SIMULADOR DE CÓDIGO ABERTO PARA A PLATAFORMA *POLOLU 3PI*

José Roberto Fonseca e Silva Júnior, Maria Helena Rocha de Alencar Bezerra, Pedro Vitor Soares Gomes de Lima, João Marcelo Xavier Natário Teixeira, João Paulo Cerquinho Cajueiro, Guilherme Nunes Melo*

*Laboratório Maracatronics, DEMEC - CTG
Universidade Federal de Pernambuco, Recife, Pernambuco, Brasil

Email: jrfsj@cin.ufpe.br, m.helenalencar@gmail.com, p.vitor31@hotmail.com, jmxnt@cin.ufpe.br, joaopaulo@ufpe.br, guinm03@gmail.com

Abstract— Line follower robots have the ability to recognize and follow a line drawn on a surface. Its operating principles have elements that could be used in the development of numerous autonomous technologies, with applications in education and industry. The simulator that has been developed aims at solving one of the problems that involves developing using such robots: to perform several trials in order to validate a project. By taking the *Pololu 3pi* Robot as a model, the software simulates its physical structure, behavior and operations - as being able to read surfaces -, enabling the user to observe the robot following the line as the code commands. The simulator was validated based on experiments that included motion analysis and time measurements of pre-stablished tasks, so that its execution could be more coherent based on what happens in reality.

Keywords— Simulator, Line follower robot, Autonomous robot.

Resumo— Robôs seguidores de linha têm a capacidade de detectar e seguir uma linha desenhada em uma superfície. Possuem em seu princípio de funcionamento elementos que servem como referência para o desenvolvimento de várias outras tecnologias autônomas, com aplicações em educação e na indústria. O simulador desenvolvido visa facilitar uma das problemáticas que envolvem este tipo de ferramenta: a necessidade de realizar diversos testes para validar um projeto. Usando como modelo físico o *Pololu 3pi Robot*, o *software* simula seu funcionamento, age de acordo com a estrutura do robô real e é capaz de realizar leituras da superfície, permitindo observar o desempenho do robô ao seguir a linha operando com determinado código. O simulador desenvolvido foi validado a partir de experimentos que envolveram análise de movimentos e medições de tempo de execução de tarefas previamente estabelecidas, para que sua resposta fosse análoga ao que acontece na realidade.

Palavras-chave— Simulador, Seguidor de linha, Robô autônomo.

1 Introdução

Robôs seguidores de linha têm como objetivo percorrer, de maneira autônoma, uma trajetória definida. Esses robôs geralmente utilizam sensores infravermelhos com a função de detectar a posição do robô sobre a linha. Um controlador decide com base nessa informação os comandos apropriados para que o robô continue a seguir a linha, o que significa que ele trabalha com um mecanismo de realimentação, formando um efetivo e simples sistema fechado (Pakdaman and Sanaatiyan, 2009; Hasan et al., 2012).

Algoritmos de controle são escritos baseados nesse princípio e escolher aquele que faz o robô seguir o trajeto no menor tempo possível necessita de diversos testes. Essa grande quantidade de testes serve para analisar o efeito de vários parâmetros, de modo a obter um algoritmo mais eficiente. A dificuldade desses testes vem da necessidade de sempre ter à disposição uma pista totalmente montada, o que impede de fazê-lo em qualquer lugar que a pessoa esteja; além da necessidade de possuir o *hardware*, que apresenta um valor significativo (por volta de US$ 100,00), e também da questão prática de que as baterias deste tipo de sistema geralmente não duram muito, necessitando de recarga ou troca.

Este trabalho descreve um simulador para uma plataforma de seguidor de linha, com o intuito de facilitar os testes neste tipo de sistema. Simuladores, de maneira geral, possuem o benefício de facilitar a prática de uma determinada circunstância, o que favorece o desenvolvimento e progresso daquela área.

O simulador criado e apresentado neste artigo reproduz o comportamento do *Pololu 3pi Robot*, mostrado na Figura 1. Essa plataforma possui 2 micro motores, 5 sensores de reflexão, um *LCD* com 2 linhas de 8 caracteres, um buzzer e três botões, os quais funcionam como interface de entrada com o usuário. O robô possui um regulador de tensão que faz com que permaneça com uma tensão estável de 9,25V até que a bateria descarregue totalmente, permitindo assim que alcance velocidades de até 1m/s (3pi, 2008). Esse robô apresenta um sistema de direção diferencial, ou seja, a direção do movimento é dada através do controle eletrônico dos motores, portanto, não há necessidade de uma mecânica adicional para comandar a direção do robô, o controle da tensão em cada motor é suficiente.

O microcontrolador responsável por moderar todo o funcionamento do robô é o ATmega328P da Microchip (antiga Atmel). Ele possui vários compiladores compatíveis, tais como o GNU C/C++ e o Atmel Studio, que possui inúmeras bibliotecas e funções facilitando o controle do *hardware*.

