Universidade Estadual de Londrina

ENZO GUIDO AMERICANO DA COSTA

MONITORAMENTO E PREDIÇÃO DE SISTEMAS AUTÔNOMOS PARA ROBÔS AMR

LONDRINA
2024

ENZO GUIDO AMERICANO DA COSTA

MONITORAMENTO E PREDIÇÃO DE SISTEMAS AUTÔNOMOS PARA ROBÔS AMR

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Adilson Luiz Bonifácio

LONDRINA

2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Sobrenome, Nome.
Título do Trabalho : Subtitulo do Trabalho / Nome Sobrenome. - Londrina, 2017.
100 f. : il.

Orientador: Nome do Orientador Sobrenome do Orientador.
Coorientador: Nome Coorientador Sobrenome Coorientador.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2017.
Inclui bibliografia.

1. Assunto 1 - Tese. 2. Assunto 2 - Tese. 3. Assunto 3 - Tese. 4. Assunto 4 - Tese. I. Sobrenome do Orientador, Nome do Orientador. II. Sobrenome Coorientador, Nome Coorientador. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

ENZO GUIDO AMERICANO DA COSTA

MONITORAMENTO E PREDIÇÃO DE SISTEMAS AUTÔNOMOS PARA ROBÔS AMR

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Adilson Luiz Bonifácio
Universidade Estadual de Londrina

Prof. Dr. Segundo Membro da Banca
Universidade/Instituição do Segundo
Membro da Banca – Sigla instituição

Prof. Dr. Terceiro Membro da Banca
Universidade/Instituição do Terceiro
Membro da Banca – Sigla instituição

Prof. Ms. Quarto Membro da Banca
Universidade/Instituição do Quarto
Membro da Banca – Sigla instituição

Londrina, A definir de 2024.

Este trabalho é dedicado à minha família e amigos que me incentivaram a seguir em frente.

AGRADECIMENTOS

Agradeço a todos que me motivaram a continuar com meus estudos durante a minha formação superior.

Agradecimento especial aos meus pais, Andrea Guido e Dalmar Americano da Costa Neto, que sempre estão me motivando a seguir em frente e me apoiando nos melhores e piores momentos.

Agradeço também meus amigos que sempre me alegraram e auxiliaram a terminar minha formação.

Por fim, mas não menos importante, agradeço meu orientador Adilson Luiz Bonifácio a tornar estre trabalho possível.

"It's quiet, but can you hear it? Little by little, the tides are changing, and the world with them.
(Eiichiro Oda)

COSTA, E. G. A.. **Monitoramento e predição de sistemas autônomos para Robôs AMR**. 2024. 96f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2024.

RESUMO

O objetivo deste trabalho é unir informações que geralmente se encontram separadas e segmentadas em diversos artigos e outras pesquisas relacionadas com o tema de localização e mapeamento simultâneos (SLAM), com o intuito de entender o que esse desafio proporciona como solução e avanço sobre diversos quesitos, desde comerciais e empresariais até utilidades domésticas. Além disso, será apresentada uma simulação de aplicação real utilizando todos os meios e métodos necessários que implementam a funcionalidade de uma localização espacial precisa em tempo real para robôs autônomos.

Palavras-chave: LIDAR. ICP. SLAM. Filtro de Kalman.

COSTA, E. G. A.. **Monitoring and prediction of autonomous systems for AMR Robots**. 2024. 96p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2024.

ABSTRACT

The objective of this task is to unite information that is generally separated and segmented into several articles and other research related to the topic of location and simultaneous mapping (SLAM), with the aim of understanding what this challenge provides as a solution and advancement on various issues, from commercial and business to utilities domestic. Furthermore, a simulation of a real application will be presented using all the necessary means and methods that implement the functionality of a spatial location real-time accuracy for autonomous robots.

Keywords: LIDAR. ICP. SLAM. Kalman Filter.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

LIDAR	*Light Detection and Ranging*
ICP	*Iterative Closest Point*
SLAM	*Simultaneous Localization and Mapping*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
AGV	*Automated Guided Vehicle*
AMR	*Autonomous Mobile Robot*
EKF	*Extended Kalman Filter*
GPS	*Global Positioning System*
RGB	*Red Blue Green*
TCC	Trabalho de Conclusão de Curso

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, temos testemunhado um avanço significativo na área de sistemas embarcados, robótica e inteligência artificial, impulsionado pela demanda por sistemas autônomos capazes de realizar tarefas complexas, tais como [5, 6, 7]. Um dos principais desafios enfrentados por esses sistemas é a necessidade e habilidade de compreender e interagir com o ambiente ao seu redor. O bom funcionamento desses sistemas depende de alguns aspectos importantes relacionados a arquitetura entre seus diferentes componentes de forma organizada e coordenada. Essa arquitetura é composta de partes mecânicas, tais como rodas e motores para locomoção, bem como dispositivos digitais tais como os sensores que permitem a análise do ambiente externo, computadores e controladores para realizar as atividades propostas, além da comunicação entre as partes e, por fim, uma estrutura externa capaz de armazenar todas as partes em um único modelo completo.

Nesse contexto, os algoritmos de predição de movimento, ou SLAM[1], têm se destacado como uma solução promissora a fim de orquestrar todos esses componentes em prol de um objetivo específico. O SLAM [8, 9] é uma técnica que combina a capacidade de mapeamento de um ambiente desconhecido juntamente com a localização simultânea do agente autônomo dentro desse ambiente. Em outras palavras, o algoritmo SLAM permite que um robô, veículo autônomo ou qualquer outro sistema móvel, obtenha informações sobre o ambiente e, ao mesmo tempo, determine sua própria posição e orientação.

A necessidade de algoritmos SLAM surge em diversos contextos, tais como robótica móvel, realidade aumentada, veículos autônomos, drones, aplicações industriais e até mesmo uso pessoal como aspiradores de pó automáticos. Esses algoritmos são fundamentais para que os sistemas autônomos possam navegar em ambientes desconhecidos ou em constante mudança, construindo mapas detalhados enquanto se localizam com precisão.

As primeiras ideias para lidar com sistemas autônomos surgiram com os algoritmos de filtro de Wiener [10] e Kalman [11], baseados na estatística probabilística, para estimar, de forma precisa, a posição e orientação dos objetos de estudo, com uma fonte de dados limitada e imprecisa que provém do ambiente externo. O filtro de Kalman, então, começou a ser aplicado como técnica de estimativa de estado no problema do SLAM [12, 13]. Isso levou ao desenvolvimento do SLAM baseado em filtro de Kalman como conhecemos atualmente.

[1] do inglês, *Simultaneous Localization and Mapping*

1.1 Histórico

O SLAM surgiu durante a Conferência de Robótica e Automação do IEEE em 1986. Naquela época, os métodos baseados em probabilidade estavam sendo gradualmente introduzidos nas áreas de robótica e inteligência artificial, como a proposta de Rudolf E. Kalman [11]. Kalman propôs um método estatístico, que ficou conhecido como filtro de Kalman, para estimar o estado de um sistema dinâmico em tempo real.

Diversos pesquisadores então começaram a estudar sobre técnicas estimativas para resolver questões relacionadas à mapeamento e localização, tais como Cheeseman e Smith [14], que propuseram um método para relacionar o erro estimado entre localizações relativas de objetos, Durrant-Whyte [15], que reuniu diversos modelos essenciais com implementações e demonstrações de métodos e algoritmos para solucionar o problema SLAM, Chatila [16], que discutiu problemas e possíveis soluções para mapeamento utilizando robôs autônomos, como perda de referência, ambiente volátil e condições irregulares, entre outros que também desempenharam papéis valiosos, contribuindo de maneira significativa para o desenvolvimento inicial de um problema que, mais tarde, foi considerado um dos maiores avanços na área de robótica móvel e autônoma.

O problema do SLAM ganhou destaque à medida que os avanços na robótica começaram a permitir que máquinas interagissem de maneira mais autônoma com o mundo. No início, a robótica se baseava em ambientes controlados onde os robôs seguiam rotas predefinidas, como robôs AGV que seguem faixas demarcadas e apenas podem percorrer sobre elas [17] ou eram comandados remotamente [18]. Entretanto, à medida que as aplicações robóticas se expandiram para ambientes não estruturados, mutáveis e desconhecidos, tornou-se essencial que os robôs fossem capazes de se localizar por conta própria e navegar de forma independente.

Uma das soluções para a captura de dados externos do ambiente, tais como ângulo referencial, distância percorrida e até reconhecimento de estruturas em planos 2D e 3D, foi o uso de sensores. Os dados fornecidos por sensores são reconhecidos em algoritmos e procedimentos que os transformam em equações matemáticas que resumem o ambiente, os movimentos do robô autônomo, sua posição e orientação. A abordagem probabilística é frequentemente usada para tratar a incerteza associada às medições dos sensores, às estimativas e predições.

Os primeiros trabalhos sobre SLAM exploraram métodos para resolver esse problema através de modelos estatísticos que pudessem representar incerteza nas estimativas de localização e mapeamento. Os algoritmos, como o filtro de Kalman estendido (EKF) [19], foram aplicados inicialmente para combinar dados dos sensores e estimativas de movimento em robôs a fim de atualizar o mapa e a localização.

Com o avanço das pesquisas abordagens mais sofisticadas foram surgindo, como os

Filtros de Partículas (Particle Filters) [20] e o uso de técnicas de otimização como do SLAM baseado em grafo [21]. Além disso, a crescente disponibilidade de sensores avançados, como câmeras 3D [22] e LIDAR [23], possibilitou melhorias significativas na qualidade e detalhes dos mapas gerados pelo SLAM. O desenvolvimento deste trabalho ficará mais focado nos algoritmos de predição, bem como a utilização do sensor LIDAR para mapeamentos 2D, juntamente com o filtro de Kalman [4].

1.2 Contextualização do problema

Sistemas autônomos são capazes de realizar tarefas e tomar decisões de forma independente, sem a necessidade de intervenção humana direta. Um desafio crucial para esses sistemas está na obtenção de informações precisas sobre o ambiente para estimar sua posição e orientação em relação a esse ambiente. Neste cenário o filtro de Kalman desempenha um papel fundamental, particularmente em sistemas onde os dados de entrada são afetados por erros de medição e ruído, como é o caso de sensores utilizados em sistemas autônomos.

O filtro de Kalman funciona combinando duas fontes de informações: as medições e as previsões do estado do sistema. As medições são fornecidas pelos sensores, câmeras, radares, entre outros dispositivos que capturam informações do ambiente. A partir desses dados, previsões do estado do sistema são obtidas por meio de um modelo matemático que descreve o comportamento dinâmico do sistema. O filtro de Kalman combina essas informações de maneira ponderada, levando em consideração as incertezas associadas a cada uma delas.

Em sistemas autônomos, o filtro de Kalman é frequentemente usado para estimar a posição, orientação e velocidade do veículo ou robô em relação a um mapa do ambiente. Os sensores, como GPS, giroscópios, acelerômetros e câmeras, fornecem informações sobre a movimentação e percepção do sistema. Essas informações são utilizadas pelo filtro para atualizar continuamente a estimativa do estado do sistema, permitindo que o sistema autônomo entenda sua posição em relação ao ambiente e tome decisões apropriadas. Neste sentido o SLAM desempenha um papel crucial na percepção do ambiente, permitindo que os veículos tomem decisões precisas e seguras em tempo real.

Portanto, os algoritmos SLAM representam uma solução revolucionária para a navegação e mapeamento autônomos, proporcionando a capacidade de explorar, compreender e interagir com o ambiente de maneira eficiente e precisa. Com o contínuo progresso nessa área, podemos esperar um futuro em que sistemas autônomos desempenhem um papel cada vez mais significativo em nossa sociedade, trazendo benefícios nas áreas de transporte, logística, segurança e muito mais.

Assim, o objetivo deste trabalho é aplicar o modelo SLAM, juntamente com os sen-

sores necessários e algoritmos de medição e monitoramento, e desenvolver um sistema para solucionar o problema de localização simultânea de robôs AMR em ambientes não controlados. Neste cenário, o robô AMR deve ser capaz de realizar suas tarefas e navegar de forma autônoma, sem a necessidade de interferência humana direta, além de fornecer uma estimativa de sua posição em tempo real de forma rápida e precisa.

1.3 Organização do Trabalho

O restante deste trabalho está organizado em mais quatro capítulos.

O Capítulo 2 apresenta a fundamentação teórica-metodológica. O capítulo descreve a arquitetura do modelo, com as estruturas e equipamentos necessários para sua aplicação. Também é apresentado um ambiente unidimensional, onde os conceitos e notações relacionados a aplicação são introduzidos, bem como os cálculos e suas derivações. Em seguida, o capítulo também introduz o modelo multivariado, usando um ambiente bidimensional, onde os conceitos e cálculos previamente apresentados são modificados para se adequar a este modelo bidimensional, usando transformações matriciais para lidar com múltiplas dimensões. Por fim, os algoritmos e estruturas de código relacionados a implementação dos modelos são abordados.

O Capítulo 3 propõe uma simulação, projetada para um ambiente real tridimensional, usando o modelo bidimensional. A solução do problema então consiste em localizar um robô autônomo a partir do modelo de simulação implementado. Nesta simulação, serão apresentados a modelagem do SLAM para o problema de localização, a sua implementação como linguagem, bibliotecas, funções e lógicas por trás e os resultados de testes seletos que demonstrem seu funcionamento.

Em sequência, o Capítulo 4 apresenta os experimentos e resultados obtidos a partir das simulações executadas e explicita os meios para o funcionamento devido do SLAM em um ambiente 3D.

Por fim, o Capítulo 5 explicita as conclusões obtidas a partir dos experimentos e suas implicações.

2 FUNDAMENTAÇÃO TEÓRICO-METODOLÓGICA

A fundamentação do trabalho é composta das seções que descrevem a arquitetura do sistema SLAM, os modelos unidimensional e multivariado para o problema SLAM, bem como os algoritmos relacionados a implementação do sistema. A Seção 2.1 apresenta a arquitetura do sistema SLAM, com os mecanismos e softwares externos que fazem parte da solução proposta, como por exemplo, os sensores e o sistema ICP. A seção também apresenta os conceitos-chave, juntamente com as notações e equações associadas. O modelo SLAM unidimensional é introduzido na Seção 2.2, descrevendo os cálculos e derivações necessárias para a simulação e predição dos objetos num ambiente unidimensional. Na Seção 2.3 é apresentado o modelo multivariado para o SLAM. As adaptações dos cálculos são descritas de forma a atender uma aplicação num ambiente bidimensional baseada em operações matriciais e correlações entre variáveis. A seção também apresenta as transformações de um ambiente real tridimensional para um ambiente bidimensional para que, dessa forma, uma solução seja encontrada usando o modelo bivariado. A fundamentação termina com a Seção 2.4 que descreve os algorítimos que implementam os modelos apresentados para o sistema SLAM.

2.1 Arquitetura do SLAM

A arquitetura do SLAM é composta pelo sistema de predição, propriamente dito, com o filtro de Kalman, e de outros componentes/sistemas auxiliares. Sensores, por exemplo, são componentes importantes que fazem parte do SLAM e servem para obtenção de dados em tempo real. Já o ICP é um sistema auxiliar que trata os dados obtidos e os transforma nos dados a serem manipulados pelo filtro de Kalman. Essa arquitetura é essencial para aplicações que envolvem desde rastreamento de automóveis até navegação autônoma com modelagem de mapa e localização em tempo real.

2.1.1 Sensores e o Sistema ICP

Os algoritmos SLAM utilizam uma grande variedade de sensores para coletar dados do ambiente, para que sejam processados através técnicas de visão e geometria computacional, processamento de imagens e filtros. O objetivo é extrair informações significativas do ambiente, identificar obstáculos, determinar a posição do agente autônomo e criar um mapa consistente e atualizado que possa se adaptar a variações no ambiente [24].

Os sensores LIDAR[1] [25], por exemplo, são amplamente usados em aplicações de

[1] do inglês, *Light Detection and Ranging*

robótica móvel, veículos autônomos, mapeamento topográfico, monitoramento ambiental, entre outros. Esse tipo de sensor fornece uma maneira precisa e rápida de obter informações detalhadas sobre o ambiente, permitindo que os sistemas autônomos tomem decisões confiáveis. Um sensor LIDAR é um dispositivo que usa emissão de lasers para medir distâncias e obter informações detalhadas sobre a geometria e características de um ambiente tridimensional. Por isso, o princípio de um sensor LIDAR é semelhante ao de um radar, que se utiliza das ondas de rádio, ao invés de feixes de luz.

O funcionamento de um sensor LIDAR, de maneira mais precisa, consiste na emissão de um pulso curto e intenso de luz dentro do ambiente, com uma duração e potência que podem variar dependendo do sensor. O pulso de luz viaja pelo ambiente até atingir um objeto, como uma parede, uma árvore ou qualquer outra superfície refletora, quando o pulso de luz atinge um objeto, uma parte da luz é refletida em direção ao sensor LIDAR, contendo informações sobre a distância aproximada do objeto. O sensor possui um detector sensível à luz que registra o pulso de retorno. Como o tempo entre a emissão do pulso e a detecção do pulso refletido é medido com alta precisão, o sensor LIDAR calcula a distância entre o sensor e o objeto com base nos tempos de ida e volta do pulso, e na constante da velocidade da luz. A emissão dos pulsos de luz ocorrem repetidamente e em diferentes direções, seja por meio de um scanner giratório ou de uma matriz de feixes de luz, permitindo a construção detalhada de um mapa bidimensional, ou até mesmo tridimensional, do ambiente.

A Figura 1 demonstra o funcionamento de um sensor LIDAR 2D. Os dados coletados

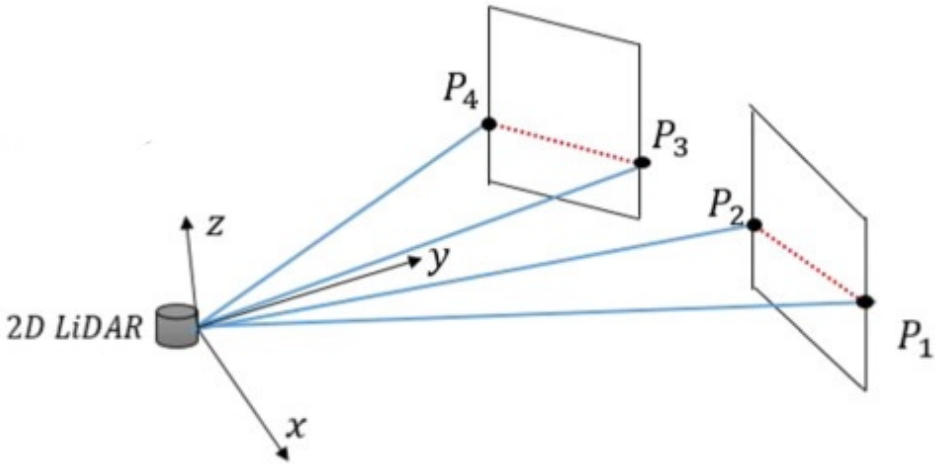

Figura 1 – Exemplo de um sensor LIDAR 2D [1].

pelo sensor LIDAR são retornados na forma de pontos no plano cartesiano (coordenadas x e y no plano 2D ou x, y e z no plano 3D) e processados por um algoritmo que remove ruídos, filtra pontos inválidos e obtém informações úteis, como a posição tridimensional dos objetos,

sua forma, tamanho e intensidade de retorno [26].

Algumas aplicações mais atuais sobre SLAM têm usado métodos de processamento de imagens e inteligência artificial com câmeras RGB de profundidade para obtenção dos dados externos e permitindo o mapeamento 3D, como no caso dos algoritmos SLAM em carros TESLA [27]. No entanto, os sensores LIDAR ainda são largamente usados devido a sua maior eficiência em ambientes menores e menos voláteis que exijam apenas um mapeamento 2D, como nos casos de residências e galpões de logística. O menor custo em relação às câmeras RGB de profundidade, maior estabilidade e confiança nos dados (em um plano bidimensional) e menor custo computacional, não exigindo um alto poder de processamento e uma base de dados muito grande, são fatores que reforçam a aplicação dos sensores LIDAR em soluções SLAM.

Os sensores, em geral, retornam apenas pontos no plano cartesiano correspondentes à posição de um objeto. Daí a importância do ICP[2], um algoritmo que transforma os dados de entrada dos sensores para o filtro de Kalman, obtendo os parâmetros de translação, rotação, orientação, velocidade e aceleração do objeto de estudo. Essa transformação se dá pelo ajuste da translação e rotação sobre uma nuvem de pontos, tridimensionais ou bidimensionais, como as obtidas a partir de um sensor LIDAR, para melhor alinhar com uma outra nuvem de pontos de referência, ou seja, quanto uma nuvem precisa transladar e rotacionar para que esteja na mesma posição e orientação de outra nuvem base.

Ao comparar a nuvem de pontos capturada pelo LIDAR com uma nuvem de pontos de referência prévia (por iteração), o ICP [28] então procura a transformação, rotação e translação, que melhor alinha os pontos de ambas as nuvens. Essa transformação é estimada iterativamente, refinando o resultado a cada iteração até convergir para uma estimativa mais precisa e ajustada de acordo com as necessidades do sistema. A estimativa dessa transformação é obtida pelo ICP através da otimização de uma função objetiva de minimização da diferença entre os pontos correspondentes das duas nuvens.

A Figura 2 mostra o funcionamento do ICP na prática. A nuvem de pontos na cor preta representa a leitura base, e o objetivo é alinhar a outra nuvem de pontos em relação a essa base. Os pontos na cor verde representa a primeira leitura do estado do objeto, que se encontra deslocada em relação a leitura base. Após uma iteração a nuvem de cor verde se torna vermelha, indicando que houve um deslocamento para um estado mais próximo do desejado. A cada iteração as nuvens de pontos são comparadas entre si ponto a ponto, procurando a menor correspondência entre eles, ou seja, a menor distância entre um ponto da nuvem preta com um ponto da nuvem verde, assim que encontrada essa correspondência

[2] do inglês, *Iterative Closest Point*

o ajuste de translação e rotação é passado à todos os pontos resultando na nuvem de ponto vermelha, mais aproximada do que se deseja chegar. A cada iteração os pontos tendem a ter uma distância cada vez menor, onde o usuário pode definir o grau de exatidão necessário com base no número de iterações que devem ser realizadas.

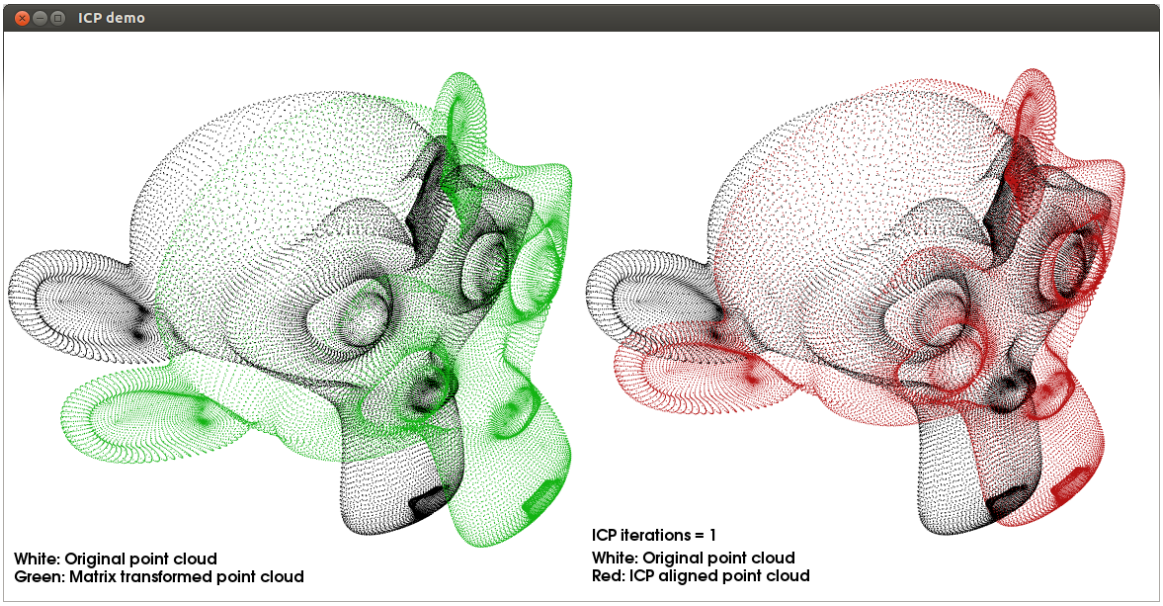

Figura 2 – Exemplo de funcionamento ICP 3D [2]

Uma vez que a transformação é estimada, o resultado pode ser usado para se obter a posição e orientação do sensor LIDAR em relação à nuvem de pontos de referência ou ao mapa. Além da estimativa de localização, o ICP também pode ser utilizado para estimar outros valores, como velocidade e aceleração, através do cálculo da diferença entre as transformações estimadas em diferentes instantes ao longo do tempo. Com os dados obtidos da transformação, assim, o filtro de Kalman é alimentado com os dados obtidos da transformação, ou seja, a nuvem de pontos para posição, rotação, translação, velocidade, entre outros.

2.1.2 Filtro de Kalman

O filtro de Kalman, componente central do SLAM, é um algoritmo de predição e rastreamento de objetos que utiliza medições realizadas ao longo do tempo (contaminadas com ruído e outras incertezas), gerando resultados próximos aos valores reais das grandezas medidas e associadas. Esse filtro combina informações de medições anteriores, usando um modelo linear onde os erros de medição e da dinâmica do sistema são independentes, com informações do modelo dinâmico para estimar o estado atual de maneira ótima. Com base nesse modelo, o algoritmo fornece uma estimativa de estado mais precisa que os métodos anteriores, mesmo na presença de ruído.

O modelo deste filtro utiliza de medidas espaciais realizadas iterativamente para realizar suas predições, são elas posição (x, y) e orientação (graus) no plano cartesiano, velocidade de deslocamento e aceleração do objeto de estudo. Combinando essas informações é possível, assim, estimar a posição atual no plano e prever qual será a posição na próxima iteração.

Existem dois métodos de se utilizar o filtro de Kalman, a primeira é utilizando de medições externas onde os aparelhos de medição não fazem parte do objeto de estudo, como um radar ou câmera, assim como em [29], onde é apresentado um exemplo em que um astrônomo utiliza de um telescópio e do filtro de Kalman para calcular a posição de objetos celestiais. Dessa forma as medições obtidas não são relativas ao objeto de estudo, isto é, tendo a posição espacial fixa do aparelho de medição no plano/mapa é, então, resultado um valor de medição correspondente à coordenada espacial no plano, dessa forma, se um radar na posição x, y mede o objeto de estudo a três metros em sua frente com orientação de 90ž graus, a medição resultante do objeto é de $x, y + 3$.

O segundo método é utilizando medições internas, onde os aparelhos de medição fazem parte do objeto de estudo, como odometria nas rodas ou, neste caso, um lidar, assim como em [30]. Dessa forma as medições são obtidas relativas ao objeto de estudo de modo contrário à primeira forma, ou seja, como não se sabe a posição espacial do aparelho de medição no plano/mapa, pois o mesmo faz parte do objeto de estudo, e deste não se sabe sua posição real, já que este é o valor que se deseja estimar, as medições não podem ser dadas como posição no plano e sim como deslocamento, orientação e sentido, da mesma forma como apresentado em 2.1.1, dessa forma, supondo que o objeto de estudo está estimado na posição x, y e o ICP apresenta uma medição de deslocamento de três metros em sua frente com orientação de 180ž, a medição resultante do objeto é de $x - 3, y$.

