UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Pedro Virgilio Basílio Jeronymo

Sistema de Sensoriamento para Localização de Robô Móvel Autônomo

São Carlos

2019

Pedro Virgilio Basílio Jeronymo

Sistema de Sensoriamento para Localização de Robô Móvel Autônomo

Monografia apresentada ao Curso de Engenharia de Computação, da Escola de Engenharia de São Carlos e Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Carlos Dias Maciel

São Carlos

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Jeronymo, Pedro Virgilio Basílio
J56s Sistema de Sensoriamento para Localização de Robô Móvel Autônomo / Pedro Virgilio Basílio Jeronymo; orientador Carlos Dias Maciel. São Carlos, 2019.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos e Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2019.

1. Sensoriamento. 2. Robô Móvel Autônomo. 3. Localização. 4. Sistema de Navegação Inercial. 5. Dead Reckoning. 6. Fusão sensorial. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Pedro Virgilio Basílio Jeronymo

Título: "Sistema de sensoriamento para localização de robô móvel autônomo"

Trabalho de Conclusão de Curso defendido em 12 / 06 / 2019.

Comissão Julgadora:	**Resultado:**
Prof. Associado Carlos Dias Maciel - Orientador - SEL/EESC/USP	APROVADO
Mestre Victor Hugo Batista Tsukahara Doutorando - SEL/EESC/USP	Aprovado
Mestre Talysson Manoel de Oliveira Santos Doutorando - SEL/EESC/USP	Aprovado

Coordenador do Curso Interunidades Engenharia de Computação:

Prof. Dr. Maximiliam Luppe

AGRADECIMENTOS

Agradeço ao Professor Maciel pela oportunidade de realização deste projeto e por toda a orientação durante os meus anos como aluno de graduação na USP. Talysson e Victor, agradeço pela revisão do texto e auxílio durante a execução do projeto. Obrigado ao Rui por me auxiliar na parte de modificação física do robô. Ao meu amigo e colega de turma Guilherme Prearo, obrigado por me ajudar com o plano de projeto e com o modelo para a monografia. Por fim, sou grato à minha família e à minha namorada Isabela por todo o apoio que me dão.

RESUMO

JERONYMO, P. **Sistema de Sensoriamento para Localização de Robô Móvel Autônomo**. 2019. 79p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

O objetivo deste TCC é iniciar o desenvolvimento de um sistema de localização para um Robô Móvel Autônomo. Através do projeto de um sistema de sensoriamento para o robô e da coleta de dados, foi realizada a análise de duas estratégias para resolver esse problema. A primeira estratégia é a implementação de um Sistema de Navegação Inercial, que independe de um modelo cinemático do robô. No entanto, essa estratégia se mostrou inviável, devido à rápida acumulação de erros sensoriais e ruído. A segunda estratégia utiliza um modelo cinemático do robô. Com dados odométricos e de orientação espacial, implementou-se o chamado *Dead Reckoning*, com resultados melhores, porém agora sofrendo de outra fonte de erros: a derrapagem das rodas. Com o desenvolvimento futuro de um sistema de movimentação para o robô que evite derrapagens, e utilizando-se fusão sensorial com outra fonte de localização, como um sensor de ultrassom, os erros na localização podem ser reduzidos e um sistema completo de localização pode ser implementado.

Palavras-chave: Sensoriamento, Robô Móvel Autônomo, Localização, Sistema de Navegação Inercial, *Dead Reckoning*, Fusão sensorial.

ABSTRACT

JERONYMO, P. **Sensing System for Autonomous Mobile Robot Localization**. 2019. 79p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

The goal of this work is to start the developmente of a localization system for a Autonomous Mobile Robot. Through the design of a sensing system for the robot and data gathering, a analysis of two strategies for solving this problem was conducted. The first strategy is the implementation of a Inertial Navigation System, that is independent of a kinematic model for the robot. However, this strategy showed to be impracticable, due to the rapid accumulation of sensorial erros and noise. The second strategy uses a kinematic model for the robot. With odometric and spacial orientation data, Dead Reckoning was implemented, with better results, although now suffering from a new error source: wheel slipping. In the future, with the development of a motion system for the robot that avoids wheel slipping, and using sensor fusion with another localization system, such as a ultrasound sensor, the erros in localization can be reduced and a full locatization system can be implemented.

Keywords: Sensing, Autonomous Mobile Robot, Localization, Inertial Navigation System, Dead Reckoning, Sensor fusion.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EESC	Escola de Engenharia de São Carlos
USP	Universidade de São Paulo
TCC	Trabalho de Conclusão de Curso
RMA	Robô Móvel Autônomo
GDL	Graus De Liberdade
9AMS	*9 Axis Motion Sensor*
SNI	Sistema de Navegação Inercial
DR	*Dead Reckoning*
GPS	*Global Positioning System*

SUMÁRIO

1 INTRODUÇÃO

Robôs móveis são aqueles que tem a capacidade de locomoção. Eles tem a capacidade de se mover por seu ambiente e não são fixos a uma localização física fixa. Podem ser autônomos (RMA - Robô Móvel Autônomo) o que significa que são capazes de navegar em um ambiente não controlado, sem a necessidade de dispositivos físicos ou eletromecânicos de guia. (Wikipedia contributors, 2019b)

Robôs Móveis tem se tornado mais comuns em cenários comerciais e industriais. Hospitais tem usado RMAs para mover materiais. RMAs são também atualmente um foco de pesquisa em quase todas as grandes universidades. (Wikipedia contributors, 2019b) A multinacional Amazon utiliza RMAs em seus armazéns, que cumprem um papel importante em suas operações. A automação completa não acontecerá por no mínimo 10 anos, pois ainda é necessário muito trabalho e descobertas a serem feitas. (HUMPHRIES, 2019), o que justifica o estudo e o desenvolvimento de RMAs melhores.

No intuito de navegar autonomamente e realizar tarefas úteis, como mapear seu ambiente, um RMA precisa de saber sua posição e orientação exatas. Localização é portanto um problema chave em prover capacidades autônomas a um robô móvel (GOEL P.; ROUMELIOTIS, 1999).

No contexto desse projeto, a navegação de RMAs será discutida através da implementação de um sistema de localização e orientação para um RMA que se move sobre uma superfície bidimensional plana. Uma foto do robô utilizado nesse trabalho pode ser vista na Fig. 1.

O robô é equipado com um Sistema de Direção Diferencial, sistema utilizado por muitos robôs que é essencialmente o mesmo arranjo usado em uma cadeira de rodas (LUCAS, 2000). Quando uma das rodas tem velocidade superior à outra, o robô faz curvas. Caso as velocidades das rodas sejam próximas, o robô anda em linha reta.

No plano de projeto, foi proposto resolver a questão da localização e orientação do robô através da implementação de um Sistema de Navegação Inercial (SNI) (WOODMAN, 2007). Navegação inercial é um uma técnica de navegação em que medidas providas por acelerômetros e giroscópios são utilizadas para acompanhar a posição e orientação de um objeto relativo a uma posição, uma orientação e uma velocidade iniciais. (WOODMAN, 2007)

A Navegação Inercial tem um equacionamento simples em um cenário ideal, mas complexidades surgem devido à erros sensoriais e ruídos no sistema (STOVALL, 1997). Devido à essa complexidade ser maior do que inicialmente imaginada, optou-se por outra estratégia: *Dead Reckoning* (DR), que utiliza dados de odometria e um modelo cinemático

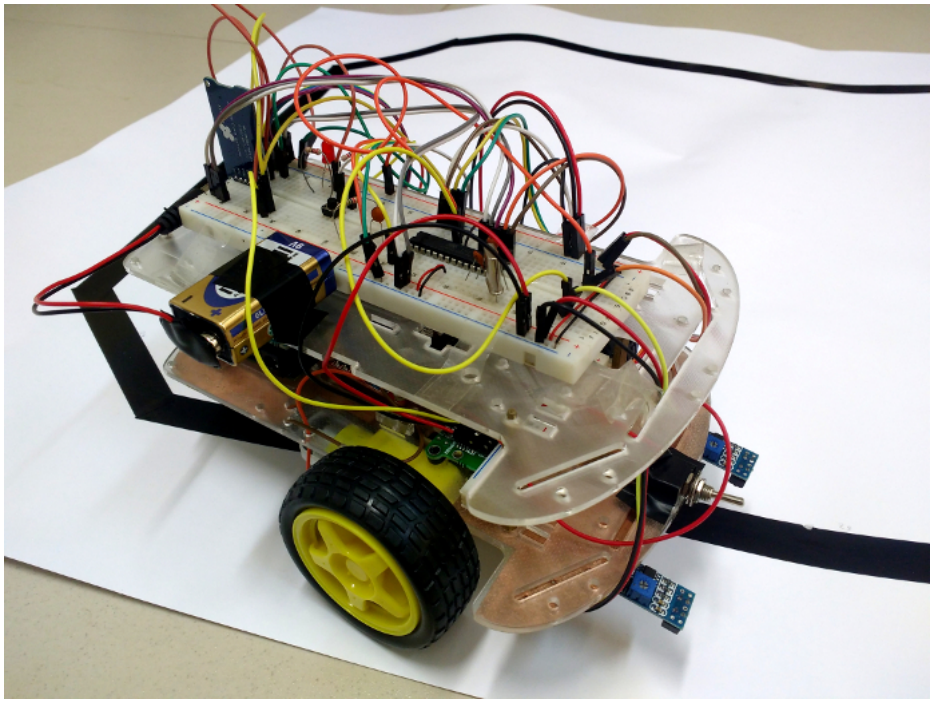

Figura 1: Foto do robô desenvolvido neste trabalho.

para o robô para cumprir a mesma tarefa (GOEL P.; ROUMELIOTIS, 1999).

Utilizando-se *Dead Reckoning*, foram obtidos resultados melhores aos da Navegação Inercial, porém ainda não suficientes para localização e orientação do robô, devido à outros erros sensoriais e ruídos no sistema.

Como será discutido em 2.3, 5 e 6, com a fusão dos dados de ultrassom com o sistema proposto nesse TCC, e realizando as melhorias necessárias na movimentação do robô, um sistema completo de navegação pode ser desenvolvido futuramente.

Sobre a execução do projeto, o primeiro passo foi definir o conjunto de sensores a serem utilizados, então foi projetado um sistema de coleta dos dados fornecidos por esses sensores. O sistema de sensoriamento é discutido em 3.3.

Os sensores escolhidos foram codificadores de roda para realizar a odometria e o sensor de orientação absoluta **BNO055** (BNO055,). Esse sensor é capaz de fornecer a sua orientação espacial em relação ao norte magnético da terra, através da fusão sensorial de magnetômetros, giroscópios e acelerômetros. Ele também fornece outros dados sofisticados, discutidos na seção 2.2.

Para realizar uma trajetória previsível para testes, foi implementado um sistema de seguidor de linha (CANDIDO, 2018), discutido em 3.2. Enquanto os sinais de controle das rodas são enviados pelo sistema seguidor de linha, o sistema de sensoriamento armazena em um cartão SD as leituras dos sensores.

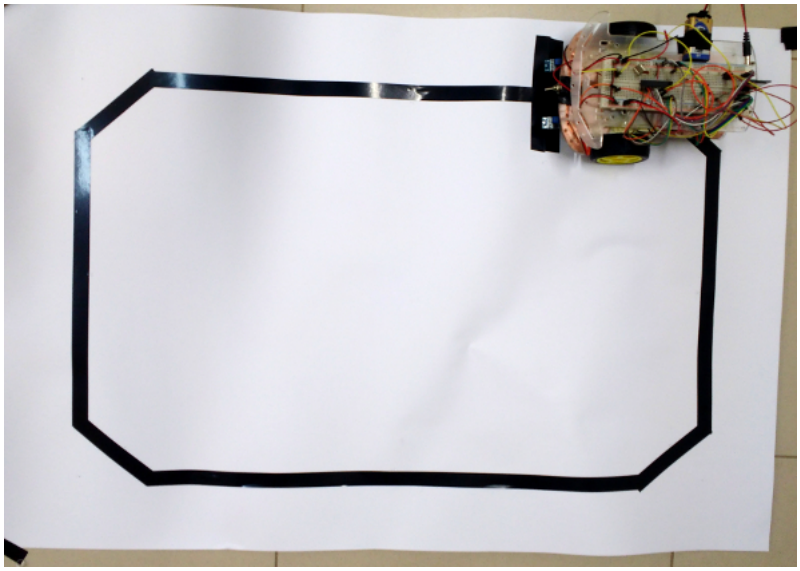

Figura 2: Foto da pista de teste, confeccionada para sensoriamento e validação das estratégias de estimativa de trajetória do robô, com o robô sobre ela.

Com os dois sistemas funcionando simultaneamente, o robô realizou algumas voltas em uma pista de teste confeccionada utilizando uma folha A2 e fita isolante para o trajeto. A Fig. 2 mostra uma foto da pista com o robô sobre ela.

Para validar cada estratégia (SNI e DR), a estimativa da posição e orientação do robô em função do tempo foi realizada a partir dos dados coletados. A trajetória estimada computacionalmente foi comparada com a trajetória esperada. Os critérios de decisão foram a distância percorrida e a aderência visual do trajeto estimado ao trajeto esperado.

Alguns problemas na movimentação do robô foram observados, sendo o mais crítico a derrapagem das rodas, que foi contornada nas estimativas, mas tem que ser resolvida para a locomoção e estimativa de localização correta do robô.

