UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUSTAVO DA SILVA CRUZ

DESENVOLVIMENTO DE UM LABORATÓRIO VIRTUAL DE ROBÓTICA PARA ENSINO DE CONCEITOS BÁSICOS DE ALGORITMOS E PROGRAMAÇÃO PARA INICIANTES

VITÓRIA DA CONQUISTA – BA
2023

GUSTAVO DA SILVA CRUZ

DESENVOLVIMENTO DE UM LABORATÓRIO VIRTUAL DE ROBÓTICA PARA ENSINO DE CONCEITOS BÁSICOS DE ALGORITMOS E PROGRAMAÇÃO PARA INICIANTES

Trabalho apresentado para aprovação na disciplina Trabalho Supervisionado II e como requisito parcial para obtenção do título de Bacharel em Ciência da Computação, na Universidade Estadual do Sudoeste da Bahia – UESB.

Orientadora: Profa. Dra. Maísa Soares dos Santos Lopes

VITÓRIA DA CONQUISTA - BA

2023

AGRADECIMENTOS

Primeiramente agradeço a Deus, que sempre me deu forças para superar as dificuldades e alcançar meus objetivos.

À toda minha família, por todo amor, educação e apoio incondicional fornecidos durante a minha vida.

À minha orientadora Maísa, pelo auxílio, sugestões, disponibilidade, paciência e todos os incentivos durante o processo de realização deste trabalho.

Por fim, estendo meus agradecimentos a todos aqueles que, de uma maneira ou de outra, fizeram parte da minha história no decorrer de todos esses anos na Universidade Estadual do Sudoeste da Bahia.

Muito Obrigado!

RESUMO

As disciplinas de algoritmos e programação são a base dos cursos da área de Tecnologia da Informação (TI), por conta disso os alunos possuem contato com essas matérias logo em seu primeiro semestre de vida acadêmica. Entretanto, para muitos desses estudantes, aprender a programar acaba sendo uma tarefa complexa e entre os motivos está a dificuldade em colocar em prática os assuntos abordados nas aulas. Na tentativa de melhorar essa situação, os laboratórios de robótica estão sendo cada vez mais utilizados como uma grande ferramenta para aumentar o interesse dos alunos e facilitar a aprendizagem. No entanto, os laboratórios geralmente são caros, difíceis de manter em boas condições e possuem uma baixa quantidade de equipamentos disponíveis. Com base nisso, este trabalho buscou desenvolver um laboratório virtual de robótica para resolver diversas limitações dos laboratórios físicos e ajudar os alunos no processo de aprendizagem. Para tal, foi utilizado o Blender para modelar o robô e o programa Webots para criar o ambiente virtual e a simulação. O robô que foi desenvolvido é um seguidor de linha com detecção de obstáculos e o ambiente é um percurso com diversos desafios. A partir dos resultados apresentados e dos testes realizados, foi possível verificar que o laboratório garante a interação dos usuários com toda a simulação e faz com que os mesmos possam praticar uma série de conceitos de programação.

Palavras-chave: robótica; laboratório virtual; simulador; programação de computadores; tecnologia educacional.

ABSTRACT

The disciplines of algorithms and programming are the basis of the courses in the area of Information Technology (IT), and because of this, students have contact with these subjects in their first semester of academic life. However, for many of these students, learning to program turns out to be a complex task and one of the reasons is the difficulty in putting into practice the subjects covered in the classes. In an attempt to improve this situation, robotics laboratories are being increasingly used as a great tool to increase students' interest and facilitate learning. However, laboratories are often expensive, difficult to maintain in good condition, and have a low amount of equipment available. Based on this, this work sought to develop a virtual robotics laboratory to solve several limitations of physical laboratories and help students in the learning process. For this purpose, Blender was used to model the robot and the Webots program to create the virtual environment and simulation. The robot that was developed is a line follower with obstacle detection and the environment is a route with several challenges. From the results presented and the tests carried out, it was possible to verify that the laboratory guarantees the interaction of the users with the entire simulation and allows them to practice a series of programming concepts.

Keywords: robotics; virtual laboratory; simulator; computer programming; educational technology.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

2D	Bidimensional
3D	Tridimensional
AJAX	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
AVA	Ambiente Virtual de Aprendizagem
AWS	*Amazon Web Services*
CAD	*Computer-aided Design*
cm	Centímetros
CPU	*Central Processing Unit*
CSS	*Cascading Stylesheets*
DART	*Dynamic Animation and Robotics Toolkit*
DDR4	*Double Data Rate 4*
DSR	*Design Science Research*
EJS	*Easy Java Simulations*
GB	*Gigabyte*
GHz	*Gigahertz*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
HD	*High-definition*
HTML	*Hypertext Markup Language*
IP	*Internet Protocol*
IWVL	*Web-based Virtual Lab*
L1R2	*Lara Remote Robot*
LabVIR	Laboratório Virtual Interativo de Robótica
LARA	Laboratório Remoto em AVA
LTS	*Long-term Support*
MATLAB	*MATrix LABoratory*
ODE	*Open Dynamics Engine*
OpenGL	*Open Graphics Library*
OSRF	*Open Source Robotics Foundation*
PC	*Personal Computer*
PHP	*Personal Home Page*

PR2	*Personal Robot 2*
RAD	Radiano
RAM	*Random-access Memory*
ROS	*Robot Operating System*
s	Segundo
SCARA	*Selective Compliance Assembly Robot Arm*
STL	*Standard Triangle Language*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
UCM	Universidade Complutense de Madri
UESB	Universidade Estadual do Sudoeste da Bahia
UNED	Universidade Nacional Espanhola de Educação a Distância
URDF	*Unified Robotics Description Format*
URL	*Uniform Resource Locator*
V-REP	*Virtual Robot Experimentation Platform*

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, a demanda do mercado por programadores e o interesse das pessoas pela programação não param de crescer e, consequentemente, os cursos da área de Tecnologia da Informação (TI) estão se tornando cada vez mais populares. Nesses cursos, as disciplinas de algoritmo e programação são fundamentais para os alunos porque estabelecem os conceitos básicos de algoritmos, raciocínio lógico e linguagens de programação (SIROTHEAU et al., 2018). Mas, para muitos iniciantes, aprender a programar pode ser uma tarefa muito difícil e costuma ser um dos maiores fatores que levam a reprovação ou evasão nessas disciplinas, variando em torno de 60% e 70% em turmas que iniciam com uma média de 50 alunos (VIANA et al., 2019).

Na tentativa de ajudar os alunos, a comunidade de educação em ciência da computação vem desenvolvendo e experimentando diversas abordagens para o ensino dos conceitos introdutórios de programação de forma didática e pedagógica. Algumas dessas abordagens incluem cursos temáticos com o uso de jogos, computação gráfica e robótica (HAKIMZADEH; ADAIKKALAVAN; BATZINGER, 2011).

Com os avanços tecnológicos na última década, a computação acabou se tornando algo indispensável na sociedade, transformando a maneira como vivemos, trabalhamos e nos divertimos. Uma das áreas mais evidentes quando falamos desse avanço tecnológico é, certamente, a robótica. Os software e hardware existentes hoje em dia estão tornando os robôs cada vez mais capazes de executar diferentes tipos de tarefas e de interagir com as pessoas e seu ambiente de maneiras únicas (RUS, 2019).

Nos cursos de TI, as atividades utilizando robótica são muito populares porque elas conseguem materializar o comportamento abstrato dos algoritmos e programas em artefatos concretos que atraem a atenção dos estudantes (KING; GURA, 2007 apud MAGNENAT et al., 2014). No entanto, existe uma grande dificuldade para possibilitar que todos os alunos tenham acesso a um laboratório de robótica, principalmente em universidades que dispõem de poucos recursos financeiros. Alguns dos principais problemas enfrentados são: equipamentos caros, limitação de tempo para uso, configurações complexas e a necessidade de uma sala com um amplo espaço.

Como uma alternativa, os laboratórios virtuais oferecem diversas vantagens, como: possibilidade de acesso remoto, flexibilidade de horários e a redução de custos (JARA; CANDELAS; TORRES, 2008).

1.1 Justificativa

A aprendizagem dos conceitos iniciais de algoritmos e programação é um dos principais desafios para muitos alunos que entram no curso de ciência da computação ou em qualquer outro da área de TI. Essa dificuldade acaba deixando o aluno desinteressado pelas matérias que têm a programação como base, refletindo em um alto índice de reprovação (SIROTHEAU et al., 2018).

Para tentar amenizar esse problema, é necessário buscar ferramentas para diversificar os métodos de ensino para os alunos. A Robótica Educacional é um recurso que pode ser utilizado para facilitar o aprendizado, desenvolver o raciocínio lógico e aumentar o interesse dos alunos. Um exemplo é o LARA (Laboratório Remoto em AVA), um ambiente virtual de aprendizagem da UESB, campus de Vitória da Conquista, que possui suporte a diversos experimentos para melhorar o processo de aprendizagem de disciplinas do curso de Ciência da Computação, sendo o laboratório remoto de robótica um desses experimentos (LOPES et al., 2016). Entretanto, laboratórios de robótica demandam um alto custo financeiro para a sua montagem e manutenção, além da dificuldade de disponibilizar equipamentos suficientes para todos.

Esse projeto tem por foco, portanto, desenvolver um laboratório virtual de robótica, a fim de ajudar no processo de ensino-aprendizagem de conceitos de algoritmos e programação e estimular o interesse dos alunos iniciantes. Por ser virtual, esse laboratório também irá possibilitar a capacitação de um número maior de alunos, a redução dos custos da instituição com laboratórios presenciais, uma maior autonomia dos alunos na aprendizagem e maior flexibilidade de horários para experimentos. A expectativa é que esse projeto sirva de base para diversos estudos, que futuramente ele seja integrado na arquitetura do LARA e também possa ser usado no desenvolvimento de outros laboratórios virtuais.

1.2 Objetivos

1.2.1 Geral

- Desenvolver um laboratório virtual de robótica para ensinar conceitos básicos de programação para iniciantes.

1.2.2 Específicos

- Modelar e implementar um robô virtual 3D que possa ser aplicado no contexto educacional.
- Criar um ambiente virtual 3D interativo para o robô virtual 3D.
- Configurar um web service de simulação para executar o laboratório.

1.3 Organização do trabalho

O restante deste trabalho está organizado da seguinte forma: A seção 2 apresenta alguns conceitos que estão ligados com o presente trabalho, as principais características dos quatro simuladores mais populares atualmente disponíveis e também traz alguns trabalhos relacionados com o desenvolvimento de laboratórios virtuais. A Seção 3 descreve sobre a metodologia utilizada para desenvolvimento do trabalho, o processo de desenvolvimento do robô e ambiente virtual, algumas características da interface de usuário, o modelo do banco de dados, a arquitetura e o protocolo de comunicação do laboratório. Na seção 4 é apresentado os resultados obtidos, além de descrever os testes que foram realizados e as e as limitações do projeto. Por fim, a Seção 5 apresenta as conclusões e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesta seção serão apresentados alguns conceitos que estão ligados com esse trabalho: robótica e robô virtual, laboratório virtual e simuladores de robótica, além de uma descrição dos quatro principais simuladores disponíveis atualmente.

2.1 Robótica e robô virtual

Segundo Niku (2020), a robótica pode ser definida como a arte, a base de conhecimento e a experiência de projetar, aplicar e usar robôs com a finalidade de facilitar atividades humanas. Os sistemas robóticos consistem não apenas dos robôs, mas também de outros dispositivos e sistemas que são usados em conjunto com os robôs para realizar as tarefas necessárias. Hoje em dia os robôs são usados para os mais diversos propósitos e em diferentes ambientes, seja na exploração espacial e submarina, para auxiliar deficientes físicos ou até mesmo para diversão, mas, para que os robôs possam ser úteis, eles precisam ser programados e controlados.

A robótica é um ramo da tecnologia que envolve diversas áreas do conhecimento como: microeletrônica, engenharia mecânica, matemática, física (cinemática) e a inteligência artificial. Apesar da popularidade dos robôs, essa enorme multidisciplinaridade que a robótica necessita pode infelizmente acabar reduzindo a sua capacidade de atrair novos entusiastas. Um robô, no entanto, só pode existir em consequência da união de todas essas disciplinas científicas, e é essa união que possibilita a criação dos mais variados tipos de robôs (MARTINS, 2006).

Entre os diversos tipos de robôs que estão sendo desenvolvidos e adotados hoje em dia, um deles é a simulação virtual de robôs físicos, que também são conhecidos como robôs virtuais ou robótica de simulação. Esses robôs possuem como um dos principais objetivos apoiar e facilitar o desenvolvimento no campo da robótica. Dependendo do ambiente virtual, eles podem ser divididos em robôs virtuais 2D e robôs virtuais 3D (ZHONG; ZHENG; ZHAN, 2020).

Segundo o Potkonjak et al. (2016) existem diversas vantagens e desvantagens de um robô virtual quando comparado a um robô físico. Primeiro sobre as vantagens:

O robô virtual é altamente extensível e configurável, permitindo que os usuários possam facilmente mudar a aparência, o comportamento e adicionar novos componentes (ex: sensores e atuadores) sem nenhum custo envolvido. Os robôs

virtuais também não correm riscos de sofrerem nenhum tipo de dano, portanto, a colisão com o ambiente e a sobrecarga do robô é permitida, dando a possibilidade para o usuário aprender com os erros. Além disso, diferente da maioria dos dispositivos reais de laboratório que geralmente possuem uma capa para proteção ou um chassi e componentes que não podem ser removidos facilmente, os robôs virtuais são altamente manipuláveis.

Agora alguns problemas e desvantagens:

Pelo fato de o robô virtual não existir fisicamente, então nenhum imprevisto realmente sério pode acontecer durante a sua manipulação. Isso pode acabar gerando nos usuários uma falta de seriedade, responsabilidade e cuidado, já que eles irão se sentir como se estivessem jogando um tipo de videogame. A experiência real sempre vai deixar o usuário mais responsável, sério e cuidadoso.

Finalmente, mesmo que o robô virtual seja ótimo para que o usuário possa praticar os seus conhecimentos e obter confiança em seu trabalho, é muito importante que ele tenha contato com o equipamento real, porque a única maneira de adquirir um conhecimento amplo é muitas vezes através da experiência prática real.

2.2 Laboratório virtual

O laboratório virtual é um ambiente de laboratório simulado por programa em que os alunos podem acessar e realizar diversos experimentos em um espaço virtual. (ALKHALDI; PRANATA; ATHAUDA, 2016).

Os laboratórios virtuais vêm sendo utilizados como alternativa para resolver alguns problemas que dificultam a utilização de laboratórios físicos e remotos como, por exemplo, a baixa quantidade de robôs e equipamentos disponíveis em relação ao alto número de alunos e a necessidade da presença quase constante de uma pessoa qualificada na sala do laboratório para resolver possíveis problemas nos equipamentos (CHAOS et al., 2013).

A nível pedagógico, os laboratórios virtuais servem para criar uma conexão entre os conteúdos teóricos que são vistos durante as aulas e a sua compreensão prática e experimentação, facilitando assim, a aprendizagem de determinadas matérias (MAHAJAN; KULKARNI; DIWAKAR, 2016). Nas matérias ligadas a programação, esses laboratórios virtuais ajudam o aluno a aprender os conceitos básicos de programação, que incluem: tipos de dados, estruturas de controle, funções,

arrays, design de algoritmos e a mecânica de execução, teste e depuração (PRIETO-BLAZQUEZ; HERRERA-JOANCOMARTI; GUERRERO-ROLDÁN, 2009).

Além da questão educacional, os laboratórios virtuais também possibilitam que os estudantes possam se preparar para diversos tipos de competições, por exemplo: competição de robô seguidor de linha, futebol de robôs e sumô de robôs. Para essas competições geralmente é necessário que os participantes treinem com vários tipos de pistas e, utilizando a simulação, é possível fazer isso de forma virtual (SUWASONO et al., 2017).

Mas, apesar de ser amplamente aceito que os sistemas de laboratório virtual e simuladores são o passo inicial desejado na educação, alunos mais avançados muitas vezes ainda necessitam de experiência prática com equipamentos reais. No entanto, com o rápido progresso nas tecnologias de computação gráfica, realidade virtual e mundos virtuais, essa barreira entre o que só pode ser feito no mundo real e o que pode ser feito no mundo virtual está ficando cada vez menor (POTKONJAK et al., 2016).

Com base nos resultados da análise de um conjunto de ambientes de laboratórios remotos e virtuais, Budai e Kuczmann (2018) identificaram os principais fatores que devem ser levados em consideração ao avaliar um sistema de laboratório virtual e propôs um conjunto de requisitos para se implementar um moderno sistema de laboratório virtual. Os principais requisitos são:

- Na parte do front-end, a interface de usuário principal precisa ser consistente e intuitiva. Além disso, é necessário minimizar as trocas de contexto durante a simulação, ou seja, buscar sempre apresentar em uma única interface de usuário todo o material que for relevante e relacionado com o experimento.
- Ainda no Front-end, é muito importante que o laboratório tenha um cliente baseado na Web e, para cumprir este requisito, a interface web deve usar tecnologias web como HTML5 e JavaScript, porquê com elas será mais fácil para criar uma interface intuitiva, autoexplicativa e garantir que ela possa ser acessada em qualquer navegador moderno independentemente do dispositivo que esteja instalado, como desktop, smartphones e tablets.
- No Back-end, o sistema deve ser composto por uma arquitetura de microsserviços em vez de um serviço monolítico. Desta forma, é fácil criar laboratórios virtuais escaláveis com diversas opções de ferramentas

disponíveis para cobrir todas as tarefas de operação desde a implantação até o gerenciamento, monitoramento e backup.

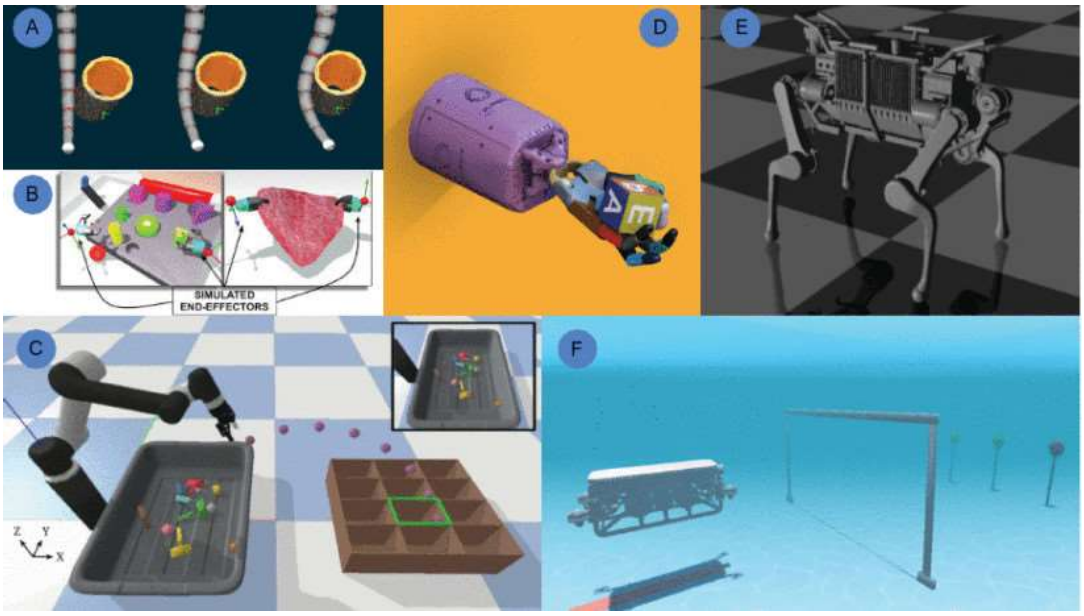
- As simulações devem ser executadas em um servidor remoto e os dados calculados são então enviados ao usuário via rede. Dessa forma, evita a necessidade do usuário instalar uma pilha de software e bibliotecas ou que exercícios complexos consumam recursos da CPU e memória do seu dispositivo.