Então, a principal diferença entre os métodos apresentados é a referência do aparelho de medição, onde o primeiro método apresenta uma medida sob o objeto de estudo relativa à posição fixa do aparelho, não dependendo da coordenada anterior, medida ou estimada, do objeto de estudo, já o segundo método apresenta uma medida sob o objeto relativa à sua própria posição estimada imediatamente anterior, dependendo deste valor. O primeiro método é demonstrado em um exemplo de aplicação na seção 2.2.3 e o segundo também será demonstrado como exemplo de aplicação na seção 2.3.2 além de ser utilizado no modelo de implementação do sistema SLAM como proposta de solução para um problema x na seção 3.

A execução do algoritmo se inicia com os parâmetros assumindo alguns valores, como posição inicial, precisão da medição, variância das estimativas e etc. A cada iteração, o algoritmo computa um valor estimado e um predito sobre o objeto em análise/observação. O algoritmo sempre converge de maneira correta, mesmo para valores atribuídos inicialmente

muito diferentes da realidade. Nesse caso, ocorre que as primeiras iterações do algoritmo resultam em valores bastante distintos, gerando uma convergência mais ou menos responsiva.

O valor estimado do sistema indica o estado atual mais próximo do exato, considerando medições imprecisadas, ruídos do ambiente e atrasos de envio e recebimento de dados (lag) entre iterações. Essa estimativa leva em conta a predição da iteração anterior. Já o valor predito numa iteração é uma suposição sobre o estado do sistema na próxima iteração e que leva em conta o valor estimado da iteração atual. Assim, a ideia central do filtro de Kalman é combinar estimativas ponderadas do estado do sistema de uma iteração prévia com uma nova medição, levando-se em conta as incertezas mencionadas. O filtro, ainda, atribui os maiores pesos às informações mais confiáveis e os menores às informações menos confiáveis, permitindo que o algoritmo se adapte dinamicamente às mudanças no sistema e melhore a precisão das estimativas realizadas.

2.2 O Sistema SLAM Unidimensional

A arquitetura previamente apresentada mostra como os sensores e sistemas auxiliares se interligam com o SLAM. Já esta seção é dedicada especialmente a introdução do SLAM com o filtro de Kalman unidimensional. Porém, primeiramente são estabelecidos alguns conceitos matemáticos básicos, tais como média e variância, e alguns conceitos mais avançados, tais como distribuição multivariada e covariância, que serão úteis mais adiante. Em seguida, as equações e transformações que definem o modelo matemático do sistema de predição unidimensional são abordadas. O objetivo é mostrar como o modelo matemático calcula as estimativas dos sistemas com base nas observações externas, no modelo de movimento e nas informações estatísticas. Por fim, o filtro de Kalman multivariado, uma extensão do modelo unidimensional que lida com múltiplas variáveis em sistemas mais complexo é apresentado.

2.2.1 Conceitos e Notações

O SLAM exige alguns conceitos fundamentais, além de uma notação apropriada, para facilitar sua compreensão e implementação. Portanto, alguns conceitos estatísticos e princípios de estimação são descritos nesta seção.

Média: estado do sistema conhecido, onde a concentração dos dados em uma distribuição é maior, dada por μ;

Valor esperado: média ponderada de eventos aleatórios levando em conta suas probabilidades associadas, ou seja, representa o valor médio de uma variável aleatória, calculado

ponderando os possíveis valores dessa variável pela probabilidade de ocorrência de cada um desses valores, dado por E;

Variância: quantifica o grau de dispersão dos valores em um conjunto de dados em relação à média, representada por σ^2;

- Variância da estimativa do estado, é o grau de erro sobre uma estimativa para o valor real, representada por p
- Variância da medição, é o desvio padrão entre o valor real e o valor medido, representada por r

Covariância: diferente da variância, não se concentra na dispersão ou variabilidade de uma única variável aleatória, mas sim na relação entre duas variáveis aleatórias. Enquanto a variância mede o quanto os valores de uma única variável se afastam de sua média, a covariância avalia como duas variáveis aleatórias se movem em conjunto. Assim, a covariância fornece informações sobre como duas variáveis tendem a variar simultaneamente, aumentando ou diminuindo juntas ou se seus comportamentos são independentes;

Desvio padrão: quantifica a dispersão dos valores sobre um conjunto de dados em relação à média, e é calculado como a raiz quadrada da variância, representado por σ;

Distribuição normal: ou distribuição gaussiana, é o valor da média mais ou menos o desvio padrão, descrita por momentos/intervalos de acordo com o desvio padrão;

Distribuição normal multivariada: descreve a distribuição conjunta de múltiplas variáveis aleatórias interdependentes em um espaço multidimensional, representando a incerteza conjunta dessas variáveis. Nesse caso, a incerteza de cada variável é considerada em conjunto com outra variáveis, modelando a propagação da incerteza e considerando suas relações e correlações para inferir como os dados estão distribuídos em torno da média numa distribuição normal para calcular intervalos de confiança em análises estatísticas.

A Figura 3 mostra como duas variáveis, x e y, estão relacionadas de acordo com o intervalo de confiança, a altura, nesse caso. Note que, quando x e y se aproximam de 0, a confiança do modelo é alta, ou seja, o grau que mede quanto a relação de valores das variáveis está próxima da realidade, tende a aumentar. Já quando x e y se afastam de 0, positivamente ou negativamente, a confiança diminui.

Incerteza: mede a variabilidade ou falta de precisão associada a uma distribuição de probabilidade, representada pela dispersão dos dados em relação à média multivariada;

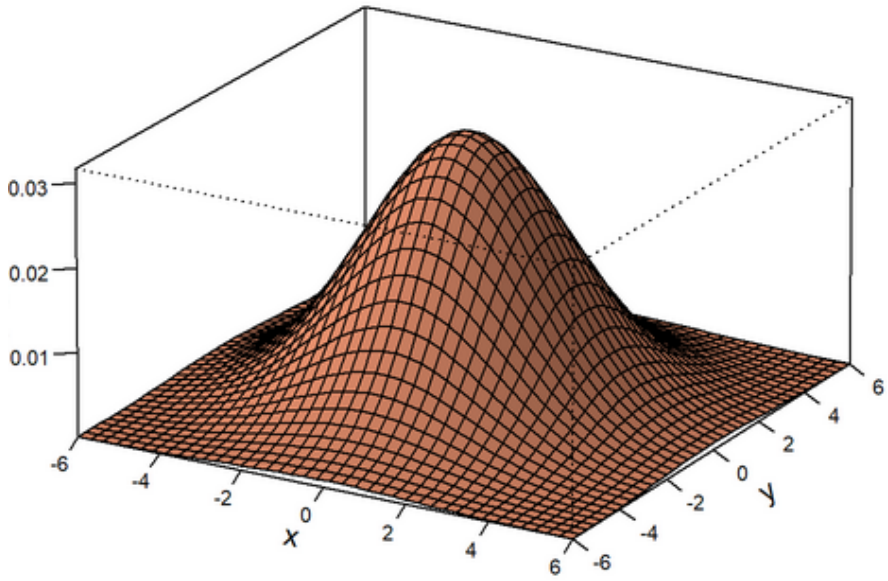

Figura 3 – Representação dos intervalos de confiança sobre uma distribuição normal gaussiana que correlaciona duas variáveis (x e y) [3].

Variável aleatória: definida por uma função que associa valores numéricos possíveis a eventos num espaço amostral de um experimento probabilístico, e pode ser de dois tipo:

- contínua, quando assume qualquer valor dentro de um intervalo dado;
- discreta, quando o número de valores possíveis que a variável assume é finito ou infinito enumerável, ou seja, é possível listar todos os valores possíveis que a variável pode assumir.

Momento: descreve a forma e o comportamento da distribuição num determinado instante, ou seja, a média e distribuição normal num instante k. O momento pode, ainda, ser classificado em dois tipos:

- Momento bruto, calculado diretamente a partir dos dados originais ou da distribuição de probabilidade, dado pelo valor esperado da k-ésima potência da variável aleatória, $E(X^k)$;
- Momento central, calculado em relação à média da distribuição ou conjunto de dados, dado pelo valor esperado da k-ésima potência da distribuição de variáveis aleatórias em torno de sua média, $E((X - \mu_X)^k)$.

Estimativa: aproximação ou cálculo de um valor desconhecido baseado em informações disponíveis, ou seja, avalia o estado "oculto"(não conhecido) do sistema de acordo com dados provenientes de maneira interna ou externa, por exemplo dos sensores;

Acurácia/exatidão: indica o quão próxima a medição está do valor real absoluto;

Peso: importância ou influência de algo (ponderação), usado no cálculo das médias ponderadas como medição, estimativa, representado por w;

Operações de matrizes: englobam a adição e multiplicação de matrizes e vetores, bem como sua transposição, inversão e simetria;

Autovalores: números que representam a escala de estiramento ou encolhimento de uma transformação linear representada por uma matriz;

Autovetores: vetores que não mudam de direção quando multiplicados por uma matriz ou operador linear, apenas modificando-a por um fator escalar (autovalor);

Álgebra de expectativas: refere-se ao cálculo de valores médios ponderados de eventos aleatórios, levando em consideração suas probabilidades;

Translação: movimento de um objeto em linha reta de um ponto a outro, sem alterar sua orientação, ou seja, o movimento retilíneo de um objeto de um lugar para outro.

Rotação: movimento de um objeto em torno de um ponto ou eixo, mantendo sua forma, mas mudando sua orientação;

Orientação: descreve a direção ou ângulo de um objeto em relação a um ponto de referência.

2.2.2 Modelo de Predição Unidimensional

O modelo de predição do filtro de Kalman unidimensional é uma abordagem amplamente utilizada para estimar e prever o estado de sistemas dinâmicos em tempo real, mesmo quando esses sistemas estão sujeitos a ruídos e incertezas. Nessa seção são descritas as equações do modelo que desempenham papéis cruciais na atualização e projeção dos estados do sistema e na gestão da incerteza associada a esses estados. O conjunto de equações do modelo pode ser dividido em equações de: atualização de estado, extrapolação de estado, relação entre variâncias de medição e predição, ganho de Kalman, atualização de covariância e predição de covariância.

As equações de atualização de estado calcula o novo estado estimado do sistema com base nas observações mais recentes e nas estimativas anteriores. O resultado dessas equações refletem como as novas informações observadas ajustam o estado estimado, levando em consideração a incerteza associada às observações. As fórmulas para calcular, respectivamente,

os estados de posição, velocidade e aceleração são dados por

$$\begin{aligned} \hat{x}_{n,n} &= \hat{x}_{n,n-1} + \alpha * (z_n - \hat{x}_{n,n-1}) \\ \hat{\dot{x}}_{n,n} &= \hat{\dot{x}}_{n,n-1} + \beta * \frac{z_n - \hat{x}_{n,n-1}}{\Delta t} \\ \hat{\ddot{x}}_{n,n} &= \hat{\ddot{x}}_{n,n-1} + \gamma * \frac{z_n - \hat{x}_{n,n-1}}{0.5 * \Delta t^2} \end{aligned} \tag{2.1}$$

Onde x representa o real valor do estado (posição) do objeto de estudo, sendo ela uma variável latente, que significa uma variável que não é diretamente observada, não sendo possível obter seu real valor, $\dot{x}$ representa a derivada do deslocamento de uma variável, neste caso de x, sobre o tempo, resultando no valor de velocidade, $\ddot{x}$ representa a derivada dupla do deslocamento de uma variável, neste caso também de x, sobre o tempo ao quadrado, resultando no valor de aceleração, $\hat{x}$ representa o valor estimado de uma variável, ou seja, é o resultado, após uma leitura dos dados z_n, dos tratamentos realizados pelo filtro originando um valor que se espera ser mais próximo do real, z_n representa a medição do estado do objeto de estudo na iteração n pelo aparelho de medição, da mesma forma que $\hat{x}_{n,n}$ representa o valor estimado de x calculado para a iteração atual e realizado na iteração atual, onde os valores n, n representam à qual iteração condiz este resultado e em qual iteração este cálculo foi realizado respectivamente, dessa forma, é correto entender $\hat{x}_{n,n}$ como a estimativa de x no tempo n realizada na iteração n e $\hat{x}_{n,n-1}$ como a predição de x no tempo n realizada na iteração anterior $n-1$.

A variável do tempo é representada por t e a diferença entre o tempo da iteração atual para com o tempo da iteração anterior é representada por Δt. α, β e γ representam, respectivamente, a precisão da medição para posição, velocidade e aceleração, de 0 a 1 em escala quantitativa crescente, indicando o quão próxima a leitura dos dados pode estar do valor real, ou seja, a precisão dos sensores utilizados para medição do estado do objeto de estudo, o cálculo depende de α, β e γ para determinar o quão impactante será cada leitura no decorrer do sistema. Assim, quanto mais precisas são as leituras, ou seja, α, β e γ maiores, maior será o seu impacto na dinâmica do sistema.

As equações de extrapolação de estado são obtidas pela estimação calculada na iteração atual para predizer a posição do sistema na próxima iteração. Essas equações de predição dos novos estados de posição, velocidade e aceleração são representados, respectivamente, por

$$\begin{aligned} \hat{x}_{n+1,n} &= \hat{x}_{n,n} + \hat{\dot{x}}_{n,n} * \Delta t + \hat{\ddot{x}}_{n,n} * (\frac{\Delta t^2}{2}) \\ \hat{\dot{x}}_{n+1,n} &= \hat{\dot{x}}_{n,n} + \hat{\ddot{x}}_{n,n} * \Delta t \\ \hat{\ddot{x}}_{n+1,n} &= \hat{\ddot{x}}_{n,n} \end{aligned} \tag{2.2}$$

Onde aqui é correto entender $\hat{x}_{n+1,n}$ como a predição de x na próxima iteração $n+1$ realizada na iteração atual n.

A relação entre as variâncias de medição e predição é uma medida de incerteza que combina informações provenientes de duas fontes diferentes, as medições reais e as estimativas de estado baseadas no modelo de predição do sistema. O resultado é uma relação entre o quão precisa é uma medição e o quão preciso é o modelo de predição do sistema. Dessa forma, as variâncias de ambos os dados são levadas em conta para determinar a melhor estimativa do estado atual do sistema, representada pela equação

$$p_{n,n} = (w_1)^2 * r_n + (w_2)^2 * p_{n,n-1}$$

onde $p_{n,n}$ representa a covariância da estimativa da iteração no tempo n calculada na iteração atual n, $p_{n,n-1}$ representa a covariância da estimativa da iteração no tempo n calculada na iteração anterior $n-1$, r_n representa a variância da medição no tempo n e w_1 e w_2 são ganhos utilizados para ajustar a influência de diferentes componentes no cálculo da covariância do erro, sendo $w_1 + w_2 = 1$. Os valores de w_1 e w_2 são calculados na equação 2.3.

A equação do ganho de Kalman K_n determina o peso que deve ser atribuído às observações atuais em comparação com as estimativas anteriores ao atualizar o estado. É uma variável que controla a contribuição das estimativas e predições anteriores e das observações mais recentes para calcular a estimativa atual do estado do sistema. Como o objetivo do filtro é achar uma estimativa ótima, deve-se minimizar a sua variância, resultando em

$$\begin{aligned} p_{n,n} &= (w_1)^2 * r_n + (1-w_1)^2 * p_{n,n-1} \Rightarrow \\ \frac{dp_{n,n}}{dw_1} &= 2 * w - 1 * r_n - 2 * (1-w_1) * p_{n,n-1} = 0 \Rightarrow \\ w_1 &= \frac{p_{n,n-1}}{p_{n,n-1} + r_n} = K_n \end{aligned} \tag{2.3}$$

Substituindo na equação de estimação do estado atual obtém-se que

$$\hat{x}_{n,n} = K_n * z_n + (1 - K_n) * \hat{x}_{n,n-1} = \hat{x}_{n,n-1} + K_n * (z_n - \hat{x}_{n,n-1}). \tag{2.4}$$

Observe que o ganho de Kalman é um balanceamento entre variâncias, da predição da estimativa anterior e da medição atual, para minimizar a variância ao calcular a estimativa do estado atual. Esse balanceamento é melhor quando comparado com os filtros α-β-γ, onde as medições são recebidas como parâmetros imutáveis, como por exemplo quando um sensor de distância possui uma precisão de 2 centímetros, indicando que o valor medido pode variar em até 2 centímetros, positivamente ou negativamente.

A atualização do valor de covariância é dada pela equação

$$p_{n,n} = (K_n)^2 * r_n + (1 - K_n)^2 * p_{n,n-1} \tag{2.5}$$

A cada iteração esse valor é atualizado e usado em outras equações, refletindo o grau de confiança que o filtro de Kalman possui sobre as estimativas. A atualização da covariância leva em consideração a precisão das medições e a dinâmica do sistema, ajustando a incerteza conforme novas iterações são realizadas.

Já para encontrar o valor de w_2, dado por $1 - K_n$, temos que

$$1 - K_n = 1 - p_{n,n-1}/(p_{n,n-1} + r_n) = r_n/(p_{n,n-1} + r_n) \tag{2.6}$$

Substituindo, o resultado obtido para $p_{n,n}$ é dado por

$$\begin{gathered} p_{n,n} = (\frac{p_{n,n-1}}{p_{n,n-1} + r_n})^2 * r_n + (\frac{r_n}{p_{n,n-1} + rn})^2 * p_{n,n-1} = \\ \frac{p_{n,n-1} * r_n}{p_{n,n-1}} + r_n * \frac{p_{n,n-1}}{p_{n,n-1} + r_n} + \frac{r_n}{p_{n,n-1} + rn} = (1 - K_n) * p_{n,n-1}. \end{gathered} \tag{2.7}$$

A equação de predição da covariância tem o objetivo de reduzir erros de estimativa ao adicionar incerteza do modelo dinâmico sobre a leitura de dados, ou seja, ruídos que afetam o resultado de uma iteração, levando a uma propagação futura, que não pode ser prevista. Por exemplo, o valor de um resistor pode mudar entre as iterações devido a temperatura do ambiente, logo, a equação representa o quão volátil é uma leitura de fato, como um intervalo de precisão, dado por

$$p_{n+1,n} = p_{n,n} + q_n. \tag{2.8}$$

Sendo q_n a variação de ruído do processo na iteração n associada à incerteza na dinâmica do sistema, representa possíveis flutuações no verdadeiro estado do sistema, causado por ruídos e outras interferências não modeladas.

De forma resumida, as equações do filtro de Kalman são:

1. Atualização de estado

   $\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n * (z_n - \hat{x}_{n,n-1})$

2. Atualização de covariância

   $p_{n,n} = (1 - K_n) * p_{n,n-1}$

3. Ganho de Kalman

   $K_n = \frac{p_{n,n-1}}{p_{n,n-1}+r_n}$

4. Extrapolação de estado

$$\hat{x}_{n+1,n} = \hat{x}_{n,n} + \hat{\dot{x}}_{n,n} * \Delta t + \hat{\ddot{x}}_{n,n} * \frac{\Delta t^2}{2}$$
$$\hat{\dot{x}}_{n+1,n} = \hat{\dot{x}}_{n,n} + \hat{\ddot{x}}_{n,n} * \Delta t$$
$$\hat{\ddot{x}}_{n+1,n} = \hat{\ddot{x}}_{n,n}$$

5. Extrapolação de covariância

$$p^{\dot{x}}_{n+1,n} = p^{\dot{x}}_{n,n} + p^{\ddot{x}}_{n,n} * \Delta t + q_n$$
$$p^{\ddot{x}}_{n+1,n} = p^{\ddot{x}}_{n,n} + q_n$$

O diagrama da Figura 4 ilustra o funcionamento e as etapas do filtro de Kalman.

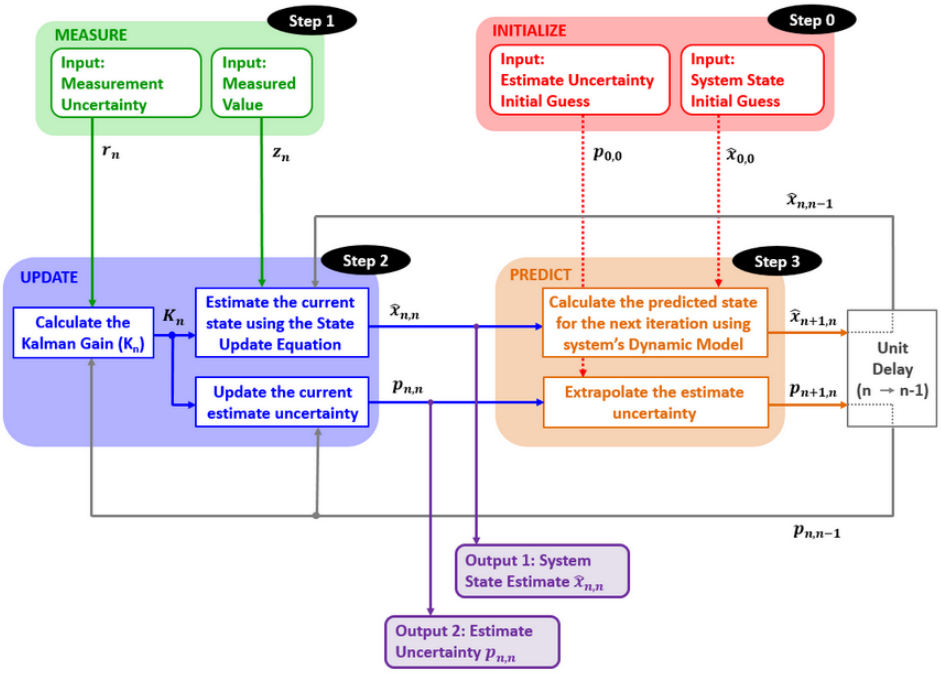

Figura 4 – Etapas do filtro de Kalman [4].

2.2.3 Aplicação do Filtro de Kalman Unidimensional

Esta subseção apresenta um exemplo prático para ilustrar a aplicação do filtro de Kalman unidimensional utilizando o primeiro método explicado em 2.1.2. Suponha um cenário simples onde se deseja estimar a distância de um carro em relação a um radar de velocidade. O carro se move em linha reta e já ultrapassou o radar, porém não se sabe sua velocidade. O sensor que mede a distância entre o centro do carro e o radar z_n realiza uma leitura a cada segundo, e a cada leitura é realizada uma iteração do filtro. Note que algumas medições podem conter ruído/erro aleatório, como pode ser observado na Figura 5. O objetivo é predizer

a localização do carro, da maneira mais precisa possível, utilizando o filtro de Kalman unidimensional, ao invés de apenas obter os dados de leitura do radar. Este exemplo apresenta uma versão mais simplificada do filtro de Kalman por se tratar de apenas uma direção e sentido, com velocidade constante e aceleração nula, para o efeito de apresentar de forma mais didática o funcionamento do filtro, mais à frente 2.3.2 será demonstrado um outro exemplo de como se utilizar o filtro de Kalman de maneira que possa ser implementado como solução em um ambiente real.

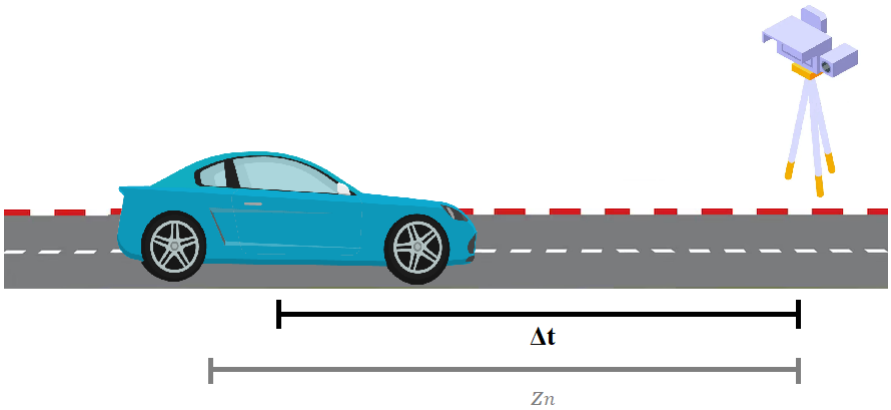

Figura 5 – Medição realizada (z_n) para com o posicionamento real do carro (Δt).

Iteração 0 (Inicialização)

Alguns parâmetros inciais devem ser definidos para a execução do algoritmo de predição e estimação do filtro de Kalman. Primeiramente, a distância entre carro e radar ($\hat{x}_{0,0}$) é definida, de forma aleatória, em 10 metros. Como o palpite é aleatório e impreciso, os parâmetros são inicializados da seguinte forma: (i) a covariância $p_{n,n}$ é iniciada com um valor alto, pois não se sabe de fato o estado atual do carro no primeiro instante. Logo, $p_{0,0} = 10.000$, indicando uma alta imprecisão no palpite inicial; (ii) a incerteza q, que varia de 0 a 1, é inicializada com $q = 0,15$, um valor relativamente alto se comparado com outros modelos quando se tem alguma base para o chute inicial; e (iii) o baixo ruído de medição $r = 0,01$, que representa a porcentagem de quão distante estão as medições em relação aos valores reais, indica que as medições são mais precisas e confiáveis.

Vale também ressaltar que a incerteza q pode ser inicializada com um valor muito alto, por exemplo, 0,9. A diferença está no número de iterações necessárias para que o modelo se alinhe com as medições e a posição real do objeto de estudo [31]. Além disso, a velocidade não é definida no modelo do exemplo não fornecendo nenhum embasamento para o palpite inicial. Já o parâmetro de ruído é, principalmente, impactado pela precisão do equipamento

utilizado. Caso o equipamento seja muito ruidoso o modelo terá, consequentemente, uma menor precisão.

A predição da posição na próxima iteração assume o valor da estimativa do estado atual, já que existe apenas um valor de medição (posição) para se usar de referência no sistema descrito. Portanto, a predição da posição do estado do sistema na próxima iteração é obtida pela estimativa realizada (chute inicial) na iteração atual. Logo, temos que $\hat{x}_{1,0} = \hat{x}_{0,0}$. A predição da variância da próxima iteração é obtida pela estimativa da variância do estado atual. Assim obtemos, $p_{1,0} = p_{0,0} + q = 10.000,15$.

Iteração 1

Nessa iteração o algoritmo usa a predição feita na iteração anterior sobre o estado $x_{1,0}$ para estimar o estado atual $x_{1,1}$ e prever a próxima posição do carro na estrada (próximo estado) $x_{2,1}$.

- Medição (z_n) = 49,732 m
- Estimativas do estado atual:

  $K_1 = \frac{10.000,15}{10.000,15+0.01} = 0,999999$

  A porcentagem alta no ganho de Kalman nessa iteração indica alta confiabilidade nos dados medidos e baixa confiabilidade nos dados estimados/preditos.

  $\hat{x}_{1,1} = 10 + 0,999999 * (49,732 - 10) = 49,732m$

  O cálculo da estimativa do estado atual $\hat{x}_{1,1}$ resultou no valor igual ao medido devido a porcentagem do ganho de Kalman estar muito alto, próximo de 1, priorizando o valor medido ao invés do estimado.

  $p_{1,1} = (1 - 0,999999) * 10.000,15 = 0,01.$

  A covariância do estado atual $p_{1,1}$ resultou num valor muito menor que da predição na iteração anterior $p_{1,0}$, pois a estimativa de $\hat{x}_{1,0}$ era bastante imprecisa, conforme visto no chute inicial dado. Como o valor de $\hat{x}_{1,1}$ é agora mais preciso, pois está próximo do valor da medição atual z_n, o valor da covariância de estado atual $p_{1,1}$ é, consequentemente, muito baixo.

