UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

SOLY WALTRICK ANTUNES NETO

PROPOSTA DE MODELO DE COORDENAÇÃO DE VEÍCULOS SEGUIDORES DE LINHA UTILIZANDO UMA ABORDAGEM DE SISTEMAS MULTIAGENTES

Joinville
2022

SOLY WALTRICK ANTUNES NETO

PROPOSTA DE MODELO DE COORDENAÇÃO DE VEÍCULOS SEGUIDORES DE LINHA UTILIZANDO UMA ABORDAGEM DE SISTEMAS MULTIAGENTES

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville..

Orientador: Prof. Dr. Benjamin Grando Moreira

Joinville
2022

Dedico este trabalho aos meus pais, meu irmão e a Beatriz Shoji.

Tudo é possível. O impossível apenas toma mais tempo a atingir (BROWN, 2009).

RESUMO

Os veículos auto guiados são robôs móveis programáveis, unindo sistemas de automação com a mobilidade, capazes de transportar objetos. Contudo, a integração de múltiplos veículos autônomos em um ambiente exige que ocorra a cooperação. Sendo introduzido o uso de sistemas multiagentes e protocolos de comunicação, capazes de executar diálogos e alcançando os objetivos individuais e globais. Neste trabalho é proposta uma estratégia de coordenação para o deslocamento de veículos seguidores de linha, utilizando estratégias de sistemas multiagentes. É apresentado o estudo da viabilidade da utilização do *Bluetooth* como método para detectar outros veículos. Além disso, é realizado o levantamento das máquinas de estados e diálogos necessários para construir os agentes dentro do ambiente. Também são definidas as condições que podem causar impasses durante a execução de percursos pelos veículos. Por fim, é apresentada a simulação utilizada para comprovar a estratégia proposta e discutir sobre os resultados.

Palavras-chave: Linguagens de Comunicação. Planejamento de Rotas. Algoritmos de Busca.

ABSTRACT

Automated guided vehicles are programmable mobile robots, combining automation systems with mobility, capable of transporting objects. However, the integration of multiple autonomous vehicles in an environment requires cooperation. Introducing the use of multi-agent systems and communication protocols, allowing dialogues and reaching the individual and global objectives. In this work, a coordination strategy is proposed for the routing of line-following vehicles, using strategies of multi-agent systems. A study of the feasibility of using Bluetooth as a method to detect other vehicles is presented. In addition, the finite state machines and dialogs necessary to build the agents within the environment are defined. The conditions that can cause conflicts during the execution of routes by vehicles are also defined. Finally, the simulation is used to prove the proposed strategy and the results are presented.

Keywords: Communication languages. Route Planning. Search Algorithms.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACL	*Agent Communication Language*
BLE	*Bluetooth Low Energy*
cfp	*call for proposal*
DARPA	*Defense Advanced Research Projects Agency*
FIPA	*Foundation for Intelligent Physical Agents*
GAP	*General Advertising Profile*
JID	*Jabber ID*
KIF	*Knowledge Interchange Form*
KQML	*Knowledge Query and Manipulation Language*
KSE	*Knowledge Sharing Effort*
OS	*Operational System*
RISC	*Reduced Instructions Set Computer*
RSSI	*Received Signal Strength Indicator*
SLAM	*Simultaneous Localization and Mapping*
SPADE	*Smart Python Multi Agent Development Enviroment*
UUID	*Universally Unique Identifier*
VAG	Veículo Auto Guiado
VAGs	Veículos Auto Guiados
XML	*Extensible Markup Language*
XMPP	*Extensible Messaging and Presence Protocol*

SUMÁRIO

1 INTRODUÇÃO

A robótica já é considerada um sucesso consolidado na produção industrial ao redor do mundo, desde a primeira implementação em ambientes industriais, em 1953, com o *Unimate* da companhia americana *Unimation* (PIRES, 2002). O uso e o desenvolvimento da robótica tem crescido exponencialmente. Durante os anos 2000, indústrias especializadas na confecção de braços robóticos — também conhecidos como manipuladores — faturaram cerca de 2 bilhões de dólares (SIEGWART; SCARAMUZZA; NOURBAKHSH, 2011).

Contudo, apesar de todo o sucesso, sistemas tradicionais de automação, os quais utilizam estruturas fixas ou braços manipuladores fixos, possuem uma desvantagem essencial: a limitação da mobilidade (WURMAN; ANDREA; MOUNTZ, 2008 apud SIEGWART; SCARAMUZZA; NOURBAKHSH, 2011). Em contrapartida, há a possibilidade de se reverter essa desvantagem através da utilização dos Veículos Auto Guiados (VAG) ou Auto Guided Vehicle (AGV).

Os VAGs são veículos móveis programáveis, utilizados na indústria para movimentação de materiais interiormente no ambiente fabril ou para o transporte de objetos em depósitos (DAS; PASAN, 2016 apud WURMAN; ANDREA; MOUNTZ, 2008). Sumariamente, são geralmente contextualizados como robôs móveis capazes de transportar objetos (LI A.C. ADRIAANSEN; POGROMSKY, 2011). Seu desempenho é definido pela capacidade de sua integração em um ambiente com autonomia, dispensando a intervenção humana direta (SIEGWART; SCARAMUZZA; NOURBAKHSH, 2011).

O uso de robôs móveis permite percorrer por espaços inexplorados e de difícil acesso para os humanos, semelhante ao robô móvel *Sojourner*, utilizado durante a missão *Pathfinder* em Marte, ou ao *MBARI's ALTEX AUV*, utilizado para missões abaixo do gelo ártico (SIEGWART; SCARAMUZZA; NOURBAKHSH, 2011). Entretanto, em ramos industriais, seu uso não contempla desbravar lugares inóspitos, mas sim, coexistir e cooperar com os humanos. Assim, sua utilização tem se estendido da área industrial para diversas outras, como saúde, construção, logística e residencial (LI A.C. ADRIAANSEN; POGROMSKY, 2011). Ademais, a indústria veicular prevalece como a maior consumidora de VAGs (YAGHOUB et al., 2012).

Pode-se observar a implementação dos VAGs em linhas de produção dos veículos do *Model S* da *Tesla* (WIRED, 2013); na área de saúde, auxiliando no transporte de alimentos, fármacos e equipamentos (ROBOTIC AUTOMATION, 2020); além da serventia na área militar, agrícola, comercial (FLIPSE, 2011), construção civil e, recentemente, a introdução a ambientes residenciais, com as empresas iRobot e

Friendly Robotics, criando robôs com aplicações domésticas, como higienização ou jardinagem (WURMAN; ANDREA; MOUNTZ, 2008).

Devido ao amplo crescimento da utilização de VAGs, os sistemas de automação necessitaram de maior complexidade. Dessa forma, sistemas multiagentes foram desenvolvidos a fim de solucionar efetivamente as possíveis problemáticas existentes em ambientes que possuíam dezenas ou centenas de veículos. Ou seja, sistemas multiagentes coordenam e auxiliam os veículos para alcançarem o destino ordenadamente (JENNINGS; BUSSMANN, 2003).

Embora o planejamento inicial de deslocamento de cada veículo preveja e evite situações nas quais dois veículos compartilhem o mesmo percurso simultaneamente, na prática, os veículos podem se deparar com imprevistos e não conseguirem cumprir a trajetória no tempo planejado. Sendo então, necessária a intervenção no deslocamento dos veículos a fim de evitar possíveis colisões.

A proposta apresentada neste trabalho não é descrever e simular funcionamento de um veículo e sim, a habilidade de vários veículos integrarem o mesmo sistema. Portanto, visa desenvolver e validar sistemas multiagentes, seguindo o padrão de comunicação *Foundation for Intelligent Physical Agents* (FIPA), capaz de adicionar e gerenciar veículos autônomos em um ambiente industrial, com o intuito de evitar colisões e bloqueios entre veículos.

É realizado o estudo da viabilidade do uso do *Bluetooth* como sistemas para detecção de proximidade, a definição dos diálogos e máquinas de estados dos agentes e do sistema que irão compor e os possíveis cenários de conflitos que podem ser encontrados. Por fim, é apresentado o desenvolvimento da simulação da proposta e seus resultados, demonstrando o funcionamento do ambiente por completo.

1.1 OBJETIVO

Avaliar uma proposta de comunicação para o deslocamento de veículos seguidores de linha em um ambiente onde diversos veículos realizam deslocamento simultaneamente a partir de sistemas multiagentes.

1.1.1 Objetivo Geral

O objetivo geral do trabalho é propor uma estratégia de coordenação para o deslocamento de veículos seguidores de linha utilizando sistemas multiagentes.

