

Universidade de São Paulo
Escola de Engenharia São Carlos
Departamento de Engenharia Mecatrônica

**PANORAMA E PERSPECTIVAS
FUTURAS DOS PRINCIPAIS
SOFTWARES DE SIMULAÇÃO PARA
ROBÓTICA MÓVEL: ESTUDO DE CASO
COM COPPELIA ROBOTICS.**

Leonardo Masson Oliveira

Trabalho de Conclusão de Curso

São Carlos, 2021

Universidade de São Paulo
Escola de Engenharia São Carlos
Departamento de Engenharia Mecatrônica
SEM0404 - Trabalho de Conclusão de Curso

PANORAMA E PERSPECTIVAS FUTURAS DOS PRINCIPAIS
SOFTWARES DE SIMULAÇÃO PARA ROBÓTICA MÓVEL: ESTUDO DE
CASO COM COPPELIA ROBOTICS.

Leonardo Masson Oliveira

Orientador: Marcelo Becker

Trabalho de Conclusão de Curso apre-
sentado à Escola de Engenharia de São
Carlos, da Universidade de São Paulo

Curso de Engenharia Mecatrônica

São Carlos, 2021

Leonardo Masson Oliveira

PANORAMA E PERSPECTIVAS FUTURAS DOS PRINCIPAIS
SOFTWARES DE SIMULAÇÃO PARA ROBÓTICA MÓVEL: ESTUDO DE
CASO COM COPPELIA ROBOTICS.

Trabalho de Conclusão de Curso

Este exemplar corresponde à redação
final do trabalho de conclusão de curso
de Leonardo Masson Oliveira

São Carlos, 2021.

Banca Examinadora

- Prof. Dr. Marcelo Becker.
- Prof. Dr. Daniel Varela Magalhães.
- Mateus Valverde Gasparino

FOLHA DE AVALIAÇÃO

Candidato: Leonardo Masson Oliveira

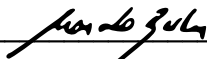
Título: Panorama e Perspectivas Futuras dos Principais *Softwares* de Simulação para Robótica Móvel: Estudo de Caso com o *Coppelia Robotics*

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos da
Universidade de São Paulo
Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Professor Marcelo Becker (Orientador)

Nota atribuída: **8,0 (OITO)**


(assinatura)

Professor Daniel Varela Magalhães

Nota atribuída: **8,0 (OITO)**


(assinatura)

Mateus Valverde Gasparino

Nota atribuída: **8,0 (OITO)**


(assinatura)

Média: **8,0 (OITO)**

Resultado: **APROVADO**

Data: 19/10/2021.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da
EESC

SIM ☐ NÃO ☐ Visto do orientador



Dedicatória

"Dedico esse trabalho aos meus pais, Marcos e Roberta, aos meus queridos irmãos, Matheus e Bianca, à minha namorada Ana Júlia e ao meu cachorrinho Rufus. "

Agradecimentos

"Agradeço aos meus amigos Felipe Marega, Letícia Cursini e João Manoel pela amizade desde o primeiro ano de faculdade. Vocês tornaram a graduação uma experiência mais feliz."

Resumo

É evidente, na atualidade, o aumento exponencial de projetos relacionados à robótica móvel, tanto em quantidade quanto em complexidade. A etapa de desenvolvimento de projetos relacionados a essa temática, por sua vez, também passa por mudanças e, na contemporaneidade, dispõe de diversas ferramentas que auxiliam no aumento tanto na eficiência quanto velocidade de cada etapa relacionada ao cronograma projetado. Nesse trabalho exploraremos as principais *features* dos softwares de simulação da atualidade, que são uma das ferramentas que mais evoluíram com o aumento do poder computacional evidenciado. Desenvolveremos, em V-REP, alguns algoritmos para demonstração dos principais marcos relacionados a esse tipo de projeto, traçando, por fim, um panorama do que mostra-se como destaque para o futuro desse segmento.

Palavras-chave: Robótica Móvel, Software de Simulação, V-REP, Algoritmos, Futuro desse Segmento.

Abstract

It is evident, nowadays, the exponential increase of projects related to mobile robotics, both in quantity and in complexity. The development stage of projects related to this theme, in turn, also undergoes changes and, nowadays, it has several tools that help both in the efficiency and speed of each milestone related to the schedule of these projects. In this project, we will explore the main features of current simulation software, which are one of the tools that have evolved the most with the increase in computational power evidenced. We will develop, in V-REP, some algorithms to demonstrate the main milestones related to this type of project, finally drawing an overview of what stands out for the future of this segment.

Keywords: *Mobile Robotics, Simulation Software, V-REP, Algorithms, Future of this Segment.*

Conteúdo

1	Introdução	12
1.1	Simulações na Robótica Móvel Autônoma.	12
1.2	Estruturação do texto.	13
2	Revião Bibliográfica.	14
2.1	Conceitos ROS:	14
2.1.1	Nós ROS:	14
2.1.2	Tópicos ROS:	15
2.1.3	Serviços ROS:	15
2.1.4	Mensagens:	16
2.1.5	Master:	16
2.1.6	Servidor de Parâmetros:	16
2.1.7	Infraestrutura de Comunicação	16
2.1.8	Entrega de mensagens	17
2.1.9	Motores Físicos:	17
2.2	VREP	18
2.2.1	Motores Físicos:	20
2.3	Perspectivas do futuro da Robótica Móvel	21
2.3.1	AWS	21
2.3.2	Viabilidade Econômica da computação em nuvem	23
2.3.3	AWS RoboMaker	23
3	Métodos.	24
3.1	Algoritmo de Braitenberg	24
3.1.1	API Coppelia Robotics - Python	26
3.1.2	Controle do modelo	27
3.2	Graduação do caminho: A*	27
3.2.1	Visão Computacional	27
3.2.2	Definição da melhor trajetória	29
3.2.3	Follow the Carrot: percorrendo a trajetória	30
3.2.4	Diagramas do sistema	30
4	Resultados	33
4.1	Simulações: Algoritmo de Braitenberg:	33
4.2	Simulações: Algoritmo Follow the Carrot	34
4.3	Tabela comparativa.	36
5	Conclusão	38

Lista de Figuras

1	Estrutura ROS.	16
2	Forma aleatória, aleatória aglutinada, convexa, convexa aglutinada, pura, pura aglutinada, demais formas, respectivamente.	19
3	Junta de rotação, prismática, trapezoidal e esférica.	20
4	Tipos de serviço de nuvem.	22
5	Cenário para simulação com algoritmo de Braitenberg	24
6	Esquemático do algoritmo de Braitenberg - Procura colisão.	25
7	Esquemático do algoritmo de Braitenberg - Evita colisão.	25
8	Robô Pioneer.	26
9	Robô Pioneer: vista frontal e lateral, respectivamente.	26
10	Imagem da cena utilizada para a simulação do algoritmo Follow the Carrot.	28
11	Cena de simulação tratada com visão computacional.	29
12	Esquemático do método	30
13	Planejamento de Trajetória realizado no começo do código.	31
14	Sistema de controle realizado toda em toda iteração do código.	31
15	Simulações considerando a variação do ganho de esterçamento.	33
16	Simulações considerando a variação da velocidade base.	33
17	Visualização da trajetória alvo à partir dos carrots.	34
18	Simulações para calibragem empírica dos controladores proporcionais. A linha roxa representa a trajetória percorrida pelo robô	35
19	Simulações para calibragem empírica dos controladores derivativos.	35
20	Sumário comparativo dos softwares.	37

1 Introdução

1.1 Simulações na Robótica Móvel Autônoma.

É fato que o aumento do poder de processamento dos computadores gerou um aumento na viabilidade dos métodos de simulação em diversos setores, como aeroespacial, manufatura e setores atrelados à engenharia, visto que esse método torna-se cada vez mais confiável e preciso à medida em que melhores modelos matemáticos são desenvolvidos e suportados por microprocessadores mais modernos.

Assim, as simulações tornaram-se, em muitos casos, alternativas de barateamento para a predição de eventos, sendo utilizada, também, no design de testes que seriam realizados de maneira efetiva e controlada por protótipos reais. Desse modo, é evidente que o impacto dessa tecnologia tem trazido benefícios para uma crescente área da Engenharia.

Além disso, o desenvolvimento dessas plataformas robóticas requer uma quantidade grande de testes, o que torna o sua etapa de desenvolvimento onerosa, Young (2018). Depreende-se, com isso, que o desenvolvimento de plataformas robóticas móveis completamente autônomas possuem seu sucesso intrinsecamente ligados à viabilidade econômica de sua fase de desenvolvimento.

Com relação aos entraves relacionados ao desenvolvimento de projetos robóticos, tais quais plataformas móveis autônomas, destacam-se três: (i) Acesso ao espaço físico, que pode ser entendido como galpões, bancadas, cenários para validações dos projetos, etc. (ii) Equipe de desenvolvimento multi-disciplinar e, finalmente, (ii) acesso aos componentes eletro-mecânicos, tais quais hardware, software e atuadores.

Nesse contexto, o auxílio de softwares de simulação de projetos robóticos tornam-se uma ferramenta cada vez mais importante na viabilização técnica e econômica dos projetos de robótica móvel, mostrando-se como catalisadores da fase tanto de desenvolvimento quanto de validação dos mesmos. Assim, o entendimento desses softwares, juntamente com o contínuo aprimoramento dos modelos utilizados, focando na representação cada vez mais precisa e versátil de robôs e suas interações com os ambientes aos quais estão inseridos, mostram-se como alavancas de crescimento importantes para esse segmento.

A obtenção de tais modelos, todavia, envolve um esforço multidisciplinar, visto que demanda a análise e representação de aspectos mecânicos, como a cinemática e dinâmica do robô, de características do ambiente, dos sensores e sua interação com o ambiente, entre outros, Martins(2019).

Assim, o emprego de ambientes simulados durante a etapa de testes é capaz de reduzir tanto o número dos mesmos quanto o seu tempo em ambientes reais, o que minimiza os riscos e os custos atrelados a essa etapa, Young (2018).

Dentre as aplicações da robótica móvel, podemos destacar a Agricultura de precisão, que apresenta desafios e peculiaridades próprias, como ambientes heterogêneos, suscetíveis à variações climáticas e à sazonalidade das culturas em questão, além de apresentarem uma onerosa etapa de testes, tendo em vista os pontos levantados anteriormente.

Podemos destacar, por exemplo, o uso de plataformas robóticas móveis no contexto da

agricultura, que se mostra uma alternativa viável para a redução da quantidade de pesticidas utilizados, visto que isso viabiliza um monitoramento contínuo de diversas culturas.

Contextualizando, o Brasil encontra-se, atualmente, como o segundo maior produtor de Soja do mundo, esta cultura ocupa uma área de 33,89 milhões de hectares, o que totalizou, em 2017, uma produção de 113,92 milhões de toneladas, de acordo com EMBRAPA (2020). Além disso, no panorama brasileiro, é cada vez mais notório a implementação da tecnologia em diversos processos agrícolas. Assim, é fato que inovações relacionadas ao melhor controle e distribuição de recursos, tais quais insumos e pesticidas, nas plantações são muito impactantes no desempenho brasileiro no setor.

