CONTROLE DE ROBÔS COM TRAÇÃO DIFERENCIAL USANDO ODOMETRIA

Gabriel Guimarães Almeida de Castro *
Geovany Araújo Borges **

* *Departamento de Engenharia Elétrica, Universidade de Brasília, DF, (e-mail: gabriel1997.castro@gmail.com).*
** *Departamento de Engenharia Elétrica, Universidade de Brasília, DF, (e-mail: gaborges@unb.br)*

Abstract: This paper presents an odometry project of a mobile robot which uses two encoders and compute the angular velocity of the robot front wheels. The communication uses the Robotic System Operating. The graphs present a good result and in practice the results are still better because the robot's dinamics does not allow abrupt oscilations and softens the velocity curve of the controllers.

Resumo: Este artigo mostra um projeto de odometria de um robô móvel que utiliza dois *encoders* e calcula-se a velocidade angular das duas rodas dianteiras do robô. A comunicação usa o *Robotic Operating System*(ROS). Gráficos mostram um bom resultado e na prática os resultados são ainda melhores já que a dinâmica do robô não permite oscilações bruscas e suaviza a curva de velocidade dos controladores.

Keywords: Odometry; robotics; velocity control;
Palavras-chaves: Odometria; robótica; controle de velocidade;

1. INTRODUÇÃO

Com o avanço da tecnologia nas últimas décadas, computadores se tornam cada vez menores e baratos assim como qualquer componente eletrônico. Assim, o uso de robôs também cresce rapidamente, principalmente na indústria, já que eles podem automatizar processos de grande risco para humanos assim como realizar tarefas repetitivas com grande eficiência.

Entretanto, um dos desafios mais comuns está em construir robôs com um preço acessível além de obter um sistema de localização precisa o suficiente para que possam cumprir suas tarefas. Existem diversas maneiras, combinações de sensores e técnicas para isso. Uma das configurações mais utilizadas é a odometria em conjunto com outros sensores de baixo custo, pois atende bem requisitos econômicos.

A odometria utiliza-se de *encoders* que convertem revoluções das rodas do robô em posição, assim obtém-se uma boa estimativa em curtas distâncias, porém como a posição é obtida da velocidade à partir de uma integração incremental, o erro acumulado em longas distâncias não pode ser desprezado.(Vargas, 2012)

Este texto apresenta a implementação da odometria de um robô com tração diferencial mostrando a arquitetura do robô, o cálculo de velocidade angular, o controlador de velocidade e as transformações necessárias para converter velocidade em posição

2. METODOLOGIA E RESULTADOS

2.1 Arquitetura do robô utilizado

O robô utilizado para implementar a odometria neste problema pertence ao Laboratória de Automação e Robótica(LARA) da Universidade de Brasília e é um dos robôs do projeto Kyle & Jesse [1]. A Figura 1 apresenta um esquema com o modelo de arquitetura dele.

A alimentação do robô é uma bateria de 12V que é ligada a dois reguladores para alimentar a controladora de servos(SSC-32), ao computador usado que é uma *CPU* industrial e a *Sabertooth* que é a controladora dos motores do *rover*.

Como pode ser visto no esquemático da Figura 1, há apenas dois *encoders* e eles estão conectado aos motores dianteiros. Há também um manipulador, mas não será usado neste trabalho.

Os *drivers* do robô são implementados de maneira que a *CPU* envia o comando *serial* especificando um *PWM* para a *SSC-32*, então a controladora de servos envia o *PWM* para a *Sabertooth* que envia para os motores.

O *rover* usado é feito de alumínio anodizado. É o modelo *A4WD1* da *lynxmotion*. (Site da Lynxmotion)

A Figura 2 mostra o robô usado.