1.1.2 Objetivos Específicos

- Discutir a viabilidade técnica do uso do *Bluetooth* como método de localização;
- Analisar algoritmos de busca para planejamento de rota dos veículos auto guiados;

- Identificar possíveis conflitos na operação dos veículos auto guiados e definir políticas para solução;
- Simular os cenários e conflitos;
- Avaliar o resultado do modelo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são introduzidos os conceitos relacionados à desenvolvimento deste trabalho. Inicialmente são discutidos os VAGs, detalhando sobre suas diferentes classificações. Em seguida são abordados os métodos de planejamento de rotas utilizados. Após isso, são apresentados os algoritmos de busca, destacando os principais utilizados para planejamento de rotas de VAGs. Logo em seguida, é realizada a definição de agentes inteligentes e sistemas multiagentes. Por fim, é apresentada a definição de linguagem e atos de discurso, apresentando ferramentas para a implementação de comunicação formalizada entre agentes.

2.1 VEÍCULOS AUTO GUIADOS

Como descrito por Das e Pasan (2016) e Yaghoub et al. (2012), o primeiro VAG surgiu em 1953 e era puxado por um reboque, a partir de um fio aéreo em uma mercearia. Durante o final dos anos 50 e início dos anos 60, VAGs com o uso de reboques já eram utilizados por linhas de operações em indústrias e em depósitos (YAGHOUB et al., 2012). Com o desenvolvimento na área, os VAGs se tornaram altamente automatizados, capazes de realizar manuseamento complexo de cargas e navegação por outras formas.

VAGs tornam o sistema automático por decidirem os melhores percursos para cada veículo no ambiente (DAS; PASAN, 2016). Atualmente, as metas em desenvolvimento dos VAGs são manter ou melhorar a qualidade do transporte de matérias, promover segurança, produtividade, controle de inventário e melhorar condições de trabalho (YAGHOUB et al., 2012).

A principal característica dos robôs móveis é a sua capacidade de percepção sobre o ambiente a partir de sensores. Podem ser utilizados sensores infravermelhos, sonares, de luminosidade, de temperatura para identificar os estados do ambiente (PSCHEIDT, 2007). Segundo Siegwart, Scaramuzza e Nourbakhsh (2011) robôs móveis são constituídos de mecanismos de locomoção a partir do desenvolvimento de *hardware* e *software* para utilizar a percepção. De acordo com Pscheidt (2007), a classificação dos robôs pode ser definida conforme a anatomia, funcionalidade e tipos de controle:

- **Anatomia**: robôs são qualificados conforme a forma e o ambiente que transitam. Podem ser robôs aquáticos, terrestres ou aéreos;
- **Funcionalidade**: divididos pelo tipo de uso dos robôs. Classificam-se entre robôs pessoais (entretenimento), robôs de serviço (auxiliam numa tarefa), robôs de campo (atividades em ambientes perigosos) e robôs industriais (utilizados em linhas de produção);

- **Tipo de Controle**: podem ser divididos segundo a forma que são controlados. Quando há um usuário controlando todos seus movimentos são considerados teleoperados, quando o robô apenas recebe uma tarefa e a executa autonomamente, são considerados semi-autônomos e quando executam todas as suas tarefas sem interferência humana, são considerados autônomos ou auto guiados.

2.1.1 Classificação de veículos auto guiados

A classificação de VAGs é definida segundo a orientação dos veículos, a qual é determinada conforme o caso de uso. Vários fatores podem influenciar nas decisões, como frequência de transporte, instalações existentes, custo de instalação e possíveis expansões. Os três sistemas mais utilizados são: orientação via laser, orientação por fios ou faixas.

- **Orientação via laser**: o veículo é guiado por laser projetado no ambiente. É a opção mais flexível já que permite alterações de rota por *software*, porém suscetível a obstruções, decorrentes do bloqueio do laser. Em VAGs, a tecnologia de mapeamento por laser *Simultaneous Localization and Mapping* (SLAM) é amplamente utilizada em veículos da *SEER*, para transporte de cargas e diversos drones (SEER, 2022);
- **Orientação por fios ou faixas**: o veículo utiliza uma fita ou fio colocada sobre o chão do ambiente como forma de guia. As fitas podem ser magnéticas, ou uma fita colorida, preferencialmente de uma cor oposta ao do chão.

Para a existência de um sistema de VAG devem existir os veículos; o(s) sistema(s) de orientação e controlador do sistema, encarregados de incorporar o sistema de controle de veículo; sistema de controle do ambiente e controle *onboard* do veículo (YAGHOUB et al., 2012). O sistema de controle pode ser implementado de forma centralizada, no qual um computador ou controlador gerencia o percurso de todos os veículos, ou descentralizada, não sendo necessário um controlador para definir cada percurso.

2.1.2 Aplicações em campo dos Seguidores de linha

Robôs seguidores de linha são veículos terrestres sobre rodas que geralmente utilizam um determinado número de sensores de luz conforme a aplicação (ALMEIDA, 2016). São veículos auto-operantes que seguem uma linha ou fita presente no chão do ambiente (PAKDAMAN; SANAATIYAN; GHAHROUDI, 2010).

Grande parte dos estudos desenvolvidos em relação aos veículos seguidores de linha tem como foco o funcionamento e desenvolvimento do controle de movimento do veículo (ALMEIDA, 2016 apud PSCHEIDT, 2007). Outros têm como intuito o

desenvolvimento para competições (PAKDAMAN; SANAATIYAN; GHAHROUDI, 2010 apud SILVA; BITTENCOURT, 2003).

Os veículos seguidores de linha podem ser encontrado em indústrias, sendo que um exemplo de utilização está na linha de montagem do Tesla Model S, no qual o veículo utiliza como meio de orientação fitas magnéticas colocadas no chão de fábrica para realizar o transporte na linha de produção (WIRED, 2013). Também são utilizados para transporte e abastecimento de peças na planta de fábrica, com a orientação realizada por uma fita preta em um chão de cor oposta (VIANNA; FERREIRA; NUNES, 2014 apud VOLKSWAGEN, 2014).

A pesquisa em sistemas multiagentes de veículos auto guiados, geralmente, não determina o método de orientação utilizado pelos veículos estudados (DAS; PASAN, 2016). O melhor exemplo de um sistema utilizado em campo encontrado pelo trabalho foi o *Kiva System* — que em 2015 se tornou *Amazon Robotics* — no qual veículos utilizam códigos de barras impressos no chão do ambiente para se guiar e movimentar as estantes com os materiais pelo ambiente (FLIPSE, 2011 apud TOBE, 2016). Cada veículo conseguia definir a sua rota, evitando colisões e dividindo recursos com outros veículos.

2.2 ALGORITMOS DE BUSCA

Algoritmos de busca procuram encontrar a lista de ações necessárias, para a partir de um estado inicial, chegarem até o estado final, definido como meta da busca. A busca requer uma infraestrutura de dados para manter o controle, composto por: estado atual, estado anterior, ação necessária para ir para o próximo estado e o custo da ação (NORVIG; RUSSELL, 2013).

As estratégias para busca são divididas entre buscas cegas, não há nenhuma informação disponibilizada além do estado inicial, e buscas informada ou heurística, no qual é conhecida a posição da meta definida pela busca no espaço de estados (NORVIG; RUSSELL, 2013). Os dois algoritmos mais utilizados no planejamento de rotas dos VAG:

- **Busca A***: uma busca em árvore em que é avaliado o menor custo possível para se encontrar o nó desejado. Avalia a partir da soma do custo para alcançar o nó e o custo para ir até o nó objetivo (NORVIG; RUSSELL, 2013). Foi publicado por Peter Hart, Nils Nilsson e Bertman Raphael, em 1968, e obtém o melhor desempenho porque utiliza heurísticas para guiar a busca (RAMADANI; HALILI; IDRIZI, 2019). Norvig e Russell (2013) define a complexidade do algoritmo de busca é $O(b^{\epsilon d})$, onde b é o máximo erro absoluto e d a profundidade da busca;
- **Algoritmo de Dijkstra**: desenvolvido por Edsger W. Dijkstra em 1956 (RAMADANI; HALILI; IDRIZI, 2019). É um algoritmo de busca que funciona

encontrando o menor caminho até o nó de destino, encontrando a distância entre cada par de nós (GOYAL et al., 2014). Como não é uma busca guiada, precisa percorrer todos os nós, pois não há como saber se inicialmente a menor distância entre dois nós irá realmente encontrar o caminho de menor custo. Segundo Ramadani, Halili e Idrizi (2019), considerando que todos os nós tem a mesma prioridade, a complexidade do algoritmo é $O(V2)$, onde V é o número de nós.

2.3 PLANEJAMENTO DE ROTAS

Para que os VAGs possam encontrar seus percursos ideais no ambiente, deve-se realizar o planejamento de rota. O sistema *Kiva* reconhece o ambiente como uma grade bidimensional com todos os percursos possíveis e utiliza uma implementação padrão do algoritmo A* (WURMAN; ANDREA; MOUNTZ, 2008). Na pesquisa de Vivaldini et al. (2010) sobre empilhadeiras robóticas, é implementado o algoritmo Dijkstra em C/C++ para o planejamento inicial de rotas e durante a execução do percurso, utiliza-se o algoritmo A* a fim de encontrar a menor rota possível para o veículo, considerando o seu modelo cinemático. Draganjac et al. (2016) utiliza o algoritmo A*, considerando todos os obstáculos estáticos na área de trabalho durante o processo de busca pelo menor percurso.