Assim, além de apresentarem aplicações relevantes no contexto brasileiro, a robótica móvel mostra-se como ferramenta importante para alavancagem de diversos setores econômicos. Nesse contexto, as simulações robóticas apresentam-se como chave para o desbloqueio dessas avenidas de crescimento relativas à robótica móvel.

1.2 Estruturação do texto.

De maneira estruturada é destacado o que será abordado em cada capítulo do trabalho:

Revisão Bibliográfica: Abordou-se as principais *features* dos Softwares de simulação Coppelia Robotics, ROS e AWS, destacando similaridades e as perspectivas para o futuro no que tange à robótica móvel.

Métodos: Foi descrito o processo detalhado dos principais marcos relativos ao desenvolvimento de 2 algoritmos voltados para a robótica móvel: Algoritmo de Braitenberg e Follow the Carrot, abordando seus escopos e limitações.

Resultados: É mostrado os resultados das simulações considerando diferentes composições de parâmetros relacionados à modelagem dos robôs. Também foi realizado um quadro comparativo entre os softwares já mencionados, sumarizando o que foi descrito na revisão bibliográfica.

Conclusão: Comentou-se sobre os resultados obtidos nas simulações, relacionando o observado ao objetivado no início do projeto. Também foi apresentado um resumo sobre os principais tópicos do trabalho, pontuando as perspectivas para estudos futuros.

2 Revisão Bibliográfica.

2.1 Conceitos ROS:

O Sistema Operacional Robótico (ROS) é uma estrutura flexível para desenvolvimento de projetos de robótica. Trata-se de um conjunto de ferramentas, bibliotecas e convenções que buscam simplificar a tarefa de desenvolver um projeto robusto e complexo, tal qual um sistema robótico, através de uma variedade de plataformas.

Criado para incentivar o desenvolvimento colaborativo de sistemas robóticos, sua estrutura é baseada em arquitetura ponto-a-ponto (P2P), na qual cada um dos pontos ou nós da rede funciona tanto como cliente como servidor, o que possibilita que os membros envolvidos no projeto compartilhem serviços e dados sem a necessidade de um servidor central. Cabem aos nós realizar a transmissão de mensagens em um tópico, sendo denominados, assim, de publishers ou então monitorar algum tópico, os quais recebem o nome de subscriber. Os nós podem, também, serem produtores em um tópico e consumidores em outro tópico. Além disso, o ROS é uma plataforma livre e de código aberto, na qual a programação pode ser realizada em diversas linguagens, como C++ e Python.

Em uma aplicação que está sendo desenvolvida em ROS, pode-se considerar os nós, como descrito por Martins (2019), como processos que realizam processamento computacional. Os nós trocam informações entre eles através de mensagens, as quais trafégam entre tópicos.

Em um contexto de projetos de sistemas robóticos cada vez maiores e mais complexos, o desenvolvimento dos mesmos de maneira modular é cada vez mais comum. Assim, a estrutura do ROS apresenta diversas ferramentas e soluções que possibilitam o reaproveitamento de códigos, modularidade e escalabilidade do projeto, mostrando-se como um software muito relevante no contexto de desenvolvimento de sistemas robóticos.

ROS possui três níveis diferentes de conceito: o Nível do sistema de arquivos, nível do grafo de computação e nível da comunidade. O nível do sistema de arquivos cobre, principalmente, os recursos ROS encontrados no disco, tais quais: Pacotes, Repositórios, tipos de mensagens e tipos de serviços. O nível de grafo de computação é uma rede ponto-a-ponto de processos ROS, que processam dados juntos. Os principais elementos desse nível são: Nós, Master, Servidor de Parâmetros, Mensagens, Serviços, Tópicos e tudo que relacionada a gerir dados para o grafo de diferentes maneiras. O nível de comunidade ROS são os recursos que possibilitam separar comunidades para trocar software e conhecimento. Seus principais elementos são: Distribuidores, Repositórios e ROS Wiki (ROS, 2021).

2.1.1 Nós ROS:

Um nós ROS é o processo de realizar processamento computacional. Nós são agrupados em uma rede e comunicam-se entre si através da transmissão de Tópicos, Serviços RPC e parâmetros de servidor. Os nós são designados para operar em uma pequena escala, de modo que o controle de um sistema robótico é usualmente realizados por diversos nós.

Dentre os benefícios da utilização de nós ROS podemos destacar a tolerância a erros, de modo que os mesmos estarão isolados em nós individuais. A complexidade do código é reduzida quando comparado a blocos únicos de algoritmos. Os detalhes de implementação

são escondidos, de modo que os nós expõe pouco do seu API para o resto da rede e possibilita a fácil alternância e implementação de diferentes linguagens de programação (ROS, 2021).

2.1.2 Tópicos ROS:

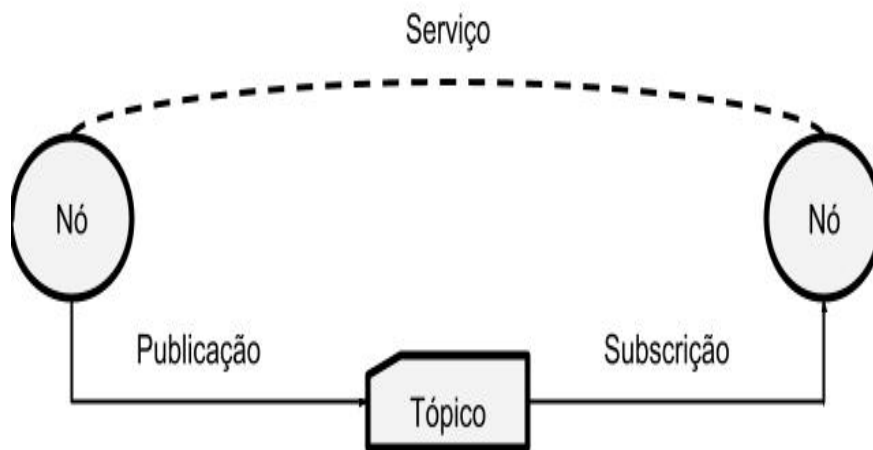
Tópicos são agrupamentos nomeados pelos quais nós trocam mensagens. Eles têm semântica de publicar/assinar anônima, o que separa a produção de informação de seu consumo. De maneira geral, os nós não sabem com quem estão se comunicando. Ao invés disso, os nós estão interessados nos dados e consomem os mesmos dos tópicos e os nós que geram dados publicam os mesmos nos tópicos. Podem haver diversos publishers e subscribers para um tópico ROS.

Atualmente o ROS suporta o transporte de mensagens via TCP/IP e UDP. O transporte TCP/IP é conhecido como TCPROS e comunica dados através de conexões TCP/IP. TCPROS é o transporte padrão utilizado em ROS e é o único transporte que os usuários de bibliotecas são obrigados a suportar. O transporte UDP, que é conhecido como UDPROS e é atualmente suportado somente por roscpp, separa as mensagens em pacotes UDP. UDPROS é um transporte de baixa latência e com perdas de informação, de maneira que é mais recomendado para tarefas de tele-operação (ROS, 2021).

2.1.3 Serviços ROS:

O modelo de publicação/assinatura é um paradigma muito flexível de comunicação, mas seu transporte many-to-many one-way não é apropriado para interações de solicitação/resposta RPC, que é normalmente demandada em sistemas distribuídos. A solicitação/resposta é realizada por meio de um Serviço, que é definido por um par de mensagens: uma para o pedido e outra para o resposta. Um nó ROS provedor oferece um serviço sob a string name, e o client chama o serviço mandando uma mensagem de pedido e esperando uma resposta. Bibliotecas geralmente apresentam essa interação ao programador como se fosse um procedimento de chamada remota. O cliente pode fazer conexões persistentes com um serviços, o que permite uma maior performance em troca de menor robustez para mudanças no provedor de serviços (ROS, 2021).

Figura 1: Estrutura ROS.



Fonte: elaborado pelo autor.

2.1.4 Mensagens:

Nós comunicam-se entre si através da publicação de mensagens em tópicos. A mensagem é uma estrutura de dados simples, contendo campos digitados (integers, floating point, boolean, etc.). Os nós também podem trocar mensagens de solicitações e respostas como uma parte da ligação ROS (ROS, 2021).

2.1.5 Master:

O ROS Master fornece a nomeação e registros de serviços para os demais nós ROS do sistema. Ele rastreia as publicações e as assinaturas até os tópicos e serviços. Seu papel é de viabilizar os nós ROS individuais a localizarem uns aos outros. Uma vez que esses nós se encontrem, a comunicação é realizada peer-to-peer. O ROS Master também fornece o Servidor de Parâmetros (ROS, 2021).

2.1.6 Servidor de Parâmetros:

O Servidor de Parâmetros é um dicionário compartilhado multivariado que é acessível através de redes APIs. Nós utilizam esse servidor para armazenar e receber parâmetros durante o tempo de execução. Como não se trata de um servidor de alta performance, seu uso mais comum é para fins estáticos, como configurações de parâmetros. Ele é designado para ser visível globalmente, de modo que as ferramentas possam acessar e configurar as configurações do sistema se necessário (ROS, 2021).

2.1.7 Infraestrutura de Comunicação

De maneira generalizada, ROS oferece uma interface de trocas de mensagens que possibilita a comunicação inter-processos e é conhecido com middleware (software que encontra-se entre o sistema operacional e os aplicativos nele executados, funcionamento, principalmente,

como uma interface oculta de tradução, possibilitando a comunicação e o gerenciamento de dados para os aplicativos distribuídos (Azure, 2021).

2.1.8 Entrega de mensagens

No desenvolvimento e implementação de uma nova aplicação robótica, o sistema de comunicação é, geralmente, um das primeiras necessidades a se apresentarem. O sistema integrado de entrega de mensagens ROS possibilita uma ágil resolução desse problema, fornecendo uma comunicação entre os diversos nós através de seu sistema anônimo de publish/subscribe. Também podemos destacar, nesse componente, é que a entrega de mensagens demanda uma implementação enxuta de interfaces entre os nós do sistema, assim, melhorando o encapsulamento e promovendo a reutilização de código.

2.1.9 Motores Físicos:

É possível, através do ROS, conectar-se com diversas bibliotecas. Nesse contexto, podemos conectar, no ambiente ROS, ao Gazebo, que é um simulador multi-robô 3D, com física dinâmica e cinemática e um motor de física plugável. A integração entre ROS e o Gazebo é realizado por um conjunto de plug-ins do Gazebo que suportam muitos robôs e sensores existentes. Como os plug-ins apresentam a mesma interface de mensagem que o resto do ecossistema ROS, é possível escrever nós ROS que são compatíveis com simulação, dados registrados e hardware. Assim, pode-se desenvolver um aplicativo em simulação e, em seguida, implantar no robô físico com pouca ou nenhuma alteração em seu código (Gazebo,2021).