[1] Repositório K&J: `https://github.com/lara-unb/Kyle_XY_Jesse_XX`

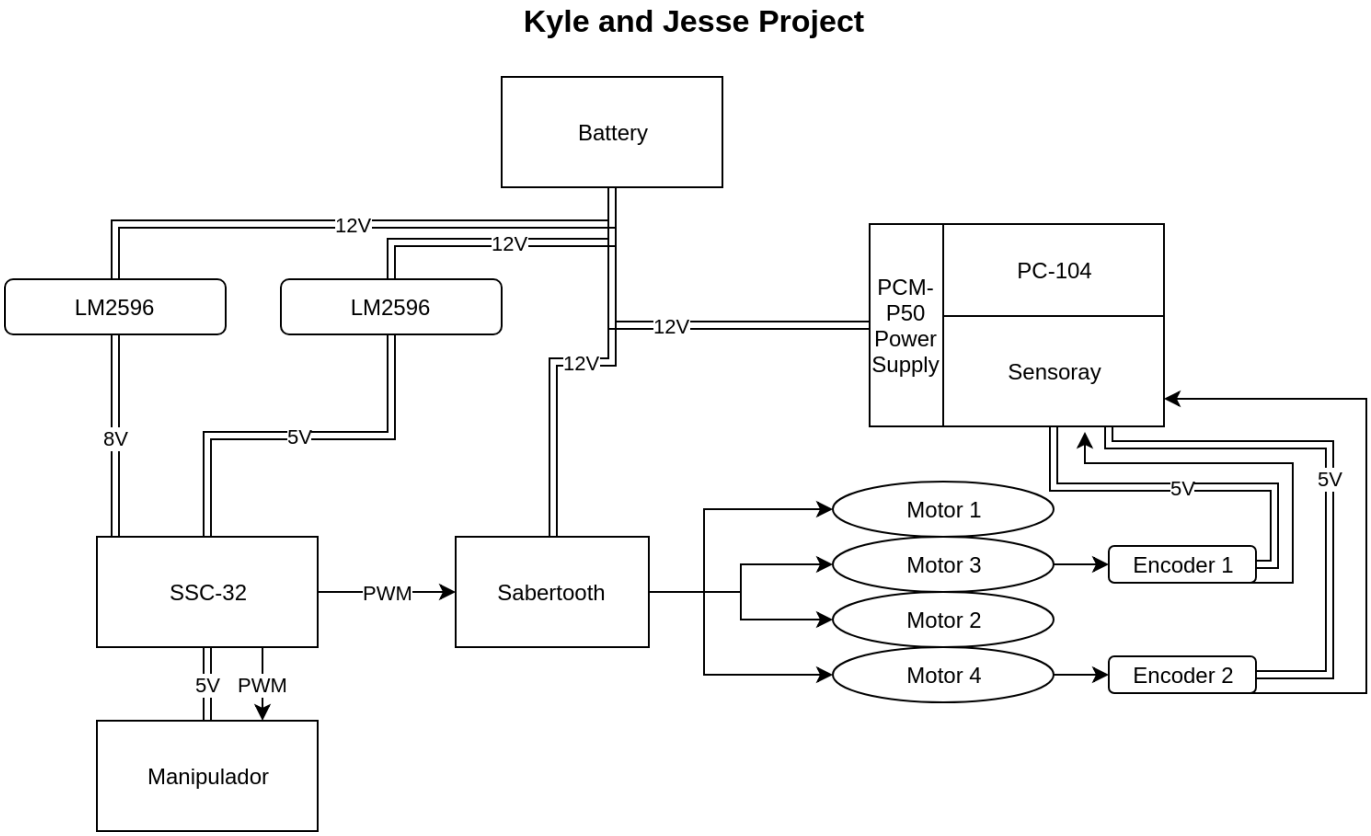

Figura 1. Arquitetura do Robô

Figura 2. Robôs Kyle & Jesse.

2.2 Comunicação

Para executar qualquer experimento o robô é controlado por um outro computador conectado à mesma rede *wireless* por *SSH*. Toda a comunicação é feita através *Robotic System Operating*(*ROS*)

A versão usada neste trabalho e compatível com o computador principal é o *ROS hydro* que funciona com o sistema operacional *Xubuntu 12*. Para gravar os dados as principais variáveis são publicados em tópicos do *ROS*, assim é possível usar o *rosbag* que é um *data logger* interno da coleção de *frameworks*. Os *bags* podem ser facilmente convertidos em arquivos do tipo *.csv* e estes podem ser lidos e processados em *MATLAB*.