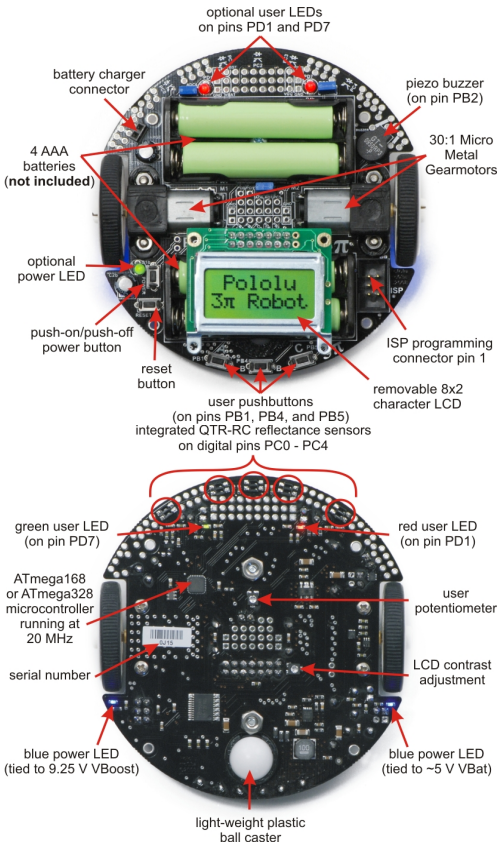

Figura 1: Plataforma robótica *Pololu 3pi*. Imagem retirada de (3pi, 2008).

Esta plataforma também pode ser programada pela (IDE) do Arduino, que foi escolhida neste estudo por se tratar de um ambiente conhecido por muitos e de fácil aprendizagem para os que desejam utilizar esse simulador.

Este artigo está organizado da seguinte forma: a Seção 2 mostra trabalhos semelhantes ao deste artigo e a problemática que motivou a criação deste simulador; a Seção 3 descreve a modelagem da movimentação do robô para possível implementação; a Seção 4 descreve o funcionamento, interfaces e funcionalidades do simulador; a Seção 5 descreve a metodologia utilizada para validar o simulador, junto com os testes comparativos entre o simulador e a plataforma real; a Seção 6 discorre sobre os resultados dos testes; por fim, a Seção 7 lista as principais contribuições deste trabalho, assim como os objetivos alcançados e os trabalhos futuros.

2 Trabalhos relacionados

Plataformas de simulação são comumente utilizadas por várias pessoas para, normalmente, eliminar os problemas existentes em realizar testes com modelos reais. Algumas referentes a robôs seguidores de linha foram encontradas e são apresentadas a seguir:

- **V-REP (Robotics, 2010):** tem como objetivo integrar um ambiente geral de simulação de robôs. Oferece sensores, mecanismos, robôs e sistemas que podem ser modelados e simulados nessa plataforma através de um conjunto de linguagens de programação acessíveis no programa.
- ***Line follower simulator* (Staněk, 2010):** simula um robô seguidor de linha, usando um modelo de rodas com rotação diferencial. A largura do sensor de linha, a posição do sensor de linha e a distância entre as duas rodas podem ser ajustadas. Esses ajustes são feitos através de barras de rolagem e não são apresentados os valores exatos de cada um.
- ***Simple autonomous robot simulator* (Bozic, 2012):** idealiza um robô autônomo implementado com controle PID e ruído gaussiano que simula o comportamento de uma bicicleta. Tem como objetivo replicar o comportamento do autônomo em busca de um caminho predefinido baseando-se nos algorítmos *pathfinder A** e *DP* (*dynamic programming*).
- **Gazebo (Howard, 2002):** simulador dinâmico em 3D com a capacidade de simular robôs com precisão e eficiência em ambientes internos e externos complexos. Oferece simulação física e um alto grau de fidelidade, um conjunto de sensores e interfaces para usuários e programas.

Entretanto, a carência de algumas funcionalidades e a dificuldade de utilizar o modelo do *Pololu 3pi Robot* nesses simuladores encontrados motivou a criação do proposto. A Tabela 1 apresenta características importantes em um simulador e analisa a existência delas no desenvolvido e em outras plataformas. Os parâmetros para comparação foram:

a) **Código aberto:** código disponível para download por qualquer pessoa e com um pensamento voltado para a colaboração entre desenvolvedores;

b) **Suporte à leitura de sensores:** possibilidade de acompanhar a leitura dos sensores que estão presentes no robô simulado;

c) **Suporte ao código nativo sem alterações:** não necessidade de modificação na estrutura ou na sintaxe do código real para ser executado na simulação;

d) **Editor de pistas:** possibilidade de customizar a pista que o robô seguirá;

e) **Simulação semelhante ao comportamento real:** o resultado da simulação é similar ao que ocorre no modelo físico;

f) **Compatibilidade com o *Pololu 3pi Robot*:** possibilidade de utilizar o modelo do *Pololu 3pi Robot* na simulação;

g) **Generalidade:** suporta o uso de diferentes modelos de robôs para a simulação;

h) **Simulador autocontido:** oferecimento de uma plataforma para edição do código fonte dentro do próprio simulador.

