para estados contínuos. Este estado diz respeito as informações da posição do veículo. O estado do *belief* no específico instante de tempo $bel(x_t)$, é representado pela sua média μ_t e sua covariância Σ_t.

O filtro de Kalman precisa de três propriedade específica como pré-requisitos [31, p. 41], e elas são:

1. O estado atual de x_t, deve ser representado por uma função linear em seus argumentos e com um ruído gaussiano adicionado, ou seja.

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.6}$$

   Onde x_t e x_{t-1} são os vetores de estado, e u_t é o vetor de controle no tempo t. A_t e B_t são matrizes. A_t é uma matriz quadrada de tamanho $n \times n$, onde n é a dimensão do vetor de estado x_t. B_t tem o tamanho $n \times m$, onde m é a dimensão do vetor de controle u_t. A natureza da fonte do ruído é gaussiana. O estado atual de x_t depende do estado anterior x_{t-1}, do controle u_t e do ruído ε_t.

2. A medição atual z_t é de natureza linear, ou seja

$$z_t = C_t x_t + \delta_t \tag{2.7}$$

   Onde C_t é uma matriz de tamanho $k \times n$, onde k é a dimensão do vetor de medição z_t e a fonte do ruído δ_t é gaussiano. A medição atual z_t depende da estado atual x_t e do ruído δ_t.

3. O ultimo pre-requisito é que a crença inicial $bel(x_0)$ deve ser uma distribuição gaussiana.

O algorítimo de filtro de Kalman mostrado na Tabela 2.2 expressa a crença do estado atual x_t no tempo específico t. O $bel(x_t)$, é expressa pela média μ_t e a covariância Σ_t no qual também são as saídas do algoritmo[31, p. 42].

Tabela 2.2: Algoritmo para filtro de Kalman

1:	**Algorithm kalman_filter(μ_{t-1}, Σ_{t-1}, u_t, z_t)**
2:	$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3:	$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4:	$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5:	$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
6:	$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7:	return μ_t, Σ_t

Basicamente, o algorítimo calcula a predição da média $\bar{\mu}_t$ e a covariância $\bar{\Sigma}_t$ nas linhas 2 e 3. Onde A_t e B_t são matrizes que multiplica o vetor de estado x_{t-1} e o vetor de controle u_t representando o estado de transição de probabilidade com argumento linear, o qual é um dos requisitos para o filtro de Kalman. Essa predição são usadas na fase de correlação, linhas 4-6. A primeira etapa da correlação é calcular o ganha do Kalman denotado por K_t. Além disso o calculo da média μ_t é calculada com relação a média predita $\bar{\mu}_t$ e medição z_t menos a medição predita $C_t\bar{\mu}_t$ multiplicado pelo ganho de Kalman. Por ultimo, a nova covariância Σ_t é calculada atoráveis do fator do ganho de Kalman e a covariância predita $\bar{\Sigma}_t$.

2.3.3 Filtro de Kalman Extendido

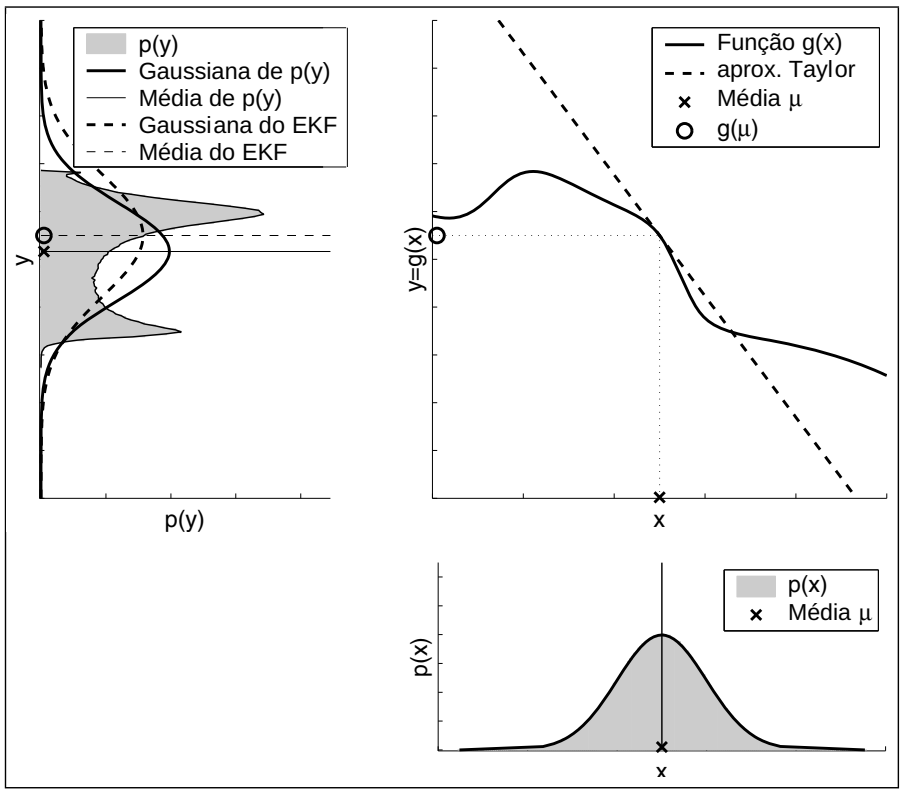
O Filtro de Kalman Estendido ou EKF (sigla proveniente do inglês para Extended Kalman Filter) é uma abordagem para lidar com modelos de aproximação não linear. Em geral os estados de transição e medição de um robô raramente são linear na prática. Por exemplo, se um robô possuir seus movimentos de translação e rotação constantes, o resultado será uma trajetória circular, que não pode ser descrita por um estado de transições linear. O EKF descreve a probabilidade do estado de transição e a probabilidade de medição como funções não linear [31]. As funções precisam passar por um processo de linearização; se alimentarmos um Gaussiano com uma função não linear, a saída não será Gaussiana, e com isso, não podemos aplicar o filtro de Kalman. Para resolver esse problema, é usado expansão de Taylor de primeira ordem em torno da função, isso pode ser visto na Figura 2.2. A distribuição gaussiana resultante pode então ser calculada de forma fechada. Isso permite a extração de características interessantes da distribuição, como a média μ_t e a covariância Σ_t.

O Algorítimo do EKF é apresentado na Tabela 2.4. Esse algoritmo é semelhante ao algoritmo do KF mostrado na Tabela 2.2, suas principais diferenças estão descritas no quado 2.3. As funções de predição linear no filtros de Kalman são substituídas por suas aproximações não lineares em EKFs. Além disso, os EKF usa os Jacobianos G_t e H_t em vez das matrizes do sistema linear correspondentes A_t, B_t e C_t nos filtros de Kalman. O Jacobiano G_t corresponde às matrizes A_t e B_t, e o Jacobiano H_t corresponde a C_t.

Tabela 2.3: Comparação KF e EKF

	KF	EKF
Predição do estado (linha 2)	$A_t\mu_{t-1} + B_t u_t$	$g(u_t, \mu_{t-1})$
Predição das medições (linha 5)	$C_t\bar{\mu}_t$	$h(\bar{\mu}_t)$

Figura 2.2: Linearização de uma função não linear $g(x)$.



Fonte: [31]

Tabela 2.4: Algoritmo para filtro de Kalman Extendido

1:	**Algorithm kalman_filter(**μ_{t-1}**,** Σ_{t-1}**,** u_t**,** z_t**)**
2:	$\bar{\mu}_t = g(u_t, \mu_{t-1})$
3:	$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4:	$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5:	$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
6:	$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
7:	return μ_t, Σ_t

3

Metodologia

Nesta sessão é apresentada a metodologia utilizada no nosso projeto. Nossa abordagem consiste no estudo de quatro etapas: Movimento, Percepção, Representação do mapa e a Localização do robô.

3.1 Movimento e Percepção

3.1.1 Movimento do robô

O movimento do robô compreende a probabilidade de transição de estado $p(x_t|u_t, x_{t-1})$ [31], do qual lê-se como a probabilidade de x_t dado u_t e x_{t-1}; onde x_{t-1} e x_t representam respectivamente o estado predecessor e sucessor do robô, e u_t o comando de movimento no tempo t. O estudo desse movimento tem como base o ruído produzido pelos atuadores. Uma só ação tomada pelo o robô pode ter muitos resultados diferentes, o atuador de um robô móvel introduz incertezas sobre o seu estado futuro, por isso, um simples movimento tende a incrementar uma incerteza na posição final de um robô. A cinemática móvel contém modelos matemáticos para calcular esse movimento e a incerteza produzida por eles [26].