2.3 Mapa PWM vs velocidade

Como a controladora de servos recebe comandos *PWM* com valores que variam de 500 a 2500 é necessário fazer um mapeamento destes comandos para velocidade. Ao fazer testes a velocidade máxima alcançada pelo *rover* com as rodas suspendidas é de aproximadamente $20 rad/s$. Já a *Sabertooth* recebe comandos na faixa de 1000 a 2000, onde o ponto em que o motor permanece parado é aproximadamente 1500. Como a variação é praticamente linear foi estabelecida uma equação de 1º grau e funções de conversão.

2.4 Estimativa de velocidade

Para estimar a velocidade é usado o método das diferenças finitas. Então conta-se os n pulsos em um pequeno intervalo de tempo T e usa-se a equação 1. (Carvalho et al., 2010)

$$\omega = \frac{2\pi n}{TP} \tag{1}$$

A constante P na equação é a resolução do *encoder*. É necessário uma pequena alteração na equação 1 porque os *encoders* estão conectados à redução do motor, então basta

multiplicá-la pelo termo $\frac{1}{n_r}$ onde $n_r = 30$ é a redução do motor.

Como a *CPU* usada é antiga e o seu desempenho não é alto em comparação às recentes, todas as constantes são convertidas em uma só no código, para reduzir o custo destas operações que são feitas com alta frequência.

A contagem dos pulsos são feitas através da placa de aquisição de dados (*Sensoray 526*) que é acoplada ao *PC104*. Como ela utiliza-se de contadores com estas tarefas específicas não é necessário interromper a *CPU* do robô para fazer contagens, assim o processo é mais rápido e pode ser executado nesta *CPU*.

2.5 Identificação

Para identificar o sistema foi usado o aplicativo de identificação de sistemas do *MATLAB*.

Para isso foi aplicada uma onda quadrada à entrada e foi estimada a velocidade de saída através dos *encoders*. Houveram alguns *outliers* devido a erros de sensor, mas foram retirados. A Figura 3 mostra estas ondas para a roda dianteira esquerda sem distúrbios e com as rodas livres.

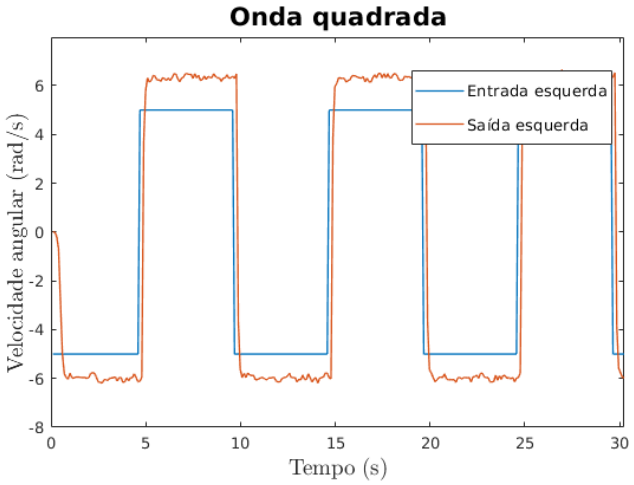

Figura 3. Onda quadrada - roda dianteira esquerda.

Na próprio aplicativo aplica-se um filtro passa-baixas para remover alguns picos que aparecem. Assim ao fornecer várias opções de combinações de polos e zeros nota-se que a função de transferência que melhor descreve este modelo é com dois polos e nenhum zero, assim encontramos a função de transferência da Equação (2).

$$G(s) = \frac{100.3}{s^2 + 12.39s + 81.3}. \tag{2}$$

O mesmo procedimento é aplicado a roda direita do *rover*.

3. CONTROLE DE VELOCIDADES E ODOMETRIA

3.1 Controlador de velocidades

O controlador implementado é um proporcional integral derivativo (PID) que é um dos mais usados para controle de velocidade de motores. No domínio de *Laplace* ele tem a estrutura mostrada na Equação (3). (Ogata, 2011)

$$G(s) = K_p + \frac{K_i}{s} + K_d s. \tag{3}$$

Para encontrar os ganhos do controlador utiliza-se o *PID tuner - MATLAB* com o modelo identificado. Segue abaixo os valores encontrados usando um tempo de resposta e comportamento transiente de aproximadamente 0.6:

$$K_p = 0.073123,$$
$$K_d = 0.00049427,$$
$$K_i = 2.7045.$$

O PID é simulado com *Simulink* através do sistema da Figura 4.