Tabela 1: Comparação entre plataformas de simulação relacionadas

Plataforma	a	b	c	d	e	f	g	h
(Robotics, 2010)		x		x	x	x	x	x
(Staněk, 2010)	x				x			
(Bozic, 2012)					x			
(Howard, 2002)	x	x		x	x	x	x	x
Simulador proposto	x	x	x	x	x	x		

3 Cálculo da cinemática

O problema da cinemática do robô simulado consiste em controlar a movimentação do a partir da velocidade das suas rodas direita v_r e esquerda v_l. A questão é encontrar a próxima posição do robô com base na anterior, o que também é conhecido como *dead reckoning*. O procedimento aqui descrito é baseado na teoria detalhada em (SIEGWART and NOURBAKHSH, 2004).

Um objeto no plano, como é o caso de um robô seguidor de linha, tem, no quadro de referência local, ξ_l definida como sua posição (x e y em coordenadas cartesianas) mais orientação, dada pelo ângulo θ entre os quadros de referência do robô e local:

$$\xi_l = \begin{bmatrix} x_l \\ y_l \\ \theta_l \end{bmatrix}. \quad (1)$$

No caso da simulação, o quadro de referência local é a tela da simulação, enquanto o quadro de referência do robô tem o ponto (0,0) no centro do robô, que localiza-se na posição central entre as rodas do mesmo.

A localização do robô ao longo da simulação é obtida incrementando a cada intervalo dt de simulação a aproximação da diferença naquele dt, obtida pela derivada de x, y e θ vezes dt:

$$\xi_l[t] = \xi_l[t-1] + \dot{\xi}_l[t]dt = \xi_l[t-1] + \begin{bmatrix} \dot{x_l} \\ \dot{y_l} \\ \dot{\theta_l} \end{bmatrix} dt. \quad (2)$$

A variação da localização no quadro de referência local $\dot{\xi}_l$ é determinada multiplicando a variação da localização no frame de referência do próprio robô $\dot{\xi}_r$ pela matriz de rotação $\mathbf{R}$:

$$\dot{\xi}_l = \mathbf{R}\dot{\xi}_r = \mathbf{R} \begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix}, \quad (3)$$

onde $\mathbf{R}$ depende apenas do ângulo θ:

$$\mathbf{R} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (4)$$

Pode-se obter $\dot{\xi}_r$ por simples inspeção: o robô não consegue andar para a esquerda nem para a direita sem girar, logo $\dot{y_r} = 0$; o robô vai para frente e para trás de acordo com a média das velocidades das rodas, logo $\dot{x_r} = \frac{v_r+v_l}{2}$; e para cada roda agindo separadamente, o robô vai girar em torno da outra roda, ou seja, com raio $2l$, onde l é a distância da roda ao centro do robô. Somando o efeito das duas rodas tem-se $\dot{\theta_r} = \frac{v_r-v_l}{2l}$. Em forma matricial:

$$\dot{\xi_r} = \begin{bmatrix} \dot{x_r} \\ \dot{y_r} \\ \dot{\theta_r} \end{bmatrix} = \begin{bmatrix} \frac{v_r+v_l}{2} \\ 0 \\ \frac{v_r-v_l}{2l} \end{bmatrix}, \quad (5)$$

o que retornando à equação (2) produz:

$$\xi_l[t] = \xi_l[t-1] + \left(\mathbf{R} \cdot \dot{\xi}_r[t]\right) dt. \quad (6)$$

A seção seguinte descreve como a cinemática descrita foi implementada no simulador proposto.

4 Simulador

O simulador foi implementado em C/C++ e executa o mesmo código utilizado no ambiente de desenvolvimento IDE do Arduino para controle do *3pi*. Tal escolha justificou-se pelo fato da grande utilização da IDE pela comunidade, além da linguagem utilizada poder ser interpretada nativamente como código C/C++, não sendo necessário criar um interpretador do código dentro do simulador, bastando apenas que o mesmo código (original, sem modificações) seja adicionado como pertencente ao projeto do simulador.

Optou-se por utilizar bibliotecas que estivessem disponíveis em diferentes plataformas (*Windows*, *Linux*, *macOS*, entre outros), com o objetivo de difundir a solução proposta. Sendo assim, para a parte gráfica e manipulação das imagens adotou-se a biblioteca *OpenCV* (Bradski and Kaehler, 2000), a qual é multiplataforma. Além dela, a biblioteca *cvui* (Bevilacqua, 2016), também baseada em *OpenCV*, foi utilizada para facilitar a implementação de elementos gráficos da interface (botões de ação, por exemplo).