2.3 Simuladores de robótica

Um simulador de robótica é um software que tem como objetivo tentar emular o mundo real e criar um ambiente virtual para robôs (KUMAR; REEL, 2011).

Os simuladores de robótica são ferramentas muito importantes porque permitem que os desenvolvedores possam projetar, prototipar e testar os sistemas robóticos de forma rápida e barata em um ambiente controlado, sem a necessidade de hardware físico. Dessa forma, os simuladores podem ser usados como uma ótima opção para realizar testes que seriam caros e/ou demorados se tivessem que ser realizados com robôs reais (AFZAL et al., 2020). Na Figura 1, Collins et al. (2021) trazem alguns exemplos de ambientes de simulações existentes na robótica: (a) robótica suave com Simulation Open Framework Architecture, (b) robótica médica com Asynchronous Multi-Body Framework, (c) manipulação com PyBullet, (d) manipulação hábil com MuJoCo, (e) locomoção com pernas com RaiSim e (f) veículos subaquáticos com URSim.

Figura 1 - Exemplos de simulações



Fonte: Collins et al. (2021)

Antigamente os simuladores de robótica geralmente precisavam ser criados do zero para simular especificamente uma linha de robôs de uma empresa ou para um objetivo específico. Porém, nos últimos anos, com a grande melhoria dos algoritmos de simulação, dos mecanismos de física, dos recursos gráficos e na velocidade de processamento dos computadores, surgiu a possibilidade de se desenvolver uma nova geração de simuladores capazes de simular qualquer tipo de sistema robótico (HARRIS; CONRAD, 2011). Os simuladores mais conhecidos atualmente, como V-REP, Gazebo e Webots, possuem a capacidade de simular uma grande quantidade de sistemas, incluindo robôs industriais, veículos autônomos e até veículos aéreos não tripulados (AFZAL et al., 2020).

Entre as principais razões para se realizar uma simulação estão a necessidade de testar protocolos de segurança e testar novos componentes, como sensores, pernas, rodas, etc (HARRIS; CONRAD, 2011).

Os simuladores de robótica modernos possuem uma vasta gama de recursos, como motor de física para movimentos realistas (Ex: ODE e Karma Engine), gráficos sofisticados com a utilização auxílios gráficos 3D (OpenGL e placas de vídeo), detecção de colisão, simulações de corpo rígido, simulação multiagente, integração com softwares de terceiros e assim por diante, possibilitando o desenvolvimento de qualquer tipo de robô. Muitos simuladores possuem bibliotecas adicionais que

fornecem recursos auxiliares, sejam de controle, gráficos ou visualização. Os robôs podem ser programados em várias linguagens de programação como C, C++, Java, C# e alguns simuladores até possuem a opção de transferir o código escrito na simulação para o robô real e então realizar a validação do hardware (KUMAR; REEL, 2011) (MELO et al., 2019).

Existem muitas opções de simuladores 3D disponíveis no mercado, pagos ou gratuitos, e é fundamental que o usuário escolha o simulador mais adequado, pois diferentes ambientes de simulação oferecem diferentes desempenhos, detalhes do modelo e recursos integrados, e essas características podem afetar o resultado, o sucesso e o mérito do experimento que está sendo realizado (MELO et al., 2019) (PITONAKOVA et al., 2018).

Logo abaixo é apresentada as principais características dos quatro simuladores mais populares atualmente disponíveis.

2.3.1 Gazebo

O Gazebo é um simulador de robótica 3D gratuito e de código aberto que é administrado pela Open Source Robotics Foundation (OSRF) com o apoio de uma comunidade muito ativa. O Gazebo é compatível com os sistemas operacionais Windows, Linux e Mac (GAZEBO, 2014). A Figura 2 mostra a interface do Gazebo.

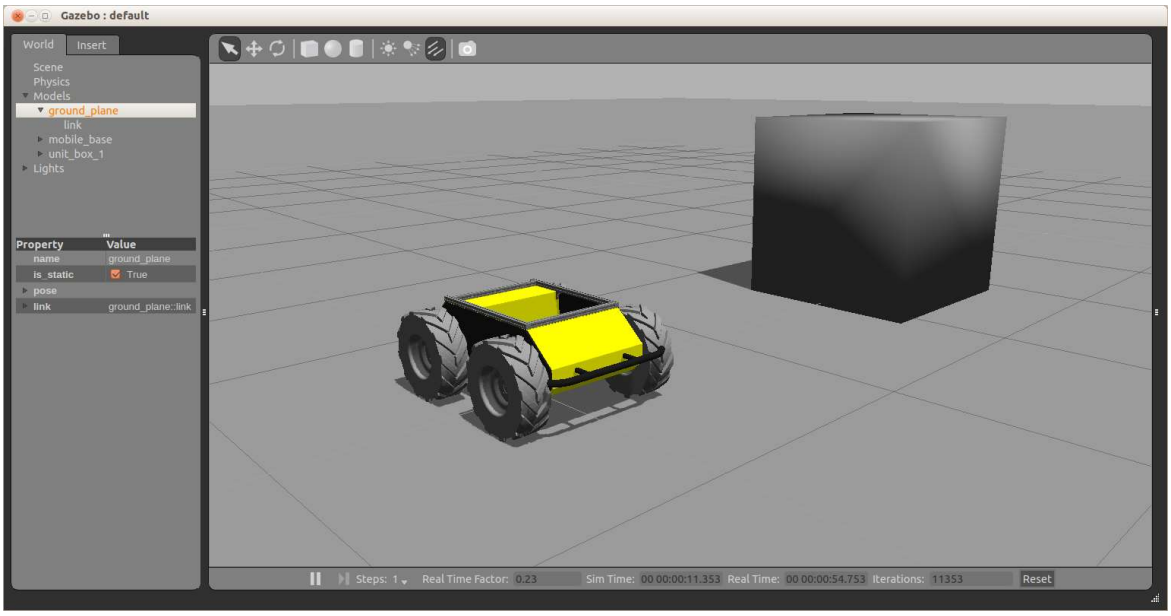
Com este simulador é possível construir ambientes realistas com uma renderização de iluminação, sombras e texturas de alta qualidade. Ele também possui suporte a diversos mecanismos de física robustos, incluindo ODE, Bullet, Simbody e DART (GAZEBO, 2014).

O Gazebo fornece uma grande quantidade de robôs prontos como PR2, TurtleBot, iRobot Create e Pioneer 2-DX, além de permitir que os usuários criem os seus próprios modelos. O Gazebo também possibilita que os robôs coletem dados realistas, opcionalmente com ruídos e falhas, e interajam com o ambiente de simulação através de diversos sensores e atuadores de alta fidelidade, como por exemplo: câmeras 2D/3D, telêmetros a laser, sensores estilo Kinect, sensores de contato, torque de força, GPS, etc (GAZEBO, 2014).

Com o Gazebo, os usuários e desenvolvedores também podem se comunicar com a simulação de diversas formas diferentes, incluindo uma interface gráfica de usuário, interfaces de linha de comando e Interface web. Além disso, é possível

executar as simulações de forma remota através do protocolo de comunicação TCP/IP ou rodar a simulação na nuvem utilizando o CloudSim e Amazon AWS. E, por fim, o Gazebo fornece suporte ao Robot Operating System (ROS), um framework muito popular utilizado para desenvolvimento de robôs (GAZEBO, 2014).

Figura 2 - Interface gráfica do Gazebo



Fonte: Ikuhara (2020)

2.3.2 CoppeliaSim (V-REP)

CoppeliaSim, anteriormente conhecido como V-REP, é um simulador de robôs multiplataforma que atualmente é desenvolvido e mantido pela Coppelia Robotics. Esse simulador é baseado em uma arquitetura de controle distribuído, onde cada objeto/modelo de um projeto pode ser controlado de forma individual por meio de um script. Esses scripts podem ser escritos em seis linguagens diferentes: C++, Java, JavaScript, Python, Matlab e Octave (COPPELIASIM, 2019).

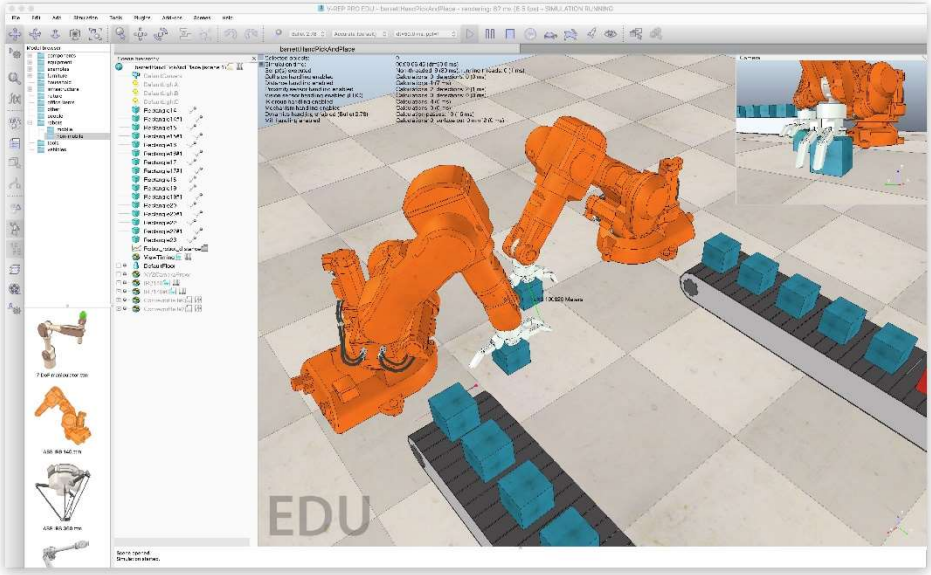
O CoppeliaSim pode ser usado, entre outras coisas, para desenvolvimento rápido de algoritmos, prototipagem e verificação rápidas, educação relacionada à robótica, simulações de automação de fábrica, verificação dupla de segurança e monitoramento remoto, ele conta com uma licença gratuita para hobbystas, estudantes, professores, escolas e universidades (COPPELIASIM, 2019).

O CoppeliaSim já conta com uma grande variedade de robôs, sensores e atuadores, mas também é possível adicionar novos sensores e atuadores por meio de um plug-in ou importar os seus próprios modelos de robôs em diferentes formatos 3D (Ex: Obj, STL, URDF). Também é possível construir sensores, atuadores e até sistemas robóticos inteiros dentro do próprio CoppeliaSim através da combinação de objetos básicos e vinculando várias funcionalidades por meio de scripts incorporados (COPPELIASIM, 2020).

O simulador conta com 4 motores de física (Bullet Physics, ODE, Newton e Vortex Dynamics), permitindo assim que a física, as interações e as colisões de objetos sejam mais fidedignas com o que vemos na vida real. Ele também suporta física de partículas dinâmicas, que é utilizada para simular jatos de ar ou água, motores a jato, hélices, etc (COPPELIASIM, 2020).

O CoppeliaSim pode ser facilmente incorporado em uma outra aplicação por meio de sua API e, além de ser multiplataforma, também é possível controlar uma simulação ou o próprio simulador remotamente por meio de outro PC ou um robô real (COPPELIASIM, 2020). A Figura 3 mostra a interface gráfica do CoppeliaSim.

Figura 3 - Interface gráfica do CoppeliaSim



Fonte: Northwestern (2020)

2.3.3 Unity

Unity3D é um motor gráfico desenvolvido pela Unity Technologies com o propósito de permitir que os usuários criem ambientes 3D e 2D atrativos, sua criação foi voltada para a criação de jogos. Com o Unity é possível desenvolver softwares que podem ser jogados diretamente num web browser (por exemplo Microsoft Edge, FireFox e Google Chrome), consoles de videogame (por exemplo, Xbox, Switch e PlayStation 5), dispositivos móveis (por exemplo, iPhone, iPad e dispositivos Android) ou computadores pessoais, por isso é extremamente atraente para uma ampla gama de desenvolvedores (ANDALUZ et al., 2016).

Apesar de seu histórico como motor de jogo, o Unity também mostra um grande potencial para ser utilizado como plataforma para modelagem, programação e simulações robóticas (KONRAD, 2019).

O Unity é uma plataforma de testes capaz de simular com precisão o ambiente e as situações do mundo real, além das interações físicas do robô com esse ambiente, possibilitando que os usuários possam avaliar o desempenho de um robô em termos de localização, planejamento do movimento e controle (UNITY, 2020).

Possui um poderoso mecanismo de física PhysX da NVIDIA, um mecanismo de renderização que fornece vários shaders e efeitos gráficos que aumentam o realismo do ambiente, mecanismo de detecção de colisão entre outros. Além disso, suporta ambientes 3D de alta fidelidade, permite a modelagem de sensores e possui suporte oficial ao ROS (HUSSEIN; GARCÍA; OLAVERRI-MONREAL, 2018).

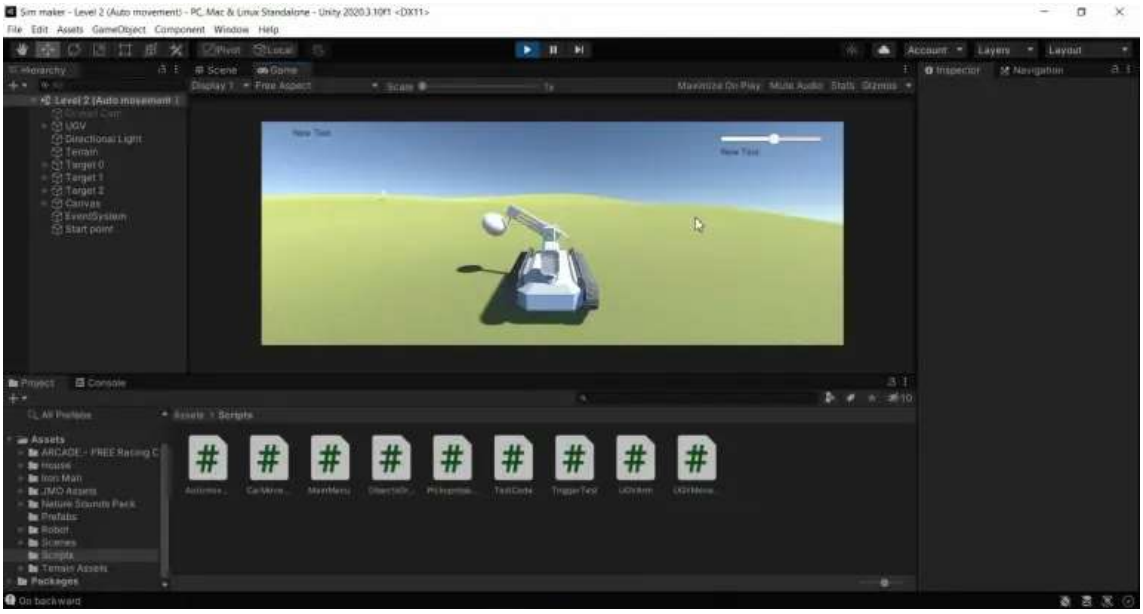
Similar a outros simuladores, cada objeto no Unity pode ser anexado a vários tipos de scripts, escritos em C#, Unity Script (JavaScript) ou Boo (Python). Esses scripts irão determinar os comportamentos desse objeto na simulação, por exemplo: um modelo de motor pode ser anexado com um script que irá indicar sua massa, sua velocidade máxima, quais partes dele podem girar e alguns outros parâmetros que irão replicar um motor real (LE; DANG; BUI, 2014).

Um modelo de robô pode ser importado diretamente para o Unity a partir de ferramentas de modelagem, como Blender ou Maya. Mais de um milhão de assets (itens que você pode usar em seu projeto) também podem ser comprados ou baixados gratuitamente na Asset Store da Unity (LE; DANG; BUI, 2014).

Além disso tudo, o Unity também possui uma ampla variedade de tutoriais on-line, uma API bem documentada e uma comunidade ativa e engajada, o que facilita a

aprendizagem e o desenvolvimento desde o primeiro dia (HUSSEIN; GARCÍA; OLAVERRI-MONREAL, 2018). A Figura 4 traz a interface gráfica do Unity.

Figura 4 - Interface gráfica do Unity



Fonte: Mecharithm (2022)

2.3.4 Webots

Webots é um simulador de robô 3D gratuito e de código aberto (licença Apache 2.0) que é mantido pela Cyberbotics e vem sendo amplamente utilizado na indústria, educação e pesquisa. Ele permite que os usuários possam criar mundos virtuais 3D com propriedades físicas como massa, juntas, coeficientes de atrito, etc. Além disso, os robôs podem ser equipados com vários dispositivos sensores e atuadores, como sensores de distância, rodas motrizes, câmeras, motores, sensores de toque, emissores, receptores, etc (CYBERBOTICS, 2020).

O Webots é um aplicativo desktop multiplataforma que roda em Windows, Linux e macOS e seus robôs podem ser programados em diferentes linguagens de programação como C, C++, Java, Python, MATLAB ou ROS (CYBERBOTICS, 2020).

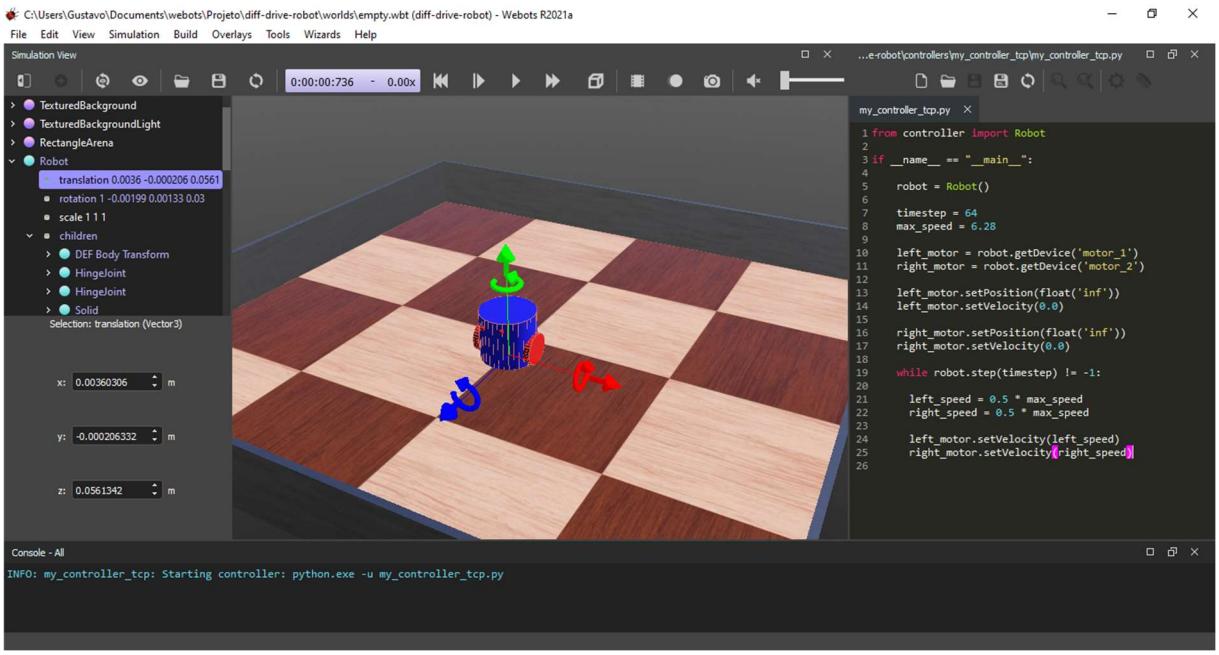
O Webots é um ambiente de desenvolvimento completo que vem pré-carregado com um grande número de modelos de robôs conhecidos, por exemplo, E-puck, Thymio II, iCub, Nao e Spot, mas também existe a possibilidade dos usuários

importarem os seus modelos feitos em softwares CAD (do Blender, AutoCAD, Solidworks ou URDF) (CYBERBOTICS, 2020).