A simulação obteve os resultados mostrados na Figura 5. Como pode ser visto há um certo sobressinal mas que não prejudica o sistema.

Como os resultados são satisfatórios, com alguns cuidados como a utilização de uma saturação para a entrada, o controlador pode ser testado na planta real.

A Figura 6 mostra os resultados obtidos ao testar o robô com as rodas no chão.

Apesar dos ruídos fazerem a velocidade oscilar muito, como pode ser visto na Figura 6, os resultados visuais do robô são bem melhores já que a dinâmica do robô impede que haja oscilações tão bruscas e suaviza a curva de velocidade.

Para mostrar um pouco dos efeitos da dinâmica do robô pode-se testar o efeito de colocar diferentes velocidades nos dois lados do robô, a Figura 7 mostra os gráficos para este caso.

Como pode ser visto no gráfico a referência não é acompanhada tão rapidamente mas mesmo assim os resultados são bastante satisfatórios para o uso de apenas dois *encoders*.

3.2 Odometria

Com a estimativa de velocidade angular obtida é possível obter a odometria e ter uma boa estimativa de posição do robô. A velocidade angular é convertida em velocidade linear multiplicando pelo raio das rodas do robô. Usando um paquímetro o raio obtido foi de $6.05cm$.

A Figura 8 ilustra o robô e o ponto entre as duas rodas da frente é tomado com referência o robô.

A partir das velocidades das duas rodas e da distância entre o centro das rodas b é possível calcular a velocidade linear v_m do ponto de referência e a velocidade angular do robô ω como mostrado nas Equações (4) e (5) (Siciliano et al., 2010):