Dentre os motores físicos fornecidos pelo Gazebo, destacamos: *OpenDE*, *Bullet*, *Simbody*, e *DART*.

OpenDE:

Trata-se de uma biblioteca gratuita de qualidade industrial focada na simulação e dinâmica de corpos rígidos articulados - por exemplo, veículos terrestres, criaturas com pernas e objetos em movimento em ambientes de realidade virtual. Também apresenta um detector de colisão embutido. É rápido, flexível e robusto (OpenDE,2021),

Bullet:

Já descrito nas engines do VREP.

Simbody:

Simbody é útil para coordenada interna e modelagem de molécula de granulação grossa, modelos mecânicos em grande escala como esqueletos e modelados como corpos interconectados por juntas, atuados por forças e restritos por restrições.

Este projeto é um conjunto de ferramentas *SimTK* que fornece capacidade de dinâmica multicorpo geral, ou seja, a capacidade de resolver a 2ª lei de Newton $F = ma$ em qualquer conjunto de coordenadas generalizadas sujeito a restrições arbitrárias. O *Simbody* é fornecido como uma API C++ de código aberto orientada a objetos e oferece resultados de qualidade científica / engenharia de alto desempenho e controle de precisão.

Simbody usa uma formulação avançada ao estilo *Featherstone* (Algoritmo de *Featherstone* é uma técnica usada para calcular os efeitos das forças aplicadas a uma estrutura de juntas e ligações em uma "corrente cinemática") de mecânica de corpo rígido para fornecer resultados

em tempo Ordem (n) para qualquer conjunto de coordenadas n generalizadas. Isso pode ser usado para modelagem de coordenadas internas de moléculas ou para modelos de granulação grossa com base em blocos maiores. Também é útil para modelos mecânicos em grande escala, como modelos neuromusculares, robótica, avatares e animação. O *Simbody* também pode ser usado em aplicativos interativos em tempo real para biosimulação, bem como para mundos virtuais e jogos (Simbody, 2021).

DART:

DART (Dynamic Animation and Robotics Toolkit) é uma plataforma colaborativa *cross, open source* desenvolvida pela Graphics Lab.

A biblioteca fornece estruturas de dados e algoritmos para aplicações cinemáticas e dinâmicas em robótica e animação por computador. O *DART* se distingue por sua precisão e estabilidade devido ao uso de coordenadas generalizadas para representar sistemas de corpo rígido articulado e o Algoritmo de Corpo Articulado de *Featherstone* para calcular a dinâmica do movimento. Para desenvolvedores, em contraste com muitos mecanismos de física populares que veem o simulador como uma caixa preta, o *DART* dá acesso total a quantidades cinemáticas e dinâmicas internas, como a matriz de massa, Coriolis e forças centrífugas, matrizes de transformação e seus derivados.

O *DART* também fornece um cálculo eficiente de matrizes Jacobianas para pontos corporais arbitrários e quadros de coordenadas. A semântica do quadro do *DART* permite aos usuários definir quadros de referência arbitrários (inerciais e não inerciais) e usar esses quadros para especificar ou solicitar dados. Para segurança de código hermético, os valores da cinemática direta e da dinâmica são atualizados automaticamente por meio de *lazy evaluation* (na teoria de programação, *Lazy Evaluation* é a expressão utilizada para a estratégia de atrasar a avaliação de uma expressão até o seu valor ser necessário, o que também evita a repetição de avaliações), tornando o *DART* adequado para controladores em tempo real. Além disso, o *DART* oferece flexibilidade para estender a API para incorporar classes fornecidas pelo usuário em estruturas de dados *DART*. Contatos e colisões são tratados usando um LCP (problema de complementaridade linear) baseado em velocidade implícita, para garantir não penetração, atrito direcional e condições aproximadas do cone de atrito de Coulomb. O *DART* tem aplicativos em robótica e animação por computador porque apresenta um simulador dinâmico multicorpo e várias ferramentas cinemáticas para controle e planejamento de movimento (DART, 2021).

2.2 VREP

O VREP (Virtual Robot Experimentation Platform) é um software que apresenta um ambiente de desenvolvimento integrado que possibilita a modelagem, edição, programação e simulação de qualquer sistema robótico

É uma plataforma que apresenta uma integração com diversas linguagens de programação, como Python, C/C++, MATLAB e Lua, através de seus APIs, o que justifica seu extensivo uso para controle, monitoramento e o teste rápido de novos algoritmos utilizados nos sistemas embarcados.

Como explicado por Martins (2019), o avanço do poder de processamento dos computadores possibilitou, no campo da simulação, o desenvolvimento de simulações com HIL (hardware-in-loop). Assim sendo, uma plataforma de simulações precisa oferecer escalabilidade e dinamismo para a fase de desenvolvimento dos protótipos, e, nesse sentido, o VREP



Figura 2: Forma aleatória, aleatória aglutinada, convexa, convexa aglutinada, pura, pura aglutinada, demais formas, respectivamente.

fonte: Coppelia Robotics

mostra-se como uma opção viável.

Encontra-se, na interface do software, diferentes objetos da cena, com destaque para as juntas, formas, sensores de proximidade, sensores de força e modelos, segundo Rohmer, Singh e Freese (2013) em Martins (2019).

As formas(shapes), são objetos rígidos, os quais podem ser importados de outros softwares de modelagem 3d, como SolidWorks/SolidEdge, e resultam em uma malha triangular. Dentro dos formas, encontram-se diferentes categorias: Forma aleatória, Forma aleatória aglutinada, forma convexa, forma convexa aglutinada e formas simples.

Segundo a Coppelia Robotics, as formas aleatórias podem representar qualquer malha. É composta por apenas uma cor e um conjunto de atributos visuais, não sendo, assim, otimizada e nem recomendada para simulações que envolvam cálculos de resposta a colisões dinâmicas.

As formas convexas simples são representadas por uma cor e um conjunto de atributos visuais, otimizada para cálculos de resposta a colisões dinâmicas.

As formas simples representam formas primitivas, como cubos, esferas e cilindros. São a melhor opção para simulações que envolvam dinâmicas de colisões e seu estudo, visto que tratam-se de estruturas estáveis e apresentam rápidas respostas.

As demais formas apresentam propriedades estéticas capazes de reproduzir, de maneira mais fidedigna, a real forma dos objetos a serem simulados, contudo, não são utilizados para os cálculos da dinâmica da cena.

A interação entre todos esses elementos na cena dá-se pelo estabelecimento de hierarquias, nas quais as formas puras são tidas como invisíveis mas associadas à dinâmica da cena, enquanto as demais formas não são contabilizadas nos cálculos da simulação mas são visíveis na mesma.

Ainda, como descrito por Martins (2019), as juntas são elementos, dentro da cena, que possibilitam a interligação entre as diferentes formas. Assim sendo, as principais juntas são: junta rotacional, prismática, trapezoidal e esférica, sendo que todas permitem 1 GDL, com exceção da esférica, que possibilita 3 GDL. Podemos, também, configurar as juntas de acordo com a sua cinemática inversa, torque-força ou no modo passivo. Além disso, pode-se, através dos scripts da cena, controlar as juntas de torque/força, atribuindo a essas posições ou velocidades, sendo, portanto, muito utilizadas nas simulações robóticas.

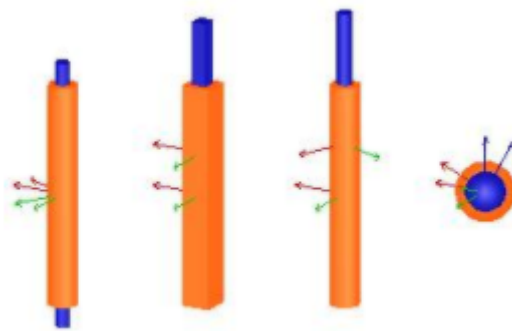


Figura 3: Junta de rotação, prismática, trapezoidal e esférica.
fonte: Coppelia Robotics

2.2.1 Motores Físicos:

Atualmente, o Coppelia Robotics suporta 4 tipos diferentes de motores: o *Bullet Physics Library*, *Open Dynamics Engine*, *Vortex Studio* e *Newtons Dynamics Engine*. O usuário pode, a qualquer momento, alternar entre os motores de acordo com as necessidades de cada simulação. O motivo pela diversidade de disponibilidade dos mesmos se dá pelo motivo de simulações físicas serem tarefas complexas, que demandam de diferentes níveis de precisão e velocidade, o que pode ser atendido com diferentes suportes de motores (Coppelia Robotics, 2021).

Bullet Physics Library:

Bullet Physics é uma biblioteca profissional de detecção de colisão de código aberto, corpo rígido e dinâmica de corpo mole. Nela visa-se o uso em tempo real e interativo em jogos, efeitos visuais em filmes e robótica. A biblioteca é gratuita para uso comercial sob a licença zlib (BulletPhysics, 2021).

Características principais:

- Código C ++ *open source* sob licença zlib e gratuito para qualquer uso comercial em todas as plataformas.
- Detecção de colisão discreta e contínua, incluindo teste de raio e varredura convexa. As formas de colisão incluem malhas côncavas e convexas e todos os primitivos básicos.
- Solucionador de restrições de dinâmica de corpos rígidos rápido e estável, dinâmica de veículo, controlador de caracteres e controle deslizante, dobradiça 6 GDL genérico.
- Dinâmica de corpo macio para volumes de tecido, corda e volumes deformáveis com interação bidirecional com corpos rígidos, incluindo suporte de restrição (Feature não disponível no Coppelia Robotics).
- Formato de arquivo .bullet binário nativo e importadores de exemplo para arquivos URDF, Wavefront obj e Quake bsp.

Open Dynamics Engine:

ODE é uma biblioteca de código aberto de alto desempenho para simular a dinâmica de corpos rígidos. É completo, estável, maduro e independente de plataforma com uma API C / C ++ fácil de usar. Possui tipos de junta avançados e detecção de colisão integrada

com fricção. ODE é útil para simular veículos, objetos em ambientes de realidade virtual e criaturas virtuais. Atualmente é usado em muitos jogos de computador, ferramentas de autoria 3D e ferramentas de simulação.

um motor de física de código aberto com dois componentes principais: dinâmica de corpo rígido e detecção de colisão. Ele tem sido usado em muitos aplicativos e jogos. Muitas vezes, é considerado um motor de física de jogo (ODE, 2021).

Vortex Studio:

Trata-se de um motor de física comercial de código fechado que produz simulações de física de alta fidelidade. O Vortex oferece parâmetros do mundo real (ou seja, correspondentes a unidades físicas) para um grande número de propriedades físicas, tornando este mecanismo realista e preciso. O Vortex é usado principalmente em aplicações industriais e de pesquisa de alto desempenho/precisão. O plugin Vortex para CoppeliaSim é baseado no Vortex Studio, que requer que cada usuário se registre no CM Labs, para obter uma chave de licença gratuita (Vortex, 2021).