O texto dessa monografia é dividido em cinco capítulos, sendo o primeiro este. O capítulo 2 discute os aspectos teóricos do trabalho, mas já abordando questões de implementação. O capítulo 3 aborda como foi implementada a teoria apresentada, a fim de se obter os resultados, apresentados no capítulo 4. Então, os resultados são discutidos no capítulo 5. Por fim, o capítulo 6 apresenta a conclusão desse TCC.

2 TEORIA

2.1 Amostragem

A operação de amostragem gera um sinal discreto no tempo de um sinal contínuo no tempo. A amostragem de sinais contínuos é frequentemente realizada para que o sinal possa ser manipulado em um computador ou microprocessador (HAYKIN S.; VEEN, 2002).

A amostragem de um sinal contínuo $x(t)$ é realizada aferindo o valor de $x(t)$ em instantes regulares de tempo $t = kT_s$, em que $k = 0, ..., N-1$ e Ts é o período de amostragem. O sinal resultante x_k é uma representação digital de $x(t = kT_s)$ em cada instante de amostragem k, para N instantes de amostragem.

Para que não haja perda de informação no processo de amostragem, deve-se respeitar o Teorema da Amostragem.

> **Teorema da Amostragem** Se um sinal $x(t)$ não contém frequências maiores que B hertz, ele é completamente determinado dados os seus valores em uma série de pontos espaçados em $(1/2B)$ segundos. Uma taxa de amostragem F_s suficiente é portanto qualquer uma maior que $2B$ amostras por segundo: $F_s > 2B$ (Wikipedia contributors, 2019c).

2.2 Sensores

Fusão sensorial é a combinação de dados sensoriais ou dados de fontes diferentes de maneira que a informação resultante apresenta menos incerteza do que seria possível ao se utilizar cada uma dessas fontes individualmente (Wikipedia contributors, 2019f).

Os sensores utilizados nesse projeto são o sensor de orientação absoluta **BNO055** (BNO055,) e um codificador óptico incremental (INCREMENTAL. . . ,) para cada roda do robô.

O **BNO055** usa como referência o norte magnético, por isso é chamado de sensor de orientação absoluta. Ele possui três acelerômetros, três giroscópios e três magnetômetros. **Acelerômetros** medem aceleração em seu referencial inerente, **giroscópios** medem velocidade angular e **magnetômetros** medem intensidade do campo magnético ao seu redor. O **BNO055** vem integrado a um *shield*, que facilita a sua conexão ao Arduino, chamado *9 Axis Motion Sensor* (9AMS).

Com um de cada tipo desses sensores para cada direção espacial (x,y,z) o **BNO055** é capaz de realizar fusão sensorial e fornecer informações sofisticadas como:

- Orientação espacial em relação ao Norte Magnético da Terra em dois formatos:

Ângulos de Euler e Quaternião;

- Retornar a aceleração em cada direção espacial, descontando a influencia da gravidade, chamadas de *acelerações lineares*;
- Retornar apenas um vetor representando a aceleração gravitacional.

Um **codificador óptico incremental** (INCREMENTAL...,) provê um meio de se medir a posição de uma roda ou motor através da codificação de pulsos em um disco com furos igualmente espaçados utilizando um sensor infravermelho. Conforme a roda gira, um trem de pulsos é formado com frequência proporcional à velocidade angular da roda. Cada pulso representa um passo dado pela roda.

Através do quaternião de orientação espacial, pode-se obter o ângulo θ entre o referencial do robô (ou dos sensores) e o referencial inercial do observador; definidos na subseção 2.3.1. As acelerações lineares são utilizadas no Sistema de Navegação Inercial proposto em 2.4.

O quaternião de orientação espacial fornecido pelo **BNO055** tem como referência o norte magnético da terra. Como pode ser visto na Fig. 9, na seção de resultados, o ângulo inicial do robô em cada volta é próximo de *zero* pois a pista foi convenientemente posicionada em alinhamento com o norte magnético.

Os codificadores são utilizados em 2.5. São contadas as quantidades de pulsos (ou passos dados pela roda) $n_{d,k}$ e $n_{e,k}$, para as rodas direita e esquerda respectivamente, ocorridas entre os instantes de amostragem $k-1$ e k.

2.3 Localização

As técnicas diferentes para se resolver o problema da localização podem ser classificadas em duas categorias principais (GOEL P.; ROUMELIOTIS, 1999):

- **Localização Relativa (local)**: avaliar a posição e orientação baseado em informação fornecida por sensores embarcados (codificadores, giroscópios, acelerômetros, etc);
- **Localização absoluta (global)**: obter a posição absoluta usando guias, pontos de referência ou sinais geoestacionários (GPS).

SNIs e *Dead Reckoning* são da primeira categoria. O erro na localização cresce com o tempo. Os métodos absolutos, como GPS conseguem medidas independentes das anteriores, portanto, o erro não acumula de uma medida para a próxima, porém muitas vezes não é possível se utilizar GPS, especialmente para pequenas distâncias ou portas a dentro. (GROVES, 2008)

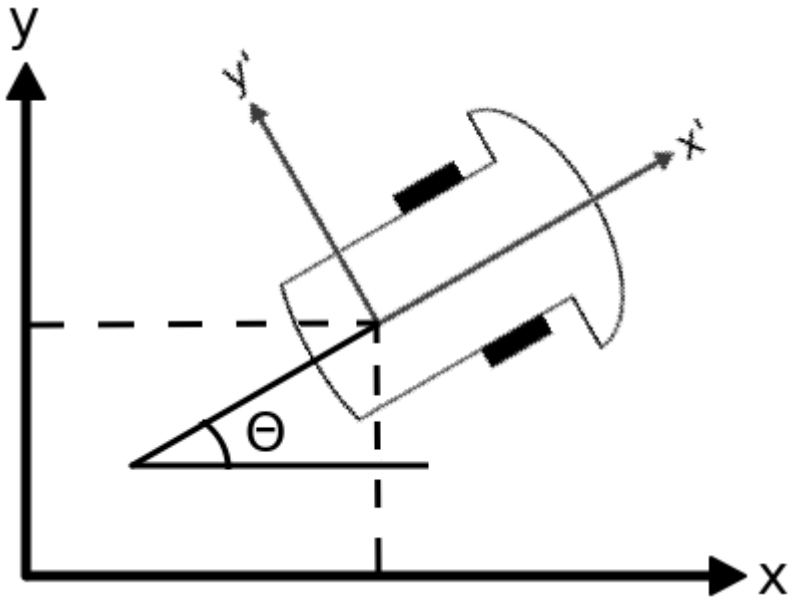

Figura 3: Sistemas de Coordenadas do Robô Didático Móvel - O sistema sem linha (x-y) descreve o referencial inercial à qual as coordenadas do robô são estimadas. O sistema com linha (x'-y') é inerente às medidas realizadas pelos sensores embarcados no robô.

Um sensor de ultrassom possui características de um sistema absoluto de orientação, pois mede a distância entre o robô e os objetos ao seu redor, sendo cada medida independente da anterior. Com *Dead Reckoning* provendo correções à localização obtida através do sensor de ultrassom, cuja integração entre os sistemas podendo ser realizada por um Filtro de Kalman (LEVY,), é possível se construir uma solução completa de localização. (GROVES, 2008).

2.3.1 Referenciais para Localização

Um robô móvel, ou veículo, possui 6 graus de liberdade (GDL) expressados pela *pose*: $(x, y, z, Roll, Pitch, Yaw)$. Informalmente, *Roll* pode ser definido como sendo a rotação lateral e *Pitch* a rotação para frente e para trás. *Yaw*, comumente denominado de *Heading* ou *Orientação*, refere-se à direção a qual o robô se move no plano x-y. Para um robô em uma superfície bidimensional, a *pose 2D* (x, y, θ), em que θ denota Orientação, é suficiente para descrever seu movimento. (HELLSTRöM, 2011) Ou seja, a movimentação espacial se restringe a 3 GDL.

A Fig. 3 descreve os sistemas de coordenadas presentes no problema. O sistema de coordenadas sem linha (x-y) descreve o referencial inercial à qual as coordenadas do robô são estimadas, esse é o sistema de interesse. O sistema com linha (x'-y') é inerente às medidas realizadas pelos sensores embarcados no robô (acelerômetros, giroscópios, magnetômetros, codificadores para rodas).

2.3.2 Quaterniões

Quaterniões são uma extensão dos números complexos, geralmente representados na forma:

$$\mathbf{q} = (q_w, q_x, q_y, q_z) = (a, b, c, d) = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$$

A aritmética dos quaterniões é similar à dos números complexos. A soma de $\mathbf{q}_1 = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ e $\mathbf{q}_2 = e + f\mathbf{i} + g\mathbf{j} + h\mathbf{k}$ é $(a + e) + (b + f)\mathbf{i} + (c + g)\mathbf{j} + (d + h)\mathbf{k}$. O conjugado de $\mathbf{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ é $\mathbf{q}^{-1} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$. E a multiplicação entre quaterniões é também como a dos números complexos mas seguindo a tabela 1. Nota-se que, diferentemente dos complexos, a multiplicação de quaterniões não é comutativa.

Tabela 1: Tabela multiplicativa dos quaterniões.

×	**1**	**i**	**j**	**k**
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

2.3.3 Quaterniões para orientação espacial

Um quaternião é dito unitário quando seu módulo igual **1**:

$$|\mathbf{q}| = q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$$

Quaterniões unitários fornecem uma notação matemática conveniente para representar orientações e rotações de objetos em três dimensões.

Neste projeto, um quaternião $\mathbf{q_k} = (q_{w,k}, q_{x,k}, q_{y,k}, q_{z,k})$ é fornecido a cada instante de amostragem k, representando a rotação em θ_k graus entre (x-y) e (x'-y') naquele instante. Como a movimentação é restrita à uma superfície bidimensional plana, as componentes q_x e q_y são aproximadamente *zero*.

Para que seja possível realizar a estimativa de posição do robô, deseja-se transformar as leituras feitas no referencial (x'-y') para o referencial (x-y). Realiza-se isso aplicando o procedimento a seguir.

Seja $u' = (u'_x, u'_y)$ um vetor no referencial (x'-y'), transforma-se suas coordenadas para o vetor equivalente $u = (u_x, u_y)$ no referencial (x-y) através da fórmula:

$$\begin{pmatrix} u_x \\ u_y \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} u'_x \\ u'_y \end{pmatrix} \tag{2.1}$$

em que $\sin(\theta) = 2q_w q_z$ e $\cos(\theta) = q_w q_w - q_z q_z$.

Utilizou-se quaterniões para se evitar problemas de mudança brusca de ângulo quando o robô cruza os 180°, como pode ser notado na Fig. 9, mostrada nos resultados. Mais detalhes sobre quaterniões e suas aplicações em rotação espacial podem ser encontrados em (Wikipedia contributors, 2019d), (Wikipedia contributors, 2019e) e (Wikipedia contributors, 2019a).

2.4 Sistema de Navegação Inercial

A partir do sensor de posição absoluta **BNO055** embarcado no robô, é possível, em teoria, desenvolver um Sistema de Navegação Inercial (SNI) utilizando os dados de *orientação espacial* e *aceleração linear* fornecidos pelo sensor.

A partir o quaternião de orientação espacial q_k e das acelerações lineares $a'_{x,k}$ e $a'_{y,k}$ medidos a cada instante de amostragem k, pode-se estimar a posição e velocidade do robô seguindo o procedimento:

1. Transformar $a'_{x,k}$ e $a'_{y,k}$ medidos no referencial (x'-y') nos vetores equivalentes em (x-y) $a_{x,k}$ e $a_{y,k}$ utilizando a transformação linear 2.1.

2. Integrar $a_{x,k}$ e $a_{y,k}$ em função do tempo para se obter as velocidades $v_{x,k}$ e $v_{y,k}$.

3. Integrar $v_{x,k}$ e $v_{y,k}$ em função do tempo para se obter as coordenadas x_k e y_k do robô.

A integração dos vetores de aceleração e velocidade são realizadas através da aproximação discreta de uma integral, que é um somatório:

$$F(t) = \int_0^t f(\tau)d\tau \sim F_k = \sum_{j=0}^{j=k} f_j \Delta t_j$$

2.5 *Dead Reckoning*

Essa seção discute o desenvolvimento das equações de movimento implementadas neste projeto, que modelam um robô com Sistema de Direção Diferencial.

Partindo de equações de movimento que utilizam apenas as leituras $n_{d,k}$ e $n_{e,k}$ dos codificadores de roda, inclui-se as informações de orientação espacial fornecidas pelo sensor **BNO055**. Por fim, modela-se a derrapagem, incluindo-a também nas equações, até se obter o modelo cinemático final.

2.5.1 Equações cinemáticas apenas com dados de odometria

Considerando o sistema de coordenadas apresentado, se temos no instante k a pose (x_k, y_k, θ_k), para se estimar a pose $(x_{k+1}, y_{k+1}, \theta_{k+1})$, utilizamos as equações de movimento do robô, cuja derivação pode ser encontrada em (HELLSTRöM, 2011).