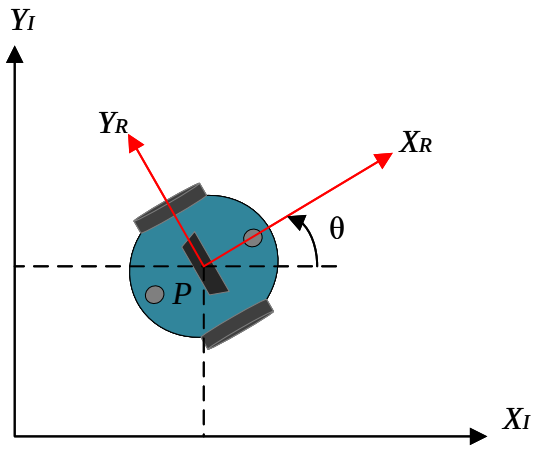
Configuração cinemática

As configurações de um robô rígido é descrito por seis variáveis, sendo três coordenadas cartesianas (x, y, z) e três ângulos de Euler $(roll, pitch, yaw)$. Para um robô operando no plano, essas seis variáveis se restringem a três (x, y, yaw). Então a posição do robô no tempo t pode ser representado pelo vetor de estado:

$$x_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.1}$$

Para determinar a posição especifica do robô no plano, nós estabelecemos a relação entre a referência global do plano e a referência local do robô (ver Figura 3.1). Os eixos X_I e Y_I definem uma base inercial arbitrária no plano como referência global de alguma origem $O : \{X_I, Y_I\}$. Para especificar a posição do robô, escolhemos um ponto P no centro do robô como o ponto de referência da posição. A base $\{X_R, Y_R\}$ é definida por dois eixos relativo ao ponto P e é portanto a referência local. A posição de P na referência global é descrita pelas coordenadas x e y, e a diferença angular entre os referenciais globais e locais é dada por θ [27]. Neste trabalho vamos considerar o exemplo de um robô de drive diferencial com dois pontos de contato.

Figura 3.1: Referência Global e Referência Local



A fórmula geral para a transformação homogênea 2-D [10] é definida por uma matrix 3x3:

$$T = \begin{bmatrix} R_{11} & R_{12} & x \\ R_{21} & R_{22} & y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & x \\ sin\theta & cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

Modelos de movimentos típicos

Os modelos de movimento preveem as mudanças de estado, dada uma sequência de entradas. Todos os sistemas de navegação de robôs móveis com rodas, dependem criticamente desses modelos [25]. É comum encontrar dois tipos de modelos de movimentos para lidar com incertezas: O modelo baseado na odometria e o modelo baseado na velocidade [31]. Modelos baseados na odometria são usados quando os sistemas são equipados com codificadores nas roda. Os modelos baseados em velocidade devem ser aplicados quando nenhum codificador de roda é fornecido. Na prática, os modelos de odometria tendem a ser mais precisos que os modelos de velocidade, pela simples razão de que a maioria dos robôs comerciais não executam comandos de velocidade com o mesmo nível de precisão que pode ser obtido medindo a

revolução das rodas do robô.

Modelo baseado na velocidade (*dead reckoning*)

O modelo de movimento de velocidade ou *dead reckoning* (derivado de *deduced reckoning*) pressupõe que podemos controlar um robô através da velocidade de translação e rotação. Muitos robôs comerciais oferecem interfaces de controle em que o programador especifica essas velocidades. Podemos denotar a velocidade de translação no tempo t por v_t, e a velocidade de rotação por ω_t. Então o vetor de controle pode ser escrito como:

$$u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \tag{3.3}$$

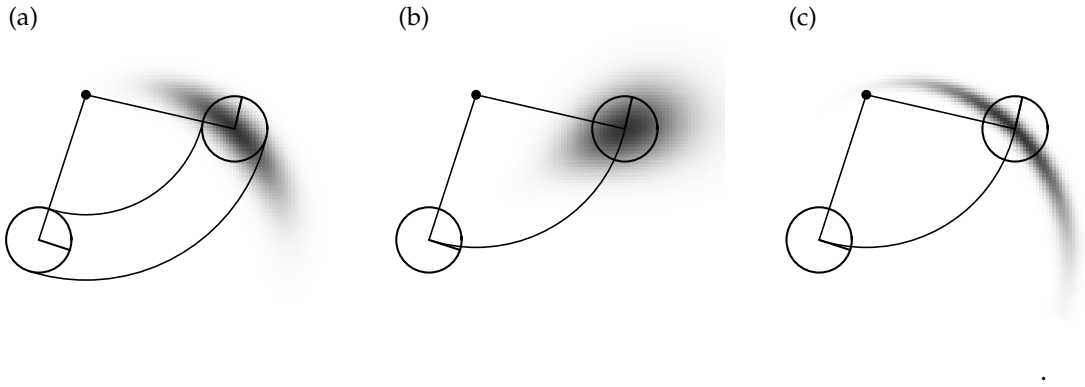
A Tabela 3.1 mostra um algoritmo para calcular a probabilidade $p(x_t|u_t, x_{t-1})$ [31, p. 123], ele recebe como entrada o vetor de estado inicial da posição $x_{t-1} = (x, y, \theta)^T$, o vetor de controle $u_t = (v, w)^T$ e uma hipótese da posição posterior $x_t = (x', y', \theta')^T$. Sua saída é a probabilidade $p(x_t|u_t, x_{t-1})$. Os parâmetros α_1 à α_6 são parâmetros de erro do movimento do robô e a função **prob**(x, b^2) [31] modela esses erros. Ela calcula a probabilidade de seu parâmetro x sob uma variável aleatória centrada em zero com variância b^2.

A Figura 3.2 apresenta três distribuições de probabilidade do modelo de movimento baseado em velocidade. Nos três casos foram atribuídos a mesma velocidade de translação e rotação no robô. A Figura 3.2a possui uma distribuição com um erro moderado nos parâmetros α_1 à α_6. A distribuição da Figura 3.2b é obtida com um pequeno erro angular (parâmetros α_3 à α_4) e um alto erro translacional (parâmetros α_1 à α_2). Figura 3.2c mostra uma distribuição com alto erro angular e um pequeno erro translacional.

Tabela 3.1: Algoritmo de movimento baseado na velocidade

1:	**Algorithm motion_model_velocity(**x_t**,** u_t**,** x_{t-1}**)**
2:	$\mu = \frac{1}{2}\frac{(x-x')cos\theta+(y-y')sin\theta}{(y-y')cos\theta-(x-x')sin\theta}$
3:	$x^* = \frac{x+x'}{2} + \mu(y-y')$
4:	$y^* = \frac{y+y'}{2} + \mu(x-x')$
5:	$r^* = \sqrt{(x-x^*)^2+(y-y^*)^2}$
6:	$\Delta\theta = atan2(y'-y^*, x'-x^*) - atan2(y-y^*, x-x^*)$
7:	$\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$
8:	$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$
9:	$\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{\omega}$
10:	return **prob**$(v-\hat{v}, \alpha_1 v^2 + \alpha_2\omega^2) \cdot$
	prob$(\omega-\hat{\omega}, \alpha_3 v^2 + \alpha_4\omega^2) \cdot$ **prob**$(\hat{\gamma}, \alpha_5 v^2 + \alpha_6\omega^2)$

Figura 3.2: O modelo de velocidade, para diferentes configurações de parâmetros de ruído.

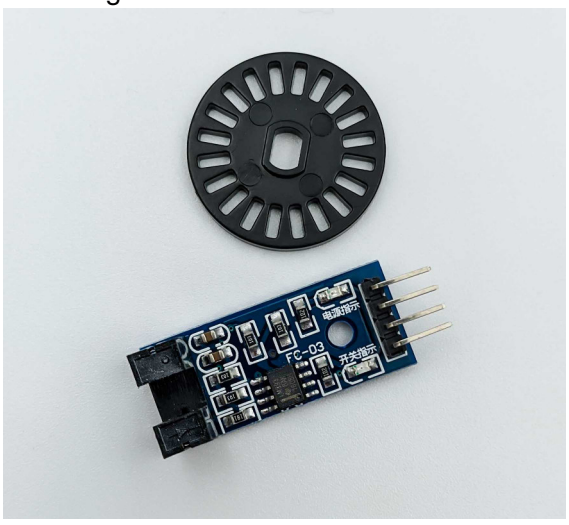


Fonte: [31]

Modelo baseado na odometria

Os modelo de odometria são geralmente aplicados para estimativa, ele não pode ser usado em planejamento de movimento, como o modelo de velocidade, pois a odometria estar disponível somente após a execução do movimento. A odometria é obtida integrando informações do codificador de roda, a Figura 3.3 mostra um exemplo de módulo para odometria, esse módulo requer + 5V e GND para alimentá-los e fornecem uma saída de 0 à 5V. O módulo para o sensor de odometria fornece saída de +5V quando "ver" o branco e uma saída de 0V quando "ver" o preto.