$$v_m = \frac{v_e + v_d}{2}, \tag{4}$$

$$\omega = \frac{v_e - v_d}{b}. \tag{5}$$

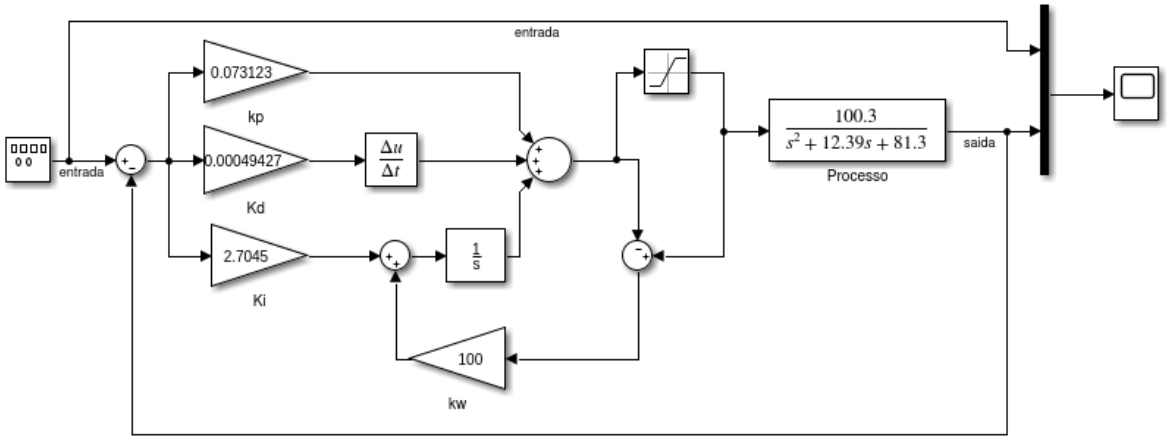

Figura 4. Modelo do sistema em Simulink

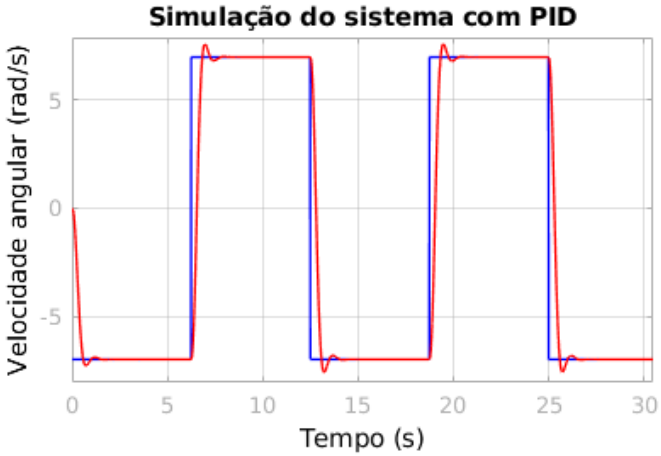

Figura 5. Simulação do PID

O valor de b medido no *rover* é de $28cm$. Estas equações de velocidade podem ser demonstradas simplesmente usando trigonometria.

Então a partir de v e ω também por trigonometria percebe-se que para calcular as variações de posição pelo tempo basta aplicar as equações 6 e 7.

$$v_x = v_m \cos \omega \tag{6}$$

$$v_y = v_m \sin \omega \tag{7}$$

Como v_x é o mesmo que a variação da posição em x pelo tempo, e como o robô foi implementado de forma que os dados são obtidos a aproximadamente cada $0.1s$, então a posição em x por integração numérica multiplicando o lado direito da equação pelo tempo Δt entre as operações. Assim, aplicando o mesmo princípio encontra-se a posição em y como mostrado nas equações 8 e 9.

$$x = v_m \cos \omega \Delta t \tag{8}$$

$$y = v_m \sin \omega \Delta t \tag{9}$$

A implementação foi feita usando várias ferramentas do *ROS* que ajudam calcular a estimativa de erros entre outros dados interessantes. (ROS wiki) Ao fazer uma experiência com 10 testes fazendo com que o robô ande em linha reta até que ultrapasse $2.98m$, ao medir a distância que o robô percorreu obtém-se os seguintes valores mostrados na Tabela 1

Ao calcular a média destes valores encontra-se 2.987. O erro é menor que 1%. Entretanto o erro de desvio da trajetória é calculado pelo *ROS* e pode ser usado para fazer também um controle que faça com que a distância percorrida pelas rodas seja a mesma, assim corrigindo isto o robô poderá seguir uma reta com uma precisão maior.

4. CONCLUSÃO

Como foi visto neste artigo, a odometria é de fácil implementação, baixo custo e de grande utilidade. A partir do que foi mostrado neste texto é possível criar um controlador para que o robô siga uma reta mais precisamente. Assim, uma trajetória poderia ser dividida em várias retas e o robô poderia segui-la. É possível combinar a odometria com câmeras, *kinects* ou outros sensores e programar o robô para cumprir tarefas bem mais elaboradas e com uma grande precisão.

REFERÊNCIAS

Carvalho, E.A.N. et al. (2010). Medição de velocidade angular com alta resolução usando encoders de baixa resulução e ppl. *Revista Controle & Automação*, 21.

Ogata, K. (2011). *Engenharia de controle moderno*. PRENTICE HALL BRASIL.

ROS wiki (2016). Publishing odometry information over ros. `http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom`.

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London. URL `https://books.google.com.br/books?id=jPCAFmE-logC`.

Site da Lynxmotion (2019). Lynxmotion - rover a4wd1. `http://www.lynxmotion.com/c-111-a4wd1-no-electronics.aspx`.

