UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA MECATRÔNICA
BACHARELADO EM ENGENHARIA MECATRÔNICA

Estudo Sobre Localização de Robôs Móveis com Rodas para Fins de Navegação Local

Delano Augusto Lucena de Medeiros Costa

Orientador: Prof. Dr. Marcelo Borges Nogueira

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Engenharia Mecatrônica.

Natal, RN, Dezembro de 2016

Estudo Sobre Localização de Robôs Móveis com Rodas para Fins de Navegação Local

Delano Augusto Lucena de Medeiros Costa

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Engenharia Mecatrônica.

Prof. Dr. Marcelo Borges Nogueira (orientador) . . . . . ECT/UFRN

Prof. Dr. Pablo Javier Alsina . . . . . DCA/UFRN

Prof. Dr. Diogo Pinheiro Fernandes Pedrosa . . . . . DCA/UFRN

A minha mãe por todo o incentivo e suporte durante toda a minha vida acadêmica

Agradecimentos

Em primeiro lugar agradeço a Deus.

Agradeço a toda minha família, por toda a ajuda que recebir durante todos esses anos, em especial ao meu avô Manoel de Afro *(in menorian)* e ao meu Tio Geraldo.

Agradeço a minha mãe Maria Daguia e minhã irmã Tâmara Stephanie pelo apoio incondicional.

Aos amigos que fiz durante a vida e no Curso, em especial a todos do DCA pelos inúmeros projetos juntos.

Aos professores que tive durante toda a minha vida.

Ao meu orientador, Professor Dr. Marcelo Borges Nogueira, sou grato pela orientação.

Ao Professor Dr. Pablo Javier Alsina por todo o suporte oferecido pela coordenação do curso.

A Gutemberg Santiago por todos os ensinamentos no laboratório de robótica do DCA.

E a todo o quadro de funcionários do DCA da UFRN.

"Por ser estreita a senda - eu não declino,
Nem por pesada a mão que o mundo espalma;
Eu sou o senhor de meu destino;
Eu sou o comandante de minha alma."
(William Ernest Henley)

Resumo

Este trabalho tem como objetivo realizar um estudo sobre localização de robôs móveis, uma área importante para a navegação em robótica, muitas vezes utilizada em conjunto com técnicas de mapeamento, para que seja possível realizar a Localização e Mapeamento Simultâneos (SLAM). O trabalho foi realizado inicialmente em simulações com o MATLAB e em seguida realizando experimentos com uma plataforma baseada em Arduino, com os resultados obtidos comprovando o funcionamento do sistema sendo demonstrados. O problema consiste em um robô móvel inserido em um ambiente em que ele conhece a sua posição inicial e as posições dos obstáculos desse local, dessa forma com o auxílio de sensores ele será capaz de se locomover nesse ambiente sabendo a sua localização com base na sua localização anterior e na localização dos obstáculos identificados. Para realizar essa navegação, o robô precisa processar os dados obtidos nos sensores, por isso optou-se por utilizar o Filtro de Kalman Estendido (EKF), que é um método matemático que produz estimativas dos valores reais de uma grandeza a partir de medições realizadas ao longo de um período, as quais podem conter ruídos. Trata-se de um método que realiza essas estimativas a cada iteração, comparando os dados medidos pelos sensores com os resultados esperados, fazendo assim uma atualização aproximada da localização do robô.

Palavras-chave: SLAM, EKF, Localização, Mapeamento, MATLAB, Navegação, Robótica Móvel, Arduino.

Abstract

This work aims to perform a study on the location of mobile robots, used for navigation in mobile robotics, often used together with mapping techniques, so that it is possible to perform the Simultaneous Localization and Mapping (SLAM). Initially applying in simulations with MATLAB and then performing practical tests, with results obtained proving the operation of the system being demonstrated. The problem consists of a mobile robot inserted in an environment that he knows its initial position and the positions of the obstacles of that place, so with the help of sensors, it will be able to move around in this environment, knowing its location based on its previous location and the location of the obstacles identified. In order to perform this navigation, the robot needs to process the data obtained in the sensors, so we opted to use the Extended Kalman Filter (EKF), which is a mathematical method that produces estimates of the actual values of measurements performed over a period, which may contain noise. It is a method that performs these estimates at each iteration, comparing the measured data in the sensors with the expected results, thus it makes an approximate update of the robot location.

Keywords: SLAM, EKF, Location, Mapping, MATLAB, Navigation, Mobile Robotics, Arduino.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Nos últimos anos vem ocorrendo um grande avanço na área da robótica móvel. Com o desenvolvimento de hardwares e softwares cada vez mais potentes, esses sistemas robóticos vem a cada ano ficando mais robustos, realizando tarefas que antes seriam apenas realizadas por humanos. Esses robôs possuem a tarefa de navegar e interagir com o ambiente em que eles estão inseridos, usando para isso sensores e atuadores. Para que um robô seja capaz de movimentar-se de forma autônoma em um ambiente, é necessário utilizar alguma técnica de navegação para que não ocorra nenhum tipo de problema nessa interação. Dessa forma, o presente trabalho foi proposto para realizar o estudo sobre uma técnica que resolva o problema da localização, usando para isso o Filtro de Kalman Estendido. Utilizando essa técnica em conjunto com sensores conectados ao robô, o mesmo será capaz de localizar os obstáculos presentes no ambiente, assim os identificando e realizando a tarefa de saber onde ele se encontra atualmente nesse ambiente, mesmo que o seu ponto de partida não tenha sido definido de forma precisa.

1.1 Objetivo

Este trabalho tem como objetivo realizar um estudo sobre localização de robôs móveis e como aplicá-la para realizar simulações e testes em situações reais. Para isso, serão introduzidos Landmarks (obstáculos) no ambiente para que ele possa investigar e processar os dados obtidos com os sensores utilizando o Filtro de Kalman Estendido, de forma a realizar a aproximação entre os valores lidos e os esperados, diminuindo assim os possíveis erros de leitura.

1.2 Motivação

Com o avanço cada vez maior da tecnologia e consequentemente da robótica móvel, ficam ainda mais em evidência atividades realizadas por robôs autônomos, como as tarefas que devem ser feitas em locais que são de difícil acesso para humanos, por exemplo. Para isso, é necessário desenvolver formas cada vez mais robustas de navegação autônoma, de forma que esses robôs sejam capazes de irem onde sejam solicitados e realizem a tarefa desejada. Com base nisso, realizou-se este estudo com o objetivo de demonstrar para o

público em geral, de uma maneira simples, uma forma de utilizar a Localização como ponto de partida para realizar esse tipo de atividade.

1.3 Organização do Trabalho

Este documento foi dividido nos seguintes capítulos e assuntos:

- No capítulo 2 será apresentada uma explicação geral sobre Localização, Mapeamento e SLAM;
- No capítulo 3 irá ser demonstrado o motivo da utilização e uma abordagem teórica sobre o Filtro de Kalman Estendido;
- No capítulo 4 será mostrado como foram realizadas as simulações, assim como o desenvolvimento das ferramentas utilizadas;
- O capítulo 5 traz os testes experimentais e resultados;
- O capítulo 6 será sobre a conclusão e a perspectiva de trabalhos futuros.

Capítulo 2

Localização, Mapeamento e SLAM

Neste capítulo será apresentado uma visão geral sobre os métodos de navegação na robótica móvel, mostrando o problema de SLAM, como ele pode ser dividido e serão citados algoritmos capazes de resolver o problema.

2.1 Localização

A posição em que se encontra o robô que navega por um determinado ambiente é a sua localização. Ela é dada com base nas informações que são obtidas do mapa e da distância para outros landmarks. É calculada geralmente utilizando a sua localização e/ou orientação anterior, as informações passadas pelo vetor de controle e a leitura obtida dos sensores. Para obter a localização de forma mais confiável, costuma-se usar as informações de sensores como sonares, sensores infravermelhos, com a odometria.

O problema da localização é exemplificado por Newman[3] da seguinte forma: "é dado um mapa **M** contendo um conjunto de obstáculos e um fluxo de observações de medições entre o veículo e os obstáculos. Partimos do princípio que as associações entre as medições e os obstáculos observados são verdadeiras. Assumimos que o veículo que estamos navegando está equipado com um sensor de distância que retorna a distância para os obstáculos". O robô é inserido no ambiente em qualquer ponto, então ele começa a navegar identificando os obstáculos através dos sensores e calculando a distância que os separa de forma que sempre terá a sua posição atual no mapa.

Podemos observar na figura 2.1 a resolução de um típico problema da localização. Os obstáculos estão representados pelos asteriscos verdes. O robô, representado pelo triângulo azul, é inserido no ambiente e faz sua primeira observação obtendo a sua localização e a incerteza dessa medição que é representada pelo círculo vermelho. Ao longo do tempo ele realiza mais observações e essa incerteza diminui cada vez mais, porém ao parar de realizar observações (esse momento é representado na figura conforme a elipse vermelha cresce) a sua incerteza volta a aumentar. Em seguida o robô volta a realizar novas observações dos obstáculos e a incerteza volta a diminuir novamente.

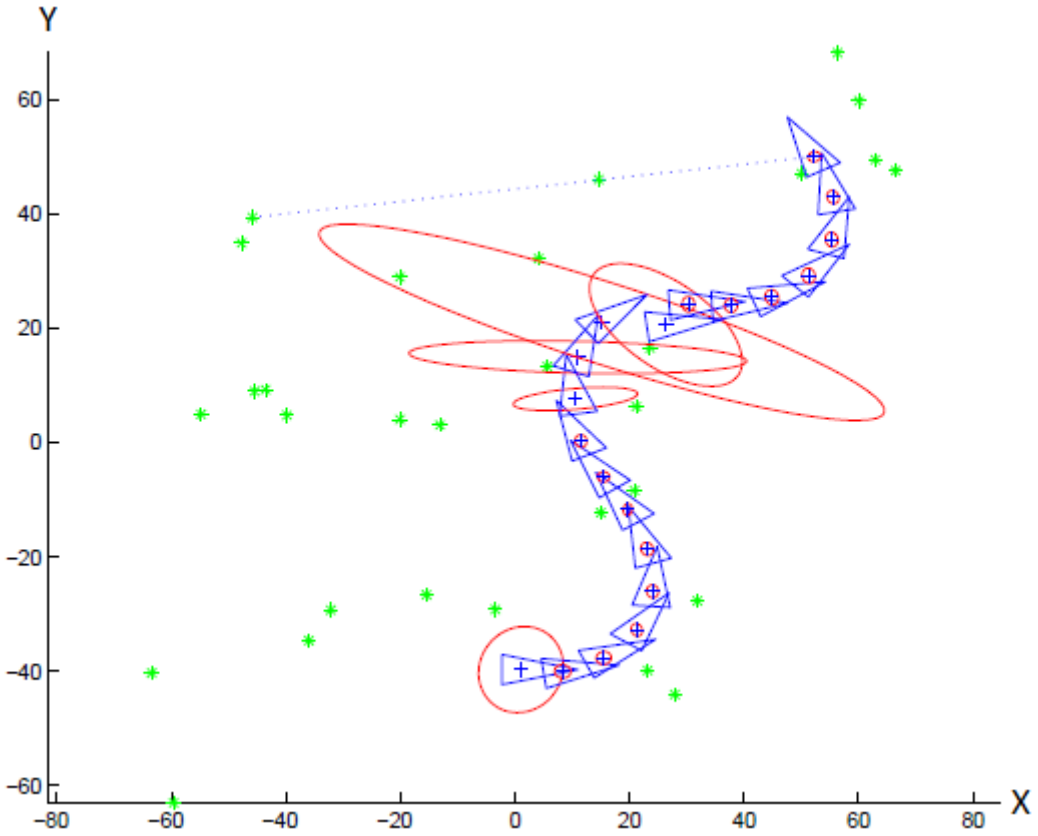

Figura 2.1: Problema da Localização: Um robô inserido em um mapa conhecido calculando sua localização ao longo do tempo.
Fonte: Disponível em EKF Based Navigation and SLAM: Background Material, Notes and Example Code[3]

2.2 Mapeamento

Mapeamento de ambientes são realizados em várias áreas como geografia, geologia, construção civil, entre outras. Essa técnica é usada quando é necessário representar uma área previamente conhecida e geralmente é feito em uma escala diferente do ambiente representado, existindo dois tipos de mapeamento que são o topológico e o métrico. Neste trabalho iremos utilizar o mapeamento métrico, que consiste na representação métrica do ambiente, onde são respeitados o tamanho e as distâncias entre os objetos inserido nele.

Podemos exemplificar o problema do mapeamento em robôs da seguinte forma: imagine um robô equipado com um GPS, de forma que mesmo que ele não tenha um mapa do local, saberá a sua localização aproximada. Portanto, a partir da sua localização atual ele fará leituras dos seus sensores e passará a identificar os obstáculos e a partir disso dará início a construção do mapa. A figura 2.2 demonstra o problema, onde os asteriscos verdes representam os obstáculos. O veículo é inserido em um ambiente desconhecido onde ele conhece a sua localização. Com isso ele navega pelo ambiente e realiza observações dos obstáculos para formar o mapa do ambiente, que de início possuem grandes

incertezas representadas pelas elipses pretas. Quanto mais ele navegar por esse ambiente, mais observações irão ser realizadas e as incertezas desse mapa irão diminuir.

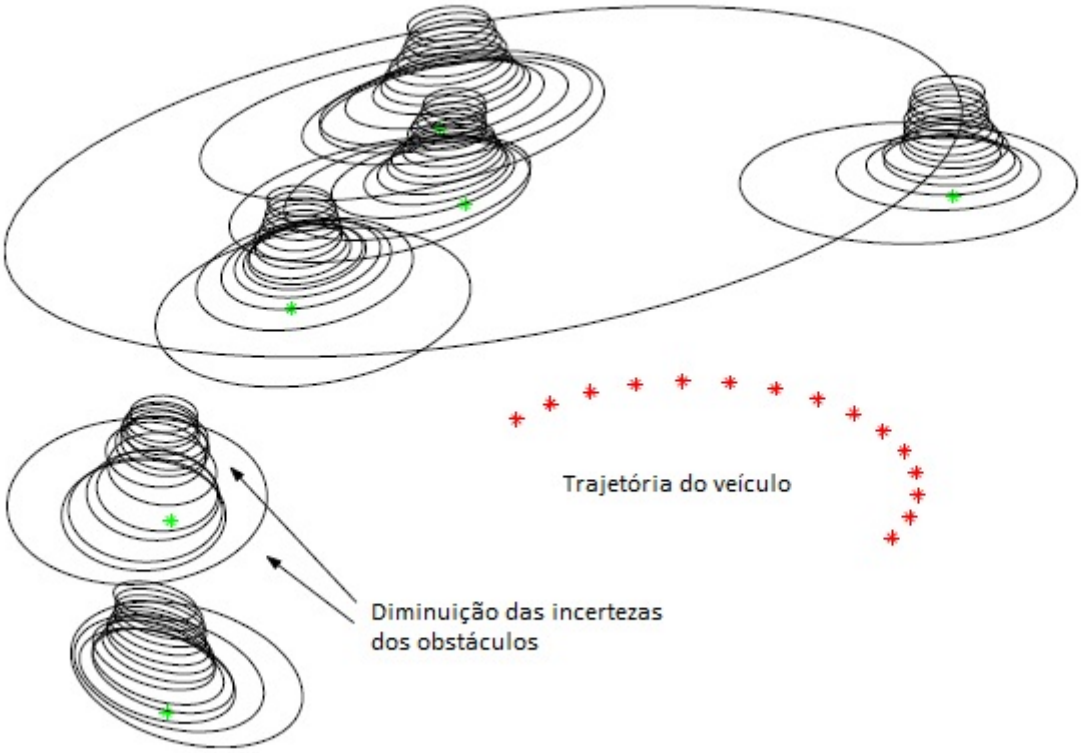

Figura 2.2: Problema do mapeamento: Um robô inserido em um ambiente desconhecido, mas sabendo sua localização realizando o mapeamento.
Fonte: Disponível em EKF Based Navigation and SLAM: Background Material, Notes and Example Code[3]

2.3 SLAM

A navegação de robôs é um problema difícil de ser resolvido e é definido por Latombe[1] da seguinte forma: "A construção autônoma de modelo é uma questão fundamental na robótica móvel. Basicamente o problema é: depois de inserido em um ambiente desconhecido, um robô, ou uma equipe de robôs, deve executar operações com os sensores em vários locais e integrar os dados adquiridos em uma representação do ambiente. No entanto, na prática, este problema acaba por ser difícil. Primeiro, é preciso escolher uma representação adequada do ambiente - por exemplo, mapas topológicos, layouts poligonais, grades de ocupação, modelos tridimensionais ou mapa baseados em obstáculos. Em segundo lugar, essa representação deve ser extraída das leituras imperfeitas de sensores. Finalmente, para ser verdadeiramente autônomo, o robô deve decidir por si próprio os movimentos necessários para construir o modelo".