Figura 3.3: Sensor de odometria

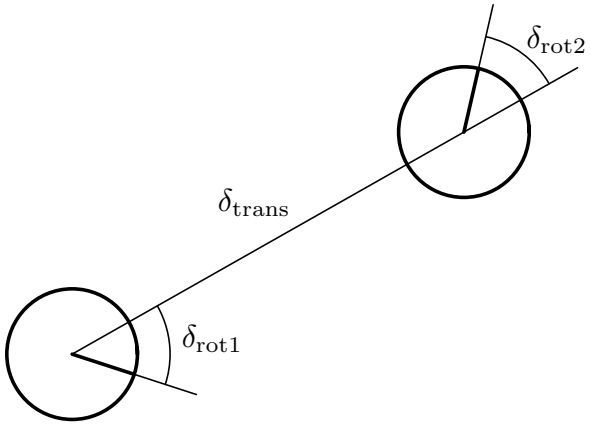


O modelo baseado na odometria usa as informações de movimento relativo [31], conforme o robô avança da posição x_{t-1} para x_t, a odometria informa o avanço relativo no intervalo de tempo $(t-1,t]$ de $\bar{x}_{t-1} = (\bar{x},\bar{y},\bar{\theta})^T$ para $\bar{x}'_t = (\bar{x}',\bar{y}',\bar{\theta}')^T$. As barras indicam que essas são coordenadas internas do robô e a relação com as coordenadas globais são desconhecidas. A informação do movimento é dada pelo par

$$u_t = \begin{bmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{bmatrix} \tag{3.4}$$

Para extrair a odometria relativa, u_t é transformado em uma sequencia de três etapas, conforme mostrado na Figura 3.4. O movimento do robô no intervalo de tempo $(t-1,t]$ é aproximado por uma rotação δ_{rot1}, em seguida por uma translação δ_{trans} e uma segunda rotação δ_{rot2}.

Figura 3.4: Modelo baseado na odometria



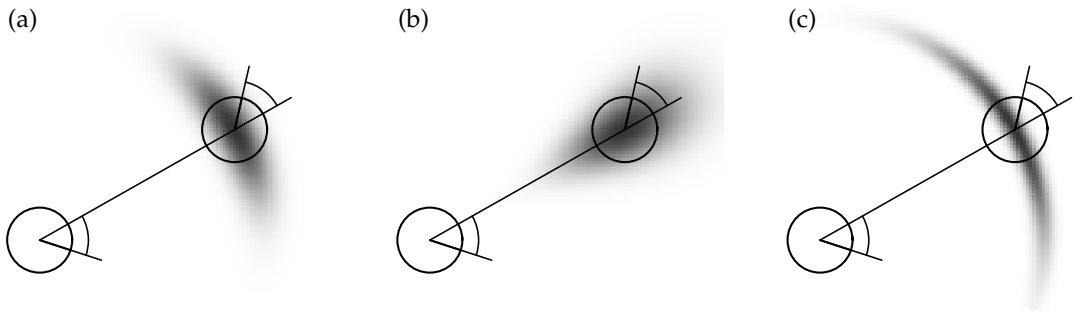
Fonte: [31]

A Tabela 3.2 descreve o algoritmo para calcular $p(x_t|u_t,x_{t-1})$ da odometria, este algoritmo recebe como entrada a posição inicial x_{t-1}, o par das posições obtido pela odometria $u_t = (\bar{x}_{t-1},\bar{x}_t)$ e a hipótese da posição final x_t. A saída é a probabilidade $p(x_t|u_t,x_{t-1})$. A Figura 3.5 apresenta o resultado deste algoritmo para diferentes paramentos de ruído, essas configurações são as mesmas usada para o modelo de movimento baseado na velocidade.

Tabela 3.2: Algoritmo de movimento baseado na odometria

1:	**Algorithm motion_model_odometry(**x_t**,** u_t**,** x_{t-1}**)**
2:	$\delta_{rot1} = atan2(\bar{y}'-\bar{y},\bar{x}'-\bar{x}) - \bar{\theta}$
3:	$\delta_{trans} = \sqrt{(\bar{x}-\bar{x}')^2+(\bar{y}-\bar{y}')^2}$
4:	$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$
5:	$\hat{\delta}_{rot1} = atan2(y'-y,x'-x) - \theta$
6:	$\hat{\delta}_{trans} = \sqrt{(x-x')^2+(y-y')^2}$
7:	$\hat{\delta}_{rot2} = \theta' - \theta - \hat{\delta}_{rot1}$
8:	$p1 = \textbf{prob}(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1\hat{\delta}^2_{rot1} + \alpha_2\hat{\delta}^2_{trans})$
9:	$p2 = \textbf{prob}(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3\hat{\delta}^2_{trans} + \alpha_4\hat{\delta}^2_{rot1} + \alpha_4\hat{\delta}^2_{rot2})$
10:	$p1 = \textbf{prob}(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1\hat{\delta}^2_{rot2} + \alpha_2\hat{\delta}^2_{trans})$
11:	return $p1 \cdot p2 \cdot p3$

Figura 3.5: O modelo da odometria, para diferentes configurações de parâmetros de ruído.



Fonte: [31]

3.1.2 Percepção

A percepção tem como tarefa adquirir conhecimento do ambiente coletando medições dos sensores e extraindo informações dessas medições. Vários tipos de sensores são usados na robótica móvel e muitos deles possuem ruídos que comprometem a acurácia dessas medições [9]. A Figura 3.6 apresenta alguns tipos de sensores mais comuns na robótica móvel.

Figura 3.6: Exemplos de sensores

A robótica probabilística tem por objetivo modelar os ruídos nas medições dos sensores, tais modelos são responsáveis pela incerteza inerente nos sensores do robô. Formalmente o modelo de medição é definido como a distribuição da probabilidade condicional $p(z_t|x_t, m)$, onde x_t é a posição do robô, z_t é a leitura do sensor no tempo t, e m é o mapa do ambiente. Muitos

sensores geram mais que um valor numérico de medição quando solicitada, nós denotamos o número de medições como k e portanto podemos escrever:

$$z_t = \{z_t^1, z_t^2, ..., z_t^k\} \tag{3.5}$$

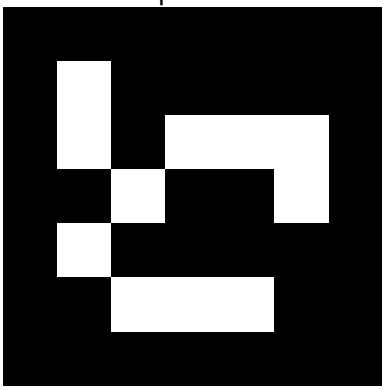
z_t^k representa uma única medição no tempo t e na posição k do vetor z_t. A probabilidade $p(z_t|x_t, m)$ é obtida como o produto da probabilidade das medições individuais

$$p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m) \tag{3.6}$$

Medições com *landmark* artificial

Em muitas aplicações de robótica, as características correspondem a objetos distintos no mundo físico, esses objetos são comumente chamados de *landmark*, que servem como ponto de referência na navegação em robótica móvel. Neste trabalho utilizaremos o marcador ArUco [21] como *landmark* artificial. O ArUco (ver Figura 3.7) é um marcador quadrado sintético composto por uma borda preta larga e uma matriz binária interna que determina seu identificador *id*. A borda preta facilita sua rápida detecção na imagem e a codificação binária permite sua identificação e aplicação de técnicas de detecção e correção de erros.

Figura 3.7: Exemplo de Marcador ArUco



Cada posição do *landmark* é representado pelo símbolo l_n, onde n é o n-esímo *landmark* do vetor L, que é chamado de mapa de características [31]. Um mapa de características do ambiente é uma lista de objetos do ambiente e suas localizações, formalmente definido na equação 3.7.

$$L = \{l_1, l_2, ..., l_n\} \tag{3.7}$$

Uma associação de dado conhecidos, significa que quando o robô mede distância e o ângulo de um *landmark*, ele sempre sabe qual *landmark* está medindo. As características extraídas pelo sensor (em nosso caso uma câmera) vem com informação da distância e o ângulo, que

iremos representar respectivamente por r e ϕ e sua assinatura por s. Se f é uma função de extração de característica, logo as características extraídas pelo sensor serão dadas por $f(z_t)$ e portanto o vetor de características será representado por uma tupla:

$$f(z_t) = \{f_t^1, f_t^2, ...\} = \{ \begin{pmatrix} r_t^1 \\ \phi_t^1 \\ s_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^2 \\ s_t^2 \end{pmatrix}, ... \} \tag{3.8}$$

O resultado do modelo de medição é formado pela correspondência do i-esímo vetor de característica no tempo t com o j-esímo mapa de característica.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(L_{j,x} - x)^2 + (L_{j,y} - y)^2} \\ atan2(L_{j,y} - y, L_{j,x} - x) - \theta \\ s_j \end{pmatrix} \tag{3.9}$$

Posição do robô em relação ao marcador ArUco

A posição do robô em relação ao marcador ArUco é dado por:

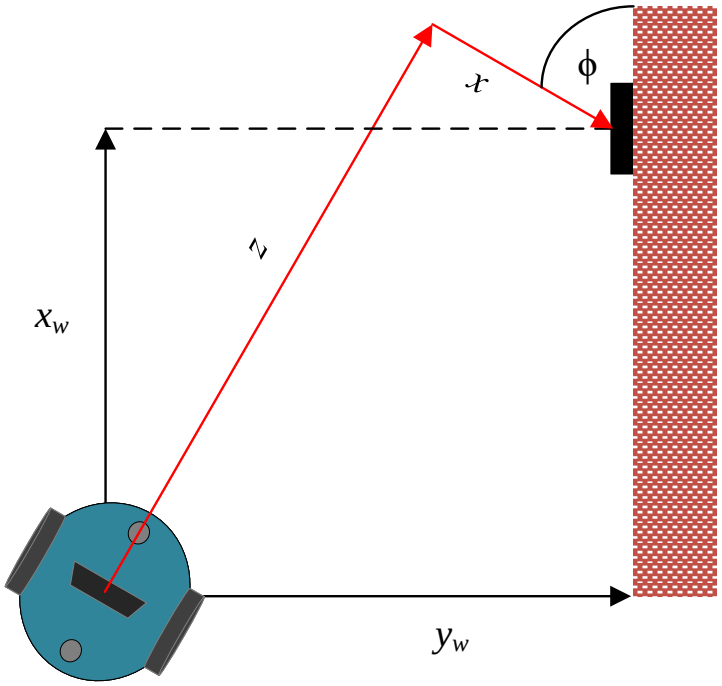
$$x_w = x \cdot cos(\theta) - z \cdot sin(\theta) \tag{3.10}$$

$$y_w = z \cdot cos(\theta) - x \cdot sin(\theta) \tag{3.11}$$

$$\theta_w = \theta = 180 - \phi \tag{3.12}$$

A Figura 3.8 mostra a relação entre a posição da câmera e a tag ArUco. Onde x é a posição do marcador em relação ao centro óptico da câmera ao longo do eixo x, z é a distância entre a câmera e o plano perpendicular ao eixo óptico da câmera que está cruzando o centro do marcador, θ é o ângulo complementar entre a câmera e o marcador [1].

Figura 3.8: Determinando posição do robô em relação ao marcador



Autor: [1]

A posição global do robô x_w, y_w e θ_w de acordo com a Figura 3.9 é dado por:

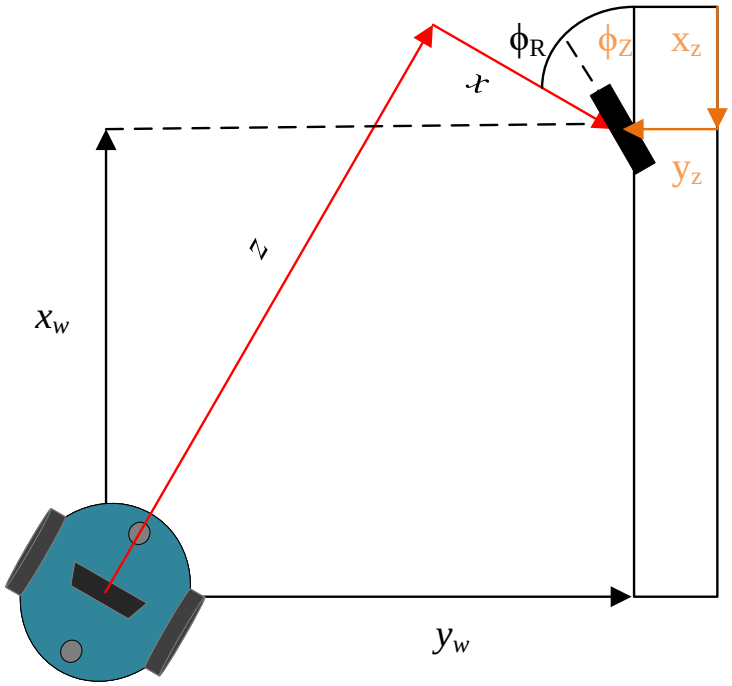
$$x_w = x \cdot cos(\theta_R + \theta_Z) - z \cdot sin(\theta_R + \theta_Z) \tag{3.13}$$

$$y_w = z \cdot cos(\theta_R + \theta_Z) - x \cdot sin(\theta_R + \theta_Z) \tag{3.14}$$

$$\theta_w = \theta_R + \theta_Z \tag{3.15}$$

O sistema de detecção da ArUco é capaz de localizar mais de um marcador em uma determinada imagem. Então é possível calcular a posição do robô com base na posição de cada marcador.

Figura 3.9: Determinando posição global do robô em relação ao marcador



Autor: [1]

3.2 Representação do mapa

3.2.1 A Representação

A abordagem escolhida para representar o mapa do ambiente, tem um impacto nas escolhas disponíveis do robô. Frequentemente, a precisão da representação da posição é limitada pela precisão do mapa [27]. Quando escolhemos o tipo de abordagem para representar o mapa, três critérios devem ser avaliados:

- A precisão do mapa deve corresponder a precisão da qual o robô precisa para executar suas tarefas;
- A precisão do mapa e as características contidas nele, devem corresponder a precisão dos sensores e as informações extraídas deles;
- A complexidade da representação do mapa tem impacto direto na complexidade computacional do mapeamento, localização e navegação.

Neste trabalho, utilizaremos uma abordagem bastante popular na robótica móvel, conhecida como *occupancy grid* [8], proposta pelo Alagoano Alberto Elfes *et al*.

3.2.2 Mapa com grade de ocupação

A grade de ocupação é um campo aleatório multidimensional que mantém estimativas estocásticas do estado de ocupação das células [8]. No nosso trabalho, utilizamos uma grade de ocupação com duas dimensões. Formalmente a grade de ocupação é definida como:

$$p(m|z_{1:t}, x_{1:t}) \tag{3.16}$$

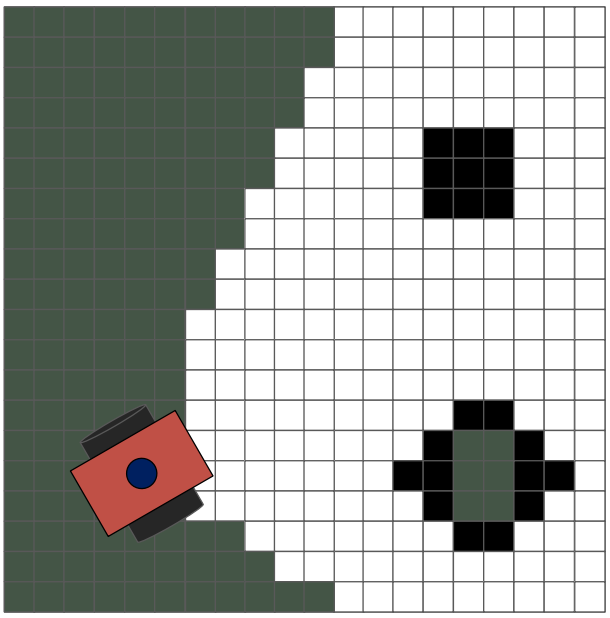
Onde m é o mapa, $z_{1:t}$ é um vetor com todas as medições do sensor no tempo t, $x_{1:t}$ é a trajetória feita pelo robô definida por todas as suas posições. Vamos denotar m_i como a célula da grade de ocupação com índice i, que possui um número finito de índice.

$$m = m_i \tag{3.17}$$

Cada m_i contém um valor de ocupação binária, que representa se a célula estar ocupada ou vazia. Vamos definir o valor 1 para ocupada e 0 para vazia. A notação $p(m_i = 1)$ refere-se a probabilidade da célula estar ocupada.

Figura 3.10 mostra uma representação de um mapa com representação em grade, as células verdes representam os espaços que não foram explorados, as células brancas e pretas representam respectivamente os espaços livres e ocupados por algum objeto no mapa.

Figura 3.10: Exemplo de uma representação de mapa de grade de ocupação

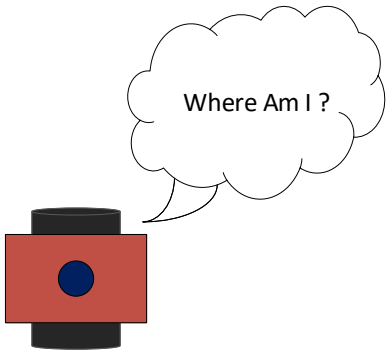


3.3 Localização do robô

3.3.1 O problema da localização e sua estratégia de solução

A localização na robótica móvel é um problema para determinar a posição do robô em relação a um dado mapa do ambiente [31]. Considere um robô móvel em um ambiente conhecido, quando ele começa a se mover, sabe precisamente sua localização, pelas informações coletadas da odometria. Devido à incerteza da odometria, depois de alguns movimentos o robô passa a receber informações incertas. Para evitar que a incerteza de posição cresça sem limites, o robô deve se localizar em relação ao mapa do ambiente. Para localizar, o robô pode usar seus sensores integrados (ultrassom, sensor de alcance, câmera) para fazer observações de seu ambiente. As informações fornecidas pela odometria do robô, mais as informações fornecidas por tais observações exteroceptoras, podem ser combinadas para permitir que o robô se localize novamente em relação ao seu mapa [27].

Figura 3.11: Robô tentando se localizar



3.3.2 Localização com EKF

A localização com Filtro de Kalman estendido ou Localização com EKF é derivado da localização por Cadeira de Markov [31]. A localização do EKF representa a crença $bel(x_t)$ em seu primeiro e segundo momento, através da média μ_t e da covariância Σ_t. A aplicação do filtro de Kalman à localização requer colocar o problema de localização do robô como um problema de fusão de sensores [27].