O simulador desenvolvido utiliza como base as bibliotecas *Orangutan* e *3pi Robot*, fornecidas para a plataforma *Arduino* pela *Pololu*. Todos os arquivos de cabeçalho destas bibliotecas foram substituídos por versões equivalentes controladas

pelo simulador. Dessa forma, as funções, que em um robô real são acessadas no ambiente de desenvolvimento do *Arduino*, são substituídas por suas versões equivalentes no simulador, como pode ser visto na Figura 2. Isso permite simular o sistema usando o código embarcado original.

A "tradução" dos arquivos das bibliotecas priorizou o código de exemplo fornecido pela Pololu, chamado `Simple3piLineFollower.ino`. Este pode ser executado sem problemas no simulador. Outras funções, não utilizadas neste código, serão implementadas futuramente para tornar o simulador mais abrangente.

Para garantir o mínimo funcionamento da simulação, um seleto grupo de bibliotecas e funções foram implementadas:

- `Pololu3pi.h`: função `readLine` – a fim de simular a leitura de linha com os sensores;
- `OrangutanPushButtons.h`: funções `isPressed` e `waitForRelease` – para o uso dos botões do *Pololu 3pi*;
- `OrangutanMotors.h`: função `setSpeeds` – para o comando de acionamento dos motores;
- `OrangutanLCD`: funções `gotoXY`, `clear` e `print` (com duas assinaturas - uma para `string` e outra para `char`) – para simular o *display LCD*.

Além das bibliotecas reimplementadas, desenvolveu-se o arquivo `Simulator.cpp`, que centraliza todos os processos que ocorrem no simulador. As bibliotecas implementadas foram incluídas neste código.

As funções `setup` e `loop`, que são comuns ao código fonte na IDE do Arduino, foram declaradas como externas e são chamadas pela *thread* de simulação. Resumidamente, a execução do Simulador inicia com a criação de uma *thread* que lida com a inicialização da simulação física e da interface gráfica. Logo em seguida a função `setup` do código original é executada uma vez e, por fim, o simulador executa repetidas vezes a função `loop`.

4.1 Simulação física

A *thread* do simulador concentra-se na função `simulateAndShow`, responsável por executar periodicamente a função `simulate`, que controla a movimentação do robô pelo método descrito na seção 3, e todas as outras funções que atualizam informações gráficas na tela. A função `draw3piRobot` desenha o robô na tela, na posição e rotação corretas.

A equação (6) é diretamente implementada no simulador numa *thread* que executa a cada dt, apenas convertendo a velocidade de cada roda para o equivalente a *pixels* por segundo.

Assim que a localização atual do robô é definida, calcula-se a posição dos sensores do robô no plano do simulador e faz-se uma comparação de cada componente RGB do valor da pista com o limiar definido (atualmente o valor utilizado é 10, encontrado por tentativa e erro). Valores abaixo do limiar para as três componentes (R, G e B) indicam a cor preta. Caso contrário, assume-se que a cor branca foi lida.

4.2 Interface

A interface gráfica do simulador possui várias opções e pode ser observada na Figura 3.

Pode-se observar, na Figura 3, uma imagem do *3pi* com cinco LEDs superiores, os quais representam os cinco sensores do robô real. Eles indicam de forma discreta se os referidos sensores leram ou não alguma linha. Também é possível observar os botões A, B e C, cujas funcionalidades são definidas pelo código Arduino carregado, assim como no robô real.

O botão *play/pause* controla a permissão da execução do cálculo da cinemática dentro da *thread* de simulação, sem alterar a execução da *thread* de controle. Portanto, o robô não se move quando a simulação está pausada, embora o código arduino continue a medir os sensores e atuar nos motores.

O robô é considerado "ligado" assim que o simulador começa a executar. Caso, segundo o código original, seja necessário apertar algum dos botões A, B ou C para dar início ao movimento do robô, o usuário deverá apertá-lo na interface. O botão *reset*, ao ser clicado, põe o seguidor na posição e rotação iniciais, zera o cronômetro e pausa a simulação.

O percurso a ser seguido pelo robô é um *bitmap* de 600 por 600 *pixels*, o que resulta numa resolução de 2,83 mm por *pixel* em relação ao sistema real. É possível ainda arrastar o robô para qualquer posição dentro desta tela, o que automaticamente pausa a simulação, necessitando acionar o *play/pause* para reiniciar.

Apertando a tecla `ESC`, o programa é encerrado e toda a simulação é salva, na forma de vídeo, com o nome `output.avi`, dentro da pasta do projeto. Além disso, a tecla `p` do teclado tem como função a captura de tela, e todas as imagens capturadas em uma simulação são salvas na pasta `prints`, dentro do diretório do projeto. Como exemplo dessa função e também para demonstrar o funcionamento do simulador, uma sequência de 10 capturas de tela foi feita, espaçadas em aproximadamente 2 segundos de simulação, e o resultado pode ser observado na Figura 4.