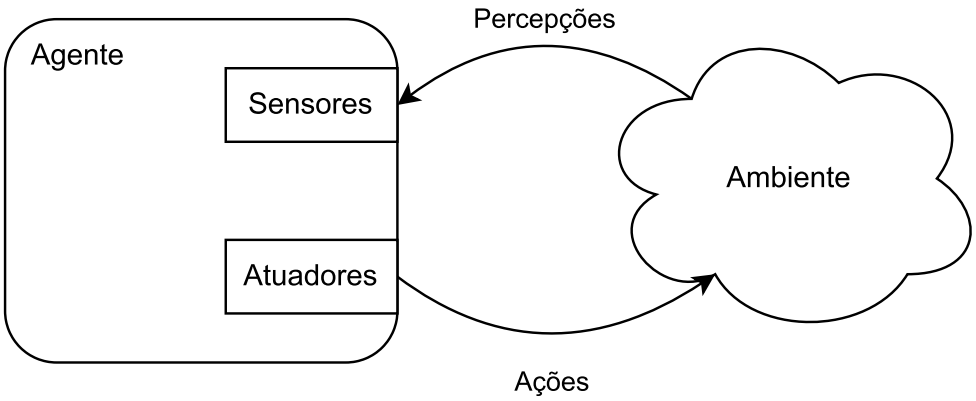
2.4 AGENTES INTELIGENTES

Com o intuito de definir entidades, que possam agir e possuir comportamentos, além de interagir sobre o ambiente, são utilizados agentes inteligentes (OSKOUEI; VARZEGHANI; SAMADYAR, 2014). Segundo Norvig e Russell (2013) um agente inteligente é tudo que pode de ser considerado capaz de perceber seu ambiente por meio de sensores e agir sobre o ambiente por meio de atuadores. Já Ferber (1999) define agentes como entidades autônomas que podem ser organizadas em uma comunidade e trabalhar para resolver problemas de complexidades diferentes.

De acordo com Bittencourt (2006) um agente é uma entidade capaz de agir sobre ela mesma ou sobre o seu ambiente, dispondo de uma representação parcial do ambiente. Em um mundo multiagente, é capaz de se comunicar com outros agentes e suas ações são devido às suas observações, ao seu conhecimento e às interações com outros agentes, conforme a Figura 1. A escolha das ações dos agentes em qualquer instante de tempo, só pode depender de uma sequência de percepções recebidas, um agente nunca deve tomar decisões considerando percepções não existentes (NORVIG; RUSSELL, 2013).

O comportamento de um agente é descrito pela função do agente, que mapeia uma ação específica para toda sequência de percepções. Ele pode ser mapeado em uma tabela que, caso não seja determinado em projeto, pode ter tamanho infinito.

Figura 1 – Agente



Fonte: Elaborado pelo autor

Caracterizar todas as ações a partir de percepções pela função do agente é conhecido como caracterização externa. Internamente, a função do agente é implementada pelo programa do agente (NORVIG; RUSSELL, 2013).

Para construir um agente, primeiro é necessário definir o ambiente de tarefa, ou seja, os problemas que os agentes irão solucionar. Ao projetar um agente, inicialmente, o ambiente deve ser descrito da forma mais completa possível. Sendo assim, o ambiente de tarefa do agente deve ser descrito pela medida de desempenho, ambiente, atuadores e sensores (NORVIG; RUSSELL, 2013):

- **Medida de desempenho**: são as qualidades desejáveis do agente, algumas vezes podem ser conflitantes. Utilizada para avaliar o comportamento;
- **Ambiente**: onde o agente será inserido para executar suas tarefas, também deve-se definir quais serão os possíveis obstáculos ou se existem outros agentes;
- **Atuadores**: meio pelo qual o agente exerce ações sobre o ambiente;
- **Sensores**: meio que o agente realiza percepções sobre o ambiente.

Pode-se dividir as dimensões que determinam o projeto apropriado dos agentes em categorias, segundo Norvig e Russell (2013):

- **Completamente observável x parcialmente observável**: um ambiente só é definido completamente observável quando um sensor permite total acesso ao ambiente a cada instante e todas as percepções são relacionadas a aspectos relevantes para escolha de uma ação. É conveniente tornar o ambiente completamente observável, pois o agente não precisa manter qualquer estado interno para guardar as mudanças do mundo. Quando um agente não possui nenhum sensor, o ambiente é definido como inobservável;
- **Agente Único x multiagente**: no caso de um agente único, este agente reconhece tudo que está a sua volta como objetos. Em sistemas multiagentes, cada agente pode enxergar outros agentes como agentes. Desta forma, pode existir a cooperação entre agentes a fim de realizar um objetivo global ou objetivos

únicos para cada agente. Porém, da cooperação, duas características podem surgir, quando um agente tenta maximizar a sua medida de desempenho, tendo como efeito, diminuir a medida de desempenho de outro agente, o ambiente é competitivo, ou quando os agentes procuram trabalhar para que todos maximizem suas medidas de desempenho, o ambiente é cooperativo, que ainda pode ser parcialmente competitivo, já que em algum instante de tempo, os agentes podem disputar pelo uso de algum recurso;

- **Determinístico x estocástico**: quando o próximo estado é completamente determinado apenas pelo estado atual e suas ações, o ambiente é chamado de determinístico, caso contrário, quando um estado pode sofrer alterações de acordo com situações que não há como prever, quando um ambiente é parcialmente observável, é chamado de estocástico. Ainda podemos chamar o ambiente de não determinístico, quando as ações são caracterizadas pelos seus resultados possíveis, não considerando probabilidade;
- **Episódico x sequencial**: quando a cada instante, o agente recebe uma percepção e executa uma ação de acordo, é conhecido como ambiente de tarefas episódico. É crucial que o instante seguinte não dependa das ações do episódio anterior. Quando um ambiente tem tarefas sequencias, as decisões atuais podem alterar as decisões futuras, desta forma o agente precisa pensar não apenas no instante;
- **Estático x dinâmico**: se o ambiente pode ser alterado devido às ações do agente, consideramos um ambiente dinâmico, caso contrário ele é estático;
- **Discreto x contínuo**: em ambientes discretos, existe um número limitado de percepções, estados e ações para um agente, entretanto, em ambientes contínuos, não existem limitações para as percepções, estados e ações;
- **Conhecido x desconhecido**: essa descrição não se refere ao ambiente em si, mas sim ao estado de conhecimento do agente para todas as suas ações e resultados derivados.

2.5 SISTEMAS MULTIAGENTES

Desenvolver sistemas complexos não envolve apenas criar sistemas com tarefas divididas, mas também sistemas capazes de usar protocolos e mecanismos de comunicação existentes (DONANCIO; CASALS; BRANDãO, 2019). Parte do desenvolvimento dos sistemas multiagentes envolve a integração dos agentes por serviço web (DONANCIO; CASALS; BRANDãO, 2019). Sistemas multiagentes com o uso de protocolos web, começaram a ser utilizados por volta dos anos 2000 (HUHNS; STEPHENS, 1998).

Sistemas multiagentes são uma generalização técnica dos agentes (WEISS,

2000). Em ambientes, há a possibilidade de que múltiplos agentes, dependentes ou não, troquem informações entre si para a realização dos objetivos (WEISS, 2000). Com a expansão das redes de computadores, há interconexão entre dispositivos, o que torna cada vez mais difícil a operação de forma isolada por um agente (WEISS, 2000).

Para a operacionalidade entre os agentes, deve-se prover um ambiente adequado, onde a infraestrutura entre agentes deve ser baseada em protocolos, que permitam que eles consigam executar conversas entre si com contexto e trocar mensagens estruturadas. Implementar protocolos permite a automação e sistematização das tarefas planejadas (SILVA, 2009)

Sistemas multiagentes não necessariamente deve ser um sistema homogêneo, podem existir vários agentes com ações e funções diferentes a fim de resolver os problemas propostos e alcançar os objetivos globais eficientemente (SILVA, 2009). Portanto, deve-se garantir que os sistemas multiagentes tenha um nível aceitável de coordenação.

2.5.1 Coordenação

A coordenação é uma propriedade de sistemas multiagentes executando uma tarefa em um ambiente compartilhado e acontece quando eles precisam realizar a comunicação entre si para realizar ações e comportamentos, atingindo sistemas mais coerentes (HUHNS; STEPHENS, 1998). Aguilar A. Rios e Cerrada (2013) define como um conjunto de atividades necessárias para uma comunidade de agentes poderem atuar coletivamente e complementa sendo necessário, pois: agentes precisam de informações, recursos são limitados, para otimizar os custos, evitando redundâncias e que todos os objetivos de agentes que sejam distintos sejam realizados. A coordenação gera ou a competição, ou a cooperação.