Newton Dynamics:

Trata-se de uma biblioteca de simulação de física semelhante a uma plataforma cruzada. Ele implementa um solver determinístico, que não é baseado no LCP tradicional ou métodos iterativos, mas possui a estabilidade e a velocidade de ambos, respectivamente. Esse recurso torna o Newton Dynamics uma ferramenta não apenas para jogos, mas também para qualquer simulação de física em tempo real. A implementação atual do plugin é uma versão BETA (Newton Dynamics, 2021).

2.3 Perspectivas do futuro da Robótica Móvel

2.3.1 AWS

A computação em nuvem pode ser entendida como a entrega, sob demanda, de diferentes recursos, tais quais poder computacional, banco de dados, aplicativos e recursos de TI através de internet e o preço a ser pago é definido de acordo com o seu uso. Esses recursos só são possíveis pois são executados através de computadores servidores que localizam-se em grandes datacenters ao redor do mundo.

Atualmente, organizações de todos os tipos, setores e portes utilizam o serviço. Dentre os serviços disponíveis no mercado, destaca-se a AWS (Amazon Web Services), e, de acordo com o site da empresa, backup de dados, recuperação de desastres e desenvolvimento e teste de softwares estão entre os serviços mais utilizados.

Tradicionalmente entende-se que a infraestrutura é composta por hardware, o qual exige espaço físico, equipe, segurança física e planejamento de despesas de capital, o que, além de um CAPEX inicial elevado, apresenta um período longo entre sua aquisição, implantação e utilização por parte do usuário final, o que, de acordo com o "MATERIAL DO CURSO", é caracterizado como um ciclo longo de aquisição.

Além disso, a volatilidade da utilização do processamento de dados torna difícil seu dimensionamento, o que, em momentos de pico pode resultar em incompatibilidade com o projeto, que demanda mais poder computacional, e em momentos de baixa utilização a ociosidade dos recursos alocados.

Nesse contexto, a computação em nuvem possibilita considerar que a infraestrutura é, na verdade, composta por software. Utilizando a AWS não há necessidade de provisionamento de recursos em excesso para absorver picos de atividades empresariais no futuro, de modo que o aumento ou diminuição da escala desses recursos dá-se de maneira instantânea, ajustando-se às necessidades de todos os projetos. Além disso, a nuvem permite que as despesas de implantação (CAPEX), tornem-se custos variáveis (Amazon AWS, 2021).

Modelos de serviço em nuvem

Podemos agrupar em 3 os principais modelos de serviços em nuvem, sendo que cada um representa um nível diferente de controle sobre os recursos de TI.



Figura 4: Tipos de serviço de nuvem.
fonte: Amazon AWS

Infraestrutura como Serviço (IaaS):

Esse grupo apresenta o maior nível de controle sobre os recursos de TI, e, em consequência, a maior flexibilidade. É, também, o mais parecido com os que os departamentos dedicados de TI das empresas possuem tradicionalmente, com acessos aos recursos de rede e espaço de armazenamento.

Suas vantagens são a facilidade de adoção e alta escalabilidade, contudo seu custo a longo prazo pode ser maior.

Plataforma como Serviço (PaaS):

É a segunda camada e possibilita a redução da necessidade de manutenção e gerenciamento da infraestrutura. Suas vantagens são a possibilidade de dedicação maior ao escopo do projeto.

Software como Serviço (SaaS):

Observamos nesse grupo que os serviços apresentam-se como plataformas e produtos completos, no qual seu gerenciamento e manutenção dá-se integralmente pelo fornecedor. É caracterizado, principalmente, como serviços voltados ao usuário final (front-end). Suas vantagens são possibilitar a integralidade da dedicação das pessoas envolvidas em como utilizar da melhor maneira essa ferramenta.

Dentro das ferramentas disponibilizadas pela a AWS, podemos destacar a Amazon Elastic Comput Cloud (Amazon EC2), a qual fornece máquinas virtuais para hospedagem de aplicativos que seriam hospedados em servidores tradicionais, fornecendo capacidade computacional segura e redimensionável na nuvem, sendo eles servidores de aplicativos, Web, bancos de dados, jogos, e-mail, computação, etc.

Nesses ambientes virtuais, as chamadas instâncias do EC2 na nuvem, há acesso total ao sistema operacional convidado em cada instância, sendo que é possível a criação de instâncias simultâneas, de qualquer tamanho, em qualquer uma das zonas de disponibilidades distribuídas pelo mundo.

2.3.2 Viabilidade Econômica da computação em nuvem

Quando analisamos a AWS, existem 3 aspectos que são, segundo a AWS Academy, preponderantes na definição de preço, sendo eles: computação, armazenamento e transferência de dados, sendo que cada um desses tópicos apresentam um mecanismo de precificação atrelado somente ao uso dos mesmos, de maneira que aumento ou quedas de demanda computacional possam ser entendidos como variáveis.

Destaca-se, também, que é possível, através de provisionamento do uso de dados utilizados antecipadamente, e dos ganhos de economia de escala, realizados pela AWS, economizar até 75 por cento dos custos da operação.

2.3.3 AWS RoboMaker

AWS RoboMaker é um serviço que facilita a criação de aplicações robóticas em escala. Esse serviço amplia a estrutura do Robot Operating System (ROS) com serviços de nuvem, o que inclui serviços de aprendizado de máquina AWS, monitoramento de serviços, analytics dentre outros. Esses combinados fornecem, ao robô, a capacidade de realizar diversas atividades de maneira independente: stream de dados, navegação, comunicação, compreensão e aprendizado. Vemos, assim, que o AWS RoboMaker fornece diversas soluções para aplicações de robótica (AWS Robomaker, 2021).

Com os recentes avanços nas área de Inteligência Artificial, a computação em nuvem e a robótica convergem cada vez mais para serem tecnologias que se combinam de maneira a criar e operar robôs de maneiras cada vez mais diversa. Já é possível realizar projetos que agregam serviços de Computação em Nuvem, Armazenamento e outros produtos AWS com Robótica Móvel, aérea e terrena, abrindo possibilidades de compartilhamento simultâneo de dados através de nuvem, o que possibilita o seu alinhamento com seus pares e para a realização de tarefas complexas.

Portanto, vemos que essa ferramenta está diretamente relacionada ao enfrentamento das dificuldades atreladas às fases de testes de plataformas robóticas móveis, fornecendo, para isso, ambientes de simulação em nuvem que agrega

3 Métodos.

Path Planning and Tracking.

Evitar obstáculos, no contexto da robótica móvel, implica no ajuste autônomo de trajetória do robô, visando, através dos atuadores, orientar seu deslocamento para mais longe dos obstáculos.

O planejamento de trajetória, por sua vez, busca, através de seu algoritmo, encontrar o melhor caminho possível para se percorrer entre obstáculos visando chegar em algum ponto específico.

3.1 Algoritmo de Braitenberg

Valentino Braitenberg, idealizador do algoritmo abordado nessa seção, foi um Neurocientista/cibernetista que, em seu livro "Vehicles: Experiments in Synthetic Psychology", demonstrou como veículos hipotéticos poderiam demonstrar comportamentos que parecessem inteligentes através de simples mecanismos (combinações de sensores, atuadores e suas interconexões).

Considerando o cenário abaixo:

Figura 5: Cenário para simulação com algoritmo de Braitenberg



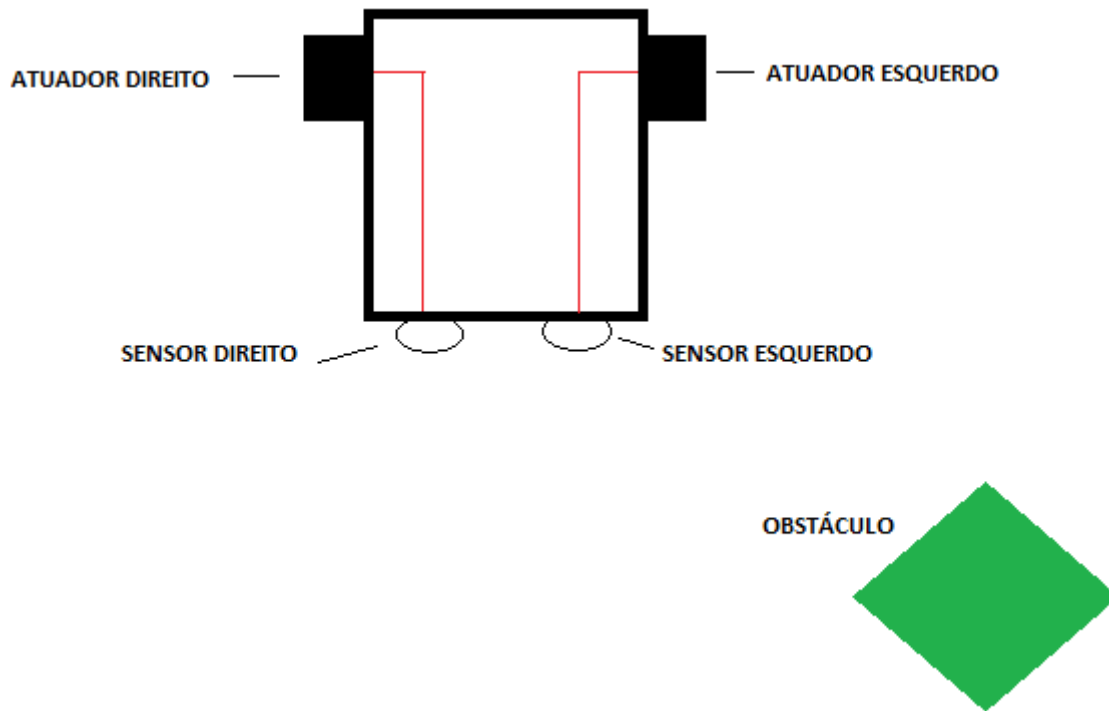
fonte: Elaborado pelo autor.

Podemos observar os seguintes elementos:

Nesse contexto, observa-se que o obstáculo apresenta-se mais próximo do sensor esquerdo do que o direito. Considerando que o atuador esquerdo está diretamente ligado ao sensor esquerdo, esse será acionado com mais velocidade do que o direito, acarretando em uma mudança da direção do robô no sentido de afastar-se do obstáculo, o que caracteriza o algoritmo que evita colisões.