A Localização e Mapeamento Simultâneos (Simultaneous Localization and Mapping - SLAM) é o problema que pergunta se é possível que um robô movél seja colocado em um local desconhecido que faz parte de um ambiente desconhecido e ele construa incrementalmente um consistente mapa deste ambiente, ao mesmo tempo que determina a sua localização dentro deste mapa [2]. Basicamente, ele consiste na possibilidade de um robô móvel sem nenhuma informação anterior ser capaz de movimentar-se em um ambiente e, a partir disso, ser capaz de construir um mapa com as informações obtidas. Além disso, ele deve ser capaz de saber qual a sua localização nesse mapa e assim planejar sua trajetória durante o processo de mapeamento [3]. Portanto, é um processo que ocorre de forma paralela, a medida que o robô vai obtendo informação para construir um mapa do ambiente em que ele se encontra, é necessário usar essas informações para saber em que ponto ele estar inserido nesse ambiente. Para que o veículo seja capaz de obter os dados para realizar o mapeamento e localização é necessário que ele obtenha isso do ambiente. Ou seja, o veículo deverá ser equipado com atuadores e sensores, para que seja capaz de encontrar e identificar obstáculos, se locomover e coletar as informações de odometria geradas e processar todos esses dados.

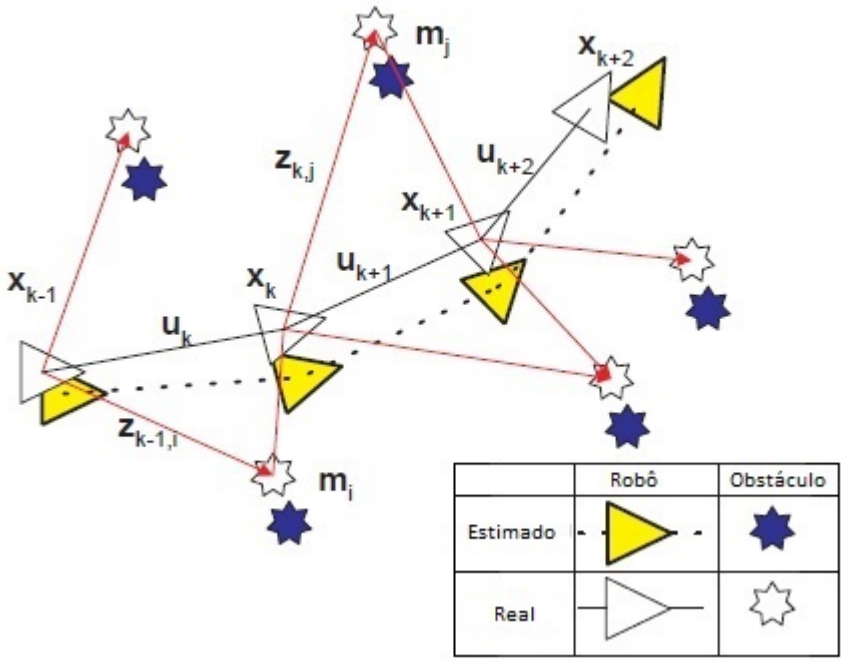

Figura 2.3: Problema de SLAM: k é o tempo de execução, x_k é a posição atual do robô, u_k é o vetor de controle aplicado em k-1, m_i é a posição atual do i-ésimo landmark e $z_{k,i}$ é a distância entre o robô e o i-ésimo landmark.

Fonte: Durrant-Whyte e Bailey [2]

Podemos definir o processo da seguinte forma: o veículo móvel é colocado em um ambiente desconhecido, sem nenhuma informação prévia. Em seguida, ele faz uma busca por obstáculos pelo local para obter informações deles como, por exemplo, distância e orientação e assim ir formando o mapa. Com base nas informações obtidas dos obstáculos

ele encontra sua própria localização nesse mapa e por fim se movimenta no ambiente e volta a procurar obstáculos reiniciando o ciclo. Esse exemplo pode ser conferido na figura 2.3 de Durrant-Whyte e Bailey [2], onde é feita a comparação entre as posições reais (branco) e estimadas (amarelo e azul).

2.4 Método Desenvolvido neste Trabalho

Para uma boa introdução no estudo sobre navegação, foi escolhido para ser desenvolvido neste trabalho uma resolução para o problema da localização, já que o esforço computacional utilizado para esse problema não é grande e por apresentar uma menor complexidade em relação ao SLAM. O grande problema encontrado neste trabalho foi a identificação dos obstáculos que ainda não funciona de forma satisfatória, porém no futuro com a adição de melhores ferramentas para fazer essa identificação, o SLAM poderá ser implementado.

2.5 Algoritmos Utilizados para SLAM

Ao longo do tempo foram desenvolvidos alguns algoritmos com base no filtro de Bayes para que fosse possível utilizar o SLAM em robôs móveis. Dentre eles podemos citar o Extended Information Form SLAM (EIF SLAM), Filtro de partículas, Filtro de Kalman e o Filtro de Kalman Estendido[16]. Abordaremos sobre uma variação bastante utilizada do filtro de Bayes no próximo capítulo, o Filtro de Kalman Estendido (EKF), algoritmo utilizado nesse trabalho que foi escolhido por ser simples de aplicar e funcionar bem, tanto para localização como mapeamento, e por consequência no SLAM também. Apesar de exigir um grande esforço computacional conforme o número de landmarks aumenta, esse algoritmo é muito utilizado por apresentar bons resultados e foi desenvolvido a partir do Filtro de Kalman, pois ele só trabalha com sistemas lineares.

2.6 Considerações Finais

Ao longo desse capítulo, vimos uma explicação sobre o que é SLAM e quais os problemas que precisam ser resolvidos para que a técnica seja aplicada em um veículo. Costuma-se dividir o problema em dois (localização e mapeamento) como foi apresentado para que sejam entendidos e resolvidos de forma isolada e assim a sua aplicação seja feita de forma mais rápida, pois a dificuldade é bem maior quando é utilizado o SLAM em vez da localização ou mapeamento isolados. Ao final do capítulo podemos notar a importância do tema na área da robótica, pois com o desenvolvimento de sistemas cada vez mais autônomos, como robôs que são enviados para explorar outros planetas, ou submarinos para acessar áreas remotas do oceano, é cada vez mais necessário que os sistemas de navegação desses robôs sejam robustos e capazes de fornecer dados para tomadas de decisões importantes.

Capítulo 3

Filtro de Kalman e Filtro de Kalman Estendido

Neste capítulo será mostrado uma explicação sobre como funcionam o Filtro de Kalman e o Filtro de Kalman Extendido.

3.1 Filtro de Kalman

O Filtro de Kalman foi desenvolvido por Rudolf Kalman, e trata-se de um método matemático recursivo que trabalha com a realização de medições de sistemas dinâmicos lineares[17]. O algoritmo é dividido em três partes: predição, observação e atualização. Ele utiliza um modelo dinâmico de um sistema, os sinais de controle enviados e os valores obtidos nos sensores para realizar suas estimativas.

Vamos definir um modelo para o Filtro de Kalman de um sistema linear como:

$$\mathbf{x}(k) = \mathbf{F}\mathbf{x}(k-1) + \mathbf{B}\mathbf{u}(k) + \mathbf{G}\mathbf{v}(k) \tag{3.1}$$

$$\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{w} \tag{3.2}$$

Onde:
x(k): é o vetor de estado que descreve a localização e a orientação do veículo;
F: é a matriz de transição de estados;
B: é o modelo das entradas de controle;
u(k): é o vetor de controle com as entradas;
Gv(k): é o ruído do processo com covariância Q_k;
z: é o vetor de observação;
H: é o modelo de observação;
w: é o ruído da observação com covariância R_k.

3.1.1 Predição

O algoritmo primeiro realiza a predição do estado antes da próxima observação. As equações que definem essa etapa são dadas por:

$$\hat{x}(k|k-1) = \mathbf{F}\hat{x}(k-1|k-1) + \mathbf{B}\mathbf{u}(k) \tag{3.3}$$

$$\mathbf{P}(k|k-1) = \mathbf{F}\mathbf{P}(k-1|k-1)\mathbf{F}^T + \mathbf{G}\mathbf{Q}\mathbf{G}^T \tag{3.4}$$

Onde a primeira equação calcula o valor do estado e a segunda a sua covariância, que significa a incerteza do veículo em sua localização.

3.1.2 Observação

A observação é feita usando as seguintes equações:

$$\mathbf{v}(k) = \mathbf{z}(k) - \mathbf{H}\hat{x}(k|k-1) \tag{3.5}$$

$$\mathbf{S} = \mathbf{H}\mathbf{P}(k|k-1)\mathbf{H}^T + \mathbf{R} \tag{3.6}$$

Onde a primeira equação calcula a inovação, a segunda sua covariância e R representa a covariância do ruído de observação.

3.1.3 Atualização

Por último é feita a atualização. É necessário deixar claro que caso o robô não faça nenhuma observação no período atual, não é necessário que o passo da atualização seja realizado. As equações que definem esse passo são:

$$\mathbf{W}(k) = \mathbf{P}(k|k-1)\mathbf{H}^T\mathbf{S}^{-1} \tag{3.7}$$

$$\hat{x}(k|k) = \hat{x}(k|k-1) + \mathbf{W}(k)\mathbf{v}(k) \tag{3.8}$$

$$\mathbf{P}(k|k) = \mathbf{P}(k|k-1) - \mathbf{W}(k)\mathbf{S}\mathbf{W}(k)^T \tag{3.9}$$

Onde a primeira é o ganho ótimo de Kalman, a segunda representa o estado atualizado e a terceira a sua covariância estimada.

3.1.4 Características do Filtro de Kalman

Paul Michael Newman cita em EKF Based Navigation and SLAM[3] as características desse algoritmo, que são:

- Ele é recursivo, ou seja, a saída de uma iteração entra na próxima;

- Terão que ser fornecidas as estimativas iniciais em $\hat{x}(0|0)$ e P(0|0), ou seja, a localização e as incertezas iniciais;
- A inovação é a diferença entre a observação real e a observação prevista;
- A atualização não precisa ser feita se nenhuma observação for realizada, logo a melhor estimativa para aquele momento é a predição;
- Os elementos da diagonal da matriz de covariância P são as principais incertezas em cada um dos elementos do vetor de estado.

Podemos ver abaixo um resumo de como funciona o algoritmo também presente em EKF Based Navigation and SLAM[3].

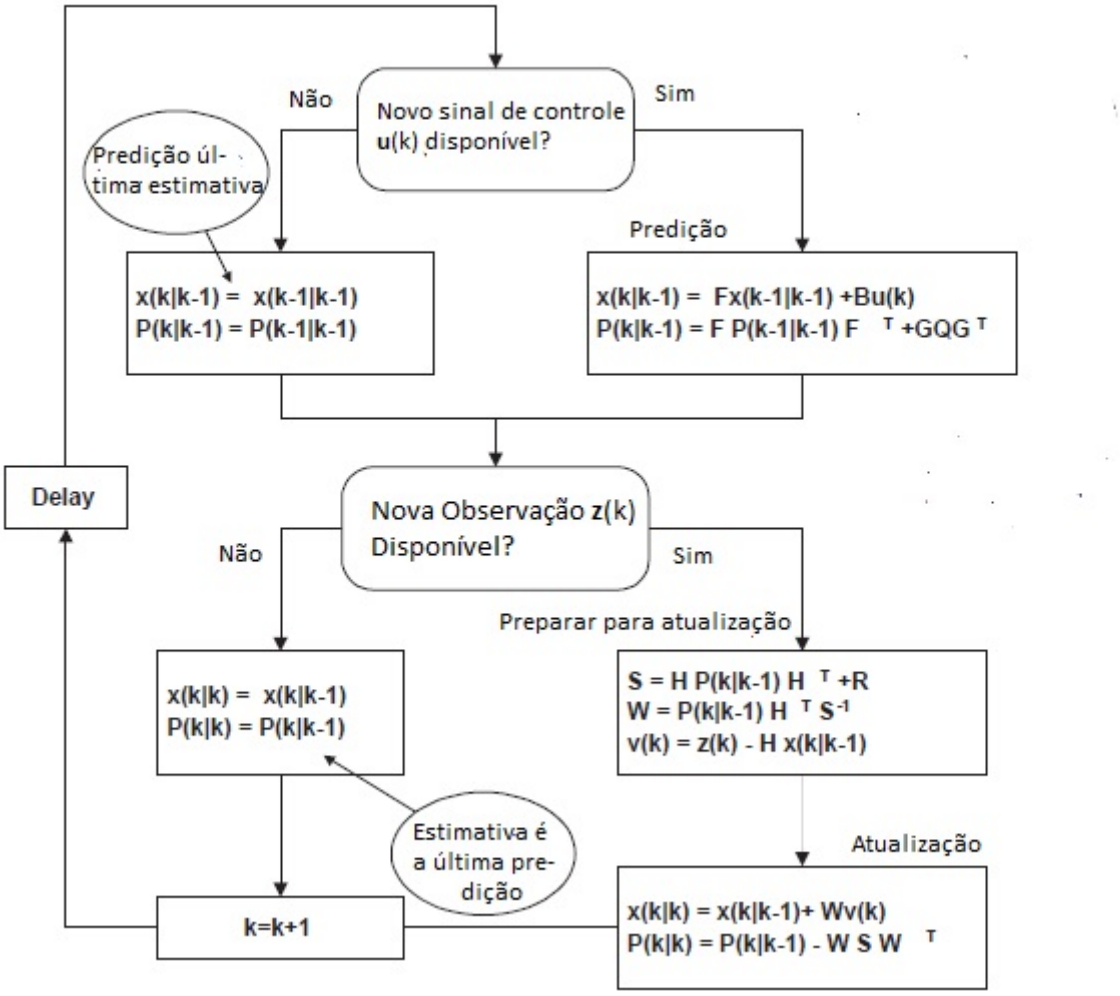

Figura 3.1: Filtro de Kalman: Resumo do seu funcionamento

3.2 Filtro de Kalman Estendido

O Filtro de Kalman Estendido foi desenvolvido com base no Filtro de Kalman por causa que a maioria das aplicações requerem o uso de modelo não lineares. É definido como um método matemático que produz estimativas dos valores reais de medições realizadas ao longo de um período, as quais podem conter ruídos. Assim como no Filtro de Kalman, este algoritmo também é dividido em três partes: predição, observação e atualização.