- Predição do próximo estado:

  $\hat{x}_{2,1} = \hat{x}_{1,1} = 49,732m$

  A extrapolação de estado $\hat{x}_{2,1}$ é obtida pela forma reduzida, já que o exemplo considera apenas o parâmetro de posição.

$p_{2,1} = 0,01 + 0,15 = 0,16$

A extrapolação de covariância $p_{2,1}$ também é obtida pela forma reduzida, usando a equação 2.8.

Iteração 2

Na iteração 2 o processo se repete para estimar a posição do estado atual com base na predição realizada na iteração 1, bem como realizar a predição do próximo estado usando a medição atual. Nas próximas iterações o processo continua com a realização das medições e estimativas.

- Medição (z_n) = 59,781 m
- Estimativas do estado atual:

  $K_2 = \frac{0,16}{0,16+0,01} = 0,9412$

  A porcentagem alta no ganho de Kalman nessa iteração também indica alta confiabilidade nos dados medidos, já que os parâmetros do filtro ainda não foram devidamente ajustados e o erro de medição q_n é baixo.

  $\hat{x}_{2,2} = 49,732 + 0,9412 * (59,781 - 49,732) = 59,190m$

  O cálculo da estimativa do estado atual $\hat{x}_{1,1}$ resultou em um valor próximo ao medido porém não exatamente igual, ou seja, as estimativas do filtro nesta iteração já começam a se aproximar mais do real em comparação com o medido z_n, ainda que priorize o valor medido ao invés do estimado.

  $p_{2,2} = (1 - 0,9412) * 0,16 = 0,0094$

  A covariância do estado atual $p_{2,2}$ resultou em um valor baixo, assim como na predição da iteração anterior $p_{2,1}$, pois agora o valor estimado em $\hat{x}_{1,1}$ é considerado preciso seguindo de acordo com a medição atual z_n.

- Predição do próximo estado:

  $\hat{x}_{3,2} = \hat{x}_{2,2} = 59,190m$

  A extrapolação de estado $\hat{x}_{2,1}$ continua considerando apenas o parâmetro de posição estimada atual para calcular sua predição.

  $p_{3,2} = 0,0094 + 0,15 = 0,1594$

  A extrapolação de covariância $p_{2,1}$ também é obtida pela forma reduzida, usando a equação 2.8.

Iteração 3

- Medição (z_n) = 68,545 m
- Estimativas do estado atual:

  $K_3 = \frac{0,1594}{0,1594+0,01} = 0,941$

  $\hat{x}_{3,3} = 59,190 + 0,941 * (68,545 - 59,190) = 67,993m$

  $p_{3,3} = (1 - 0,941) * 0,1594 = 0,0094$

- Predição do próximo estado:

  $\hat{x}_{4,3} = \hat{x}_{3,3} = 67,993m$

  $p_{4,3} = 0,0094 + 0,15 = 0,1594$

  ⋮

Iteração 12

- Medição (z_n) = 161,131 m
- Estimativas do estado atual:

  $K_{12} = \frac{0,1594}{0,1594+0,01} = 0,941$

  $\hat{x}_{12,12} = 150,170 + 0,941 * (161,131 - 150,170) = 160,484m$

  $p_{12,12} = (1 - 0,941) * 0,1594 = 0,0094$

- Predição:

  $\hat{x}_{13,12} = \hat{x}_{12,12} = 160,484m$

  $p_{13,12} = 0,0094 + 0,15 = 0,1594$

Após doze iterações o valor da predição calculado pelo algoritmo se torna muito próximo do valor real medido, conforme ilustra a Figura 6. No gráfico é possível observar a convergência dos parâmetros e a proximidade dos valores derivados através dos cálculos obtidos pelo filtro.

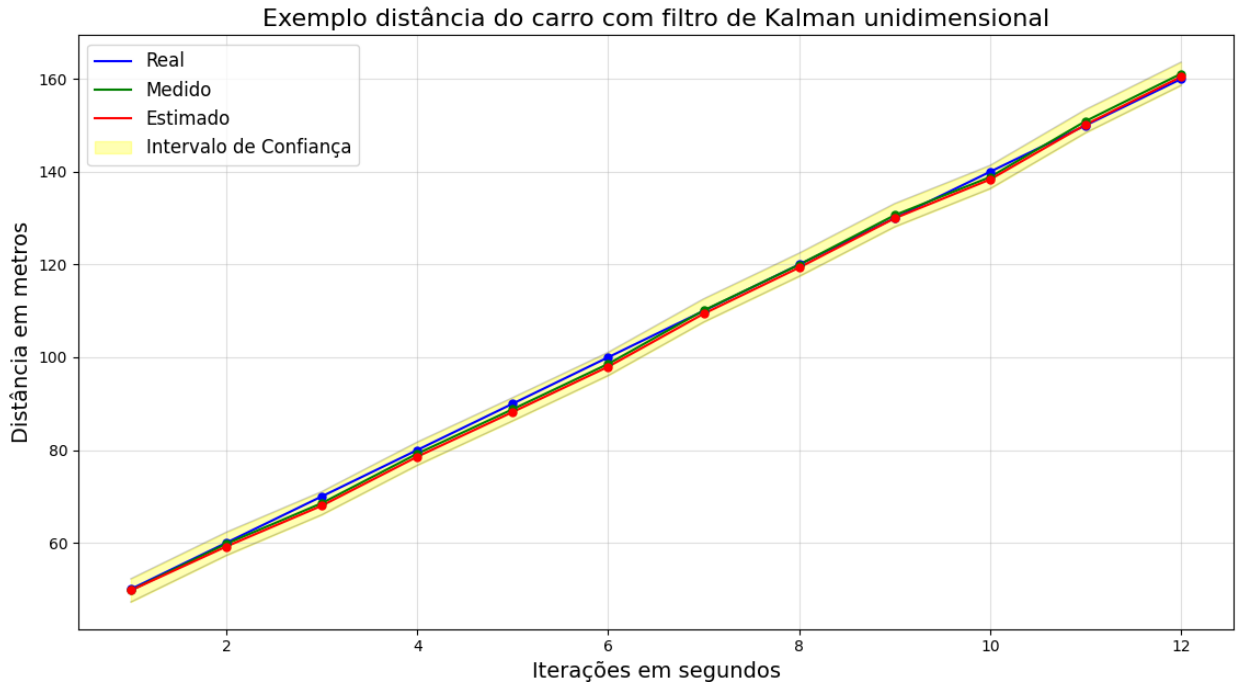

Figura 6 – Exemplo prático do filtro de Kalman unidimensional

Resultados

A Figura 6 apresenta os resultados da aplicação do filtro de Kalman no cenário descrito. O gráfico mostra que os dados estimados tendem a permanecer próximos dos valores medidos. Observe que, apesar da leitura real dos dados possuir uma precisão de 1% de erro ($r = 0,01$), o resultado da estimativa do filtro de Kalman é, ainda, mais preciso do que essa medição, resultando num valor mais próximo do estado real do objeto de estudo. Esse fato tende a ser mais aparente quando a precisão do sensor diminui.

2.3 O Sistema SLAM Multivariado

O filtro de Kalman multivariado estende o modelo unidimensional para resolver problemas em SLAM que possuam duas ou mais dimensões. Em ambientes multidimensionais, vetores e matrizes são utilizados para representar as dimensões do ambiente analisado. Num ambiente tridimensional, por exemplo, o vetor de estado $[x, y, z]$ descreve a posição de um objeto no espaço 3D. Assim, o vetor de estado que descreve a posição e velocidade de um objeto no espaço é dado por $[x, y, z, \dot{x}, \dot{y}, \dot{z}]$, enquanto que o vetor de estado que descreve a posição, velocidade e aceleração de um objeto no espaço é representado por $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}]$.

O objetivo é então adaptar o filtro de Kalman, originalmente desenvolvido para sistemas unidimensionais, de modo a lidar com sistemas mais complexos que operam num espaço

multidimensional. Neste cenário, o modelo adaptado deve ser capaz de estimar e prever estados do sistema utilizando das mesmas fórmulas e teorias já apresentadas anteriormente no modelo unidimensional, de acordo com a quantidade de dimensões. Assim, as equações do modelo unidimensional, apresentadas na Subseção 2.2.2, devem ser adaptadas para um modelo multivariado.

Os princípios fundamentais do filtro de Kalman, tais como a atualização do estado com base em observações, a atualização da covariância e a previsão do próximo estado, permanecem os mesmos, no entanto, a representação de matrizes é introduzida para que diversas dimensões sejam modeladas. As equações são então aplicadas sobre as matrizes que modelam o espaço multidimensional, correlacionando suas variáveis num único modelo e relacionando suas derivações e variâncias. As equações são calculadas, basicamente, da mesma maneira que no modelo unidimensional, porém uma vez para cada plano bidimensional e interligadas de acordo com o cálculo da covariância e da distribuição normal multivariada.

Na adaptação do cálculo da covariância (Equação 2.8), por exemplo, as duas principais mudanças são: (a) a inserção da matriz de covariância, que captura as relações entre todas as variáveis de estado, ou seja, considera covariâncias cruzadas entre as variáveis, refletindo como essas covariâncias influenciam umas às outras; e (b) a utilização de modelagem multivariada, que envolve a consideração das influências mútuas e das interações entre as variáveis de estado em sistemas multidimensionais. Assim, o filtro de Kalman leva em consideração como as variáveis de estado estão relacionadas e como suas mudanças afetam umas às outras. A principal vantagem desse modelo é sua capacidade de lidar com sistemas mais complexos, onde as variáveis de estado estão interligadas e se influenciam. Essas e outras mudanças necessárias nas formulações das equações são abordadas na Subseção 2.3.1.

Nas próximas subseções são apresentados o modelo de predição bidimensional, juntamente com suas equações adaptadas, e uma transformação de um modelo tridimensional em bidimensional, para que este último seja aplicado em ambientes reais tridimensionais.

2.3.1 Modelo de Predição Bidimensional

O modelo de predição bidimensional é uma instanciação do modelo multivariado, extensível para qualquer dimensão. Da mesma forma que o modelo unidimensional, o modelo bidimensional segue as cinco principais das equações do filtro de Kalman unidimensional. Esta seção mostra como o modelo é definido de acordo com as medições sobre cada dimensão e como relacioná-las.

No modelo bidimensional temos duas variáveis, x e y, e os parâmetros associados de posição, velocidade e aceleração representados numa única matriz da forma $\vec{w} = [x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}]$,

onde $\vec{w}$ representa que w está em formato matricial. Assim, o cálculo da extrapolação de estado (Equação 2.2) adaptada para o ambiente bidimensional é dado pela Equação 2.9.

$$\vec{\hat{w}}_{n+1,n} = \vec{F} * \vec{\hat{w}}_{n,n} + \vec{Q} \tag{2.9}$$

Note que a predição da próxima iteração é obtida pela multiplicação da matriz $\vec{F}$ de transição de estado [4] pelo vetor de estimativas do estado atual do sistema e somada com a matriz $\vec{Q}$ de variâncias do objeto de estudo.

$$\begin{bmatrix} \hat{x}_{n+1,n} \\ \hat{y}_{n+1,n} \\ \hat{\dot{x}}_{n+1,n} \\ \hat{\dot{y}}_{n+1,n} \\ \hat{\ddot{x}}_{n+1,n} \\ \hat{\ddot{y}}_{n+1,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0.5\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0.5\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \hat{x}_{n,n} \\ \hat{y}_{n,n} \\ \hat{\dot{x}}_{n,n} \\ \hat{\dot{y}}_{n,n} \\ \hat{\ddot{x}}_{n,n} \\ \hat{\ddot{y}}_{n,n} \end{bmatrix} + \begin{bmatrix} q_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_6 \end{bmatrix}$$

Logo, a predição da próxima iteração, $\vec{\hat{w}}_{n+1,n}$, usando as estimativas do estado atual do sistema, $\vec{\hat{w}}_{n,n}$, é obtida pelo sistema de equações dado abaixo, com base na Equação 2.9.

$$\vec{\hat{w}}_{n+1,n} = \begin{bmatrix} \hat{x}_{n+1,n} \\ \hat{y}_{n+1,n} \\ \hat{\dot{x}}_{n+1,n} \\ \hat{\dot{y}}_{n+1,n} \\ \hat{\ddot{x}}_{n+1,n} \\ \hat{\ddot{y}}_{n+1,n} \end{bmatrix} = \begin{cases} \hat{x}_{n,n} + \hat{\dot{x}}_{n,n}\Delta t + \frac{1}{2}\hat{\ddot{x}}_{n,n}\Delta t^2 + q_1 \\ \hat{y}_{n,n} + \hat{\dot{y}}_{n,n}\Delta t + \frac{1}{2}\hat{\ddot{y}}_{n,n}\Delta t^2 + q_2 \\ \hat{\dot{x}}_{n,n} + \hat{\ddot{x}}_{n,n}\Delta t + q_3 \\ \hat{\dot{y}}_{n,n} + \hat{\ddot{y}}_{n,n}\Delta t + q_4 \\ \hat{\ddot{x}}_{n,n} + q_5 \\ \hat{\ddot{y}}_{n,n} + q_6 \end{cases}$$

A matriz de transição de estado, $\vec{F}$, o vetor de estimativas do estado atual, $\vec{\hat{w}}_{n,n}$, e a matriz de covariância de ruído do processo, $\vec{Q}$, são dados como segue.

$$\vec{F} = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \Delta 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \vec{\hat{w}}_{n,n} = \begin{bmatrix} \hat{x}_{n,n} \\ \hat{y}_{n,n} \\ \hat{\dot{x}}_{n,n} \\ \hat{\dot{y}}_{n,n} \\ \hat{\ddot{x}}_{n,n} \\ \hat{\ddot{y}}_{n,n} \end{bmatrix} \quad \vec{Q} = \begin{bmatrix} q_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_6 \end{bmatrix}$$

Vale ressaltar que os valores $q_1, q_2, \ldots$ da diagonal principal da matriz $\vec{Q}$ são as variâncias de cada variável de estado do objeto de estudo (posição, velocidade, etc) em um exemplo onde ruído de processo é independente, ou seja, sem relação ou correlação entre as

variáveis de estado. Cada elemento do vetor resultante da extrapolação de estado é então somado com seu respectivo valor de ruído da medição.

Porém, assumindo um modelo discreto onde o ruído é volátil e, portanto, varia de um instante para outro de forma constante, a matriz $\vec{Q}$ de um modelo bidimensional pode ser simplificada. Essa simplificação é obtida pela propriedade de simetria da covariância $COV(x, \dot{x}) = COV(\dot{x}, x)$ e assumindo um exemplo em que as variáveis de estado estão correlacionadas, então, a matriz $\vec{Q}$ é dada como segue.

$$\vec{Q} = \begin{bmatrix} V(x) & COV(x, \dot{x}) \\ COV(\dot{x}, x) & V(\dot{x}) \end{bmatrix}$$

Além disso, como a variância é definida pela média dos quadrados das diferenças entre os valores da variável e sua média, e todo o valor esperado de uma constante não muda, a média se mantém constante, ou seja, a média permanece a mesma, independente das probabilidades e variâncias. Assim, seja z uma variável qualquer, sua variância é obtida do valor esperado E e pela média de z e μ_z.

$$\begin{aligned} V(z) = E((z-\mu_z)^2) = E(z^2 - 2z\mu_z + \mu_z^2) = E(z^2) + E(-2z\mu_z) + E(\mu_z^2) \\ E(-2z\mu_z) = -2\mu_z * E(z) = -2\mu_z^2 \\ E(\mu_z^2) = \mu_z^2 \\ V(z) = E(z^2) - \mu_z^2 \end{aligned} \tag{2.10}$$

O valor esperado de uma variável A é calculado pela fórmula

$$E(A) = \sum_{i=1}^{\infty} a_i P(a_i),$$

onde a_i é um valor específico que A pode assumir e $P(a_i)$ é probabilidade de A assumir o valor a_i. Os valores das variâncias de posição (x) e velocidade $(\dot{x})$ são obtidos pelas Equações 2.11 e 2.12, respectivamente, e o valor da covariância é obtido pela Equação 2.13.

$$\begin{aligned} V(x) = E(x^2) - \mu_x^2 = E((\frac{1}{2}\ddot{x}\Delta t^2)^2) - (\frac{1}{2}\mu_{\ddot{x}}\Delta t^2) = \\ \frac{\Delta t^2}{4}(E(\ddot{x}^2) - \mu_{\ddot{x}}^2) = \frac{\Delta t^2}{4}\sigma_{\ddot{x}}^2 \end{aligned} \tag{2.11}$$

$$\begin{aligned} V(\dot{x}) = E(\dot{x}^2) - \mu_{\dot{x}}^2 = E(\dot{x}^2) - \mu_{\dot{x}}^2 = \\ E((\ddot{x}\Delta t)^2) - (\mu_{\ddot{x}}\Delta t)^2 = \Delta t^2(E(\ddot{x}^2) - \mu_{\ddot{x}}^2) = \Delta t^2\sigma_{\ddot{x}}^2 \end{aligned} \tag{2.12}$$

$$\begin{aligned} COV(x, v) = COV(v, x) = E(xv) - \mu_x\mu_v = \\ E(\frac{1}{2}\ddot{x}\Delta t^2\ddot{x}\Delta t) - (\frac{1}{2}\mu_{\ddot{x}}\Delta t^2\mu_{\ddot{x}}\Delta t) = \frac{\Delta t^3}{2}(E(\ddot{x}^2) - \mu_{\ddot{x}}^2) = \frac{\Delta t^3}{2}\sigma_{\ddot{x}}^2 \end{aligned} \tag{2.13}$$

Portanto, a matriz da covariância de ruído $\vec{Q}$ do processo é definida por

$$\vec{Q} = \begin{bmatrix} \frac{\Delta t^2}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix} * \sigma_{\ddot{x}}^2,$$

onde $\sigma_{\ddot{x}}^2$ é a variância de aceleração, determinada com base no conhecimento do sistema dinâmico modelado, dados experimentais ou observações do sistema. Nota-se que tanto a variância de posição e velocidade quanto a covariância entre esses dois parâmetros são obtidas em função da variância de aceleração, através de derivações da álgebra de probabilidade [4].

No cálculo da extrapolação de covariância para o modelo bidimensional é necessário adaptar a Equação 2.8 do modelo unidimensional. Essa adaptação depende da matriz de ruído $\vec{Q}$ e da matriz de transição de estado $\vec{F}$. O resultado é a Equação 2.14 que retorna a matriz de covariância da incerteza quadrática de uma predição (próximo estado) $\vec{P}_{n+1,n}$, usando a matriz de covariância da incerteza quadrática (variância) de uma estimativa (estado atual) $\vec{P}_{n,n}$, a matriz de transição de estado $\vec{F}$ e sua transposta $\vec{F}^T$, e a matriz de covariância de ruído do processo $\vec{Q}$.

$$\vec{P}_{n+1,n} = \vec{F} * \vec{P}_{n,n} * \vec{F}^T + \vec{Q} \tag{2.14}$$

A covariância da incerteza quadrática de uma estimativa (estado atual) é dada pela matriz $\vec{P}_{n,n}$.

$$\vec{P}_{n,n} = \begin{bmatrix} p_x & p_{x\dot{x}} & p_{x\ddot{x}} & 0 & 0 & 0 \\ p_{\dot{x}x} & p_{\dot{x}} & p_{\dot{x}\ddot{x}} & 0 & 0 & 0 \\ p_{\ddot{x}x} & p_{\ddot{x}\dot{x}} & p_{\ddot{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & p_y & p_{y\dot{y}} & p_{y\ddot{y}} \\ 0 & 0 & 0 & p_{y\dot{y}} & p_{\dot{y}} & p_{\dot{y}\ddot{y}} \\ 0 & 0 & 0 & p_{y\ddot{y}} & p_{\dot{y}\ddot{y}} & p_{\ddot{y}} \end{bmatrix}$$

A diagonal principal da matriz representa as variâncias de posição, velocidade e aceleração estimadas, de cada variável, como por exemplo $p_x, p_{\dot{x}}$ e $p_{\ddot{x}}$, respectivamente, para a variável x. O mesmo ocorre para a variável y. Já os elementos fora da diagonal principal são as covariâncias, obtidas com o filtro de Kalman ao longo das iterações. O exemplo prático da Subsecção 2.3.2 mostra que a matriz $\vec{P}_{n,n}$ é inicializada apenas com os elementos da diagonal principal, enquanto o restante dos elementos são iniciados com o valor 0. Conforme as iterações do filtro são executadas, os valores das covariâncias são gerados. Esses valores das covariâncias representam a relação entre as variáveis de estado, portanto, assim como $COV(x, \dot{x}) = COV(\dot{x}, x)$, $p_{x\dot{x}} = p_{\dot{x}x}$, $p_{\dot{x}\ddot{x}} = p_{\ddot{x}\dot{x}}$, etc, na matriz $\vec{P}_{n,n}$. Logo, a matriz resultante

$\vec{P}_{n+1,n}$ do próximo estado é obtida como segue.

$$\vec{P}_{n+1,n} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0.5\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0.5\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x & p_{x\dot{x}} & p_{x\ddot{x}} & 0 & 0 & 0 \\ p_{\dot{x}x} & p_{\dot{x}} & p_{\dot{x}\ddot{x}} & 0 & 0 & 0 \\ p_{\ddot{x}x} & p_{\ddot{x}\dot{x}} & p_{\ddot{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & p_y & p_{y\dot{y}} & p_{y\ddot{y}} \\ 0 & 0 & 0 & p_{y\dot{y}} & p_{\dot{y}} & p_{\dot{y}\ddot{y}} \\ 0 & 0 & 0 & p_{y\ddot{y}} & p_{\dot{y}\ddot{y}} & p_{\ddot{y}} \end{bmatrix}$$

$$* \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \Delta t & 0 & 1 & 0 & 0 & 0 \\ 0 & \Delta t & 0 & 1 & 0 & 0 \\ 0.5\Delta t^2 & \Delta t & 0 & 0 & 1 & 0 \\ 0 & 0.5\Delta t^2 & 0 & \Delta t & 0 & 1 \end{bmatrix} + \begin{bmatrix} q_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_6 \end{bmatrix}$$

Os valores das variâncias na diagonal principal da matriz $\vec{P}_{n+1,n}$ são, geralmente, inicializados com valores altos em comparação com o erro da medição, por se tratar de um palpite na inicialização do filtro.

Já a multiplicação pela matriz $\vec{F}^T$, a transposta de $\vec{F}$, garante a coerência matemática do filtro, ou seja, mantém a orientação e o sentido do objeto de estudo representado na forma matricial. Vale lembrar que a matriz $\vec{F}$ de transição é necessária para se obter o sistema de equações que resulta no cálculo da extrapolação de estado. A matriz $\vec{F}^T$ também reflete o relacionamento entre as variáveis de estado em diferentes instantes de tempo. Na equação de covariância, por exemplo, as variáveis x e y estão inversamente relacionadas por uma multiplicação. Logo, o aumento da variável x implica na diminuição de y para que a proporção entre tais variáveis se mantenha. Já numa divisão, a relação se mantém diretamente proporcional entre as variáveis, pois o aumento de x também implica no aumento de y. Observe também que as matrizes de covariância são sensíveis à orientação e escala dos sistemas de coordenadas em que estão definidas. Por isso, a multiplicação pela matriz transposta permite uma transformação apropriada da matriz de covariância de incerteza do estado atual para o próximo estado.

Assim a simulação de movimento do objeto de estudo no ambiente se dá, a cada iteração do filtro, através de uma rotação, um giro do objeto sobre seu centro de acordo com um certo ângulo, e uma translação, o movimento do objeto de um ponto a outro mantendo sua orientação. Esses movimentos de rotação e translação são realizados sobre o mapa do ambiente em relação a uma leitura anterior e a atual, enquanto que a multiplicação pela matriz $\vec{F}^T$ resulta numa matriz de rotação e translação 2D, conforme os cálculos obtidos da

extrapolação de covariância entre as iterações. A Subseção 2.3.2 apresenta um exemplo que ilustra o papel dessas multiplicações de forma prática.

Já a equação de atualização de estado do modelo é adaptada para o sistema bidimensional com a inserção da matriz de observação $\vec{H}$. Essa matriz tem a função de converter o estado do sistema (input) em medições (output) usando transformações lineares. O cálculo de atualização de estado (Equação 2.4) é obtido para o modelo bidimensional através da inclusão da matriz de observação na fórmula. A atualização de estado $\vec{\hat{w}}_{n,n}$ no modelo bidimensional é então definido pela Equação 2.15.

$$\vec{\hat{w}}_{n,n} = \vec{\hat{w}}_{n,n-1} + \vec{K}_n * (\vec{z}_n - \vec{H} * \vec{\hat{w}}_{n,n-1}) \tag{2.15}$$

A matriz z_n representa a medição do sensor sobre o ambiente, retornando sua posição medida na iteração atual sobre cada dimensão. No caso de um sensor que mede apenas posição, a matriz $\vec{z_n}$ possui um tamanho 2×1, com os valores da posição nas coordenadas x e y, respectivamente.

Já $\vec{\hat{w}}_{n,n-1}$, de dimensões 6×1, representa os valores das variáveis de estado preditos na iteração anterior relativos a cada dimensão, posição, velocidade e aceleração em x e y, como descrito na Equação 2.9. Cada valor da matriz $\vec{K}_n$, do ganho de Kalman no modelo bidimensional, representa o valor do ganho de Kalman resultante da associação de uma subtração e uma multiplicação dos valores de $\vec{\hat{w}}_{n,n-1}$ e $\vec{z_n}$. A matriz $\vec{K}_n$ é então multiplicada por $\vec{z}_n - \vec{H} * \vec{\hat{w}}_{n,n-1}$, resultando numa 6×1, e em seguida somada com $\vec{\hat{w}}_{n,n-1}$, de dimensões 6×1.

A matriz de observação $\vec{H}$, de dimensões $n \times m$, representa os valores mensuráveis do ambiente, onde n é o número de variáveis de estado mensuráveis do sistema, de acordo com $\vec{z_n}$, e m é o número total de variáveis de estado, de acordo com $\vec{\hat{w}}_{n,n-1}$. No caso do exemplo anterior a matriz $\vec{H}$ tem dimensões 2×6. Como o vetor de estado $\vec{\hat{w}}_{n,n-1}$ tem seis variáveis de estado, mas apenas duas são consideradas mesuráveis no vetor de medições $\vec{z_n}$ (a primeira, posição em x, e a quarta, posição em y), a matriz de observação $\vec{H}$ tem um tamanho de dimensões mensuráveis em $\vec{z_n} \times \vec{\hat{w}}_n$ variáveis de estado em $\vec{\hat{w}}_{n,n-1}$, resultando numa matriz 2×6.

$$\vec{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Cada dimensão mensurável tem o valor 1 no elemento que representa seu valor (posição, velocidade e aceleração neste ordem para cada variável), dessa forma, em $H_{[0,0]}$ e $H_{[1,3]}$. O cálculo de $(\vec{z_n} - \vec{H} * \vec{\hat{w}}_{n,n-1})$, numa nova iteração nas etapas do filtro, adiciona informação apenas das dimensões que de fato são mensuradas por um sensor no ambiente. Assim, a

matriz resultante deste cálculo se reduz a

$$(\vec{z_n} - \vec{H} * \vec{\hat{w}}_{n,n-1}) = \begin{bmatrix} (z_{[0,0]} - \hat{w}_{[0,0]}) \\ (z_{[1,3]} - \hat{w}_{[1,3]}) \end{bmatrix}$$

Logo, a função básica da matriz de observação é excluir dimensões não mensuráveis nas iterações.