- **Competição**: como descrito nas propriedades do ambiente de um agente, em sistemas multiagentes, quando há a necessidade de disputar um recurso entre agentes, ou quando um agente precisa maximizar seu desempenho diminuindo o de outro, ocorre a competição. Para resolver a competição entre agentes, quando agentes procuram pelo mesmo resultado, deve-se realizar a negociação entre eles;
- **Coordenação**: quando vários agentes estão existindo em um ambiente, mas procurando realizar tarefas distintas ou procurando atingir um objetivo comum para todos, os agentes devem cooperar, planejando o uso dos recursos, seja de forma centralizada ou distribuída. Os agentes buscam resolver conflitos, garantir a sobrevivência e melhorar o rendimento de todos (AGUILAR A. RIOS; CERRADA, 2013).

2.6 LINGUAGEM

A comunicação entre agentes, diferente de outros sistemas de computação, é tratada em alto-nível por serem linguagens de programação muito próximas da linguagem humana. Existem dois fins para o ato de comunicação: compartilhar informações e coordenar atividades entre agentes. A linguagem pode ser considerada interna, quando há a intenção de representar o conteúdo da mensagem de alguma forma, ou externa, quando o interesse é enviar a mensagem para alguém. Quando existe conversa, deve-se garantir que não haja alterações nas ordens das mensagens (REIS, 2003). Um agente pode enviar e receber mensagens com dois tipos de mensagens básicas: asserções e perguntas (HUHNS; STEPHENS, 1998).

2.6.1 Atos de discurso

Segundo Reis (2003), Austin realizou a análise do discurso humano sobre as ações, pedidos, sugestões, compromissos, e respostas e é hoje um modelo para a comunicação dos agentes. Ações do discurso são executadas da mesma forma que qualquer outra ação e segundo o cumprimento de seus objetivos. Atos de discurso são semelhantes às ações, já que podem mudar o estado do mundo de forma análoga às ações (REIS, 2003).

Existem três aspectos: locução (ato do discurso), elocução (sentido atribuído à locução) e perlocução (efeito da ação resultante da locução).

Já Searle complementou a teoria, identificando condições de transmissão, preparatórias e de sinceridade (REIS, 2003). Além de criar as classes: diretivas, promessas, expressivas, representativas e declarativas. Para identificar a elocução utilizada para definir o sentido que deve ser interpretada a mensagem, foram definidas as performativas no Quadro 1.

2.6.2 Linguagens externas de comunicação

Com a necessidade da formalização da comunicação foram desenvolvidos protocolos para padronizar a troca de mensagens entre agentes, permitindo a troca de conhecimento (GLUZ; VICCARI, 2003). Tal protocolo é definido por emissor, receptor, linguagem utilizada, funções de codificação e ações (HUHNS; STEPHENS, 1998). Em 1990, foi fundado o *Knowledge Sharing Effort* (KSE) financiado pelo *Defense Advanced Research Projects Agency* (DARPA) nos Estados Unidos da América, visando desenvolver protocolos de comunicação para sistemas multiagentes. A instituição produziu duas linguagens: a *Knowledge Query and Manipulation Language* (KQML) e *Knowledge Interchange Form* (KIF) (REIS, 2003).

Quadro 1 – Tabela de Performativas

Performativa	Elocução	Resultado
Pergunta	Questionar	Resposta
Resposta	Informativo	Aceitação
Pedido	Pedido	
Explicação	Informativo	Acordo
Comando	Pedido	
Permissão	Informativo	Aceitação
Recusa	Informativo	Aceitação
Oferta/Factura	Informativo	Aceitação
Aceitação		
Concordância		
Proposta	Informativo	Oferta
Confirmação		
Retracção		
Negação		

Fonte: adaptado de Reis (2003).

- **KQML**: uma linguagem externa, que define o invólucro para formatar a mensagem que determina o significado locutório. Não existe preocupação em como o conteúdo é interpretado;
- **KIF**: uma linguagem interna, destinada a representar o conhecimento. Foi desenvolvida para representar os conteúdos da linguagem KQML.

Além dessa iniciativa, outras linguagens também foram desenvolvidas. A linguagem FIPA ACL, desenvolvida pela FIPA está entre as mais utilizadas atualmente (FIPA, 2002a). Seu desenvolvimento começou em 1995, foi apresentada em 1997, mas apenas disponibilizada em 2000 com o lançamento do FIPA-OS (GLUZ; VICCARI, 2003). Até então, a KQML era a única linguagem padronizada utilizada entre agentes, porém, as maiores dificuldades em usá-las eram devido às imprecisões da semântica de suas performativas (GLUZ; VICCARI, 2003).

A linguagem possui 20 normativas para definir a interpretação desejada à mensagem pelos agentes. Essas normativas ou performativas, são mais próximas da execução dos processos de negociação, facilitando a compreensão do sentido da mensagem sendo projetadas para corresponderem com a teoria dos atos de Searle (REIS, 2003). As normativas que podem ser utilizadas nesse trabalho estão definidas no Quadro 2.

Cada mensagem é composta por um identificador do tipo do ato comunicativo seguido de um conjunto de *slots* opcionais (GLUZ; VICCARI, 2003). Esses *slots* contêm, usualmente, o emissor, o receptor e o campo de conteúdo. Também há a possibilidade de criar *slots* adicionando “x-” no prefixo do novo *slot* (GLUZ; VICCARI, 2003). Assim como o KQML, é uma linguagem que não utiliza uma linguagem interna para representar

Quadro 2 – Tabela de Performativas FIPA ACL

Performativa	Significado
accept-proposal	Aceita proposta em uma negociação
agree	Aceita desempenhar uma dada ação
cfp “ Call for proposals”	Utilizada para iniciar uma negociação
confirm	Confirmação da veracidade de uma mensagem
disconfirm	Desconfirma a veracidade de uma mensagem
failure	Uma tentativa de executar uma dada ação (usualmente requirida por outro agente) que falhou
inform	Uma das performativas mais utilizadas da FIPA ACL. Comunica uma informação que deve ser considerada verdade
propose	Envio de proposta, por exemplo, como resposta a uma mensagem cfp
refuse	Recusa executar determinada ação
reject-proposal	Recusa de uma proposta feita no contexto de uma negociação
request	Uma das performativas mais importantes da FIPA ACL. Consiste em um pedido a um agente para executar uma ação
Subscribe	Pedido para ser informado sobre alterações relacionadas determinada ação ou informação

Fonte: adaptado de Reis (2003).

conteúdo (GLUZ; VICCARI, 2003).

2.7 PLATAFORMAS FIPA

Segundo Gluz e Viccari (2003) as principais plataformas de desenvolvimento FIPA são: FIPA-OS, JADE, LEAP, ZEUS e ADK. Todas essas linguagens são descritas em Java. Porém, existem dificuldades relacionadas a falta de opções de ambientes de programação. A linguagem JAVA dificulta construir agentes pela escolha da arquitetura, que utiliza apenas a arquitetura de máquina (máquina de pilha interpretada). Dessa forma, o desempenho das tarefas do agente pode ser inferior ao de outras arquiteturas mais complexas, como Reduced Instruction Set Computer (RISC) (FLYNN; FLYNN, 1995). Também surge a necessidade de traduzir todos os agentes preexistentes para JAVA ou que todos os agentes já sejam criados na linguagem.

2.7.1 SPADE

Outras plataformas foram projetadas utilizando o padrão da linguagem FIPA ACL, evitando a dependência do JAVA. Dentre elas, o SPADE, foi desenvolvido em 2005 na universidade politécnica de Valência (AGUILAR A. RIOS; CERRADA, 2013). Ele é definido pela troca de mensagens instantânea por meio de*Extensible Messaging and Presence Protocol* (XMPP)(DONANCIO; CASALS; BRANDãO, 2019).

XMPP é uma tecnologia *Extensible Markup Language* XML para comunicação em tempo real que possui várias aplicações como mensagens instantâneas, notificação de presença e colaboração (XMPP, 2006).

Um agente é composto pelo mecanismo de comunicação à plataforma, um distribuidor de mensagens e um distribuidor de tarefas recebidas pelo distribuidor de mensagens. Também cada agente necessita um Jabber ID (JID) e senha, para poder realizar conexão com a plataforma XMPP.

O SPADE define os processos que um agente pode executar suas tarefas ou comportamentos nos seguintes padrões: *one shot* (execução única do comportamento), *cyclic* (execução cíclica do comportamento), *periodic* (execução periódica do comportamento), *timeout* (tempo para parar a execução), máquina de estados (definidos por estados e transições) e baseados em eventos.