Vamos começar definindo o sistema não-linear como:

$$\mathbf{x}(k) = \mathbf{f}(x(k-1), \mathbf{u}(k), k) + \mathbf{v}(k) \tag{3.10}$$

$$\mathbf{z}(k) = \mathbf{h}(\mathbf{x}(k), \mathbf{u}(k), k) + \mathbf{w}(k) \tag{3.11}$$

Onde:
x(k): é o vetor de estado que descreve a localização e a orientação do veículo;
f(): é o modelo de transição de estados;
u(k): é o vetor de controle com as entradas;
v(k): é o vetor de erro de ruído do processo com covariância Q_k;
z(k): é o vetor de observação;
h(): é o modelo de observação;
w(k): é o vetor de ruído da observação e com covariância R_k.

3.2.1 Predição

O primeiro passo do algoritmo é a predição. A mesma é calculada com base nas seguintes equações, onde a primeira faz a predição do estado e a segunda a predição da covariância:

$$\hat{x}(k|k-1) = \mathbf{f}(\hat{x}(k-1|k-1), \mathbf{u}(k), k) \tag{3.12}$$

$$\mathbf{P}(k|k-1) = \nabla\mathbf{F}_x\mathbf{P}(k-1|k-1)\nabla\mathbf{F}_x^T + \nabla\mathbf{G}_V\mathbf{Q}\nabla\mathbf{G}_V^T \tag{3.13}$$

Onde:

$$\nabla\mathbf{F}_x = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}.$$

3.2.2 Observação

Após a predição, o algoritmo inicia a observação com base nas seguintes equações, onde primeiro calcula a inovação e em seguida a sua covariância:

$$\mathbf{v}(k) = \mathbf{z}(k) - \mathbf{H}(k)\hat{x}(k|k-1) \tag{3.14}$$

$$\mathbf{S} = \nabla\mathbf{H}_x\mathbf{P}(k|k-1)\nabla\mathbf{H}_x^T + \mathbf{R} \tag{3.15}$$

Onde:
R é a covariância do ruído de observação;

$$\nabla\mathbf{H}_x = \frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial x_1} & \cdots & \frac{\partial h_n}{\partial x_m} \end{bmatrix}.$$

3.2.3 Atualização

Por último ocorre a etapa de atualização, definido pelas seguintes equações:

$$\mathbf{W} = \mathbf{P}(k|k-1)\nabla\mathbf{H}_x^T\mathbf{S}^{-1} \tag{3.16}$$

$$\hat{x}(k|k) = \hat{x}(k|k-1) + \mathbf{W}\mathbf{v}(k) \tag{3.17}$$

$$\mathbf{P}(k|k) = \mathbf{P}(k|k-1) - \mathbf{W}\mathbf{S}\mathbf{W}^T \tag{3.18}$$

Onde a primeira é o ganho ótimo de Kalman, a segunda representa o estado do sistema atualizado e a terceira a sua covariância estimada.

3.2.4 Considerações Finais

Neste capítulo foi explicado como funcionam o Filtro de Kalman e o Filtro de Kalman Estendido. Vimos as características presentes no Filtro de Kalman e consequentemente também presentes no EKF. Com base nisso, podemos notar que a sua implementação não é tão complicada como foi dito anteriormente, além de se tratar de uma ferramenta bastante útil para resolver o problema de SLAM.

Capítulo 4

Materiais e Métodos

Neste capítulo irá ser mostrado o desenvolvimento do trabalho. Para isso, o problema da localização foi escolhido para ser desenvolvido e utilizado no restante do trabalho, pois ele exemplifica como funciona parte do SLAM, podendo ser adaptado posteriormente para funcionar dessa maneira.

4.1 Simulações com o MATLAB

Para compreender melhor como funciona a Localização, foi utilizado o algoritmo para MATLAB desenvolvido em EKF Based Navigation and SLAM[3]. Como demonstrado anteriormente, na Localização é fornecido o mapa com a localização dos obstáculos ao veículo para que ele seja capaz de fazer as observações desses obstáculos e assim obter a sua localização no ambiente. Essa é a localização obtida tratando o ambiente como 2D e o vetor de estados é composto pela sua posição X, Y e pela orientação θ. Podemos ver um exemplo do algoritmo sendo executado na figura 4.1.

Ainda observando a figura 4.1, a posição inicial do veículo foi definida em $[10\,10\,1]^T$, onde o primeiro valor representa o eixo X, o segundo o eixo Y, ambos em centímetro, e o terceiro o ângulo θ em graus. Podemos ver que tanto a posição real representada pelo triângulo de cor vermelha como a posição estimada pelo Filtro de Kalman representada pelo triângulo azul coincidem bastante, mesmo que no código sejam introduzidos ruídos na leitura da odometria e no sensor que faz as observações dos obstáculos. Em determinado momento essas posições começam a divergir e as incertezas representadas pelas elipses vermelhas crescem, pois o algoritmo para propositalmente de fazer observações, porém ao recomeçar a observação dos obstáculos representados pelos asteriscos verdes, nota-se que a posição real e estimada voltam a se aproximar bastante.

Com base nesse algoritmo, podemos ver que o Filtro de Kalman funciona muito bem para a situação em estudo, onde o mapa com os seus obstáculos são conhecidos. Mesmo ele simulando essa falha no sensor de distância, uma situação que pode realmente acontecer em situações práticas, o veículo consegue obter novamente sua localização com as observações posteriores e assim ter valores bem próximos dos valores reais.

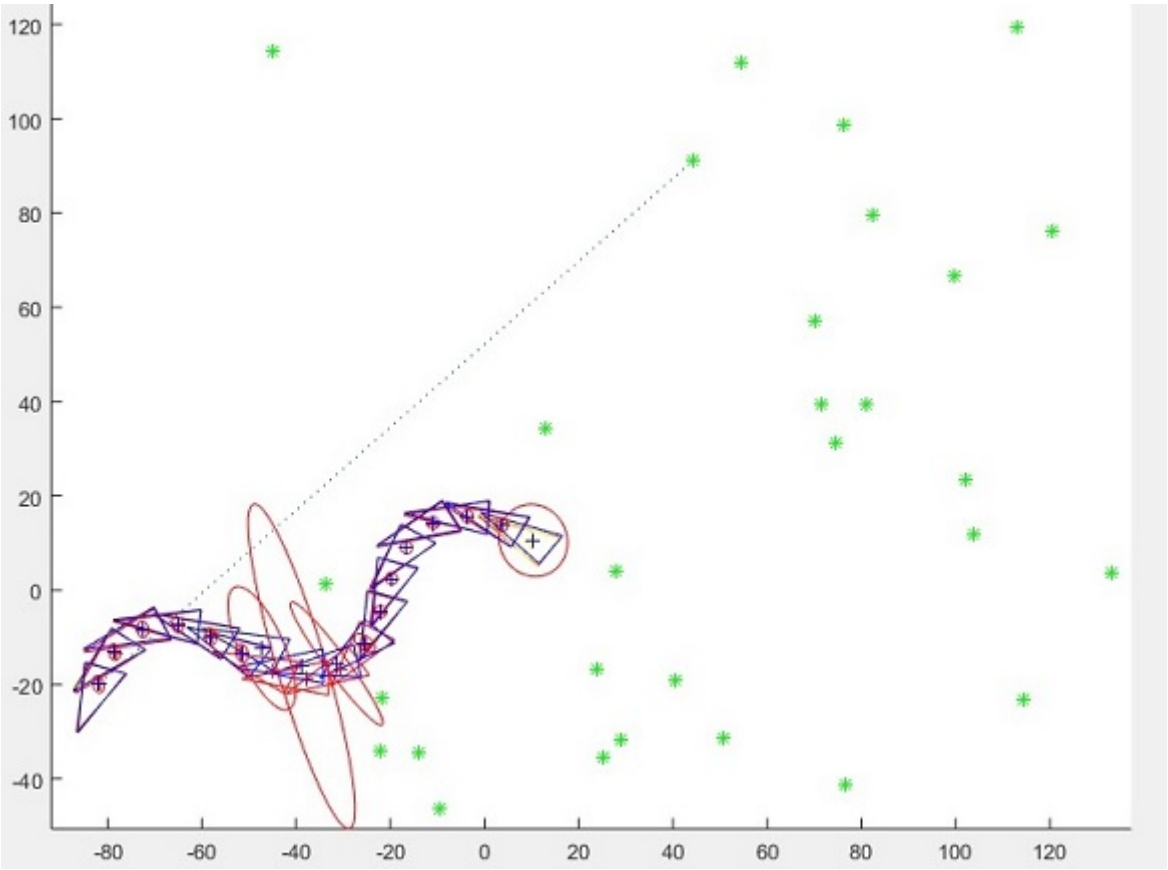

Figura 4.1: Algoritmo no MATLAB - Localização, onde o triângulo vermelho representa a posição real do robô, o triângulo azul a sua posição dada pelo Filtro de Kalman, as elipses vermelhas representam a incerteza e os asteriscos verdes representam os obstáculos.
Fonte: Elaborado pelo autor

4.2 Algoritmos Desenvolvidos

Com base no algoritmo para MATLAB desenvolvido por Newman, foi desenvolvido um algoritmo em C++ afim de ser utilizado para o trabalho. Enquanto que no MATLAB se tem uma facilidade para trabalhar com matrizes e vetores, isso não acontece em C++. Então foi necessário encontrar uma forma de utilizá-los no código desenvolvido sem que fosse algo complexo. Para isso a biblioteca Eigen[4] foi escolhida para auxiliar nessa tarefa. Essa biblioteca permite criar variáveis, sejam matrizes ou vetores, e realizar operações com elas através de funções já implementadas. Resolvendo os problemas que possam aparecer de álgebra linear, trata-se de uma ferramenta bastante poderosa e versátil que foi usada para resolver várias questões neste trabalho e que pode ser aplicada em diversos outros sem muita complexidade.

Com isso o código pôde ser desenvolvido de forma eficiente e pode ser visto no apêndice. Podemos ver de que forma ele funciona na figura 4.2. No primeiro bloco são fornecidas as informações iniciais para realizar a localização. Em seguida, o segundo bloco é onde ocorre a comunicação do computador com o Arduino para receber as informa-

ções obtidas dos sensores para odometria e de possíveis observações. Essa comunicação acontece via porta serial e é programada para que o Arduino acumule as informações de odometria enquanto não for solicitado. Com as informações de odometria obtidas, é realizada a etapa de predição do EKF no terceiro bloco. No quarto bloco temos a verificação se alguma observação válida de algum obstáculo foi feita. Caso ocorra uma observação válida, o algoritmo identifica qual o obstáculo observado para que as etapas de observação e atualização do EKF sejam aplicadas e assim seja possível obter um novo valor para a posição estimada do robô. No bloco seguinte, caso nenhuma observação tenha sido feita na iteração atual e não tenha ocorrido a atualização da posição estimada do robô no passo anterior, a posição estimada do robô é a que foi calculada na etapa de predição e o laço reinicia com o algoritmo pedindo novamente os dados ao Arduino.

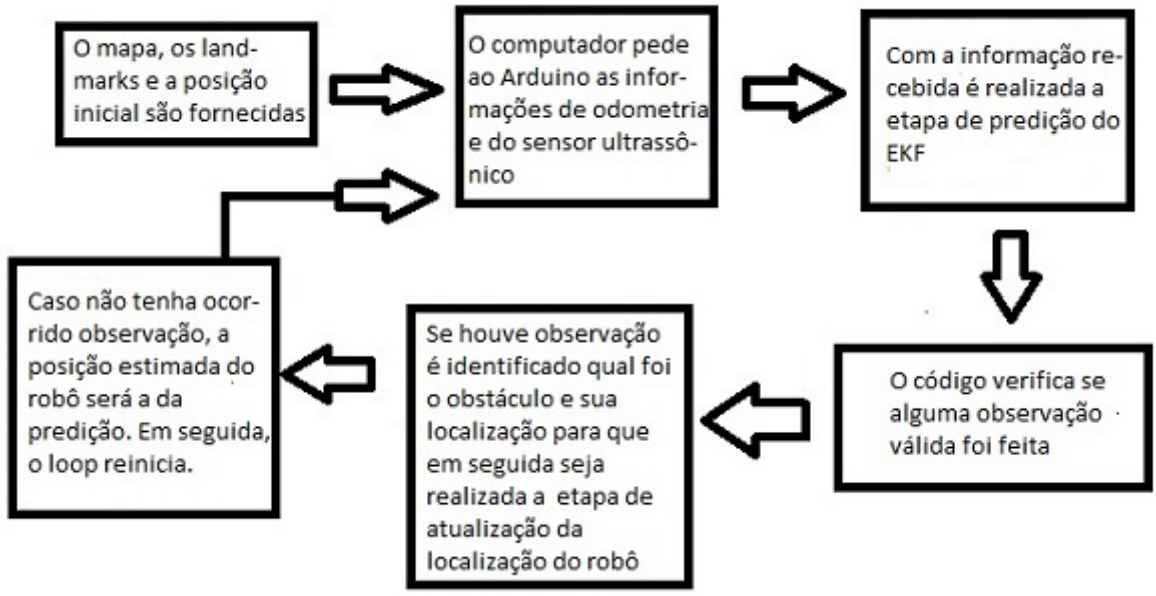

Figura 4.2: Fluxograma de funcionamento do algoritmo desenvolvido.
Fonte: Elaborado pelo autor

Durante o desenvolvimento do algoritmo, surgiu o problema da identificação dos obstáculos. Como iria ser utilizado o sensor ultrassônico para fazer essa identificação, foi necessário desenvolver uma estratégia para que funcionasse da maneira correta. Então foi implementada no código uma função que calcula as distâncias entre a posição estimada atual do veículo com a posição de cada obstáculo. Feito esse cálculo, o algoritmo escolhe o obstáculo que possui a menor distância do veículo. Para que essa solução funcionasse corretamente, foi necessário restringir a distância que o veículo faz uma observação e não deixar obstáculos próximos uns dos outros, pois o principal problema que poderia surgir seria se uma observação fosse feita com a mesma distância para dois obstáculos. Além de medir a distância para o obstáculo, também seria necessário medir o ângulo de observação, no entanto como o sensor ultrassônico foi colocado na frente do robô, este ângulo sempre será constante e igual a zero. A equação usada para calcular essa distância

para cada obstáculo é mostrada a seguir, onde PosX e PosY são as coordenadas em X e Y da posição estimada do veículo e ObsX e ObsY são as coordenadas em X e Y de cada obstáculo.

$$distancia = \sqrt{(PosX - ObsX)^2 + (PosY - ObsY)^2} \tag{4.1}$$

Também foi necessário desenvolver um algoritmo para o Arduino, já que ele é responsável por todos os sensores e atuadores do veículo. Foi necessário o uso de interrupções para contar cada pulso dos encoders, pois assim o código é interrompido em qualquer parte do laço de repetição e nenhuma informação de odometria é perdida. A cada laço ele checa se existe algum obstáculo a frente do veículo para também armazenar o valor da distância entre eles e enviar para o computador, caso seja solicitado os dados nessa iteração. Outro dado armazenado pelo algoritmo é o tempo decorrido desde a última comunicação via serial com o computador, pois essa informação é necessária para os cálculos de odometria. Em razão da forma que o modelo foi construído, ao ser solicitado pelo computador via porta serial para enviar os dados armazenados e assim os enviar, o algoritmo zera os pulsos dos encoders armazenados para recomeçar a contagem.