A equação de atualização de covariância e sua derivação vêm de duas outras equações, da atualização de estado (Equação 2.1), e da covariância estimada (Equação 2.5). A atualização de estado do modelo bidimensional é então definida pela Equação 2.16 através da matriz de observação ($\vec{H}$) e da matriz de erro da medição $\vec{v}_n$, que possui a mesma dimensão da matriz $\vec{Q}$ na Equação 2.9, e que representa o quão distante, positivamente ou negativamente, a medição está do valor real.

$$\vec{\hat{w}}_{n,n} = \vec{\hat{w}}_{n,n-1} + \vec{K}_n * (\vec{H}\vec{w}_n + \vec{v}_n - \vec{H} * \vec{\hat{w}}_{n,n-1}) \tag{2.16}$$

Note que a matriz $\vec{w}_n$ na fórmula, de dimensões iguais à $\vec{\hat{w}}_{n,n}$, representa o estado real do sistema, ou seja, onde de fato o objeto de estudo se encontra na realidade, assim como $\vec{v}_n$, também de dimensões iguais à $\vec{\hat{w}}_{n,n}$, representa o erro de medição do sensor em relação ao valor real. Os parâmetros $\vec{w}_n$ e $\vec{v}_n$ normalmente não aparecem nos cálculos [4], uma vez que são desconhecidos numa aplicação real, com exceção dos exemplos ilustrativos, onde são usados para fins de verificação dos valores estimados.

O erro de estimativa é dado pela fórmula $\vec{e_n} = \vec{w}_n - \vec{\hat{w}}_{n,n}$. Usando a Equação 2.16 obtemos a seguinte derivação.

$$\begin{aligned} \vec{e}_n = \vec{w}_n - \vec{\hat{w}}_{n,n} = \vec{w}_n - \vec{\hat{w}}_{n,n-1} - \vec{K}_n * (\vec{H}\vec{w}_n + \vec{v}_n - \vec{H} * \vec{\hat{w}}_{n,n-1}) = \\ \vec{w}_n - \vec{\hat{w}}_{n,n-1} - \vec{K}_n * \vec{H} * (\vec{w}_n - \vec{\hat{w}}_{n,n-1}) - \vec{K}_n * \vec{v}_n = \\ (\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{w}_{n,n-1}) - \vec{K}_n\vec{v}_n \end{aligned}$$

Logo, o erro de estimativa $\vec{e}_n$ se reduz a Equação 2.17.

$$\vec{e}_n = (\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{w}_{n,n-1}) - \vec{K}_n\vec{v}_n \tag{2.17}$$

Note que a matriz identidade, $\vec{I}$, de dimensões iguais as de $\vec{P}_{n,n}$, tem a função de não descartar os dados mensurados em $\vec{w}_n - \vec{w}_{n,n-1}$. Como a multiplicação pela matriz $\vec{H}$ ignora determinadas dimensões, a subtração de $\vec{I}$ por $\vec{K}_n\vec{H}$ impede que as medições sejam desconsideradas, obtendo valores maiores que 0 na diagonal principal.

Na sequência, a covariância estimada é aplicada para se obter a covariância $\vec{P}_{n,n}$ usando o erro estimativa.

$$\vec{P}_{n,n} = E(e_n * e_n^T) =$$
$$E(((\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{\hat{w}}_{n,n-1}) - \vec{K}_n\vec{v}_n)((\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{w}_{n,n-1}) - (\vec{K}_n\vec{v}_n)^T))$$
$$= E(((\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{\hat{w}}_{n,n-1}) - \vec{K}_n\vec{v}_n)((\vec{w}_n - \vec{\hat{w}}_{n,n-1})^T(\vec{I} - \vec{K}_n\vec{H})^T - (\vec{K}_n\vec{v}_n)^T))$$
$$= (\vec{I} - \vec{K}_n\vec{H})\underbrace{E((\vec{w}_n - \vec{\hat{w}}_{n,n-1})(\vec{w}_n - \vec{\hat{w}}_{n,n-1})^T)}_{1}(\vec{I} - \vec{K}_n\vec{H})^T$$
$$- \underbrace{E((\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{\hat{w}}_{n,n-1})(\vec{K}_n\vec{v}_n)^T)}_{2} - \underbrace{E(\vec{K}_n\vec{v}_n(\vec{w}_n - \vec{\hat{w}}_{n,n-1})^T(\vec{I} - \vec{K}_n\vec{H})^T)}_{3}$$
$$+ \vec{K}_n * \underbrace{E(\vec{v}_n\vec{v}_n^T)}_{4} * \vec{K}_n^T$$

Como $\vec{w}_n - \vec{\hat{w}}_{n,n-1}$ é o erro da previsão de uma iteração prévia e não está relacionado com o ruído da medição atual, o valor esperado do produto de duas variáveis não relacionadas, tais como $E(\vec{w}_n - \vec{\hat{w}}_{n,n-1})$ e $E((\vec{w}_n - \vec{\hat{w}}_{n,n-1})^T)$, é 0. Portanto, as derivações a partir de 1, 2, 3 e 4, do cálculo de $\vec{P}_{n,n}$, são apresentados a seguir.

$$1\left\{E((\vec{w}_n - \vec{w}_{n,n-1})(\vec{w}_n - \vec{w}_{n,n-1})^T) = \vec{P}_{n,n-1}\right.$$

$$2\begin{cases} E((\vec{I} - \vec{K}_n\vec{H})(\vec{w}_n - \vec{\hat{w}}_{n,n-1})(\vec{K}_n\vec{v}_n)^T) = \\ E(\vec{I} - \vec{K}_n\vec{H}) * E(\vec{w}_n - \vec{\hat{w}}_{n,n-1}) * E((\vec{K}_n\vec{v}_n)^T) = \\ E(\vec{I} - \vec{K}_n\vec{H}) * 0 * E((\vec{K}_n\vec{v}_n)^T) = 0 \end{cases}$$

$$3\begin{cases} E(\vec{K}_n\vec{v}_n(\vec{w}_n - \vec{\hat{w}}_{n,n-1})^T(\vec{I} - \vec{K}_n\vec{H})^T) = \\ E(\vec{K}_n\vec{v}_n) * E((\vec{w}_n - \vec{\hat{w}}_{n,n-1})^T) * E((\vec{I} - \vec{K}_n\vec{H})^T) = \\ E(\vec{K}_n\vec{v}_n) * 0 * E((\vec{I} - \vec{K}_n\vec{H})^T) = 0 \end{cases}$$

$$4\left\{E(\vec{v}_n\vec{v}_n^T) = \vec{R}_n\right.$$

A matriz $\vec{R}_n$ representa as covariâncias das medições, ou seja, incerteza nas medições, enquanto que $\vec{v}_n$ representa o erro da medição na iteração n. Portanto, $E(\vec{v}_n\vec{v}_n^T) = \vec{R}_n$ implica que a matriz de covariância das medições é igual à média ponderada das matrizes de covariância dos erros da medição. Logo, o cálculo de $\vec{P}_{n,n}$ se reduz a Equação 2.18.

$$\vec{P}_{n,n} = (\vec{I} - \vec{K}_n\vec{H})\vec{P}_{n,n-1}(\vec{I} - \vec{K}_n\vec{H})^T + \vec{K}_n\vec{R}_n\vec{K}_n^T \tag{2.18}$$

Por fim, o cálculo do Ganho de Kalman é obtido pela minimização dos valores da diagonal principal da matriz de covariância $\vec{P}_{n,n}$ para que se tenha o valor mínimo da variância estimada. Pela aplicação da Equação 2.18, juntamente com a propriedade de matriz transposta ($(\vec{A}\vec{B})^T = \vec{B}^T\vec{A}^T$) obtemos

$$\begin{aligned}\vec{P}_{n,n} = (\vec{P}_{n,n-1} - \vec{K}_n\vec{H}\vec{P}_{n,n-1})(\vec{I} - \vec{H}^T\vec{K}_n^T) + \vec{K}_n\vec{R}_n\vec{K}_n^T = \\ \vec{P}_{n,n-1} - \vec{P}_{n,n-1}\vec{H}^T\vec{K}_n^T - \vec{K}_n\vec{H}\vec{P}_{n,n-1} + \vec{K}_n(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)\vec{K}_n^T\end{aligned} \tag{2.19}$$

A soma da diagonal principal de uma matriz quadrática é chamada de traço da matriz e denotada por $tr(\vec{P}_{n,n})$. O objetivo é minimizar $tr(\vec{P}_{n,n})$ através das condições necessárias que produzam esse mínimo. O traço é então aplicado na Equação 2.19, derivado em relação a $\vec{K}_n$ e definido como zero para minimizar o traço.

$$\begin{aligned}&tr(\vec{P}_{n,n}) = \\ &tr(\vec{P}_{n,n-1}) - tr(\vec{P}_{n,n-1}\vec{H}^T\vec{K}_n^T) - tr(\vec{K}_n\vec{H}\vec{P}_{n,n-1}) + tr(\vec{K}_n(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)\vec{K}_n^T) \\ &= tr(\vec{P}_{n,n-1}) - 2tr(\vec{K}_n\vec{H}\vec{P}_{n,n-1}) + tr(\vec{K}_n(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)\vec{K}_n^T)\end{aligned}$$

A derivação da equação resultante, usando $\frac{d}{dA}(tr(AB)) = B^T$ e $\frac{d}{dA}(tr(ABA^T)) = 2AB$, simplifica novamente o cálculo [4].

$$\begin{aligned}&\frac{d(tr(\vec{P}_{n,n}))}{d\vec{K}_n} = \frac{d(tr(\vec{P}_{n,n-1}))}{d\vec{K}_n} - \frac{d(2tr(\vec{K}_n\vec{H}\vec{P}_{n,n-1}))}{d\vec{K}_n} + \\ &\frac{d(tr(\vec{K}_n(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)\vec{K}_n^T))}{d\vec{K}_n} = 0 - 2(\vec{H}\vec{P}_{n,n-1})^T + \\ &2\vec{K}_n(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n) = 0\end{aligned}$$

Logo, temos que $(\vec{H}\vec{P}_{n,n-1})^T = \vec{K}_n(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)$ e o ganho de Kalman é então dado pela Equação 2.20.

$$\begin{aligned}\vec{K}_n &= (\vec{H}\vec{P}_{n,n-1})^T(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)^{-1} \\ &= \vec{P}_{n,n-1}^T\vec{H}^T(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)^{-1} \\ &= \vec{P}_{n,n-1}\vec{H}^T(\vec{H}\vec{P}_{n,n-1}\vec{H}^T + \vec{R}_n)^{-1}\end{aligned} \tag{2.20}$$

2.3.2 Aplicação Prática do Filtro de Kalman Multivariado

Esta seção apresenta um exemplo prático para ilustrar a aplicação do filtro de Kalman multivariado utilizando o segundo método descrito em 2.1.2. Agora, em comparação com o exemplo anterior demonstrado em 2.2.3, alguns valores precisam ser modificados justamente por conta de se utilizar um sensor que faz parte do objeto de estudo, dessa forma, as medições são relativas ao seu posicionamento atual, ou seja, o deslocamento e ângulo atual do objeto

de estudo descritos pelas iterações anteriores afeta diretamente na medição atual, uma vez que depende delas para descrever seu estado atual. Por conta disso, erros são acumulados e carregados durante as iterações do filtro, de forma que, uma leitura destoante do valor real realizada na iteração n irá afetar as iterações consequentes, $n+1, n+2, ...$, uma vez que, sem elas, não é possível descrever o estado do objeto de estudo no plano/mapa. Então, o objetivo deste exemplo prático é demonstrar que o filtro de Kalman consegue também solucionar este problema.

O cenário descreve um robô autônomo se movendo num plano 2D e o objetivo é estimar sua posição atual com medições realizadas de 1 em 1 segundo. As iterações do processo são realizadas a cada medição, usando um mapeamento com o sensor LIDAR. O robô se move de acordo com a função $2^{\frac{x}{25}}$, variando o valor de x e mantendo a coordenada Y praticamente constante ($y \approx 0$). Em seguida, o robô faz uma curva para esquerda, aumentando os valores de x e y. Após a curva, o valor de y aumenta, mantendo o valor de x praticamente constante. O desvio padrão da aceleração é de $\alpha_a = 0.1m/s^2$ e o desvio padrão do erro de medição é de $\alpha_x = \alpha_y = 15m$, valor este consideravelmente maior se em comparação com o exemplo anterior justamente por se tratar de um outro método que considera consigo erros de medições anteriores, tornando-o menos preciso.

Iteração 0

Inicialização:

No início do processo de medição não se sabe a localização atual do objeto. Por isso, os parâmetros de posição, velocidade e aceleração assumem valor 0, como apresentado no vetor de inicialização $\vec{\hat{w}}_{0,0}$.

$$\vec{\hat{w}}_{0,0} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Como este método necessita de medições anteriores para sua localização no plano/-mapa é correto considerar seu ponto inicial sendo a origem da localização e mapeamento (SLAM), sendo assim, ao se utilizar deste método, a posição inicial do objeto de estudo será sempre 0, pelo motivo de não haver nenhuma outra medição anterior a esta. A maioria dos outros algoritmos SLAM, que utilizam ou não o filtro de Kalman, seguem este mesmo modelo, como o Hector slam [32], Gmmaping slam [33], Cartographer slam [34], entre muitos outros. Assim, como a coordenada inicial estimada é o mesma da verdadeira, a incerteza

da estimativa recebe um valor bem menor, dez vezes menos se comparado com o exemplo anterior, como descrito na Subseção 2.3.1 no cálculo da extrapolação de covariância $\vec{P}_{0,0}$ (Equação 2.14).

$$\vec{P}_{0,0} = \begin{bmatrix} 500 & 0 & 0 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 500 & 0 \\ 0 & 0 & 0 & 0 & 0 & 500 \end{bmatrix}$$

A matriz de transição de estado $\vec{F}$ é dada a seguir, seguindo o intervalo de 1 segundo a cada iteração.

$$\vec{F} = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \Delta 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0,5 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0,5 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Em seguida, a matriz de ruído de processo $\vec{Q}$ é obtida de acordo com o desvio padrão de aceleração de $0.1m/s^2$. Como este modelo não inclui dados de controle, que são correspondentes ao movimento do objeto de estudo, ou seja, comandos sobre o robô tais como virar a direita ou seguir reto, é necessário projetar a variância da aceleração $\alpha_{\ddot{x}}^2$ usando a matriz de transição de estado [4, 19]. Logo, o cálculo é obtido pela fórmula $\vec{Q} = \vec{F} * \vec{Q}_{\ddot{x}} * \vec{F}^T$, onde as matrizes $\vec{F}$ e $\vec{Q}_{\ddot{x}}$ são dadas como segue.

$$\vec{Q}_{\ddot{x}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \alpha_{\ddot{x}}^2 \quad \vec{F} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

A matriz resultante $\vec{Q}$ é então obtida como descrito a seguir.

$$\vec{Q} = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t & 0 & 0 & 0 \\ \frac{\Delta t^2}{2} & \Delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ 0 & 0 & 0 & \frac{\Delta t^2}{2} & \Delta t & 1 \end{bmatrix} * \alpha_{\ddot{x}}^2 = \begin{bmatrix} 0,0025 & 0,005 & 0,005 & 0 & 0 & 0 \\ 0,005 & 0,01 & 0,01 & 0 & 0 & 0 \\ 0,005 & 0,01 & 0,01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0,0025 & 0,005 & 0,005 \\ 0 & 0 & 0 & 0,005 & 0,01 & 0,01 \\ 0 & 0 & 0 & 0,005 & 0,01 & 0,01 \end{bmatrix}$$

Já a matriz de covariância de medição $\vec{R}$ é obtida com base no desvio padrão do erro de medição no valor de 15 metros.

$$\vec{R} = \begin{bmatrix} \alpha_x^2 & 0 \\ 0 & \alpha_y^2 \end{bmatrix} = \begin{bmatrix} 225 & 0 \\ 0 & 225 \end{bmatrix}$$

Predição:

Com as matrizes inicializadas, o filtro de Kalman bidimensional realiza as predições, calculando as matrizes $\vec{\hat{w}}_{1,0}$ e $\vec{P}_{1,0}$, que são, respectivamente, as matrizes de extrapolação de estado e covariância.

$$\vec{\hat{w}}_{1,0} = \vec{F} * \vec{\hat{w}}_{0,0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{P}_{1,0} = \vec{F} * \vec{P}_{0,0} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 75,0011 & 50,003 & 16,6699 & 0 & 0 & 0 \\ 50,003 & 83,3412 & 44,453 & 0 & 0 & 0 \\ 16,6699 & 44,453 & 48,1574 & 0 & 0 & 0 \\ 0 & 0 & 0 & 75,001 & 50,003 & 16,6699 \\ 0 & 0 & 0 & 50,003 & 83,3412 & 44,453 \\ 0 & 0 & 0 & 16,6699 & 44,453 & 48,1574 \end{bmatrix}$$

Iteração 1

Medição:

Com a predição inicial calculada, a próxima etapa do processo mede a posição atual do robô no plano obtida pelo LIDAR.

$$\vec{z}_1 = \begin{bmatrix} 4,98 \\ 1,14 \end{bmatrix}$$

Estimativa:

Na sequência as estimativas dessa iteração são calculadas: o ganho de Kalman $\vec{K_1}$, a estimativa do estado atual $\vec{\hat{w}}_{1,1}$ e a covariância atual $\vec{\hat{P}}_{1,1}$. A matriz $\vec{H}$, usada neste exemplo, é definida a seguir, com 6 dimensões (posição, velocidade e aceleração para x e y, respectivamente), onde apenas duas são dimensões mensuráveis (posição de x e y).

$$\vec{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\vec{K_1} = \vec{P}_{1,0} * \vec{H}^T * (\vec{H} * \vec{P}_{1,0} * \vec{H}^T + \vec{R})^{-1} = \begin{bmatrix} 0,59 & 0 \\ 0,43 & 0 \\ 0,15 & 0 \\ 0 & 0,59 \\ 0 & 0,43 \\ 0 & 0,15 \end{bmatrix}$$

O ganho de Kalman para a posição do robô, tanto para a coordenada x quanto para y, é de $0,5957$. O valor médio no ganho de Kalman indica que a estimativa tem um valor não muito próximo do medido, indicando que quanto maior o ganho de Kalman mais o resultado da estimativa vai depender do valor medido, proporcionalmente, em detrimento ao valor predito. O resultado é de 59,57% do valor medido e 40,43% do valor predito, respectivamente. Logo, o objeto está a uma distância de aproximadamente 60% do valor medido e a 40% do valor predito pelo filtro. Como esperado, na primeira iteração, o valor mais confiável, por mais que o erro de medição seja consideravelmente maior, é o medido e não o estimado/predito, pelo motivo de que ainda não houveram iterações suficientes para ajustar o modelo do filtro e poder se adaptar às medições de forma correta.

Em seguida a estimativa do estado atual $w_{1,1}$ e a estimativa da covariância atual $P_{1,1}$ são calculadas.

$$\vec{\hat{w}}_{1,1} = \vec{\hat{w}}_{1,0} + \vec{K_1} * (\vec{z_1} - \vec{H} * \vec{\hat{w}}_{1,0}) = \begin{bmatrix} 5,50 \\ 2,91 \\ 0,76 \\ 1,25 \\ 0,66 \\ 0,17 \end{bmatrix}$$

$$\vec{P}_{1,1} = (\vec{I} - \vec{K}_1\vec{H})\vec{P}_{1,0}(\vec{I} - \vec{K}_1\vec{H})^T + \vec{K}_1\vec{R}\vec{K}_1^T = \begin{bmatrix} 543,95 & 349,23 & 107,77 & 0 & 0 & 0 \\ 349,23 & 262,88 & 90,88 & 0 & 0 & 0 \\ 107,77 & 90,88 & 35,13 & 0 & 0 & 0 \\ 0 & 0 & 0 & 543,95 & 349,24 & 107,77 \\ 0 & 0 & 0 & 349,23 & 262,88 & 90,88 \\ 0 & 0 & 0 & 107,77 & 90,88 & 35,13 \end{bmatrix}$$

Os valores da estimativa do estado atual do objeto, no vetor $w_{1,1}$, representam a posição, velocidade e aceleração do robô nas coordenadas x e y. Esses valores mostram de que forma o robô está se movimentando de uma iteração para outra. Já os valores para a estimativa da covariância na matriz $\vec{P}_{1,1}$, representam a incerteza associada ao estado estimado nesta iteração. Dessa forma, cada posição da matriz representa a faixa de variação de cada parâmetro associado. Neste caso, os valores da posição x e y podem variar em relação resultado de $\vec{\hat{w}}_{1,1}$ por $\pm 134,03\%$. Já o valor da aceleração de ambas as coordenadas pode variar de $\pm 35,12\%$. Note que os valores obtidos na matriz $\vec{P}_{n,n}$ tendem a diminuir, de uma forma geral, pois o nível de incerteza diminui conforme as iterações são realizadas.

Predição:

Na primeira iteração é preciso também calcular a predição do próximo estado, que será usada para calcular as estimativas da próxima iteração, juntamente com a leitura da posição real do robô.

$$\vec{\hat{w}}_{2,1} = \vec{F} * \vec{\hat{w}}_{1,1} = \begin{bmatrix} 5,50 \\ 2,91 \\ 0,76 \\ 1,25 \\ 0,66 \\ 0,17 \end{bmatrix}$$

$$\vec{P}_{2,1} = \vec{F} * \vec{P}_{1,1} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 134,03 & 97,34 & 34,44 & 0 & 0 & 0 \\ 97,34 & 116,24 & 55,75 & 0 & 0 & 0 \\ 34,44 & 55,75 & 35,12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 134,03 & 97,34 & 34,44 \\ 0 & 0 & 0 & 97,34 & 116,24 & 55,75 \\ 0 & 0 & 0 & 34,44 & 55,75 & 35,12 \end{bmatrix}$$

Iteração 2

Medição:

$$\vec{z}_2 = \begin{bmatrix} 10,04 \\ 1,32 \end{bmatrix}$$

Estimativa:

$$\vec{K}_2 = \vec{P}_{2,1} * \vec{H}^T * (\vec{H} * \vec{P}_{2,1} * \vec{H}^T + \vec{R})^{-1} = \begin{bmatrix} 0,7074 & 0 \\ 0,4542 & 0 \\ 0,1402 & 0 \\ 0 & 0,7074 \\ 0 & 0,4542 \\ 0 & 0,1402 \end{bmatrix}$$

$$\vec{\hat{w}}_{2,2} = \vec{\hat{w}}_{2,1} + \vec{K}_2 * (\vec{z}_2 - \vec{H} * \vec{\hat{w}}_{2,1}) = \begin{bmatrix} 8,71 \\ 4,97 \\ 1,39 \\ 1,30 \\ 0,69 \\ 0,18 \end{bmatrix}$$

$$\vec{P}_{2,2} = (\vec{I} - \vec{K}_2\vec{H})\vec{P}_{2,1}(\vec{I} - \vec{K}_2\vec{H})^T + \vec{K}_2\vec{R}\vec{K}_2^T = \begin{bmatrix} 159,16 & 102,18 & 31,53 & 0 & 0 & 0 \\ 102,18 & 104,27 & 41,94 & 0 & 0 & 0 \\ 31,53 & 41,94 & 20,02 & 0 & 0 & 0 \\ 0 & 0 & 0 & 159,16 & 102,18 & 31,53 \\ 0 & 0 & 0 & 102,18 & 104,27 & 41,94 \\ 0 & 0 & 0 & 31,53 & 41,93 & 20,02 \end{bmatrix}$$

Percebe-se que a incerteza associada a cada medição e suas covariâncias aumentou levemente da matriz $\vec{P}_{1,1}$ para a $\vec{P}_{2,2}$. Este fato ocorre pois o erro na matriz de covariância de medição $\vec{R}$ é consideravelmente alto, o que leva a uma baixa confiança nos dados medidos, levando ao filtro necessitar de mais iterações para que consiga se estabilizar. Além disso, uma convergência mais lenta desses valores pode ocorrer devido as grandezas dos parâmetros, tais como velocidade, que é a resultante da integral da posição do objeto, e aceleração, que é a resultante da integral da velocidade desse objeto.

Predição:

$$\vec{\hat{w}}_{3,2} = \vec{F} * \vec{\hat{w}}_{2,2} = \begin{bmatrix} 14,38 \\ 6,37 \\ 1,39 \\ 2,08 \\ 0,87 \\ 0,18 \end{bmatrix}$$

$$\vec{P}_{3,2} = \vec{F} * \vec{P}_{2,2} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 546,29 & 310,92 & 83,49 & 0 & 0 & 0 \\ 310,92 & 208,19 & 61,97 & 0 & 0 & 0 \\ 83,49 & 61,97 & 20,03 & 0 & 0 & 0 \\ 0 & 0 & 0 & 546,30 & 310,92 & 83,49 \\ 0 & 0 & 0 & 310,92 & 208,19 & 61,97 \\ 0 & 0 & 0 & 83,49 & 61,97 & 20,03 \end{bmatrix}$$

Iteração 35

Após várias iterações os valores das matrizes do ganho de Kalman, $\vec{K}_{35}$, e da estimativa de covariância, $\vec{P}_{35,35}$, estimados e preditos pelo filtro, acabaram convergindo para valores mais condizentes do estado real do robô. Nota-se que o número de iterações para que valores mais precisos sejam obtidos pelo filtro depende de cada exemplo/aplicação. A estabilidade do processo pode ser notada de uma iteração para outra quando os valores de $\vec{P}_{n,n}$ para $\vec{P}_{n+1,n}$ variam pouco.

Medição:

$$\vec{z}_{35} = \begin{bmatrix} 169,97 \\ 125,15 \end{bmatrix}$$

Estimativa:

$$\vec{K}_{35} = \vec{P}_{35,34} * \vec{H}^T * (\vec{H} * \vec{P}_{35,34} * \vec{H}^T + \vec{R})^{-1} = \begin{bmatrix} 0,3145 & 0 \\ 0,0592 & 0 \\ 0,0056 & 0 \\ 0 & 0,3146 \\ 0 & 0,0592 \\ 0 & 0,0056 \end{bmatrix}$$

$$\vec{\hat{w}}_{35,35} = \vec{\hat{w}}_{35,34} + \vec{K}_{35} * (\vec{z}_{35} - \vec{H} * \vec{\hat{w}}_{35,34}) = \begin{bmatrix} 169,97 \\ 4,93 \\ -0,01 \\ 111,77 \\ 10,76 \\ 0,61 \end{bmatrix}$$

$$\vec{P}_{35,35} = (\vec{I} - \vec{K}_{35}\vec{H})\vec{P}_{35,34}(\vec{I} - \vec{K}_{35}\vec{H})^T + \vec{K}_{35}\vec{R}\vec{K}_{35}^T = \begin{bmatrix} 70,77 & 13,32 & 1,25 & 0 & 0 & 0 \\ 13,32 & 4,02 & 0,51 & 0 & 0 & 0 \\ 1,25 & 0,51 & 0,09 & 0 & 0 & 0 \\ 0 & 0 & 0 & 70,77 & 13,33 & 1,25 \\ 0 & 0 & 0 & 13,32 & 4,02 & 0,51 \\ 0 & 0 & 0 & 1,24 & 0,51 & 0,09 \end{bmatrix}$$

Predição:

$$\vec{\hat{w}}_{36,35} = \vec{F} * \vec{\hat{w}}_{35,35} = \begin{bmatrix} 174,90 \\ 4,92 \\ -0,01 \\ 122,84 \\ 11,37 \\ 0,61 \end{bmatrix}$$

$$\vec{P}_{36,35} = \vec{F} * \vec{P}_{35,35} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 103,25 & 19,43 & 1,82 & 0 & 0 & 0 \\ 19,43 & 5,17 & 0,62 & 0 & 0 & 0 \\ 1,82 & 0,62 & 0,10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 103,25 & 19,44 & 1,82 \\ 0 & 0 & 0 & 19,42 & 5,17 & 0,62 \\ 0 & 0 & 0 & 1,82 & 0,62 & 0,10 \end{bmatrix}$$

Resultados

A Figura 7 apresenta o gráfico dos resultados obtidos da aplicação do filtro de Kalman bidimensional no cenário descrito. A linha *Real* mostra a a posição real do robô no ambiente. A linha *Medido* é obtida pelos valores mensurados durante as iterações através das leituras do sensor LIDAR. A linha *Estimado* mostra os valores estimados pelo filtro, na matriz $\vec{\hat{w}}_{n,n}$, ao longo das iterações. Por fim, a linha do *Intervalo de Confiança* representa a faixa de valores

confiáveis do filtro, ou seja, de acordo com a incerteza associada $\vec{P}_{n,n}$, onde valores mais altos indicam um maior intervalo de confiança, é gerada uma faixa em torno dos valores Reais que representa 95% de precisão. Logo, os valores dentro da faixa de confiança são os 5% mais precisos.