4.3 Editor

Para facilitar a criação das pistas de testes, um segundo programa foi implementado, que é um simples editor de *bitmaps* de tamanho 600 por 600, já com a linha de uma largura padrão para servir como a linha do simulador (equivale à largura de

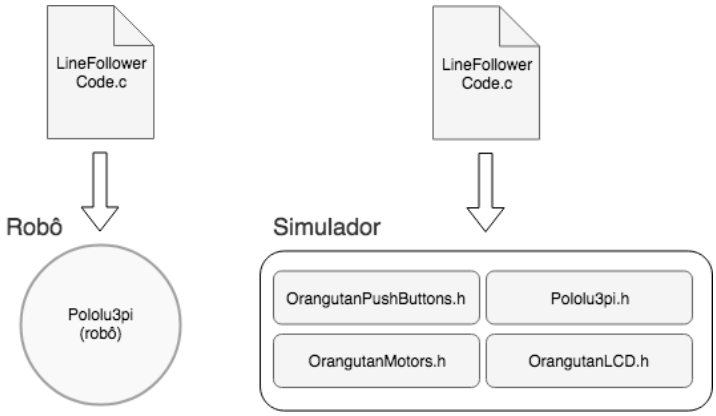

Figura 2: Arquitetura do simulador. À esquerda, é representado um código sendo usado para o robô seguidor de linha. À direita, o mesmo código é usado como fonte para a simulação.

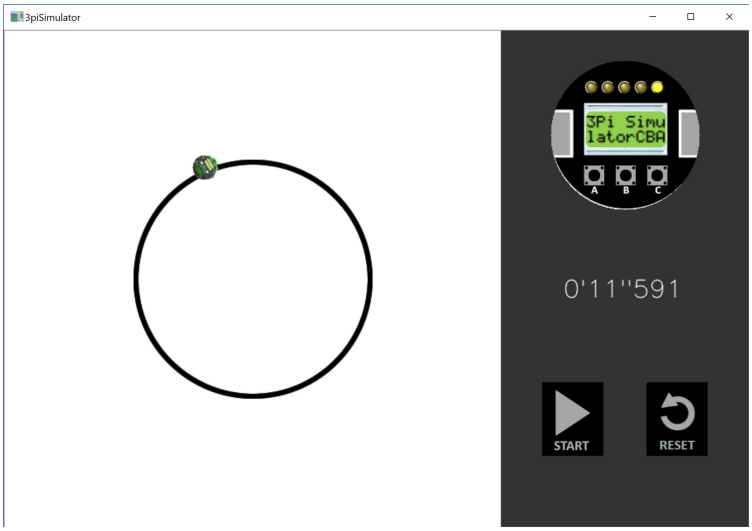

Figura 3: Interface do Simulador implementado.

uma fita isolante em relação ao *Pololu 3pi*). Este editor é mostrado na Figura 5.

5 Validação e testes

Ajustadas as dimensões físicas, apenas mais uma característica deve ser analisada para que o simulador tenha o comportamento próximo ao sistema real: a relação entre a velocidade do robô simulado e a velocidade real do robô.

Usando a própria biblioteca do *Pololu*, define-se a velocidade das rodas através da função `OrangutanMotors::setSpeeds(vr, vl)`, onde `vr` e `vl` são as velocidades das rodas direita e esquerda, respectivamente, e estão na faixa de -255 a 255. Apesar do nome `setSpeed`, esta função na realidade ajusta a potência dos motores, através da aplicação de um sinal PWM (do inglês *Pulse Width Modulation* - modulação por largura de pulso).

Numa primeira análise, a velocidade das rodas – e, portanto, do sistema como um todo – é proporcional a este sinal, mas três fatores influenciam sobre esta consideração:

- A **inércia do sistema** causa uma aceleração e/ou desaceleração até alcançar a velocidade final;
- Existe uma **faixa morta de operação**, então o sistema não consegue se mover para sinais de potência muito baixa;
- O **escorregamento das rodas** devido à falta de atrito entre as rodas e a pista.

Para determinar a relação entre as velocidades, foi feito o seguinte teste: foram atribuídos os mesmos valores para os dois motores e estabelecidas diferentes potências, a partir da função `setSpeed`, durante variados intervalos de tempo Δt. Mediu-se, então, a distância percorrida d_p em linha reta pelo robô em cada situação e, com isso, encontrou-se a velocidade v_m utilizando a relação velocidade = distância/tempo. Cada situação foi repetida três vezes e o resultado do teste, além do desvio padrão amostral DP, é mostrado na Tabela 2.