- Seja Δt_k o tempo corrido entre o instante $k-1$ e o instante k;
- Seja $n_{d,k}$ e $n_{e,k}$ o número de passos contados pelos codificadores das rodas direita e esquerda, respectivamente, entre os instantes $k-1$ e k;
- Seja *step* o tamanho do passo do robô;

$$\begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix} = \begin{pmatrix} \cos(\omega_k \Delta t_k) & -\sin(\omega_k \Delta t_k) & 0 \\ \sin(\omega_k \Delta t_k) & \cos(\omega_k \Delta t_k) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{k-1} - ICC_{x,k} \\ y_{k-1} - ICC_{y,k} \\ \theta_{k-1} \end{pmatrix} + \begin{pmatrix} ICC_{x,k} \\ ICC_{y,k} \\ \omega_k \Delta t_k \end{pmatrix} \tag{2.2}$$

em que

$$R_k = \left(\frac{L}{2}\right) \frac{n_{d,k} + n_{e,k}}{n_{d,k} - n_{e,k}} \tag{2.3}$$

$$\omega_k \Delta t_k = \frac{(n_{d,k} - n_{e,k}) step}{L} \tag{2.4}$$

$$ICC_k = [x_k - R_k \sin(\theta_k), y_k + R_k \cos(\theta_k)] \tag{2.5}$$

2.5.2 Equações cinemáticas com odometria e orientação espacial

As equações de movimento descritas em (HELLSTRöM, 2011) (2.2, 2.3, 2.4 e 2.5) estimam a pose do robô baseando-se apenas nas leituras dos codificadores de roda. Para uma melhor estimativa, pode-se utilizar o quaternião de orientação espacial $\mathbf{q} = (q_w, q_x, q_y, q_z)$ fornecida pelo **BNO055** para se obter $\cos(\theta)$, $\sin(\theta)$ como discutido na subseção 2.3.3, e $\Delta\theta$ como discutido a seguir.

A derivada de $\sin(\Delta\theta)$ em relação a $\Delta\theta$ é $\cos(\Delta\theta)$. No caso discretizado, pode ser aproximada por:

$$(\sin(\Delta\theta_{k-1}))' = \cos(\Delta\theta_{k-1}) \approx \frac{\sin(\Delta\theta_k) - \sin(\Delta\theta_{k-1})}{\theta_k - \theta k - 1} = \frac{\sin(\Delta\theta_k) - \sin(\Delta\theta_{k-1})}{\Delta\theta_k} \tag{2.6}$$

Logo, aproxima-se $\Delta\theta_k$ o isolando na equação 2.6:

$$\Delta\theta_k \approx \frac{\sin(\Delta\theta_k) - \sin(\Delta\theta_{k-1})}{\cos(\Delta\theta_{k-1})} \tag{2.7}$$

Para se incluir 2.7 nas equações de movimento, deve-se notar que:

- $\omega_k \Delta t_k = \Delta\theta_k$
- A partir de 2.3, 2.4 e da relação anterior pode-se chegar em: $R_k = \left(\frac{step}{2}\right)\frac{n_{d,k}+n_{e,k}}{\Delta\theta_k}$

Com essas substituições, simplifica-se as equações de movimento para:

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} \cos(\Delta\theta_k) & -\sin(\Delta\theta_k) \\ \sin(\Delta\theta_k) & \cos(\Delta\theta_k) \end{pmatrix} \begin{pmatrix} x_{k-1} - ICC_{x,k} \\ y_{k-1} - ICC_{y,k} \end{pmatrix} + \begin{pmatrix} ICC_{x,k} \\ ICC_{y,k} \end{pmatrix} \tag{2.8}$$

em que

$$\Delta\theta_k = \frac{\sin(\theta_k) - \sin(\theta_{k-1})}{\cos(\theta_{k-1})} \tag{2.9}$$

$$R_k = \left(\frac{step}{2}\right)\frac{n_{d,k} + n_{e,k}}{\Delta\theta_k} \tag{2.10}$$

$$ICC_k = [x_k - R_k \sin(\theta_k), y_k + R_k \cos(\theta_k)] \tag{2.11}$$

2.5.3 Equações cinemáticas modelando derrapagem

Para se contornar o problema da derrapagem, optou-se por uma solução simples, *ad hoc* e baseada na intuição. Apesar de não ter respaldo acadêmico, acabou obtendo resultados congruentes com o esperado se não houve-se derrapagem.

Modelou-se a velocidade angular observada da roda ($\hat{\Omega} = \frac{step \cdot \hat{n}}{\Delta t}$) como sendo diretamente proporcional à velocidade angular esperada caso não houvesse derrapagem (Ω). Tem-se que:

$$\Omega \propto \hat{\Omega} \Rightarrow \frac{step \cdot n}{\Delta t} \propto \frac{step \cdot \hat{n}}{\Delta t} \Rightarrow n \propto \hat{n}$$

Seja γ o coeficiente de proporcionalidade, que chamaremos de *coeficiente de derrapagem*, tem-se que

$$n_d = \gamma \hat{n_d}$$

$$n_e = \gamma \hat{n_e}$$

Logo,

$$\Rightarrow R_k = \left(\frac{step}{2}\right)\frac{n_{d,k} + n_{e,k}}{\Delta\theta_k} = \left(\frac{step}{2}\right)\frac{(\gamma \hat{n_{d,k}}) + (\gamma \hat{n_{e,k}})}{\Delta\theta_k}$$

Portanto,

$$R_k = \gamma\left(\frac{step}{2}\right)\frac{\hat{n_{d,k}} + \hat{n_{e,k}}}{\Delta\theta_k} \tag{2.12}$$

Para estimar γ, utilizou-se o comprimento do trajeto realizado por cada roda durante a volta. E então, fez-se a razão entre o comprimento de trajeto esperado para cada roda e o comprimento registrado. γ foi considerado a média dessas duas razões.

O comprimento de trajeto esperado para a roda direita é de 236 cm, pois é o comprimento da pista. Se considerarmos que a roda esquerda idealmente fica parada durante as quatro curvas presentes no trajeto, cada uma com 10 cm, o comprimento esperado para a roda esquerda é de $236 - 4 \times 10 = 196$ cm.

Os comprimentos dos trajetos das rodas são dados por:

$$L_d = step \cdot \sum_{k=0}^{k=N-1} n_d, L_e = step \cdot \sum_{k=0}^{k=N-1} n_e \tag{2.13}$$

A fórmula para γ é:

$$\gamma = \left(\frac{1}{2}\right)\left(\frac{236}{L_d} + \frac{196}{L_e}\right) \tag{2.14}$$

As equações cinemáticas do robô modelando derrapagem são as equações 2.8, 2.9 e 2.11, apresentadas na subseção 2.5.2, com a utilização da equação 2.12 em vez da equação 2.10.

2.6 Sistema de Navegação Inercial *vs Dead Reckoning*

SNIs são autocontidos (WOODMAN, 2007), ou seja, independem de um modelo cinético de movimentação veículo em que estão embarcados. Já *Dead Reckoning* depende de um modelo cinético.

Dead reckoning, assim como um SNI, não podem ser usados por longas distâncias, porque sofrem com várias desvantagens. O modelo cinético sempre sofre de inacurárias, codificadores tem precisão limitada e existem diversas fontes externas de erro afetando a movimentação que não sao observáveis pelos sensores, por exemplo derrapagem das rodas. O erro de localização cresce com o tempo (GOEL P.; ROUMELIOTIS, 1999).

No caso de um SNI é simples ver que o erro na velocidade cresce linearmente com o tempo e o erro na posição cresce de forma quadrática. Seja δa o erro na medida da aceleração $a(t)$. Seja $\bar{a}(t)$ a aceleração observada na prática.

Logo,

$$\bar{a}(t) = a(t) + \delta a$$

Ao se integrar $\bar{a}$(t) em relação ao tempo, obtém-se:

$$\bar{v}(t) = \int_0^t \bar{a}(\tau)\, d\tau = \int_0^t a(\tau) + \delta a\, d\tau = v(t) + \delta a \cdot t$$

Ao se integrar $\bar{v}(t)$ em relação ao tempo temos:

$$\bar{x}(t) = \int_0^t \bar{v}(\tau)\, d\tau = \int_0^t v(\tau) + \delta a \cdot \tau\, d\tau = x(t) + \delta a \cdot t^2$$

3 IMPLEMENTAÇÃO

3.1 Eletromecânica

O funcionamento eletromecânico do robô se da a partir de sinais de controle enviados por um Arduino (Arduino. . . ,) à uma Ponte-H (WIKIPéDIA, 2019) que envia corrente aos atuadores das rodas, fazendo-as rodar conforme os sinais de controle. Cada roda tem quatro estados de operação: *para frente*, *reverso*, *parada rápida do motor* e *parada livre do motor*.

Para alimentar a Ponte-H, uma bateria 9V fixada na parte inferior do robô foi utilizada. Para alimentar o Arduino e o microcontrolador foi usada outra bateria 9V fixada na lateral do robô, esse arranjo pode ser vistos na Fig. 1.

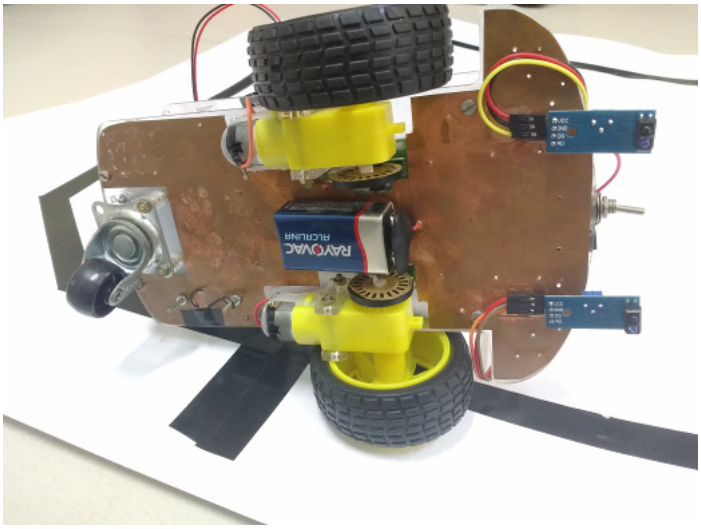

Figura 4: Foto mostrando a parte de baixo do robô, em que pode-se ver a bateria que alimenta a Ponte-H, os discos furados para codificação do giro das rodas e os atuadores que movimentam as rodas.

Na Fig. 4, pode-se ver as rodas, que tem 65mm de diâmetro, os atuadores, e os discos codificadores das rodas, com 20 furos cada. Além disso, pode-se ver a bateria que alimenta a Ponte-H.

Na Fig. 5 pode-se ver o Arduino com o *shield* que contém o sensor **BNO055** na parte esquerda. Ao meio, pode-se ver a Ponte-H e à esquerda os sensores infravermelhos, que junto aos discos compõem os codificadores ópticos incrementais, como discutido na seção 2.2.

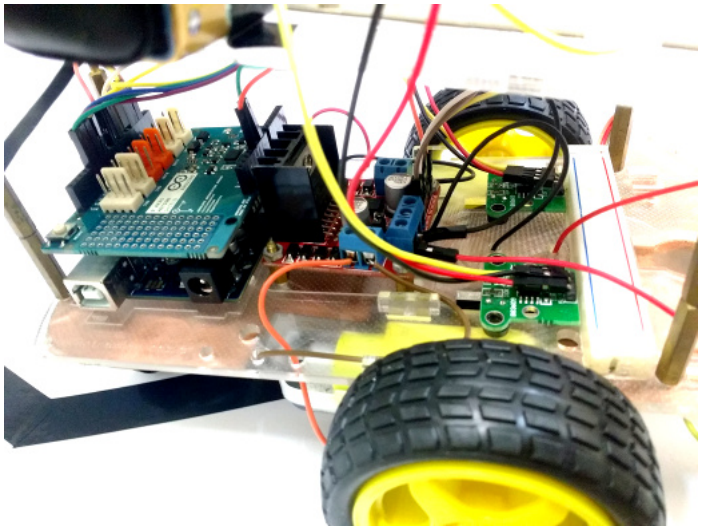

Figura 5: Foto mostrando a parte interior do robô, aonde estão localizados o Arduino com o *shield* que contém o sensor **BNO055**, a Ponte-H e os sensores infravermelhos do codificador de roda.

3.2 Seguidor de linha

Um microcontrolador *ATMEGA328PU* (MEGAAVR...,), mesmo microcotrolador presente no Arduino (Arduino...,), foi utilizado para implementar um seguidor de linha. Optou-se por se utilizar apenas o microcontrolador, em vez do Arduino completo , por conveniência. O código para o seguidor de linha está no apêndice B. O esquemático do seguidor de linha está na Fig. 6.

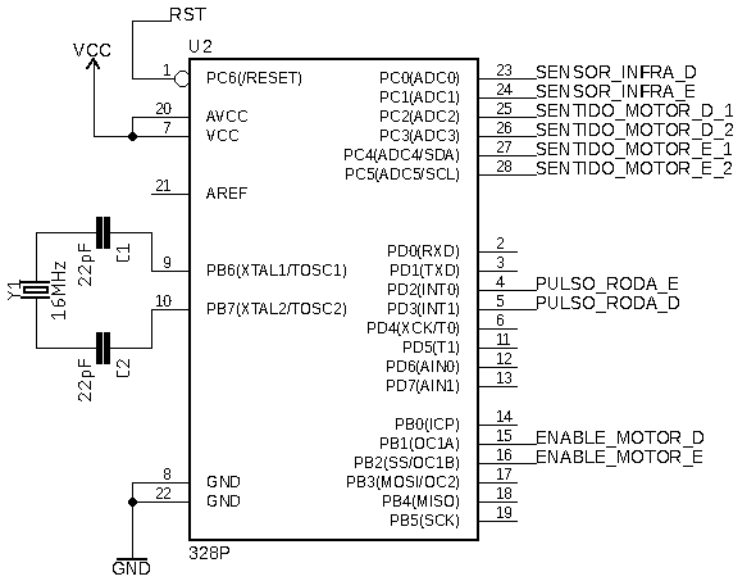

Figura 6: Esquemático do seguidor de linha, mostrando as ligações feitas no microcontrolador ATMEGA328PU.