4.3 Modelo do Veículo

O veículo utilizado neste trabalho possui 4 rodas. Para facilitar, foi considerado que as duas rodas da esquerda são uma só, assim como as duas da direita, representando assim um modelo uniciclo. Para o caso desenvolvido neste trabalho, a localização é feita em 2D, portanto seu vetor de estado é definido como:

$$X_v = [x_v \, y_v \, \theta_v]^T \tag{4.2}$$

O modelo de uniciclo mais comum encontrado na literatura é mostrado na próxima equação, onde v é a velocidade linear do veículo e v_θ a velocidade angular[17].

$$\begin{bmatrix} \dot{x_v} \\ \dot{y_v} \\ \dot{\theta}_v \end{bmatrix} = \begin{bmatrix} v\cos(\theta_v) \\ v\sin(\theta_v) \\ v_\theta \end{bmatrix} \tag{4.3}$$

Fazendo algumas considerações e decompondo a equação anterior, chegamos na seguinte equação para esse modelo:

$$\begin{bmatrix} \dot{x_v} \\ \dot{y_v} \\ \dot{\theta_v} \end{bmatrix} = \begin{bmatrix} v_x\cos\theta_v - v_y\sin\theta_v \\ v_x\sin\theta_v + v_y\cos\theta_v \\ v_\theta \end{bmatrix} \tag{4.4}$$

Podemos ver a seguir a equação que nos fornece a função de movimento discreto do veículo:

$$X_v(k+1) = f_v(X_v(k), u(k)) \tag{4.5}$$

Os Jacobianos dessa função para as variáveis de estado e de controle são respectivamente:

$$D_{X_v} f = \begin{bmatrix} 1 & 0 & -v_x(k)\sin\theta_v(k) - v_y(k)\cos\theta_v(k) \\ 0 & 1 & v_x(k)\cos\theta_v(k) - v_y(k)\sin\theta_v(k) \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.6}$$

$$D_u f = \begin{bmatrix} \cos\theta_v(k) & -\sin\theta_v(k) & 0 \\ \sin\theta_v(k) & \cos\theta_v(k) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.7}$$

Como foi considerado que as duas rodas da esquerda são uma só, assim como as da direita, foram utilizados dois encoders, um em cada roda dianteira. Com esse modelo adotado, foi notado a existência de um problema. O problema notado foi que os motores, apesar de serem semelhantes, não possuem rotações idênticas, portanto as duas rodas do mesmo lado do veículo não giram na mesma velocidade, o que pode acabar gerando uma discrepância nos dados da odometria. Uma implicação deste problema é que as rodas da direita e da esquerda também não possuem as mesmas rotações, fazendo com que o veículo não se movimente totalmente em linha reta, porém a odometria é capaz de fornecer essa diferença.

4.4 Hardware

4.4.1 Plataforma móvel

Foi escolhido o Kit Chassi 4WD que pode ser visto na figura 4.3 para utilizar como veículo móvel. Suas especificações são:

- 02 - Chassi em acrílico;
- 04 - Motores DC (3-6v);
- 04 - Rodas de Borracha;
- 04 - Discos de Encoder.

4.4.2 Arduino

O Arduino é uma plataforma eletrônica de código livre que pode ser usado em projetos variados. Sua linguagem de programação é de fácil aprendizado, além de comunicar-se facilmente com computadores, sensores e atuadores. Foi escolhido para este trabalho pela facilidade em utilizá-lo como intermediário entre os sensores, os atuadores e o computador para o qual os valores lidos serão enviados para a realização da localização.

4.4.3 Acionamento dos Motores

Para simplificar, foi definido que os dois motores da esquerda recebem o mesmo sinal de controle, assim como os dois da direita. Para realizar esse controle foi utilizado o

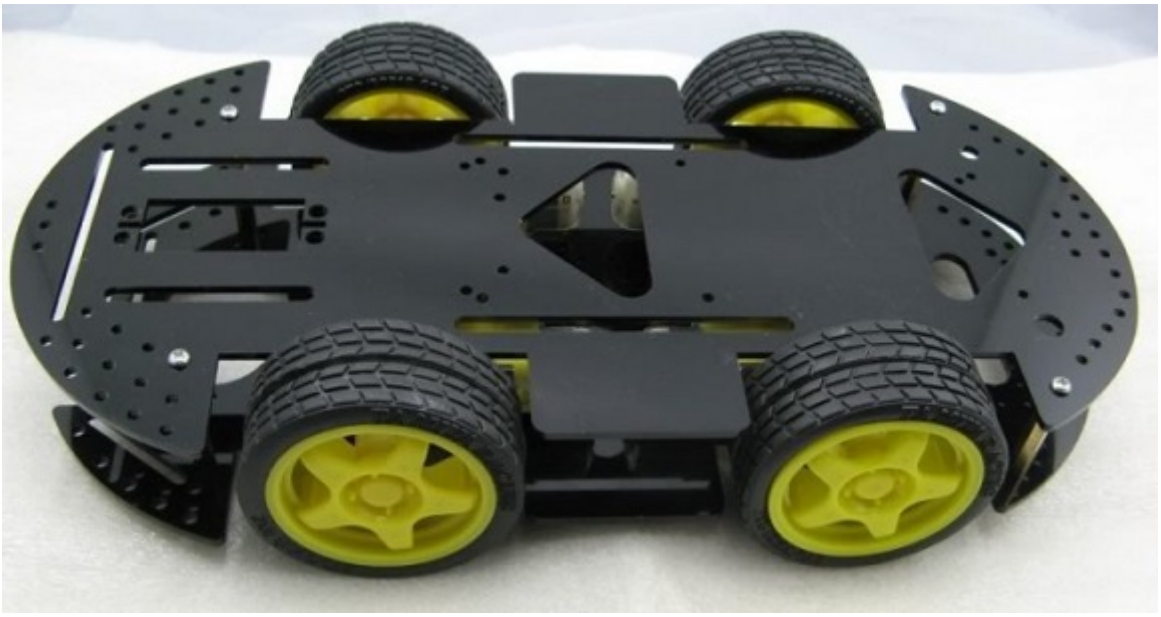

Figura 4.3: Chassi utilizado.
Fonte: Elaborado pelo autor

CI L293D [6] que pode ser visto na figura 4.4. O mesmo foi escolhido pelos seguintes motivos:

- Possui duas pontes H em um único CI, onde uma foi utilizada para os motores da direita e outra para os da esquerda;
- Suporta até 1,2A;
- Suporta dois motores de até 36V;
- Já vem com os diodos de proteção.

Figura 4.4: CI L293D utilizado para acionamento dos motores.
Fonte: https://i0.wp.com/suhanko.files.wordpress.com/2011/08/l293d.jpg

4.4.4 Sensor de Distância

Para realizar as medições da distância que o veículo está dos obstáculos, foi utilizado o sensor ultrassônico HC-SR04 [7] similar ao mostrado na figura 4.5. O mesmo permite medir distâncias de 2 centímetros até 5 metros, no entanto é recomendável utilizá-lo para medições abaixo de 1 metro para que sejam medidos valores mais precisos. Este sensor emite um sinal ultrassônico (trigger) que reflete em um objeto e retorna ao sensor (echo). Com esse sinal de retorno captado é possível medir a distância para o objeto através do tempo que o sinal levou entre ser emitido e refletido.

Depois de realizar algumas medidas com o sensor, foi notado que o erro aumentava cada vez mais conforme a distância medida fosse maior. Para corrigir esse problema, foram feitos alguns testes com distâncias variadas para que fosse possível traçar um gráfico composto de um eixo com as distâncias esperadas e de outro com as distâncias medidas, e a partir dos pontos de interseção existentes foi traçado uma reta para descobrir a equação que passava o mais próximo possível de todos esses pontos, método conhecido como Método dos Mínimos Quadrados (MMQ). A equação obtida e inseria no código em C++ para dar a distância corrigida foi:

$$D = 1.535 + 0.737 * dmedida; \tag{4.8}$$

Onde D é a distância aproximada e dmedida é a distância medida no sensor.

Figura 4.5: HC-SR04 - Sensor de distância utilizado.
Fonte: https://zarelli.files.wordpress.com/2013/02/sensor.jpg

4.4.5 Joystick

Os motores do veículo são acionados pelo módulo Joystick Arduino 3 Eixos [8] que pode ser observado na figura 4.6. O mesmo possui dois potenciômetros que são referenciados para o eixo X e Y, além do botão que pode ser pressionado e é utilizado para o eixo Z que não foi utilizado neste trabalho. Para utilizar esse módulo, é necessário ligar seus pinos X e Y em portas analógicas e o Z em uma digital.

4.4.6 Encoder

Para termos controle sobre a odometria do veículo, que é uma informação necessária para este trabalho, utilizou-se o sensor de velocidade LM393[9]. Ele é acoplado no carrinho, de forma que o disco de encoder que está girando junto com o eixo do motor passe

Figura 4.6: Joystick utilizado para controlar os motores DC.
Fonte: http://mlb-s2-p.mlstatic.com/318321-MLB20773948405_062016-Y.jpg

entre ele e interrompa o feixe de luz infravermelha e assim ele possa contar a quantidade de pulsos. Com essa quantidade de pulsos, juntamente com o número de pulsos por volta do disco, o tempo gasto e o raio do disco, calcula-se a velocidade por segundo do veículo.

Figura 4.7: Encoder utilizado para obter informações sobre odometria.
Fonte: http://img.dxcdn.com/productimages/sku_150663_3.jpg

4.4.7 Notebook

Responsável por executar o algoritmo da Localização recebendo os dados dos sensores via porta serial pelo Arduino. É onde os dados enviados pelo arduino são tratados, utilizando os mesmos no algoritmo da Localização e assim obtendo a posição atual do veículo. O programa é executado no notebook com um Sistema Operacional Linux, portanto pode ser utilizado em outras máquinas com esse mesmo tipo de sistema sem precisar de adaptação no código, além de poder ser substituído por uma placa Raspberry ou Beagleboard, por exemplo. Para realizar a comunicação serial entre o Arduino com um Linux, foi utilizado a biblioteca arduino-serial[10].

4.5 Visão geral do Hardware

Na figura 4.8 é apresentada uma visão geral de como os componentes do Hardware interagem. O Arduino atua recebendo informações do sensor ultrassônico, do joystick e

dos encoders, ao mesmo tempo que envia os sinais para o CI L293D controlar os motores do veículo. Também está representado a comunicação via porta serial entre o Arduino e o computador, pois ao solicitar as informações do Arduino, o computador envia um sinal para que a comunicação seja liberada e o Arduino responde enviando os dados.

Já a figura 4.9 apresenta o diagrama elétrico do trabalho com todas as ligações que foram feitas entre o Arduino e os componentes utilizados na montagem do veículo. Por último, na figura 4.10 temos o veículo pronto para a realização dos testes.

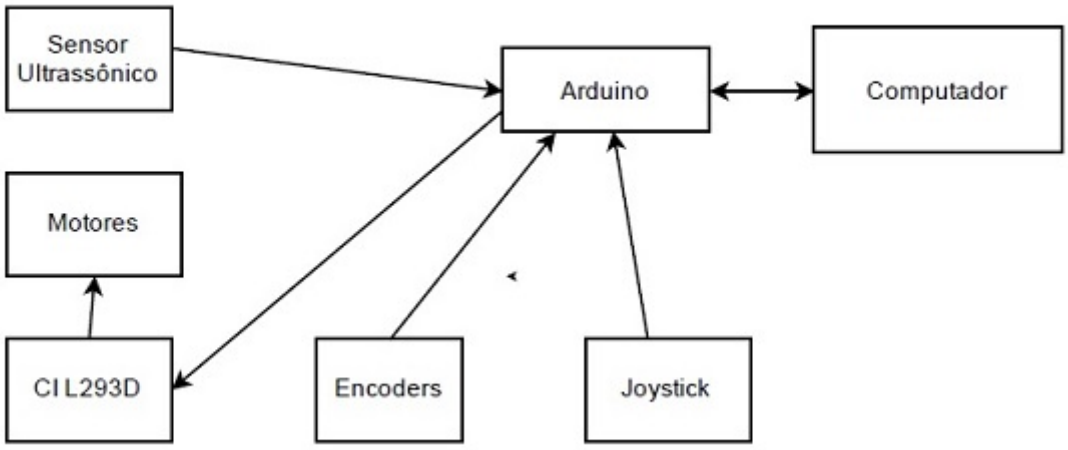

Figura 4.8: Diagrama de Blocos.
Fonte: Elaborado pelo autor

4.6 Considerações Finais

Neste capítulo, foram mostradas as ferramentas utilizadas para desenvolver o trabalho. Vimos a importância da biblioteca Eigen para o código desenvolvido em C++, assim como todo o hardware utilizado para a parte prática, com breve explicações da motivação para o uso de cada componente. Com base no que foi desenvolvido e utilizado neste capítulo, podemos passar para o capítulo seguinte que trata de testes e resultados.