3.3.3 Conceito do SLAM

A localização e o mapeamento simultâneos são um problema em que um robô em movimento precisa construir um mapa de um ambiente desconhecido, enquanto calcula simultaneamente sua posição dentro desse mapa [6]. Se tivermos o mapa real do ambiente disponível,

estimar a trajetória do robô seria um problema de localização simples [5]. Da mesma forma, se a real trajetória do robô fosse conhecida, construir um mapa seria uma tarefa relativamente simples [32]. No entanto, quando a trajetória do robô e o mapa são desconhecidos, a localização e o mapeamento devem ser considerados simultaneamente, daí o nome SLAM (*Simultaneous Localization and Mapping*) [18].

Diversas áreas se beneficiam da implementação de veículos autônomos com algoritmos SLAM, alguns exemplos são: industria de mineração, exploração subaquática e exploração planetária [6]. O problema de SLAM em geral pode ser formulado usando uma função de densidade de probabilidade denotada por $p(x_t, m|z_{1:t}, u_{1:t})$, onde x_t é a posição do veículo, m é o mapa, $z_{1:t}$ é um vetor com todas as medições e $u_{1:t}$ é um vetor dos sinais de controle do veículo, que são os próprios comandos de controle ou a odometria, dependendo da aplicação.

3.3.4 Associação de Dados

Basicamente, o conceito de associação de dados é investigar a relação entre os dados mais antigos e os novos dados coletados. Em um contexto SLAM, é necessário relacionar medições mais antigas a medições mais recentes. Isso permite o processo de determinar a localização dos *landmarks* no ambiente e, portanto, fornece informações sobre a posição do robô no mapa [27].

3.3.5 EKF SLAM

O algoritmo EKF SLAM aplica o EKF (Tabela 2.4) ao SLAM usando associação de dados de máxima verossimilhança. Ao fazer isso, o EKF SLAM está sujeito a uma série de aproximações e suposições limitantes, entre elas o seu mapa. O mapa do EKF SLAM, são baseados em características. Eles são compostos de *landmarks*. Por razões computacionais, o número de *landmarks* geralmente é pequeno (por exemplo, menor que 1.000) [31]. Além disso, a abordagem do EKF tende a funcionar bem quanto menos ambíguos forem os *landmarks*. Por este motivo, o EKF SLAM requer uma engenharia significativa de detectores de características, às vezes usando *landmarks* artificiais como características.

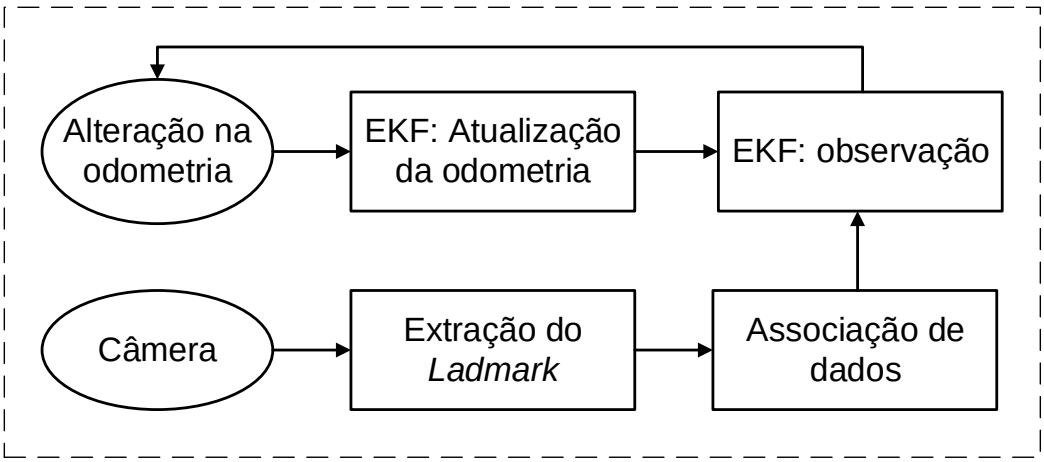
Correspondência de dados conhecidas

Em nosso projeto, fizemos a implementação do algoritmo EKF SLAM com correspondências conhecidas. Ao fazer SLAM com correspondência de medições conhecidas, o número de *landmarks* é fixo. Além disso, quando obtemos uma medição de distância e direção, sabemos a qual *landmark* essa medição corresponde. Isso significa que não precisamos usar o método de associação de dados de máxima verossimilhança.

Processo do algorítmo

O objetivo do processo do algoritmo é usar o ambiente para atualizar a posição do robô. Quando o robô é ligado, o codificador da roda (odometria) coletará as informações da posição do robô. Além disso, os *landmarks* do ambiente também serão extraídos por meio das observações da câmera que é montada no robô. Essas observações são associadas à base de dados e atualizadas no algoritmo EKF [23]. Este processo é ilustrado na Figura 3.12.

Figura 3.12: Processo do EKF SLAM



A alteração na odometria começa quando o robô se move, então as incertezas pertinentes a posição do robô, são adicionados no EKF pela atualização da odometria. Os *landmarks* são então extraídos do ambiente com a nova posição do robô. O robô então tenta associar esses *landmarks* à base de dados. Os *landmarks* observados são então usados para atualizar a posição dos robôs no EKF. De forma resumida o processo do EKF SLAM, consiste em duas etapas:

- Predição

  A etapa da predição é expressa pela média e covariância predita. Elas são calculadas com base na entrada do controle, as matrizes, a média anterior e a covariância anterior [30].

- Correlação

  A ideia principal por trás a etapa de correlação é a associação de dados, ou seja comparar um *landmarks* observado e a base de dados. Além disso, o ganho de correção, também conhecido como ganho de Kalman, é calculado. Basicamente, é um fator de ganho de correção necessário para atualizar a média atual e a covariância do estado atual. A média e a covariância atuais não dependem apenas do ganho de Kalman, mas também levam em consideração a média e a covariância previstas [30].

Complexidade Computacional

O custo computacional do algoritmo EKF SLAM é bastante caro em comparação com outros algoritmos SLAM. Quando os *landmarks* são detectados, eles são adicionados ao vetor de estado do filtro, e o mapa com N *landmarks* identificados aumentará linearmente [30]. Em termos de números, o uso de memória é $O(N^2)$. O custo de computação para calcular uma etapa do algoritmo é de aproximadamente $O(N^2)$ [30], o custo completo para calcular todo o algoritmo é $O(N^3)$, que depende fortemente do tamanho do mapa [18].

4

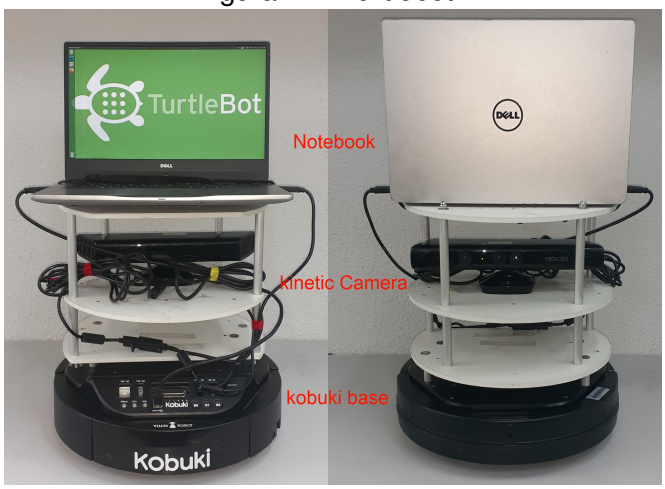
Implementação e Resultados

Para implementar o algoritmo de SLAM utilizamos a linguagem de programação python junto com o *Robot Operating System* (ROS). O ROS é um conjunto de bibliotecas e ferramentas de software que ajuda a criar aplicativos para robôs. O ROS funciona em um sistema distribuído que utiliza uma rede de *Node* (Nós). *Node* é o termo para um executável conectado à rede ROS. O *publisher* (locutor) envia uma mensagem para o *subscriber* (ouvinte), que recebe a mensagem.

Em nosso projeto utilizamos o Turtlebot 2 (ver Figura 4.1). O TurtleBot é um robô pessoal de baixo custo com software de código aberto criado na Willow Garage por Melonee Wise e Tully Foote em novembro de 2010 [29]. O kit TurtleBot consiste em uma base móvel, sensor de distância 2D/3D, laptop e o kit de hardware de montagem e seu SDK estar disponível no wiki do ROS. Foram utilizados também o Gazebo e o Rviz que são simuladores 3D para robótica.

A implementação do SLAM foi dividida em três partes: Percepção, Localização e Mapeamento.