Sendo assim, a SPADE auxilia no desenvolvimento de agentes com o controle do ciclo de vida dos agentes, seus comportamentos e comunicação. Embora permita realizar a comunicação distribuída entre os agentes baseados no padrão FIPA-ACL, a SPADE não implementa os demais aspectos determinados para o padrão FIPA de agentes, como os protocolos de comunicação, cabendo ao programador garantir o funcionamento desses protocolos.

3 A PROPOSTA, MATERIAIS E MÉTODOS

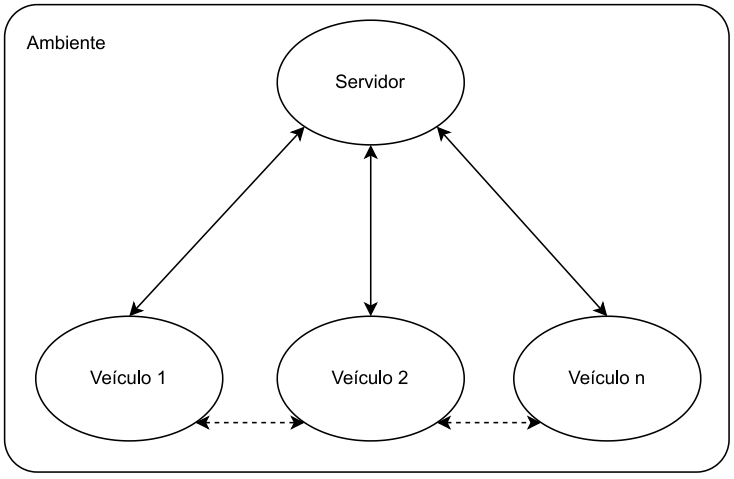
Este capítulo descreve a proposta realizada, as ferramentas utilizadas para o desenvolvimento da simulação e a aquisição dos resultados. A proposta deste trabalho consiste no desenvolvimento de uma estratégia de coordenação para que veículos possam percorrer um ambiente, exercendo suas tarefas sem que ocorram bloqueios e evitando atrasos. Com o uso de um sistema multiagentes, a tomada de decisões torna-se descentralizada, permitindo a independência entre as ações de cada veículo.

Desta forma, se faz necessário existir pelo menos um agente descrevendo cada veículo. Além disso, também existe a necessidade de outro agente que auxilie na comunicação entre agentes que definem os veículos, para facilitar na escalabilidade do ambiente. Este agente pode ser considerado um servidor, ou seja, é proposta a utilização de uma arquitetura como visto na Figura 2.

Assim, existe um ambiente com indicação dos caminhos com linhas que podem ser utilizados como percurso, possibilitando o deslocamento ponto a ponto dos veículos. Os veículos, quando requisitados, calculam a melhor rota e a submetem para aprovação do servidor. Após a aprovação, cada veículo começa seu deslocamento e, caso encontre algum conflito, reduz sua velocidade e se comunica com o servidor para decidir o que deve ser feito para resolver o conflito.

O programa responsável por realizar a simulação foi escrito em na linguagem de programação Python, na versão 3.9. Nele se faz o usado das bibliotecas SPADE, para realizar a comunicação entre os agentes, e Pygame, para visualização e detecção de colisões (PALANCA, 2020 apud PYGAME, 2009).

Figura 2 – Ambiente



Fonte: Elaborado pelo autor

O conflito é identificado quando as antenas *Bluetooth*, utilizadas pelos veículos,

detectam outro dispositivo com a mesma antena *Bluetooth*, dentro de um raio de ação de até 10 metros (KAMMER et al., 2002). Então, é requisitado informações ao servidor sobre quem está próximo. O servidor identifica os agentes em conflito, contando com a interação mútua entre os envolvidos, ou seja, que ambos realizaram a requisição e estejam executando o mesmo processo em simultâneo, ao se aproximarem.

Enquanto os veículos, junto ao servidor, tratam do conflito, eles devem sempre reduzir a sua velocidade, e assim que encontrarem a solução, podem voltar a velocidade normal de trabalho. As soluções para os conflitos são discutidas mais adiante neste capítulo.

3.1 VIABILIDADE TÉCNICA DO BLUETOOTH

Para encontrar colisões num modelo real, a implementação do Bluetooth Low Energy (BLE) foi avaliada para verificar a viabilidade técnica. A avaliação foi realizada considerando o uso de uma placa Arduíno com o uso da biblioteca ArduinoBLE, que habilita a conectividade de baixo consumo e baixas taxas de transferência, que pode operar com pequenas baterias de lítio (ARDUINOBLE, 2018).

Diferente do *Bluetooth* convencional - podendo alcançar distâncias acima de 140 metros em aplicações industriais, que age como uma comunicação serial - o BLE disponibiliza um quadro de informações nas quais, os dispositivos podem ter acesso, podendo ler ou alterar os dados disponíveis, por meio de serviços (SIG, 2021).

Cada serviço possui um endereço único, conhecido como *Universally Unique Identifier* (UUID). Um serviço que pode ser utilizado para encontrar proximidade entre dispositivos é o *General Advertising Profile* (GAP), que permite que dispositivos BLE notifiquem sua existência para outros pelo sinal de rádio e seu pacote possui uma palavra de 128 bits. Contudo, a utilização do serviço de notificação não define a distância entre os agentes e também pode não possuir os dados necessários para poderem trocar mensagens pelo servidor XMPP.

Para definir a distância entre os dispositivos conectados, se usa o *Received Signal Strength Indicator* (RSSI), o qual um é valor indicando a potência de sinal de rádio, que pode ser utilizado para encontrar a distância entre duas antenas (PARAMESWARAN; HUSAIN; UPADHYAYA, 2009). Parameswaran, Husain e Upadhyaya (2009) também destaca que o uso de RSSI não garante que a localização do componente possa ser encontrada por algoritmos de busca.

O RSSI é definido com dez vezes o logaritmo da razão entre a potência do sinal recebido e a potência de referência em decibéis, podendo ser simplificado pela Equação 1.

$$RSSI = -log(distance)\,. \tag{1}$$

O código que pode ser utilizado para encontrar dispositivos em um raio definido pela

distância conforme a potência no Listing 3.1.

```
#include <BLEAdvertisedDevice.h>
#include <BLEDevice.h>
#include <BLEScan.h>

const int DISTANCE = -60; // distance in log

void setup() {
  BLEDevice::init("");
}

void loop() {
  BLEScan *scan = BLEDevice::getScan();
  scan->setActiveScan(true);
  BLEScanResults results = scan->start(1); // get devices that were
   found
  int distance_max = DISTANCE; //set maximum distance that the sensor
   will trigger
  for (int i = 0; i < results.getCount(); i++) {
    BLEAdvertisedDevice device = results.getDevice(i); //get device from
    results
    int rssi = device.getRSSI(); //find distance from the signal
    if (rssi <= distance_max) {
        printf("Device found\n"); //if rssi distance is smaller than
   distance_max
    }
  }
}
```

Listing 3.1 – Código dispositivo BLE

É possível utilizar o BLE para detectar a proximidade de outros veículos, ajustando a distância a partir do potência do sinal do BLE detectado.

Embora os veículos identifiquem possíveis outros veículos na proximidade, seu reconhecimento dependeria de um mapeamento prévio de cada dispositivo BLE dos veículos. Com o uso da arquitetura proposta isso não é necessário, sendo o servidor utilizado para identificar os veículos a partir do aviso de que eles identificaram mutualmente.

3.2 AGENTES NO ESPAÇO DE TRABALHO

O processo deve ser dividido entre os agentes, compreendendo o cálculo de rotas, e a comunicação entre agentes para solucionar conflitos. O processo de movimentação e identificação do ambiente será abstraído, demonstrado como conceito

em simulação posteriormente. Para a execução do sistema proposto, serão utilizados dois tipos de agentes diferentes, um servidor e um ou mais veículos.

- **Servidor**: agente que mantém controle do ambiente e auxilia na comunicação entre agentes. Deve manter registro dos veículos que estão executando percursos no ambiente, verifica se há possibilidade de um veículo percorrer no ambiente, seja por limitação de rota ou de quantidade de veículos que já estão inseridos no ambiente e auxilia a solução de conflitos entre veículos ou objetos que não são agentes;
- **Veículo**: agente que descreve o veículo que deve percorrer um percurso dentro de um ambiente. Deve receber de um usuário um destino, calcular a rota até o destino desejado, pedir autorização para executar seu percurso, observar seu ambiente por obstáculos e caso encontre algum obstáculo, tentar resolver o conflito para conseguir chegar até o seu destino.

3.3 SERVIDOR

Como descrito anteriormente na Seção 3.2, o servidor será o agente capaz de lidar com todos os veículos no ambiente, uma vez que, cada veículo que deseja começar um percurso, deve inscrever sua tarefa no servidor e receber autorização para poder executá-la. O servidor pode possuir um limite de veículos que podem ser inscritos no ambiente, evitando que exista uma facilidade para conflitos.