Dois sensores infravermelhos para seguidor de linha (inserir modelo) foram fixados na frente do robô, com 6,5 cm de distância entre eles. O sensor possui um receptor e um emissor de infravermelho, lado a lado, que podem ser vistos na Fig. 4. O sensor é ativo quando a luz refletida pelo emissor de infravermelho atinge o receptor de infravermelho, ou seja, quando a superfície é branca ou reflexiva. Quando a superfície é escura, ou pouco reflexiva, o sensor é inativo. A Fig. 7 ilustra os estados possíveis do segue linha aqui descritos.

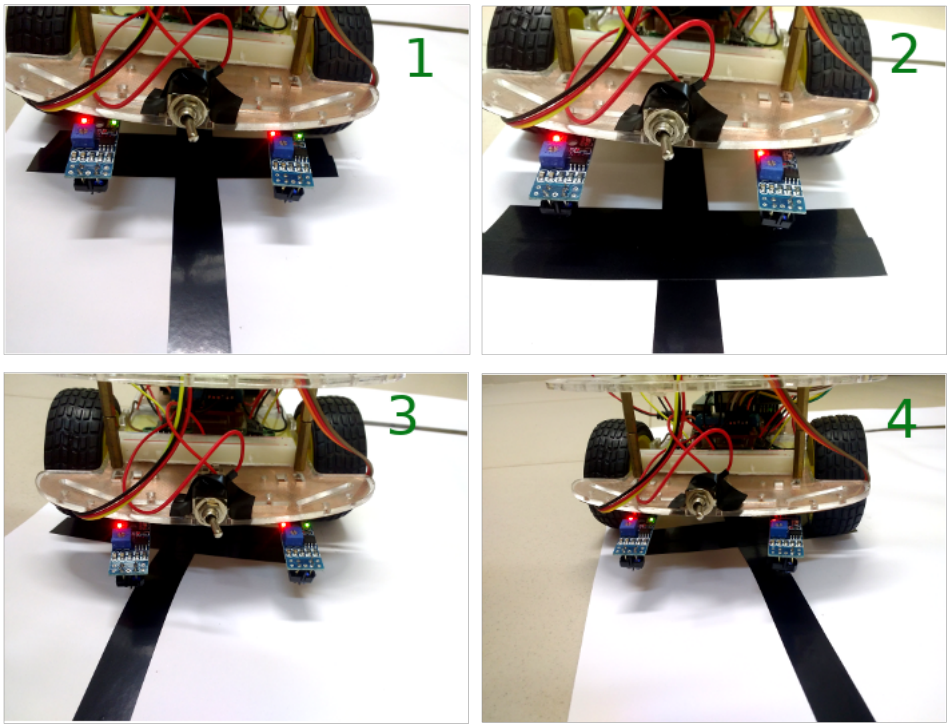

Figura 7: Fotos ilustrando os estados possíveis do Seguidor de Linha. No estado 1, o robô anda para frente. No estado 2, o robô fica parado sobre a linha de chegada. Os estados 3 e 4 mostram as curvas para a direita e esquerda respectivamente.

Enquanto ambos os sensores estão ativos, o robô caminha para frente. Quando o sensor direito fica inativo, o robô vira para a direita, até ambos os sensores ficarem ativos novamente. Quando o sensor esquerdo fica inativo, o comportamento é espelhado. Dessa forma, o robô segue o trajeto traçado na pista confeccionada para testes, com papel A2 e fita isolante. Quando ambos os sensores ficam inativos, ou seja, quando encontra com a linha de chegada, o robô para.

3.3 Sistema de sensoriamento

O código para o sistema de sensoriamento está no apêndice A. A cada 10 ms foram coletadas as seguintes informações detalhadas na tabela 2, salvas em um arquivo

Tabela 2: Detalhamento das informações coletadas pelo sistema de sensoriamento.

Nome	Tipo	Fonte	Descrição
Tempo corrente	Inteiro	Arduino	Tempo corrente desde o início da execução do programa
Calibração Acc	Categórico	BNO055	Status de calibração dos acelerômetros (0-3)
Calibração Mag	Categórico	BNO055	Status de calibração dos magnetômetros (0-3)
Calibração Gir	Categórico	BNO055	Status de calibração dos giroscópios (0-3)
Calibração Sys	Categórico	BNO055	Status de calibração do sistema (0-3)
linAccX	Real	BNO055	Aceleração Linear em X
linAccY	Real	BNO055	Aceleração Linear em Y
linAccZ	Real	BNO055	Aceleração Linear em Z
quatW	Real	BNO055	Componente W do quaternião de orientação espacial
quatX	Real	BNO055	Componente X do quaternião de orientação espacial
quatY	Real	BNO055	Componente Y do quaternião de orientação espacial
quatZ	Real	BNO055	Componente Z do quaternião de orientação espacial
Num. Dir.	Inteiro	Codificador	Número de pulsos entre a amostragem anterior e agora da roda direita
Num. Esq.	Inteiro	Codificador	Número de pulsos entre a amostragem anterior e agora da roda esquerda

binário no cartão SD. Usando o código C, os dados foram convertidos para o formato *Comma-separated values* para mais fácil importação nos demais códigos em Python.

O esquemático do coletor de dados, suprimento as ligações do *shield* 9AMS, está na Fig. 8. Os sensores de codificação das rodas não estão representados, por simplicidade, apenas os pinos de conexão no microcontrolador estão presentes.

A escolha do período amostral $T_s = 10$ ms se deu devido à frequência máxima de saída de dados do **BNO055**, que é de 100 Hz para as saídas de fusão de dados utilizadas. Assume-se que essa taxa de amostragem respeita o Teorema da Amostragem (2.1), pois o sensor foi projetado para esse tipo de aplicação (BNO055,).

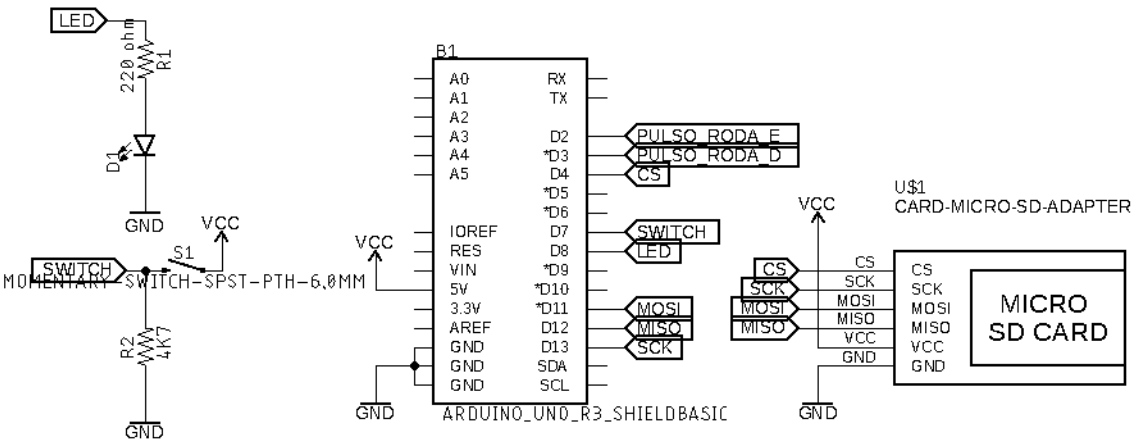

Figura 8: Esquemático do sistema de sensoriamento, suprimindo as ligações do *shield* 9AMS.

4 RESULTADOS

Esse capítulo apresenta os resultados experimentais do projeto. Para as Figs. com as trajetórias estimadas, se incluiu uma representação digital da pista confeccionada para testes, além de duas pistas tracejadas. Como a distância entre os sensores infravermelhos do seguidor de linha é de 6,5 cm, as pistas tracejadas demarcam para dentro e para fora essa margem. Com os códigos disponíveis nos apêndices, calculou-se que o cumprimento da trajetória mínima é de 188 cm e o comprimento da trajetória máxima é de 284 cm. Lembrando que o comprimento nominal é de 236 cm.

Além das trajetórias estimadas, são incluídos alguns gráficos com a evolução temporal do ângulo de orientação espacial θ, das coordenadas, velocidades e acelerações relevantes para a discussão. Há uma análise do intervalo de tempo entre uma amostra e outra, ou seja, sobre a variação na prática do período amostral.

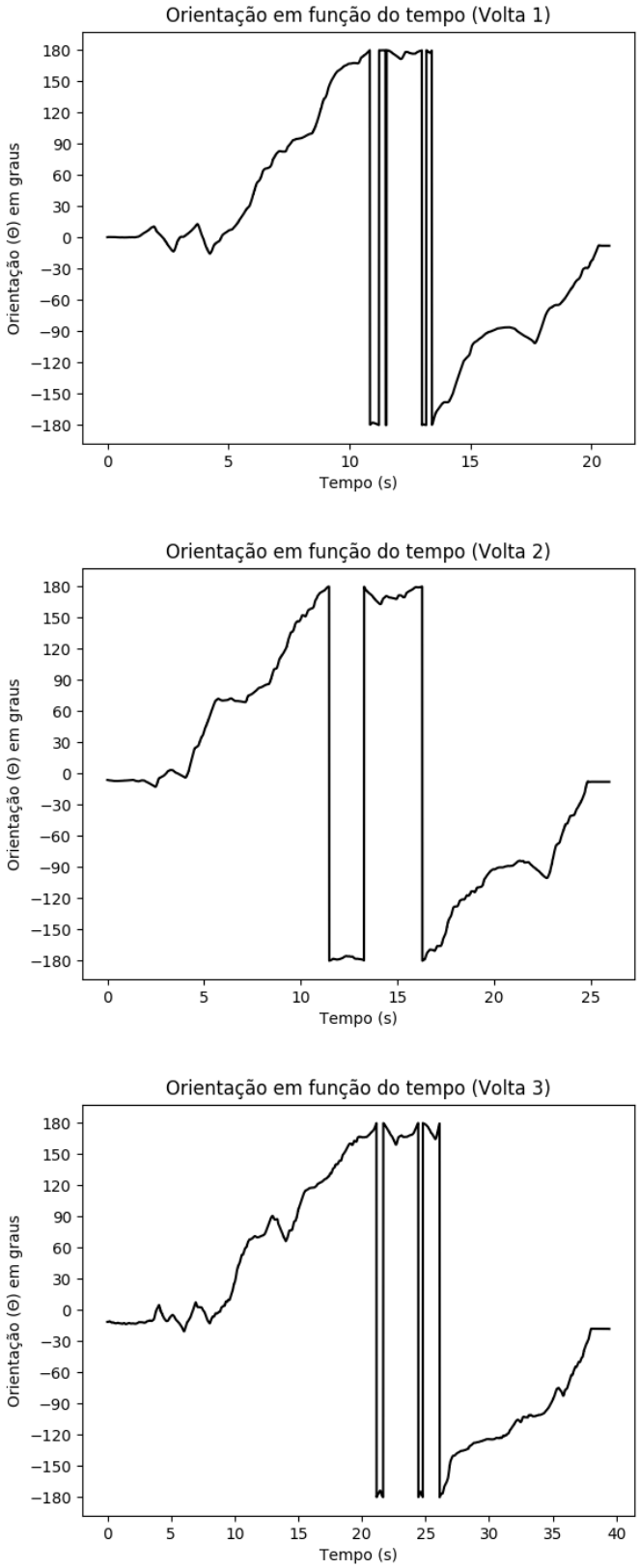

Figura 9: Evolução temporal do ângulo de orientação espacial θ para cada volta realizada pelo robô.

4.1 Intervalo de tempo entre amostras

A Fig. 10 mostra um histograma do intervalo de tempo entre as amostras coletadas, com o eixo vertical em escala logarítmica para melhor visualização. O período de amostragem nominal é de 10 ms, porém, como pode-se observar, esse tempo variou devido a fatores aleatórios discutidos no capítulo 5.

A mediana do tempo entre amostras é de 10 ms, e 95% das vezes o tempo foi menor que 14 ms, ou seja, em 5% das vezes houve atraso maior que 4 ms. O código que gerou a Fig. 10 e as estatísticas apresentadas se encontra no apêndice D.