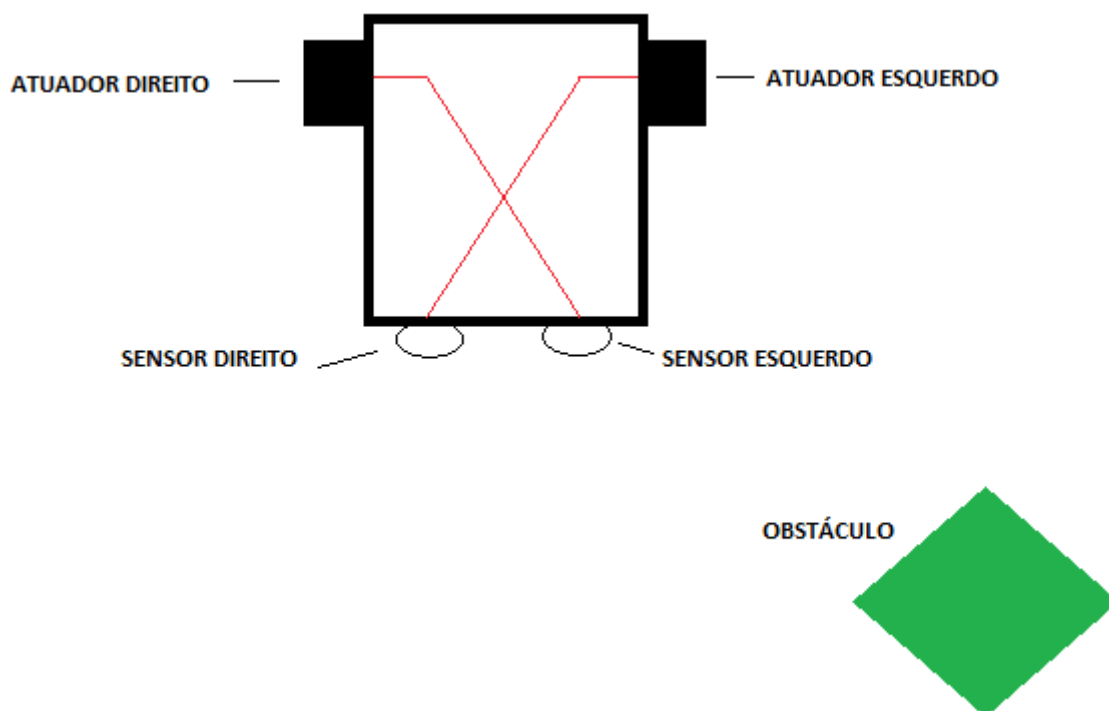
Agora, ao inverter-se a conexão dos sensores com os atuadores, o comportamento esperado é o oposto, visto que o obstáculo fornece um estímulo maior ao sensor esquerdo, esse irá acionar o atuador direito com mais intensidade, o que posiciona o robô na direção desse obstáculo, o que o caracteriza como um algoritmo que busca esse obstáculo.

Figura 6: Esquemático do algoritmo de Braitenberg - Procura colisão.



fonte: Elaborado pelo autor.

Figura 7: Esquemático do algoritmo de Braitenberg - Evita colisão.



fonte: Elaborado pelo autor.

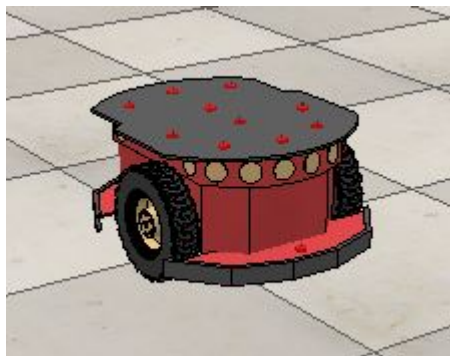
Para a aplicação desenvolvida nesse trabalho utilizou-se a plataforma robótica móvel Pioneer, que acompanha o software Coppelias Robotics em sua versão estudantil. Nela dispomos de 16 sensores de proximidade, distribuídos de maneira a cobrir o máximo raio de cobertura para a varredura durante suas navegações. A orientação dos sensores dá-se da

Sensor 1	$-\pi/2$
Sensor 2	$-50/180 * \pi$
Sensor 3	$-30/180 * \pi$
Sensor 4	$-10/180 * \pi$
Sensor 5	$10/180 * \pi$
Sensor 6	$30/180 * \pi$
Sensor 7	$50/180 * \pi$
Sensor 8	$\pi/2$

Tabela 1: Distribuição dos sensores [radianos]

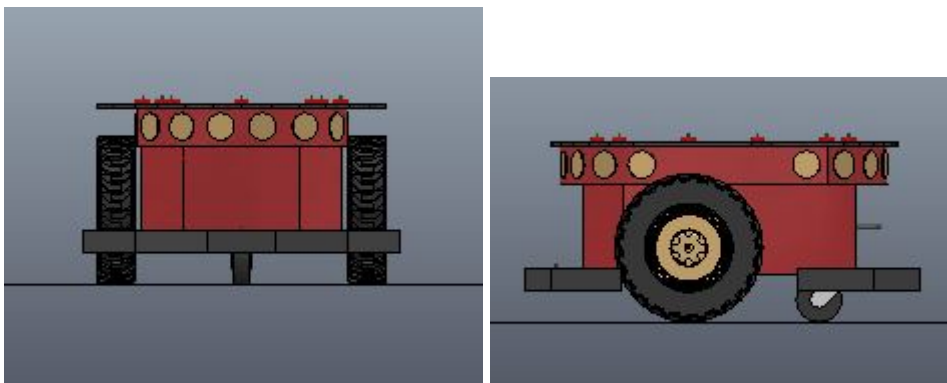
seguinte maneira:

Figura 8: Robô Pioneer.



fonte: Elaborado pelo autor.

Figura 9: Robô Pioneer: vista frontal e lateral, respectivamente.



fonte: Elaborado pelo autor.

3.1.1 API Coppelja Robotics - Python

A realização do API entre Coppelja Robotics e o ambiente de desenvolvimento Python se deu da seguinte maneira:

```

1
2 # c digo para movimenta o do Pioneer e estabelecimento do servidor Python -
  V-REP
3
4 import sim
5 import simConst

```

```

6 import numpy as np
7 import math
8 import sys
9 import time #Utilizado para acompanhar o tempo de simulacao decorrido.
10
11 sim.simxFinish(-1) # Encerra todas as coneccoes abertas.
12
13 clientID=sim.simxStart('127.0.0.1',19997,True,True,5000,5) # Coneccao com o
    Coppelias Robotics.
14
15 # Abre a cena desse algoritmo no ambiente de simulacao #
16
17 errorCode = sim.simxLoadScene(clientID,'cena_evita_colisao.ttt',0,sim.
    simx_opmode_blocking)

```

3.1.2 Controle do modelo

Em posse da distribuição dos sensores é possível iterar entre eles durante um período arbitrário de tempo e encontrar o mais próximo a obstáculos através da função “mínimo” inclusa nas bibliotecas mapeadas.

Assim, pode-se atribuir o valor de velocidade aos atuadores responsáveis pelo acionamento dos pneus, de modo que o Pioneer afaste-se dos obstáculos. Para isso, implementou-se a seguinte lógica: verifica-se se o sensor mais próximo está a menos de 0.2 unidades de distância do obstáculo, compondo, se verdadeiro, o valor do esterçamento “steer”.

$$steer = -1/orientação[min_{i,nd}] \quad (1)$$

Assim, tendo em vista que os sensores mais próximos da lateral do robô apresentam um valor de orientação maior do que os que estão na parte frontal do mesmo, ao realizar-se a inversão desse número, esterçamos mais o veículo quando o sensor mais próximo do obstáculo está posicionado na sua dianteira, e esterça-se menos quando o sensor mais próximo localiza-se em sua lateral, fornecendo, assim, um senso de direcionamento melhor para o algoritmo.

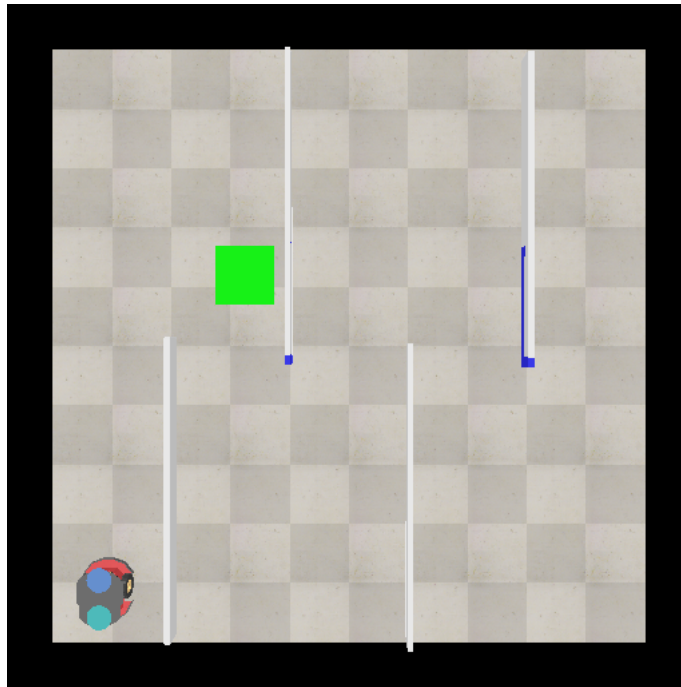
Por fim, em posse da variável de esterçamento, atribui-se essa tanto ao atuador direito quanto ao esquerdo, contudo, com sinais trocados. A velocidade atuador também é função de um ganho K_p , que auxilia na calibração da sensibilidade dos sensores.

3.2 Graduação do caminho: A*

3.2.1 Visão Computacional

Para o desenvolvimento desse algoritmo, utilizou-se de uma imagem da cena da simulação, capturada com o auxílio de uma câmera posicionada no ambiente de simulação, que será tratada com bibliotecas de visão computacional e utilizada para a elaboração da trajetória do robô. A malha obtida através da imagem retorna “1” para a área sem obstáculos e “0” para a área com obstáculos.

Figura 10: Imagem da cena utilizada para a simulação do algoritmo Follow the Carrot.



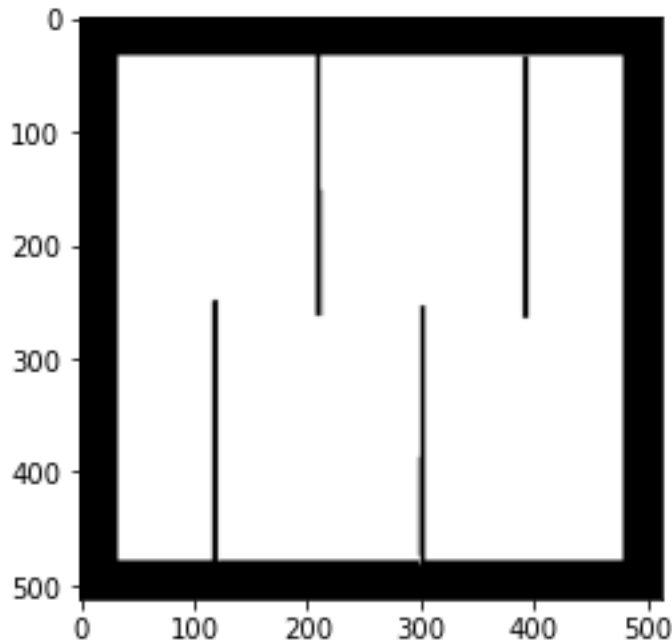
fonte: Elaborado pelo autor

mapeou-se, para essa aplicação, as seguintes bibliotecas:

```
1 import simConst
2 import cv2
3 import numpy as np
4 import sim
5 import time
```

Para o algoritmo, utilizamos das funções "imread" e "cvtColor", da biblioteca "CV2", para importar a imagem da cena de simulação para o ambiente de desenvolvimento Python. Em seguida, delimitou-se, através da leitura de cores da imagem, qual é a grade de obstáculos observada no ambiente de simulação.