3.3.1 Funcionamento do Servidor

O servidor mantém uma máquina de estados para cada veículo inscrito no ambiente, seguindo a ordem e observado no Apêndice A.

- **Estado 1**: aguarda um veículo pedir para ser inserido no ambiente, quando é confirmada a inserção, modifica o estado para o estado 2;
- **Estado 2**: aguarda a rota proposta do veículo, caso não encontre nenhum problema, avança para o estado 3, caso encontre um problema avisa o veículo e aguarda até receber a rota corrigida;
- **Estado 3**: veículo em percurso, aguarda a confirmação que o veículo chegou ao seu destino, caso receba um pedido de informação sobre uma possível colisão, verifica se algum outro veículo que está no mesmo estado também envia um aviso de colisão, caso encontre dois agentes nessa condição, repassa para o veículo com maior prioridade o endereço do outro veículo, para poderem executar a negociação, quando o veículo indica que terminou seu trajeto, passa para o estado 4;
- **Estado 4**: estado final, encerra a comunicação com aquele veículo e limpa todos os recursos utilizados por ele. Ao finalizar, retorna ao estado inicial, esperando

por novos veículos.

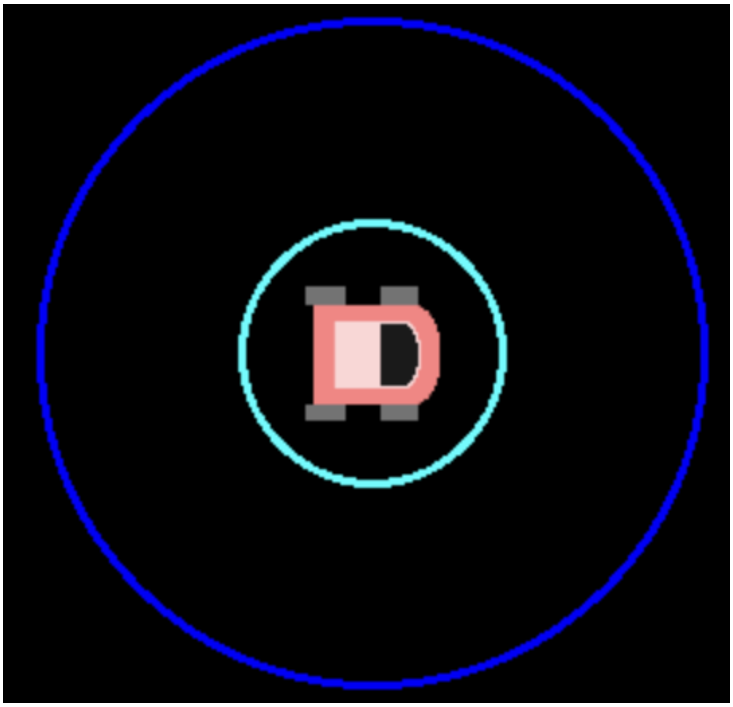
3.4 VEÍCULO

O veículo simulado representa um VAG seguidor de linha e suas características, como velocidade, carga e posição inicial, são definidas na inicialização do agente. Incrementando o peso da carga, resultará em redução na velocidade do mesmo.

No começo da execução do agente, deve ser inserida a posição de destino no terminal, o veículo então começa a comunicação com o servidor para poder se inscrever no ambiente e posteriormente, procura a melhor rota. Cada veículo possui uma representação do ambiente, possuindo detalhado todos os obstáculos conhecidos. Após a definição e aprovação da rota, o veículo então pode percorrê-lo para alcançar seu destino.

Enquanto está em movimentação no ambiente, o veículo observa por meio da antena *Bluetooth* se existe aproximação de outro agente, sendo caracterizada por um circunferência ao retor do veículo, indicado na Figura 3 pela circunferência externa. Objetos reconhecidos pela antena *Bluetooth*, indicam uma potência de sinal ao receptor, com isso, pode-se definir que existe um dispositivo dentro da área de alcance da antena do veículo, causando a requisição ao servidor de qual outro agente está próximo naquele momento. O raio de ação pode ser definido como parâmetro para o sensor, como demonstrado no Código 3.1.

Figura 3 – Representação do veículo em movimento



Fonte: Elaborado pelo autor

A circunferência com raio menor é a representação de um sensor de proximidade, idealizado para evitar colisões com objetos que não podem ser identificados pelo sensor *Bluetooth*. Seu comportamento não foi simulado nesse trabalho, tendo em conta que não existiria negociação, já que o objeto não seria parte do sistema multiagente.

Para representar o funcionamento do mapeamento e dos sensores de proximidade, a simulação deve conseguir ilustrar e representar o funcionamento e comportamento semelhante ao do veículo, O funcionamento deste agente é modelado pela máquina de estados mostrada no Apêndice B e cujo estados são:

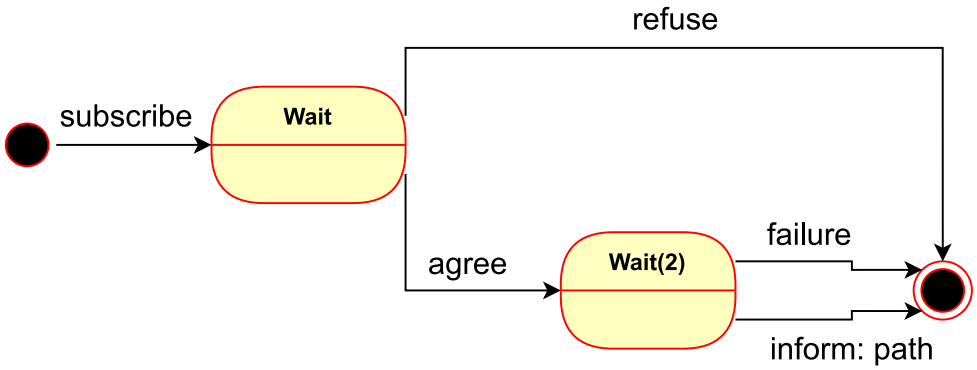
- **Estado 1(aguarda receber instruções)**: mantém uma posição estática no ambiente, ao receber as instruções muda seu estado para o Estado 2;
- **Estado 2**: ao receber o destino, calcula a melhor rota, com o conhecimento da planta do ambiente e muda para o Estado 3;
- **Estado 3**: com a rota definida, envia a requisição para inscrição ao servidor. Caso receba uma afirmação, segue para o Estado 4, caso receba uma negativa, tenta novamente após um período;
- **Estado 4**: com a autorização para percorrer no ambiente, envia ao servidor e aguarda a confirmação para seguir ao Estado 5, se receber uma negativa, recalcula sua rota utilizando os trechos bloqueados recebidos na mensagem, após recalcular, repete o estado;
- **Estado 5**: estado que define a movimentação do veículo. Nesse estado mantém observação constante dos seus arredores e caso encontre algum objeto, parte para o Estado 6. Se conseguir chegar até o seu destino, avança para o Estado 9;
- **Estado 6**: estado para iniciar resolução de conflitos. Assim, o veículo reduz a velocidade e entra em contato com o servidor, indicando que está próximo de algum objeto. Caso receba uma informação indicando que está próximo de outro agente, parte para o Estado 7. Se não for identificado nenhum agente, parte para o Estado 8;
- **Estado 7**: estado para negociar com outro agente a solução do conflito. Assim que o conflito for solucionado, retorna para o Estado 5. A solução do conflito será discutida posteriormente;
- **Estado 8**: estado para evitar colisão com um objeto inanimado. Deve recalcular a rota considerando a posição do objeto desconhecido como obstáculo. Assim que possuir uma rota, deve executá-la com a velocidade reduzida. Ao perder contato com o objeto, deve retornar ao Estado 5;
- **Estado 9**: estado que indica que o agente finalizou o seu percurso. Envia uma mensagem ao servidor indicando a conclusão e retorna ao Estado 1.

3.4.1 Diálogo entre agentes

Silva, Bavaresco e Silveira (2008) descreve que o diagrama de diálogo entre agentes é formado por uma máquina de estados finita, definindo a comunicação entre agentes de duas classes. Os diálogos descritos devem seguir as performativas do protocolo FIPA, adaptados do *Request Interaction Protocol Specification* (FIPA, 2002b).