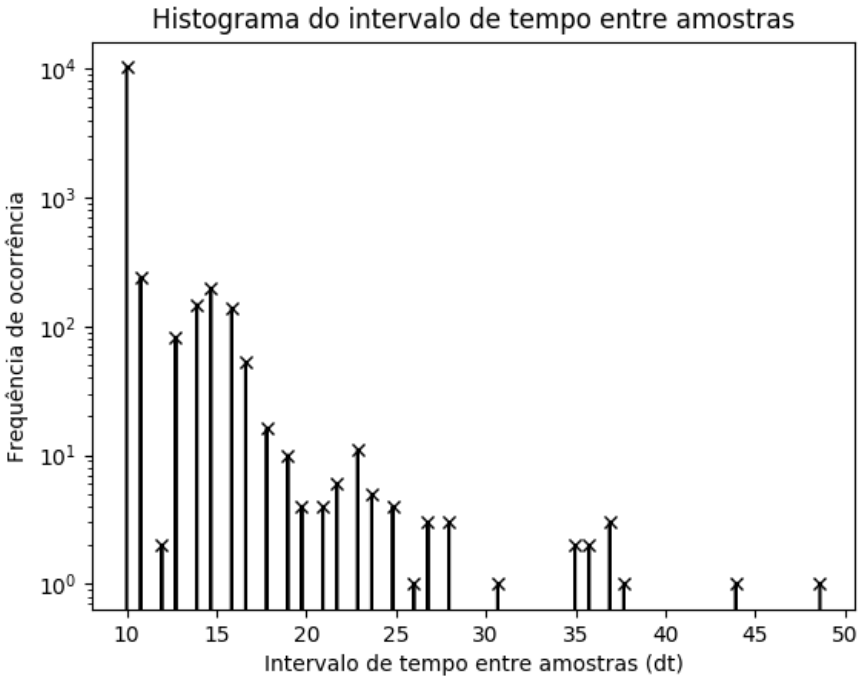

Figura 10: Histograma do intervalo de tempo entre as amostras coletadas, com o eixo vertical em escala logarítmica para melhor visualização.

4.2 SNI com Dados de Aceleração e Orientação

A partir do código apresentado no apêndice F que implementa as equações apresentadas na subseção 2.4 obteve-se as Figs. 11 e 12. A Fig. 11 mostra o trajeto estimado e a Fig. 12 mostra a evolução temporal da velocidade estimada em X, tendo como base a aceleração em X, para cada volta realizada pelo robô. As trajetórias são mostradas parcialmente, para permitir a visualização da pista.

4.3 *Dead Reckoning*

As Figs. 13, 14 e 15 foram obtidas através do código G.

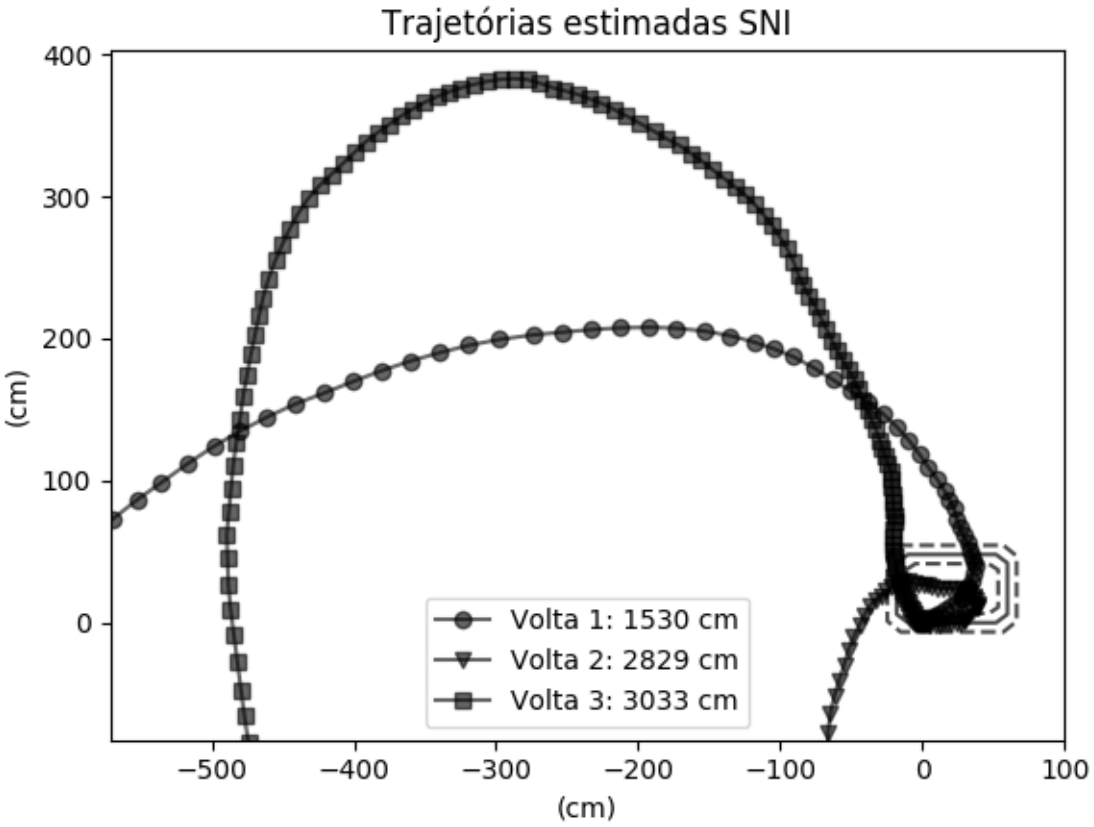

Figura 11: Trajetórias estimadas para cada volta realizada pelo robô,utilizando SNI. As trajetórias são mostradas parcialmente, para permitir a visualização da pista.

A trajetória com derrapagem representa o resultado da utilização dos dados coletados nas equações apresentadas na subseção 2.5.2. A trajetória sem derrapagem utiliza as equações da subseção 2.5.3. A Fig. 15 mostra a evolução das coordenadas do robô em função do tempo, juntamente com a velocidade em cada coordenada, para cada volta realizada pelo robô.

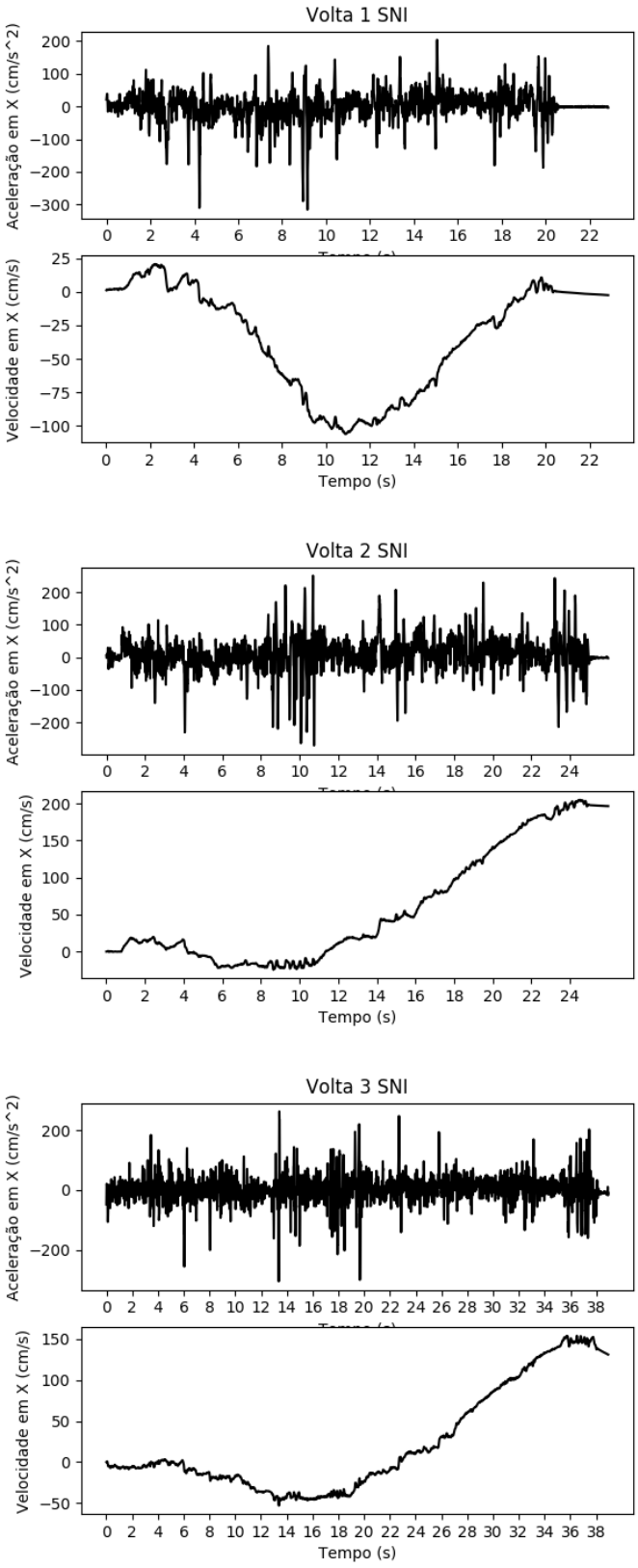

Figura 12: Evolução temporal da velocidade estimada em X, tendo como base a aceleração em X, para cada volta realizada pelo robô, utilizando a estratégia SNI.

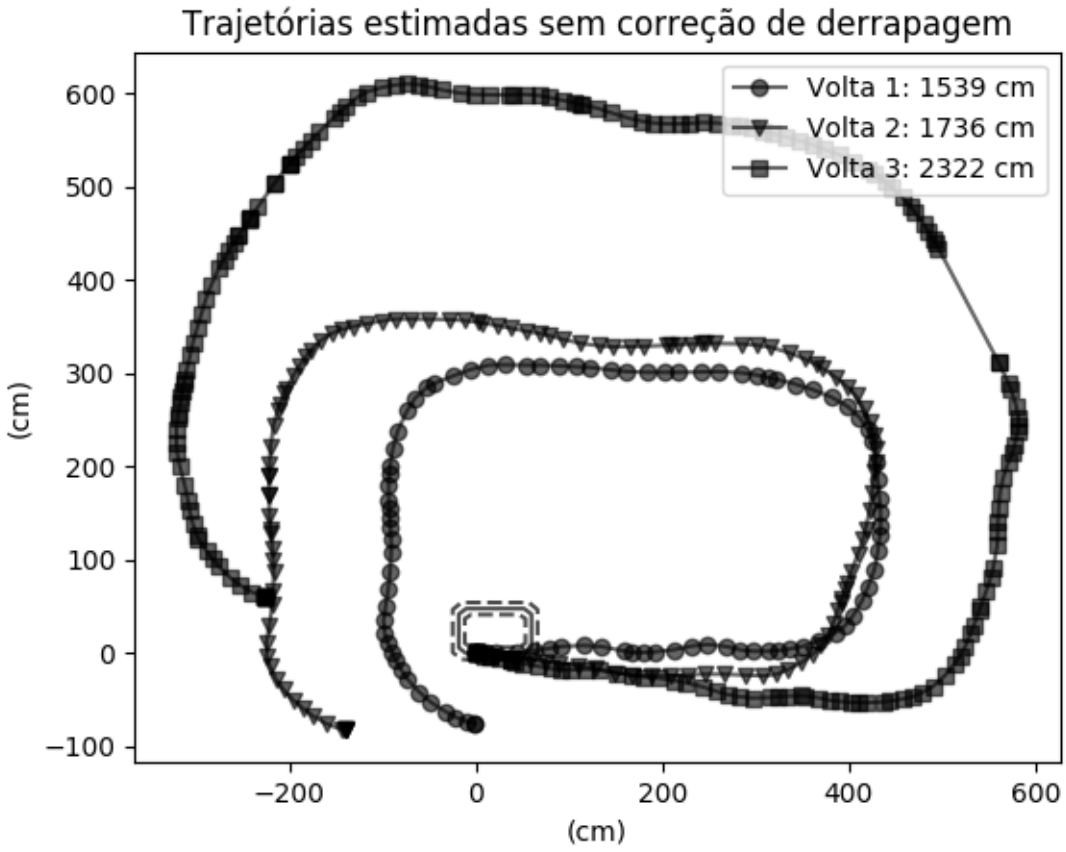

Figura 13: Trajetórias estimadas para cada volta realizada pelo robô, sem modelagem de derrapagem, utilizando *Dead Reckoning*.

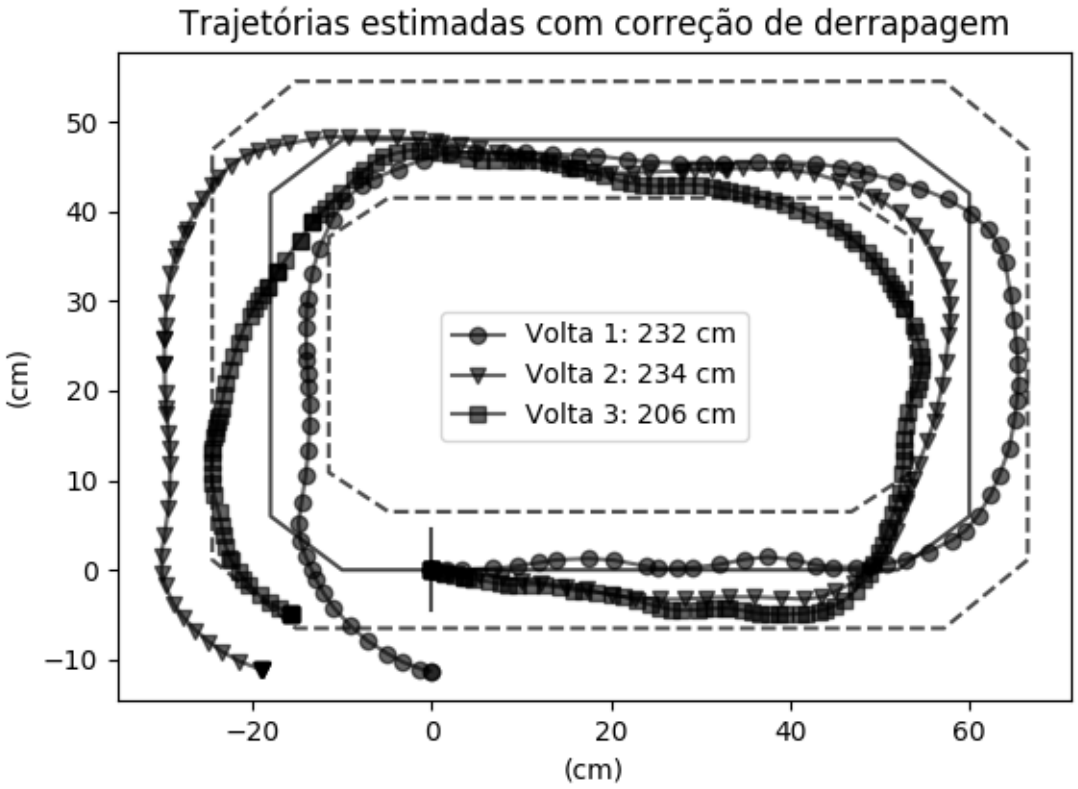

Figura 14: Trajetórias estimadas para cada volta realizada pelo robô, com modelagem de derrapagem, utilizando *Dead Reckoning*.