O Webots possui um mecanismo de física open dynamics engine (ODE fork) e um mecanismo de renderização OpenGL 3.3 (wren) que possibilitam a criação de uma ampla variedade de simulações de forma realista, incluindo robôs de mesa de duas rodas, braços industriais, robôs com várias pernas, automóveis, drones voadores, veículos submarinos autônomos, veículos aeroespaciais, etc (CYBERBOTICS, 2020).

Além de ser muito robusto, o Webots também é bem documentado. No seu site oficial é possível consultar o Guia do usuário do Webots, o Manual de referência e também encontrar os links para uma comunidade ativa e um suporte técnico oficial, onde o usuário pode tirar dúvidas e relatar problemas (CYBERBOTICS, 2020). A interface do Webots é mostrada na Figura 5.

Figura 5 - Interface gráfica do Webots



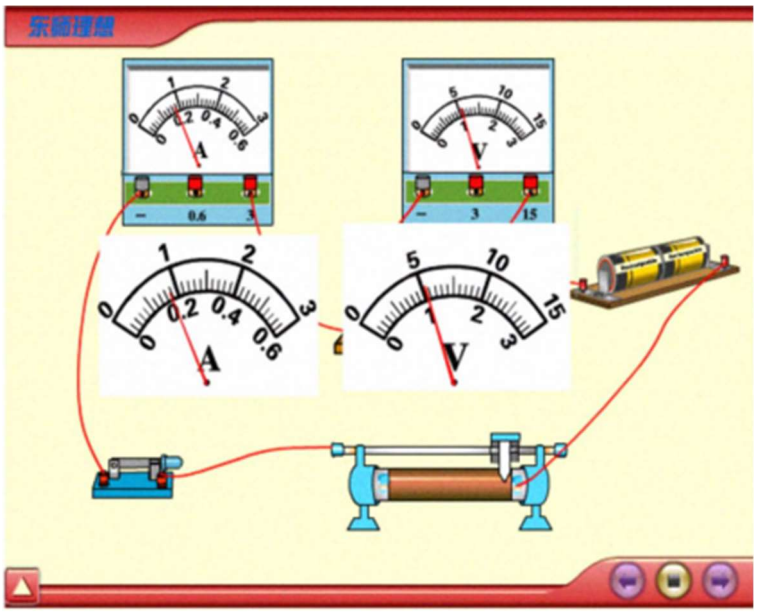
Fonte: De autoria própria

2.4 Trabalhos relacionados

Esta seção apresenta uma breve descrição de alguns trabalhos envolvendo laboratórios virtuais. O primeiro é um laboratório virtual de física, os outros três são laboratórios virtuais relacionados à robótica.

Li, Zhong e Zhon (2010) desenvolveram na Universidade Normal do Nordeste, China, um laboratório virtual baseado na web (IWVL) voltado para o ensino de física no ensino médio, principalmente nas atividades práticas. Os experimentos do IWVL foram organizados de acordo com as séries do ensino médio e também é possível realizar experimentos compartilhados, onde os alunos geograficamente distribuídos podem colaborar, comentar e discutir no mesmo ambiente. O laboratório possui uma arquitetura cliente-servidor e foi criado através da utilização de tecnologia multimídia, software flash, modelagem 3D com o 3Dmax e banco de dados. Segundo os autores, o IWVL ajudou os alunos a modelar e simular fenômenos físicos e a compreender mais facilmente os métodos e princípios físicos. A Figura 6 traz um exemplo de um experimento realizado no laboratório IWVL.

Figura 6 - Interface de um experimento do IWVL



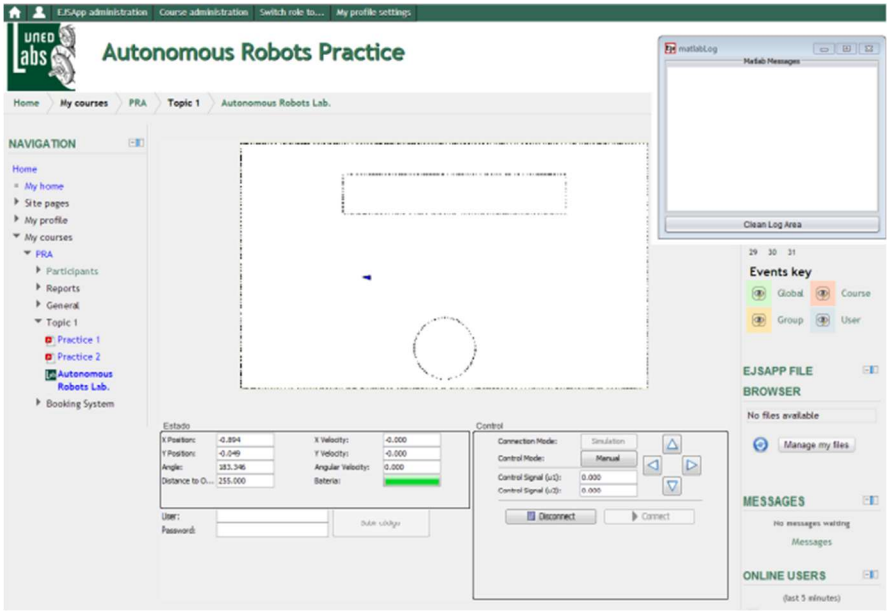
Li, Zhong e Zhon (2010)

O trabalho de Chaos et al. (2013) apresenta os laboratórios virtual e remoto desenvolvidos para os alunos da disciplina de Robôs Autônomos, no programa de Mestrado em Engenharia de Sistemas e Controle Automático da Universidade Complutense de Madri (UCM) e da Universidade Nacional Espanhola de Educação a Distância (UNED). Os laboratórios foram criados para o ensino a distância e online com objetivo de aumentar o entendimento dos alunos sobre a importância do papel que os sensores desempenham na robótica, possibilitando que os estudantes

programem os robôs com base nas leituras dos sensores e possam lidar com os problemas que ocorrem nos experimentos do mundo real. Além disso, o aplicativo permite que os alunos interajam com a simulação do robô (laboratório virtual) ou com um robô real (laboratório remoto), através da mesma interface gráfica simples e intuitiva desenvolvida em Easy Java Simulations (EJS). A Figura 7 mostra a interface do laboratório virtual.

Segundo Chaos et al. (2013), a utilização dos laboratórios desenvolvidos está atingindo ótimos resultados, o laboratório virtual, por exemplo, está possibilitando que os alunos se sintam mais confortáveis e confiantes ao poder testar o seu trabalho e fazer uma primeira depuração do código na simulação antes de testá-lo no robô real.

Figura 7 - Interface do laboratório virtual de Chaos et al.



Li, Zhong e Zhon (2010)

A Jara et al. (2011) também utilizaram a ferramenta EJS para desenvolver o RoboUALab, um laboratório virtual e remoto criado no curso de Ciência da Computação da Universidade de Alicante, na Espanha, para as disciplinas de Automação e Robótica. Esse laboratório permite que os usuários simulem e testem comandos de posicionamento de um robô por meio de um ambiente virtual que possui suporte à realidade aumentada, bem como executem comandos de alto nível em um robô industrial real de forma remota por meio da internet. Com o RoboUALab, os alunos podem aprender diversos conceitos complexos de maneira mais fácil, como:

cinemática, dinâmica, planejamento de caminho, modelagem de ambiente e programação de algoritmos.

Para realizar a comprovação dessas vantagens descritas, durante os anos de 2009 e 2010 os autores analisaram a percepção dos alunos sobre a experiência de aprendizagem através de um questionário que os estudantes tiveram de preencher. Os resultados deste estudo mostram que alunos gostaram de usar o ambiente híbrido proposto, se sentiram mais motivados e conseguiram aprender melhor com ele.

Também com o objetivo melhorar a qualidade do ensino da robótica, Xie et al. (2018) propôs um conceito de laboratório de robótica virtual baseado em uma plataforma virtual para simulação de robôs chamada V-REP. O laboratório virtual apresentado tem como foco os braços robóticos e conta com diversos modelos como Puma 560, Palletizer, SCARA e Delta. Além disso, ele possui experimentos relacionados à transformação de coordenadas, planejamento de trajetória cinemática, e dinâmica.

Segundo Xie et al. (2018), o laboratório virtual proposto conseguiu apresentar para os alunos, de maneira simples e interativa, conceitos básicos de robótica e também conteúdos que são mais difíceis de compreender. Além disso, mostrou ser também uma boa ferramenta para a realização de pesquisa sobre robôs, design de protótipos e validação de algoritmos.

3 LABVIR

O Laboratório Virtual Interativo de Robótica (LabVIR) é um sistema que oferece aos usuários a oportunidade de programar um robô móvel dentro de um ambiente predefinido. Esse sistema é uma aplicação web que adota a arquitetura cliente-servidor e que disponibiliza uma simulação 3D com física realista e uma interface simples onde o usuário pode ter uma grande interação com o ambiente virtual através do mouse, da barra de ferramentas ou dos menus de contexto. Além disso, também é possível escrever scripts para controlar o robô virtual, assim como faria com o real.

O objetivo do usuário no LabVIR é programar um robô seguidor de linha para que ele possa percorrer e completar o percurso, enquanto supera os desafios pelo caminho.

Essa seção apresenta a metodologia escolhida para desenvolvimento do projeto, as ferramentas que foram utilizadas, o processo de desenvolvimento do robô e ambiente virtual, algumas características da interface de usuário, o modelo do banco de dados, a arquitetura e o protocolo de comunicação do laboratório.

3.1 Metodologia

O método de pesquisa utilizado no desenvolvimento deste trabalho é o Design Science Research (DSR). De acordo com Lacerda et al. (2013), o DSR tem como objetivo projetar artefatos a fim de resolver problemas, avaliar o que foi projetado, comunicar os resultados obtidos e, dessa forma, produzir ciência sobre a realidade.

Os artefatos não são somente objetos físicos, mas qualquer solução desenvolvida com a finalidade de resolver algum problema em um determinado contexto, tendo como base conhecimentos obtidos e conjecturas sobre o mundo (PIMENTEL; FILIPPO; SANTORO, 2019).

O DSR tem como base a ideia de que, para que soluções inovadoras possam ser criadas, é necessário avançar o estado da arte. Sendo assim, a utilização do DSR para criar artefatos inovadores que solucionam um problema produz como efeito colateral um avanço no estado da arte (HORITA; NETO; SANTOS, 2018).

Na literatura existem diversos processo metodológico para DSR, sendo escolhido o processo proposto por Lacerda et al. (2013), que é composto por cinco etapas:

- **Conscientização:** essa é a etapa de compreensão do problema de pesquisa, com a finalidade de definir e formalizar o problema, suas fronteiras e as possíveis soluções.
- **Sugestão:** com os conhecimentos obtidos na etapa anterior, infere-se uma ou mais alternativas de artefatos para solucionar o problema. Posteriormente, um ou mais artefatos são escolhidos para serem desenvolvidos.
- **Desenvolvimento:** corresponde ao processo de criação do artefato. Como resultado, obtém-se o artefato em estado funcional.
- **Avaliação:** nesta etapa é observado e avaliado, no ambiente para o qual foi projetado, o desempenho do artefato para a solução do problema.
- **Conclusão:** formalização do processo e divulgação dos resultados em artigo científicos, relatórios técnicos e trabalhos acadêmicos.

Tais etapas foram instanciadas para este projeto da seguinte forma:

A etapa de Conscientização foi realizada pelos alunos e professores envolvidos no LARA. Durante o desenvolvimento e implementação do laboratório remoto de robótica eles encontraram uma grande dificuldade para possibilitar que todos os alunos pudessem utilizar o ambiente. Por conta disso, eles perceberam a necessidade de desenvolver uma ferramenta capaz de comportar um número maior de usuários.

Com o problema definido, na etapa de Sugestão foi feito um levantamento e avaliação das soluções já existentes para problemas semelhantes, e chegaram à conclusão de que a melhor alternativa seria o desenvolvimento de um laboratório virtual de robótica.

Na etapa de Desenvolvimento foi feita a criação do laboratório virtual. Os detalhes dessa etapa são explicados nas próximas seções e subseções.

Na etapa de Avaliação foram realizados testes para verificar se o laboratório estava funcionando conforme o esperado. Todos os testes realizados nessa etapa são descritos na seção 4.

A etapa de Comunicação consiste na elaboração e apresentação desta monografia.

3.2 Ferramentas e tecnologias utilizadas

3.2.1 Simulador

Entre os simuladores anteriormente descritos, o Webots foi o escolhido para esse trabalho. Além das características já citadas, um outro grande motivo para esse simulador ter sido selecionado é o fato dele possibilitar a configuração de um serviço web para executar a simulação na nuvem.

Alguns dos outros simuladores até possuem uma Interface web ou permitem executar as simulações de forma remota, mas com uma quantidade muito limitada de recursos. Somente o Webots oferece a opção para que os usuários possam editar o controlador do robô nas simulações online e ainda possibilita uma interação bem maior com o ambiente.

Para esse projeto foi utilizada a versão R2021a do Webots, porque as versões mais recentes apresentaram alguns problemas com as texturas na parte da simulação web.

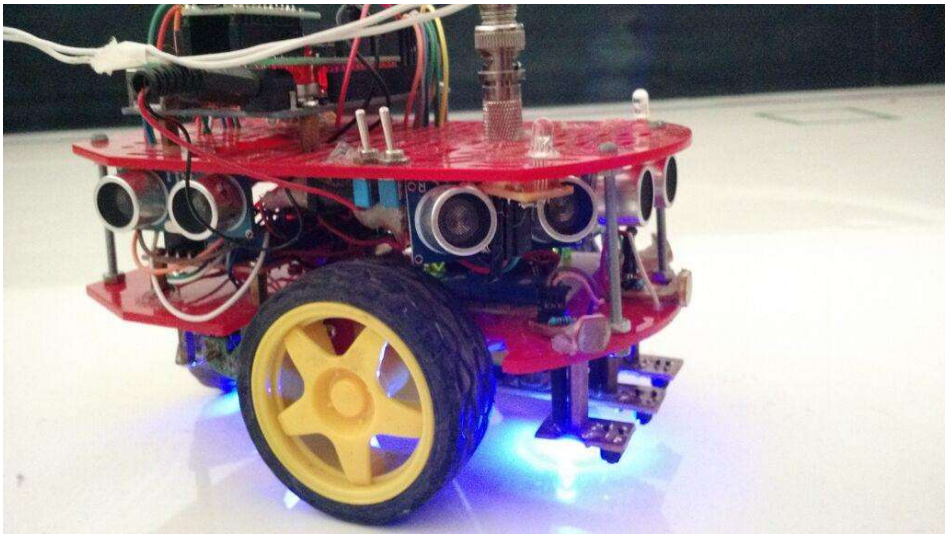
Mais detalhes sobre o funcionamento do serviço web serão explicados em outras seções e subseções deste trabalho.

3.2.2 O robô virtual

O próximo passo para a criação do LabVIR foi fazer a modelagem 3D do robô virtual e exportar para o Webots. O modelo foi baseado em um robô real chamado L1R2 (Lara Remote Robot), que é um carro seguidor de linha e resgate que pertence ao LARA, do curso de Ciência da Computação, na Universidade Estadual do Sudoeste da Bahia (UESB). A Figura 8 mostra a versão real do L1R2.

Para modelar o L1R2 foi utilizado um software de criação em 3D chamado Blender, esse programa disponibiliza uma variedade de ferramentas que podem ser usadas para fazer modelagens 3D, texturização, animação, renderização e edição de vídeos. Ele foi desenvolvido pela Blender Foundation, uma organização independente e sem fins lucrativos (BLENDER, 2021). O Blender foi escolhido porque é gratuito, possui uma grande biblioteca com componentes prontos que ajudam na modelagem, tem uma interface relativamente fácil de ser utilizada e porque é possível exportar os projetos nos formatos suportados pelo Webots.

Figura 8 - Robô L1R2



Fonte: LARA (2019)

3.2.3 O ambiente virtual

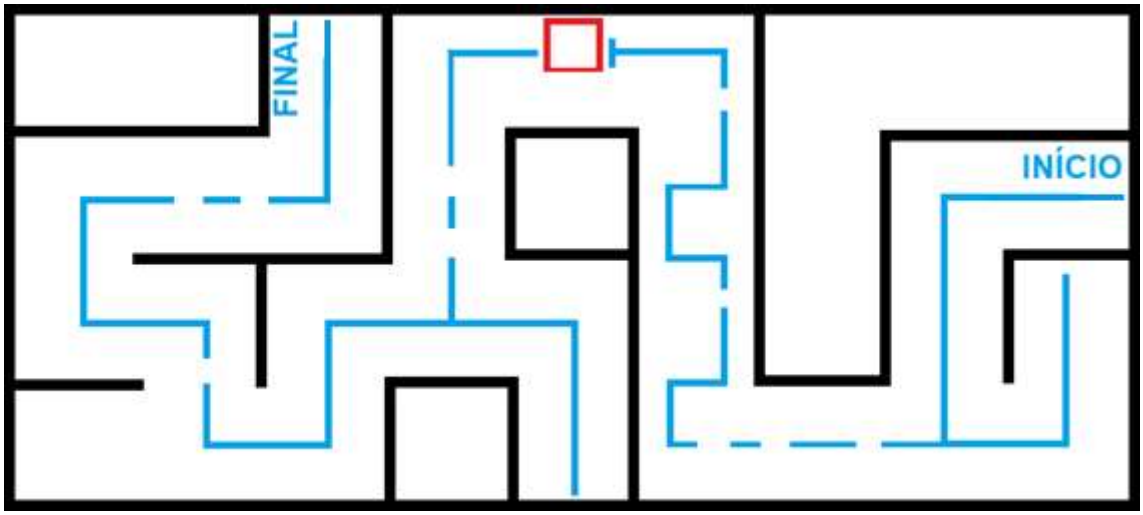
Além do Robô Virtual, também foi necessário criar o ambiente virtual 3D interativo. O ambiente construído é um circuito para um robô seguidor de linha, onde basicamente o robô tem que ser pré-programado para que, através da leitura de sensores, consiga detectar onde existe um caminho em uma superfície (uma linha) e também consiga desviar do obstáculo que existe no trajeto. Além de completar o percurso, mostrado na Figura 9, deve-se tentar fazer isso no menor tempo possível.

A pista para o robô é toda preta com a linha branca. Para desenhar as linhas foi utilizado o Tinkercad, uma ferramenta online e gratuita para modelagens 3D e também para simulação de circuitos elétricos analógicos e digitais, desenvolvida pela Autodesk (TINKERCAD, 2022). Todo o resto do ambiente foi desenvolvido utilizando o próprio Webots e a grande biblioteca de recursos que ele oferece.

O Webots representa a simulação com uma estrutura de árvore hierárquica em que o nó raiz é o mundo e seus nós filhos são os diferentes itens do ambiente (Robot, Solid, Shape, Group, etc.). Para verificar se o robô conseguiu completar o percurso, foi utilizado um tipo especial de robô chamado Supervisor. O Supervisor possui poderes adicionais e pode ser usado para diversos objetivos, como ler ou alterar a posição dos robôs, alterar a cor dos objetos e para obter medições sobre o estado da simulação. Para esse ambiente virtual, foi criado um Supervisor sem massa ou

quaisquer propriedades físicas na simulação, apenas com o objetivo de acompanhar a evolução do robô principal. Assim como no robô normal, o comportamento do supervisor foi definido por um controlador escrito na linguagem Python.