Figura 4.1: Turtlebot 2

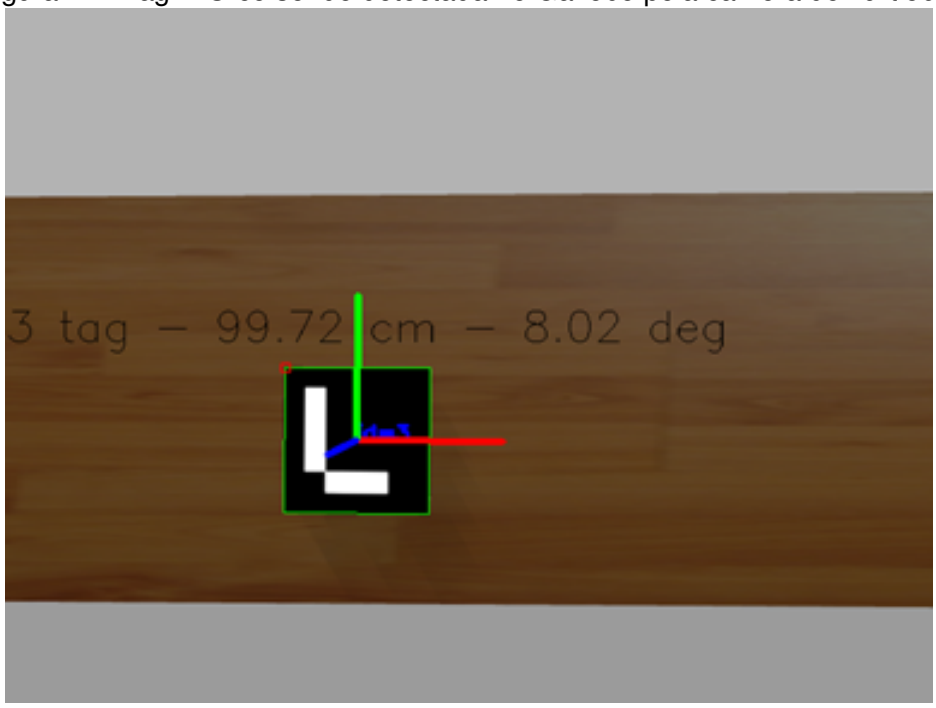


4.1 Percepção

Para a parte de percepção, criamos um *node* no ROS, neste *node* usamos a câmera RGB do robô para detectar as tags ArUco que funcionarão como landmark artificial. Como estamos trabalhando com um problema de associação de dados conhecidos, o robô conhece as posições e quantidades de landmaks. Por meio da biblioteca OpenCV [21] detectamos as coordenadas x, y, e z das tags ArUco em relação a câmera do robô. O *node* publica uma lista com as características das tags detectadas, essas características incluem distância, ângulo e identificador.

A Figura 4.2 apresenta uma tag ArUco no simulador Gazebo sendo detectada pela câmera do Turtlebot 2. As informações do id, distância e o ângulo em relação a câmera são mostrados na figura. Embora estejamos trabalhando com um simulador, os erros de medição estão inerentes ao modelo, através da câmera do simulador.

Figura 4.2: Tag ArUrco sendo detectada no Gazebo pela câmera do Turtlebot.



4.2 Localização

Na parte de localização usamos o filtro de kalman estendido para mesclar as informações da posição fornecida pela odometria que provem do pacote padrão de navegação do Turtlebot e a estimativa da posição fornecida pela associação de dados dos landmarks que provém do node de percepção criado anteriormente, com isso, conseguimos uma média e uma covariância da real posição do robô. O algoritmo de filtro de Kalman, requer que os erros adicionados ao

espaço de estado sejam todos gaussiano, por isso atribuímos empiricamente os erros associados a detecção do id, distância e ângulo, os erros da odometria já são simulados pelo pacote de de Navegação do Turtlebot 2.

4.3 Mapeamento

Para o mapeamento, fizemos um subscribers para dois *node*. Um fornecido pelo pacote de navegação padrão do Turtlebot 2, o scan, que retorna as medições do infravermelho Kinetec 360, e outro para posição estimada, que foi criado na parte de localização. O node scan fornece um vetor com as medições feita pelo sensor de profundidade e o ângulo de cada uma dessas medições. Com os pontos coletados pelo sensor de medição e a estimativa da posição fornecida construímos um mapa de ocupação das medições feita pelo laser e as posições dos landmarks.

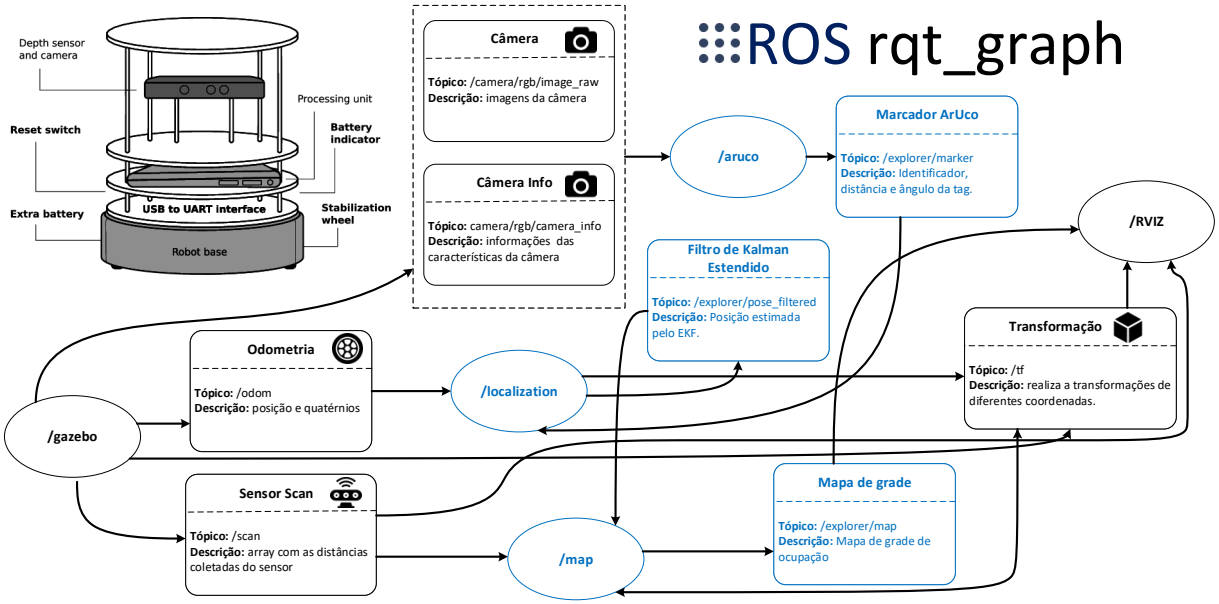
4.4 Gráfico de computação ROS

O cálculo em ROS é feito usando uma rede de processos chamada de node. Essa rede de computação pode ser chamada de gráfico de computação. Os dois principais conceitos no gráfico de computação são: node, e topics. Cada conceito contribui de maneira diferente no gráfico.

- *Node*: é o processo que executa a computação. Cada node é escrito usando bibliotecas de cliente ROS, como roscpp e rospy. No gráfico são representados por elipses.
- *Topics*: Cada mensagem no ROS é transportada usando barramentos nomeados chamados topics. Quando um node envia uma mensagem por meio de um topic, podemos dizer que o node está publicando um topic. Quando um node recebe uma mensagem por meio de um topic, podemos dizer que o node está escutando um topic. No gráfico, os topics são representados por retângulos.

A Figura 4.3 apresenta um gráfico gerado pelo comando rqt_graph do ROS. Os *nodes* e *topics* destacados na cor azul foram desenvolvidos em nosso projeto. O *node* /aruco, recebe informações da câmera e publica o identificador, distância e ângulo da tag ArUcos detectada através do *topic* /explorer/marker. O *node* /localization recebe informações da odometria e do *topic* /explorer/marker, então publica o *topic* /explorer/pose_filtered, que contém a estimativas da posição robô calculado pelo EKF. O *node* map, recebe informações da câmera de profundidade do robô, por meio do /scan e do /explorer/pose_filtered. O /tf tem a função de fazer a transformação das coordenadas do robô. Finalmente o /frontier_map gera um mapa de grade que é enviado para o simulador RVIZ.

Figura 4.3: Gráfico de Computação - ROS



4.5 Resultados na simulação

Para esta fase do trabalho utilizamos a versão simulada do Turtlebot 2 pelo Gazebo. Foi construído um quadrado com as paredes do simulador, onde o robô ficará inicialmente posicionado no centro (ver Figura 4.4 e Figura 4.5), vários algoritmos de visão computacional possuem certas dificuldades para distinguir locais simétricos, porém a abordagem utilizada neste projeto consegue identificar cada lado do quadrado sem ambiguidade, porque sabemos exatamente a posição do *landmark* que a câmera está detectando. Em cada parede foi coloca dois *landmarks*, totalizando 8 *landmarks* que são suficientes para o tamanho do ambiente construído.

4.5.1 Teste de desempenho

Para testar a eficiência do nosso algoritmo de localização, comparamos com o pacote de localização *Adaptive Monte Carlo Localization* (amcl) [11] que usa um filtro de partículas [31] para rastrear a posição de um robô diante um mapa conhecido. Primeiro gravamos um percurso feito pelo robô com o comando *rosbag*, em seguida utilizamos o comando *rqt_multiplot* e visualizamos as trajetórias geradas pelos algoritmos. Figura 4.6 apresenta a janela do *rqt_multiplot*, a trajetória em vermelho é a posição real do simulador, a trajetória em azul é o nosso algoritmo de localização e a trajetória em verde é a posição estimada do AMCL. Como pode ser observado na figura, tivemos um bom resultado nesse teste.