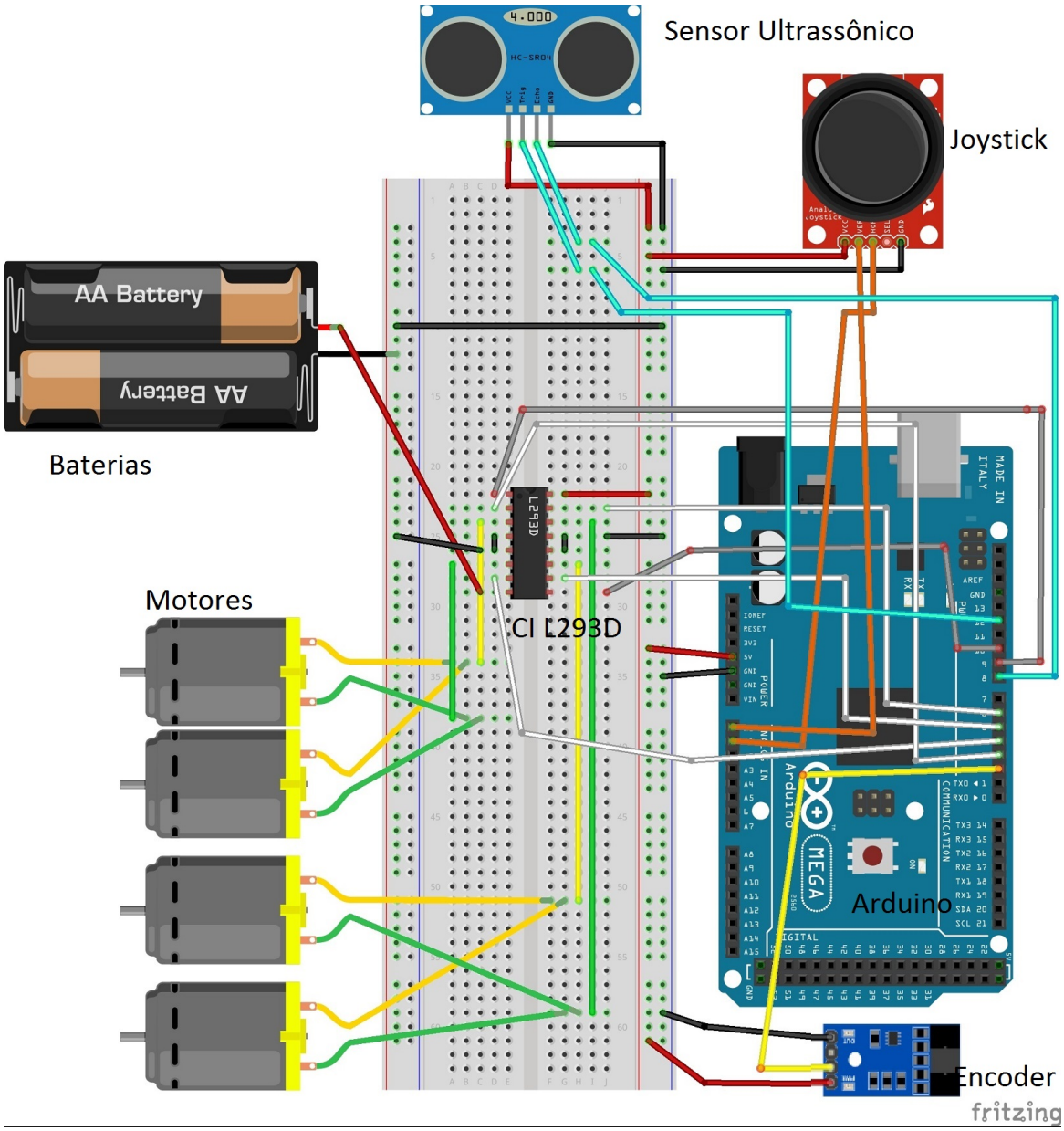

Figura 4.9: Ligações no Arduino.
Fonte: Elaborado pelo autor

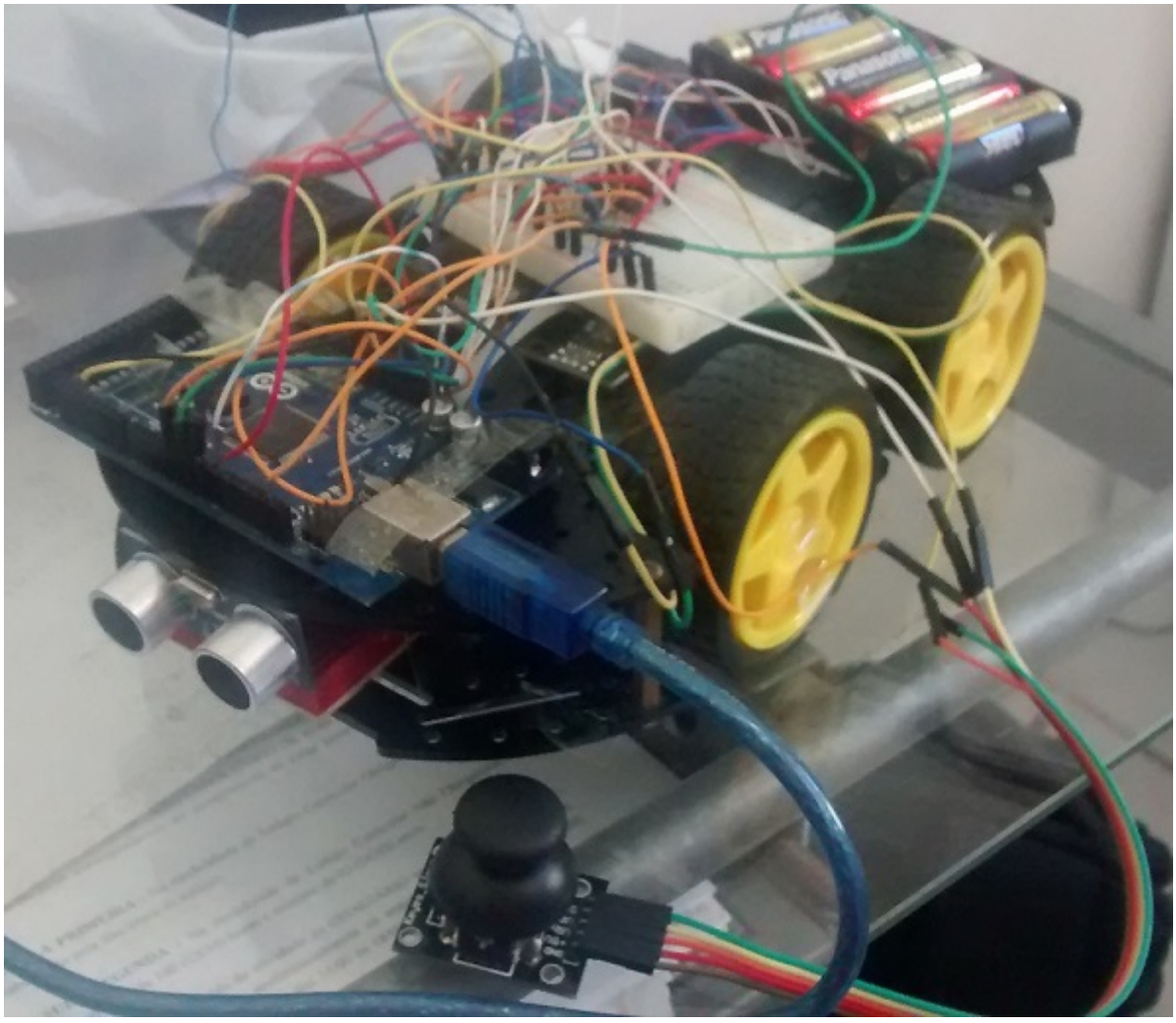

Figura 4.10: Veículo Montado.
Fonte: Elaborado pelo autor

Capítulo 5

Resultados e Discussões

Neste capítulo serão mostrados e comentados todos os testes feitos com o software e hardware desenvolvidos.

5.1 Testes de Odometria

Depois do protótipo de robô montado, os primeiros testes realizados foram para ver se a odometria estava funcionando de forma aceitável. O algoritmo desenvolvido foi alterado para que trabalhasse apenas com as informações de odometria. Para isso foi realizado testes com o robô andando apenas para frente, depois apenas para trás e por último para frente e para trás. Todos os testes de odometria foram realizados partindo da posição $[10\,10\,0]^T$, onde as posições X e Y estão em centímetros e a posição θ é dada em graus. Os gráficos obtidos foram os seguintes, mostrados nas figuras 5.1, 5.2 e 5.3:

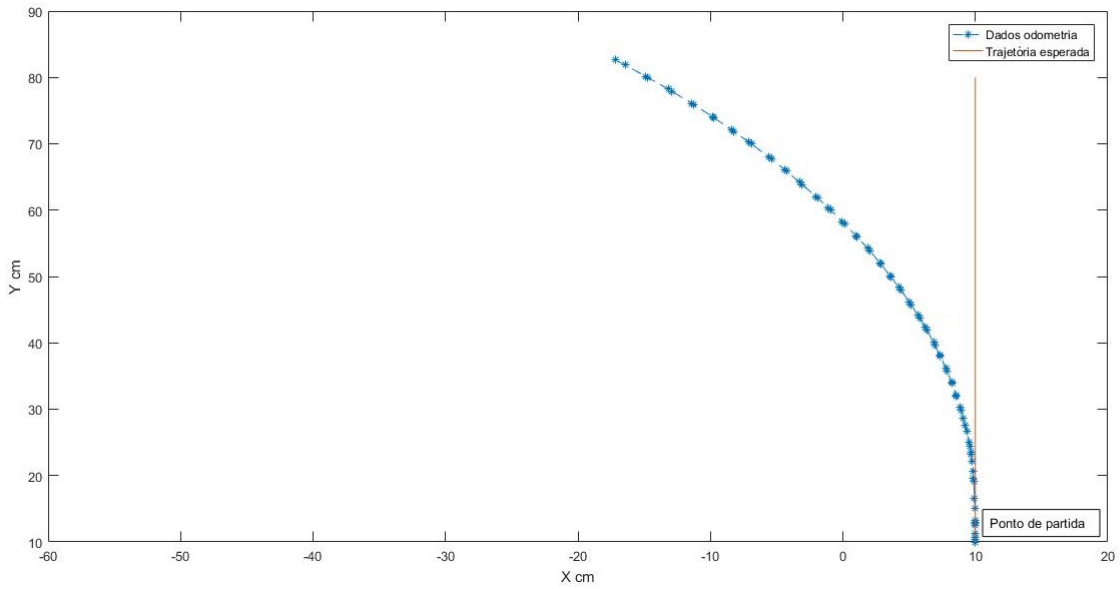

Figura 5.1: Veículo Para Frente.
Fonte: Elaborado pelo autor

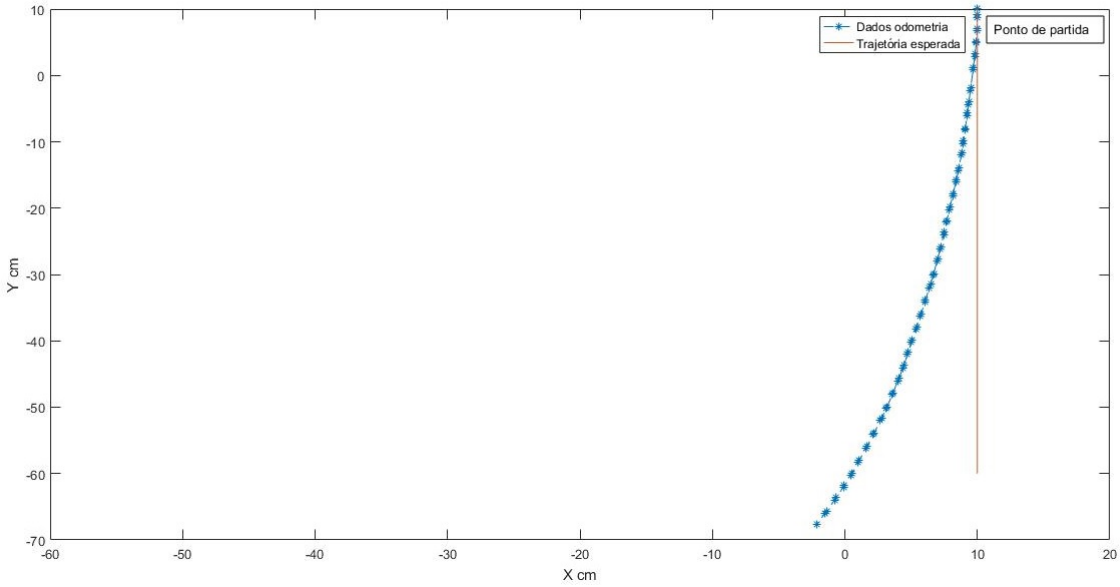

Figura 5.2: Veículo de Ré.
Fonte: Elaborado pelo autor

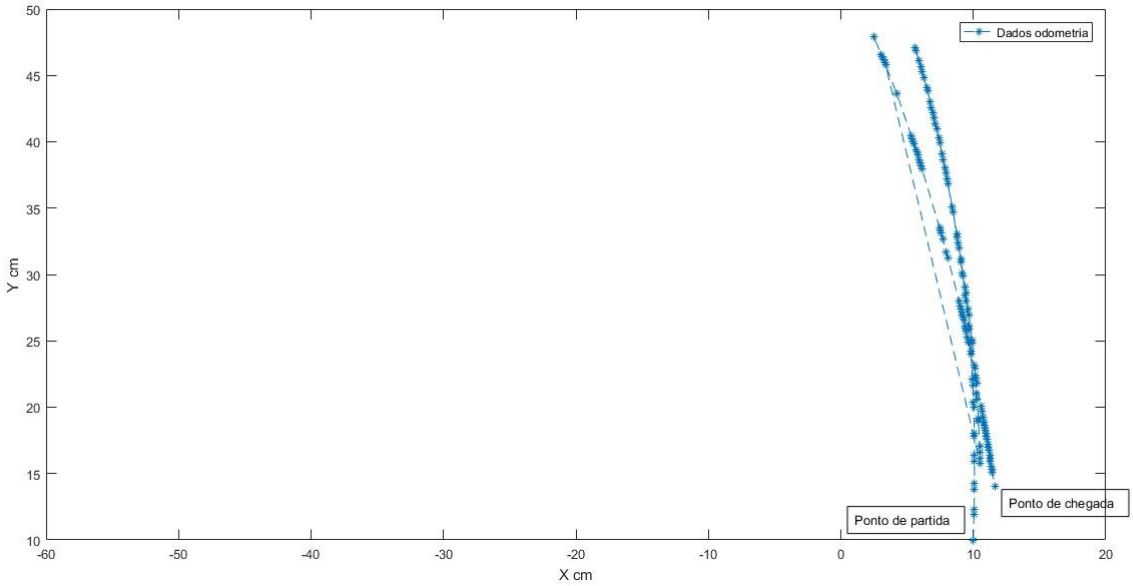

Figura 5.3: Veículo Para Frente e Depois Ré.
Fonte: Elaborado pelo autor

Na figura 5.1 podemos observar a trajetória do veículo para frente, onde o ponto de chegada ideal seriam os valores de 10 cm no eixo X e 80 cm no eixo Y. Foi verificado antes de realizar os testes, que as rodas tem uma diferença na velocidade como mencionado anteriormente, portanto o veículo acaba não indo totalmente reto. Porém existiu uma maior diferença neste figura, onde foi verificado o que causou esse problema. As conexões

que conectam o Arduino aos sensores parecem causar um pouco de ruído umas nas outras, o que ocasiona o problema de que quando as rodas estão indo nessa direção, um encoder conta alguns pulsos a mais do que o outro por volta, levando a pensar que o veículo estava fazendo uma curva. No entanto, ao trocarmos alguns conectores de lugares, esse efeito foi contornado. Portanto, em vez da figura mostrar uma curva suave que seria resultado da pequena diferença entre as velocidades de rotações dos motores, acabou mostrando uma curva maior, pois os dados da odometria foram contaminados com o problema mencionado. Em relação ao eixo Y, a odometria obteve a coordenada 82.6 cm no ponto de chegada, obtendo assim um erro relativo de 3.25%. Já em relação ao eixo X que obteve a coordenada -17.1 cm, devido ao problema relatado acima, o erro relativo foi de 271%, mostrando que o mesmo comprometeria o sistema caso não fosse solucionado. Na figura 5.2, o ponto de chegada ideal seriam os valores 10 cm no eixo X e -60 cm no eixo Y. O erro relativo ao eixo X, onde a informação de odometria forneceu a coordenada -2.1, foi de 121%. Já em relação ao eixo Y, onde a coordenada final foi de -67.6, o erro foi de 12.6%. É importante ressaltar que esses valores calculados para os erros acima estão contaminados pelo problema detectado, portanto acabou gerando valores altos, já que esses dados acabam dando a informação que o robô esta fazendo uma curva, e não em linha reta.