Figura 9 - Percurso do ambiente



Fonte: De autoria própria

3.2.4 Interface do usuário

Segundo Miyadera, Huang e Yokoyama (2000 apud Mota, Pereira e Favero, 2008), para que um sistema educacional seja realmente eficiente ele deve permitir que os estudantes possam visualizar não somente as animações, mas também os códigos dos programas e as explicações por escrito. Além disso, os estudantes gostam de interagir com a animação e ter o controle sobre ela, por isso o software deve trazer aos alunos a possibilidade de executar do algoritmo continuamente, passo a passo, ir e voltar, controlar a velocidade da execução da animação e deve permitir também a entrada de dados pelos usuários, melhorando assim a aprendizagem. A escolha do Webots para o projeto também levou em consideração todos estes princípios.

A interface do LabVIR utiliza como base o robotbenchmark.net, um site gratuito e de código aberto (licença Apache 2.0) que oferece uma série de desafios de robótica, ele foi desenvolvido pela Humain Brain Project e pela Cyberbotics. O robotbenchmark executa as simulações de robôs na nuvem e exibe os resultados em 3D utilizando uma pilha de software de código aberto 100% gratuito, como: HTML,

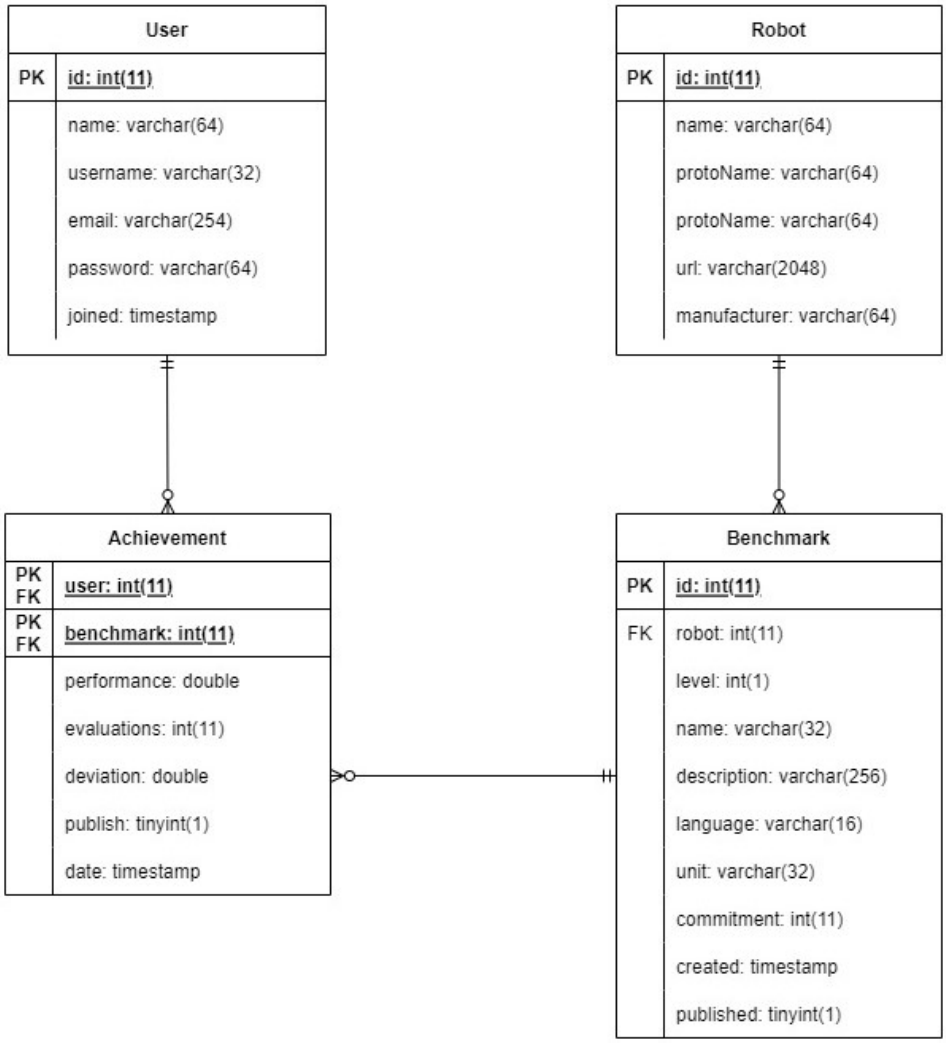
CSS, PHP, JavaScript e o próprio Webots (ROBOTBENCHMARK, 2018). O principal arquivo do robotbenchmark utilizado nesse projeto é o webots.min.js, responsável por exibir a cena da simulação no navegador do usuário.

Para que os estudantes possam compreender mais facilmente a simulação, foi criada uma janela personalizada para o robô. Essa janela apresenta aos alunos uma visão geral, instruções e explicações da simulação, além de permitir que eles visualizem o seu progresso. A janela do robô foi desenvolvida utilizando o HTML para definir o seu conteúdo, o CSS para definir a sua aparência e a linguagem de programação JavaScript para possibilitar que a interface se comunique diretamente com a simulação e com o controlador do robô.

3.2.5 Base de dados

Para armazenar as informações sobre os usuários, simulações e robôs foi utilizado o banco de dados relacional MySQL. O MySQL é seguro, fácil de usar e mais leve do que a maioria dos outros sistemas de gerenciamento de banco de dados, além de possibilitar que todos os tipos de dados necessários sejam armazenados. A Figura 10 mostra parte deste banco de dados.

Figura 10 - Banco de dados do LabVIR



Fonte: De autoria própria

A tabela user é responsável por armazenar as informações do usuário como: nome, email, senha, username, foto, data de cadastro, etc. As informações dos robôs virtuais são armazenadas na tabela robot. O nome, descrição, nível de dificuldade, linguagem e outras informações das simulações são salvas na tabela benchmark. Os dados dos desempenhos dos usuários nas simulações são armazenados na tabela achievement.

Várias dessas informações são utilizadas em scripts da aplicação web que são muito importantes para a simulação, como por exemplo: o download-project.php e o upload-file.php.

O download-project.php é responsável por retornar um arquivo compactado contendo os arquivos de simulação do Webots a serem executados. Já o upload-

file.php tem a função de carregar a nova versão do controlador quando o usuário a modifica e a salva do editor na interface web.

Esses dois scripts precisaram receber alguns parâmetros, como por exemplo: nome do projeto a ser carregado, nome do diretório onde o arquivo deve ser carregado, conteúdo do arquivo a ser armazenado, etc.

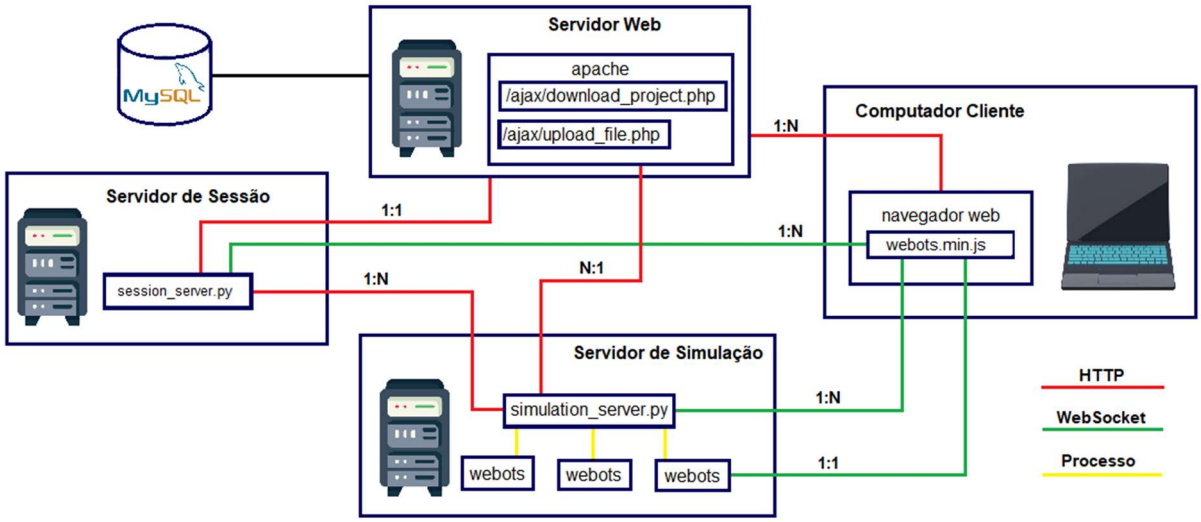
3.3 Arquitetura do sistema

Para que as simulações do LabVIR possam ser executadas, são necessários dois tipos de servidores: o servidor de sessão e o servidor de simulação. O servidor de sessão é o ponto de entrada para uma nova simulação, ele tem como objetivo gerenciar os servidores de simulação disponíveis, fazer um balanceamento de carga e, por fim, enviar a URL do servidor de simulação disponível para o cliente. O servidor de simulação tem como uma de suas tarefas criar e iniciar as instâncias do Webots com as simulações desejadas pelos usuários e enviar as URLs dos WebSocket do Webots aos usuários para que eles possam se comunicar diretamente com o Webots. Ambos os servidores são scripts escritos na linguagem Python (CYBERBOTICS, 2021).

É essencial que pelo menos um servidor de sessão seja executado e também um ou mais servidores de simulação. Os servidores de simulação precisam rodar em máquinas diferentes, enquanto o servidor de sessão é muito leve e pode rodar na mesma máquina que o servidor de simulação, lembrando que o Webots deve estar instalado em todas as máquinas em que um servidor de simulação está sendo executado (CYBERBOTICS, 2021).

A Figura 11 mostra a arquitetura do sistema e como é a comunicação entre os componentes.

Figura 11 - Arquitetura do LabVIR



Fonte: Adaptado de Cyberbotics (2021)

3.4 Protocolo de comunicação

Quando um cliente web desejar saber se pode iniciar uma simulação, primeiramente ele precisará abrir uma conexão com o servidor de sessão utilizando o protocolo WebSocket para verificar a disponibilidade dos servidores de simulação. Se algum servidor de simulação estiver disponível, então o servidor de sessão responderá '1', caso contrário, o servidor de sessão responderá '0'. Sempre que houver alterações na disponibilidade dos servidores de simulação, o servidor de sessão enviará uma nova notificação para o cliente web (CYBERBOTICS, 2021).

Quando um cliente web quiser iniciar uma simulação, primeiramente ele precisa fazer uma solicitação AJAX para o servidor de sessão. O servidor de sessão então enviará a URL do WebSocket de um servidor de simulação que esteja disponível ou, se nenhum estiver disponível, enviará um erro. Para iniciar a simulação, o cliente web irá se comunicar diretamente com o servidor de simulação (CYBERBOTICS, 2021).

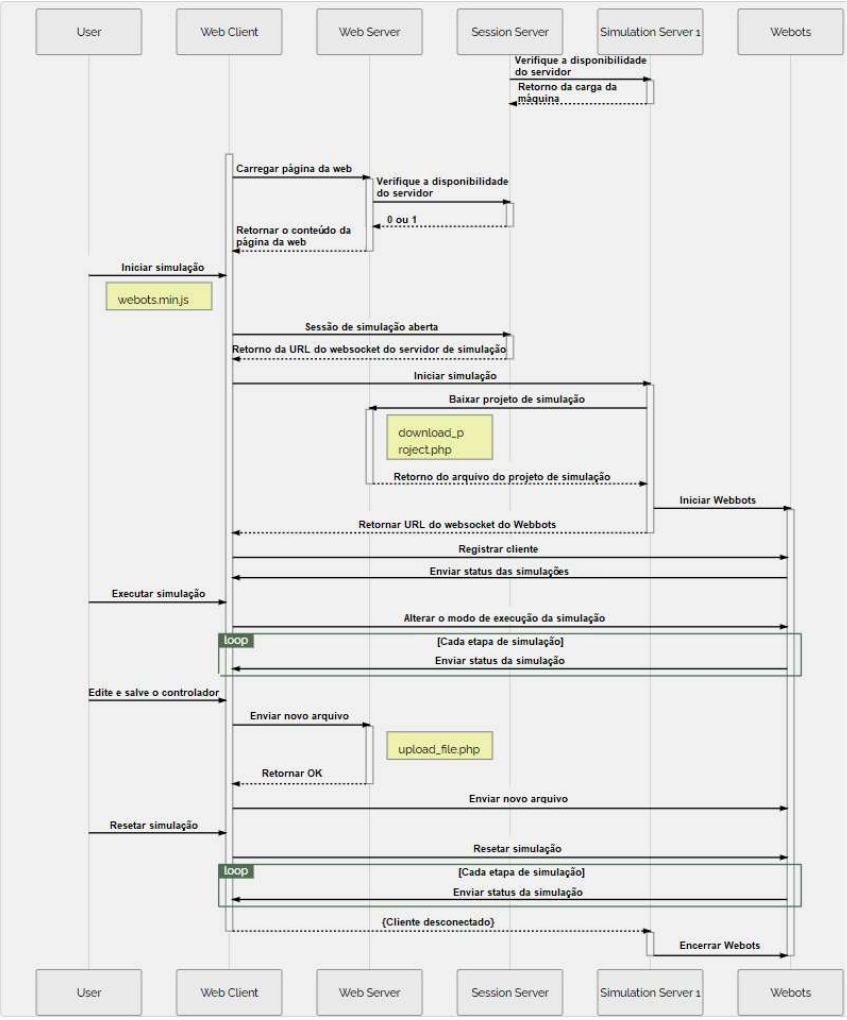
O servidor de simulação iniciará o Webots com a simulação solicitada pelo usuário. Os arquivos de projetos solicitados pelo cliente devem ser armazenados no host do site e solicitados pelo servidor de simulação através de uma requisição AJAX no endereço <host>/ajax/download_project.php. Em seguida, o servidor de simulação enviará a URL do WebSocket do Webots, para que o cliente possa se comunicar diretamente com o Webots. Se, por algum motivo, a conexão do WebSocket com o

servidor de simulação for suspensa ou fechada, então o servidor de simulação fechará o Webots, notificará o cliente e encerrará a conexão com ele (CYBERBOTICS, 2021).

O servidor de sessão está constantemente consultando os servidores de simulação para monitorar as suas cargas. Caso um servidor de simulação fique sobrecarregado, então o servidor de sessão não permitirá que nenhuma simulação seja iniciada nele (CYBERBOTICS, 2021).

A Figura 12 ilustra a sequência de tarefas realizada pelos servidores quando uma nova simulação é iniciada e quando o controlador é editado.

Figura 12 - Diagrama de sequência do protocolo da simulação web



Fonte: Adaptado de Cyberbotics (2021)

4 RESULTADOS E TESTES

Nessa seção é apresentada a interface do LabVIR, o robô e o ambiente virtual desenvolvido, a descrição do algoritmo, as interações com o usuário, os testes de sistema realizados e as limitações do projeto.

4.1 Robô e ambiente virtual

O ambiente virtual foi feito tendo como inspiração uma cidade, onde o trajeto são as ruas e as paredes são as casas, prédios, muros e cercas.

O piso da arena possui um fundo escuro e linhas claras que representam o caminho a ser seguido, como mostra a Figura 13. Essas linhas ficam a uma distância mínima das bordas da área de percurso e não é um trajeto perfeitamente reto, existindo diversos desafios que devem ser superados para que o robô possa continuar avançando. Esses desafios podem ser caminhos onde o trajeto não pode ser reconhecido, elementos desconhecidos que devem ser desviados, caminhos sem saída que precisam ser evitados.

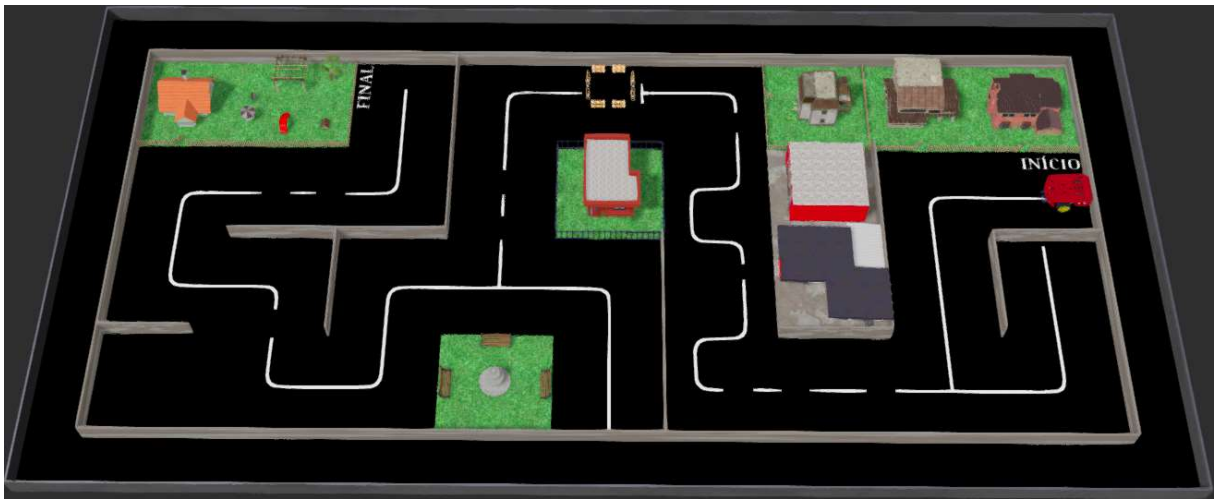
Dentro da área de percurso existe um obstáculo, que é uma barreira intransponível e que força o robô a desviar, saindo do caminho traçado pela linha branca durante alguns instantes. Ao desviar desse obstáculo, o robô deve retornar para a linha logo em seguida ao obstáculo desviado.

As encruzilhadas são caminhos de 3 ramificações e perpendiculares (90°), seguir pela direção errada na encruzilhada levará o robô a um beco sem saída.

Existem também Gaps (ou lacunas), que são “falhas” na linha do percurso onde o robô não consegue distinguir o caminho a ser seguido. Todos os Gaps são sempre em linhas retas.

Como já foi falado anteriormente, para verificar se o robô conseguiu completar o percurso, foi utilizado um tipo especial de robô chamado Supervisor. O controlador do supervisor escrito em Python para definir o comportamento do supervisor pode ser visualizado no Apêndice A.