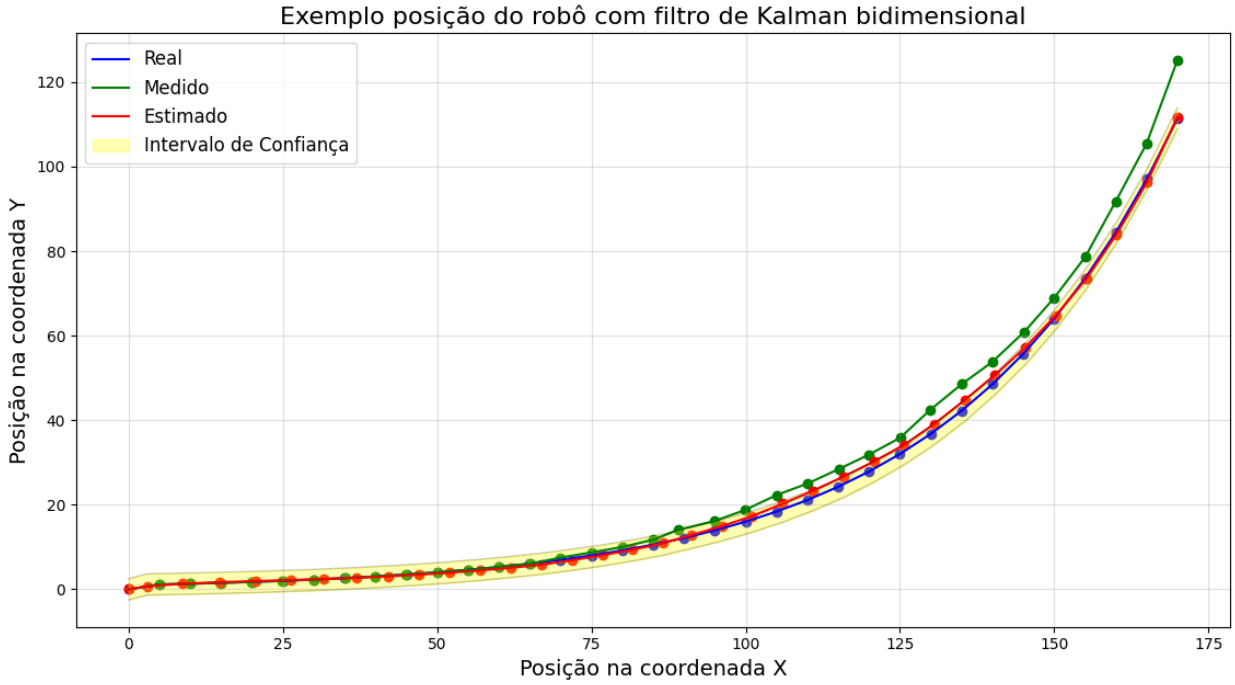

Figura 7 – Exemplo prático do filtro de Kalman bidimensional

Percebe-se que os valores estimados tendem a permanecer no intervalo da faixa de confiança, indicando que a execução do filtro de Kalman é confiável mesmo que os valores medidos estejam bem destoantes dos verdadeiros. Outra observação importante pode ser concluída com respeito ao exemplo dado no filtro de Kalman unidimensional. No exemplo multidimensional é possível perceber que os valores estimados tendem a ser mais próximos do resultado real quando comparados aos valores medidos. Isso ocorre devido as estimativas do modelo serem mais precisas do que os sensores usados para capturar as informações sobre o objeto, naquele caso. Outro motivo está relacionado ao número de variáveis no modelo bidimensional. Assim, uma medição que com um erro aproximado de 1% em relação aos valores reais fica mais visível do que a mesma medição considerando apenas uma única dimensão. No caso do modelo bidimensional as imprecisões ocorrem tanto para a coordenada x quanto para a coordenada y, permitindo uma melhor visualização do funcionamento do filtro.

2.3.3 Projetando o Ambiente Tridimensional para Bidimensional

A predição em sistemas tridimensionais muitas vezes ocorre com a transformação de um ambiente 3D para o espaço 2D, para então ser aplicado o modelo bidimensional. Essa projeção do plano, também chamada de planificação, é uma etapa crucial em muitas aplicações que envolvem ambientes reais. O objetivo principal dessa planificação é a representação de informações 3D num formato 2D, anulando, ou ignorando, o eixo z, de modo a evitar que medições e predições sobre este plano sejam calculadas durante as etapas do processo de execução do filtro. Assim, este processo de planificação quando aplicado nas múltiplas dimensões que estão relacionadas nos cálculos realizados, como por exemplo na determinação do valor da covariância, impede que erros acumulados e aleatórios calculados de medições sobre o plano z sejam considerados nas etapas do processo de predição.

Esse processo pode ou não ser possível de se realizar dependendo do objetivo da aplicação. No caso de simulação de robôs em logística industrial ou uso doméstico, como os aspiradores de pó automático, que se locomovem apenas em solo, a terceira dimensão (altura) nas equações do filtro seriam desnecessárias e, possivelmente, prejudiciais no cálculos. Já em aplicações com drones e robôs submersíveis, essa estratégia de eliminar o plano z não pode ser aplicada. Além disso, o método de planificação mencionado, e utilizado no decorrer deste trabalho, tem relação com uma projeção ortogonal que gera uma visão panorâmica superior do ambiente (uma "vista de cima"). Todos os objetos presentes no ambiente medido se tornam planificados sobre o eixo z, como por exemplo uma parede, que nesta visão se torna um contorno sem a medida de altura, mantendo apenas a largura e o comprimento.

A Figura 8 ilustra essa transformação na simulação de um ambiente real. O sensor LIDAR percorre apenas em duas dimensões, largura e comprimento, ignorando a altura. As medições dos objetos no ambiente são enviadas ao filtro diretamente planificado, com apenas duas dimensões. Como se observa na figura, o sensor LIDAR 2D realiza a leitura, localizando as paredes e obstáculos, num plano 2D para obter o mapa gerado e representado na lupa no canto inferior direito.

Portanto, uma solução num ambiente tridimensional pode ser obtida através das equações apresentadas na Subseção 2.3.1 usando o modelo do filtro de Kalman bidimensional.

2.4 Algoritmos e Estrutura de Código

Esta seção apresenta as estruturas e algoritmos relacionados ao filtro de Kalman bivariado usando o modelo multivariado para o problema SLAM. Num primeiro momento são apresentadas as inicializações das matrizes para a execução do filtro. Em seguida, o

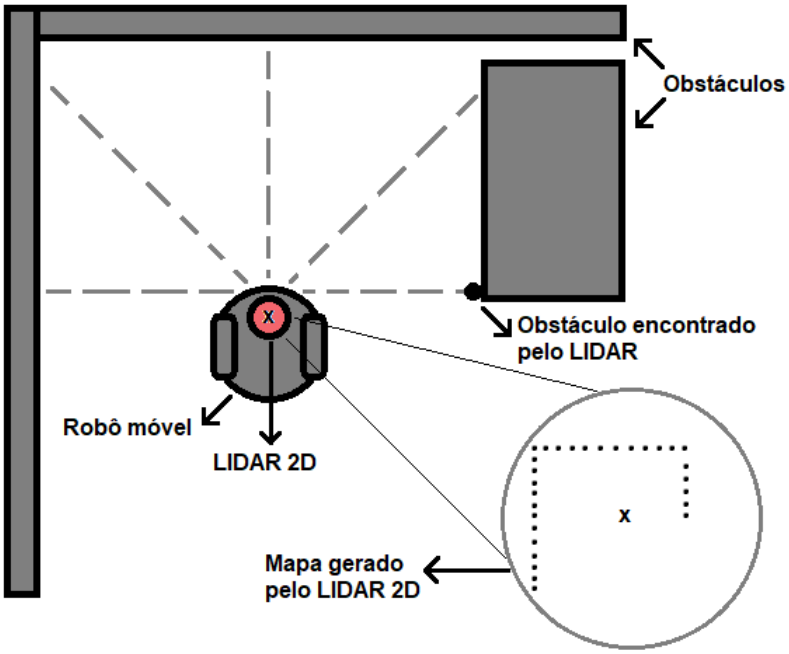

Figura 8 – Leitura planificada de um LIDAR 2D em um ambiente 3D.

algoritmo relacionado a captura de dados pelo sensor LIDAR é descrito de forma a retornar um vetor de pontos relacionados as leituras dos objetos detectados pelo sensor. Por fim, o principal algoritmo, que efetivamente resolve o filtro, é apresentado.

A inicialização do filtro define as matrizes $\vec{F}$, $\vec{Q}$ e $\vec{R}_n$ com os valores constantes frequentemente usados ao longo do processo [4, 35, 19], bem como as matrizes que representam o palpite inicial sobre a posição do objeto, $\vec{\hat{w}}_{0,0}$, e da covariância estimada, $\vec{P}_{0,0}$. Já as variâncias estimadas, definidas na matriz $\vec{P}_{0,0}$, são usualmente iniciadas com valores altos, pois o intuito é que, por mais distante que estejam os valores do palpite inicial (posição, velocidade e aceleração), o filtro deve convergir sempre para um valor próximo.

As inicializações das matrizes são obtidas de acordo com o período das medições, Δt, desvio padrão da aceleração aleatória, σ_a, desvio padrão do erro de medição, σ_x e σ_y, o palpite inicial de posição, velocidade e aceleração do objeto de estudo nos eixos x e y, $\vec{\hat{w}}_{n,n}$, e a variância estimada da posição do objeto de estudo nos eixos x e y, p_x e p_y. A seguir são apresentadas todas as matrizes mencionadas.

$$\vec{\hat{w}}_n = \begin{bmatrix} x_n \\ y_n \\ \dot{x}_n \\ \dot{y}_n \\ \ddot{x}_n \\ \ddot{y}_n \end{bmatrix} \qquad \vec{F} = \begin{bmatrix} 1 & \Delta t & 0.5\Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \Delta 0.5\Delta t^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\vec{P}_{n,n} = \begin{bmatrix} p_x & 0 & 0 & 0 & 0 & 0 \\ 0 & p_{\dot{x}} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{\ddot{x}} & 0 & 0 & 0 \\ 0 & 0 & 0 & p_y & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{\dot{y}} & 0 \\ 0 & 0 & 0 & 0 & 0 & p_{\ddot{y}} \end{bmatrix} \qquad \vec{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\vec{R}_n = \begin{bmatrix} \sigma_{xm}^2 & 0 \\ 0 & \sigma_{ym}^2 \end{bmatrix} \qquad \vec{Q} = \begin{bmatrix} \frac{\Delta t^2}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t & 0 & 0 & 0 \\ \frac{\Delta t^2}{2} & \Delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\Delta t^2}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ 0 & 0 & 0 & \frac{\Delta t^2}{2} & \Delta t & 1 \end{bmatrix} * \sigma_{\ddot{x}}^2$$

O Algoritmo 1 é responsável pelo processamento dos dados provenientes do sensor LIDAR [23, 36]. O sensor LIDAR realiza uma volta completa (360 graus) em torno do próprio eixo, a cada iteração, coma duração de um segundo. Dessa forma, cada leitura do sensor é dividida pela quantidade de graus que existe entre essas leituras. Esse valor pode variar de acordo com a especificação de cada sensor LIDAR, como por exemplo, um sensor LIDAR que mede os pontos girando em torno de si mesmo com uma diferença de um grau por ponto, são totalizadas 360 leituras. Portanto, o laço for deve percorrer de 0 até $\frac{360}{\theta}$, onde θ indica o parâmetro com a quantidade de graus entre as leituras.

Algorithm 1: LIDAR_Data_3D-2D(θ)

```
1 v⃗et = [ ];
2 for i ∈ 0... 360/θ do
3 |   rad = i*θ * π/180;
4 |   distance = getLidarDist(rad);
5 |   x = distance * cos(rad);
6 |   y = distance * sin(rad);
7 |   v⃗et[i] = (x,y);
8 return v⃗et
```

O número de pontos lidos pelo sensor LIDAR é igual a $\frac{360}{1} = 360$. Num outro cenário, se considerarmos a diferença de 0,5 graus entre as leituras, o número de pontos retornados é de $\frac{360}{0,5} = 720$ pontos. Dessa forma, o Algoritmo 1 calcula as coordenadas x, y sobre as leituras do LIDAR e armazena no vetor $\vec{vet}[(x_1, y_1); (x_2, y_2); ...; (x_i, y_i)]$. A função *getLidarDist* no Algoritmo 1 recebe como parâmetro com o valor do ângulo em radianos e retorna a distância

obtida pelo LIDAR para este ângulo. As funções *cos* e *sin* retornam, respectivamente, o valor do cosseno e do seno do ângulo em radianos. O custo deste algoritmo é constante, uma vez que apenas três multiplicações e uma consulta são realizadas, e o número de iterações depende do ângulo entre as leituras do aparelho de medição que tem valor fixo, não sofrendo alteração durante a execução do algoritmo.

O Algoritmo 2 [37] descreve o funcionamento do ICP (Veja Seção 2.1.1) para calcular o valor da translação e da rotação do objeto a partir de duas nuvens de pontos distintas. Essas nuvens de pontos são obtidas entre as iterações do filtro de Kalman pelo sensor LIDAR no Algoritmo 1. O ICP compara cada ponto de uma nuvem com todos os outros pontos da outra nuvem, buscando o ponto mais próximo. Logo, o algoritmo deve retornar o menor valor de translação e de rotação. Em seguida, o algoritmo translada e rotaciona todos os pontos para essa direção, de tal forma que uma nuvem sobreponha a outra. O processo se repete até que cada ponto ache seu correspondente, somando as translações e rotações de cada iteração e obtendo o resultado final do algoritmo [38].

O algoritmo recebe por parâmetro os dois vetores com as nuvens de pontos, como descrito na Subseção 2.1.1, usando o Algoritmo 1 para obter o vetor de pontos $\vec{vet}$ lidos em uma iteração. Assim, a cada iteração do LIDAR são realizadas as leituras e envidadas, a leitura atual para a primeira nuvem de pontos ($\vec{n}_1$), enquanto que a leitura da iteração anterior para a segunda nuvem de pontos ($\vec{n}_2$). Cada leitura de nuvem de pontos é realizada no Algoritmo 3, através da chamada do Algoritmo 1, armazenando a leitura atual em $atualMeasurement$ e a leitura anterior em $prevMeasurement$, respectivamente, $\vec{n}_1$ e $\vec{n}_2$. Note que as duas nuvens de pontos possuem tamanhos dependentes da quantidade de leituras que o LIDAR faz por rotação.

As linhas 1 à 4 do algoritmo inicializam os valores da translação, rotação, e também dos vetores de pontos. O vetor de pontos_não_correspondidos, contém a primeira nuvem de pontos e tem como função armazenar os pontos que ainda não foram encontrados os respectivos pontos correspondentes mais próximo na segunda nuvem de pontos. O vetor de pontos_correspondidos é inicializado vazio e vai armazenando os pontos encontrados e correspondentes à primeira nuvem. Quando todos os pontos correspondentes entre os vetores são encontrados o algoritmo termina. Essa condição é descrita pelo laço da linha 5, onde os tamanhos dos vetores são comparados. Quando as dimensões dos vetores são iguais então todos os pontos foram devidamente correspondidos. O algoritmo então termina e retorna a translação e a rotação entre as leituras.

As linhas de 6 a 8 inicializam as variáveis usadas na comparação da translação e rotação de cada ponto para achar o menor valor da iteração. A variável translação_mínima

Algorithm 2: ICP($\vec{n}_1$, $\vec{n}_2$)

```
1  translação = 0 ;                          // Definição dos valores e parâmetros
2  rotação = 0;
3  pontos_correspondidos = [ ];
4  pontos_não_correspondidos = n⃗1;
5  while ‖pontos_correspondidos‖ ≠ ‖n⃗1‖ do
6      translação_mínima = -1 ;              // Inicialização dos menores valores
7      rotação_mínima = 0 ;                  // encontrados nesta iteração
8      ponto_mínimo = 0;
9      for ponto1 ∈ pontos_não_correspondidos do
10         for ponto2 ∈ n⃗2 do
11             translação_atual = √((ponto2_x − ponto1_x)² + (ponto2_y − ponto1_y)²) ;
                // Calcula a translação entre ponto1 e ponto2
12             if translação_mínima == -1 ∨ translação_atual < translação_mínima
                then
13                 translação_mínima = translação_atual ; // Atualiza os menores
                    valores encontrados
14                 rotação_mínima = θ = arctan((ponto2_y − ponto1_y)/(ponto2_x − ponto1_x));
15                 ponto_mínimo = ponto1;
16     translação + = translação_mínima ;    // Soma a translação e
17     rotação + = rotação_mínima ;          // a rotação encontradas
18     pontos_correspondidos + = ponto_minimo ; // Atualiza os dois vetores
19     pontos_não_correspondidos − = ponto_mínimo;
20     for ponto ∈ pontos_não_correspondidos do
21         ponto_x = ponto_x+ translação_mínima * cos(rotação_mínima) ;
            // Atualiza o vetor de pontos_não_correspondidos para
            receberem a translação e rotação da iteração atual
22         ponto_y = ponto_y+ translação_mínima * sin(rotação_mínima);
23 return [translação, rotação]
```

é inicializada com o valor -1, para que ao menos uma translação seja obtida já que uma translação entre nuvens não pode ter valor negativo.

Os laços aninhados das linhas 9 e 10 realizam a busca de todos os pares de pontos que ainda não foram correspondidos do primeiro vetor com todos os pontos da segunda nuvem. A linha 11 calcula a distância entre os pontos (percorridos nas linhas 9 e 10), através da fórmula da distância euclidiana, $\sqrt{(ponto2_x - ponto1_x)^2 + (ponto2_y - ponto1_y)^2}$, entre dois pontos em um plano, onde $ponto1_x, ponto1_y$ são as coordenadas do primeiro ponto e $ponto2_x, ponto2_y$ são as coordenadas do segundo ponto, resultando na distância entre os dois pontos no plano bidimensional. A linha 12 verifica se essa distância é a menor encontrada até o momento e,

se for o caso, as linhas de 13 a 15 atualizam as menores translação e rotação, armazenando-o na variável ponto_mínimo para em seguida ser inserido no vetor pontos_correspondidos e removido do vetor pontos_não_correspondidos. A fórmula apresentada na linha 14 calcula o ângulo entre a reta que conecta dois pontos em um plano bidimensional. A função arco tangente ($arctan$) calcula o ângulo formado pela reta que passa pelos dois pontos em relação ao eixo x positivo. De forma geral, a razão $\frac{ponto2_y - ponto1_y}{ponto2_x - ponto1_x}$ representa a inclinação (ou coeficiente angular) da linha entre os dois pontos. Logo, a função arco tangente retorna o ângulo correspondente a essa inclinação.

As linhas de 16 a 19 atualizam os valores de translação e rotação com os menores pontos encontrados da iteração corrente, bem como os vetores de pontos_correspondidos e pontos_não_correspondidos. Já nas linhas de 20 a 22 o vetor de pontos_não_correspondidos é atualizado com a distância e ângulo encontrados no processo de translação e rotação de um ponto em um plano bidimensional. O cálculo é obtido pela multiplicação do valor da translação pelo cosseno e seno do ângulo de rotação, adicionando esse valor às coordenadas x e y originais do ponto, fazendo com que o ponto se mova de acordo com a direção definida pela rotação. Por fim, o algoritmo encerra retornando o valor total de translação e rotação entre a primeira e a segunda nuvem.

O Algoritmo 2 [38] tem custo computacional $O(n^3)$ no tamanho da nuvem de pontos. Como pode ser observado, na linha 5 os pontos de $\vec{n}_1$, de tamanho n, são copiados para o vetor pontos_correspondidos, resultando num custo de $O(n)$. Em seguida, os pontos de $\vec{n}_1$ são copiados novamente para o vetor pontos_não_correspondidos, na linha 9, também resultando num custo $O(n)$. Por fim, a linha 10 compara os pontos da primeira nuvem com os pontos da segunda nuvem, o vetor $\vec{n}_2$, com custo $O(n^2)$. Logo, os três ciclos de repetições aninhados (linhas 5, 9 e 10) de tamanho n resulta num custo computacional da ordem $O(n^3)$ para o algoritmo.

Já o Algoritmo 3 descreve a computação das estimativas e predições do filtro [31], com base nos dados obtidos do Algoritmo 1 e da inicialização das matrizes. O algoritmo retorna a estimativa do estado atual $\vec{\hat{w}}_{n,n}$ e a estimativa da covariância atual $\vec{P}_{n,n}$ do objeto. No processo, os valores da matriz do ganho de Kalman também são calculados, bem como os valores de posição e covariância, tanto estimados quanto preditos, onde apenas os valores preditos a cada iteração são armazenados.

O algoritmo recebe, primeiramente, os parâmetros de inicialização do filtro de Kalman: as matrizes $\vec{\hat{w}}_{n,n}$, que representa o palpite inicial do estado do objeto de estudo, $\hat{w}$, a matriz de observação, e $\vec{P}_{n,n}$, a matriz que representa a covariância estimada. Além disso, o algoritmo também precisa do ângulo sob o qual cada leitura do LIDAR deve ocorrer. Em seguida, os

Algorithm 3: SLAM_Kalman_3D-2D($\Delta t, \sigma_{xm}, \sigma_{ym}, \vec{\hat{w}}_{n,n}, \vec{H}, p_x, p_y, \theta$)

```
1  n = 0;
2  prevMeasurement = LIDAR_Data_3D-2D(θ) ;          // Leitura do ambiente
   inicial
3  while verdadeiro do
4      ŵ⃗_{n+1,n} = F⃗ * ŵ⃗_{n,n} ;                          // Predição da posição
5      P⃗_{n+1,n} = F⃗ * P⃗_{n,n} * F⃗^T + Q⃗ ;        // Predição da covariância estimada
6      n = n + 1 ;                                        // Próxima iteração
7      atualMeasurement = LIDAR_Data_3D-2D(θ) ;     // Leitura do ambiente
       atual
8      z⃗_n = ICP(atualMeasurement, prevMeasurement) ;          // Medição do
       deslocamento do objeto
9      K⃗_n = P⃗_{n,n-1} * H⃗^T * (H⃗ * P⃗_{n,n-1} * H⃗^T + R⃗_n)^{-1} ; // Estimativa do Ganho de
       Kalman
10     ŵ⃗_{n,n} = ŵ⃗_{n,n-1} + K⃗_n * (z⃗_n − H⃗ * ŵ⃗_{n,n-1}) ;        // Estimativa da posição
11     P⃗_{n,n} = (I⃗ − K⃗_n * H⃗) * P⃗_{n,n-1} * (I⃗ − K⃗_n * H⃗)^T + K⃗_n * R⃗_n * K⃗_n^T ;
       // Estimativa da covariância
12     prevMeasurement = atualMeasurement ;   // Leitura do ambiente atual
       passa a ser o prévio
```

cálculos descritos nos modelos uni e multivariado são realizados dentro de uma estrutura de repetição, enquanto o sistema de monitoramento estiver em funcionamento. Logo, o algoritmo realiza as operações de estimativa e predição enquanto há dados de medição e até quando o usuário desejar.

A linha 4 do algoritmo calcula a predição da posição do objeto, usando o chute inicial $w_{n,n}$ quando $n = 0$. A multiplicação de $w_{n,n}$ pela matriz de transição de estado $\vec{F}$ segue sempre as mesmas dimensões para ambas as matrizes ($6 \times 6 * 6 \times 1$). Já a linha 5 calcula a extrapolação de covariância do próximo estado através da multiplicação das matrizes de transição de estado $\vec{F}$, de covariância estimada $\vec{P}_{n,n}$ e de transição de estado transposta $\vec{F}^T$, de dimensões 6×6. A resultante, uma matriz 6×6, é somada a matriz de covariância de ruído $\vec{Q}$, também resultando numa matriz 6×6. Como o custo computacional para a multiplicação de duas matrizes de dimensões $p \times q$ e $q \times r$, respectivamente, é da ordem de $O(p \times q \times r)$, onde p, q, r são as dimensões das matrizes, e as dimensões das matrizes consideradas nos cálculos do Algoritmo 3 são constantes, o custo das linhas 4 e 5 são constantes.

Na linha 6 o número de iterações especificado pelo usuário da aplicação, n, é incrementado para o cálculo dos parâmetros da próxima iteração. Na linha 7 uma nova leitura do ambiente é realizada pelo Algoritmo 1. Na linha 8 o algoritmo do ICP calcula a translação e rotação do objeto de estudo comparando as matrizes de pontos da iteração anterior com a

atual.

Em seguida, as linhas de 9 a 11 realizam os cálculos do ganho de Kalman, da estimativa do estado, e da estimativa da covariância, respectivamente, do estado atual. As matrizes da linha 9 tem dimensões 6×6, com exceção da matriz de covariância da medição $\vec{R}_n$ com dimensões 2×2. Assim, a multiplicação de $\vec{H} * \vec{P}_{n,n-1} * \vec{H}^T$, que resulta em uma matriz 2×2, pode ser somada a matriz $\vec{R}_n$. A inversa dessa resultante é então multiplicada pela matriz resultante da multiplicação de $\vec{P}_{n,n-1} * \vec{H}^T$, que resulta em uma matriz 6×2. Já na linha 10 a matriz $\vec{w}_{n,n-1}$ de dimensões 6×1, a matriz de ganho de Kalman $\vec{K}_n$ de tamanho 6×2, a matriz de medição $\vec{z}_n$ de tamanho 2×1 e a matriz de observação $\vec{H}$ com dimensões 2×6, resultam na matriz do ganho de Kalman $\vec{K}_n$ de dimensões 6×2. A linha 11 calcula a matriz de covariância $\vec{P}_{n,n}$, através das matrizes $\vec{I}$ e $\vec{P}_{n,n-1}$ de tamanho 6×6, $\vec{K}_n$ e $\vec{H}$ de tamanhos 2×6, e $\vec{R}_n$ de tamanho 2×2. As operações $\vec{I} - \vec{K}_n * \vec{H} * \vec{P}_{n,n-1}$, $(\vec{I} - \vec{K}_n * \vec{H})^T$ e $\vec{K}_n * \vec{R}_n * \vec{K}_n^T$ resultam em matrizes de tamanho 6×6. Logo, a matriz de covariância $\vec{P}_{n,n}$ também possui dimensões 6×6. Por fim, a leitura do ambiente atual passa a ser a leitura do ambiente na iteração anterior na linha 12, para que uma nova iteração seja iniciada.