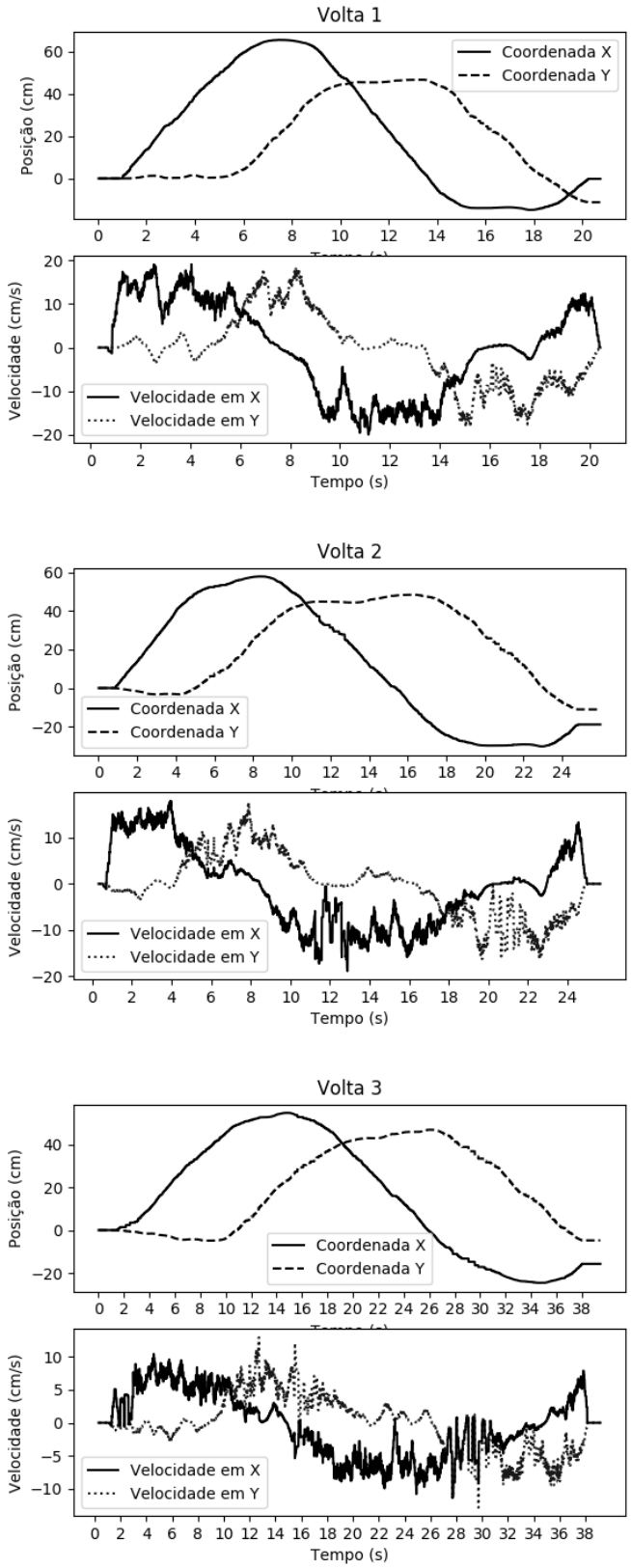

Figura 15: Evolução das coordenadas do robô em função do tempo, juntamente com a velocidade em cada coordenada, para cada volta realizada pelo robô.

5 DISCUSSÃO

Esse capítulo discute os resultados experimentais obtidos, a partir da análise computacional dos dados coletados em laboratório. O robô realizou três voltas na pista confeccionada para testes. A partir das equações apresentadas no capítulo 2, implementadas como descrito no capítulo 3, obteve-se os resultados apresentados no capítulo 4.

Primeiramente, pode-se observar que a escolha de quaterniões é fortuita, pois se o ângulo θ fosse utilizado diretamente para calcular $\sin(\theta)$, $\cos(\theta)$ e $\Delta\theta$ haveriam problemas com saltos em $\Delta\theta$ quando θ passa pela região dos 180°, oscilando entre -180° e 180°, como pode-se observar na Fig. 9.

Sobre as variações no período amostral, apresentados na seção 4.1. Essas variações ocorrem pois ocorrem interrupções no processamento do microcontrolador (MEGAAVR...,) de duas fontes. A primeira são as operações de escrita no cartão SD, que para ser efetuadas demandam processamento. A segunda são as interrupções geradas pelo codificador óptico, que realiza contagem de pulsos conforme as rodas giram. Para se contornar tais variações, um microcontrolador mais potente, com maior frequência de operação, deve ser utilizado.

Como pode-se observar no histograma da Fig. 10, o valor mais comum, e como dito em 4.1, a mediana dos valores, é o valor nominal para o período amostral, 10 ms. O segundo valor mais frequente é mais que 10 vezes menos comum. Caso as variações no período amostral prejudiquem demasiadamente as estimativas, o que não pode ser aferido devido à incompletude do sistema de navegação, deve-se utilizar um microcontrolador mais rápido, porém possivelmente de maior custo.

A trajetória estimada utilizando o método apresentado em 2.4, mostrada parcialmente na Fig. 11, diverge rapidamente devido à uma acumulação rápida dos erros sensoriais e ruídos presentes na leitura da aceleração linear fornecida pelo **BNO055**.

Como pode-se observar na Fig. 12, a aceleração é muito ruidosa, fazendo a velocidade se divergir rapidamente do valor real. Filtros digitais foram tentados para se filtrar o ruído, porém não foi obtido sucesso. Por essa razão, optou-se pelo *Dead Reckoning* na esperança de que os erros se acumulassem mais devagar.

Implementou-se o DR e de fato, os erros se acumulam mais devagar, havendo deformações muito grandes na trajetória mais ao final do trajeto, como pode-se observar na Fig. 14. Porém, teve-se que se modelar uma nova fonte de erros que vem com essa estratégia: a derrapagem.

Na Fig. 13 pode-se observar o efeito severo que a derrapagem teve no comprimento do trajeto estimado, chegando a quase 10 vezes o comprimento nominal de 236 cm na

terceira volta. A modelagem de derrapagem foi totalmente *ad hoc* e intuitiva, porém obteve-se resultados que aproximam o trajeto nominal, desviando inicialmente pouco dele, e dentro das faixas de tolerância de 6,5 cm para mais ou para menos tracejadas nas Figs. 13 e 14.

A Fig. 15 mostra que a trajetória estimada é bem suave nas coordenadas, porém houveram grandes variações na velocidade, que são compatíveis com o observado na prática e também com a forma com que o seguidor de linha foi programado, justamente na tentativa de manter a movimentação em passos pequenos e evitar derrapagem ou escapada da trilha.

A derrapagem pode ser corrigida melhorando os sinais de controle do robô, criando um sistema de movimentação robusto, que evita derrapagens. Principalmente quando o robô parte da inercia para o início do movimento, momento em que foi observada a maior derrapagem.

6 CONCLUSÃO

O trabalho realizado nesse TCC é o início de um processo de desenvolvimento de um sistema de localização espacial de um robô móvel autônomo (RMA). A contribuição desse projeto foi a escolha de uma estratégia para se realizar a estimativa de localização, descartando-se SNI em prol do *Dead Reckoning*.

Os sistemas de seguidor de linha e sensoriamento podem ser aperfeiçoados futuramente. Com um sistema mais robusto de controle de movimentação, evitando derrapagem, a fusão sensorial com um sensor de ultrassom pode providenciar dados suficientes para um sistema de navegação completo, como discutido na seção 2.3.

O ultrassom, por medir a distância entre o robô e os objetos ao seu redor, fornece uma medida direta de posicionamento do robô em seu ambiente. Essa medida pode ser refinada utilizando os dados de odometria e orientação espacial para se construir um Filtro de Kalman (LEVY,), que melhore a precisão das estimativas e provenha localização suficientemente precisa para as funções desejadas ao robô.

REFERÊNCIAS

Arduino Introduction. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 03 maio 2019.

BNO055. Disponível em: <https://www.bosch-sensortec.com/bst/products/all_products/bno055>. Acesso em: 06 maio 2019.

CANDIDO, G. **Robô seguidor de linha com sensor Infravermelho e PWM**. 2018. Disponível em: <https://portal.vidadesilicio.com.br/robo-seguidor-de-linha-sensor-infravermelho-e-pwm/>. Acesso em: 02 maio 2019.

GOEL P.; ROUMELIOTIS, S. I. S. G. S. **Robot Localization Using Relative and Absolute Position Estimates**. Departamento de Ciência da Computação, Instituto de Robótica e Sistemas Inteligentes, Universidade de Southern California, E.U.A., 1999. Disponível em: <https://www-users.cs.umn.edu/~stergios/papers/iros99_pioneer.pdf>. Acesso em: 02 maio 2019.

GROVES, D. P. **Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems**. [S.l.]: Artech House, 2008. 3-14 p. ISBN 978-1-58053-255-6.

HAYKIN S.; VEEN, V. B. **Signals and Systems**. 2. ed. [S.l.]: John Wiley & Sons, Inc., 2002. 362-374 p. ISBN 0471-37851-8.

HELLSTRöM, T. **Kinematics Equations for Differential Drive and Articulated Steering**. Departamento de Ciência da Computação, Universidade de Umeå, Suécia, 2011. Disponível em: <https://webapps.cs.umu.se/uminf/reports/2011/019/part1.pdf>. Acesso em: 01 maio 2019.

HUMPHRIES, M. **Amazon: Full Warehouse Automation Is 10 Years Away**. 2019. Disponível em: <https://www.pcmag.com/news/368117/amazon-full-warehouse-automation-is-10-years-away>. Acesso em: 04 maio 2019.

INCREMENTAL Optical Encoders. Disponível em: <http://thedenneys.org/pub/robot/encoders/>. Acesso em: 06 maio 2019.

LEVY, D. S. **The Extended Kalman Filter: An Interactive Tutorial for Non-Experts**. Disponível em: <https://home.wlu.edu/~levys/kalman_tutorial/>. Acesso em: 08 maio 2019.

LUCAS, G. **A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators**. 2000. Disponível em: <http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html>. Acesso em: 02 maio 2019.

MEGAAVR Data Sheet. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf>. Acesso em: 07 maio 2019.

STOVALL, S. H. **Basic Inertial Navigation**. Naval Air Warfare Center Weapons Division, United States of America Navy, 1997. Disponível em: <https://www.globalsecurity.org/space/library/report/1997/basicnav.pdf>. Acesso em: 09 agosto 2018.

Wikipedia contributors. **Conversion between quaternions and Euler angles — Wikipedia, The Free Encyclopedia**. 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Conversion_between_quaternions_and_Euler_angles&oldid=884156366>. Acesso em: 08 maio 2019.

_____. **Mobile robot — Wikipedia, The Free Encyclopedia**. 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Mobile_robot&oldid=893797597>. Acesso em: 04 maio 2019.

_____. **Nyquist–Shannon sampling theorem — Wikipedia, The Free Encyclopedia**. 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Nyquist%E2%80%93Shannon_sampling_theorem&oldid=890776007>.

_____. **Quaternion — Wikipedia, The Free Encyclopedia**. 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Quaternion&oldid=892599740>. Acesso em: 08 maio 2019.

_____. **Quaternions and spatial rotation — Wikipedia, The Free Encyclopedia**. 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Quaternions_and_spatial_rotation&oldid=895748599>. Acesso em: 08 maio 2019.

_____. **Sensor fusion — Wikipedia, The Free Encyclopedia**. 2019. Disponível em: <https://en.wikipedia.org/w/index.php?title=Sensor_fusion&oldid=891975550>.

WIKIPéDIA. **Ponte H — Wikipédia, a enciclopédia livre**. 2019. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Ponte_H&oldid=53991655>. Acesso em: 07 janeiro 2019.

WOODMAN, O. J. **An introduction to inertial navigation**. Laboratório de Computação, Universidade de Cambridge, Reino Unido, 2007. Disponível em: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf>. Acesso em: 02 maio 2019.