Já na figura 5.3, com o problema mencionado acima já resolvido, foi realizado um teste onde o objetivo era ir para frente e para trás com o veículo e parar no ponto de partida, para ver como a odometria se comportava. No eixo X tivemos um erro relativo de 16%, enquanto que no eixo Y o erro foi de 40%. Por se tratar de um teste mais completo, este resultado pode ser considerado positivo devido a limitação de hardware existente.

Depois desses testes iniciais, foram realizados mais dois para testarem a funcionalidade tanto do eixo Y como do eixo X do robô. Para isso, primeiro foi feito um teste para realizar uma curva, e em seguida o teste para ver como ficaria o gráfico ao mover-se para formar um quadrado. Os resultados obtidos foram:

Ao realizar o primeiro teste fazendo uma curva em "L", notou-se que tanto o eixo Y como o eixo X estão funcionando, pois com base na informação de odometria obtida e mostrada no gráfico, o veículo realizou uma boa curva. Já em relação ao quadrado, foi observado que realmente a odometria fornece uma primeira curva de forma bastante suave, no entanto, ao realizar as outras o veículo começa a se perder, ocasionando assim que o mesmo não consiga fornecer o gráfico corretamente do quadrado. No entanto, levando em consideração que para realizar essa tarefa ele está usando apenas as informações obtidas nos encoders, o resultado é satisfatório.

5.2 Experimentos com o Problema da Localização

Com a odometria funcionando, foi possível implementar o código no Arduino para a leitura dos dados do sensor ultrassônico, além de habilitar novamente todo o algoritmo desenvolvido em C++. Assim, foram realizados os testes para saber o funcionamento desse algoritmo. Para mostrar que a solução é válida, optou-se por fazer o teste partindo com o veículo da coordenada $[0\,0\,0]^T$ e foi colocado um obstáculo (copo reutilizável) com a posição $[0\,60]^T$. Feito isso, o gráfico obtido é mostrado na figura 5.6.

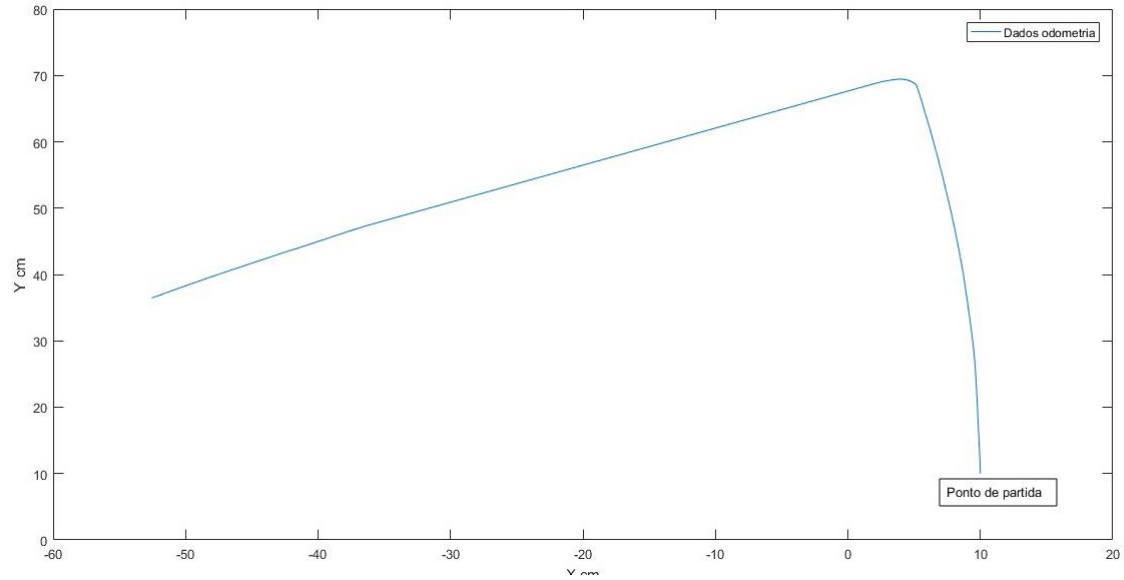

Figura 5.4: Veículo Fazendo uma Curva.
Fonte: Elaborado pelo autor

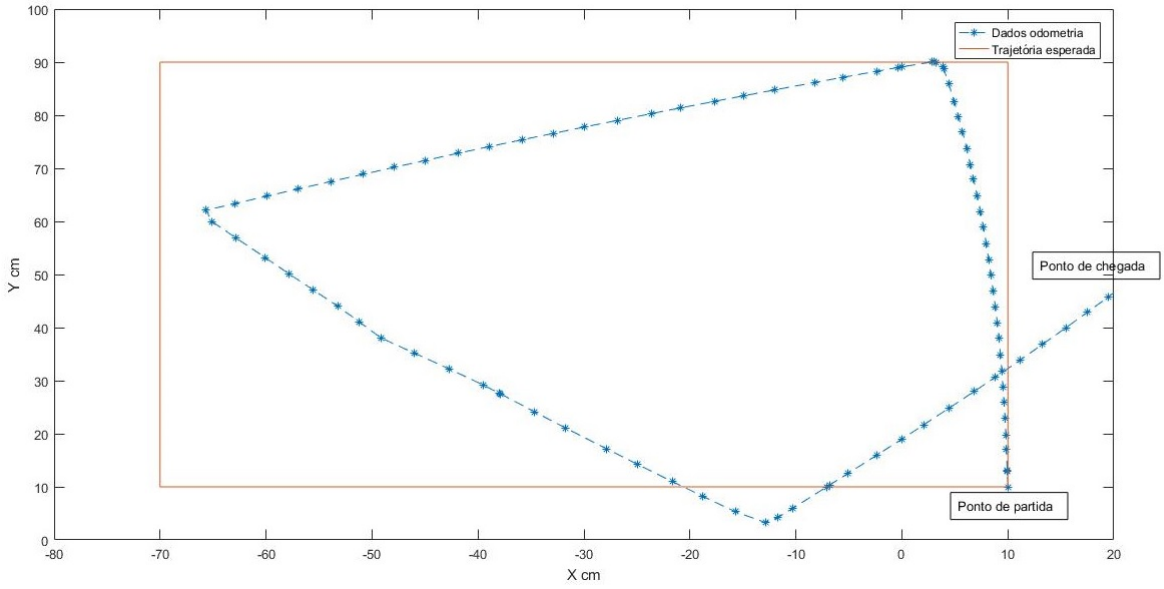

Figura 5.5: Veículo Percorrendo um Quadrado.
Fonte: Elaborado pelo autor

Analisando a figura citada, onde a posição do obstáculo está marcada pelo quadrado vermelho e as posições do robô estão marcadas pelos asteriscos azuis, podemos notar que o veículo tem um comportamento dentro do esperado para o hardware utilizado, e que para um obstáculo os testes realmente funcionaram. Podemos notar na figura que existe um salto na localização do veículo de uma iteração pra outra. Isto ocorre por causa que

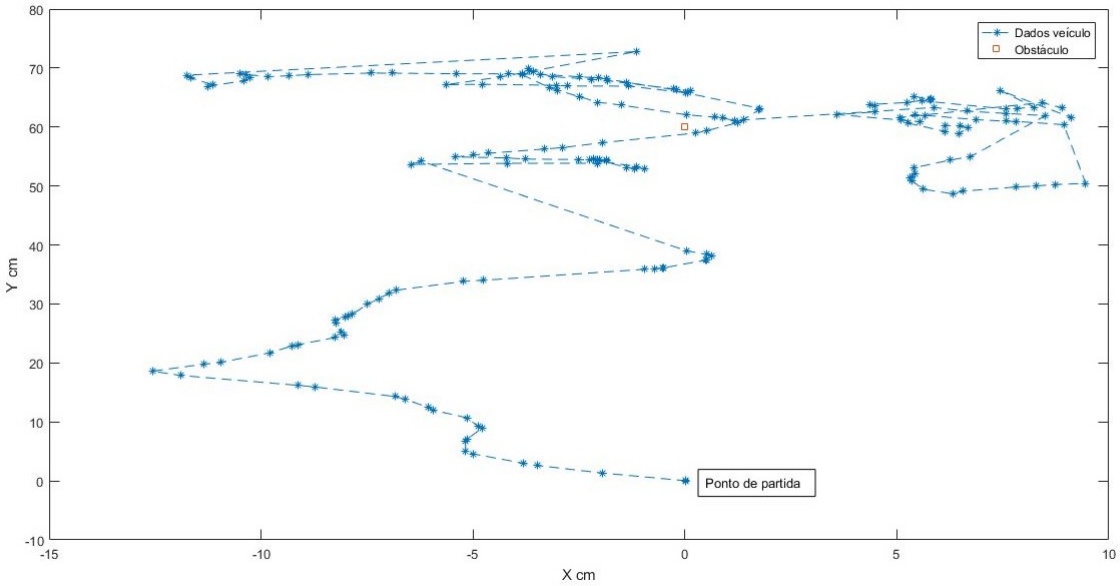

Figura 5.6: Veículo com o Método da Localização Funcionando.
Fonte: Elaborado pelo autor

até aquele momento o mesmo estava usando apenas as informações da odometria para obter a sua localização, porém após realizar a primeira observação, o veículo obtém a sua localização com maior precisão graças ao algoritmo que utiliza o Filtro de Kalman Estendido para calcular um valor estimado mais próximo.

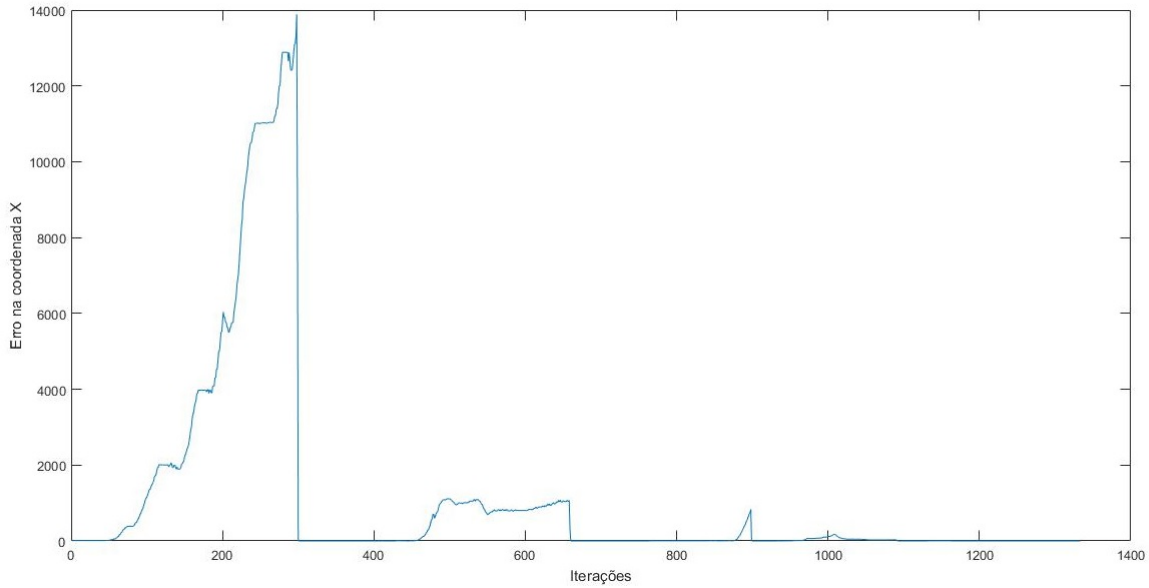

Figura 5.7: Erro calculo para X na matriz de covariância do EKF.
Fonte: Elaborado pelo autor

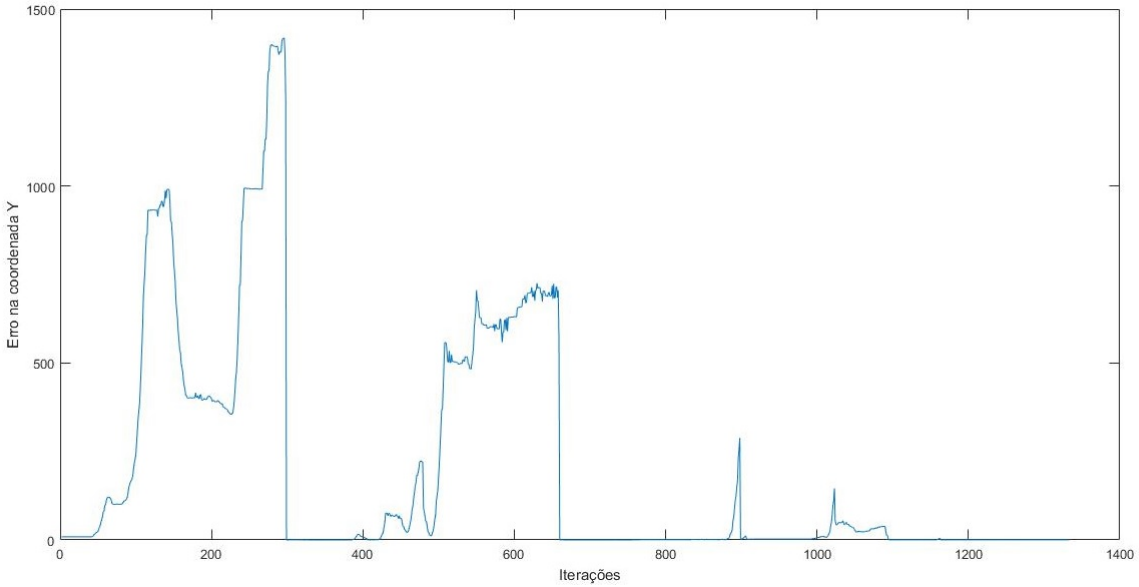

Figura 5.8: Erro calculo para Y na matriz de covariância do EKF.
Fonte: Elaborado pelo autor

Na figura 5.7 e 5.8, temos os erros calculados pelo Filtro de Kalman Estendido em sua matriz de covariância que é dado pelo Desvio Padrão, medido em centímetros, ao quadrado, onde o erro na coordenada X é dado pelo primeiro elemento da diagonal principal desta matriz, e o erro em Y pelo seu segundo elemento. A primeira figura demonstra o erro na coordenada X do veículo e a segunda o erro na coordenada Y. A medida que o tempo vai passando, e consequentemente o veículo vai navegando no ambiente ao longo das iterações do algoritmo, os erros calculados aumentam. Também é importante notar que este erro praticamente desaparece sempre que o veículo faz uma observação, pois o EKF fornece uma estimativa mais aproximada da localização do veículo.