O custo associado aos cálculos das matrizes no Algoritmo 3 é constante já que suas respectivas dimensões são fixas e pré determinadas. Logo, o custo computacional do algoritmo é da ordem de $O(n)$, em função do número de iterações realizadas ao longo da execução do filtro de Kalman.

Portanto, o custo computacional do SLAM, com base nos três algoritmos que compõem a solução, é dominado pelo custo do ICP, $O(n^3)$, em relação ao tamanho da nuvem de pontos. Considere um cenário em que o LIDAR usa 360 pontos (um ponto a cada grau). O custo deste algoritmo seria da ordem de 360^3, ou seja, 46.656.000 iterações no total, enquanto o algoritmo do LIDAR resultaria em apenas 360 operações e como o algoritmo do filtro de Kalman não tem relação com a nuvem de pontos, seu custo linear é relativo apenas com o tempo de execução do algoritmo. Por exemplo, se o usuário deseja executar o SLAM durante uma hora e espera o resultado da posição atual do robô a cada um segundo, o número de iterações realizadas seriam de 3600. Assim, fica evidente que o maior custo computacional no SLAM está associado ao processo de alinhamento das nuvens de pontos, enquanto o filtro de Kalman e o LIDAR apresentam uma carga computacional relativamente constante e previsível.

3 SLAM NO PROBLEMA DE LOCALIZAÇÃO

O SLAM desempenha um papel crucial em sistemas autônomos para estimar a posição de veículos em relação ao ambiente em que se localiza enquanto, simultaneamente, mapeia esse ambiente. No entanto, este problema é desafiador devido à presença de incertezas nos sensores, imprecisões nos modelos de movimento dos veículos e a necessidade de lidar com grandes volumes de dados em tempo real. O filtro de Kalman então surge como uma ferramenta valiosa para lidar com essas incertezas, permitindo estimativas precisas do estado do sistema com base em dados de sensores imprecisos. Dessa forma, o objetivo deste trabalho é aplicar o filtro de Kalman como uma solução SLAM para o problema de localização de robôs AMR obtendo estimativas mais confiáveis sobre a movimentação desses robôs e, assim, desenvolver um sistema que possa monitorar e predizer o posicionamento dos robôs autônomos.

O problema de localização de sistemas autônomos em ambientes variados é, em diversos casos, solucionado com as utilidades providas pelos robôs AMR[1], geralmente usados em fábricas e indústrias na área de logística industrial para a automatização de determinados processos como o transporte de objetos de maneira automática [39], juntamente com os dados e estimativas provenientes do SLAM. As utilidades, funções e capacidades de robôs AMR serão descritas na seção a seguir.

3.1 Os Robôs AMR

No âmbito da logística industrial, o transporte autônomo de cargas tem se destacado como uma solução inovadora e eficiente para otimizar operações logísticas em ambientes industriais, reduzir custos e aumentar a segurança geral no carregamento e transporte de cargas. Nesse contexto, a precisão na localização de robôs autônomos AMR é crucial para garantir a eficiência, segurança e fluidez das operações. O problema de localização neste cenário é complexo devido à natureza dinâmica e não estruturada dos ambientes industriais, que apresentam desafios, tais como a presença de obstáculos móveis, variações na iluminação e a necessidade de ter conhecimento, em tempo real, do estado e posição dos robôs para que suas tarefas sejam coordenadas corretamente.

Os AMR são sistemas robóticos projetados para operar de forma independente e autônoma em diversos ambientes, especialmente industriais e logísticos. Este robôs são capazes de realizar uma variedade de tarefas, como transporte de carga, inspeção, mapeamento e outras atividades específicas, sem a necessidade de intervenção humana direta. A princi-

[1] do inglês, *Autonomous Mobile Robots*

pal característica que define os AMRs é sua capacidade de se mover de maneira autônoma, adaptando-se a ambientes dinâmicos e variáveis, o que os torna ideais para lidar com os desafios encontrados nesses ambientes.

A capacidade autônoma de navegação dos robôs AMRs é fundamental para sua utilidade prática. Por isso, os robôs são equipados com sensores avançados, como LIDAR, que lhes permitem mapear o ambiente em tempo real, identificar obstáculos e calcular trajetórias ótimas para o transporte de carga. Essa autonomia reduz a dependência de infraestrutura fixa, como trilhos ou guias, tornando os AMRs altamente flexíveis e adaptáveis a diferentes cenários industriais.

Além da localização precisa, os AMRs também podem ser projetados para colaborar de forma segura com humanos e outros equipamentos no ambiente de trabalho. Algumas funcionalidades precisas são sistemas de detecção de obstáculos e algoritmos de planejamento de trajetória segura, que garantem uma interação suave e segura, minimizando o risco de colisões ou danos.

Um dos desafios enfrentados pelos robôs AMR é a precisão na localização em ambientes dinâmicos. Neste sentido, o SLAM desempenha um papel crucial juntamente com o filtro de Kalman, onde o objetivo é manter uma localização precisa mesmo em ambientes onde ocorrem mudanças constantes, como a movimentação de obstáculos ou variações na iluminação.

Métodos tradicionais de localização, como a odometria das rodas, ou seja, uma leitura direta nas voltas das rodas para estimar sua posição, baseiam-se em estimativas sobre o movimento dos robôs, mas são suscetíveis a erros acumulativos (Veja a Subseção 2.1.2), especialmente em ambientes não estruturados, proporcionando uma navegação menos confiável e precisa. O SLAM surge como uma solução versátil e poderosa para resolver os desafios de localização enfrentados pelo transporte autônomo de cargas. A abordagem permite que o robô construa um mapa do ambiente enquanto estima sua própria posição em tempo real, usando informações sensoriais. Este método pode ser aplicado em diversos cenários, desde a navegação em armazéns até o transporte de materiais em linhas de produção, proporcionando uma localização precisa e atualizada do robô autônomo.

Os benefícios do SLAM na logística industrial são significativos, tanto na localização robusta e precisa do robô, reduzindo a probabilidade de colisões, atrasos na entrega e danos à carga, quanto na construção de um mapa do ambiente em tempo real, permitindo que o robô ajuste sua trajetória e evite obstáculos de forma eficiente, melhorando a segurança das operações.

As motivações para realizar um projeto deste são diversas e fundamentais para o

sucesso das operações de uma empresa. A busca por uma maior eficiência e produtividade, por exemplo, impulsiona a adoção de tecnologias inovadoras, como o transporte autônomo de cargas. Além disso, a preocupação com a segurança dos operadores e dos ativos presentes no ambiente industrial motiva a busca por soluções que ofereçam uma localização ideal dos robôs autônomos. A necessidade de se adaptar a ambientes dinâmicos e não estruturados também impulsiona o interesse em soluções como o SLAM, que são capazes de lidar com mudanças no ambiente em tempo real.

Apesar dos benefícios, sua aplicação em ambientes industriais pode encontrar algumas dificuldades, tais como ruído sensorial significativo, proveniente de fontes como máquinas operatrizes, empilhadeiras e outros equipamentos, que podem gerar interferências eletromagnéticas. Essas interferências podem perturbar os sinais transmitidos e recebidos pelo LIDAR, introduzindo ruídos nas leituras e comprometendo a qualidade dos dados capturados.

3.2 A Modelagem do SLAM para a Simulação Proposta

A solução proposta neste trabalho consiste na simulação de robôs AMR autônomos usando sensores LIDAR para detecção de obstáculos, usando algoritmos do ICP e do filtro de Kalman para estimativa e atualização da posição dos robôs. A aplicação desenvolvida fornece a visualização do posicionamento de um robô, tanto pelos dados obtidos do ICP quanto das estimativas obtidas pelo filtro de Kalman.

A modelagem do SLAM para a simulação proposta começa com a inicialização da matriz de transição de estado, $\vec{F}$.

$$\vec{F} = \begin{bmatrix} 1 & 0.1 & 0.005 & 0 & 0 & 0 \\ 0 & 1 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.1 & 0.005 \\ 0 & 0 & 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Os valores da matriz são obtidos sobre o intervalo de tempo 0,1 segundo entre as leitura dos pontos do LIDAR. A matriz de ruído de processo, $\vec{Q}$, definida a seguir, também depende

desse intervalo de tempo.

$$\vec{Q} = \begin{bmatrix} 0.00000025 & 0.000005 & 0.00005 & 0 & 0 & 0 \\ 0.000005 & 0.01 & 0.1 & 0 & 0 & 0 \\ 0.00005 & 0.1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.00000025 & 0.000005 & 0.00005 \\ 0 & 0 & 0 & 0.000005 & 0.01 & 0.1 \\ 0 & 0 & 0 & 0.00005 & 0.1 & 1 \end{bmatrix}$$

Já a matriz de covariância de medição, $\vec{R}$, é dada abaixo.

$$\vec{R} = \begin{bmatrix} \alpha_x^2 & 0 \\ 0 & \alpha_y^2 \end{bmatrix} = \begin{bmatrix} 225 & 0 \\ 0 & 225 \end{bmatrix}$$

A matriz de extrapolação de estado, $\vec{W}$, é inicialmente dada por $\vec{\hat{w}}_{0,0}$.

$$\vec{\hat{w}}_{0,0} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

E a matriz de observação, $\vec{H}$, da forma:

$$\vec{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Por fim, a matriz de extrapolação de covariância, $\vec{P}$, é iniciada com valor 10 metros na diagonal, já que o robô é iniciado sempre na mesma posição referente a esta distância como chute inicial mais aproximado da sua posição real no método (Veja Seção 2.3.1).

$$\vec{P}_{0,0} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

Após a inicialização das matrizes é possível simular a movimentação do robô, considerando velocidade e orientação constante, aumentando sua coordenada x de 1 unidade e a

coordenada y em $0,5$ de unidade a cada iteração. Assim, a predição realizada pelo filtro de Kalman é dada pelo cálculo das matrizes $\vec{\hat{w}}_{1,0}$ e $\vec{P}_{1,0}$.

Predição:

$$\vec{\hat{w}}_{1,0} = \vec{F} * \vec{\hat{w}}_{0,0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{P}_{1,0} = \vec{F} * \vec{P}_{0,0} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 10.10 & 1.01 & 0.05 & 0.00 & 0.00 & 0.00 \\ 1.01 & 10.10 & 1.10 & 0.00 & 0.00 & 0.00 \\ 0.05 & 1.10 & 11.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 10.10 & 1.01 & 0.05 \\ 0.00 & 0.00 & 0.00 & 1.00 & 10.10 & 1.10 \\ 0.00 & 0.00 & 0.00 & 0.05 & 1.10 & 11.00 \end{bmatrix}$$

Iteração 1

Medição:

A partir da predição inicial, a próxima etapa do processo mede a posição atual do robô pelo LIDAR no plano.

$$\vec{z}_1 = \begin{bmatrix} 1,02 \\ 0,54 \end{bmatrix}$$

Estimativa:

Na sequência as estimativas desta iteração são calculadas.

$$\vec{K}_1 = \vec{P}_{1,0} * \vec{H}^T * (\vec{H} * \vec{P}_{1,0} * \vec{H}^T + \vec{R})^{-1} = \begin{bmatrix} 0,43 & 0 \\ 0,04 & 0 \\ 0,02 & 0 \\ 0 & 0,43 \\ 0 & 0,04 \\ 0 & 0,02 \end{bmatrix}$$

O ganho de Kalman para a posição do robô, tanto para a coordenada x quanto para y, é de $0,43$, indicando que o valor da estimativa atual está distante do valor medido.

Em seguida, a estimativa do estado atual $w_{1,1}$ e a estimativa da covariância atual $P_{1,1}$ são calculadas.

$$\vec{\hat{w}}_{1,1} = \vec{\hat{w}}_{1,0} + \vec{K}_1 * (\vec{z}_1 - \vec{H} * \vec{\hat{w}}_{1,0}) = \begin{bmatrix} 1,04 \\ 0,04 \\ 0,02 \\ 0,51 \\ 0,06 \\ 0,12 \end{bmatrix}$$

$$\vec{P}_{1,1} = (\vec{I} - \vec{K}_1\vec{H})\vec{P}_{1,0}(\vec{I} - \vec{K}_1\vec{H})^T + \vec{K}_1\vec{R}\vec{K}_1^T = \begin{bmatrix} 6,01 & 1,05 & 0,55 & 0 & 0 & 0 \\ 1,05 & 11,25 & 2,34 & 0 & 0 & 0 \\ 0,55 & 2,34 & 12,62 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6,01 & 1,05 & 0,55 \\ 0 & 0 & 0 & 1,05 & 11,25 & 2,34 \\ 0 & 0 & 0 & 0,55 & 2,34 & 12,62 \end{bmatrix}$$

O filtro de Kalman mostra que as coordenadas x e y no vetor $w_{1,1}$, relativas a estimativa da posição atual do robô, são de $1,04$ e $0,51$, respectivamente.

Predição:

Na primeira iteração também é preciso calcular a predição do próximo estado.

$$\vec{\hat{w}}_{2,1} = \vec{F} * \vec{\hat{w}}_{1,1} = \begin{bmatrix} 1,98 \\ 0,08 \\ 0,15 \\ 1,02 \\ 0,09 \\ 0,18 \end{bmatrix}$$

$$\vec{P}_{2,1} = \vec{F} * \vec{P}_{1,1} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 6,12 & 0,98 & 0,52 & 0 & 0 & 0 \\ 0,98 & 10,78 & 2,13 & 0 & 0 & 0 \\ 0,52 & 2,13 & 13,31 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6,12 & 0,98 & 0,52 \\ 0 & 0 & 0 & 0,98 & 10,78 & 2,13 \\ 0 & 0 & 0 & 0,52 & 2,13 & 13,31 \end{bmatrix}$$

Percebe-se que a matriz $\vec{P}_{2,1}$, que realiza a predição da covariância do próximo estado, manteve seus valores bem similares em relação a sua estimativa do estado atual, dado pela

matriz $\vec{P}_{1,1}$. Essa similaridade mostra que a confiança da estimativa do filtro de Kalman é alta, pois os valores obtidos logo nas primeiras iterações são muito próximos dos valores da posição inicial já conhecida do robô, ao contrário de exemplos apresentados anteriormente.

As próximas iterações consistem no cálculo iterativo das medições com base no LIDAR, bem como a estimativa da posição atual e a predição da próxima posição do robô.

Iteração 2

Medição:

$$\vec{z}_2 = \begin{bmatrix} 2,32 \\ 1,15 \end{bmatrix}$$

Estimativa:

$$\vec{K}_2 = \vec{P}_{2,1} * \vec{H}^T * (\vec{H} * \vec{P}_{2,1} * \vec{H}^T + \vec{R})^{-1} = \begin{bmatrix} 0,41 & 0 \\ 0,04 & 0 \\ 0,02 & 0 \\ 0 & 0,41 \\ 0 & 0,04 \\ 0 & 0,02 \end{bmatrix}$$

$$\vec{\hat{w}}_{2,2} = \vec{\hat{w}}_{2,1} + \vec{K}_2 * (\vec{z}_2 - \vec{H} * \vec{\hat{w}}_{2,1}) = \begin{bmatrix} 2,03 \\ 0,06 \\ 0,12 \\ 1,01 \\ 0,04 \\ 0,11 \end{bmatrix}$$

$$\vec{P}_{2,2} = (\vec{I} - \vec{K}_2\vec{H})\vec{P}_{2,1}(\vec{I} - \vec{K}_2\vec{H})^T + \vec{K}_2\vec{R}\vec{K}_2^T = \begin{bmatrix} 6,23 & 1,02 & 0,54 & 0 & 0 & 0 \\ 1,02 & 10,71 & 2,15 & 0 & 0 & 0 \\ 0,54 & 2,15 & 13,13 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6,23 & 1,02 & 0,54 \\ 0 & 0 & 0 & 1,02 & 10,71 & 2,15 \\ 0 & 0 & 0 & 0,54 & 2,15 & 13,13 \end{bmatrix}$$

Predição:

$$\vec{\hat{w}}_{3,2} = \vec{F} * \vec{\hat{w}}_{2,2} = \begin{bmatrix} 2,97 \\ 0,04 \\ 0,09 \\ 1,49 \\ 0,02 \\ 0,06 \end{bmatrix}$$

$$\vec{P}_{3,2} = \vec{F} * \vec{P}_{2,2} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 6,15 & 1,06 & 0,59 & 0 & 0 & 0 \\ 1,06 & 9,67 & 2,02 & 0 & 0 & 0 \\ 0,59 & 2,02 & 12,91 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6,15 & 1,06 & 0,59 \\ 0 & 0 & 0 & 1,06 & 9,67 & 2,02 \\ 0 & 0 & 0 & 0,59 & 2,02 & 12,91 \end{bmatrix}$$

Iteração 100

Após a centésima iteração, o resultado das matrizes de medição, estimativa e predição são dadas a seguir.

Medição:

$$\vec{z}_{100} = \begin{bmatrix} 92,02 \\ 46,25 \end{bmatrix}$$

Estimativa:

$$\vec{K}_{100} = \vec{P}_{100,99} * \vec{H}^T * (\vec{H} * \vec{P}_{100,99} * \vec{H}^T + \vec{R})^{-1} = \begin{bmatrix} 0,24 & 0 \\ 0,03 & 0 \\ 0,01 & 0 \\ 0 & 0,24 \\ 0 & 0,03 \\ 0 & 0,01 \end{bmatrix}$$

$$\vec{\hat{w}}_{100,100} = \vec{\hat{w}}_{100,99} + \vec{K}_{100} * (\vec{z}_{100} - \vec{H} * \vec{\hat{w}}_{100,99}) = \begin{bmatrix} 101,37 \\ -2,41 \\ -0,01 \\ 50,30 \\ 6,22 \\ 0,15 \end{bmatrix}$$

$$\vec{P}_{100,100} = (\vec{I} - \vec{K}_{100}\vec{H})\vec{P}_{100,99}(\vec{I} - \vec{K}_{100}\vec{H})^T + \vec{K}_{100}\vec{R}\vec{K}_{100}^T$$

$$= \begin{bmatrix} 4,12 & 0,87 & 0,35 & 0 & 0 & 0 \\ 0,87 & 4,92 & 1,46 & 0 & 0 & 0 \\ 0,35 & 1,46 & 8,91 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4,12 & 0,87 & 0,35 \\ 0 & 0 & 0 & 0,87 & 4,92 & 1,46 \\ 0 & 0 & 0 & 0,35 & 1,46 & 8,91 \end{bmatrix}$$

Predição:

$$\hat{\vec{w}}_{101,100} = \vec{F} * \hat{\vec{w}}_{100,100} = \begin{bmatrix} 102,33 \\ -2,32 \\ -0,02 \\ 50,84 \\ 6,83 \\ 0,13 \end{bmatrix}$$

$$\vec{P}_{101,100} = \vec{F} * \vec{P}_{100,100} * \vec{F}^T + \vec{Q} = \begin{bmatrix} 4,15 & 0,97 & 0,43 & 0 & 0 & 0 \\ 0,97 & 4,64 & 1,39 & 0 & 0 & 0 \\ 0,43 & 1,39 & 9,28 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4,15 & 0,97 & 0,43 \\ 0 & 0 & 0 & 0,97 & 4,64 & 1,39 \\ 0 & 0 & 0 & 0,43 & 1,39 & 9,28 \end{bmatrix}$$

As matrizes resultantes $\vec{K}$ e $\vec{P}$ mantiveram valores próximos ao longo das iterações de 2 a 100, reduzindo minimamente seus valores. Este fato mostra que os cálculos de estimativa e predição do filtro são coerentes desde as primeiras iterações, resultando apenas em pequenos ajustes nos valores das matrizes. Além disso, o resultado mostra que o modelo se adequou ao problema proposto, de forma rápida e eficaz, a partir dos parâmetros iniciais estipulados. Logo, podemos observar que a simulação proposta para o movimento do robô nos eixos x e y com 1 e 0,5 pixels, respectivamente, por iteração, com velocidade constante, após cem iterações de leituras do LIDAR, estimativas e predições, está de acordo com o comportamento do modelo.

3.3 Implementação do SLAM para a Simulação Proposta

Esta seção apresenta a estrutura de código dos algoritmos elaborados para o SLAM num ambiente simulado, bem como as bibliotecas e funções implementadas para o monitoramento dos robôs na simulação desenvolvida.

3.3.1 Estrutura de Código e Bibliotecas

A estrutura de código da aplicação é organizada por funções implementadas com o auxílio de várias bibliotecas que facilitam o desenvolvimento da simulação.

A primeira função da solução tem o papel de detectar obstáculos usando o LIDAR. A função deve percorrer todos os feixes do LIDAR verificando se existe intersecção entre um feixe e o obstáculo. Caso exista uma intersecção, a função retorna essa coordenada como um obstáculo encontrado.

A próxima função está relacionada ao cálculo da distância de um obstáculo até o centro do LIDAR. A função recebe a coordenada de um obstáculo encontrado e calcula a distância entre o obstáculo e o centro do LIDAR usando o teorema de Pitágoras. Com estas distâncias calculadas, a função do LIDAR pode então armazenar e retornar a nuvem de pontos atual para o algoritmo do ICP.

Uma outra função é responsável pela implementação do algoritmo do ICP (Algoritmo 2) que compara a nuvem de pontos atual retornada pelo LIDAR com a nuvem de pontos da iteração anterior, calculando a translação e rotação entre as nuvens. Os dados calculados são somados ao atual posicionamento do robô de acordo com o ICP, que por sua vez são repassados como argumento ao filtro de Kalman, a função que implementa o Algoritmo 3, onde a estimativa e a predição são realizadas em relação ao centro do robô.

Python [40] foi a linguagem escolhida para implementar o SLAM para robôs AMR, devido à sua popularidade, legibilidade, ampla disponibilidade de bibliotecas, e fácil implementação. As bibliotecas utilizadas foram *pygame*, *matplotlib*, *numpy*, *shapely* e *pcl*, além das bibliotecas do *Python* como a *sys*, *math*, *threading* e *time*.

A biblioteca *Pygame* [41] é utilizada para criar a interface gráfica da simulação, permitindo a exibição e interação de elementos visuais em tempo real. Portanto, a biblioteca fornece ferramentas para criar janelas, desenhar formas e gerenciar eventos de entrada, essenciais para a visualização do movimento do robô e detecção de colisões com obstáculos.

A *Matplotlib* [42] é empregada na visualização de dados, para plotar pontos vermelhos que representam as interseções detectadas pelo sensor LIDAR durante a simulação, os obstáculos que limitam a área de locomoção e sensoriamento do robô. Essa biblioteca é am-

plamente utilizada em *Python* para criar gráficos e visualizações de dados de forma simples e eficaz.

Já a *Numpy* [43] é uma biblioteca fundamental para computação científica em *Python*, usada na manipulação de matrizes e cálculos numéricos essenciais para a implementação de algoritmos como o filtro de Kalman e o método de registro de nuvens dos pontos do ICP.

Shapely [44] é uma biblioteca usada para cálculos geométricos, verificando interseções entre linhas a partir da função de intersecção entre retas, essencial para simular o comportamento do sensor LIDAR ao detectar obstáculos no ambiente.

A *Pcl* [45] é a biblioteca responsável pelas operações em nuvens de pontos, essencial para a implementação do algoritmo ICP, que busca alinhar duas nuvens de pontos para estimar o movimento do robô entre leituras do LIDAR e retornar sua localização.

Sys é uma biblioteca que fornece acesso a algumas variáveis usadas ou mantidas pelo interpretador *Python* e a funcionalidades relacionadas ao sistema. A biblioteca suporta a manipulação de argumentos em linha de comando e controla a execução do programa de acordo com as entradas do usuário. Math fornece funções matemáticas para operações comuns, como cálculos trigonométricos, logarítmicos e aritméticos. Nesta trabalho é usada para realizar cálculos relacionados à posição e orientação do robô, bem como para calcular distâncias e ângulos necessários para o funcionamento do sensor LIDAR.

Threading é uma biblioteca que fornece suporte para *threads* em *Python*. As *threads* permitem que funções sejam executadas em paralelo, realizando a simulação do sensor LIDAR, juntamente com a movimentação do robô e seu controle a partir do teclado. Por fim, a biblioteca *time* fornece várias funções relacionadas ao tempo, tais como medir o tempo decorrido entre iterações, suspender a execução por um certo período e obter informações sobre o tempo do sistema. Neste código, a biblioteca é usada principalmente para adicionar atrasos entre as iterações do programa principal, controlando a taxa de atualização da simulação, do intervalo de leituras e das atualizações do LIDAR e ICP para o filtro de Kalman, evitando que a CPU seja sobrecarregada e mantendo a ordem correta de execução das funções de acordo com seu respectivo período.

3.3.2 Funções-Chave da Solução

As principais funções implementadas para a simulação são: *get_lidar_obstacles*; *calculate_lidar_distances*; *ICP*; e *Kalman_Filter*. Tais funções são responsáveis por armazenar as intersecções encontradas pelo LIDAR, calcular as distâncias dos obstáculos de acordo com essas intersecções, realizar a medição com o ICP a partir das nuvens de pontos geradas pelas distâncias dos obstáculos e fornecer os parâmetros de medição para o filtro de Kalman esti-

mar a posição do robô, respectivamente. Os detalhes de implementação que suportam toda a aplicação, tais como definição de variáveis e visualização gráfica, bem como as classes e as funções, estão descritos nos Apêndices A, B e C, respectivamente.

A função responsável pela leitura dos pontos obtidos pelo LIDAR é descrito pelo Código 3.1.

Código 3.1 – Função get_lidar_obstacles

```
1  def get_lidar_obstacles():
2    ini_lidar_dot = (robot.lidar_x, robot.lidar_y)
3    i = 0
4    for angle in range(-180, 181, 15):
5      for obstacles_index in range(0, len(obstacles), 2):
6        verify_intersec = robot.lidar_intersection(obstacles[obstacles_index],
             obstacles[obstacles_index + 1], ini_lidar_dot, (robot.lidar_x + 1000
             * math.cos(math.radians(robot.angle + angle)), robot.lidar_y - 1000 *
              math.sin(math.radians(robot.angle + angle))))
7        end_lidar_dot = (verify_intersec[1], verify_intersec[2])
8        if verify_intersec[0]:
9          last_intersections[i] = [verify_intersec[1], verify_intersec[2]]
10     i = i + 1
```

Após a definição da função na linha 1, a leitura dos dados do LIDAR inicia na linha 2 definindo a variável *ini_lidar_dot* como o ponto central do LIDAR nesta iteração. A linha 3 define a variável i que indexa a matriz de intersecções que é atualizada quando um obstáculo é encontrado para cada feixe do LIDAR. A linha 4 calcula o ângulo de cada feixe do LIDAR, a cada 15 graus, a partir de -180° até 180°, realizando uma volta completa de 360 graus e simulando a angulação de 25 feixes do LIDAR.