Como já foi mencionado, a precisão da localização, influencia diretamente a precisão do mapa. O pacote *gmapping* [13] fornece SLAM baseado em laser, com o *gmapping* podemos criar um mapa de grade de ocupação 2D a partir de dados de laser e pose coletados por

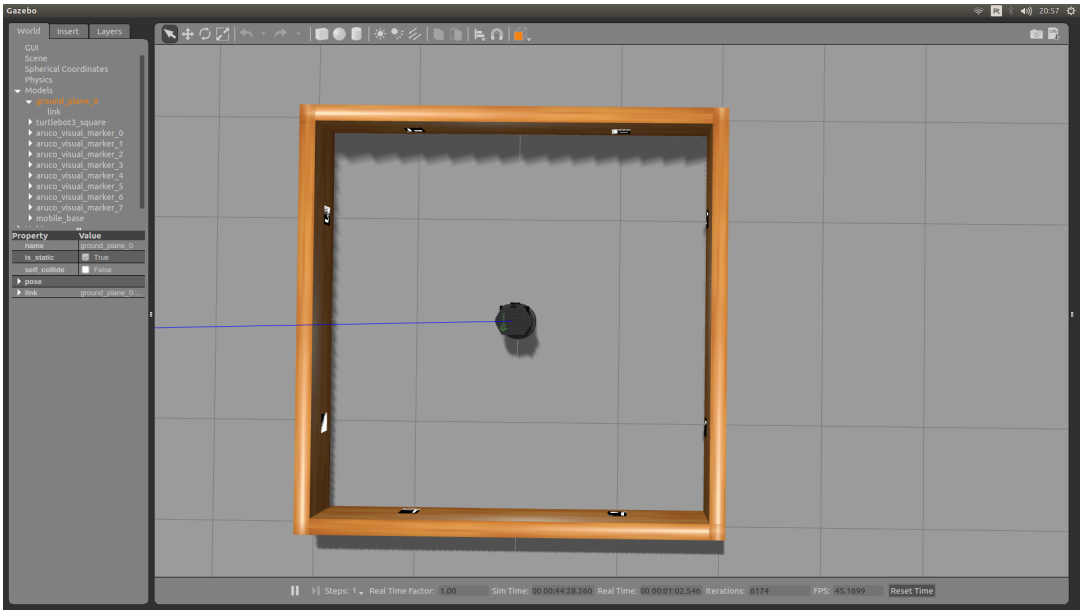


Figura 4.4: Simulador Gazebo com o Turtlebot 2 - Superior

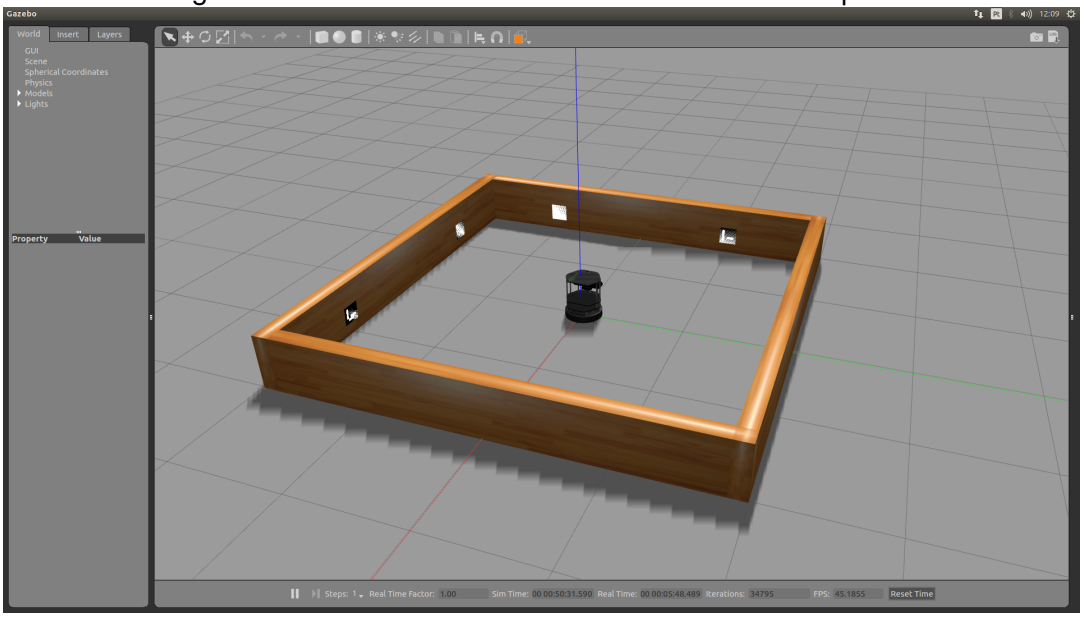
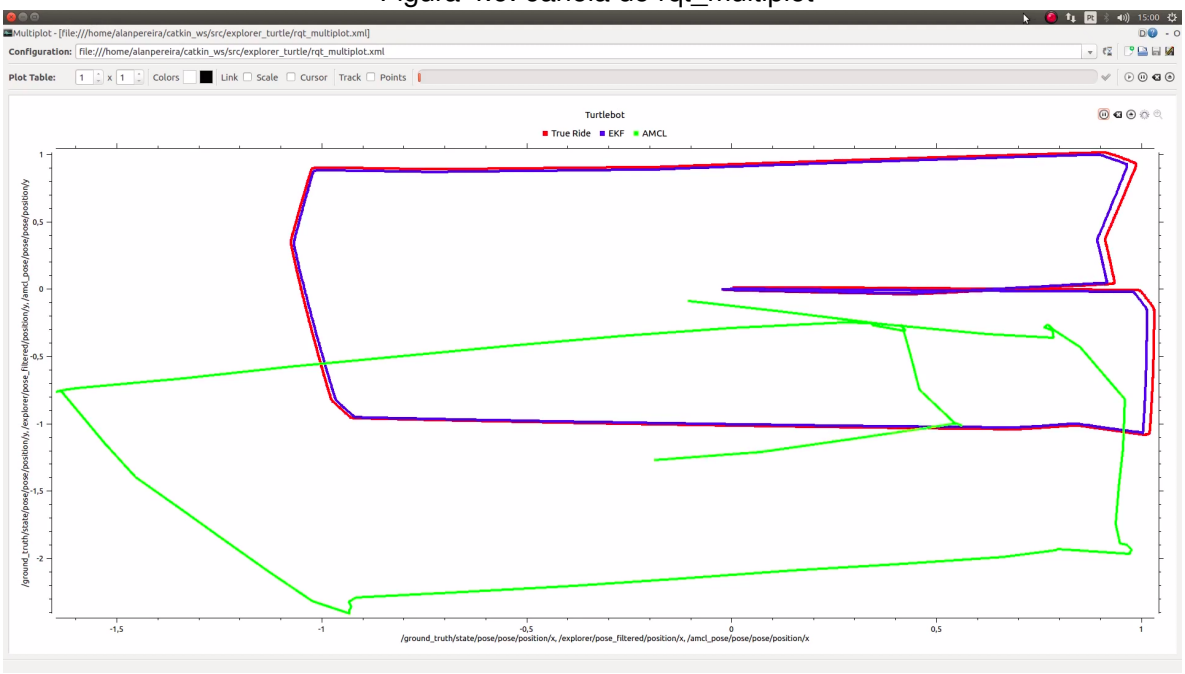


Figura 4.5: Simulador Gazebo com o Turtlebot 2 - Lateral

Figura 4.6: Janela do rqt_multiplot



um robô móvel. A Figura 4.7 mostra o mapa gerado pelo *gmapping* e a Figura 4.8 o mapa gerado pelo nosso algoritmo no simulador Rviz. Apesar do *gmapping* ser um dos melhores pacote do ROS para geração de mapa, ele não lida bem com ambientes simétricos, por ser baseado em laser, a nossa implementação com ArUrco retorna uma localização mais precisa. Na Figura 4.8 podemos ver o quadrado que foi construído anteriormente e a posição do robô no final da simulação. As falhas no mapa é consequência dos erros inerente a câmera do simulador, no início da simulação a odometria entrega informações consistente da posição do robô, enquanto que o pacote de percepção entrega dados que contém uma aproximação dessa posição; após algum tempo os erros da odometria vai se acumulado e o filtro de Kalman mescla as informações, retornando a posição mais próxima da real.