- **Inscrição e envio do caminho**: compreende os estados 2 e 3 apresentados na Seção 3.4. A Figura 4 demonstra o diálogo para inscrição do veículo no ambiente e envio do caminho ao servidor;

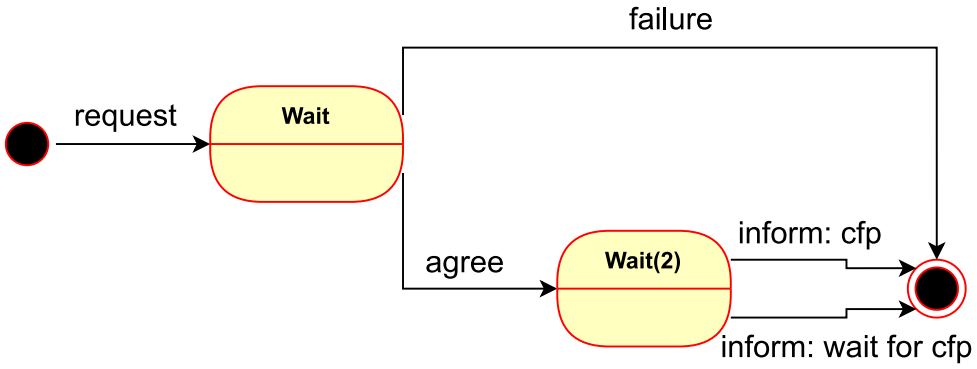
Figura 4 – Inscrição e envio do caminho



Fonte: Elaborado pelo autor

- **Requisição de informações sobre o conflito**: compreende o estado 6 e estado 7, apresentados na Seção 3.4. A Figura 5 demonstra o diálogo do veículo requisitando informações e a ordem da *call for proposal* (cfp);

Figura 5 – Requisição de informações sobre o conflito

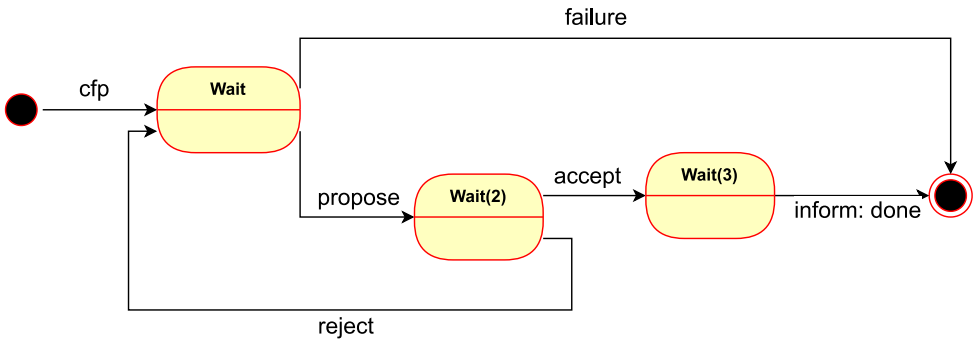


Fonte: Elaborado pelo autor

- **Proposta entre agentes**: compreende o estado 7 apresentado na Seção 3.4. A Figura 6 demonstra a execução da performativa cfp, requisitando uma proposta para a solução do conflito.

O exemplo da implementação em Python utilizando a máquina de estados do

Figura 6 – Proposta entre agentes



Fonte: Elaborado pelo autor

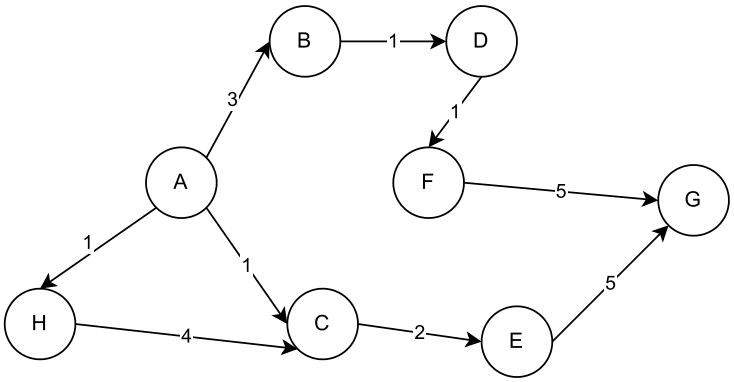
SPADE está no Apêndice C.

3.4.2 Planejamento de rotas dos veículos

Dentre os algoritmos apresentados na Seção 2.2, foi decidido implementar tanto o Dijkstra quanto o A* inicialmente, a fim de analisar o desempenho de cada um. Como o algoritmo proposto por Dijsktra não possui uma busca guiada, este precisa primeiro conhecer todas as distâncias dos nós na árvore até encontrar a solução. Tal fato faz com que ele precise percorrer todos os possíveis nós, mesmo aqueles que não desencadeiam em uma solução satisfatória, necessitando mais buscas para encontrar a solução.

Por outro lado, o algoritmo A*, por fazer uma busca guiada, sabendo a distância entre cada nó na árvore de busca e o nó desejado, consegue eliminar os nós que vão ter uma maior distância do destino dos que os que já foram encontrados com menor distância. Por isso, pode realizar menos buscas entre os nós até encontrar a sequência ideal para o caminho menos custoso. Seu funcionamento pode ser ilustrado conforme a Figura 7.

Figura 7 – Busca A*



Fonte: Elaborado pelo autor

Desta forma foi decidido utilizar apenas o algoritmo A*, presente no Apêndice D, para realizar a busca do percurso para cada veículo, considerando o desempenho e que a distância entre cada ponto é sempre conhecida, já que todos os veículos devem ter acesso à planta do ambiente.

Essa é composta por uma matriz com mxn aonde cada célula pode ser, ou um segmento de percurso de uma linha, ou um objeto estático, como estantes ou paredes. Assim que um veículo é inserido no sistema, ele recebe sua tarefa, sabendo sua posição atual e qual seu destino. Ele então procura a melhor rota utilizando o algoritmo de busca A*. O algoritmo realiza a busca dentro do ambiente, reconhecendo a posição do veículo na matriz como a raiz da árvore de busca e cada posição adjacente na matriz como um nó.

3.4.3 Comunicação com o servidor

Com a rota definida pelo veículo, o agente envia ao servidor uma mensagem para se inscrever como veículo dentro do ambiente, por meio de um *subscribe*, conforme ao acordo com a performativa FIPA. O veículo pode receber uma mensagem não autorizando a inscrição, por meio de um *refuse*, quando o ambiente não possui mais condições de ter outro veículo em trânsito. Assim, o veículo deve aguardar por um período para tentar se inscrever novamente.

Com a confirmação do servidor, o veículo então envia sua rota por meio da performativa *inform*, que deve ser conferida pelo servidor para garantir que não haja bloqueio existente na rota. Caso o veículo receba a confirmação da rota, pode começar a percorrer o seu percurso.

Todavia, caso receba um *disconfirm* com as posições que estão bloqueadas, deve utilizá-las para recalcular a rota e tentar a confirmação novamente. Um caminho pode ser recusado quando o percurso que o veículo está pedindo para realizar pode ser impossibilitado por outro veículo que já está circulando, quando não há outro percurso possível que possa ser realizado, como passagem por portões.

3.4.4 Comunicação com outro veículo

Ao longo do percurso, os veículos que sensorearem a outro agente e identificarem uma possível colisão, contactarão o servidor para poderem se comunicar com o outro veículo a fim de evitar o choque. Assim, envia-se uma performativa *request* requisitando o endereço do outro veículo e então, conjunto a este, pode-se definir potenciais ações preventivas às colisões ou impasses. Durante tal processo, ambos os veículos deverão reduzir suas velocidades.

Dessa forma, cabe ao servidor detectar quais os veículos próximos de colidir e sinalizar a ambos com qual veículo isto pode ocorrer. Cabe salientar que não é a

proposta deste trabalho apresentar um cenário com a participação de três ou mais veículos. Caso isso ocorra, o terceiro veículo não será mapeado e dessa forma, será ignorado e não participará das negociações.

Para que os agentes possam iniciar a negociação a fim de evitar uma colisão, o servidor envia o endereço XMPP do veículo com menor prioridade para o veículo que tiver maior prioridade. Forçando com que os veículos tenham uma comunicação direcional, evitando que os dois veículos precisem enviar mensagens para começar a negociação. Essa negociação segue a performativa FIPA *call for proposal*, na qual um agente requisita ao outro agente uma proposta para um determinado problema. A comunicação pode ser observada na Figura 6.

Desta forma, os veículos lidam de forma independente como solucionar a possível colisão, e na maioria dos cenários previstos, não existe a necessidade de mudança de rota por nenhum dos veículos. Porém, caso isso ocorra, o veículo precisa atualizar o servidor sobre o seu novo percurso. Quando cada veículo finaliza sua tarefa, ele envia uma mensagem ao servidor para sinalizá-lo que encerrou sua tarefa e para que o servidor possa removê-lo dos agentes em execução.

A estratégia de comunicação entre os veículos mostra que a arquitetura do sistema é centralizada no servidor, embora exista uma decentralização associada com a negociação para resolução do conflito entre os veículos. Mesmo assim, ao final da negociação, pode ser necessário novamente atualizar o servidor com a proposta de uma nova rota.

3.4.5 Objetos desconhecidos

Caso o servidor não responda com o endereço do outro veículo, pode-se inferir que o objeto sensorizado não se trata de outro agente, podendo ser uma pessoa, animal ou algum objeto inanimado. Nesse caso, em um cenário ideal, o agente deveria tentar contornar o objeto com velocidade reduzida. Cabe salientar que não é a proposta deste trabalho simular tal situação, pois veículos seguidores de linha, tradicionalmente, apenas seguem a rota imposta, não podendo desviar ou contornar o obstáculo se isto não fora previamente imposto.