Figura 13 - Ambiente virtual desenvolvido



Fonte: De autoria própria

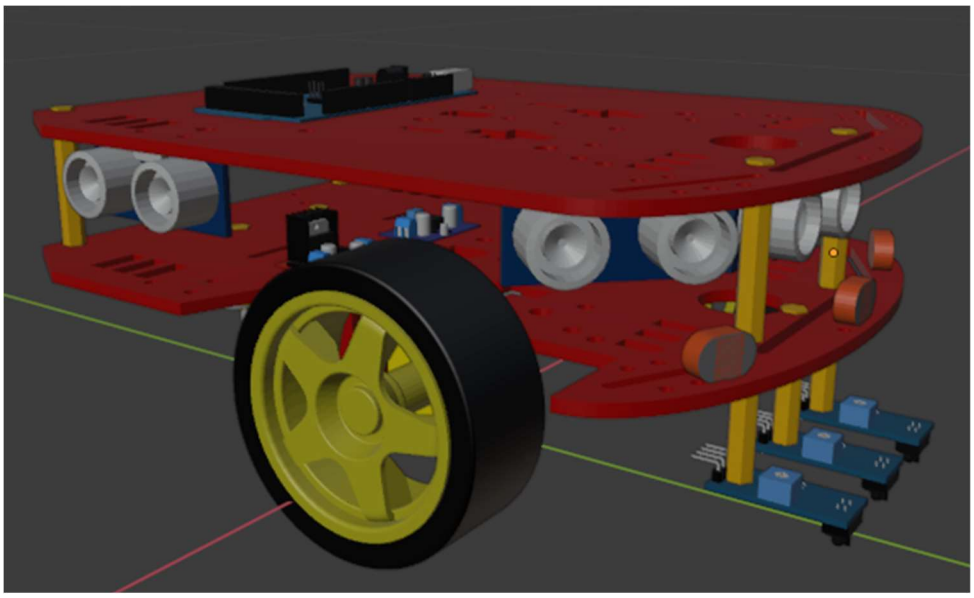
A Figura 14 mostra como ficou o robô virtual, ele possui três sensores de luminosidade na parte da frente, três sensores ultrassônicos também na frente e um em cada lateral e três sensores infravermelhos na parte inferior.

Para concluir o percurso desse ambiente virtual é necessário utilizar apenas os três sensores infravermelhos para detectar a linha e um dos sensores ultrassônicos que fica centralizado na parte frontal do robô para detectar o obstáculo na pista, mas uma grande vantagem desse veículo é que ele pode ser empregado em diversos outros ambientes que irão fazer uso dos demais sensores.

Para que o L1R2 possa fazer as curvas, ele utiliza como mecânica de locomoção o conceito de acionamento diferencial. Segundo Malu e Majumdar (2014), um robô é considerado de acionamento diferencial quando ele possui dois motores montados em posições fixas no seu lado esquerdo e direito, e cada um desses motores é acionado de forma independente. Para o robô ir para frente é necessário que ambos os motores sejam movidos na mesma velocidade e, para fazer uma curva para a esquerda, o motor direito precisa estar a uma velocidade maior do que o motor esquerdo e vice-versa para virar à direita. Também é possível fazer com que o robô gire em torno do seu próprio eixo, movendo ambas as rodas em direções opostas com a mesma velocidade. Para manter a estabilidade do robô é necessária uma terceira roda do tipo omnidirecional.

Em cima destas lógicas e conceitos foi elaborado o código do robô, constantemente fazendo a leitura dos sensores e enviando o comando correto para os motores.

Figura 14 - Robô virtual desenvolvido



Fonte: De autoria própria

4.2 Descrição do algoritmo

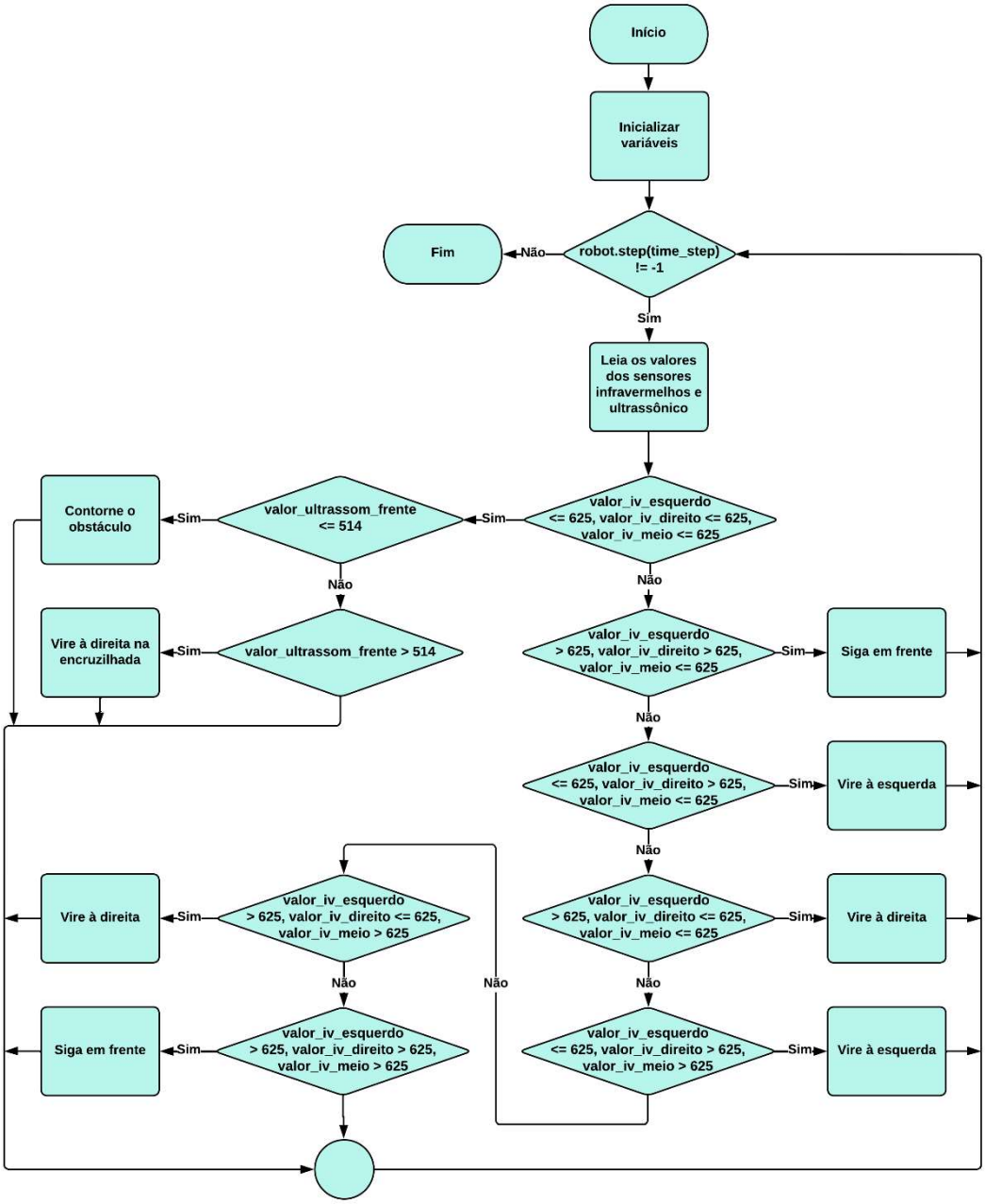
O algoritmo para manter o robô sobre o percurso é bastante simples. O objetivo principal é manter sempre o sensor infravermelho do meio sobre a linha, pois dessa maneira o robô estará centralizado no caminho que deve percorrer. Portanto, a lógica básica que deve ser implementada é:

1. Quando o sensor infravermelho do meio detectar a linha branca, o robô avançará.
2. Quando o sensor infravermelho esquerdo detectar a linha branca, o robô se moverá para a esquerda.
3. Quando o sensor infravermelho esquerdo e o sensor infravermelho do meio detectarem a linha branca, o robô se moverá para a esquerda.
4. Quando o sensor infravermelho direito detectar a linha branca, o robô se moverá para a direita.
5. Quando o sensor infravermelho direito e o sensor infravermelho do meio detectarem a linha branca, o robô se moverá para a direita.

6. Quando todos os sensores infravermelhos detectarem a linha branca, mas o sensor ultrassônico não detectar nenhum obstáculo, o robô primeiramente avançará brevemente para frente e depois irá fazer uma curva de 90 graus para a direita.
7. Quando todos os sensores infravermelhos detectarem a linha branca e o sensor ultrassônico detectar algum obstáculo, o robô irá fazer o desvio através da execução de movimentos como curvas 90 graus e avanços para frente.
8. Também existe uma sexta hipótese onde os três sensores infravermelhos estão fora da linha, nesse caso o robô continuará andando para frente até achar uma linha ou até bater na parede.

Como pode ser observado no fluxograma da Figura 15 ou lendo o código implementado em Python que está no Apêndice B, com o robô e o ambiente desenvolvido nesse projeto o aluno consegue praticar alguns conceitos de programação, como: variáveis, estruturas condicionais, estruturas de repetição, funções e operadores (aritméticos, lógicos e relacionais).

Figura 15 - Fluxograma do algoritmo desenvolvido para o robô



Fonte: De autoria própria

4.3 Interface do laboratório

A interface do LabVIR é mostrada na Figura 16. Além de exibir a simulação na parte central da tela, na parte inferior também existe uma barra de ferramentas com uma grande quantidade de opções onde o usuário pode: sair da simulação, abrir uma janela contendo informações introdutórias sobre o robô e a simulação, salvar o controlador e reiniciar a simulação, executar a simulação passo a passo, executar em

tempo real, pausar, visualizar o tempo atual da simulação, abrir o console e abrir uma janela de ajuda contendo a documentação da interface. Os códigos utilizados para o desenvolvimento da janela personalizada do robô se encontram no Apêndice C e no Apêndice D.

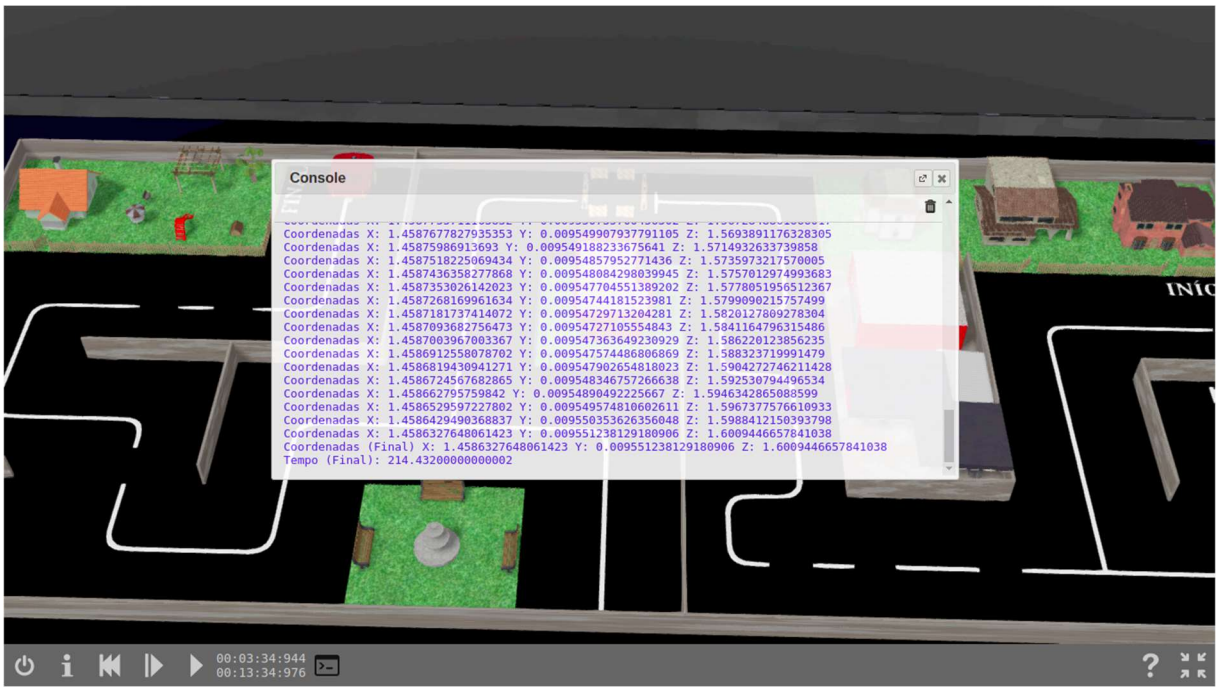
Figura 16 - Janela do robô e janela de ajuda

Fonte: De autoria própria

O console da simulação exibe informações de simulação, avisos e mensagens de erro (Figura 17). Essas mensagens podem vir do núcleo de simulação ou do controlador do robô.

A execução passo a passo e o console acabam sendo duas ferramentas muito importantes para os usuários porque facilitam a realização da depuração, que é o nome dado ao processo de encontrar e remover os erros que podem impedir que os códigos funcionem adequadamente.

Figura 17 - Janela do console

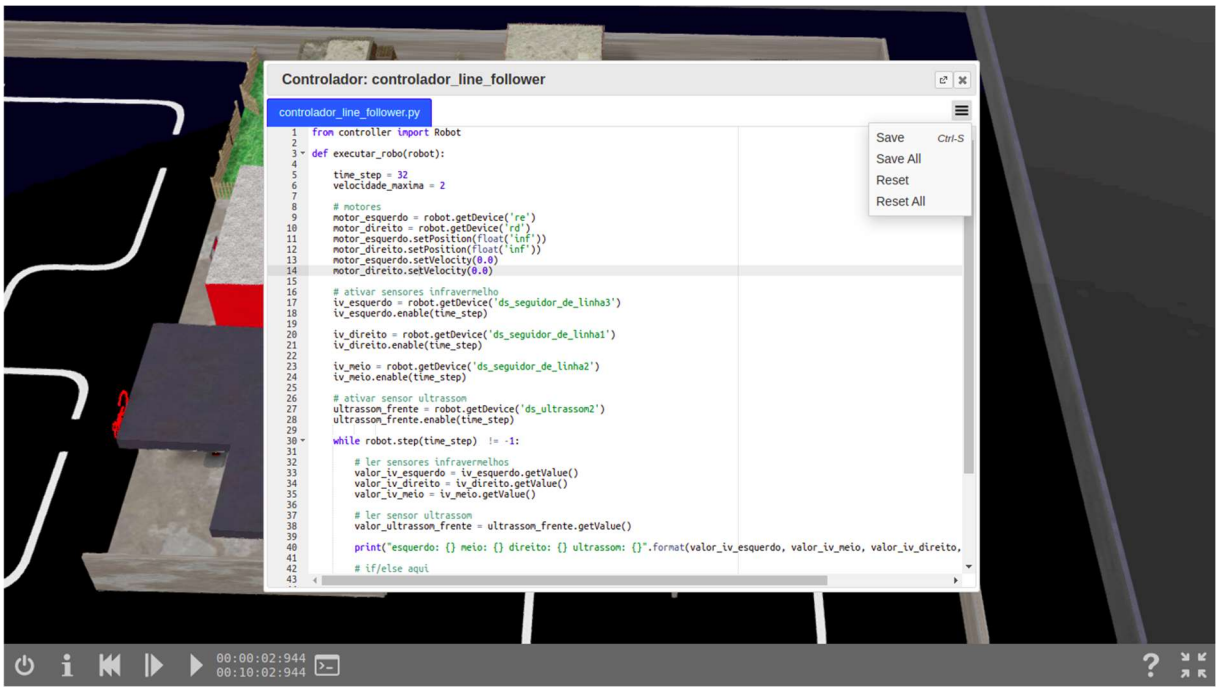


Fonte: De autoria própria

Além da barra de ferramentas, ao clicar com o botão direito do mouse em um objeto ou robô na simulação, o usuário terá acesso a um menu de contexto que fornece algumas ações sobre o objeto. A principal ação ao clicar em um robô é a opção de abrir a janela para editar o controlador. Nessa janela é possível editar o código do controlador do robô, modificando, assim, o seu comportamento (Figura 18).

Quando uma nova simulação é iniciada, o controlador do robô já vem com uma estrutura básica do código pronta. Isso é bem importante porque evita que o usuário tenha que se preocupar com a inicialização de algumas variáveis, importação das bibliotecas e as configurações iniciais do robô. Dessa forma, o usuário pode focar em implementar as partes mais importantes do programa.

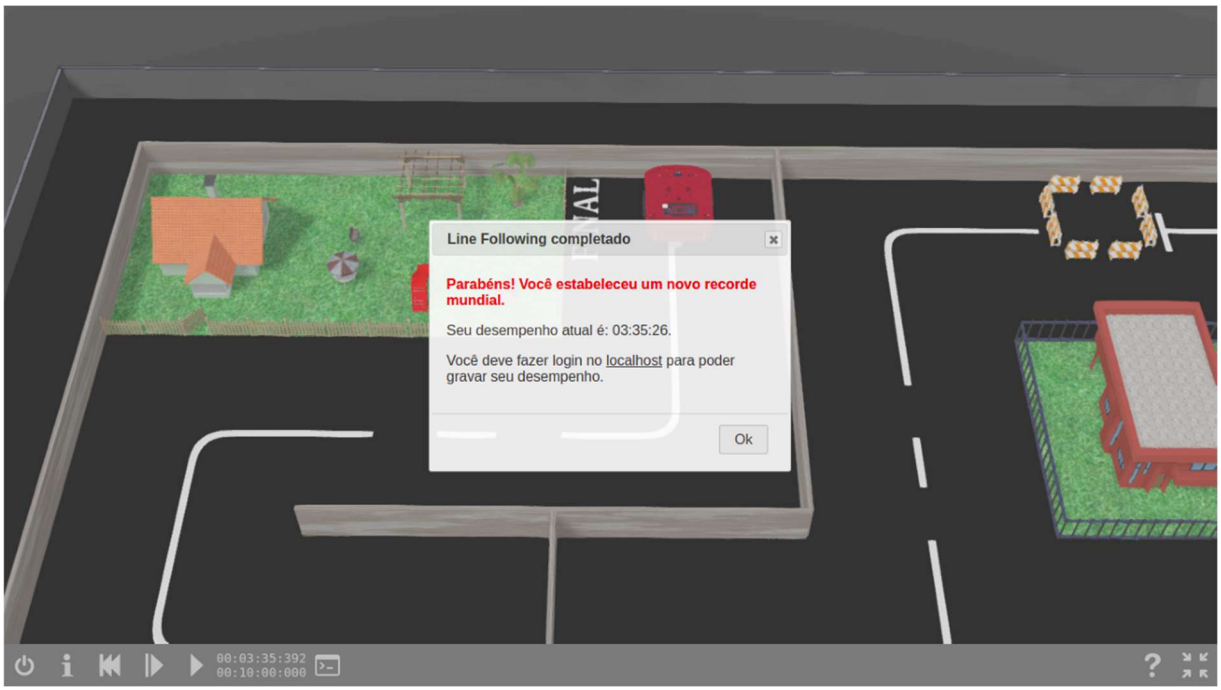
Figura 18 - Janela do controlador



Fonte: De autoria própria

Quando o usuário consegue completar a simulação, é exibida uma janela que informa o seu desempenho e também se ele conseguiu estabelecer um novo recorde pessoal ou mundial, além de possibilitar que o resultado da simulação seja salvo (Figura 19).

Figura 19 - Janela de simulação completa



Fonte: De autoria própria

4.4 Testes

Devido ao cronograma e os prazos estabelecidos para realização deste trabalho, não foi possível realizar testes com um grupo real de alunos para observar e monitorar as suas interações com o laboratório.

Todos os testes foram realizados de forma manual sem usar nenhuma ferramenta de automação.

Os testes com o robô foram realizados no próprio ambiente virtual desenvolvido durante o projeto. Nesses testes foi possível verificar a capacidade do robô em seguir a linha, detectar obstáculos, analisar encruzilhadas, escolher o caminho correto e atingir a posição final.

O console da simulação foi usado em alguns dos testes para imprimir a posição do robô ou os valores lidos pelos três sensores infravermelhos e pelo sensor ultrassônico, assim como a velocidade das rodas, durante a execução de um determinado trecho do algoritmo.

A velocidade padrão dos motores utilizada nos testes é 2 radianos por segundo [rad/s], independente do sentido de rotação. Os sensores infravermelhos sempre retornavam valores que variaram abaixo de 625 na superfície branca e acima de 625

na superfície preta. Já o sensor ultrassônico retorna um valor máximo de 1000 (equivalente a 10 cm).

4.4.1 Caso de teste 1: Robô segue em linha reta

O primeiro caso de teste teve como objetivo verificar se o robô estava se movendo para frente quando apenas o sensor infravermelho do meio estivesse sobre a linha.