APÊNDICE A – CÓDIGO DE SENSORIAMENTO

```
/*  Codigo de sensoriamento (coleta de dados) para o robô
 *  utilizando o sensor inercial BNO055 presente no shield 9 Axis Motion
 *  Autor: Pedro V. B. Jeronymo
 *  03/2019
 */

/* Bibliotecas */
#include <SPI.h>
#include <SD.h>
#include "NAxisMotion.h"
#include <Wire.h>

/* Variaveis globais */
File myFile;
NAxisMotion mySensor; //sensor inercial BNO055 (9 Axis Motion Shield)
unsigned long lastStreamTime = 0;  //armazenar ultimo timestamp transmitido
const int streamPeriod = 10;       //taxa de amostragem

// Estrutura que contem os dados coletados
struct readings_t {
  // 4 bytes - tempo corrente
  unsigned long lastStreamTime;
  // 1 byte  - status de calibracao dos acelerometros
  uint8_t accelCalibStatus;
  // 1 byte  - status de calibracao dos magnetometros
  uint8_t magCalibStatus;
  // 1 byte  - status de calibracao dos giroscopios
  uint8_t gyroCalibStatus;
  // 1 byte  - status de calibracao do sistema
  uint8_t sysCalibStatus;

  // 4 bytes - Aceleracao Linear no eixo X
  float linAccX;
  // 4 bytes - Aceleracao Linear no eixo Y
  float linAccY;
  // 4 bytes - Aceleracao Linear no eixo Z
```

```
  float linAccZ;

 // 2 bytes   Quaternion de orientacao espacial
  int16_t quatW;
  // 2 bytes - q = (qW, qX, qY, qZ)
  int16_t quatX;
  int16_t quatY; // 2 bytes
  int16_t quatZ; // 2 bytes
  // 2 bytes - contagem de pulsos na roda direita
  int16_t rightWheel;
  // 2 bytes - contagem de pulsos na roda esquerda
  int16_t leftWheel;
};
readings_t readings;

// Definicao dos pinos do led e o botao
const int LED_PIN = 8;
const int SWITCH_PIN = 7;

// Estado do botao (0 = aberto, 1 = em curto)
int state = 0;

// Contatores de pulsos das rodas
unsigned int counterD = 0;
unsigned int counterE = 0;

/* Funcoes auxiliares */
void docountD()  // funcao auxiliar para contagem de pulsos da roda direita
{
  counterD++;
}

void docountE()  // funcao auxiliar para contagem de pulsos da roda esquerda
{
  counterE++;
}

void blink_led() { //pisca o led para debug e indicar funcionamento
  int i;
```

```
  for (i = 0; i < 5; i++) {
    digitalWrite(LED_PIN, HIGH);
    delay(100);
    digitalWrite(LED_PIN, LOW);
    delay(100);
  }
}

/* Funcao de inicializacao do Arduino */
void setup() {
  //Configuracao dos pinos
  pinMode(LED_PIN, OUTPUT);
  pinMode(SWITCH_PIN, INPUT);

  // Led desligado por padrao
  digitalWrite(LED_PIN, LOW);

  // Abre comunicação serial
  Serial.begin(115200);

  // Inicializacao do cartao SD
  Serial.print("Initializing SD card...");
  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  //Abertura do arquivo onde sao armazenados os dados sensoriados
  myFile = SD.open("READINGS.DAT", FILE_WRITE);
  if (!myFile) {
    Serial.println("Error opening file!");
    digitalWrite(LED_PIN, HIGH);
    while(1);
  }

  // Aguarda usuario apertar botao para continuar
  while(state == LOW) {
    state = digitalRead(SWITCH_PIN);
```

```
  }
  blink_led();
  // Acende o led para indicar que a leitura comecou
  digitalWrite(LED_PIN, HIGH);
  // Inicializacao e configuracao do sensor inercial
  I2C.begin();
  mySensor.initSensor();
  mySensor.setOperationMode(OPERATION_MODE_NDOF);
  mySensor.setUpdateMode(MANUAL);

  //Anexa interrupcoes as funcoes de contagem de pulsos das rodas
  attachInterrupt(0, docountE, RISING);
  attachInterrupt(1, docountD, RISING);
}

/* Loop do Arduino*/
void loop() {
  //Quando o botao for apertado novamente, finaliza a leitura
  state = digitalRead(SWITCH_PIN);
  if(state == HIGH) {
    blink_led();
    digitalWrite(LED_PIN, LOW);
    myFile.close();
    while(1); //espera ocupada ate ser desligado/reiniciado
  }

  // Leitura de dados caso tenha se passado o tempo de amostragem
  if ((millis() - lastStreamTime) >= streamPeriod)
  {
    // leitura do tempo corrente
    lastStreamTime = millis();
    readings.lastStreamTime = lastStreamTime;

    //Leituras do sensor inercial
    mySensor.updateQuat();
    mySensor.updateLinearAccel();
    mySensor.updateCalibStatus();

    readings.accelCalibStatus = mySensor.readAccelCalibStatus();
```

```
    readings.magCalibStatus = mySensor.readMagCalibStatus();
    readings.gyroCalibStatus = mySensor.readGyroCalibStatus();
    readings.sysCalibStatus = mySensor.readSystemCalibStatus();

    readings.linAccX = mySensor.readLinearAccelX();
    readings.linAccY = mySensor.readLinearAccelY();
    readings.linAccZ = mySensor.readLinearAccelZ();

    readings.quatW = mySensor.readQuatW();
    readings.quatX = mySensor.readQuatX();
    readings.quatY = mySensor.readQuatY();
    readings.quatZ = mySensor.readQuatZ();

    //Leitura dos pulsos de cada roda
    detachInterrupt(0);
    detachInterrupt(1);
    readings.rightWheel = counterD;
    readings.leftWheel = counterE;
    counterD = 0;
    counterE = 0;
    attachInterrupt(0, docountE, RISING);
    attachInterrupt(1, docountD, RISING);

    //Escrita no SD da estrutura contendo as leituras
    myFile.write((const char*)&readings, sizeof(readings_t));
  }
}
```

APÊNDICE B – CÓDIGO SEGUIDOR DE LINHA

```
/*  Codigo de robo seguidor de linha com Arduino (chassi carrinho de Arduino)
 *  Autor: Pedro V. B. Jeronymo
 *  03/2019
 */

//Pinos de controle do sentido de rotacao do motor direito
#define MotorD_sentido1 16
#define MotorD_sentido2 17
//Pinos de controle do sentido de rotacao do motor esquerdo
#define MotorE_sentido1 18
#define MotorE_sentido2 19
//Pinos de controle de ativacao dos motores
#define MotorD_atividade 9
#define MotorE_atividade 10
//Pinos dos sensores infravermelhos
#define Sensor_direita 14
#define Sensor_esquerda 15

//Estado dos sensores infravermelhos
bool direita, esquerda;

/* Funcao de inicializacao do Arduino */
void setup() {
  //Inicializa comunicação serial para debug
  Serial.begin(115200);
  //Inicializacao dos pinos
  pinMode(MotorD_sentido1, OUTPUT);
  pinMode(MotorD_sentido2, OUTPUT);
  pinMode(MotorE_sentido1, OUTPUT);
  pinMode(MotorE_sentido2, OUTPUT);
  pinMode(MotorD_atividade, OUTPUT);
  pinMode(MotorE_atividade, OUTPUT);
  pinMode(Sensor_direita, INPUT);
  pinMode(Sensor_esquerda, INPUT);
}
```

```
/* Loop do Arduino*/
void loop() {
  //Define o sentido de ambos os motores de forma que permita
  //o carrinho andar para frete
  digitalWrite(MotorD_sentido1, LOW);
  digitalWrite(MotorD_sentido2, HIGH);
  digitalWrite(MotorE_sentido1, HIGH);
  digitalWrite(MotorE_sentido2, LOW);

  //Leituras dos Sensores infravermelhos
  direita = digitalRead(Sensor_direita);
  esquerda = digitalRead(Sensor_esquerda);
  //Saida serial para debug
  Serial.print(direita);
  Serial.print(" || ");
  Serial.println(esquerda);

  /*Rodando os motores dependendo das leituras */

  //Para frente
  if(direita == false && esquerda == false){
    digitalWrite(MotorD_atividade, HIGH);
    digitalWrite(MotorE_atividade, HIGH);
  //Para a direita
  } else if(direita == false && esquerda == true){
    digitalWrite(MotorD_atividade, HIGH);
    digitalWrite(MotorE_atividade, LOW);
  //Para a esquerda
  }else if(direita == true && esquerda == false){
    digitalWrite(MotorD_atividade, LOW);
    digitalWrite(MotorE_atividade, HIGH);
  //Parar
  }else if(direita == true && esquerda == true){
    digitalWrite(MotorD_atividade, LOW);
    digitalWrite(MotorE_atividade, LOW);
 }
  //Mantem ativados por 5 ms e entao entra em parada em rotacao livre por 5 ms
  delay(5);
  // Alterar ambos para HIGH para parada forcada
```

```
  digitalWrite(MotorD_atividade, LOW);
  digitalWrite(MotorE_atividade, LOW);
  //Executa a parada forcada quando ativada
  digitalWrite(MotorD_sentido1, LOW);
  digitalWrite(MotorD_sentido2, LOW);
  digitalWrite(MotorE_sentido1, LOW);
  digitalWrite(MotorE_sentido2, LOW);
  delay(5);
}
```

APÊNDICE C – CÓDIGO DE CONVERSÃO DO ARQUIVO DE DADOS BINÁRIO PARA FORMATO CSV

```
from struct import unpack

msgsize = 32 # 32+4

fout = open('readings.csv','w')

with open("READINGS.DAT", 'rb') as f:
    s = f.read(msgsize)
    try:
        while s != b'':
            tup = unpack('IBBBBfffhhhhhh', s)
            line = ",".join([str(x) for x in tup])+'\n'
            fout.write(line)
            s = f.read(msgsize)
    except:
        pass

fout.close()
```

APÊNDICE D – CÓDIGO DE ANÁLISE DO INTERVALO ENTRE AMOSTRAS

```
##########################################
# Análise do intervalo intra-amostral
# Autor: Pedro V. B. Jeronymo
# Data: 2019-05-06
##########################################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Carrega os dados em um Pandas DataFrame
df = pd.read_csv('../data/readings_2019-03-26__11-45.csv', header=None, names=[
      "lastStreamTime",
      "accelCalibStatus",
      "magCalibStatus",
      "gyroCalibStatus",
      "sysCalibStatus",

      "linAccX",
      "linAccY",
      "linAccZ",

      "quatW",
      "quatX",
      "quatY",
      "quatZ",

      "rightWheel",
      "leftWheel"
      ])

# Calcula histograma do intervalo de tempo entre amostras
dt = df.lastStreamTime.diff().values[1:]
hist, bins = np.histogram(dt, bins=100)

# Estatísticas
print('Mediana:', np.median(dt))
```

```
print('95%:', np.percentile(dt, 95))

# Gráfico com o histograma
plt.semilogy(bins[:-1], hist, 'k', marker='x')
plt.xlabel('Intervalo de tempo entre amostras (dt)')
plt.ylabel('Frequência de ocorrência')
plt.title('Histograma do intervalo de tempo entre amostras')
plt.show()
```

APÊNDICE E – CÓDIGO DE UTILIDADES PARA SNI E DR

```
################################################################
# Utilidades
# Autor: Pedro V. B. Jeronymo
# Data: 2019-05-06
################################################################

import numpy as np
import matplotlib.pyplot as plt

def quatToYaw(qw, qz):
    """
    Convert orientation quaternion into Yaw (euler angle) given as input
    qw and qz, where q = (qw, qx, qy, qz) and qx and qy are discarded in this
    estimation.
    """
    return np.arctan2(2*qw*qz, 1-2*qz*qz)

class Track():
    def __init__(self):
        track_x = [0, 52, 60, 60, 52, -10, -18, -18, -10, 0]
        track_y = [0, 0,  6,  42, 48,  48, 42,  6,    0,  0]
        self.track_ref = np.array([*zip(track_x, track_y)])
        scalex_o = (78+6.5*2)/78
        scaley_o = (48+6.5*2)/48
        scale_o = np.array([[scalex_o, 0], [0, scaley_o]])
        track_outer = self.track_ref.dot(scale_o)
        self.track_outer = self.translate(track_outer, self.track_ref)
        scalex_i = (78-6.5*2)/78
        scaley_i = (48-6.5*2)/48
        scale_i = np.array([[scalex_i, 0], [0, scaley_i]])
        track_inner = self.track_ref.dot(scale_i)
        self.track_inner = self.translate(track_inner, self.track_ref)
        self.start_line_x = [0, 0]
        self.start_line_y = [-4.5, 4.5]

    def get_center(self, track):
```

```
        center = []
        for k in range(2):
            liminf = track[:,k].min()
            limsup = track[:,k].max()
            center.append(liminf+(limsup-liminf)/2)
        return np.array(center)

    def translate(self, track, track_ref):
        center = self.get_center(track)
        center_ref = self.get_center(track_ref)
        dc = center-center_ref
        return track-dc

    def get_track_len(self, track):
        x = track[:,0]
        y = track[:,1]
        d = 0
        for k in range(1, x.shape[0]):
            dx = x[k]-x[k-1]
            dy = y[k]-y[k-1]
            d += np.sqrt(dx*dx+dy*dy)
        return d

    def plot_paths(self, paths, title):
        fig = plt.figure()
        ax = fig.add_subplot(111)

        ax.plot(self.track_ref[:,0], self.track_ref[:,1], 'k', alpha=0.7)
        ax.plot(self.start_line_x, self.start_line_y, 'k', alpha=0.7)
        ax.plot(self.track_outer[:,0], self.track_outer[:,1], 'k--', alpha=0.7)
        ax.plot(self.track_inner[:,0], self.track_inner[:,1], 'k--', alpha=0.7)

        marker = ['o', 'v', 's']

        for path in paths:
            x, y, t, d, i = path
            ax.plot(x[::20], y[::20], label='Volta {}: {:.0f} cm'.format(i+1, d), \
                alpha=0.6, marker=marker[i], c='k')
```

```
ax.set_title(title)
ax.xaxis.set_label_text('(cm)')
ax.yaxis.set_label_text('(cm)')
ax.legend()
ax.set_aspect('equal')
plt.show()
```