Figura 11: Cena de simulação tratada com visão computacional.



fonte: Elaborado pelo autor

Utilizou-se de 2 discos posicionados na parte frontal e traseira da plataforma para a obtenção da orientação do Pioneer, que possibilitam definir tanto a distância para o ponto objetivo quanto a direção que o robô está seguindo. Essa informação será medida constantemente para alimentar o sistema de controle utilizado no modelo.

3.2.2 Definição da melhor trajetória

A lógica por trás do algoritmo é que ele combina as informações que o Algoritmo de Dijkstra usa (favorecendo os vértices que estão próximos do ponto de partida) e as informações do Greedy-Best-First-Search usa (favorecendo os vértices que estão próximos da meta). O algoritmo Greedy Best-First-Search funciona de maneira semelhante, exceto que tem alguma estimativa (chamada de heurística) de quão longe do objetivo qualquer vértice está. Em vez de selecionar o vértice mais próximo do ponto inicial, ele seleciona o vértice mais próximo da meta. Não há garantia de que o Greedy Best-First-Search encontrará o caminho mais curto. No entanto, ele é executado muito mais rápido do que o Algoritmo de Dijkstra porque usa a função heurística para guiar seu caminho em direção ao objetivo muito rapidamente.

Na terminologia padrão usada ao falar sobre A*, $g(n)$ representa o custo exato do caminho do ponto de partida até qualquer vértice n e $h(n)$ representa o custo estimado heurístico do vértice n até o objetivo. Cada vez que passa pelo loop principal, ele examina o vértice n que possui o menor $f(n) = g(n) + h(n)$.

Para a aplicação desse algoritmo, utilizou-se a mesma metodologia da seção anterior, com desenvolvimento em Python e API em Coppelia Robotics. A plataforma robótica utilizada para as simulações continua como sendo o Pioneer.

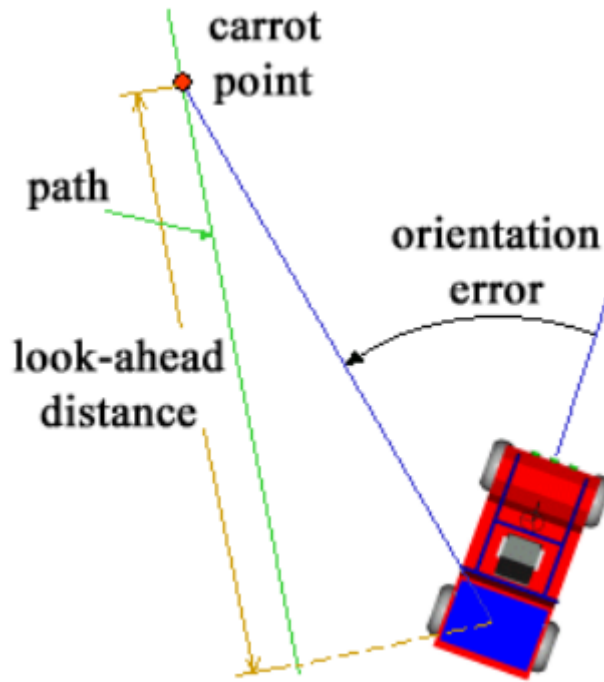
Podemos, assim, a melhor trajetória disponível para a malha de pontos obtida através do trabalho de visão computacional realizado de maneira prévia.

3.2.3 Follow the Carrot: percorrendo a trajetória

O rastreamento da trajetória ao qual o robô irá percorrer é um processo baseado na determinação da velocidade e direção em que esse precisa apresentar em cada instante de tempo, segundo. A trajetória, por sua vez, consiste de um conjunto de pontos que representam as coordenadas da rota. LUNDGREN (2003)

O algoritmo Follow de Carrot baseia-se em encontrar um ponto alvo e, após chegar nesse ponto alvo, mirar no próximo ponto, e assim sucessivamente.

Figura 12: Esquemático do método



fonte: Lundgreen.

Uma linha é desenhada do referencial/centro de coordenadas do robô perpendicularmente à trajetória. Assim, o carrot point, ou ponto objetivo, é definido como sendo o ponto no caminho à frente da distância de interseção desse ponto com a linha. O parâmetro mais importante é o erro de orientação, definido como o ângulo entre a direção atual ao qual o robô está apontando e a linha desenhada do centro de coordenadas do veículo ao ponto objetivo. Então, através de uma lei de controle PID, focamos em minimizar esse erro de orientação, sendo que o erro de orientação igual a 0 mostra que o veículo está apontando na direção exata rumo ao ponto objetivo.

$$\phi = K_p * E_o \quad (2)$$

Onde K_p é o ganho proporcional e e_0 é o erro de orientação.

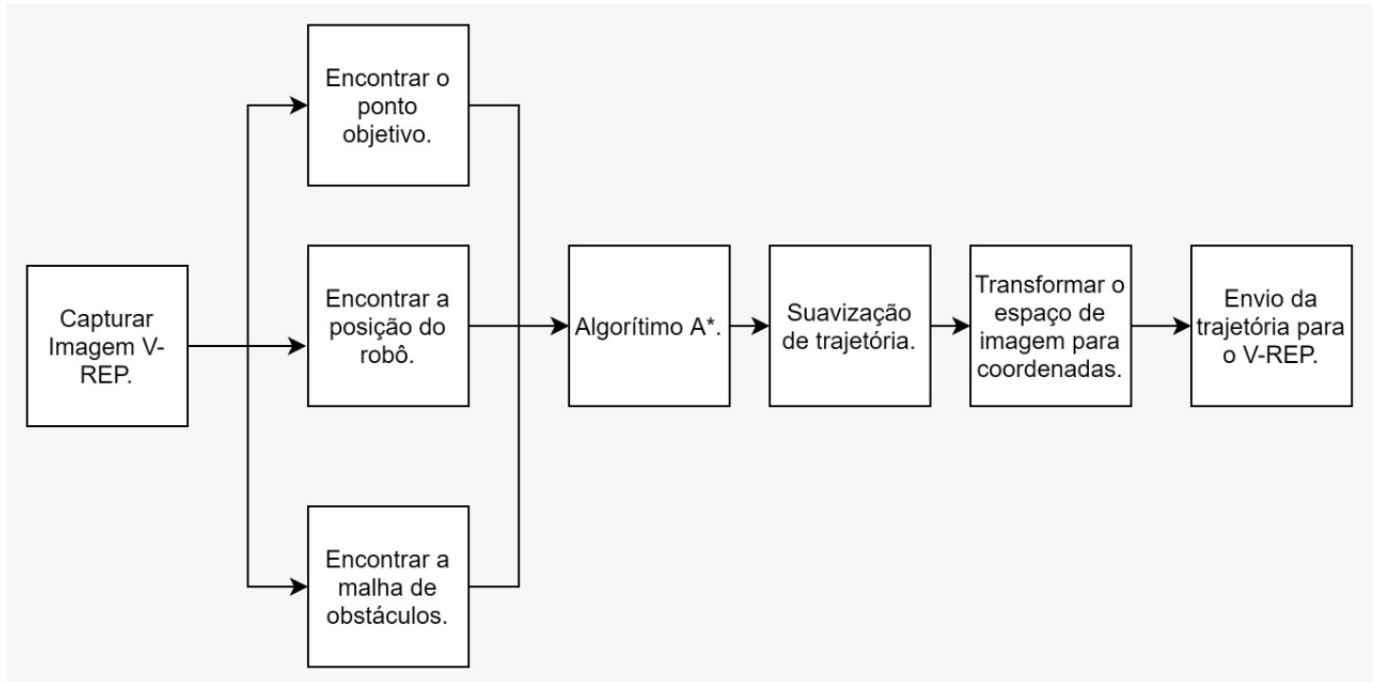
3.2.4 Diagramas do sistema

Assim, demonstramos como encontrar a melhor trajetória para alcançar a localização objetivo da plataforma, através do Algoritmo A*, além de como percorrer a mesma, utilizando-se da técnica Follow the Carrot. Podemos, então, esquematizar o sistema desenvolvido em

duas etapas:

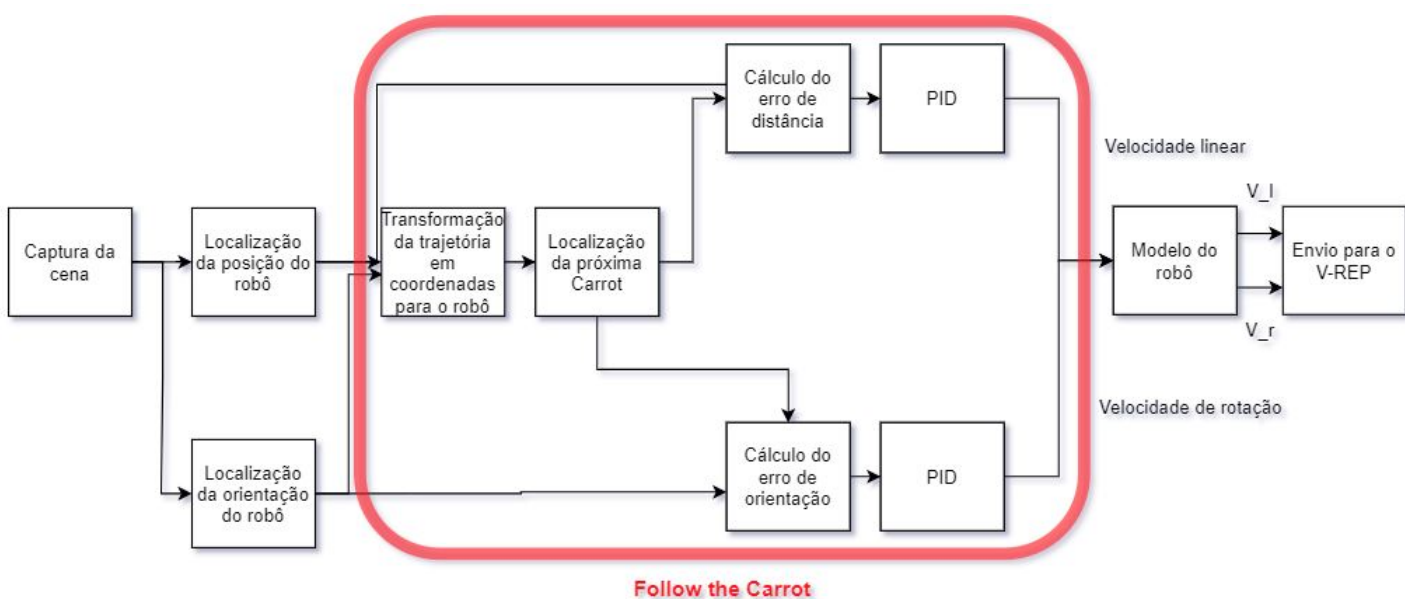
Primeiro realizamos a obtenção de uma imagem do V-REP, da qual obtemos a posição do robô, a rede de obstáculos e a posição objetivo, então realizamos o planejamento e suavização da trajetória, transformamos a trajetória de imagem para coordenadas no simulador, por fim, enviamos os pontos dessa trajetória para serem desenhados no software.