Para isso, o robô foi posicionado de forma que apenas o sensor infravermelho central ficou sobre a superfície branca e sem nenhum obstáculo na frente do veículo. Sendo assim, somente esse sensor do meio deveria retornar valores menores do que 625, o sensor ultrassônico sempre manter o valor máximo (1000) e o robô teria que andar em linha reta.

Como pode ser visto na Figura 20, o resultado do teste foi o esperado. Somente o sensor do meio detectou a linha branca, nenhum obstáculo foi detectado pelo sensor ultrassônico e o algoritmo definiu as mesmas velocidades positivas para os dois motores (2 rad/s), fazendo com que o robô se movesse para frente.

Trecho de código:

```
elif (valor_iv_esquerdo > 625) and (valor_iv_direito > 625) and (valor_iv_meio <= 625):
    motor_esquerdo.setVelocity(velocidade_maxima)
    motor_direito.setVelocity(velocidade_maxima)
```

Figura 20 - Saída do console no caso de teste 1

```
Infravermelho Esquerdo: 996.007453472852 Meio: 606.2067110600054 Direito: 1006.7756990972363 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: 2.0 Motor Direito: 2.0
Infravermelho Esquerdo: 997.7560467211781 Meio: 598.8230332061377 Direito: 1019.138639540768 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: 2.0 Motor Direito: 2.0
Infravermelho Esquerdo: 998.2364457050381 Meio: 603.4692303554319 Direito: 998.0979561379214 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: 2.0 Motor Direito: 2.0
```

Fonte: De autoria própria

4.4.2 Caso de teste 2: Curva à esquerda

No segundo caso foram realizados testes para verificar os dois cenários onde o robô deve fazer uma curva para a esquerda. No primeiro cenário, o robô foi colocado em uma posição em que apenas o infravermelho esquerdo ficou sobre a linha branca e sem nenhum obstáculo na frente do robô. Dessa forma, somente o sensor esquerdo deveria retornar valores menores do que 625, o sensor ultrassônico sempre retornar o valor máximo (1000) e o veículo virar para esquerda.

Como mostrado na figura 21, o primeiro cenário teve o resultado esperado. Somente o sensor esquerdo detectou a linha branca e nenhum obstáculo foi detectado pelo sensor ultrassônico. Sendo assim, a velocidade do motor esquerdo foi definida como -2 rad/s, o motor direito como 2 rad/s e como consequência o robô realizou uma curva para a esquerda.

Trecho de código:

```
elif (valor_iv_esquerdo <= 625) and (valor_iv_direito > 625) and (valor_iv_meio > 625):
    motor_esquerdo.setVelocity(-velocidade_maxima)
    motor_direito.setVelocity(velocidade_maxima)
```

Figura 21 - Saída do console no primeiro cenário do caso de teste 2

```
Infravermelho Esquerdo: 605.4843324943581 Meio: 999.6076618601921 Direito: 993.8906674877685 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: -2.0 Motor Direito: 2.0
Infravermelho Esquerdo: 608.9747222464613 Meio: 1010.4191800915479 Direito: 993.6685464504711 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: -2.0 Motor Direito: 2.0
Infravermelho Esquerdo: 601.9507875089095 Meio: 995.9900978644066 Direito: 992.6287678648939 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: -2.0 Motor Direito: 2.0
```

Fonte: De autoria própria

No segundo cenário o robô foi posicionado de modo que, além do sensor infravermelho esquerdo, o sensor do meio também estava sobre a linha branca. Sendo assim, ambos os sensores infravermelhos deveriam apresentar valores menores que 625, o sensor ultrassônico o valor 1000 e o veículo virar para a esquerda.

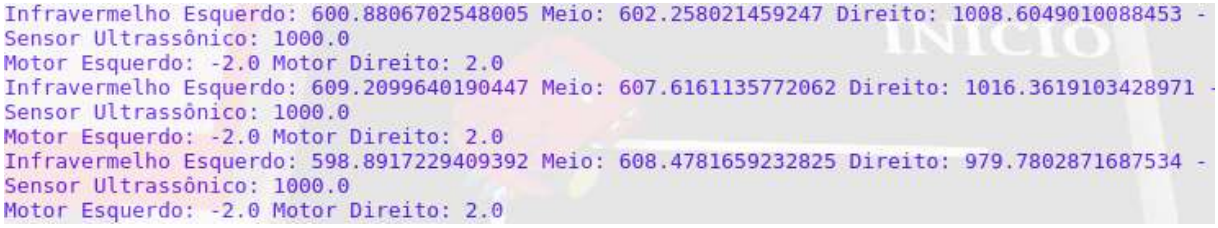
O segundo cenário também foi bem sucedido (Figura 22). Os sensores do meio e esquerdo detectaram a linha branca, nenhum obstáculo foi detectado pelo sensor ultrassônico e, assim como no primeiro cenário, a velocidade do motor esquerdo foi

definida como -2 rad/s, o motor direito como 2 rad/s e o robô realizou a curva para a esquerda.

Trecho de código:

```
elif (valor_iv_esquerdo <= 625) and (valor_iv_direito > 625) and (valor_iv_meio <= 625):
    motor_esquerdo.setVelocity(-velocidade_maxima)
    motor_direito.setVelocity(velocidade_maxima)
```

Figura 22 - Saída do console no segundo cenário do caso de teste 2



Fonte: De autoria própria

4.4.3 Caso de teste 3: Curva à direita

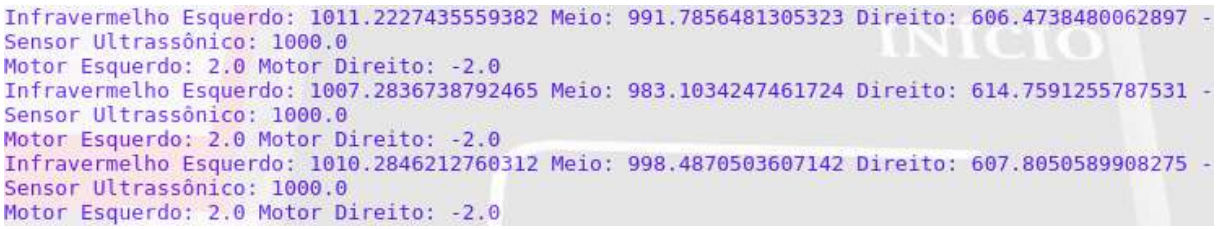
Esse caso de teste é muito parecido com o anterior, e a finalidade foi verificar os dois cenários onde o robô deve fazer uma curva para a direita. No primeiro cenário, o robô foi posicionado para que apenas o infravermelho direito ficasse sobre a linha branca e sem nenhum obstáculo. Dessa forma, apenas o sensor direito deveria retornar valores menores do que 625, o sensor ultrassônico sempre retornar o valor máximo (1000) e o veículo virar para direita.

O teste foi bem sucedido, como mostra a Figura 23. Somente o sensor direito detectou a linha branca, nenhum obstáculo foi detectado pelo sensor ultrassônico e as velocidades dos motores esquerdo e direito foram definidas como 2 rad/s e -2 rad/s, respectivamente. Como resultado, o robô realizou uma curva para a direita.

Trecho de código:

```
elif (valor_iv_esquerdo > 625) and (valor_iv_direito <= 625) and (valor_iv_meio > 625):
    motor_esquerdo.setVelocity(velocidade_maxima)
    motor_direito.setVelocity(-velocidade_maxima)
```

Figura 23 - Saída do console no primeiro cenário do caso de teste 3



Fonte: De autoria própria

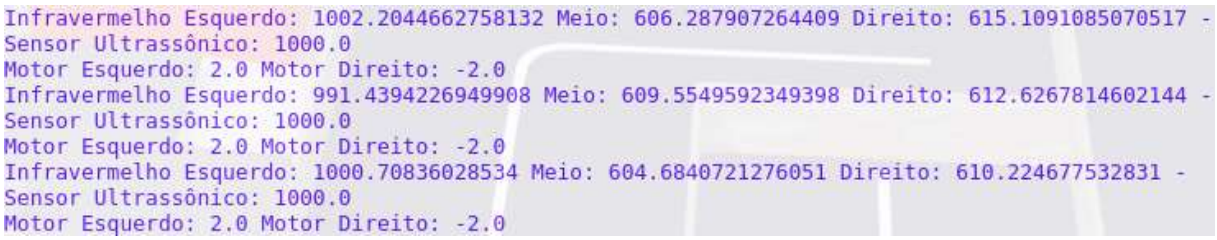
No segundo cenário o robô foi posicionado de modo que, além do sensor infravermelho direito, o sensor do meio também estava sobre a linha branca. Sendo assim, ambos os sensores infravermelhos deveriam apresentar valores menores que 625, o sensor ultrassônico o valor 1000 e o veículo virar para a direita.

O segundo cenário também foi bem sucedido (Figura 24). Os sensores do meio e direito detectaram a linha branca, nenhum obstáculo foi detectado pelo sensor ultrassônico e, assim como no primeiro cenário, a velocidade do motor direito foi definida como -2 rad/s, o motor esquerdo como 2 rad/s e o robô realizou a curva para a direita.

Trecho de código:

```
elif (valor_iv_esquerdo > 625) and (valor_iv_direito <= 625) and (valor_iv_meio <= 625):
    motor_esquerdo.setVelocity(velocidade_maxima)
    motor_direito.setVelocity(-velocidade_maxima)
```

Figura 24 - Saída do console no segundo cenário do caso de teste 3



Fonte: De autoria própria

4.4.4 Caso de teste 4: Encruzilhada

Esse caso de teste teve como objetivo verificar o comportamento do robô ao encontrar uma encruzilhada. Nesse caso de teste, o robô foi posicionado de forma que os três sensores infravermelhos ficassem sobre a linha, mas sem nenhum obstáculo na frente do veículo. Dessa forma, todos os três sensores infravermelhos deveriam detectar a superfície branca e retornarem valores menores que 625, o sensor ultrassônico não poderia detectar nenhum obstáculo e apresentar o valor 1000 e o robô deveria avançar para frente durante 1.3 segundos e depois girar para a direita durante 1.8 segundos.

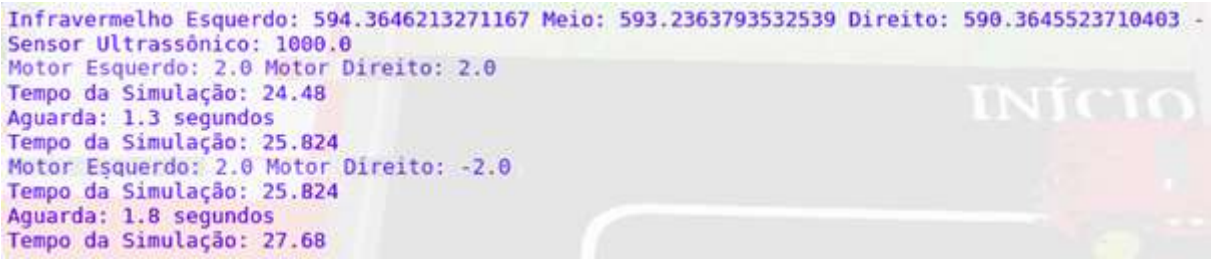
O funcao_delay é justamente a função responsável por fazer o robô realizar o comando definido por um determinado tempo. Esse tempo que robô avançou para frente e depois girou foram definidos manualmente levando em consideração as velocidades dos motores.

Como pode ser visto na figura 25, o teste foi bem sucedido. Todos os três sensores detectaram a linha branca, o sensor ultrassônico não detectou nenhum obstáculo e o robô realizou os movimentos corretos.

Trecho de código:

```
if(580 < valor_iv_esquerdo < 625) and (580 < valor_iv_direito < 625) and (580 < valor_iv_meio < 625):
    .
    .
    .
    else:
        motor_esquerdo.setVelocity(velocidade_maxima)
        motor_direito.setVelocity(velocidade_maxima)
        funcao_delay(robot, 1.3)

        motor_esquerdo.setVelocity(velocidade_maxima)
        motor_direito.setVelocity(-velocidade_maxima)
        funcao_delay(robot, 1.8)
```

Figura 25 - Saída do console no caso de teste 4



Fonte: De autoria própria

4.4.5 Caso de teste 5: Desvio de obstáculo

Esse caso de teste teve como objetivo verificar o comportamento do robô ao encontrar um obstáculo. Nesse caso de testes o robô foi posicionado de modo que os três sensores infravermelhos detectassem a linha e com um obstáculo posicionado a uma distância de 5,14 cm na frente do veículo. Dessa forma, todos os três sensores infravermelhos deveriam detectar a superfície branca e retornarem valores menores que 625, o sensor ultrassônico detectar o obstáculo e apresentar valores inferiores a 514 e o robô então realizar uma sequência de comandos para desviar do obstáculo.

Como pode ser visto na figura 26, o teste teve o resultado esperado. Todos os três sensores detectaram a linha branca, o sensor ultrassônico detectou o obstáculo e o robô realizou os movimentos corretos.

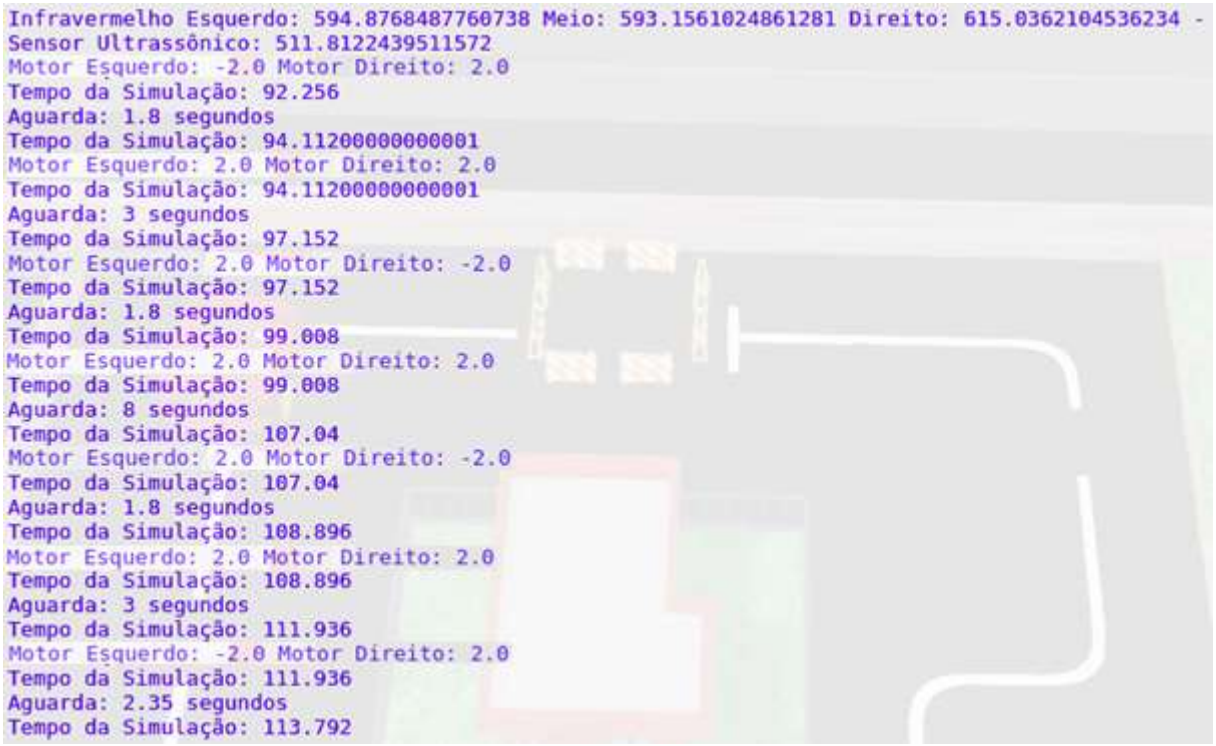
Para esse projeto não foi implementado nenhum algoritmo complexo desvio de obstáculos, que seja capaz de fazer o robô navegar em segurança em ambientes dinâmicos.

Trecho de código:

```
if(580 < valor_iv_esquerdo < 625) and (580 < valor_iv_direito < 625) and (580 < valor_iv_meio < 625):

    if(valor_ultrassom_frente < 514):
        motor_esquerdo.setVelocity(-velocidade_maxima)
        motor_direito.setVelocity(velocidade_maxima)
        funcao_delay(robot, 1.8)
```

```
motor_esquerdo.setVelocity(velocidade_maxima)
motor_direito.setVelocity(velocidade_maxima)
funcao_delay(robot, 3)

motor_esquerdo.setVelocity(velocidade_maxima)
motor_direito.setVelocity(-velocidade_maxima)
funcao_delay(robot, 1.8)

motor_esquerdo.setVelocity(velocidade_maxima)
motor_direito.setVelocity(velocidade_maxima)
funcao_delay(robot, 8)

motor_esquerdo.setVelocity(velocidade_maxima)
motor_direito.setVelocity(-velocidade_maxima)
funcao_delay(robot, 1.8)

motor_esquerdo.setVelocity(velocidade_maxima)
motor_direito.setVelocity(velocidade_maxima)
funcao_delay(robot, 3)

motor_esquerdo.setVelocity(-velocidade_maxima)
motor_direito.setVelocity(velocidade_maxima)
funcao_delay(robot, 2.35)
```

Figura 26 - Saída do console no caso de teste 5



Fonte: De autoria própria

4.4.6 Caso de teste 6: Falha na linha do percurso

Esse teste teve como objetivo verificar se o robô continuaria se movendo para frente quando nenhum dos sensores infravermelhos estivessem sobre o trajeto.

Nesse caso de testes foi colocado em uma posição onde nenhum dos sensores infravermelhos estavam sobre a linha e sem nenhum obstáculo na frente do veículo. Dessa forma, nenhum dos três sensores infravermelhos deveriam retornar valores menores que 625, o sensor ultrassônico não poderia detectar nenhum obstáculo e apresentar o valor 1000 e o robô deveria andar em linha reta.

O teste foi executado com êxito, como mostra a Figura 27. Nenhum sensor infravermelho retornou valores incorretos e nenhum obstáculo foi detectado. Sendo assim, os dois motores foram configurados com a mesma velocidade (2 rad/s) e o robô andou para frente.

Por ser um simulador, mesmo que os motores estejam na mesma velocidade o robô não andará perfeitamente em linha reta, por conta do atrito das rodas com o chão e outros fatores.

Trecho de código:

```
elif (valor_iv_esquerdo > 625) and (valor_iv_direito > 625) and (valor_iv_meio > 625):
    motor_esquerdo.setVelocity(velocidade_maxima)
    motor_direito.setVelocity(velocidade_maxima)
```

Figura 27 - Saída do console no caso de teste 6

```
Infravermelho Esquerdo: 997.2763576905536 Meio: 1002.3297294237415 Direito: 986.4794984395545 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: 2.0 Motor Direito: 2.0
Infravermelho Esquerdo: 998.982010066999 Meio: 991.5125851574657 Direito: 991.591215908421 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: 2.0 Motor Direito: 2.0
Infravermelho Esquerdo: 989.6473605630247 Meio: 1007.9712884185168 Direito: 995.3089406011163 -
Sensor Ultrassônico: 1000.0
Motor Esquerdo: 2.0 Motor Direito: 2.0
```

Fonte: De autoria própria

4.4.7 Caso de teste 7: Edição do controlador

Esse teste teve como objetivo assegurar que a opção de editar o controlado do robô estava funcionando corretamente. Com uma nova simulação inicializada, primeiramente foi verificado se era possível abrir a janela de edição do controlador através do menu de contexto do robô. Como esperado, a janela abriu normalmente e exibiu o código que vem por padrão no robô.