APÊNDICE F – CÓDIGO SNI

```
################################################################
# Estimativas de trajetória a partir de Navegação Inercial
# Autor: Pedro V. B. Jeronymo
# Data: 2019-05-06
################################################################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as signal
from util import Track

# Carrega os dados em um Pandas DataFrame
df = pd.read_csv('../data/readings_2019-03-26__11-45.csv', header=None, names=[
      "lastStreamTime",
      "accelCalibStatus",
      "magCalibStatus",
      "gyroCalibStatus",
      "sysCalibStatus",

      "linAccX",
      "linAccY",
      "linAccZ",

      "quatW",
      "quatX",
      "quatY",
      "quatZ",

      "rightWheel",
      "leftWheel"
      ])

# Normaliza quaternions
df.quatW /= (1<<14)
df.quatX /= (1<<14)
df.quatY /= (1<<14)
```

```
df.quatZ /= (1<<14)

# Separa dados em voltas
loop = []
loop.append(df.iloc[:2200])
loop.append(df.iloc[3000:5500])
loop.append(df.iloc[7000:10750])

# Para cada volta, computa velocidade e posição
paths = []
other = []
for i in range(3):
    # Tempo corrente
    t = loop[i].lastStreamTime.values*1e-3
    t = t-t[0]
    # Intervalo intra-amostral
    dt = np.diff(t)

    # Transformação entre o referencial (x'-y') e o referencial (x-y)
    qw = loop[i].quatW.values
    qz = loop[i].quatZ.values

    sin_theta = 2*qw*qz
    cos_theta = qw*qw-qz*qz

    Rot = np.array([[cos_theta, -sin_theta], [sin_theta, cos_theta]])
    A_ = loop[i][['linAccX', 'linAccY']].values

    a = []
    for k in range(A_.shape[0]):
        a.append(Rot[:,:,k].dot(A_[k,:]))
    a = np.array(a)*1e2 # converte p/ cm/s^2

    # Determinação da velocidade
    vx = np.cumsum(a[1:,0]*dt)
    vy = np.cumsum(a[1:,1]*dt)

    # Determinação da posição
    x = np.cumsum(vx*dt)
```

```
    y = np.cumsum(vy*dt)

    # Comprimento da trajetória
    d = 0
    for k in range(1, x.shape[0]):
        dx = x[k]-x[k-1]
        dy = y[k]-y[k-1]
        d += np.sqrt(dx*dx+dy*dy)

    paths.append((x, y, t, d, i))
    other.append((a, vx, vy, t, i))

# Gráfico Trajetórias
track = Track()
track.plot_paths(paths, "Trajetórias estimadas SNI")

# Gráficos de posição e velocidade
for tup in other:
    a, vx, vy, t, i = tup
    fig = plt.figure()
    ax = fig.add_subplot(211)
    ax.set_title('Volta {} SNI'.format(i+1))
    ax.plot(t, a[:,0], 'k')
    ax.xaxis.set_ticks(np.arange(0, t.max(), 2))
    ax.xaxis.set_label_text('Tempo (s)')
    ax.yaxis.set_label_text('Aceleração em X (cm/s^2)')
    ax = fig.add_subplot(212)
    ax.plot(t[1:], vx, 'k')
    ax.xaxis.set_ticks(np.arange(0, t.max(), 2))
    ax.xaxis.set_label_text('Tempo (s)')
    ax.yaxis.set_label_text('Velocidade em X (cm/s)')
    plt.show()
```

APÊNDICE G – CÓDIGO *DEAD RECKONING*

```
############################################################
# Estimativas de trajetória a partir de Dead Reckoning
# Autor: Pedro V. B. Jeronymo
# Data: 2019-05-06
############################################################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from util import Track, quatToYaw

# Parâmetros físicos do Robô
step = 6.5*np.pi/20 # cm/hole - tamanho do passo
l = 15 # Tamanho do eixo (separação entre rodas)

# Pistas e comprimentos
track = Track()

track_ref_len = track.get_track_len(track.track_ref)
track_outer_len = track.get_track_len(track.track_outer)
track_inner_len = track.get_track_len(track.track_inner)

print('Comprimento trajetória mínima:  {:.0f} cm'.format(track_inner_len))
print('Comprimento nominal da pista:   {:.0f} cm'.format(track_ref_len))
print('Comprimento trajetória máxima:  {:.0f} cm'.format(track_outer_len))
print('Num. passos nomimal (esq/dir): {:.0f}/{:.0f}'.format(196/step, 236/step))

# Carrega dados em um Pandas DataFrame
df = pd.read_csv('../data/readings_2019-03-26__11-45.csv', header=None, names=[
      "lastStreamTime",
      "accelCalibStatus",
      "magCalibStatus",
      "gyroCalibStatus",
      "sysCalibStatus",

      "linAccX",
      "linAccY",
```

```
      "linAccZ",

      "quatW",
      "quatX",
      "quatY",
      "quatZ",

      "rightWheel",
      "leftWheel"
      ])

# Normaliza quaternions
df.quatW /= (1<<14)
df.quatX /= (1<<14)
df.quatY /= (1<<14)
df.quatZ /= (1<<14)

# Separa dados em voltas realizada pelo robô
loop = []
loop.append(df.iloc[:2000])      # 1st run
loop.append(df.iloc[3000:5500])  # 2nd run
loop.append(df.iloc[7000:10800]) # 3rd run

# Gráfico da orientação em relação ao tempo
for i in range(3):
    yaw = quatToYaw(loop[i].quatW.values, loop[i].quatZ.values)
    t = loop[i].lastStreamTime.values
    t = (t-t[0])*1e-3
    plt.plot(t, np.rad2deg(yaw), 'k')
    plt.title('Orientação em função do tempo (Volta {})'.format(i+1))
    plt.xlabel('Tempo (s)')
    plt.ylabel('Orientação em graus')
    plt.yticks(np.arange(-180, 181, 30))
    plt.show()

# Estimativa de trajetória com correção de derrapagem
paths = []
max_step = 4
for i in range(3):
```

```
print('-'*10, 'Volta {}'.format(i+1), '-'*10)
## Set initial time to 0
t = loop[i].lastStreamTime.values
t = (t-t[0])*1e-3
t = t[1:]



nr = loop[i].rightWheel.values[1:]
nl = loop[i].leftWheel.values[1:]
r_total = nr.sum()
l_total = nl.sum()
print('Num. passos (esq/dir): {}/{}'.format(l_total, r_total))

#slipping_correction = 236/(step*r_total)
#slipping_correction = 196/(step*l_total)
slipping_correction = 0.5*(236/(step*r_total)+196/(step*l_total))

qw = loop[i].quatW.values
qz = loop[i].quatZ.values

sin_theta = 2*qw*qz
cos_theta = qw*qw-qz*qz
dtheta = np.diff(sin_theta)/cos_theta[:-1]
sin_theta = sin_theta[1:]
cos_theta = cos_theta[1:]

x0, y0 = (0, 0)
theta0 = 0
x = []
y = []

x.append(x0)
y.append(y0)

for k in range(t.shape[0]):
    R = 0
    ICCx = 0
    ICCy = 0
    if (nr[k] or nl[k]):
```

```
            if (nr[k] != nl[k]) and dtheta[k]:
                R = slipping_correction*0.5*(nr[k]+nl[k])*step/dtheta[k]
                ICCx = x[-1]-R*sin_theta[k]
                ICCy = y[-1]+R*cos_theta[k]

                x_ = (x[-1]-ICCx)*1-(y[-1]-ICCy)*dtheta[k]+ICCx
                if np.abs(x_-x[-1]) > max_step:
                    x_ = x[-1]
                x.append(x_)
                y_ = (x[-1]-ICCx)*dtheta[k]+(y[-1]-ICCy)*1+ICCy
                if np.abs(y_-y[-1]) > max_step:
                    y_ = y[-1]
                y.append(y_)
            else:
                ICCx = x[-1]
                ICCy = y[-1]
                x_ = (x[-1]-ICCx)*1-(y[-1]-ICCy)*dtheta[k]+ICCx
                x.append(x_)
                y_ = (x[-1]-ICCx)*dtheta[k]+(y[-1]-ICCy)*1+ICCy
                y.append(y_)
        else:
            x.append(x[-1])
            y.append(y[-1])

    x = np.array(x)
    y = np.array(y)

    d = 0
    for k in range(1, x.shape[0]):
        dx = x[k]-x[k-1]
        dy = y[k]-y[k-1]
        d += np.sqrt(dx*dx+dy*dy)
    print('Comprimento da trajetória: {:.0f} cm'.format(d))
    paths.append((x, y, t, d, i))

# Gráfico trajetória com correção de derrapagem
track.plot_paths(paths, 'Trajetórias estimadas com correção de derrapagem')

# Gráficos de posição e velocidade
```

```
for path in paths:
    x, y, t, d, i = path
    x = x[1:]
    y = y[1:]
    vx = []
    vy = []
    tv = []
    W = 15
    for k in range(2*W, x.shape[0]-2*W):
        vx.append((-x[k+2*W]+8*x[k+W]-8*x[k-W]+x[k-W])/(3*(t[k+2*W]-t[k-2*W])))
        vy.append((-y[k+2*W]+8*y[k+W]-8*y[k-W]+y[k-W])/(3*(t[k+2*W]-t[k-2*W])))
        tv.append(t[k])

    fig = plt.figure()
    ax = fig.add_subplot(211)
    ax.set_title('Volta {}'.format(i+1))
    ax.plot(t, x, 'k', label='Coordenada X')
    ax.plot(t, y, 'k--', label='Coordenada Y')
    ax.xaxis.set_ticks(np.arange(0, t.max(), 2))
    ax.legend()
    ax.xaxis.set_label_text('Tempo (s)')
    ax.yaxis.set_label_text('Posição (cm)')
    ax = fig.add_subplot(212)
    #ax.set_title('Velocidade vs tempo (Volta {})'.format(i+1))
    ax.plot(tv, vx, 'k', label='Velocidade em X', alpha=1.0)
    ax.plot(tv, vy, 'k:', label='Velocidade em Y', alpha=0.9)
    ax.xaxis.set_ticks(np.arange(0, t.max(), 2))
    ax.xaxis.set_label_text('Tempo (s)')
    ax.yaxis.set_label_text('Velocidade (cm/s)')
    ax.legend()
    plt.show()

# Estimativa de trajetória sem correção de derrapagem
paths = []
for i in range(3):
    print('-'*10, 'Volta {}'.format(i+1), '-'*10)
    ## Set initial time to 0
    t = loop[i].lastStreamTime.values
    t = (t-t[0])*1e-3
```

```
t = t[1:]

nr = loop[i].rightWheel.values[1:]
nl = loop[i].leftWheel.values[1:]
r_total = nr.sum()
l_total = nl.sum()
print('Num. passos (esq/dir): {}/{}'.format(l_total, r_total))

#slipping_correction = 236/(step*r_total)
#slipping_correction = 196/(step*l_total)
#slipping_correction = 0.5*(236/(step*r_total)+196/(step*l_total))
slipping_correction = 1.0

qw = loop[i].quatW.values
qz = loop[i].quatZ.values

sin_theta = 2*qw*qz
cos_theta = qw*qw-qz*qz
dtheta = np.diff(sin_theta)/cos_theta[:-1]
sin_theta = sin_theta[1:]
cos_theta = cos_theta[1:]

x0, y0 = (0, 0)
theta0 = 0
x = []
y = []

x.append(x0)
y.append(y0)

for k in range(t.shape[0]):
    R = 0
    ICCx = 0
    ICCy = 0
    if (nr[k] or nl[k]):
        if (nr[k] != nl[k]) and dtheta[k]:
            R = slipping_correction*0.5*(nr[k]+nl[k])*step/dtheta[k]
            ICCx = x[-1]-R*sin_theta[k]
            ICCy = y[-1]+R*cos_theta[k]
```

```
                x_ = (x[-1]-ICCx)*1-(y[-1]-ICCy)*dtheta[k]+ICCx

                x.append(x_)
                y_ = (x[-1]-ICCx)*dtheta[k]+(y[-1]-ICCy)*1+ICCy

                y.append(y_)
            else:
                ICCx = x[-1]
                ICCy = y[-1]
                x_ = (x[-1]-ICCx)*1-(y[-1]-ICCy)*dtheta[k]+ICCx
                x.append(x_)
                y_ = (x[-1]-ICCx)*dtheta[k]+(y[-1]-ICCy)*1+ICCy
                y.append(y_)
        else:
            x.append(x[-1])
            y.append(y[-1])

    x = np.array(x)
    y = np.array(y)

    d = 0
    for k in range(1, x.shape[0]):
        dx = x[k]-x[k-1]
        dy = y[k]-y[k-1]
        d += np.sqrt(dx*dx+dy*dy)
    print('Comprimento da trajetória: {:.0f} cm'.format(d))
    paths.append((x, y, t, d, i))

# Gráfico trajetória sem correção de derrapagem
track.plot_paths(paths, 'Trajetórias estimadas sem correção de derrapagem')
```