Figura 13: Planejamento de Trajetória realizado no começo do código.



fonte: Elaborado pelo autor.

Figura 14: Sistema de controle realizado toda em toda iteração do código.



fonte: Elaborado pelo autor.

De maneira geral, no ciclo responsável pela movimentação do robô, pegamos uma nova

imagem da cena de simulação, da qual obtemos a posição e orientação do robô, então transformamos a trajetória para coordenadas do Pioneer, encontramos o ponto localizado no próximo "look ahead distance", encontramos o erro de orientação e distância, que alimentam os controladores PID, que controlam a velocidade rotacional e linear do robô. Por fim, o modelo do robô é utilizado para obtenção da velocidade de cada atuador, o que é passado para o V-REP, para a movimentação do mesmo. Esse ciclo é realizado até que a localização objetivo esteja próximo da plataforma móvel.

4 Resultados

4.1 Simulações: Algoritmo de Braitenberg:

Assim, o modelo depende de 3 principais parâmetros arbitrários, sendo eles: velocidade base, ganho de esterçamento e tempo de simulação.

Realizou-se, então, diferentes simulações alternando entre diferentes cenários de parâmetros:

Cenário 1: $v=1.5$; $t=40s$

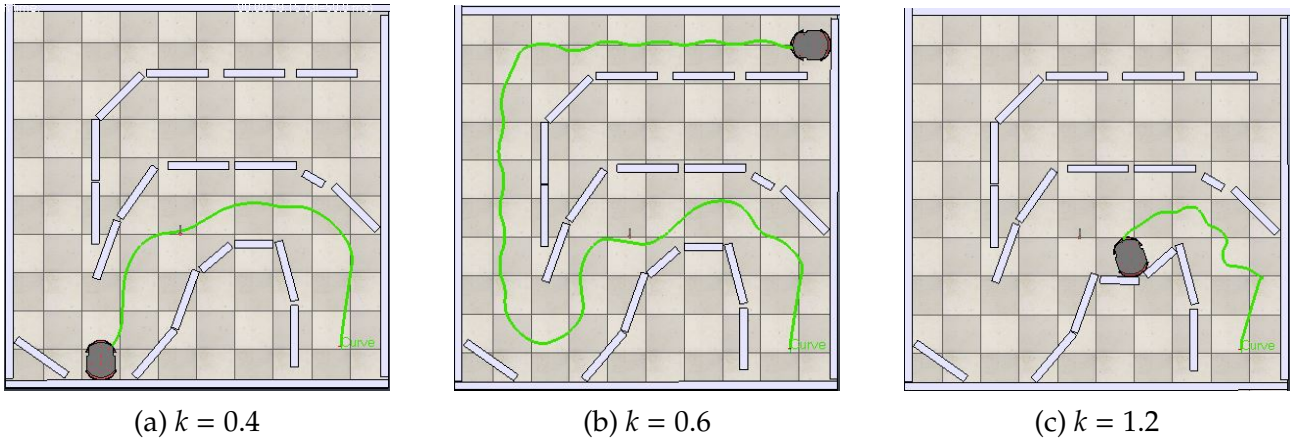


Figura 15: Simulações considerando a variação do ganho de esterçamento.
fonte: Elaborado pelo autor.

Cenário 2: $k=0.6$; $t=40s$

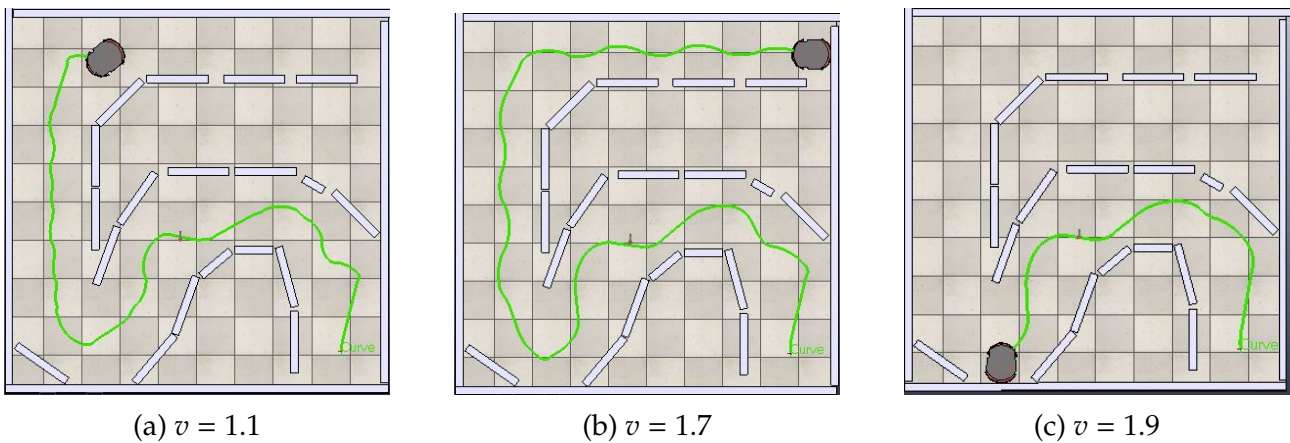
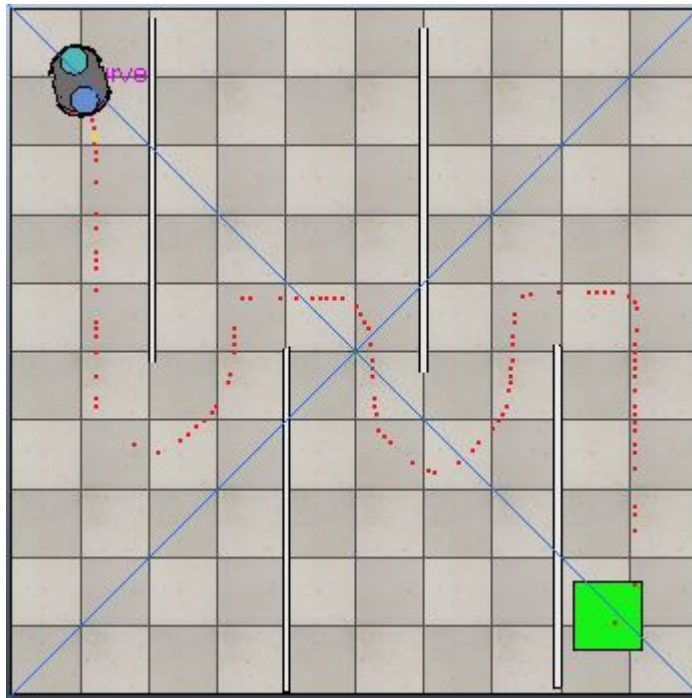


Figura 16: Simulações considerando a variação da velocidade base.
fonte: Elaborado pelo autor.

Podemos observar, com essas simulações, o efeito de cada componente no controle da plataforma e a sensibilidade da resposta do mesmo, sendo possível, assim, visualizar os principais parâmetros envolvidos no desenvolvimento de um algoritmo que evita obstáculos e os princípios e dificuldades relacionadas ao mesmo.

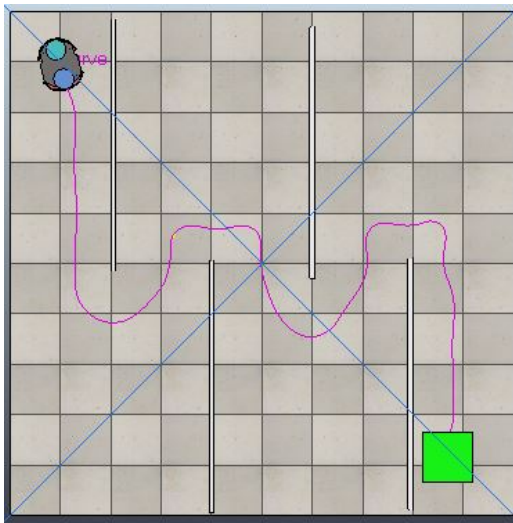
[illegible]

fonte: Elaborado pelo autor.

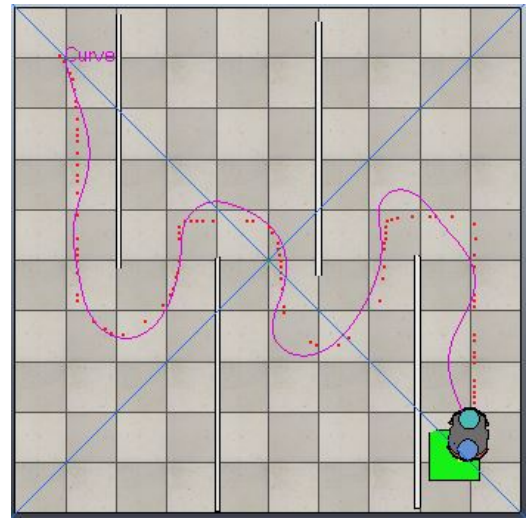
4.2 Simulações: Algoritmo Follow the Carrot

Podemos, à partir da imagem à seguir, visualizar a trajetória planejada através dos *carrots* plotados na cena da simulação:

Enfim, com posse da trajetória planejada e do modelo do robô, realizou-se a calibragem dos controladores de velocidade rotacional e linear do robô - $w(k_p, k_i, k_d)$ representa os ganhos para o controlador angular e $v(k_p, k_i, k_d)$ representam os ganhos para o controlador linear.



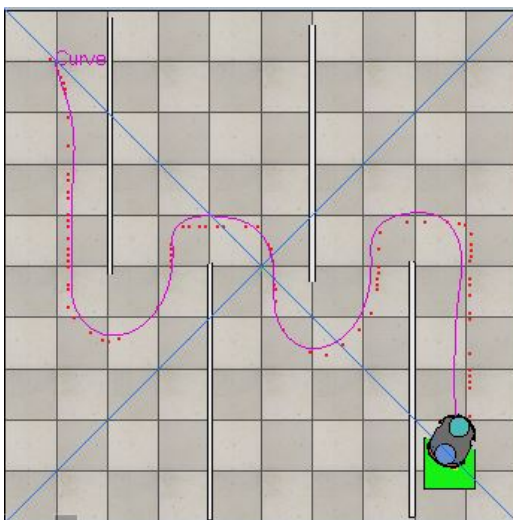
(a) $v(0.3, 0, 0)$; $w(0.3, k_0, 0)$



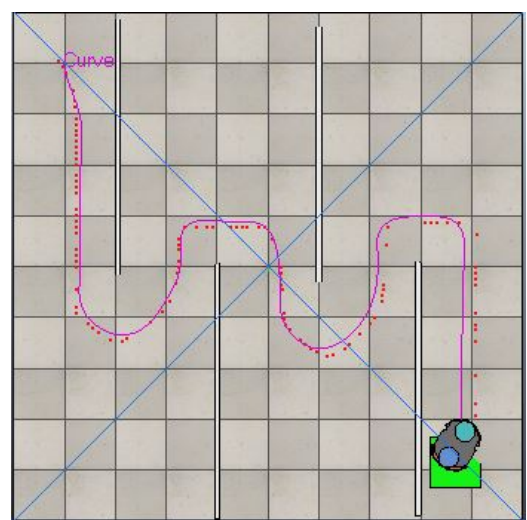
(b) $v(0.4, 0, 0)$; $w(0.3, k_0, 0)$

Figura 18: Simulações para calibragem empírica dos controladores proporcionais. A linha roxa representa a trajetória percorrida pelo robô

fonte: Elaborado pelo autor.