A linha 5 consulta as coordenadas de todos os objetos definidos na simulação e verifica se algum feixe se intersecta com tais coordenadas, resultando em um obstáculo encontrado. A linha 6 verifica se o feixe atual do LIDAR, encontrado na linha 4, possui alguma intersecção com o obstáculo atual, encontrado na linha 5. Na linha 7 uma variável é então definida para armazenar uma intersecção entre o obstáculo e o feixe. Já a linha 8 verifica se uma intersecção foi encontrada. Caso ocorra uma intersecção, a matriz de obstáculos é atualizada na linha 9. A linha 11 então atualiza a variável i para avaliar a próxima linha da matriz de obstáculos, repetindo este processo até obter todas as intersecções encontradas pelos feixes.

Já a função *calculate_lidar_distances*, descrita no Código 3.2, recebe a leitura atual de obstáculos encontrados pelo LIDAR, calcula a distância relativa ao LIDAR, repassa estes parâmetros de distância da leitura atual e anterior de obstáculos para a função ICP (Código 3.3), e então atualiza a leitura anterior com os dados da atual.

Código 3.2 – Função calculate_lidar_distances

```
1  def calculate_lidar_distances(robot):
2    global last_intersections
3    global last_points
4    verify = False
5    while True:
6      time.sleep(0.1)
7      points_out = np.zeros((25, 3), dtype=np.float32)
8      for i in range(25):
9        delta_x = last_intersections[i][0] - robot.lidar_x
10       delta_y = last_intersections[i][1] - robot.lidar_y
11       distance = math.sqrt(delta_x**2 + delta_y**2)
12       angle_rad = math.atan2(delta_y, delta_x)
13       x = distance * math.cos(angle_rad)
14       y = distance * math.sin(angle_rad)
15       points_out[i] = [x, y, 0]
16     if verify:
17       ICP(points_out, last_points, robot)
18     last_points = points_out
19     verify = True
```

A função *calculate_lidar_distances* inicialmente declara as variáveis *last_intersections*, que salva as últimas intersecções encontradas pelos feixes do LIDAR no plano cartesiano, e *last_points*, que salva as intersecções encontradas pelos feixes do LIDAR da iteração anterior referente à distância encontrada pelo LIDAR. A linha 4 define a variável $verify$ como $False$, que tem como objetivo verificar se é a primeira iteração dessa função no decorrer da simulação. Caso seja verdade, a função armazena apenas a leitura atual, e caso seja falsa, a função retorna a última leitura como sendo da iteração anterior e atualiza a leitura atual, enviando essas nuvens de pontos para a função *ICP*. A linha 5 executa as iterações da simulação até que o usuário deseje encerrá-la. Na linha 6 um intervalo de 0.1 segundo é definido para que o restante do código seja executado conforme as respectivas leituras do LIDAR, garantindo que tais leituras sejam distintas entre as iterações, fazendo com que o ICP realize seus cálculos de forma eficiente.

Na linha 7 a matriz *points_out* é inicializada com tamanho de acordo com os 25 feixes do LIDAR, conforme as leituras a cada 15 graus, por 3 dimensões, x, y e z, onde z tem sempre valor 0 indicando o modelo num plano bidimensional, para salvar os pontos de intersecção do LIDAR na iteração atual. A linha 8 então atualiza a variável i, responsável por armazenar a linha atual da matriz de intersecções, para calcular a distância entre o LIDAR e o obstáculo. As linhas 9 e 10 então calculam a diferença entre as coordenadas x e y em relação à intersecção do feixe atual com a posição do LIDAR no robô (*delta_x* e *delta_y*,

respectivamente). Com isso, a linha 11 calcula a distância entre os pontos através do teorema de Pitágoras e a linha 12 calcula o ângulo em radianos usando a função arco-tangente entre os deltas. Nas linhas 13 e 14 as coordenadas x e y referentes ao posicionamento do LIDAR são calculadas através da distância e do ângulo obtidos nas linhas 11 e 12.

Na linha 15 a coordenada transformada é então copiada para a matriz *points_out* na linha i. A linha 16 verifica se é a primeira iteração desta função no decorrer da simulação, e então não há duas nuvens de pontos para serem comparadas no ICP. Caso contrário, a nuvem de pontos da iteração atual juntamente com a nuvem da iteração anterior são enviadas como argumento para a função ICP na linha 17. A linha 18 atualiza a nuvem de pontos da iteração anterior com os dados da atual e a linha 19 atribui $True$ para a variável $verify$, indicando que a função já foi executada ao menos uma vez.

A função *ICP*, chamada na linha 19 da função de cálculo do LIDAR, é descrita no Código 3.3. Essa função compara as duas nuvens de pontos, da iteração anterior e atual, para retornar a translação e rotação estimada entre essas nuvens usando a biblioteca *pcl*.

Código 3.3 – Função ICP

```
1  def ICP(points_in, points_out, robot):
2    global ICP_x
3    global ICP_y
4    cloud_in = pcl.PointCloud()
5    cloud_out = pcl.PointCloud()
6    cloud_in.from_array(points_in)
7    cloud_out.from_array(points_out)
8    icp = cloud_in.make_IterativeClosestPoint()
9    converged, transf, estimate, fitness = icp.icp(cloud_in, cloud_out)
10   translation_2d = transf[:2, 3]
11   rotation_matrix = transf[:2, :2]
12   rotation_angle = math.atan2(rotation_matrix[1, 0], rotation_matrix[0, 0]) *
         (180.0 / math.pi)
13   if(abs(rotation_angle) < 0.1):
14     ICP_x = ICP_x + (translation_2d[0])
15     ICP_y = ICP_y + (translation_2d[1])
16   Kalman_Filter(ICP_x, ICP_y, robot)
```

A função recebe as duas nuvens de pontos como argumento para calcular a posição do robô representada pelas variáveis ICP_x e ICP_y definidas nas linhas 2 e 3. As linhas 4 e 5 criam as nuvens de acordo com a biblioteca *Pcl* como matrizes de 3 colunas x, y e z, enquanto que as linhas 6 e 7 preenchem as matrizes com os valores dos vetores *points_in* e *points_out* recebidos como argumento. Já na linha 8 a variável *icp* recebe o resultado da translação e rotação entre as nuvens de pontos, de acordo com a função *make_IterativeClosestPoint*

da biblioteca *Pcl*, calculados na linha 9, resultando na matriz de transformação homogênea através da função $icp.icp(cloud_in, cloud_out)$, também da biblioteca *Pcl*. Essa função permite a extração da translação 2D na linha 10, sua matriz de rotação na linha 11 e seu ângulo de rotação com a função arco-tangente na linha 12.

Já a linha 13 da função ICP tem o papel de verificar se a rotação entre as nuvens é maior que 0.1 graus, já que na simulação realizada o robô gira apenas em torno de seu próprio eixo. Com isso, quando o robô faz um movimento para rotacionar seu ângulo, o limiar usado é de 0.1 graus, escolhido de forma empírica baseado na observação comportamental dos resultados do ICP, que não apresentou rotações maiores que 0.1 graus em movimentos retilíneos. Dessa forma, nenhuma translação ocorre quando o robô realiza um giro em seu próprio eixo, sendo possível ignorar dados imprecisos do ICP que atualizariam as coordenadas medidas de maneira equivocada. Caso a rotação seja menor que 0.1 grau, as coordenadas do ICP são atualizadas nas linhas 14 e 15 e as medições do posicionamento atual do robô nos eixos x e y são passadas para a função do filtro de Kalman na linha 16.

Por fim, a função da simulação do filtro de Kalman é descrito pelo Código 3.4.

Código 3.4 – Função Kalman_Filter

```
1  def Kalman_Filter(ICP_x, ICP_y, robot):
2    global W
3    global P
4    global F
5    global Q
6    global R
7    global H
8    Z = [[0 for j in range(2)] for i in range(1)]
9    Z[0][0] = Kalman_x + dist_x
10   Z[0][1] = Kalman_y + dist_y
11   K = np.dot(np.dot(P, np.transpose(H)), np.linalg.inv(np.dot(np.dot(H, P), np
         .transpose(H)) + R))
12   W = W + np.dot(K, np.transpose(Z) - np.dot(H, W))
13   P = np.dot(np.dot((np.identity(6) - np.dot(K, H)), P), np.transpose(np.
         identity(6) - np.dot(K, H))) + np.dot(np.dot(K, R), np.transpose(K))
14   Kalman_x = W[0][0]
15   Kalman_y = W[3][0]
16   W = np.dot(F, W)
17   P = np.dot(np.dot(F, P), np.transpose(F)) + Q
```

A função recebe inicialmente os parâmetros da translação nas coordenadas x e y provenientes do ICP em *dist_x* e *dist_y*. A linha 8 constrói a matriz $\vec{Z}$, que representa a medição atual da posição do robô de acordo com o sensor, enquanto que as linhas 9 e 10

atribuem à Z a posição atual em relação ao ICP.

O cálculo da estimativa realizada pelo filtro ocorre na linha 11, usando a biblioteca *Numpy* para calcular a multiplicações de matrizes, suas transpostas, inversas e matriz identidade com as funções $np.dot, np.transpose, np.linalg.inv$ e $np.identity$, respectivamente, atualizando o ganho de Kalman. A linha 12 atualiza o vetor de extrapolação de estado e a linha 13 atualiza a matriz de transição de estado, ambos referentes as matrizes $\vec{w}$ e $\vec{P}$ do filtro de Kalman, que representam a posição atual da estimada e a da covariância quadrática de incerteza da estimativa. As linhas 14 e 15 atualizam o valor estimado da posição do robô em relação ao filtro de Kalman nas posições 0 e 3 da matriz W, que representam as coordenadas x e y da matriz $\vec{w}$. Já a predição do filtro é realizada na linha 16 atualizando a matriz de extrapolação de estado, enquanto que a linha 17 atualiza a matriz de transição de estado, também referentes as matrizes $\vec{w}$ e $\vec{P}$ do filtro de Kalman.

4 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta os resultados da aplicação do filtro de Kalman na estimativa e predição de trajetórias dos robôs AMR num ambiente simulado. Os testes foram realizados num mesmo ambiente simulado, mantendo os mesmos obstáculos e as mesmas velocidades linear e angular do robô, alterando apenas o seu trajeto. Ao todo foram realizados três experimentos com duração de quinze minutos cada. O primeiro realiza um trajeto de uma reta, variando sua coordenada apenas no eixo x. O segundo realiza um trajeto no formato de um retângulo, enquanto o terceiro realiza um trajeto no formato de um triângulo.

O ambiente simulado se trata de um espaço com 800 pixels de largura (x) e 600 pixels de altura (y), onde existem obstáculos simulando paredes nos quatro cantos desta área, ou seja, os obstáculos refletem as coordenadas de $(0, 0)$ até $(800, 0)$, $(0, 500)$ até $(800, 500)$, $(0, 0)$ até $(0, 500)$ e $(800, 0)$ até $(800, 500)$. A simulação trata um robô com um LIDAR de 25 feixes acoplado, que se move com velocidade constante de 10 pixels por segundo, de acordo com um movimento específico pré-determinado para cada experimento. Os resultados apresentados representam a variação das coordenadas x e y medidas pelo ICP, estimadas pelo filtro de Kalman e reais do robô, no decorrer do tempo de execução da simulação em determinados trechos. O objetivo é demonstrar o comportamento dos algoritmos utilizados em diferentes situações, realizando análises afim de mensurar a proximidade das estimativas do filtro ao valor real em decorrência das medições derivativas do ICP.

4.1 Simulando o Trajeto de uma Reta

A primeira simulação realiza o percurso de uma reta, onde o robô inicia nas coordenadas $(100, 300)$ e percorre em linha reta até as coordenadas $(700, 300)$. Ao chegar nas coordenadas $(700, 300)$ o robô então gira sobre seu próprio eixo, até que esteja alinhado com o ponto inicial, e volta a andar em linha reta até que chegue no ponto $(100, 300)$, realizando outro giro para se alinhar com o ponto final repetindo o processo ao longo de quinze minutos de execução. A Figura 9 representa as coordenadas x e y reais do robô (em vermelho), a medição feita pelo ICP (em azul) e a estimativa feita pelo filtro de Kalman (em verde), durante apenas dois minutos de simulação.

Neste cenário, os valores do ICP não se distanciaram significativamente do valor real do robô. A média da diferença entre os valores reais do robô, comparados com os valores medidos do ICP e estimados pelo filtro de Kalman, está 27 pixels distantes daqueles medidos pelo ICP, enquanto das estimativas do filtro de Kalman está em torno de apenas 3 pixels.

Figura 9 – Resultado das medições do ICP, filtro de Kalman e da posição real do robô ao longo do teste no trajeto de reta

Esta simulação mostra que o comportamento dos dados do ICP permaneceram igualmente constante, com relação às coordenadas reais ro robô, durante toda a execução da simulação, ou seja, quando o robô passa por uma coordenada específica a medição do ICP é sempre muito próxima. Por exemplo, quando o robô passa pela coordenada $(300, 300)$ a medição do ICP é sempre próxima de $(300, 300)$, do mesmo jeito que quando passa pela coordenada $(700, 300)$ a medição do ICP é sempre próxima de $(670, 300)$. Dessa forma, as medições do ICP foram praticamente as mesmas durante a execução, permitindo que as estimativas do filtro de Kalman ficassem muito mais próximas do valor real da posição do robô ao longo das iterações, visto que a variação entre medições é muito baixa, implicando em estimativas e predições do filtro mais precisas e confiáveis.

4.2 Simulando o Trajeto Retangular

Esta simulação usa o percurso de uma rota retangular, onde o robô inicia nas coordenadas $(100, 100)$ e percorre em linha reta até as coordenadas $(700, 100)$. Em seguida, o robô gira sobre seu próprio eixo para direita e se movimenta até a coordenada $(700, 500)$, e assim consecutivamente para as coordenadas $(100, 500)$ e retornando para $(100, 100)$. A Figura 10 representa as coordenadas x e y em relação a posição real do robô (em vermelho), a medição feita pelo ICP (em azul) e a estimativa feita pelo filtro de Kalman (em verde), no decorrer de dez minutos de simulação.

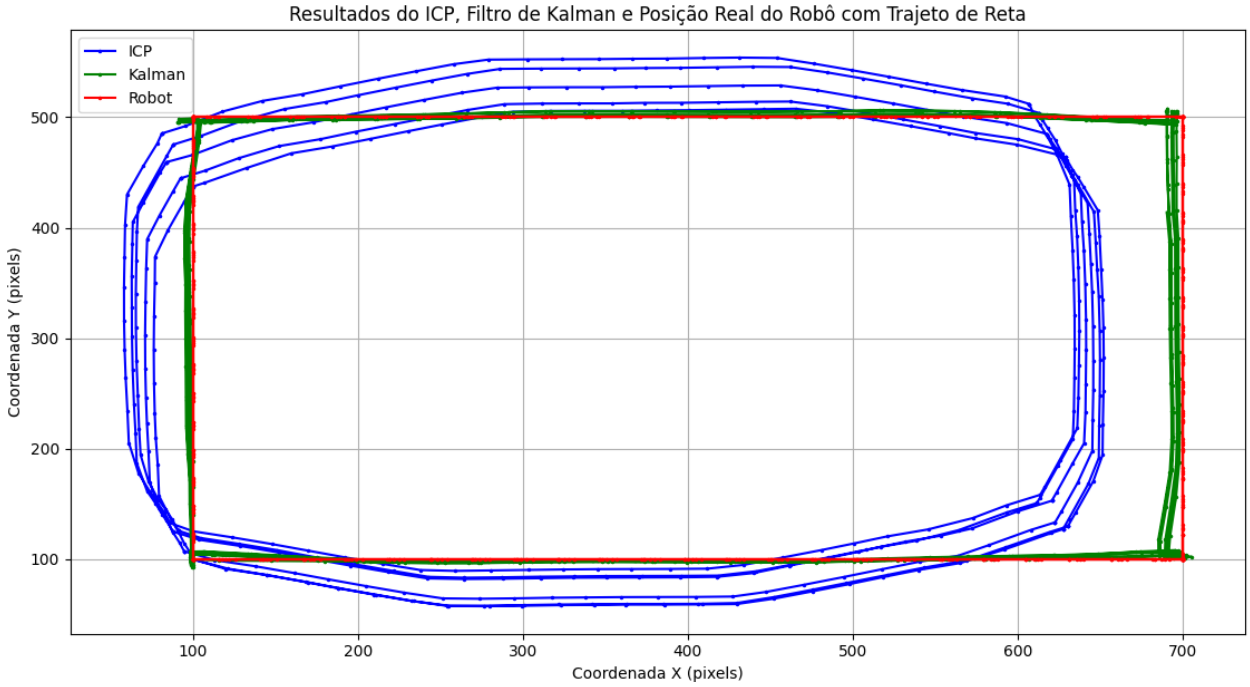

Figura 10 – Resultado das medições do ICP, filtro de Kalman e da posição real do robô ao início do teste no trajeto retangular

Neste experimento é possível observar que os valores do ICP se distanciaram de forma mais significativa em relação ao valor real do robô. Também pode-se notar que as medições não permaneceram similares para as mesmas coordenadas, por exemplo, quando o robô passa pela coordenada $(400, 500)$ as medições acabam variando de $(400, 500)$ à $(370, 560)$, aproximadamente. Neste caso, a atuação do filtro de Kalman fica mais evidente aproximando suas estimativas mais ainda da posição real do robô, mesmo com as inconsistências do modelo de medição do ICP.

Pela Figura 10 é possível ainda notar que a média da diferença entre os valores reais e, os valores medidos pelo ICP e estimados pelo filtro de Kalman, está em 73 pixels e 8 pixels, respectivamente, uma diferença de quase dez vezes entre a medição do ICP e o estimado pelo filtro. Logo, as estimativas e predições do filtro estão mais distantes do valor real se comparado com a simulação em linha reta, pois as medições neste simulação foram mais imprecisas. No entanto, vale ressaltar que os dados convergem de forma satisfatória com relação ao posicionamento real do robô.

A Figura 11 representa uma parte da simulação do percurso retangular, onde o robô realiza um trajeto retilíneo da coordenada $(160, 100)$ até $(100, 100)$, gira em torno do seu próprio eixo e efetua outro trajeto retilíneo até a coordenada $(100, 200)$. Este intervalo de tempo selecionado, com duração de dez minutos, ajuda a elucidar e tornar mais visual a diferença entre as medidas, do ICP e dos cálculos do filtro de Kalman neste trecho. Nota-se que

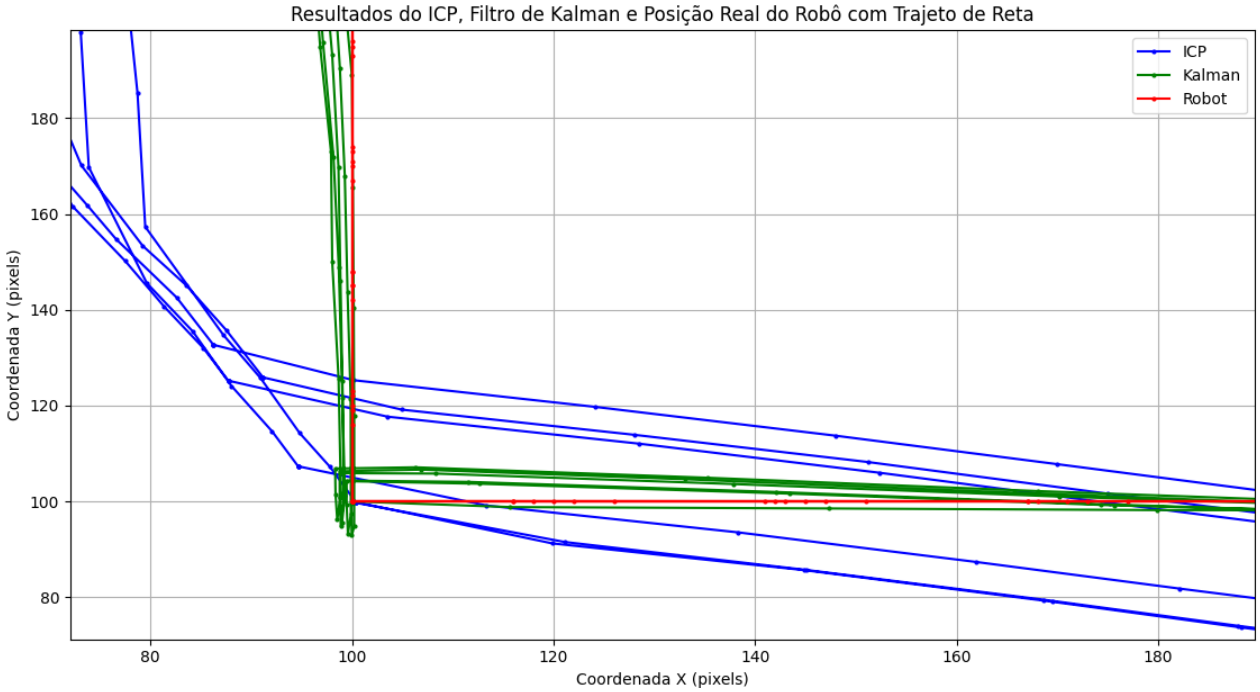

Figura 11 – Resultado das medições do ICP, filtro de Kalman e da posição real do robô ao final do teste no trajeto retangular das coordenadas $(500, 100)$ até $(430, 100)$

a medição do ICP neste intervalo oscila significativamente (até 20 pixels de diferença) tanto no eixo y quanto no eixo x, ou seja, os resultados se distanciam do real conforme a leitura em questão. Outro ponto a se destacar é sobre a interpretação do ICP para esta rota, onde na curva realizada pelo robô, com a rotação sobre o próprio eixo na coordenada $(100, 100)$, não ocorre translação. Ainda assim, de acordo com os resultados do ICP, a movimentação foi diferente, não indicando uma curva sobre o próprio eixo, mas sim uma curva acentuada, fazendo com que os valores de x diminuam mais que o desejado e os valores de y aumentem antes da curva ser efetivamente realizada. Essas irregularidades nas medições acabam prejudicando os resultados do filtro de Kalman, com estimativas menos precisas.

4.3 Simulando o Trajeto Triangular

Por fim, o último experimento realiza a simulação de um percurso em rota triangular, desde as coordenadas $(100, 500)$ até as coordenadas $(680, 500)$, dessas até as coordenadas $(400, 100)$, e então finalizando em $(100, 500)$. A Figura 12 representa as coordenadas x e y sobre a posição real do robô (em vermelho), a medição feita pelo ICP (em azul) e a estimativa calculada pelo filtro de Kalman (em verde), ao longo de dez minutos de simulação. Novamente este intervalo foi selecionado afim de tornar mais visual os resultados obtidos.

Nesta simulação pode-se observar que as medições do ICP se distanciaram de forma

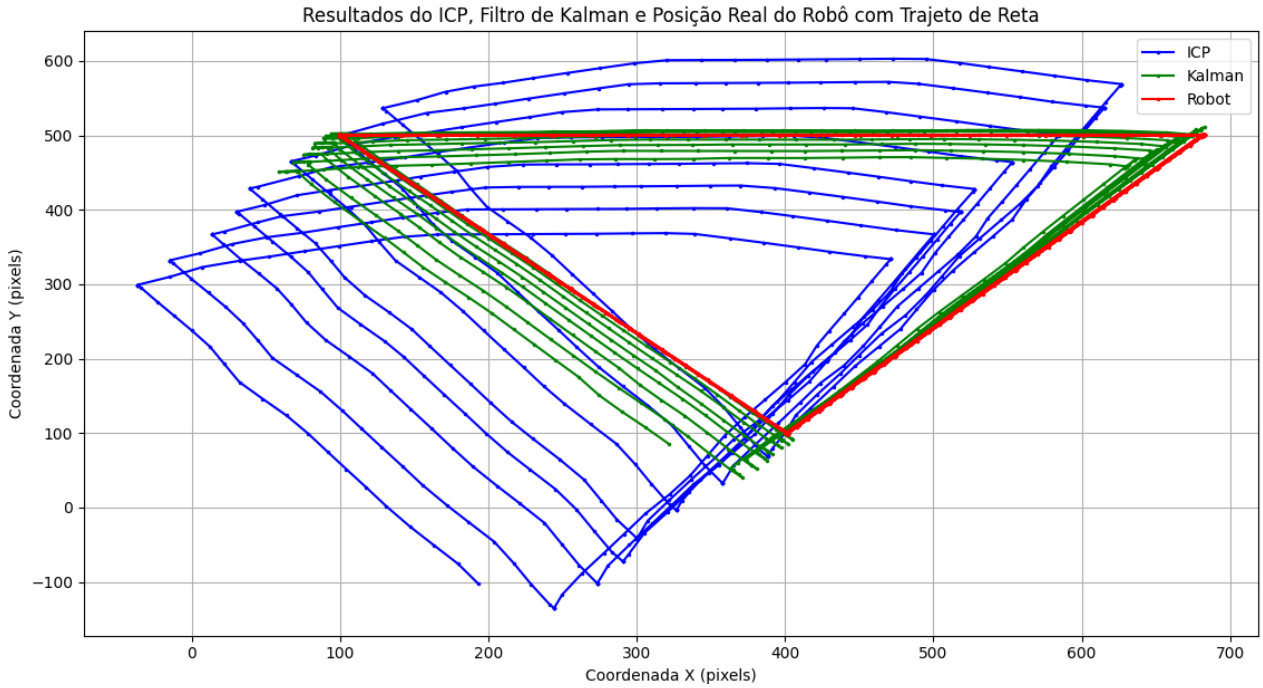

Figura 12 – Resultado das medições do ICP, filtro de Kalman e da posição real do robô ao início do teste no trajeto triangular

constante e contínua, onde a cada leitura do ICP ambos os valores de x e y vão diminuindo conforme o robô passa pela mesma coordenada. Assim, é possível identificar um padrão de comportamento nas medições, permitindo que o filtro de Kalman tenha estimativas e predições mais consistentes. Porém, como o erro das medições aumentaram durante esta execução os resultados do filtro não permaneceram coerentes com os valores reais, tornando as estimativas muito imprecisas.

O resultado desse experimento, comparado com os resultados obtidos nas trajetórias em linha reta e retangular, mostra que a inconsistência das medições provenientes do algoritmo ICP, distanciando gradativamente do valor real, eventualmente torna as estimativas do filtro de Kalman mais imprecisas. Observe que as medições desta simulação estão mais distantes em comparação com as simulações anteriores. Nas outras duas execuções as medições resultavam num erro médio de 30 a 70 pixels, enquanto que neste caso o erro está em torno de 250 pixels. Assim, o erro se torna tão grande que impossibilita o filtro de Kalman estimar de forma precisa o posicionamento do robô.

5 CONCLUSÃO

Os experimentos realizados com a simulação das trajetórias (reta, retangular e triangular) mostraram que o algoritmo do ICP, usado para obter medições de deslocamento e orientação do robô, demostrou resultados mais consistentes na simulação da rota linear e menos consistentes na simulação da rota triangular, mesmo em ambientes com obstáculos fixos e velocidade constante do robô. Essas variações se devem as maiores mudanças de direção e orientação da trajetória simulada. Estas rotas resultam em nuvens de pontos com mais rotações entre as iterações, e não apenas translações, implicando em imprecisões nos cálculos realizados pelo algoritmo em determinados casos.

No trajeto em linha reta, por exemplo, onde o robô vai e volta entre dois pontos, a medição numa determinada coordenada é sempre próxima das outras medições, apresentando certa constância nos resultados. Em contrapartida, na rota triangular, as medições do ICP não conseguiram retornar sempre um mesmo valor próximo de uma determinada coordenada, implicando em medições cada vez mais distantes da posição real. Tal comportamento impacta negativamente no funcionamento do filtro de Kalman, uma vez que o modelo não consegue se ajustar a um padrão definitivo, o que implicaria em uma maior precisão nos cálculos.