A partir da Tabela 2 é possível observar a relação resultante no gráfico da Figura 6. O valor

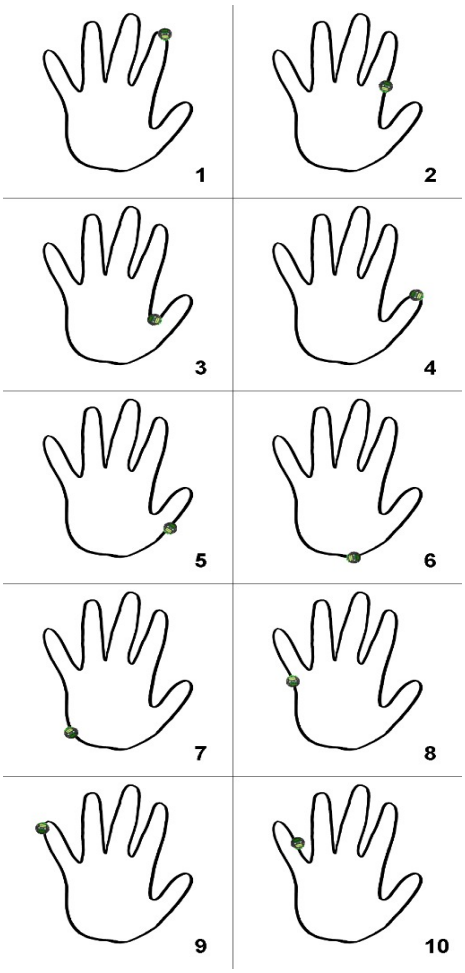

Figura 4: Conjunto de 10 frames consecutivos, espaçados em aproximadamente 2 segundos de simulação cada, representando o percurso realizado pelo robô simulado usando como base o código *SimpleLineFollower.cpp*.

Tabela 2: Resultado do teste para relacionar as velocidades do robô simulado e do robô real

setSpeed	Δt(ms)	d_p(mm)	v_m(m/s)	DP
100	500	226	0,452	
100	1000	469	0,469	0,0095
100	2000	936	0,468	
75	500	170	0,340	
75	1000	345	0,345	0,0038
75	2000	695	0,347	
50	500	106	0,212	
50	1000	220	0,220	0,0046
50	2000	440	0,220	
25	500	47	0,094	
25	1000	98	0,098	0,0023
25	2000	196	0,098	

de `setSpeed` pode variar de -255 a 255, porém, o teste foi feito desconsiderando os valores negativos, visto que o robô não se movimenta para trás quando executando a tarefa de seguidor de linha. Para as velocidades positivas, foram feitos testes com os valores de 25 a 100 e foi visto que a relação encontrada também é válida para outros valores

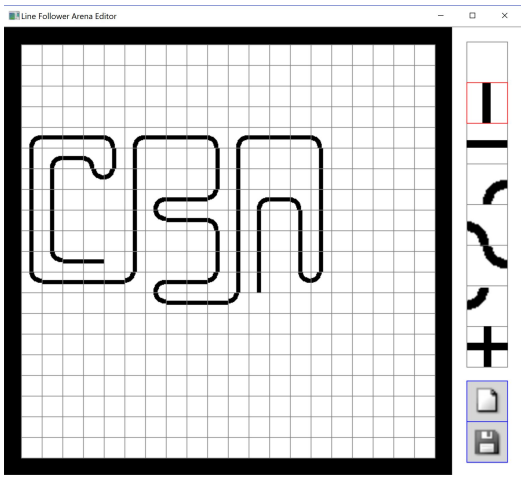

Figura 5: Interface do Editor de percursos.

de velocidade entre 0 e 255.

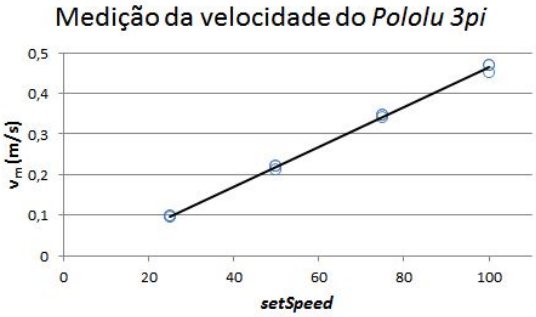

Figura 6: Relação entre os valores comandados via código e a velocidade real do robô.

Fazendo o ajuste linear da relação das velocidades da Figura 6, obtém-se que a relação entre a velocidade real do robô v_{real} e o valor comandado v_c é descrito por:

$$v_{real} = (0,005v_c - 0,031)\text{m/s}, \qquad (7)$$

onde o termo constante descreve a faixa morta, ou seja, o robô não se movimenta para v_c menor que $\frac{0,0310}{0,005} = 6,2$.

A Equação (7) foi implementada no simulador com os devidos ajustes, transformando-a para *pixels* por segundo e adaptando-a para que a zona morta fosse considerada.Novos testes foram feitos para observar se essa relação é suficiente para uma correta simulação do robô.

Como primeiro teste, comparou-se as distâncias percorridas pelo robô real com as do simulado ao aplicar diferentes potências (definida pelo `setSpeed`) durante 1 segundo. Para facilitar a comparação, converteu-se a distância do robô real para o equivalente em *pixels* da simulação.

A Tabela 3 compara as distâncias percorridas e apresenta o erro percentual conforme a potência aplicada. Os valores exibidos são resultados da média de cinco rodadas de teste.