Logo depois, o controlador foi modificado, o novo código foi salvo e foi selecionada a opção de reiniciar a simulação. Como resultado, o comportamento do robô foi alterado, ele executou perfeitamente as instruções que foram programadas e nenhum erro foi exibido no console da simulação.

4.4.8 Caso de teste 8: Percurso completado

Esse teste teve como objetivo verificar se o supervisor estava funcionando corretamente e se ele estava detectando corretamente quando o robô finalizava o percurso. No controlador do supervisor foi definido que a simulação seria considerada completada somente quando o robô chegasse no ponto final trajeto, que fica entre as seguintes coordenadas do ambiente virtual:

X: Entre 1.26 e 1.75

Y: Não é levado em consideração

Z: Entre 1.60 e 1.75

Nesse teste o robô foi posicionado inicialmente no começo do trajeto e o console foi utilizado para exibir os seguintes valores obtidos pelo supervisor durante a simulação: posição do robô durante o trajeto, a posição final do robô e o tempo que ele levou para completar.

Como pode ser observado na Figura 28, o supervisor identificou que o robô completou a simulação quando a sua posição estava entre as coordenadas que foram estabelecidas no controlador. Sendo assim, tudo aconteceu como esperado.

Logo após o robô chegar no destino, foi exibida a janela que informava o desempenho.

O controlador do supervisor pode ser visualizado no Apêndice A.

Figura 28 - Saída do console no caso de teste 8

```
Coordenadas X: 1.4586529597227802 Z: 1.5967377576610933
Coordenadas X: 1.4586429490368837 Z: 1.5988412150393798
Coordenadas X: 1.4586327648061423 Z: 1.6009446657841038
Coordenadas (Final) X: 1.4586327648061423 Z: 1.6009446657841038
Tempo (Final): 214.43200000000002
```

Fonte: De autoria própria

4.5 Limitações do projeto

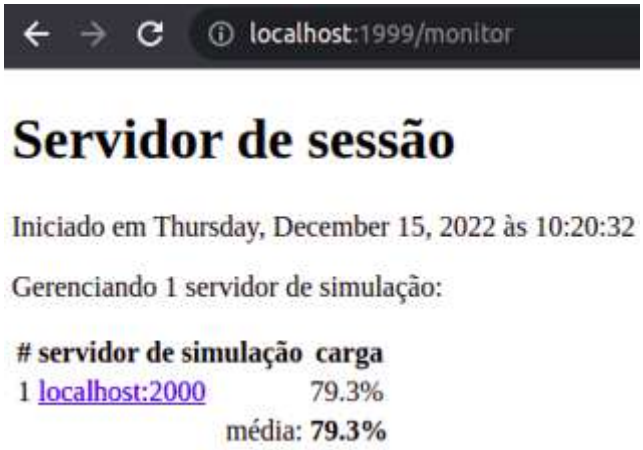
Apesar de ter sido criada a tabela User no banco de dados do projeto, infelizmente não foi possível implementar o sistema de login. O principal motivo é porque quando o login é realizado está acorrendo algum problema e a simulação não é carregada. Este problema provavelmente está relacionado com algum arquivo do servidor web ou até mesmo com o próprio webots.min.js, mas até a conclusão desse trabalho não foi possível solucioná-lo. Por conta disso, também não está sendo possível salvar os desempenhos dos usuários.

O computador que foi utilizado durante o projeto é um notebook Dell Inspiron com processador Intel Core i3 6006U de 6ª Geração e 2.0GHz, 4 GB DDR4 de memória RAM, placa de vídeo integrada Intel HD Graphics 520 e sistema operacional Ubuntu 20.04 LTS. Nele foi executado o servidor da aplicação web, o servidor de sessão e o servidor de simulação.

Por ser um hardware relativamente simples, foi possível criar apenas uma instância do Webots com o projeto antes que o servidor de simulação atingisse quase 80% de sua carga, como pode ser visto no monitor do servidor de sessão (Figura 29).

Quanto mais complexo for o ambiente virtual criado, maior será o consumo de recursos como CPU, GPU, memória e rede para executar uma instância do projeto no servidor de simulação, pois cada elemento diferente inserido no ambiente faz com que sejam adicionadas uma série de texturas na pasta do projeto.

Figura 28 - Monitor do servidor de sessão



Fonte: De autoria própria

Apesar de possibilitar que as instituições de ensino reduzam despesas com a compra de robôs reais e com a montagem de laboratórios físicos, esse laboratório virtual também possui uma certa despesa, já que necessita da aquisição de máquinas para o servidor de simulação que atendam aos requisitos do sistema ou do aluguel de instâncias de computadores virtuais baseados em nuvem, como o serviço Amazon WorkSpaces.

Mesmo com o custo, o laboratório virtual ainda vale muito a pena quando se leva em consideração a possibilidade de aumentar o número de alunos que podem acessar o laboratório de robótica, aumentar a flexibilidade de horários, ter acesso remoto, acabar com a limitação de tempo para uso, atualizar o laboratório sem a necessidade de aquisição de novos equipamentos e simular diferentes modelos de robôs nos mais variados tipos de ambientes.

Se quiser aumentar o número de instâncias de Webots que podem ser executadas em paralelo em um servidor de simulação, pode ser uma boa ideia

diminuir as configurações gráficas. Diminuindo as configurações gráficas (que estão presentes na aba Preferences/OpenGL do Webots), é possível multiplicar o número de instâncias paralelas por 3 ou 4 e resolver eventuais problemas com gargalo na GPU.

5 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho teve como objetivo principal o desenvolvimento de um laboratório virtual de robótica para ensino de conceitos básicos de algoritmos e programação para iniciantes. Para tanto foi necessário modelar e implementar o robô L1R2 virtual e o ambiente virtual para o robô, nesse processo foram utilizadas as ferramentas Blender, Tinkercad e Webots. Além disso, também foram feitas as configurações da interface de usuário e dos servidores (seção e simulação) para executar o web service.

Inicialmente foram apresentados o referencial teórico, alguns trabalhos similares e uma análise dos principais simuladores de robótica que poderiam ser utilizados no laboratório virtual. Dentre os simuladores citados, o Webots é o que mais se adequava ao problema. Sendo assim, ele foi a principal ferramenta utilizada para o desenvolvimento do Laboratório Virtual Interativo de Robótica (LabVIR). Infelizmente, por causa de um problema nos arquivos do servidor web, não foi possível implementar o sistema de login, mas todo o restante do projeto funcionou como esperado. Com base nos resultados apresentados e nos testes realizados, constata-se que o presente trabalho cumpriu os objetivos propostos.

Acredita-se na relevância desse trabalho ao possibilitar que mais alunos possam ter acesso a um laboratório de robótica e também estimular os alunos iniciantes a aprenderem sobre os conceitos introdutórios de algoritmos e programação.

Trabalhos futuros terão como objetivo implementar o sistema de login do LabVIR e, dessa forma, possibilitar que os desempenhos dos usuários possam ser salvos. Além disso, um outro trabalho é disponibilizá-lo em um Ambiente Virtual de Aprendizagem (AVA) acessado via Web, dessa forma, será possível comprovar a sua eficiência através de testes em instituições de ensino com grupos de alunos reais. O laboratório também pode ser aprimorado com a adição de novos robôs e de novos ambientes que irão possibilitar aos estudantes aprender sobre diferentes tipos de sensores, atuadores e aperfeiçoar a sua lógica de programação. Outra possível melhoria é a adição de recursos colaborativos para uma maior interação entre os alunos e professores no laboratório virtual.

REFERÊNCIAS

AFZAL, A; KATZ, D. S.; GOUES, C. L.; TIMPERLEY, C. S. A Study on the Challenges of Using Robotics Simulators for Testing. **arXiv preprint arXiv:2004.07368**, *[S. l.],* p. 1-8, 15 abr. 2020. DOI: https://doi.org/10.48550/arXiv.2004.07368. Disponível em: https://arxiv.org/abs/2004.07368. Acesso em: 21 jul. 2022.

ALKHALDI, T.; PRANATA, I; ATHAUDA, R. I. A review of contemporary virtual and remote laboratory implementations: observations and findings. **Journal of Computers in Education**, Pequim, v. 3, n. 3, p. 329-351, 29 jun. 2016. DOI: https://doi.org/10.1007/s40692-016-0068-z. Disponível em: https://link.springer.com/article/10.1007/s40692-016-0068-z. Acesso em: 15 jul. 2022.

ANDALUZ, V. H.; CHICAIZA, F. A.; GALLARDO, C.; QUEVEDO, W. X.; VARELA, J.; SÁNCHEZ, J. S.; ARTEAGA, O. Unity3D-MatLab Simulator in Real Time for Robotics Applications. *In*: INTERNATIONAL CONFERENCE ON AUGMENTED REALITY, VIRTUAL REALITY AND COMPUTER, 3., 2016, Lecce. **Proceedings** [...]. Cham: Springer, 2016. p. 246 – 263. DOI: https://doi.org/10.1007/978-3-319-40621-3_19. Disponível em: https://link.springer.com/chapter/10.1007/978-3-319-40621-3_19. Acesso em: 7 abr. 2022.

BLENDER. **About**. *[S. l.]*, 2021. Disponível em: https://www.blender.org/about/. Acesso em: 17 jun. 2022.

BUDAI, T.; KUCZMANN, M. Towards a Modern, Integrated Virtual Laboratory System. **Acta Polytechnica Hungarica**, Gyor, v. 15, n. 3, p. 191-204, mar. 2018. Disponível em:http://epa.niif.hu/02400/02461/00080/pdf/EPA02461_acta_polytechnica_hungarica_2018_03_191-204.pdf. Acesso em: 21 abr. 2022.

CHAOS, D.; CHACÓN, J.; LOPEZ-OROZCO, J. A.; Dormido, S. Virtual and Remote Robotic Laboratory Using EJS, MATLAB and LabVIEW. **Sensors**, Madrid, v. 13, n. 2, p. 2595-2612, fev. 2013. DOI: https://doi.org/10.3390/s130202595. Disponível em: https://www.mdpi.com/1424-8220/13/2/2595. Acesso em: 3 mar. 2022.

COLLINS, J.; CHAND, S.; VANDERKOP, A.; HOWARD, D. A Review of Physics Simulators for Robotic Applications. **IEEE Access**, *[S. l.]*, v. 9, p. 51416-51431, 2021. DOI: https://doi.org/10.1109/ACCESS.2021.3068769. Disponível em: https://ieeexplore.ieee.org/document/9386154. Acesso em: 03 marc. 2023.

COPPELIASIM. **Coppelia Robotics**. *[S. l.]*, 2019. Disponível em: https://www.coppeliarobotics.com/. Acesso em: 3 ago. 2022.

COPPELIASIM. **Features**. *[S. l.]*, 2020. Disponível em: https://www.coppeliarobotics.com/. Acesso em: 3 ago. 2022.

CYBERBOTICS. **Web Simulation**. *[S. l.]*, 2021. Disponível em: https://www.cyberbotics.com/doc/guide/web-simulation?version=R2021a. Acesso em: 13 set. 2022.

CYBERBOTICS. **Webots: robot simulator**. *[S. l.]*, 2020. Disponível em: https://cyberbotics.com/. Acesso em: 4 ago. 2022.

GAZEBO. **Página inicial**. *[S. l.]*, 2014. Disponível em: https://classic.gazebosim.org/. Acesso em: 1 ago. 2022.

HAKIMZADEH, H.; ADAIKKALAVAN, R.; BATZINGER, R. Successful implementation of an active learning laboratory in computer science. *In*: ACM SIGUCCS CONFERENCE ON USER SERVICES, 39., 2011, San Diego. **Proceedings** [...]. Nova Iorque: Association for Computing Machinery, 2011. p. 83 – 86. DOI: https://doi.org/10.1145/2070364.2070386. Disponível em: https://dl.acm.org/doi/abs/10.1145/2070364.2070386. Acesso em: 25 marc. 2022.

HARRIS, A.; CONRAD, J. M. Survey of popular robotics simulators, frameworks, and toolkits. *In*: IEEE SOUTHEASTCON, 25., 2011, Nashville. **Proceedings** [...]. Piscataway: IEEE, 2011. p. 243 – 249. DOI: https://doi.org/ 10.1109/SECON.2011.5752942. Disponível em: https://ieeexplore.ieee.org/abstract/document/5752942. Acesso em: 11 jul. 2022.

HORITA, F. E. A.; NETO, V. V. G.; SANTOS, R. P. Design Science Research em Sistemas de Informação e Engenharia de Software: Conceitos, Aplicações e Trabalhos Futuros. *In*: JORNADA LATINO-AMERICANA DE ATUALIZAÇÃO EM INFORMÁTICA, 1., 2018, São Paulo. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2018. p. 191 – 210. Disponível em: https://sol.sbc.org.br/index.php/jolai_clei/issue/view/283. Acesso em: 4 jun. 2022.

HUSSEIN, A.; GARCÍA, F.; OLAVERRI-MONREAL, C. ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation. *In*: IEEE INTERNATIONAL CONFERENCE ON VEHICULAR ELECTRONICS AND SAFETY, 20., 2018, Madrid. **Proceedings** [...]. Piscataway: IEEE, 2018. p. 1 – 6. DOI: https://doi.org/10.1109/ICVES.2018.8519522. Disponível em: https://ieeexplore.ieee.org/abstract/document/8519522. Acesso em: 17 out. 2022.

IKUHARA, L. **Gazebo: o Poder das Simulações.** *[S. l.]*, 2020. Disponível em: https://www.ufrjnautilus.com/post/gazebo-o-poder-das-simula%C3%A7%C3%B5es-1?lang=pt. Acesso em: 1 dez. 2022.

JARA, C. A.; CANDELAS, F. A.; PUENTE, S. T.; TORRES, F. Hands-on experiences of undergraduate students in Automatics and Robotics using a virtual and remote laboratory. **Computers & Education**, San Vicente del Raspeig, v. 57, n. 4, p. 2451-2461, dez. 2011. DOI: https://doi.org/10.1016/j.compedu.2011.07.003. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0360131511001515. Acesso em: 23 mai. 2022.

JARA, C. A.; CANDELAS, F. A.; TORRES, F. Virtual and remote laboratory for robotics e-learning. *In*: EUROPEAN SYMPOSIUM ON COMPUTER AIDED PROCESS ENGINEERING, 18., 2008, Lyon. **Proceedings** [...]. Amsterdã: Elsevier, 2008. p. 1193 – 1198. DOI: https://doi.org/10.1016/S1570-7946(08)80205-2. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1570794608802052. Acesso em: 24 mai. 2022.

KONRAD, A. **Simulation of Mobile Robots with Unity and ROS: A Case-Study and a Comparison with Gazebo.** 2019. 56 f. Dissertação (Mestrado em Ciência) - University West, Trollhättan, 2019.

KUMAR, K.; REEL, P. S. Analysis of contemporary robotics simulators**.** *In*: INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN ELECTRICAL AND COMPUTER TECHNOLOGY, 1., 2011, Nagercoil. **Proceedings** [...]. Piscataway: IEEE, 2011. p. 661-665. DOI: https://doi.org/10.1109/ICETECT.2011.5760200. Disponível em: https://ieeexplore.ieee.org/abstract/document/5760200. Acesso em: 4 out. 2022.

LARA. **Página inicial**. *[S. l.]*, 2019. Disponível em: http://lara.uesb.br/. Acesso em: 13 set. 2021.

LACERDA, D. P.; DRESCH, A.; PROENÇA, A.; ANTUNES JÚNIOR, J. A. V. Design Science Research: método de pesquisa para a engenharia de produção. **Gestão & Produção**, São Carlos, v. 20, n.4, p. 741-761, 2013. Disponível em: https://www.scielo.br/j/gp/a/3CZmL4JJxLmxCv6b3pnQ8pq/?format=pdf&lang=pt. Acesso em: 6 jun. 2022.

LE, B. S.; DANG, V. L.; BUI, T. T. Swarm Robotics Simulation Using Unity. *In*: INTERNATIONAL CONFERENCE ON INTEGRATED CIRCUITS, DESIGN, AND VERIFICATION, 5., 2014, Vietnam. **Proceedings** [...]. Vietnam, 2014. p. 1 – 4. Disponível em: https://www.researchgate.net/publication/269701693_Swarm_Robotics_Simulation_Using_Unity. Acesso em: 26 set. 2022.

LI, L.; ZHONG, Y.; ZHON, S. Research on the Design and Development of a Web-based Physics Virtual Lab for Junior High Schools. *In*: INTERNATIONAL CONFERENCE ON EDUCATION TECHNOLOGY AND COMPUTER, 2., 2010, Shanghai. **Proceedings** [...]. Piscataway: IEEE, 2010. p. 484-489. DOI: https://doi.org/10.1109/ICETC.2010.5529204. Disponível em: https://ieeexplore.ieee.org/abstract/document/5529204. Acesso em: 13 marc. 2022.

LOPES, M. S. S.; GOMES, I. P.; TRINDADE, R. M. P.; SILVA, A. F.; LIMA, A. C. C. Web Environment for Programming and Control of a Mobile Robot in a Remote Laboratory. **IEEE Transactions on Learning Technologies**, v. 10, n. 4, p. 526-531, 2016. DOI: https://doi.org/10.1109/TLT.2016.2627565. Disponível em: https://ieeexplore.ieee.org/abstract/document/7740908/. Acesso em: 01 marc. 2023.

MAGNENAT, S.; SHIN, J.; RIEDO, F.; SIEGWART, R., BEN-ARI, M. Teaching a core CS concept through robotics. *In*: CONFERENCE ON INNOVATION & TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 19., 2014, Uppsalsa. **Proceedings** [...]. Nova Iorque: Association for Computing Machinery, 2014. p. 315 – 320. DOI: https://doi.org/10.1145/2591708.2591714. Disponível em: https://dl.acm.org/doi/abs/10.1145/2591708.2591714. Acesso em: 16 marc. 2022.

MAHAJAN, S.; KULKARNI, S.; DIWAKAR, A. S. A Pilot Study: The Effect of Using Virtual Laboratory on Students' Conceptual Understanding in Mobile Communications. *In*: IEEE EIGHT INTERNATIONAL CONFERENCE ON TECHNOLOGY FOR EDUCATION (T4E), 8., 2016, Mumbai. **Proceedings** [...]. Piscataway: IEEE. 2016, p. 7-14. DOI: https://doi.org/10.1109/T4E.2016.011. Disponível em: https://ieeexplore.ieee.org/abstract/document/7814786. Acesso em: 26 abr. 2022.

MALU, S. K.; MAJUMDAR, J. Kinematics, Localization and Control of Differential Drive Mobile Robot. **Global Journals of Research in Engineering**, *[S. l.]*, v. 14, n. 1, p. 1-7, 2014. Disponível em: https://engineeringresearch.org/index.php/GJRE/article/view/1233. Acesso em: 14 ago. 2022.

MARTINS, A.; **O que é robótica**. 2. ed. São Paulo: Brasiliense, 2006.

MECHARITHM. **Robotic Simulator: Creating an Advanced UGV Robot in Unity with C# (15/27)**. *[S. l.]*, 2022. Disponível em: https://www.mecharithm.com/robotic-simulator-creating-an-advanced-ugv-robot-in-unity-with-c-sharp/. Acesso em: 25 nov. 2022.

MELO, M. S. P.; NETO, J. G. S.; SILVA, P. J. L.; TEIXEIRA, J. M. X. N.; TEICHRIEB, V. Analysis and Comparison of Robotics 3D Simulators. *In*: SYMPOSIUM ON VIRTUAL AND AUGMENTED REALITY (SVR), 21., 2019, Rio de Janeiro. **Proceedings** [...]. Piscataway: IEEE, 2019. p. 242-251. DOI: https://doi.org/10.1109/SVR.2019.00049. Disponível em: https://ieeexplore.ieee.org/abstract/document/8921035. Acesso em: 21 set. 2022.