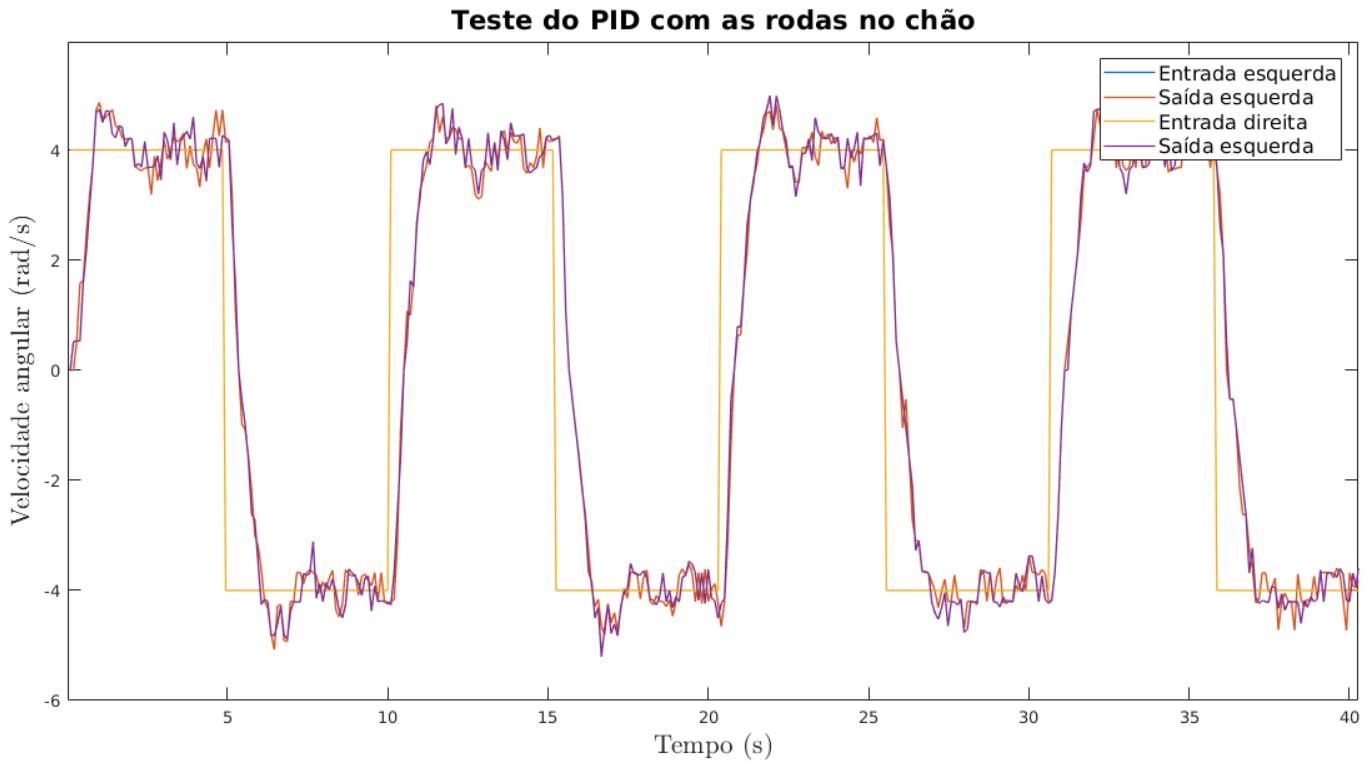

Figura 6. Teste do PID no robô

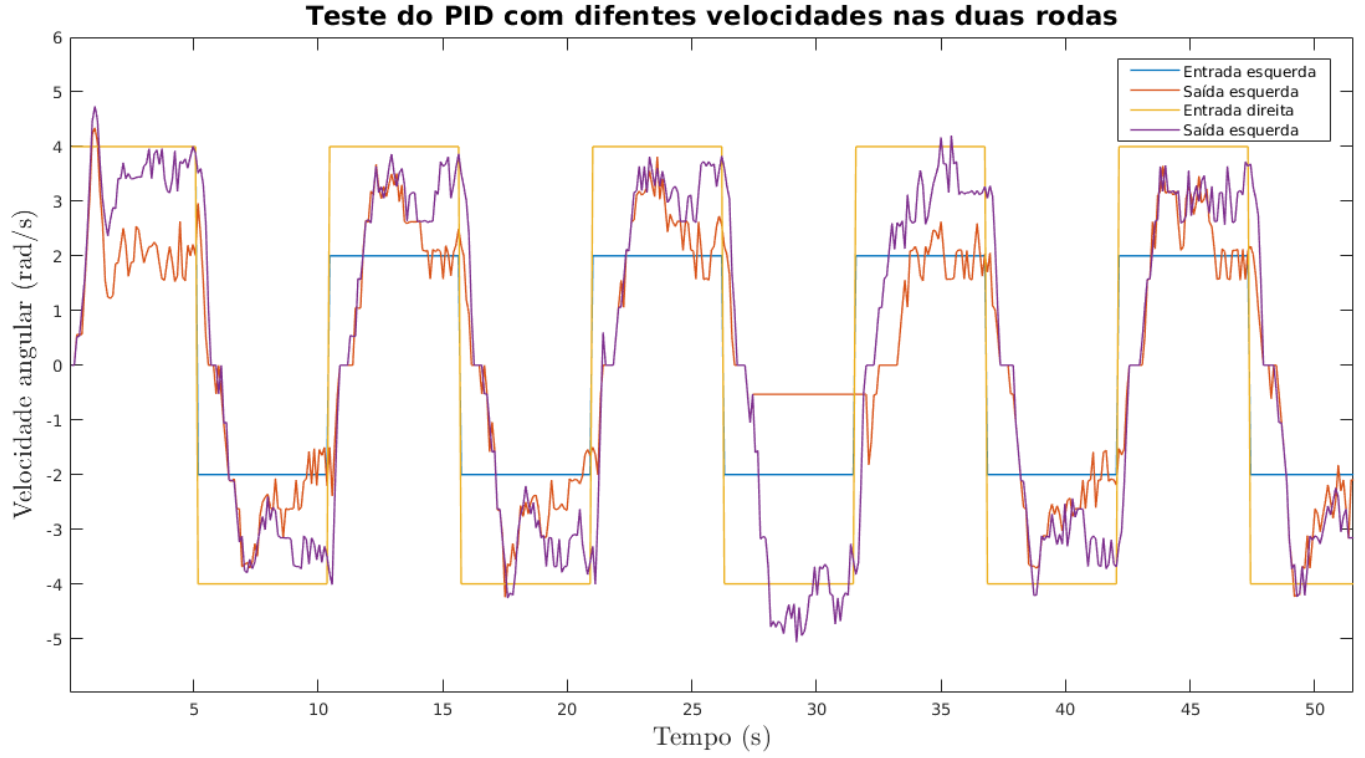

Figura 7. Teste com diferentes velocidades nas rodas do robô.

Distância percorrida									
3.01m	2.98m	2.99m	2.97m	2.99m	3.00m	2.96m	2.98m	2.99m	3.00m