Tabela 3: Comparação entre o comportamento do modelo físico e da simulação ao percorrer uma reta

Potência	50	100	150	200	250
Real (px)	77,3	165,6	253,9	342,2	430,5
Simulado (px)	81,5	165,5	245,9	327,4	432,7
Erro (%)	-5,18	0,06	3,25	4,53	-0,52

Um segundo teste compara o comportamento em curvas usando um algoritmo de detecção de linha, para ver se a relação encontrada na Equação (7) permanece válida.

Utilizando o código de exemplo `3piSimpleLineFollower.ino`, analisou-se o tempo necessário para o robô dar uma volta completa em uma circunferência de raio R em seu modelo físico, t_{real}, e na simulação, t_{sim}, operando com diferentes velocidades. Cada condição foi repetida cinco vezes e foi analisada a média dos valores obtidos. A Tabela 4 apresenta uma comparação entre os tempos levados para o robô percorrer uma circunferência de raio $R = 395$mm, mostrando também o erro percentual associado. A Figura 7 ilustra uma comparação quadro a quadro entre o comportamento do robô real e o robô simulado.

Tabela 4: Comparação entre o comportamento do modelo físico e da simulação ao percorrer uma circunferência de raio $R = 395$mm.

Potência	50	100	150	200	250
t_{real} (s)	13,7	6,4	4,22	4,36	3,49
t_{sim} (s)	12,5	6,0	3,8	3,0	2,5
Erro (%)	8,89	5,96	12,23	47,30	40,73

O custo computacional envolvido em um passo da simulação física também foi avaliado. O tempo de execução da função responsável pelo cálculo da posição do robô, a `simulate`, foi medido 256 vezes. A aquisição foi feita em um computador com processador *Intel Core i5* de 6ª geração e 4 GB de memória *RAM*, adquirindo valores na ordem de microsegundos. A média e o desvio padrão referentes a esses valores estão apresentados na Tabela 5.

Tabela 5: Média e desvio padrão referentes ao tempo decorrido para a execução completa da simulação física.

Média (μs)	Desvio padrão (μs)
50,56	18,88

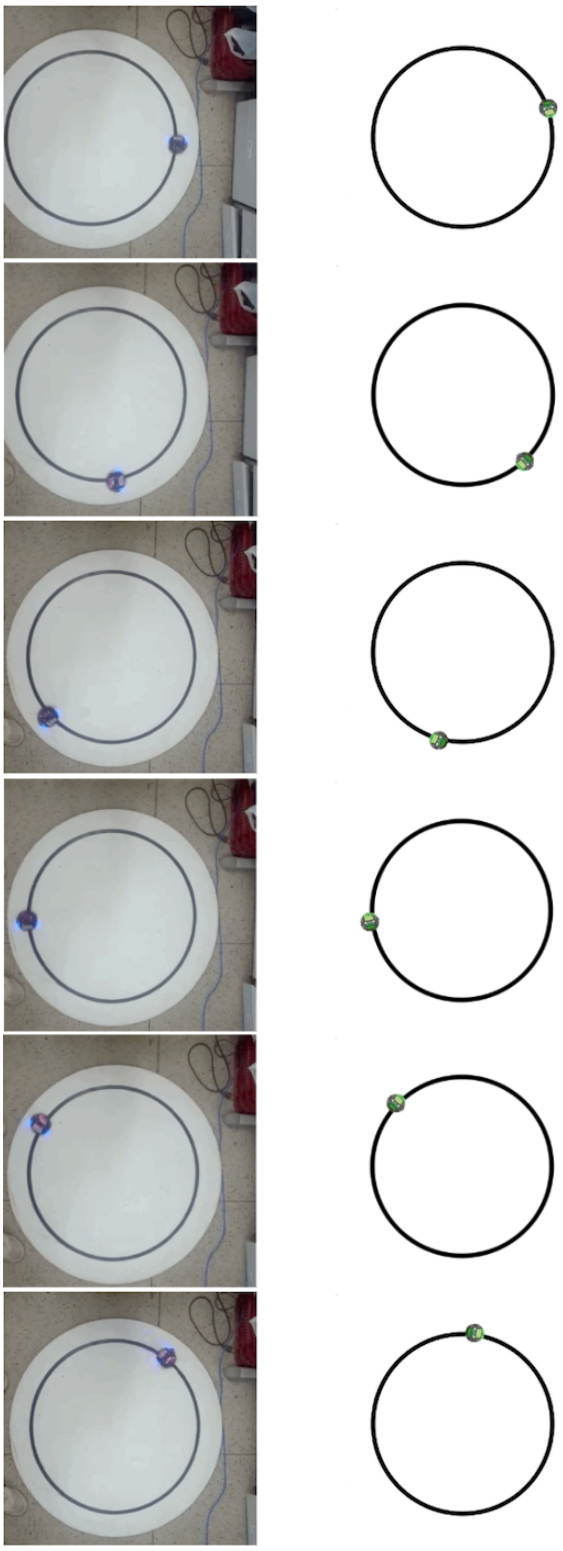

Figura 7: Comparação (alguns quadros) entre robô real e simulação.