Em seguida, foi realizado um teste onde a trajetória do veículo era conhecida. Novamente partindo da posição $[0\ 0\ 0]^T$ e com o obstáculo na posição $[60\,40]^T$, o veículo percorreu a trajetória pré-estabelecida em forma de "L"e os resultados são mostrados nas figuras 5.9, 5.10 e 5.11. Podemos mais uma vez notar na figura 5.9 um salto na posição do veículo, devido a imprecisão da odometria como ocorreu no teste anterior, ao realizar a primeira observação do obstáculo e assim fornecer a sua localização estimada usando o Filtro de Kalman Estendido com uma precisão melhor. Nas figuras 5.10 e 5.11 temos os erros nas coordenadas X e Y, respectivamente, calculados da mesma forma que no exemplo anterior. Já a figura 5.12 traz uma foto do veículo e obstáculo.

Confirmado que a parte de software e hardware estavam funcionando, tentamos realizar testes com múltiplos obstáculos, porém eles não funcionaram de acordo com o esperado. O problema é que o sensor ultrassom pode ser facilmente influenciado por ruídos ou outra interferência, além da precisão dos sensores que fazem a odometria não serem tão confiáveis, assim quando tentamos sair de um obstáculo para ir tentar identificar outro, o algoritmo sempre entendia que se tratava do primeiro e acabava dando a posição em relação a ele. A identificação do obstáculo observado foi feita determinando-se qual obstáculo

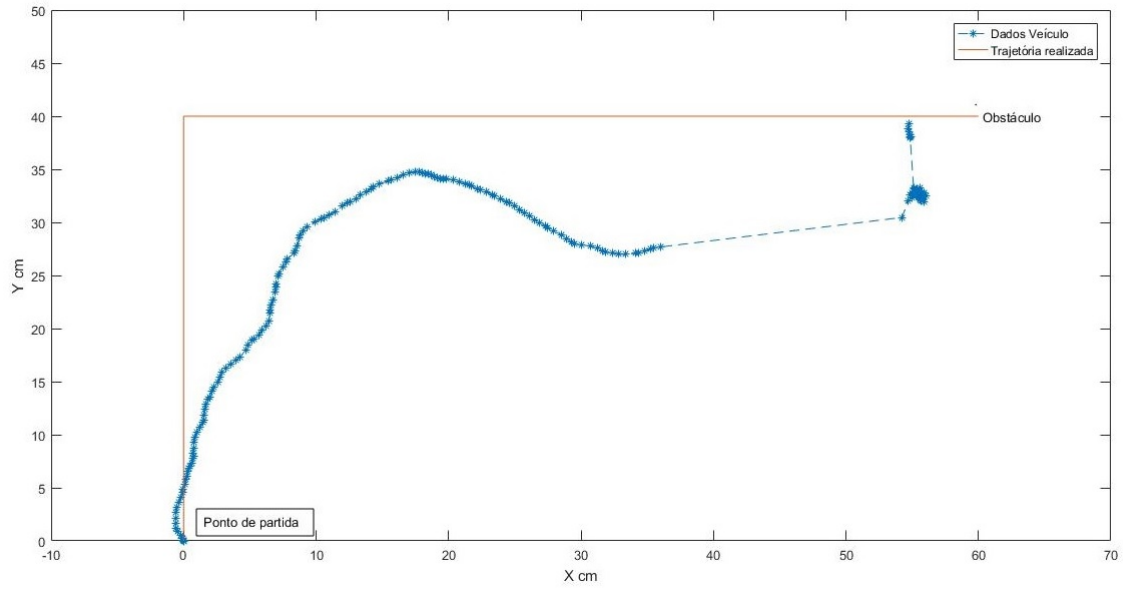

Figura 5.9: Veículo percorrendo trajetória pré-estabelecida
Fonte: Elaborado pelo autor

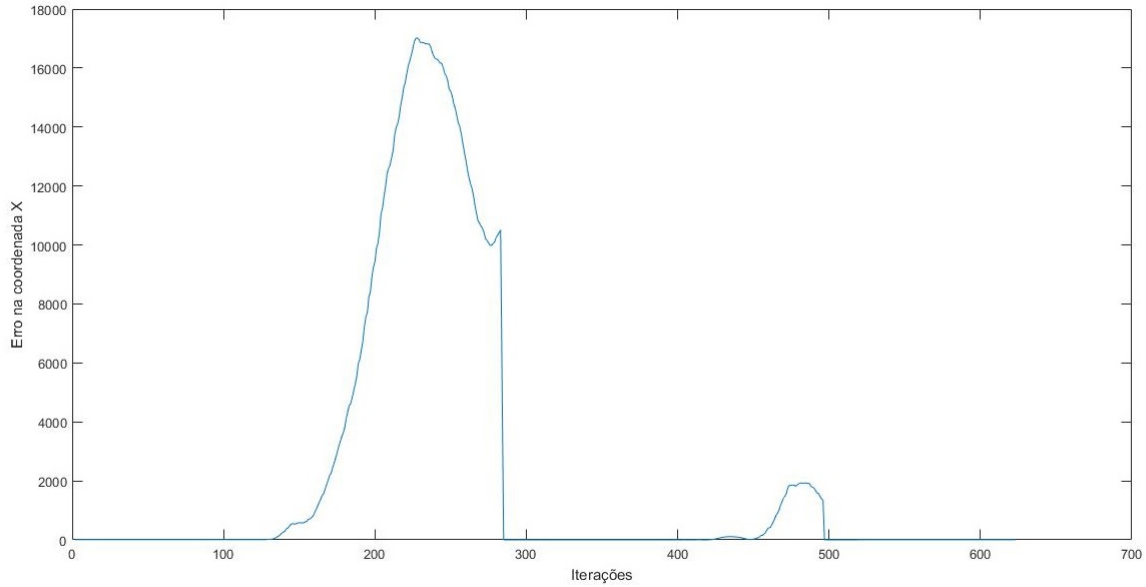

Figura 5.10: Erro calculo para X na matriz de covariância do EKF.
Fonte: Elaborado pelo autor

estava mais próximo da posição prevista do veículo no momento da observação, então por causa do erro de odometria, essa identificação não é feita corretamente. Depois de tentar-

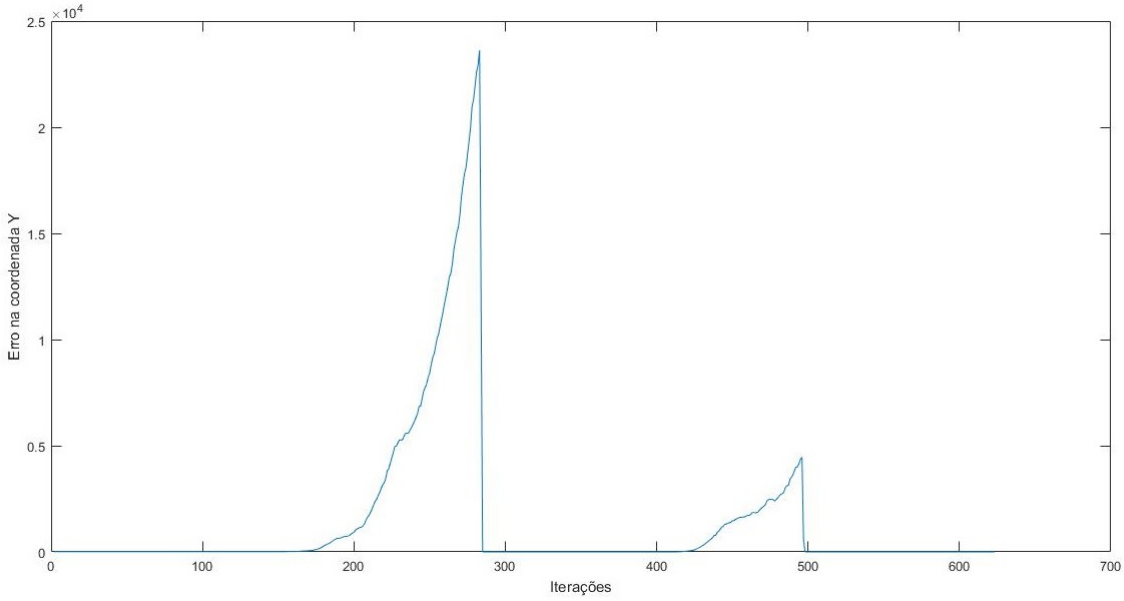

Figura 5.11: Erro calculo para Y na matriz de covariância do EKF.
Fonte: Elaborado pelo autor

mos outras formas e analisarmos o código para ver se não existia algum erro que estava causando isso e percebermos que não havia nenhum, podemos concluir que a limitação existente realmente é de hardware. Ao pensarmos mais sobre a estratégia de identificação de obstáculos, percebemos que ela tem uma falha, pois quando o robô faz uma observação de um obstáculo mais próximo, para em seguida observar um mais distante, ele vai identificar esse obstáculo como sendo o mais próximo dele por possuir a menor distância para a posição estimada do veículo, gerando assim um erro na observação. No futuro, para identificar os obstáculos de uma forma melhor e assim obter resultados para situações com mais de um obstáculo, uma forma interessante seria a utilização de uma câmera ou um kinect, por exemplo.

5.3 Considerações Finais

Ao longo deste capítulo foram mostrados os testes realizados com o veículo desenvolvido para o trabalho. Os testes de odometria realizados tiveram resultados positivos. Já em relação a localização, podemos analisar que os resultados obtidos dentro das limitações foram bons e que o algoritmo desenvolvido para o problema da localização foi feito de forma correta.

Figura 5.12: Veículo e obstáculo.
Fonte: Elaborado pelo autor

Capítulo 6

Conclusões

Este trabalho foi desenvolvido como forma de realizar uma introdução em uma das mais importantes áreas da robótica que é a navegação. Por isso foi realizado uma explicação sobre os tipos de problemas que são enfrentados nessa área, que são a Localização, o Mapeamento e a Localização e Mapeamento Simultâneos (SLAM). Foi escolhido para ser tratado nesse trabalho o problema da localização, pois ele serviu muito bem como um desafio e uma introdução neste tema, além do fato de possibilitar a implementação do SLAM com um pouco mais de esforço.

Para que esses problemas sejam resolvidos, foram citados os algoritmos mais utilizados nessa área, sendo escolhido para ser desenvolvido neste trabalho o Filtro de Kalman Extendido, pois sua implementação não é tão complexa e ele funciona bem para os três problemas.

A partir disso, o veículo foi desenvolvido para que fosse possível realizar uma implementação prática com equipamentos didáticos, dos quais muitos foram vistos e utilizados ao longo do curso, e de baixo custo. Devido a não serem equipamentos robustos e de uma precisão alta, os resultados apresentados não foram totalmente precisos, no entanto, eles são próximos disso e mostram dados correspondentes a realidade.

Durante a implementação tanto de software quanto de hardware alguns problemas surgiram, dentre eles a implementação do código do Filtro de Kalman Extendido e os pulsos dos encoders serem contados de forma incorreta pelo arduino, porém depois de algumas pesquisas e testes realizados foi possível implementá-los e deixar o projeto funcional.

Por fim, este trabalho sintetiza uma boa parte do conteúdo que foi apresentado ao longo do curso e mostra que estamos preparados para os desafios que irão surgir nesta nova etapa profissional.

6.1 Trabalhos Futuros

Para melhorar o que foi desenvolvido ao longo desse trabalho, podemos considerar os seguintes itens:

- Substituir os sensores responsáveis pela odometria por outros mais precisos, afim de obter resultados ainda mais precisos;

- Confecção de uma placa de circuito para tornar o sistema mais robusto em relação principalmente a ruídos, já que um dos problemas que surgiram foi exatamente esse;
- Inserir um módulo Bluetooth para controlar os motores do veículo de forma mais prática;
- Acoplar ao chassi do veículo uma Raspberry ou outro microcomputador para substituir o notebook usado, de forma a deixar o sistema mais independente;
- Inserir uma câmera de vídeo no sistema para ajudar na detecção e identificação de obstáculos e assim deixá-lo apto para funcionar com mais landmarks;
- Modificar o código em C++ do Método da Localização para que resolva o problema do SLAM.

Capítulo 7

Referências Bibliográficas

[1] LATOMBE, Jean-Claude; GONZÁLEZ-BANÕS, Hector H. **Navigation Strategies for Exploring Indoor Environments**. 25 f. Int. Journal of Robotics Research, 2002.

[2] DURRANT-WHYTE, Hugh; Fellow; IEEE; BAILEY, Tim. **Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms**. 9 f. IEEE, 2006.

[3] NEWMAN, Paul Michael. **EKF Based Navigation and SLAM: Background Material, Notes and Example Code**. 94 f. SLAM Summer School 2006, Oxford.

[4] JACOB, Benoît; GUENNEBAUD, Gaël. **Eigen**. 2016. Disponível em: <http://eigen.tuxfamily.org/index.php?title=Main_Page>. Acesso em: 10 jun. 2016.

[5] **Arduino**. 2016. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 10 jun. 2016.

[6] CSACOMANI. **Usando Arduino e o CI L293D para controlar dois motores**. 2016. Disponível em: <https://csacomani.wordpress.com/2012/09/30/usando-o-ci-l293d-para-controlar-dois-motores/>. Acesso em: 10 jun. 2016.

[7] ELECFREAKS. **Ultrasonic Ranging Module HC - SR04**. 2016. Disponível em: <http://www.micropik.com/PDF/HCSR04.pdf>. Acesso em: 10 jun. 2016.

[8] BRAINY BITS. **Arduino Joystick Tutorial**. 2016. Disponível em: <https://brainy-bits.com/tutorials/arduino-joystick-tutorial/>. Acesso em: 10 jun. 2016.

[9] ARDUINOECIA. **Como medir a rotação de um motor com o sensor de velocidade LM393**. 2016. Disponível em: <http://www.arduinoecia.com.br/2016/02/sensor-de-velocidade-lm393-arduino.html>. Acesso em: 10 jun. 2016.

[10] **arduino-serial**. 2016. Disponível em: <https://github.com/todbot/arduino-serial>. Acesso em: 10 jun. 2016.

[11] **Matlab to Eigen**. 2016. Disponível em: <http://igl.ethz.ch/projects/libigl/matlab-to-eigen.html>. Acesso em 20 out. 2016.

[12] LABORATÓRIO DE GARAGEM. **Tutorial: Controlando plataforma robótica através de aplicativo Android com Arduino**. 2016. Disponível em: <http://labdegaragem.com/profiles/blogs/tutorial-controlando-plataforma-robotica-atraves-de-aplicativo-an>. Acesso em 26 nov. 2016.

[13] ARDUINOECIA. **Módulo Joystick Arduino**. 2016. Disponível em: <http://www.arduinoecia.com.br/2013/09/modulo-joystick-arduino.html>. Acesso em: 26 nov. 2016.

[14] RIBEIRO, Lucas Grativol. **Sistema Inteligente de Navegação e Localização de Robôs Móveis**. 2016. Disponível em: <http://www.puc-rio.br/Pibic/relatorio_resumo2014/relatorios_pdf/ctc/ELE/ELE-Lucas20Grativol20Ribeiro.pdf>. Acesso em: 26 nov. 2016.

[15] GIOBLU ROBOTICS. **Robot cartesiano e odometria con Arduino**. 2016. Disponível em: <http://www.gioblu.com/tutorials/robotica/217-robot-cartesiano-e-odometria-con-arduino>. Acesso em: 26 nov. 2016.