Tabela 1. Distâncias percorridaspelo robô ao programar para percorrer $2.98m$

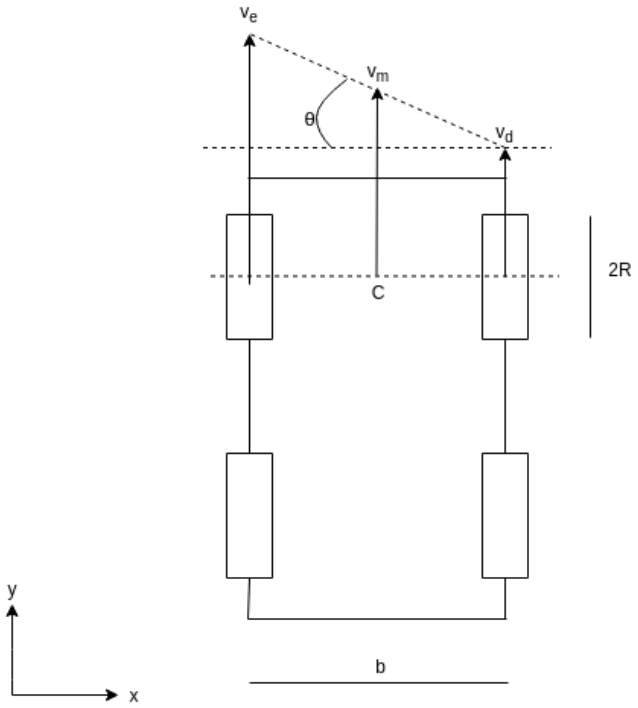

Figura 8. Representação de um robô com tração diferencial.

Vargas, J.A.D. (2012). Localização e navegação de robô autônomo através de odometria e visão estereoscópica. 76.