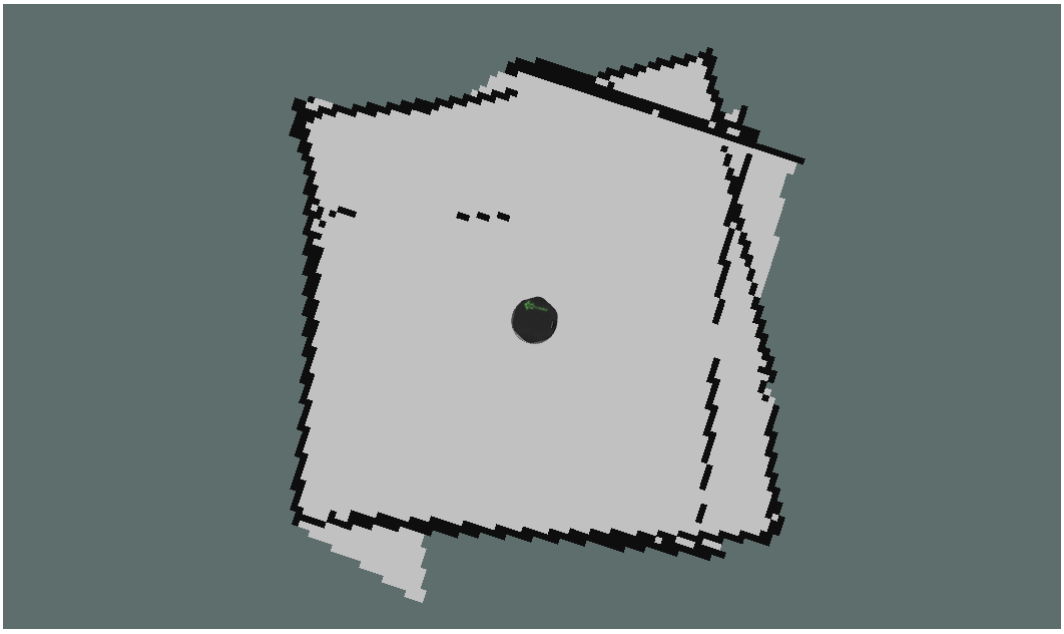


Figura 4.7: Mapa gerado pelo *gmapping*

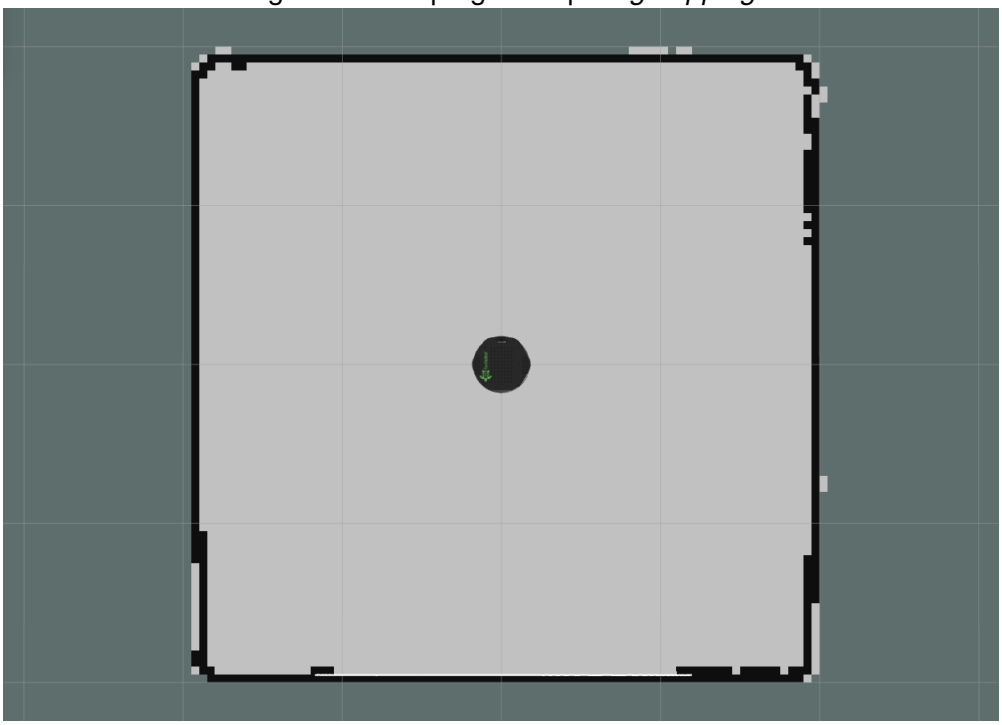


Figura 4.8: Mapa gerado pelo EKF SLAM

5

Conclusão e Considerações Finais

Neste projeto, implementamos um pacote de localização e mapeamento simultâneo (SLAM) para o Sistema operacional de robôs (ROS) que usa o filtro de kalman estendido (EKF) para mesclar as informações da odometria e das tags ArUco detectada pela câmera do robô. O pacote também gera um mapa com grade de ocupação por meio da localização estimada pelo filtro de kalman e o sensor Kinetc 360 acoplado no robô.

Criamos um ambiente com um simulador Gazebo e testamos a performance da nossa implementação, ela foi comparada com outros pacotes de SLAM e os resultados mostram uma melhoria no mapa e na localização do robô, o motivo dessa melhoria se justifica porque algumas implementações de SLAM baseada em laser ou nuvem de pontos, não tem um bom desempenho em ambientes simétricos.

Implementações de SLAM como essa podem ser encontradas em armazéns de distribuição de produtos, como por exemplo na Amazon. Esses armazéns geralmente possuem uma grande quantidade de prateleiras enfileiradas, formando grandes corredores, e esses corredores criam um ambiente simétrico, dificultando assim a execução de alguns algoritmos de SLAM, por isso diversas aplicações têm adotado o uso de tags para diferenciar cada espaço.

A princípio planejamos executar esse pacote em um robô real, disponível no instituto de computação da Universidade Federal de Alagoas, mas por motivos das restrições impostas por conta da pandemia, não conseguimos executar esse plano. Para trabalhos futuros, iremos fazer um teste em um robô real e estamos considerando implementar um algoritmo de navegação autônoma que possa ser usado junto com esse pacote.

Referências bibliográficas

[1] Andrej Babinec, Ladislav Jurišica, Peter Hubinskỳ, and František Duchoň. Visual localization of mobile robot using artificial markers. *Procedia Engineering*, 96:1–9, 2014.

[2] M Billinghurst, A Clark, and G Lee. A survey of augmented reality. found trends hum comput interact 8: 73–272, 2015.

[3] Jose Luis Blanco Claraco. mrpt ekf slam 2d, 2010. gmapping, http://wiki.ros.org/mrpt_ekf_slam_2d.

[4] Jose Luis Blanco Claraco and Machine Perception. Development of scientific applications with the mobile robot programming toolkit. *The MRPT reference book. Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Málaga, Spain*, 40, 2008.

[5] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1322–1328. IEEE, 1999.

[6] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.

[7] Valerie J Easton and John H McColl. Statistics glossary, 2002.

[8] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[9] HR Everett. *Sensors for mobile robots*. CRC Press, 1995.

[10] James D Foley, Andries Van Dam, et al. *Fundamentals of interactive computer graphics*, volume 2. Addison-Wesley Reading, MA, 1982.

[11] AH Brian P Gerkey. Adaptive monte carlo localization implementation, 2008.

[12] AH Brian P Gerkey. slam karto, 2008. slam karto, http://wiki.ros.org/slam_karto.

[13] AH Brian P Gerkey. Gmapping in ros, 2015. gmapping, http://wiki.ros.org/slam-gmapping.

[14] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1): 34–46, 2007.

[15] S. Kohlbrecher. slam karto, 2008. Hector Mapping, http://wiki.ros.org/hector.

[16] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE international symposium on safety, security, and rescue robotics*, pages 155–160. IEEE, 2011.

[17] Zhiwei Kong and Qiang Lu. A brief review of simultaneous localization and mapping. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 5517–5522. IEEE, 2017.

[18] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[19] Erik Narby. Modeling and estimation of dynamic tire properties, 2006.

[20] Norman S Nise. *CONTROL SYSTEMS ENGINEERING, (With CD)*. John Wiley & Sons, 2007.

[21] OpenCV. Opencv documentation, 2020. OpenCV Documentation, http://www.opencv.org.

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

[23] Søren Riisgaard and Morten Rufus Blas. Slam for dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*, 22(1-127):126, 2003.

[24] Joao Machado Santos, David Portugal, and Rui P Rocha. An evaluation of 2d slam techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE, 2013.

[25] Neal Seegmiller and Alonzo Kelly. Enhanced 3d kinematic modeling of wheeled mobile robots. In *Robotics: Science and Systems*, pages 2–1, 2014.

[26] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

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

[28] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Machine intelligence and pattern recognition*, volume 5, pages 435–461. Elsevier, 1988.

[29] IEEE Spectrum. Turtlebot inventors tell us everything about the robot, 2013. Interview, https://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot.

[30] Cyrill Stachniss. Robot mapping ekf slam, 2021. Acesso: 26/02/2021, http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam04-ekf-slam.pdf.

[31] Sebastian Thrun. *Probabilistic robotics*. ACM New York, NY, USA, 2002.

[32] Sebastian Thrun. Exploring artificial intelligence in the new millenium, chapter robotic mapping: a survey. *Morgan Kaufmann*, pages 1–35, 2002.

[33] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. *Chapel Hill, NC, USA*, 1995.