Apesar do impacto que o uso do ICP causa, o filtro de Kalman se demonstrou capaz de corrigir as imprecisões do ICP de forma geral e gerar estimativas mais próximas da posição real do robô. Em trajetos mais simples, como uma reta, onde o comportamento do ICP é mais constante e previsível, o filtro foi capaz de manter suas estimativas próximas do valor real ao longo do tempo. Entretanto, nas situações onde as medições do ICP divergem significativamente da realidade ao longo do tempo, por exemplo em trajetos mais complexos com curvas acentuadas, modificando substancialmente a leitura de um sensor LIDAR entre iterações, o filtro de Kalman enfrenta dificuldades em convergir as estimativas de forma precisa, resultando em diferenças substanciais entre as estimativas e a posição real do robô, como visto na simulação de uma rota triangular.

De qualquer forma, mesmo considerando leituras mais imprecisas nas medições do ICP provenientes do sensor LIDAR, o filtro de Kalman se mostrou confiável e coerente em seus resultados com relação ao posicionamento real do robô, demonstrando uma eficiência quase dez vezes maior do que a própria medição.

Vale ressaltar que experimento proposto neste trabalho apenas um sensor LIDAR foi simulado e, portanto, apenas uma fonte de medições sobre o posicionamento. Ainda assim, o SLAM, de uma forma geral, foi capaz de localizar o robô autônomo de forma bastante

precisa. A alta precisão do filtro de Kalman implementado no SLAM então demonstrou uma boa capacidade de correção de erros de localização ao longo do tempo, especialmente quando o modelo de medição consegue retornar dados mais precisos ou com um comportamento constante. Dessa forma, a solução consegue garantir uma localização confiável do robô, até mesmo em ambientes dinâmicos.

Porém, existem limitações sobre o SLAM, onde as medições distinguem muito além de uma faixa razoável, e as estimativas são prejudicadas. Algumas sugestões para melhorar ainda mais a precisão dessas estimativas são: aumentar o número de sensores, aumentando assim a quantidade de informação que o SLAM pode utilizar e, consequentemente, aumentando a variedade de medições, incluindo diferentes parâmetros para o filtro de Kalman. Por exemplo, ao adicionar um sensor de odometria [46] é possível calcular as medições baseadas nas voltas de cada roda do robô resultando em mais uma coordenada medida [47].

Além disso, podem ser considerados algoritmos para prevenção de erro nas medições, tal como um controle dos comandos de movimentação enviados ao robô [48], onde de acordo com sua posição e orientação atual, ao receber um comando de movimentação, é possível prever um determinado resultado para cada iteração do SLAM. Assim, ao se mover em linha reta com velocidade e orientação constante, deve-se esperar que, em determinado tempo, o robô esteja em determinada posição, para que não se dependa apenas de sensores externos.

REFERÊNCIAS

[1] ABANAY, A.; MASMOUDI, L.; ANSARI, M. E. A calibration method of 2D LIDAR-Visual sensors embedded on an agricultural robot. *Optik*, Elsevier, v. 249, p. 168254, 2022.

[2] ALEXANDROV, S. *PointCloudLibrary pcl*. 1. ed. [S.l.]: github, 2013.

[3] ESCOBAR, C. S. Distribuciones de probabilidad. *RPubs*, RPubs, p. 36, 2016.

[4] BECKER, A. *Kalman Filter from the Ground Up*. 1. ed. [S.l.]: 2Checkout, 2023. ISBN 978-965-598-439-2.

[5] BRYSON, J.; WINFIELD, A. Standardizing ethical design for artificial intelligence and autonomous systems. *Computer*, IEEE, v. 50, n. 5, p. 116–119, 2017.

[6] OTA, J. Multi-agent robot systems as distributed autonomous systems. *Advanced engineering informatics*, Elsevier, v. 20, n. 1, p. 59–70, 2006.

[7] HA, Q.; YEN, L.; BALAGUER, C. Robotic autonomous systems for earthmoving in military applications. *Automation in Construction*, Elsevier, v. 107, p. 102934, 2019.

[8] AULINAS, J. et al. The SLAM problem: a survey. *Artificial Intelligence Research and Development*, IOS Press, p. 363–371, 2008.

[9] GRISETTI, G. et al. A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, IEEE, v. 2, n. 4, p. 31–43, 2010.

[10] ROBINSON, E. A.; TREITEL, S. Principles of digital Wiener filtering. *Geophysical Prospecting*, European Association of Geoscientists & Engineers, v. 15, n. 3, p. 311–332, 1967.

[11] KALMAN, R. E. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, ASME, v. 82, n. 1, p. 35, 1960. Disponível em: ⟨http://dx.doi.org/10.1115/1.3662552⟩.

[12] HUANG, S.; DISSANAYAKE, G. Convergence and consistency analysis for extended Kalman filter based SLAM. *IEEE Transactions on robotics*, IEEE, v. 23, n. 5, p. 1036–1049, 2007.

[13] HUANG, G. P.; MOURIKIS, A. I.; ROUMELIOTIS, S. I. Analysis and improvement of the consistency of extended kalman filter based slam. In: IEEE. *2008 IEEE International Conference on Robotics and Automation*. [S.l.], 2008. p. 473–479.

[14] SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 4, p. 56–68, 1986.

[15] DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, IEEE, v. 13, n. 2, p. 99–110, 2006.

[16] CHATILA, R. Robot mapping: An introduction. *Robotics and cognitive approaches to spatial mapping*, Springer, p. 9–12, 2008.

[17] MOSHAYEDI, A. J.; JINSONG, L.; LIAO, L. AGV (automated guided vehicle) robot: Mission and obstacles in design and performance. *Journal of Simulation and Analysis of Novel Technologies in Mechanical Engineering*, Islamic Azad University, Khomeinishahr Branch, v. 12, n. 4, p. 5–18, 2019.

[18] SAYERS, C.; SAYERS, C. *Remote control robotics*. [S.l.]: Springer, 1999.

[19] RIBEIRO, M. I. Kalman and extended Kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, v. 43, n. 46, p. 3736–3741, 2004.

[20] THRUN, S. Particle filters in robotics. In: CITESEER. *UAI*. [S.l.], 2002. v. 2, p. 511–518.

[21] SCHUSTER, F. et al. Landmark based radar SLAM using graph optimization. In: IEEE. *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.], 2016. p. 2559–2564.

[22] ENGELHARD, N. et al. Real-time 3D visual SLAM with a hand-held RGB-D camera. In: *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*. [S.l.: s.n.], 2011. v. 180, p. 1–15.

[23] NAM, D. V.; GON-WOO, K. Solid-state LiDAR based-SLAM: A concise review and application. In: IEEE. *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*. [S.l.], 2021. p. 302–305.

[24] LI, A. et al. From timing variations to performance degradation: Understanding and mitigating the impact of software execution timing in slam. In: IEEE. *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2022. p. 13308–13315.

[25] HSU, C.-P. et al. A review and perspective on optical phased array for automotive LiDAR. *IEEE Journal of Selected Topics in Quantum Electronics*, IEEE, v. 27, n. 1, p. 1–16, 2020.

[26] FANG, H.-T.; HUANG, D.-S. Noise reduction in lidar signal based on discrete wavelet transform. *Optics Communications*, Elsevier, v. 233, n. 1-3, p. 67–76, 2004.

[27] KANEKO, M. et al. Mask-SLAM: Robust feature-based monocular SLAM by masking using semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. [S.l.: s.n.], 2018. p. 258–266.

[28] EZRA, E.; SHARIR, M.; EFRAT, A. On the performance of the ICP algorithm. *Computational Geometry*, Elsevier, v. 41, n. 1-2, p. 77–93, 2008.

[29] KIM, Y.; BANG, H. Introduction to Kalman filter and its applications. *Introduction and Implementations of the Kalman Filter*, IntechOpen London, UK, v. 1, p. 1–16, 2018.

[30] SAMAN, A. B. S. H.; LOTFY, A. H. An implementation of SLAM with extended Kalman filter. In: IEEE. *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*. [S.l.], 2016. p. 1–4.

[31] GREWAL, M. S.; ANDREWS, A. P. Applications of Kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems Magazine*, IEEE, v. 30, n. 3, p. 69–78, 2010.

[32] NAGLA, S. 2d hector slam of indoor mobile robot using 2d LiDAR. In: IEEE. *2020 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*. [S.l.], 2020. p. 1–4.

[33] BALASURIYA, B. et al. Outdoor robot navigation using Gmapping-based SLAM algorithm. In: IEEE. *2016 Moratuwa Engineering Research Conference (MERCon)*. [S.l.], 2016. p. 403–408.

[34] XU, B. et al. Research of cartographer laser SLAM algorithm. In: SPIE. *LIDAR Imaging Detection and Target Recognition 2017*. [S.l.], 2017. v. 10605, p. 49–57.

[35] CIVERA, J. et al. 1-Point RANSAC for extended Kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of field robotics*, Wiley Online Library, v. 27, n. 5, p. 609–631, 2010.

[36] YOKOZUKA, M. et al. LiTAMIN: LiDAR-based tracking and mapping by stabilized ICP for geometry approximation with normal distributions. In: IEEE. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2020. p. 5143–5150.

[37] ELKHRACHY, I. *Towards an Automatic Registration for Terrestrial Laser Scaner Data*. Tese (Doutorado), 02 2008.

[38] EZRA, E.; SHARIR, M.; EFRAT, A. On the ICP algorithm. In: *Proceedings of the twenty-second annual symposium on Computational geometry*. [S.l.: s.n.], 2006. p. 95–104.

[39] LIAQAT, A. et al. Autonomous mobile robots in manufacturing: Highway code development, simulation, and testing. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 104, p. 4617–4628, 2019.

[40] Python Software Foundation. *Python*. ⟨https://www.python.org/⟩. Acessado em 10 de março de 2024.

[41] Pygame Community. *Pygame*. ⟨https://www.pygame.org/⟩. Acessado em 10 de março de 2024.

[42] The Matplotlib development team. *Matplotlib*. ⟨https://matplotlib.org/⟩. Acessado em 10 de março de 2024.

[43] NumPy Teams. *Numpy*. ⟨https://numpy.org/⟩. Acessado em 10 de março de 2024.

[44] Python Community. *Shapely*. ⟨https://pypi.org/project/shapely/⟩. Acessado em 10 de março de 2024.

[45] PCL Community. *python-pcl*. ⟨https://pointclouds.org/⟩. Acessado em 10 de março de 2024.

[46] CHONG, K. S.; KLEEMAN, L. Accurate odometry and error modelling for a mobile robot. In: IEEE. *Proceedings of International Conference on Robotics and Automation*. [S.l.], 1997. v. 4, p. 2783–2788.

[47] GANGANATH, N.; LEUNG, H. Mobile robot localization using odometry and kinect sensor. In: IEEE. *2012 IEEE International Conference on Emerging Signal Processing Applications*. [S.l.], 2012. p. 91–94.

[48] MOSHAYEDI, A. J.; LI, J.; LIAO, L. Simulation study and pid tune of automated guided vehicles (agv). In: IEEE. *2021 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. [S.l.], 2021. p. 1–7.

Apêndices

APÊNDICE A – INICIALIZAÇÃO

A seguir estão descritas as inicializações de constantes e variáveis globais, que são essenciais para armazenar informações importantes que serão utilizadas ao longo da execução do programa. As constantes representam parâmetros fixos, como as dimensões da tela e cores utilizadas no ambiente gráfico, enquanto as variáveis globais armazenam informações dinâmicas, como as coordenadas do robô, parâmetros dos filtros de Kalman e dados relacionados ao processamento do LIDAR. Sendo assim, a definição de constantes e variáveis globais é de acordo com o trecho de código do Código A.1.

Código A.1 – Código da inicialização das variáveis Init_Vars

```
1  def Init_Vars():
2    width, height = 800, 600
3    white = (255, 255, 255)
4    red = (255, 0, 0)
5    green = (0, 255, 0)
6    blue = (0, 0, 255)
7    black = (0, 0, 0)
8    F = [[0 for j in range(6)] for i in range(6)]
9    Q = [[0 for j in range(6)] for i in range(6)]
10   R = [[0 for j in range(2)] for i in range(2)]
11   H = [[0 for j in range(6)] for i in range(2)]
12   W = [[0 for j in range(1)] for i in range(6)]
13   P = [[0 for j in range(6)] for i in range(6)]
14   last\_intersections = np.zeros((25, 2), dtype=float)
15   last\_points = np.zeros((25, 3), dtype=np.float32)
16   ICP\_x = 0
17   ICP\_y = 0
18   Kalman\_x = 0
19   Kalman\_y = 0
20   obstacles = [(0, 0), (799, 0), (799, 0), (799, 599), (799, 599), (0, 599),
         (0, 599), (0, 0)]
21   intersections = np.zeros((800, 600), dtype=int)
```

Neste trecho, a linha 1 define as dimensões da tela para visualização, sendo as proporções de 800 pixeis horizontais e 600 pixeis verticais. Nas linhas 2 à 6 são definidos os valores RGB para as cores utilizadas. Nas linhas 7 à 12 são inicializadas as matrizes $\vec{F}$, $\vec{Q}$, $\vec{R}$, $\vec{H}$, $\vec{W}$ e $\vec{P}$, e posteriormente definidas de acordo com o que foi descrito na Seção 3.2. A linha 13 inicializa um vetor que irá armazenar as coordenadas dos obstáculos encontrador pelo LIDAR na iteração atual e a linha 14 inicializa um vetor que irá armazenar as coordena-

das dos obstáculos encontrador pelo LIDAR na iteração anterior, para que ambos os vetores sejam comparados no algoritmo do ICP. As linhas 15 à 18 inicializam as coordenadas (x e y) referentes aos cálculos e estimativas realizados pelo ICP e pelo filtro de Kalman. A linha 19 define uma matriz que armazena as coordenadas dos obstáculos que serão identificados pelo LIDAR e posteriormente passados ao ICP. Por fim a linha 20 armazena todas as intersecções encontradas pelo LIDAR para com os obstáculos definidos na linha anterior, para que possam ser demonstrados em um gráfico no plano cartesiano quando o usuário finalizar a simulação.

Em sequência, é realizada a definição do ambiente gráfico, que é feita utilizando a biblioteca Pygame, isso inclui a definição das dimensões da tela, a criação da janela onde a simulação será renderizada e a definição de título para a janela. Este trecho de código, referente ao Código A.2, permite criar uma interface visual onde a simulação pode ser exibida e interagida pelo usuário.

Código A.2 – Código da função Define_view

```
1  def Define_view():
2    pygame.init()
3    screen = pygame.display.set_mode((width, height))
4    pygame.display.set_caption('SLAM Simulation')
```

Onde a linha 1 inicializa a biblioteca Pygame no ambiente python da simulação, a linha 2 descreve o tamanho da tela gráfica, de acordo com os valores definidos no Código A.1 e a linha 3 define o título da tela como "SLAM Simulation".

Por fim são descritos os objetos visuais da simulação, utilizando a biblioteca pygame, de acordo com o Código A.3.

Código A.3 – Código da função Visual_Environment

```
1  def Visual_Environment():
2    screen.fill(white)
3    rotation_surface = pygame.Surface((2 * robot.radius, 2 * robot.radius),
         pygame.SRCALPHA)
4    pygame.draw.circle(rotation_surface, robot.color, (robot.radius, robot.
         radius), robot.radius)
5    rotation_surface = pygame.transform.rotate(rotation_surface, robot.angle)
6    rotation_rect = rotation_surface.get_rect(center=(robot.x, robot.y))
7    screen.blit(rotation_surface, rotation_rect.topleft)
8    pygame.draw.circle(screen, robot.lidar_color, (int(robot.lidar_x), int(
         robot.lidar_y)), robot.lidar_radius)
9    pygame.draw.circle(screen, black, (int(ICP_x), int(ICP_y)), robot.lidar_
         radius)
10   pygame.draw.circle(screen, green, (int(Kalman_x), int(Kalman_y)), robot.
         lidar_radius)
```

```
11    pygame.draw.line(screen, black, obstacles[0], obstacles[1], 2)
12    pygame.draw.line(screen, black, obstacles[2], obstacles[3], 2)
13    pygame.draw.line(screen, black, obstacles[4], obstacles[5], 2)
14    pygame.draw.line(screen, black, obstacles[6], obstacles[7], 2)
15    pygame.display.flip()
16    pygame.time.Clock().tick(30)
```

O código começa com a linha 1 definindo o fundo da tela como branco. A linha 2 cria uma superfície de rotação que representa o robô, definindo o tamanho sendo seu diâmetro, e a linha 3 desenha efetivamente o formato do robô circular juntamente com sua cor e tamanho. A linha 4 define a rotação do desenho de acordo com o ângulo do robô e as linhas 5 e 6 realizam essa rotação. A linha 7 desenha o círculo do LIDAR de acordo com o posicionamento dele na classe Robot. As linhas 8 e 9 desenham círculos menores correspondentes ao posicionamento do centro do robô de acordo com as estimativas do ICP (linha 8) e do filtro de Kalman (linha 9). As linhas 10 a 13 desenham as paredes obstáculos de acordo com a matriz global *obstacles* descrita anteriormente no Código A.1. A linha 14 atualiza a tela visual e a linha 15 define que as atualizações serão feitas a trinta quadros por segundo.

APÊNDICE B – CLASSES

A seguir é descrita a classe principal que armazena os dados do robô. Ela que representa as estruturas do robô dividida em quatro partes, sendo a inicialização de sua posição, orientação e cor, a função de movimento, a função que atualiza a posição do LIDAR de acordo com o posicionamento atual do robô e a função que representa as intersecções do LIDAR com os obstáculos. Começando com a inicialização dos valores do robô de acordo com o Código B.1

Código B.1 – Código da classe Robot

```
1  class Robot:
2    def __init__(self, x, y):
3      self.x = x
4      self.y = y
5      self.angle = 0
6      self.color = blue
7      self.radius = 25
8      self.lidar\_color = black
9      self.lidar\_radius = radius / 5
10     self.move\_up = False
11     self.move\_down = False
12     self.move\_left = False
13     self.move\_right = False
```

Neste trecho a linha 1 inicia a classe Robot que representa a sua estrutura e a linha 2 define a função de inicialização do robô, onde os argumentos necessários para sua chamada são a posição inicial do robô (x e y). Nas linhas 3 à 7 são definidos os valores de posição, orientação, cor e raio do robô nesta ordem. As linhas 8 e 9 definem a cor e o raio do LIDAR respectivamente e as linhas 10 a 13 definem as variáveis de movimento do robô, sendo elas que descrevem a movimentação atual do robô.

A próxima função dentro do escopo da classe Robot determina como se dá a movimentação do robô, sua translação e rotação, de acordo com o trecho de código no Código B.2.

Código B.2 – Código da função Move

```
1  def Move(self):
2    if self.move\_up:
3      self.x = self.x + 1 * math.cos(math.radians(self.angle))
4      self.y = self.y - 1 * math.sin(math.radians(self.angle))
5    elif self.move\_down:
```

```
6       self.x = self.x - 1 * math.cos(math.radians(self.angle))
7       self.y = self.y + 1 * math.sin(math.radians(self.angle))
8     elif self.move\_left:
9       self.angle += 1
10    elif self.move\_right:
11      self.angle -= 1
12    if(self.angle >= 360):
13      self.angle = self.angle - 360
14    elif(self.angle < 0):
15      self.angle = self.angle + 360
16    self.update\_lidar\_position()
```

A linha 1 do Código B.2 define a função de movimentação do robô. As linhas 2, 5, 8, 10 verificam o comando atual de movimentação do robô de acordo com o input inserido pelo usuário, comparando se o comando é para ir pra frente, trás, esquerda ou direita respectivamente. As linhas 3 e 4 atualizam suas coordenadas caso vá para frente pegando sua posição atual e somando X e subtraindo Y com o valor 1 vezes o cosseno e seno, respectivamente, do seu ângulo atual em radiano, da mesma forma como as linhas 6 e 7 porém subtraindo em X e aumentando em Y. As linhas 9 e 11 atualizam o valor do ângulo, onde o robô nesta simulação faz curvas apenas no seu próprio eixo, não necessitando assim alterar as coordenadas de seu centro neste caso. A linha 12 verifica se o ângulo alterado é maior ou igual a 360 e, caso seja, é reiniciado com o valor 0 na linha 13. A linha 14 verifica se o ângulo alterado é menor que 0 e, caso seja, é reiniciado com o valor 360 na linha 15. Por fim a linha 16 realiza a chamada da função que é encarregada de atualizar a posição do LIDAR, função essa que será explicada e apresentada a seguir.

Outra função dentro da classe Robot determina o posicionamento do LIDAR de acordo com a coordenada e orientação atual do centro do robô, de acordo com o Código B.3.

Código B.3 – Código da função Update_Lidar_Position

```
1 def Update_Lidar_Position(self):
2   lidar\_offset = int(self.radius * 0.8)
3   self.lidar\_x = self.x + lidar\_offset * math.cos(math.radians(self.angle))
4   self.lidar\_y = self.y - lidar\_offset * math.sin(math.radians(self.angle))
```

A linha 1 deste trecho define a função de posicionamento do LIDAR. A linha 2 define um deslocamento entre o centro do robô e o LIDAR, para que ele fique em uma extremidade indicando a frente do robô de forma visual. As linhas 3 e 4 atualizam seu posicionamento de acordo com a coordenada e orientação do robô, pois são tratados como dois objetos distintos, necessitando que o LIDAR se posicione com relação ao robô.

Por fim, a última função dentro da classe Robot define como são realizadas as intersecções dos feixes do LIDAR com os obstáculos, e retorna para cada feixe se encontrou um obstáculo e, caso tenha encontrado, retorna também sua posição no mapa. O Código B.4 apresenta o trecho de código referente a esta função.

Código B.4 – Código da função Lidar_Intersection

```
1 def Lidar_Intersection(self, dot\_1, dot\_2, dot\_3, dot\_4):
2   lidar\_line = LineString([(dot\_3[0], dot\_3[1]), (dot\_4[0], dot\_4[1])])
3   wall\_line = LineString([(dot\_1[0], dot\_1[1]), (dot\_2[0], dot\_2[1])])
4   intersection\_point = lidar\_line.intersection(wall\_line)
5   if intersection\_point.is\_empty or not Point(intersection\_point).is\_valid
        :
6     return (False, dot\_4[0], dot\_4[1])
7   else:
8     return (True, intersection\_point.x, intersection\_point.y)
```

A linha 1 define a função que irá retornar as intersecções encontradas pelo LIDAR, ela espera como parâmetros os dois pontos, *dot*_1 inicial e *dot*_2 final, de um obstáculo, sendo que cada obstáculo é definido como uma reta, e os dois pontos, *dot*_3 inicial e *dot*_4 final, de um feixe do LIDAR. Essa função é chamada para cada feixe do LIDAR e para cada obstáculo definido na simulação, verificando entre cada um deles se há intersecção ou não, caso exista retorna o ponto de intersecção encontrado. A linha 2 cria a reta do feixe do LIDAR e a linha 3 cria a reta do obstáculo de acordo com os pontos passados por argumento. A linha 4 cria a variável *intersection_point* que recebe o retorno da função *intersection* da biblioteca Shapely, função essa que retorna a intersecção entre duas retas, no caso o feixe do LIDAR e o obstáculo. A linha 5 verifica se essa intersecção de fato existe, se não existe a linha 6 retorna False, se existe a linha 8 retorna True juntamente com o ponto de intersecção calculado.

APÊNDICE C – FUNÇÕES

A seguir é definida a leitura dos comandos de movimentação do usuário pelo teclado, como de acordo com o Código C.1.

Código C.1 – Código da função Keyboard_Input

```
1  def Keyboard_Input():
2    running = True
3    while running:
4      for event in pygame.event.get():
5        if event.type == pygame.QUIT:
6          running = False
7        elif event.type == pygame.KEYDOWN:
8          if event.key == pygame.K\_UP:
9            robot.move\_up = True
10         elif event.key == pygame.K\_DOWN:
11           robot.move\_down = True
12         elif event.key == pygame.K\_LEFT:
13           robot.move\_left = True
14         elif event.key == pygame.K\_RIGHT:
15           robot.move\_right = True
16       elif event.type == pygame.KEYUP:
17         if event.key == pygame.K\_UP:
18           robot.move\_up = False
19         elif event.key == pygame.K\_DOWN:
20           robot.move\_down = False
21         elif event.key == pygame.K\_LEFT:
22           robot.move\_left = False
23         elif event.key == pygame.K\_RIGHT:
24           robot.move\_right = False
25     robot.move()
```

Neste código a linha 1 define a variável *running* como True, na linha seguinte ocorre um loop de execução que é repetido até que a variável *running* seja falsa. A linha 3 percorre cada evento na biblioteca pygame, neste caso, verifica se o evento for de finalizar o programa (linha 4), pressionar uma tecla (linha 6) ou soltar uma tecla (linha 15). A linha 5 troca a variável *running* para False, indicando para que o programa encerre. A linha 6 verifica se uma tecla foi pressionada e as linhas 7, 9, 11 e 13 verificam se a tecla pressionada foi alguma das direcionais para cima, baixo, esquerda e direita respectivamente, ativando o movimento do robô de acordo nas linhas 8, 10, 12 e 14. A linha 15 verifica se uma deixou de ser pressionada

e as linhas 16, 18, 20 e 22 verificam se a tecla solta foi alguma das direcionais para cima, baixo, esquerda e direita respectivamente, desativando o movimento do robô de acordo nas linhas 17, 19, 21 e 23. A linha 24 ativa a função *move*() da classe Robot que atualiza o posicionamento e orientação do robô.

Então, é definida a função principal do programa, ela que inicializa a chamada das outras funções em determinada ordem para garantir o funcionamento correto do programa, em um loop de execução com um número indeterminado de iterações (de acordo com o usuário). É dividida em três partes, sendo elas a inicialização das variáveis globais e threads do programa, a descrição dos objetos visuais da aplicação e a execução simulada do LIDAR. Começando com as variáveis globais e threads, elas que são definidas de acordo com o trecho de código do Código C.2.

Código C.2 – Código da função main

```
1  if __name__ == '__main__':
2    robot = Robot(width // 2, height // 2)
3    ICP\_x = robot.x
4    ICP\_y = robot.y
5    Kalman\_x = robot.x
6    Kalman\_y = robot.y
7    W[0][0] = robot.x
8    W[3][0] = robot.y
9    W = np.dot(F, W)
10   P = np.dot(np.dot(F, P), np.transpose(F)) + Q
11   lidar\_thread = threading.Thread(target=calculate\_lidar\_distances, args=(
         robot,), daemon = True)
12   lidar\_thread.start()
```

A parte inicial da função principal começa com a linha 1 indicando para o programa que esta é, de fato, a primeira função à ser executada no programa. Na linha 2 a variável *robot* é um objeto que armazena os dados em relação a classe Robot, que passa como parâmetro sua posição inicial, neste caso no centro da tela com posicionamento em x na metade da largura da tela ($width//2$) e em y na metade da altura da tela ($height//2$). As linhas 3 a 6 inicializam as variáveis globais relativas ao posicionamento estimado do ICP e do filtro de Kalman com a posição do robô. As linhas 7 à 10 realizam a inicialização do filtro de Kalman, calculando a predição sobre as matrizes $\vec{W}$ e $\vec{P}$ (Veja Seção 2.3.2). As linhas 11 e 12 definem e inicializam, respectivamente, a thread relativa à função que calcula as distâncias dos objetos de acordo com o LIDAR.