(a) $v(0.3, 0, 0)$; $w(0.3, k_0, 0.1)$



(b) $v(0.3, 0, 0)$; $w(0.3, 0, 0.2)$

Figura 19: Simulações para calibragem empírica dos controladores derivativos.

fonte: Elaborado pelo autor

Foi possível calibrar o controlador linear à partir de somente um de seus ganhos, o proporcional, o qual já possibilitou um controle satisfatório do modelo apesar de manter-se zerado o ganho derivativo e integrativo. Portanto, para o controlador linear podemos manteu-se somente o controle proporcional.

Com relação ao controlador angular, foi necessária a utilização de somente 2 controladores, o proporcional e o diferencial, para a obtenção de resultados satisfatórios. Para a

calibragem de seus respectivos ganhos, utilizou-se uma abordagem empírica, na qual primeiro é definido o ganho proporcional necessário para o alcance do *carrot* objetivo, mesmo que isso implique em oscilações do sistema em torno deste ponto, e, de maneira posterior, com o ganho do controlador proporcional estático, variou-se o ganho do controlador diferencial à fim de buscar aquele que proporcione a melhor resposta para o sistema, evitando as oscilações e possibilitando uma navegação mais estável para o modelo. Exemplificamos, nas imagens acima, alguns dos cenários simulados para a calibragem desses parâmetros.

4.3 Tabela comparativa.

Sumarizou-se, por fim, a prospecção de informações relativas aos softwares de simulação abordados em uma tabela comparativa relativa, na qual foi elencada as principais temáticas e entraves relacionadas ao desenvolvimento de projetos de simulação robótica.

Com relação aos motores físicos, todos apresentam-se como equivalentes, apresentando diferenças em aplicações que envolvem dinâmicas mais específicas que viabilizam mais determinadas engines. Na categoria API, todos apresentaram-se equivalentes, oferecendo, assim, suporte às principais linguagens de programação da atualidade.

Na categoria modularidade, dá-se destaque ao ROS, que viabiliza, através de sua estrutura de nós, elevada modulação das diferentes partes do projeto, fornecendo ferramentas importantes para o desenvolvimento conjunto desse tipo de projeto.

Relativo à assistência técnica, o VREP destaca-se por apresentar como software pago com assistência especializada além de uma comunidade de usuários ativa e participante em fóruns relacionados. A AWS, por sua vez, apresenta assistência técnica especializada, contudo, ainda está em processo de desenvolvimento de sua comunidade. Por fim, o ROS trata-se de um software open source sem assistência técnica disponível, contudo ainda conta com uma comunidade de usuários muito ativa nos fóruns relacionados.

Destaca-se, na categoria de implantação e testes das simulações, a AWS, que traz a melhor experiência de monitoramento dos dados aliado à integração com outras ferramentas de analytics, como inteligência artificial. V-REP e ROS encontram-se em patamares parecidos.

Na categoria de escalabilidade e demanda operacional, destacamos a ferramenta da Amazon, que apresenta a possibilidade de escala do processamento e monitoramento dos dados aliados ao incremento marginal dos custos relativos à operação.

	MOTORES FÍSICOS	APIS	MODULARIDADE	ASSISTÊNCIA TÉCNICA
VREP	3	3	2	3
ROS	3	3	3	1
AWS	3	3	2	2

(a) tabela comparativa parte 1.

	IMPLANTAÇÃO/TESTES	ESCALABILIDADE	DEMANDA OPERACIONAL
VREP	2	2	1
ROS	2	2	2
AWS	3	3	3

(b) tabela comparativa parte 2.

SOFTWARE	MÉDIA
VREP	2,3
ROS	2,3
AWS	2,7

(c) Notas médias finais de cada software.

Figura 20: Sumário comparativo dos softwares.
fonte: Elaborado pelo autor.

5 Conclusão

Pode-se, durante o desenvolvimento desse trabalho de conclusão de curso, observar os principais marcos relacionados à elaboração de simulações de plataformas robóticas móveis, destacando, em cada uma delas, o que tem sido utilizado pela comunidade científica e exemplificando, através do V-REP, alguns desses tópicos.

Obteve-se êxito no desenvolvimento do algoritmo de navegação focado em evitar obstáculos, atendendo as expectativas da criação de um código introdutório ao contexto das simulações robóticas que, contudo, ainda apresente conceitos interessantes, como a modelagem do robô, conceitos relacionados ao posicionamento dos sensores e a sensibilidade do modelo às variações de seus principais parâmetros de controle. Abordou-se, também, a história e os conceitos por trás da elaboração do Algoritmo de Braitenberg, fornecendo o contexto e as necessidades relacionadas à elaboração desse código.

Além disso, utilizou-se da combinação de diversas áreas para a implementação do algoritmo *Follow the Carrot*, agregando conhecimentos de visão computacional, com a utilização da biblioteca CV2, conceitos de planejamento de trajetórias, por meio da metodologia A* e o controle de sistemas por PID, utilizados para a navegação da plataforma robótica sobre a trajetória pré-estabelecida. Explorou-se, por fim, a sensibilidade dos ganhos dos controladores no desempenho do modelo durante as simulações.

Com relação as entraves relacionados ao desenvolvimento de projetos robóticos, tais quais plataformas móveis autônomas, destacam-se três: (i) Acesso ao espaço físico, que pode ser entendido como galpões, bancadas, cenários para validações dos projetos, etc. (ii) Equipe de desenvolvimento multi-disciplinar e, finalmente, (ii) acesso aos componentes eletro-mecânicos, tais quais hardware, software e atuadores.

Sabemos que o acesso a um espaço físico limita as contribuições ao projeto às pessoas que tem acesso a esses ambientes, o que vai em contra-mão da tendência de globalização e colaboração de pesquisadores que encontram-se em diferentes partes do mundo, além de ser uma grande barreira de entrada para aqueles que ainda não possuem o capital inicial necessário para alocar nesses passivos.

Podemos destacar, também, que as equipes multi-disciplinares necessárias para esses projetos apresentam cada vez mais a necessidade de modularização do projeto, com foco em tornar independente seu desenvolvimento, através de pequenas partes que interagem entre si e que possibilitem seu processo de avaliação e melhorias independentes.

Sabemos, ainda, que componentes eletro-mecânicos são, em sua maioria, importados. Assim sendo, esses estão expostos à variação cambial e à oferta dos mesmos pelo mercado externo, o que dificulta o planejamento dos projetos e torna, em muitas vezes, inacessível diversos componentes necessários para os projetos.

Assim, quando tratamos esses problemas em conjunto, podemos evidenciar um cenário no qual a escassez de recursos e pessoas apresentam-se como gargalos no planejamento e desenvolvimento de projetos robóticos, fazendo necessário o uso de mecanismos que atenuem ou solucionem essas problemáticas.

Nesse contexto, a simulação em nuvem, como a AWS RoboMaker, explorada nesse tra-

balho, apresenta-se cada vez mais como um democratizador do acesso aos conceitos e ferramentas necessários para o desenvolvimento desse tipo de projeto e um viabilizador multiplataformas dessas simulações em ambientes reais, fornecendo escalabilidade e reprodutividade aos mesmos.

Como próximas etapas, pode-se destacar a análise das variações de outras etapas relacionadas ao código *Follow the Carrot*, como os parâmetros utilizados para o planejamento de sua trajetória, ou da visão computacional, incluindo, com isso, objetos móveis na cena de simulação, buscando reformular constantemente a melhor trajetória possível. Além disso, pode-se implementar essa simulação em ambiente de nuvem, consolidando os avanços que essa traz no enfrentamento das dificuldades apresentadas anteriormente e explorando as possibilidades que surgirão com isso.

Referências

- [1] MARTINS, H. B., Desenvolvimento de Sistema de Simulação dedicado à Plataforma Robótica Mirã 2.0
- [2] EMBRAPA. Informações gerais sobre a unidade de soja. Disponível em: <https://www.embrapa.br/>. Acesso em: (05/2021).
- [3] SANCHES, R. M., Desenvolvimento de um sistema de planejamento de trajetória para veículo autônomos agrícolas.
- [4] OSÓRIO, F. S., BERRI, R. A., Simulação de Robôs Móveis e Articulados: Aplicações e Prática.
- [5] YOUNG, H. ARDEE, A General Agricultural Robotic Development and Evaluation Environment. 2018. Dissertação (Mestrado) – University of Illinois, Urbana, 2018.
- [6] ROS, 2021. Disponível em: <http://wiki.ros.org/ROS/Concepts>. Acessado em: (07/2021).
- [7] ROS, 2021. Disponível em: <http://wiki.ros.org/Nodes>. Acesso em (07/2021).
- [8] ROS, 2021. Disponível em: <http://wiki.ros.org/Topics>. Acesso em (07/2021).
- [9] ROS, 2021. Disponível em: <http://wiki.ros.org/Services>. Acesso em (07/2021).
- [10] ROS, 2021. Disponível em <http://wiki.ros.org/Messages>. Acessado em: (07/2021).
- [11] ROS, 2021. Disponível em <http://wiki.ros.org/Master>. Acessado em: (07/2021).
- [12] ROS, 2021. Disponível em <http://wiki.ros.org/Parameter%20Server>. Acessado em: (07/2021).
- [13] Azure, 2021. Disponível em <https://azure.microsoft.com>. Acesso em: (07/2021)).
- [14] Gazebo, Disponível em <http://gazebo.org/features>, Acessado em (07/2021).
- [15] OpenDE, Disponível em <http://opende.sourceforge.net/>, Acessado em (07/2021).
- [16] Simbody, disponível em <https://simtk.org/projects/simbody/>, acessado em (07/2021).
- [17] DART, Disponível em <http://dartsim.github.io/>, Acessado em (07/2021).
- [18] Coppelia Robotics, disponível em <https://www.coppeliarobotics.com/helpFiles/index.html>. Acessado em (07/2021).
- [19] ODE, disponível em <http://www.ode.org/>. Acessado em (08.2021)
- [20] Vortex Studio <https://www.cm-labs.com/vortex-studio/>

[21] <http://newtondynamics.com/forum/newton.php>

[22] AWS Robomaker, 2021. Disponível em: [docs.aws.amazon.com/robomaker/latest/dg/what-is-robomaker.htm](https://docs.aws.amazon.com/robomaker/latest/dg/what-is-robomaker.html).