MOTA, M. P.; PEREIRA, L. W. K.; FAVERO, E. L. Javatool: Uma ferramenta para o ensino de programaçao. *In*: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 28., 2008, Belém. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2008. p. 127 – 136. Disponível em: http://www2.sbc.org.br/csbc2008/pdf/arq0111.pdf. Acesso em: 20 mar. 2022.

NIKU, S. B.; **Introduction to Robotics: Analysis, Control, Applications**. 3. ed. Hoboken: Wiley, 2020.

NORTHWESTERN. **CoppeliaSim Introduction.** *[S. l.]*, 2020. Disponível em: http://hades.mech.northwestern.edu/index.php/CoppeliaSim_Introduction. Acesso em: 19 nov. 2022.

PIMENTEL, M.; FILIPPO, D.; SANTORO, F. M. Design Science Research: fazendo pesquisas científicas rigorosas atreladas ao desenvolvimento de artefatos computacionais projetados para a educação. **RE@D-Revista de Educação a Distância e eLearning**, *[S. l.]*, v. 3, n. 1, p. 37-61, 2020. Disponível em: https://metodologia.ceie-br.org/wp-content/uploads/2018/10/cap1_5.pdf. Acesso em: 3 jun. 2022.

PITONAKOVA, L.; GIULIANI, M.; PIPE, A.; WINFIELD, A. Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators. *In*: ANNUAL CONFERENCE TOWARDS AUTONOMOUS ROBOTIC SYSTEMS, 19., 2018, Bristol. **Proceedings** [...]. Cham: Springer, 2018. p. 357-368. DOI: https://doi.org/10.1007/978-3-319-96728-8_30. Disponível em: https://link.springer.com/chapter/10.1007/978-3-319-96728-8_30. Acesso em: 21 set. 2022.

POTKONJAK, V.; GARDNER, M.; CALLAGHAN, V.; MATTILA, P.; GUETL, C.; PETROVIC, V. M.; JOVANOVIC, K. Virtual laboratories for education in science, technology, and engineering: A review. **Computers & Education**, *[S. l.]*, v. 95, p. 309-327, abr. 2016. DOI: https://doi.org/10.1016/j.compedu.2016.02.002. Disponível em: https://www.sciencedirect.com/science/article/pii/S0360131516300227. Acesso em: 11 jul. 2022.

PRIETO-BLAZQUEZ, J.; HERRERA-JOANCOMARTI, J.; GUERRERO-ROLDÁN, A. E. A Virtual Laboratory Structure for Developing Programming Labs. **International Journal of Emerging Technologies in Learning (iJET)**, Kassel, v. 4, n. 2009, p. 47-52, 24 out. 2009. DOI: https://doi.org/10.3991/ijet.v4s1.789. Disponível em: https://www.learntechlib.org/p/45237/. Acesso em: 3 mai. 2022.

ROBOTBENCHMARK. **About robotbenchmark**. *[S. l.]*, 2018. Disponível em: https://robotbenchmark.net/about.php. Acesso em: 10 ago. 2022.

RUS, D. **A Decade of Transformation in Robotics**. *[S. l.]*, 12 jun. 2019. Disponível em: https://www.bbvaopenmind.com/en/articles/a-decade-of-transformation-in-robotics/. Acesso em: 7 set. 2022.

SIROTHEAU, S; BALIEIRO, R. P.; FAVERO, E.; SANTOS, J. C. LabPy: Laboratório virtual de ensino em Python. *In*: CONGRESSO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 7., 2018, Fortaleza. **Anais** [...]. Fortaleza, 2018. p. 749 – 758. Disponível em: http://ojs.sector3.com.br/index.php/wcbie/article/view/8297. Acesso em: 17 ago. 2022.

SUWASONO, S.; PRIHANTO, D.; DWI WAHYONO, I.; NAFALSKI, A. Virtual Laboratory for Line Follower Robot Competition. **International Journal of Electrical and Computer Engineering (IJECE)**, *[S. l.]*, v. 7, n. 4, p. 2253-2260, ago. 2017. DOI: https://doi.org/10.11591/ijece.v7i3.pp2253-2260. Acesso em: 29 mar. 2022.

TINKERCAD. **Página inicial**. *[S. l.]*, 2022. Disponível em: https://www.tinkercad.com/. Acesso em: 25 jul. 2022.

UNITY. **Simulação de robótica - Unity.** *[S. l.]*, 2020. Disponível em: https://unity.com/pt/solutions/automotive-transportation-manufacturing/robotics. Acesso em: 2 ago. 2022.

VIANA, G.; LOPES, A.; PORTELA, C. OLIVEIRA, Sandro. Um Survey sobre a Aprendizagem de Programação no Curso de Sistemas de Informação. *In*: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 27., 2019, Belém. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 161-175. ISSN 2595-6175. DOI: https://doi.org/10.5753/wei.2019.6627. Disponível em: https://sol.sbc.org.br/index.php/wei/article/view/6627. Acesso em: 25 marc. 2023

XIE, M.; ZHOU, D.; SHI, Y.; JIA, R. Virtual Experiments Design for Robotics Based on V-REP. *In*: INTERNATIONAL CONFERENCE ON AUTOMATION, CONTROL AND ROBOTICS ENGINEERING, 3., 2018, Chengdu. **Proceedings** [...]. Bristol: IOP Publishing, 2018. p. 012069. DOI: https://doi.org/10.1088/1757-899X/428/1/012069. Disponível em: https://iopscience.iop.org/article/10.1088/1757-899X/428/1/012069/pdf. Acesso em: 23 jul. 2022.

ZHONG, B.; ZHENG, J.; ZHAN, Z. An exploration of combining virtual and physical robots in robotics education. **Interactive Learning Environments**, *[S. l.]*, p. 1-13, 02 jul. 2020. DOI: https://doi.org/10.1080/10494820.2020.1786409. Disponível em: https://www.tandfonline.com/doi/abs/10.1080/10494820.2020.1786409. Acesso em: 19 out. 2022.

APÊNDICE A – CÓDIGO DO CONTROLADOR DO SUPERVISOR

```
from controller import Supervisor
import os
import sys

try:
    includePath = os.environ.get("WEBOTS_HOME") +
"/projects/samples/robotbenchmark/include"
    includePath.replace('/', os.sep)
    sys.path.append(includePath)
    from robotbenchmark import robotbenchmarkRecord
except ImportError:
    sys.stderr.write("Warning: 'robotbenchmark' module not found.\n")
    sys.exit(0)

robot = Supervisor()

timestep = int(robot.getBasicTimeStep())

# Retorna um identificador para um nó no mundo a partir de seu nome DEF.
l1r2 = robot.getFromDef("L1R2")

# A função recupera um manipulador para um campo de nó. O campo é especificado
por seu nome em field_name e o nó ao qual pertence.
translation = l1r2.getField("translation")

running = True
stopMessageSent = False
while robot.step(timestep) != -1:
    # As funções wb_supervisor_field_get_sf_* recuperam o valor de um campo único
especificado (SF). O tipo do campo deve
    # corresponder ao nome da função utilizada, caso contrário o valor de retorno é
indefinido (e uma mensagem de aviso é exibida).
```

```
    t = translation.getSFVec3f()
    if running:
        time = robot.getTime()
        if ((t[2] > 1.60 and t[2] < 1.75) and (t[0] > 1.26 and t[0] < 1.75)):
            message = "stop"
            running = False

        # Essas funções permitem que o controlador do robô se comunique com uma
janela de robô HTML.
        robot.wwiSendText("time:%-24.3f" % time)
    else:  # aguarde a mensagem de gravação
        if not stopMessageSent:
            robot.wwiSendText("stop")
            stopMessageSent = True
        else:
            message = robot.wwiReceiveText()
            if message:
                if message.startswith("record:"):
                    record = robotbenchmarkRecord(message, "line_following", time)
                    robot.wwiSendText(record)
                    break
                elif message == "exit":
                    break

# Pausa a simulação quando ela é completada
robot.simulationSetMode(Supervisor.SIMULATION_MODE_PAUSE)
```

APÊNDICE B – CÓDIGO DO CONTROLADOR DO ROBÔ

```
from controller import Robot

def executar_robo(robot):

    time_step = 32
    velocidade_maxima = 2

    # motores
    motor_esquerdo = robot.getDevice('re')
    motor_direito = robot.getDevice('rd')
    motor_esquerdo.setPosition(float('inf'))
    motor_direito.setPosition(float('inf'))
    motor_esquerdo.setVelocity(0.0)
    motor_direito.setVelocity(0.0)

    # ativar sensores infravermelhos
    iv_esquerdo = robot.getDevice('ds_seguidor_de_linha3')
    iv_esquerdo.enable(time_step)

    iv_direito = robot.getDevice('ds_seguidor_de_linha1')
    iv_direito.enable(time_step)

    iv_meio = robot.getDevice('ds_seguidor_de_linha2')
    iv_meio.enable(time_step)

    # ativar sensor ultrassom
    ultrassom_frente = robot.getDevice('ds_ultrassom2')
    ultrassom_frente.enable(time_step)

    while robot.step(time_step)  != -1:
```

```
        # ler sensores infravermelhos
        valor_iv_esquerdo = iv_esquerdo.getValue()
        valor_iv_direito = iv_direito.getValue()
        valor_iv_meio = iv_meio.getValue()

        # ler sensor ultrassom
        valor_ultrassom_frente = ultrassom_frente.getValue()

        if(580 < valor_iv_esquerdo < 625) and (580 < valor_iv_direito < 625) and (580 <
valor_iv_meio < 625):
            if(valor_ultrassom_frente < 514):
                motor_esquerdo.setVelocity(-velocidade_maxima)
                motor_direito.setVelocity(velocidade_maxima)
                funcao_delay(robot, 1.8)

                motor_esquerdo.setVelocity(velocidade_maxima)
                motor_direito.setVelocity(velocidade_maxima)
                funcao_delay(robot, 3)

                motor_esquerdo.setVelocity(velocidade_maxima)
                motor_direito.setVelocity(-velocidade_maxima)
                funcao_delay(robot, 1.8)

                motor_esquerdo.setVelocity(velocidade_maxima)
                motor_direito.setVelocity(velocidade_maxima)
                funcao_delay(robot, 8)

                motor_esquerdo.setVelocity(velocidade_maxima)
                motor_direito.setVelocity(-velocidade_maxima)
                funcao_delay(robot, 1.8)

                motor_esquerdo.setVelocity(velocidade_maxima)
                motor_direito.setVelocity(velocidade_maxima)
                funcao_delay(robot, 3)
```

```
            motor_esquerdo.setVelocity(-velocidade_maxima)
            motor_direito.setVelocity(velocidade_maxima)
            funcao_delay(robot, 2.35)

        else:
            motor_esquerdo.setVelocity(velocidade_maxima)
            motor_direito.setVelocity(velocidade_maxima)
            funcao_delay(robot, 1.3)

            motor_esquerdo.setVelocity(velocidade_maxima)
            motor_direito.setVelocity(-velocidade_maxima)
            funcao_delay(robot, 1.8)
    elif (valor_iv_esquerdo > 625) and (valor_iv_direito > 625) and (valor_iv_meio
<= 625):
        motor_esquerdo.setVelocity(velocidade_maxima)
        motor_direito.setVelocity(velocidade_maxima)
    elif (valor_iv_esquerdo <= 625) and (valor_iv_direito > 625) and (valor_iv_meio
<= 625):
        motor_esquerdo.setVelocity(-velocidade_maxima)
        motor_direito.setVelocity(velocidade_maxima)
    elif (valor_iv_esquerdo > 625) and (valor_iv_direito <= 625) and (valor_iv_meio
<= 625):
        motor_esquerdo.setVelocity(velocidade_maxima)
        motor_direito.setVelocity(-velocidade_maxima)
    elif (valor_iv_esquerdo <= 625) and (valor_iv_direito > 625) and (valor_iv_meio
> 625):
        motor_esquerdo.setVelocity(-velocidade_maxima)
        motor_direito.setVelocity(velocidade_maxima)
    elif (valor_iv_esquerdo > 625) and (valor_iv_direito <= 625) and (valor_iv_meio
> 625):
        motor_esquerdo.setVelocity(velocidade_maxima)
        motor_direito.setVelocity(-velocidade_maxima)
```

```
        elif (valor_iv_esquerdo > 625) and (valor_iv_direito > 625) and (valor_iv_meio >
625):
            motor_esquerdo.setVelocity(velocidade_maxima)
            motor_direito.setVelocity(velocidade_maxima)

def funcao_delay(robot, tempo):
    tempo_atual_1 = float(robot.getTime())
    tempo_atual_2 = float(robot.getTime())

    while tempo_atual_2 < (tempo_atual_1 + tempo):

        tempo_atual_2 = float(robot.getTime())
        robot.step(1)

if __name__ == "__main__":
    my_robot = Robot()
    executar_robo(my_robot)
```

APÊNDICE C – CÓDIGO DO ARQUIVO LINE_FOLLOWING.HTML, UTILIZADO PARA O DESENVOLVIMENTO DA JANELA DE ROBÔ PERSONALIZADA

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset='UTF-8'>
 </head>
 <body>
  <div id='infotabs' class='webotsTabs webotsRobotWindowTabs'>
   <ul>
    <li><a href='#infotab-overview'>Visão geral</a></li>
    <li><a href='#infotab-metrics'>Métricas</a></li>
    <li><a href='#infotab-instructions'>Instruções</a></li>
    <li><a href='#infotab-explanations'>Explicações</a></li>
   </ul>
   <div id='infotab-overview'>
    <p>Esta simulação visa criar um algoritmo para que o robô L1R2 consiga, através da leitura de sensores, detectar e seguir a
     linha desenhada no chão, completando o trajeto no menor tempo possível. Durante o trajeto o robô encontrará diversos desafios
     que devem ser superados.</p>
    <p>A linguagem de programação utilizada no controlador é Python.</p>
   </div>
   <div id='infotab-metrics'>
    <div style='text-align:center'><em>t</em> = <span id='time-display'>00:00:00</span></div>
    <br>
    <p>A métrica de referência <em>t</em> é o tempo decorrido desde o início da simulação.</p>
    <p>A medição do tempo parará assim que o robô chegar no final do percurso.</p>
    <p>Minimizar esse tempo é um dos objetivos para esta simulação.</p>
   </div>
```

```
<div id='infotab-instructions'>
<h2>Programando um robô em Python</h2>
  <ol>
    <li><b>Abra o editor de código-fonte</b>: clique no robô com o botão direito do mouse para abrir o menu de contexto do robô e selecione a opção "Edit controller". Uma janela deve abrir contendo o código fonte (em Python) do controlador do robô.
    </li>
    <li><b>Edite o controlador</b>: uma parte do código já está pronta, como a inicialização de algumas variáveis, a importação das bibliotecas e as configurações iniciais do robô. O usuário deve implementar o restante do programa.</li>
    <li><b>Salve seu programa</b>: no editor de código-fonte, pressione <em id='saveShortcut'>Ctrl-S</em> para salvar seu programa.
    </li>
    <li><b>Reinicializar a simulação</b>: pressione o botão de reinicialização <img src='reset.png' alt='reset'> para levar em consideração suas modificações.</li>
    <li><b>Executar a simulação</b>: pressione o botão play <img src='real_time.png' alt='real-time'> para executar a simulação.
    </li>
    <li><b>Observe o resultado</b>: o robô deve avançar. Você pode acompanhar o seu desempenho em tempo real na guia <a class='tablink' onclick='$("#infotabs").tabs({active:1});'>Métricas</a>. Depois de um tempo, o robô deve parar e seu desempenho deve ser registrado (se você estiver logado).
      Tente modificar o programa do controlador novamente para obter um melhor desempenho.
    </li>
  </ol>
  <p>Durante a execução, possíveis avisos e mensagens de erro vindas do núcleo de simulação ou do controlador do robô são impressos no console.
    Você pode abrir o console clicando no botão da barra de ferramentas. <img src='console.png' alt='console'>.
  </p>
</div>
<div id='infotab-explanations'>
  <h2>L1R2</h2>
```

O L1R2 (Lara Remote Robot) é um robô seguidor de linha e resgate que pertence ao Laboratório Remoto em AVA (LARA), do curso de Ciência da Computação, na Universidade Estadual do Sudoeste da Bahia (UESB).

O robô possui três sensores de luminosidade na parte da frente, três sensores ultrassônicos também na frente e um em cada lateral e três sensores infravermelhos na parte inferior.

Para que o L1R2 possa fazer as curvas, ele utiliza como mecânica de locomoção o conceito de acionamento diferencial. Um robô é considerado de acionamento diferencial quando ele possui dois motores montados em posições fixas no seu lado esquerdo e direito, e cada um desses motores é acionado de forma independente. Para o robô ir para frente é necessário que ambas os motores sejam movidos na mesma velocidade, e para fazer uma curva para a esquerda motor direito precisa estar a uma velocidade maior do que o motor esquerdo e vice-versa para virar à direita. Também é possível fazer com que o robô gire em torno do seu próprio eixo, movendo ambas as rodas em direções opostas com a mesma velocidade. Para manter a estabilidade do robô é necessária uma terceira roda do tipo omnidirecional.

```
      </div>
    </div>
    <script src="line_following.js"></script>
  </body>
</html>
```

APÊNDICE D – CÓDIGO DO ARQUIVO LINE_FOLLOWING.JS, UTILIZADO PARA O DESENVOLVIMENTO DA JANELA DE ROBÔ PERSONALIZADA

```
$('#infotabs').tabs();

var benchmarkName = 'Line Following';
var timeString;
var roomCrossingTime;

// Permite que o controlador de robô se comunique com uma função JavaScript em execução na janela do robô HTML.
webots.window('line_following').receive = function(message, robot) {
  if (message.startsWith('time:')) {
    roomCrossingTime = parseFloat(message.substr(5));
    timeString = parseSecondsIntoReadableTime(roomCrossingTime);
    $('#time-display').html(timeString);
  } else if (message === 'stop') {
    if (typeof sendBenchmarkRecord === 'undefined' || !sendBenchmarkRecord(robot, this, benchmarkName, -roomCrossingTime, metricToString))
      $('#time-display').css('color', 'red');
  } else if (message.startsWith('record:OK:')) {
    $('#time-display').css('font-weight', 'bold');
    showBenchmarkRecord(message, benchmarkName, metricToString);
  } else if (message.startsWith('record:Error:')) {
    $('#time-display').css('color', 'red');
    showBenchmarkError(message, benchmarkName);
  } else
    console.log("Mensagem desconhecida recebida para o robô '" + robot + "': '" + message + "'");

  function metricToString(s) {
    return parseSecondsIntoReadableTime(-parseFloat(s));
  }
```

```
  // Transformar o tempo de segundos para minutos:segundos:centisegundos
  function parseSecondsIntoReadableTime(timeInSeconds) {
    var minutes = timeInSeconds / 60;
    var absoluteMinutes = Math.floor(minutes);
    var m = absoluteMinutes > 9 ? absoluteMinutes : '0' + absoluteMinutes;
    var seconds = (minutes - absoluteMinutes) * 60;
    var absoluteSeconds = Math.floor(seconds);
    var s = absoluteSeconds > 9 ? absoluteSeconds : '0' + absoluteSeconds;
    var cs = Math.floor((seconds - absoluteSeconds) * 100);
    if (cs < 10)
      cs = '0' + cs;
    return m + ':' + s + ':' + cs;
  }
};
```