[16] BIGHETI, Jeferson André. **NAVEGAÇÃO DE ROBÔS EM AMBIENTES INTERNOS USANDO SLAM**. 2011. 114f. Dissertação (Mestrado em Engenharia Elétrica)–Universidade Estadual Paulista. Faculdade de Engenharia, Bauru.

[17] NOGUEIRA, Marcelo Borges. **Cooperative Vehicle Localization in Networked Systems**. 2013. 238f. Dissertação (Doutorado em Ciências)–Universidade do Porto. Faculdade de Engenharia, Porto.

Apêndice A

Código do algoritmo de Localização

Este apêndice tem como objetivo mostrar o código desenvolvido para resolver o problema da Localização na robótica móvel.

```
1
2  #include <iostream>
3  #include <Eigen/Dense>
4  #include <cmath>
5  #include <stdlib.h>
6  #include <math.h>
7  #define pi 3.1415925
8  #include <stdio.h>
9  #include <string.h>
10 #include <fcntl.h>
11 #include <errno.h>
12 #include <termios.h>
13 #include <unistd.h>
14 #include "arduino-serial-lib.h"
15 //#include <string>
16 using namespace std;
17 using namespace Eigen;
18
19 int fd=-1;
20
21 //int i;
22
23
24 int npassos = 10000000;
25 //int mapsize = 1000;
26 bool observation;
27 float distancia;
28
29 MatrixXd Qtrue(3, 3), Rtrue(2, 2), QEst(3, 3), REst(2, 2),
      InovArm(2,npassos), SArm(2,npassos), PArm(3,npassos), XArm(3,
      npassos), XErrArm(3,npassos), jH(2,3);
30 VectorXd XPosIni(3), LastOdom(3), XOdomLast(3), u(3), xEst(3);
```

```
31 MatrixXd Mapa(2, 2);
32
33
34
35
36 void inicializacao();
37 VectorXd GetOdometry(int _v);
38 VectorXd GetRobotControl(int _k);
39 VectorXd tcomp(VectorXd tab,VectorXd tbc);
40 double AngleWrap(double angle);
41 void SimulateWorld(int i);
42 VectorXd tinv(VectorXd tab);
43 VectorXd tinv1(VectorXd tab);
44 MatrixXd J1(VectorXd xEst,VectorXd vetoru);
45 MatrixXd J2(VectorXd xEst,VectorXd vetoru);
46 VectorXd DoObservationModel(VectorXd xVeh,int iFeature,MatrixXd
      Mapa);
47 MatrixXd GetObsJac(VectorXd xPred,int iFeature,MatrixXd Mapa);
48 VectorXd GetObservation(int i, int &iFeature);
49 int deuclidiana(VectorXd xEst, MatrixXd Mapa);
50
51 int main (void){
52
53
54
55 fd = serialport_init("/dev/ttyACM0", 9600);
56 serialport_flush(fd);
57 usleep(1500);
58
59
60
61 VectorXd XOdomNow(3), vetoru(3), xPred(3), xsemfiltro(3), zPred
      (2), z(2), Innov(2);
62 int iFeature = -1;
63
64
65 MatrixXd PEst(3,3), PPred(3,3), S(2,2), W(3,2), I(3,3);
66
67
68 Mapa(0,0) = 0;
69 Mapa(0,1) = -60;
70 Mapa(1,0) = 60;
71 Mapa(1,1) = 120;
72
73 Qtrue.setZero();
74 Qtrue(0, 0) = (0.01) * (0.01);
75 Qtrue(1, 1) = 0;
```

```
76  Qtrue(2, 2) = 0.087488663;
77
78  Rtrue.setZero();
79  Rtrue(0, 0) = 0.0792;
80  Rtrue(1, 1) = pow((0,26794919/2),2);
81
82
83  QEst = 1.0 * Qtrue;
84
85  REst = 1.0 * Rtrue;
86
87
88  XPosIni(0) = 0;
89  XPosIni(1) = 0;
90  XPosIni(2) = 0;
91
92  LastOdom(0) = 9999;
93  LastOdom(1) = 9999;
94  LastOdom(2) = 9999;
95
96  u = GetRobotControl(1);
97  XOdomLast = GetOdometry(1);
98
99  xEst = XPosIni;
100 xsemfiltro = XPosIni;
101 PEst.setZero();
102 PEst(0,0) = pow(3,2);
103 PEst(1,1) = pow(3,2);
104 PEst(2,2) = pow((5*pi/180),2);
105
106 InovArm.setZero();
107 InovArm = InovArm*NAN;
108 SArm.setZero();
109 SArm = SArm*NAN;
110 PArm.setZero();
111 PArm = PArm*NAN;
112 XArm.setZero();
113 XArm = XArm*NAN;
114 XErrArm.setZero();
115 XErrArm = XErrArm*NAN;
116
117
118 for (int i = 2; i < npassos;i++){
119
120
121
122
```

```
123
124
125 SimulateWorld(i);
126
127 XOdomNow = GetOdometry(i);
128
129 vetoru = tcomp(tinv(XOdomLast),XOdomNow);
130
131 XOdomLast = XOdomNow;
132
133
134 xPred = tcomp(xEst,vetoru);
135 xPred(2) = AngleWrap(xPred(2));
136 PPred = J1(xEst,vetoru)*PEst*((J1(xEst,vetoru)).transpose()) +
       J2(xEst,vetoru)*QEst*((J2(xEst,vetoru)).transpose());
137
138 xsemfiltro = tcomp(xsemfiltro,u);
139
140
141 z = GetObservation(i, iFeature);
142
143
144
145
146 if (observation == true){
147
148 zPred = DoObservationModel(xPred,iFeature,Mapa);
149
150 jH = GetObsJac(xPred,iFeature,Mapa);
151
152
153 Innov = z - zPred;
154 Innov(1) = AngleWrap(Innov(1));
155 S = jH*PPred*jH.transpose()+REst;
156 W = PPred*jH.transpose()*S.inverse();
157 xEst = xPred+ W*Innov;
158 xEst(2) = AngleWrap(xEst(2));
159
160
161 I.setIdentity();
162 PEst = (I-W*jH)*PPred*((I-W*jH).transpose())+ W*REst*W.transpose
       ();
163 PEst = 0.5*(PEst+PEst.transpose());
164 }
165 if (observation == false) {
166 xEst = xPred;
167
```

```
168 PEst = PPred;
169 Innov.setZero();
170 Innov = Innov*NAN;
171 S.setIdentity();
172 S(0,0) = NAN;
173 S(1,1) = NAN;
174
175 }
176
177
178
179
180 InovArm(0,i) = Innov(0);
181 InovArm(1,i) = Innov(1);
182
183 VectorXd Diag1(3), Diag2(2);
184 Diag1 = PEst.diagonal();
185 Diag2 = S.diagonal();
186 PArm(0,i) = sqrt(Diag1(0));
187 PArm(1,i) = sqrt(Diag1(1));
188 PArm(2,i) = sqrt(Diag1(2));
189
190
191 SArm(0,i) = sqrt(Diag2(0));
192 SArm(1,i) = sqrt(Diag2(1));
193
194 XArm(0,i) = xEst(0);
195 XArm(1,i) = xEst(1);
196 XArm(2,i) = xEst(2);
197
198 XErrArm(0,i) = XPosIni(0)-xEst(0);
199 XErrArm(1,i) = XPosIni(1)-xEst(1);
200 XErrArm(2,i) = XPosIni(2)-xEst(2);
201
202
203 }
204
205
206 serialport_close( fd);
207
208 return 0;
209
210 }
211
212
213
214
```

```
215 VectorXd GetOdometry(int _v) {
216
217 VectorXd xnow(3);
218
219 if(LastOdom(0) == 9999 && LastOdom(1) == 9999 && LastOdom(2) ==
       9999){
220 LastOdom = XPosIni;
221 }
222
223
224 xnow = tcomp(LastOdom,u);
225 VectorXd uNoise(3), rando(3);
226
227 rando.setRandom();
228 uNoise = Qtrue.cwiseSqrt()*rando;
229
230
231 xnow = tcomp(xnow,uNoise);
232 LastOdom = xnow;
233 return xnow;
234
235 }
236
237 VectorXd GetRobotControl(int _k){
238 int bufmax = 256;
239 char buf[bufmax];
240 char eolchar = '\n';
241 int timeout = 5000;
242
243
244
245 serialport_write(fd,"A");
246
247
248
249 serialport_read_until(fd, buf, eolchar, bufmax, timeout);
250 float pulsosdir = strtof(buf, NULL);
251
252 serialport_read_until(fd, buf, eolchar, bufmax, timeout);
253 float pulsosesq =  strtof(buf, NULL);
254
255 serialport_read_until(fd, buf, eolchar, bufmax, timeout);
256 float deltat =  strtof(buf, NULL);
257
258 serialport_read_until(fd, buf, eolchar, bufmax, timeout);
259 float dist =  strtof(buf, NULL);
260
```

```
261 distancia = 1.535 + 0.737*dist;
262
263
264 deltat = deltat/1000;
265
266
267 int L = 12;
268 int pulsoporvolta = 40;
269 float r = 3.2;
270
271 float vrd, vre;
272 vrd = (pulsosdir/pulsoporvolta)*2*pi*r/deltat;
273 vre = (pulsosesq/pulsoporvolta)*2*pi*r/deltat;
274 u(0) = 0;
275 u(1) = (vrd + vre)/2*deltat;
276
277 u(2) = (vrd - vre)/L*deltat; //L distancia entre as rodas
278
279
280 return u;
281
282 }
283
284
285 VectorXd tcomp(VectorXd tab, VectorXd tbc){
286 VectorXd tac(3);
287 if(tab.rows() != 3){
288 cout << "TCOMP:  tab IS NOT transformation"<< endl;
289 exit(0);
290 }
291 if(tbc.rows() != 3){
292 cout << "TCOMP:  tbc IS NOT transformation"<< endl;
293 exit(0);
294 }
295
296 double result = tab(2)  + tbc(2);
297 if(result > pi | result <= -pi){
298 result = AngleWrap(result);
299 }
300
301 double s = sin(tab(2));
302 double c = cos(tab(2));
303
304 tac(0) = tab(0) + c*tbc(0) - tbc(1) * s;
305 tac(1) = tab(1) + s*tbc(0) + tbc(1) * c;
306 tac(2) = result;
307
```

```
308
309
310 return tac;
311 }
312
313 double AngleWrap(double angle){
314 if(angle > pi){
315 angle = angle - 2  * pi;
316 }
317 else {
318 if (angle < -pi){
319 angle = angle + 2 * pi;
320 }
321 }
322 return angle;
323 }
324
325
326 void SimulateWorld(int i) {
327 u = GetRobotControl(i);
328 XPosIni = tcomp(XPosIni,u);
329 XPosIni(2) = AngleWrap(XPosIni(2));
330
331
332
333 }
334
335 VectorXd tinv(VectorXd tab){
336
337 VectorXd aux(3);
338 VectorXd aux1(3);
339 VectorXd tba(3);
340 for (int i = 0; i < tab.rows(); i+=3) {
341 aux(0)   = tab(i);
342 aux(1)   = tab(i+1);
343 aux(2)   = tab(i+2);
344 aux1     = tinv1(aux);
345 tba(i)   = aux1(0);
346 tba(i+1) = aux1(1);
347 tba(i+2) = aux1(2);
348 }
349
350 return tba;
351
352 }
353 VectorXd tinv1(VectorXd tab) {
354
```

```
355 VectorXd tba(3);
356 double s = sin(tab(2));
357 double c = cos(tab(2));
358 tba(0) = (-1)*tab(0)*c - tab(1)*s;
359 tba(1) = tab(0)*s - tab(1)*c;
360 tba(2) = (-1)*tab(2);
361
362 return tba;
363
364 }
365
366
367 MatrixXd J1(VectorXd xEst,VectorXd vetoru){
368 double s1, c1;
369 s1 = sin(xEst(2));
370 c1 = cos(xEst(2));
371 MatrixXd Jac(3,3);
372 Jac.setZero();
373 Jac(0,0) = 1;
374 Jac(1,1) = 1;
375 Jac(2,2) = 1;
376 Jac(0,2) = -(vetoru(0))*s1-(vetoru(1))*c1;
377 Jac(1,2) = vetoru(0)*c1-vetoru(1)*s1;
378 return Jac;
379 }
380
381 MatrixXd J2(VectorXd xEst, VectorXd vetoru) {
382 double s1, c1;
383 s1 = sin(xEst(2));
384 c1 = cos(xEst(2));
385 MatrixXd Jac(3,3);
386 Jac.setZero();
387 Jac(2,2) = 1;
388 Jac(0,0) = c1;
389 Jac(0,1) = -s1;
390 Jac(1,0) = s1;
391 Jac(1,1) = c1;
392 return Jac;
393 }
394
395 VectorXd DoObservationModel(VectorXd xVeh,int iFeature, MatrixXd
      Mapa){
396 VectorXd Delta(2),z(2);
397
398 Delta(0) = Mapa(0,iFeature)-xVeh(0);
399 Delta(1) = Mapa(1,iFeature)-xVeh(1);
400
```

```
401 z(0) =  Delta.norm();
402 z(1) =  atan2(Delta(1),Delta(0)) - xVeh(2);
403 z(1) = AngleWrap(z(1));
404
405 return z;
406 }
407
408 MatrixXd GetObsJac(VectorXd xPred,int iFeature,MatrixXd Mapa){
409 MatrixXd jjh(2,3);
410 VectorXd Delta(2);
411 double r;
412
413 jjh.setZero();
414 Delta(0) = Mapa(0,iFeature)-xPred(0);
415 Delta(1) = Mapa(1,iFeature)-xPred(1);
416 r = Delta.norm();
417 jjh(0,0) = -Delta(0)/r;
418 jjh(0,1) = -Delta(1)/r;
419 jjh(1,0) = Delta(1)/(r*r);
420 jjh(1,1) = -Delta(0)/(r*r);
421 jjh(1,2) = -1;
422 return jjh;
423 }
424
425 VectorXd GetObservation(int i,int &iFeature) {
426
427 VectorXd z1(2);
428
429
430 VectorXd r(2),randd(1);
431 r.setRandom();
432 r(0) = r(0)*2;
433 r(1) = r(1)*2;
434 r(0) = r(0)-1;
435 r(1) = r(1)-1;
436 randd.setRandom();
437
438 float y;
439 y = abs(randd(0));
440
441
442 if (distancia < 5 || distancia > 10 ) {
443 iFeature = -1;
444 z1.setZero();
445 z1.setZero();
446 observation = false;
447
```

```
448 }
449 else{
450
451 iFeature = deuclidiana(xEst,Mapa);
452
453
454 z1(0) = distancia;
455 z1(1) = 0;
456 observation = true;
457 }
458
459 return z1;
460 }
461
462 int deuclidiana(VectorXd xEst, MatrixXd Mapa) {
463 double a,b = 100000;
464 int feature;
465 int ncols = Mapa.cols();
466
467 for (int t = 0; t < ncols;t++){
468 a = sqrt(pow(xEst(0)-Mapa(0,t),2)+pow(xEst(1)-Mapa(1,t),2));
469 if (a < b){
470 b = a;
471 feature = t;
472 }
473 }
474
475 return feature;
476 }
477
478
479 }
```