6 Discussão

Na Figura 6, cada resultado do eixo do valor comandado (`setSpeed`) tem associado a ele três valores de velocidade medida (devido ao teste realizado), porém, a distinção entre eles não é clara, visto que se aproximam, podendo ser observado com auxílio do desvio padrão. Isto significa que, para os valores comandados utilizados nos testes, a velocidade medida independe do tempo que o robô foi acionado com aquele determinado valor. O que influencia diretamente nessa relação já foi

citado na Seção 5 e, mesmo desconsiderando alguns desses parâmetros no simulador, foram obtidos resultados aceitáveis. Dentre os erros mostrados na Tabela 3, aquele com o maior valor, em módulo, é de $5,18\%$. Considerando que o tipo de robô que está sendo trabalhado é comumente usado para fins educativos ou em competições, onde, normalmente, uma reta não ultrapassa 4m de comprimento, esse valor de erro não é considerado alto, porém visa-se dimunuí-lo em futuras versões do simulador.

Já os erros mostrados na Tabela 4 apresentam uma maior discrepância quando o robô manifesta velocidades maiores que 150. Os fatores que influenciam são, também, os mencionados na Seção 5 e, como o resultado obtido não leva em conta a implementação de alguns deles, é perceptível a necessidade de considerá-los para que o ambiente de simulação se aproxime ainda mais do real.

7 Conclusões e trabalhos futuros

O simulador apresentado permite utilizar o mesmo código utilizado na programação da plataforma *Pololu 3pi*, sem modificações. O sistema simulado responde de maneira próxima ao sistema real, porém com limitações quanto ao uso de altas velocidades em curvas.

O sistema nessa versão já permite o teste de códigos do tipo seguidor de linha, porém, ainda é necessário fazer uma modelagem mais precisa do sistema físico para que se permita a otimização do controle. Fenômenos como inércia, atrito, etc. precisam ser modelados e implementados no simulador. Tais elementos podem ser adicionados através da incorporação de uma *engine open source* voltada para simulações físicas, como por exemplo o *Bullet* (Coumans et al., 2013). Após tal ajuste fino, é de interesse utilizar o simulador proposto em disciplinas de cursos de engenharia e obter *feedback* dos alunos.

Além disso, outras funções das bibliotecas do *Pololu 3pi* são úteis em casos além do seguidor de linha básico. Para que o simulador fique com um caráter mais geral, a implementação delas já está sendo trabalhada.

Outra possibilidade de melhoria em estudo é a executação de mais de uma simulação simultânea, ou seja, vários robôs em uma mesma arena. O que permitiria o uso desta ferramenta para estudar sistemas multi-agentes.

Visa-se também portar o simulador para html5/javascript, tornando a solução proposta ainda mais acessível, sem a necessidade de instalar e/ou compilar o projeto.

Outro método de desenvolvimento do simulador também será avaliado. Um exemplo é utilizar o *game engine* Unity 3D (Technologies, 2005), pois possui ferramental que ajudaria em questões já citadas anteriormente (e.g. dinâmica do robô) e viabilizaria o uso do simulador em distintas plataformas.

O projeto está disponível em um repositório que pode ser acessado pelo *link* https://github.com/maracatronics/3pisimulator.

Referências

3pi (2008). Pololu 3pi robot, `http://www.pololu.com/product/975`. Acessado: 2018-03-06.

Bevilacqua, F. (2016). cvui, `https://github.com/Dovyski/cvui`. Acessado: 2018-03-10.

Bozic, M. (2012). Simple autonomous robot simulator, `https://www.youtube.com/watch?v=KbqmqoQpcWY`. Acessado: 2018-03-10.

Bradski, G. and Kaehler, A. (2000). Opencv, *Dr. Dobb's journal of software tools* **3**.

Coumans, E. et al. (2013). Bullet physics library, *Open source: bulletphysics. org* **15**: 49.

Hasan, K. M., Al Mamun, A. et al. (2012). Implementation of autonomous line follower robot, *Informatics, Electronics & Vision (ICIEV), 2012 International Conference on*, IEEE, pp. 865–869.

Howard, K. (2002). Gazebo, `http://gazebosim.org/`. Acessado: 2018-06-27.

Pakdaman, M. and Sanaatiyan, M. M. (2009). Design and implementation of line follower robot, *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on*, Vol. 2, IEEE, pp. 585–590.

Robotics, C. (2010). V-rep, `http://www.coppeliarobotics.com`. Acessado: 2018-03-10.

SIEGWART, R. and NOURBAKHSH, I. R. (2004). *Introduction to Autonomous Mobile Robots*, A Bradford Book.

Staněk, O. (2010). Line follower simulator, `http://www.ostan.cz/LineFollowerSimulator/`. Acessado: 2018-03-10.

Technologies, U. (2005). Unity, `https://unity3d.com/pt`. Acessado: 2018-07-04.