Além disso, o reconhecimento de outros agentes é realizado pela detecção de sinais (*Bluetooth*), sendo então, impossível reconhecer outro objeto que não emita tal sinal. Contudo, caso encontre outro objeto que utilize *Bluetooth*, como um fone de ouvido, pode causar o início de um diálogo com o servidor, mas desde que esse fone não seja encontrado por dois veículos ao mesmo tempo, o servidor irá entender que não existem veículos próximos e não será necessário nenhum tipo de negociação.

3.5 ENCONTROS E POSSÍVEIS SOLUÇÕES

Como descrito anteriormente, quando um veículo encontra algum objeto, ele deve procurar a melhor forma para solucionar o conflito, evitando perder tempo na resolução e também garantindo que não ocorrerá nenhuma colisão. Apesar de o servidor poder conferir as rotas de cada veículo, algumas situações como obstrução do trajeto por um objeto que não seja um agente dentro do sistema, a bateria do veículo ou a carga que ele está carregando podem alterar a velocidade do veículo durante o seu trajeto, aumentando o tempo necessário para percorrer cada trecho da sua rota. Assim, ocasionais choques poderiam acontecer durante o percurso dos veículos, mesmo que a rota inicialmente não gerasse colisões.

Para facilitar a identificação de objetos, em condições reais, poderiam ser utilizados desde sensores de objeto, câmeras, antenas a combinações de componentes para encontrar objetos próximos. No instante que um veículo percebe que está dentro do campo de outro veículo ou objeto, ele reduz sua velocidade e envia uma mensagem ao servidor, requisitando o endereço do outro agente para começar a negociação entre com ele. Para simulação, foi criado um objeto circular em torno do veículo, que pode detectar colisões com outros objetos criados para cada veículo

O momento que os veículos percebem que pode ocorrer uma colisão pode ser reduzido a uma matriz $3x3$ (considerando não haver movimentação em diagonais da matriz). Desta forma, apenas são necessárias as posições atuais de cada veículo e a sua direção, não sendo necessário que um veículo envie toda a sua rota para ser feita a negociação. Para poder ser determinada a direção dentro da matriz, faz-se necessário que o veículo envie duas posições posteriores junto à posição atual. Assim, o veículo com maior prioridade para rota, envia ao endereço recebido pelo servidor, um vetor com 3 posições (x, y) e fica no aguardo pela solução. O veículo que recebe esse vetor, gera a matriz $3x3$ identificando qual o cenário que os veículos se encontram, mapeando o vetor recebido com um vetor criado com a sua posição atual e as duas posteriores.

3.5.1 Redução da matriz

Para que os veículos consigam encontrar em qual cenário eles se encontram, será muito mais simples analisar apenas o cruzamento em que eles se encontram, em vez de se verificar a rota inteira de cada um, composta por um vetor de tamanho a. Esse cruzamento pode ser compreendido como uma matriz $3x3$.

Para reduzir o espaço do ambiente que é de uma matriz mxn, sendo m e $n >= 3$, em uma matriz reduzida $3x3$, deve-se tomar um dos veículos como base para o centro do cruzamento, já que, existe a possibilidade de um dos veículos não percorrer o centro do cruzamento. Com isso, é considerado que a posição central do veículo com maior prioridade equivale à posição $1x1$ na matriz reduzida. Assim, é calculada

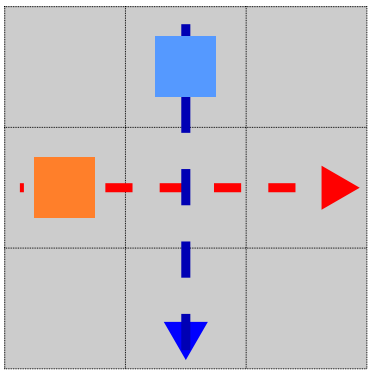
a diferença entre as posições iniciais ou finais de cada veículo, com a sua posição central. A diferença entre posição inicial é subtraída à posição $1x1$ para encontrar qual a posição inicial na matriz reduzida e a diferença da posição final é somada à posição $1x1$ para encontrar a posição final na matriz reduzida. O mesmo é feito para o vetor de posições do veículo de menor prioridade, porém este veículo, pode gerar posições fora da matriz reduzida, que devem ser ignorados durante a análise dos cenários de conflitos.

3.5.2 Cenários de conflitos

Os cenários descritos a seguir, representam as possíveis colisões que podem ocorrer na execução do deslocamento de múltiplos veículos no ambiente planejado.

- **Primeiro Cenário: Cruzamento**: ocorre quando dois veículos cruzam seus caminhos, compartilhando apenas a posição central da matriz conforme descrito na Figura 8. Os veículos entram no cruzamento por vias perpendiculares, podendo ocorrer a colisão no centro do cruzamento. Ambos veículos têm saídas diferentes, e diferem do ponto de entrada do outro veículo;

Figura 8 – Cenário 1

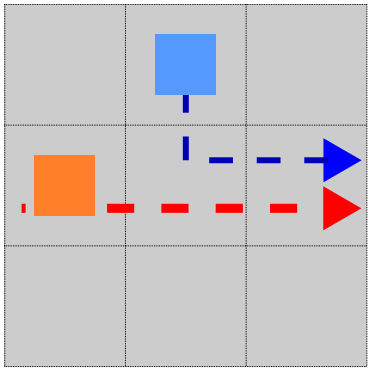


Fonte: Elaborado pelo autor

- **Segundo Cenário: Saída igual**: ocorre quando dois veículos cruzam seus caminhos, porém tem a mesma posição de saída após o cruzamento como descrito na Figura 9. Veículos entram em um cruzamento por vias perpendiculares, cruzando no ponto médio e saindo pela mesma via de saída. Nesse caso, o ponto central e o ponto de saída de cada veículo pode gerar uma colisão;

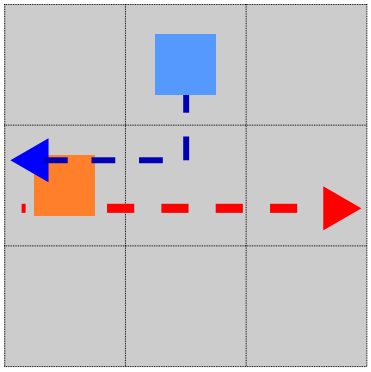
- **Terceiro Cenário: cruzamento com bloqueio**: ocorre quando dois veículos cruzam seus caminhos, porém um dos veículos se encontra bloqueando a saída do outro, como descrito na Figura 10. Veículos entram em um cruzamento por vias perpendiculares, podendo ocorrer colisão no centro do cruzamento e a saída dum veículo corresponde à entrada de outro veículo;

Figura 9 – Cenário 2



Fonte: Elaborado pelo autor

Figura 10 – Cenário 3



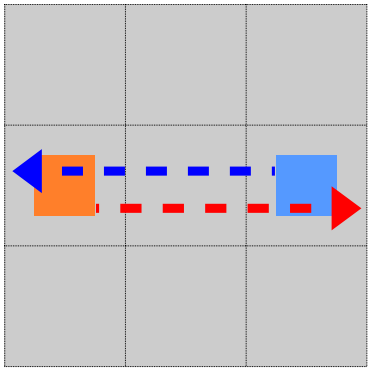
Fonte: Elaborado pelo autor

- **Quarto Cenário: bloqueio total**: ocorre quando os caminhos dos veículos são iguais, porém em sentidos diferentes como descrito na Figura 11. Veículos entram por vias opostas do cruzamento, e as vias de saída de cada veículo, corresponde a via de entrada do outro veículo. Há um bloqueio completo, pois nenhum veículo pode executar a rota sem que ocorra colisão. Nesse caso, pode ocorrer um bloqueio completo caso não exista outra possível rota para o veículo de menor prioridade;

- **Quinto Cenário: veículos com mesma direção**: ocorre quando os veículos estão próximos, mas não gera situação de conflito de posições, como descrito nas Figuras 12 e 13. Veículos se encontram no cruzamento, mas estão percorrendo na mesma direção. Enquanto um veículo está entrando no cruzamento, o outro já está saindo;

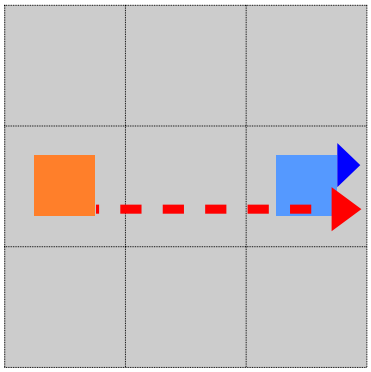
- **Sexto Cenário: sem conflito e direções opostas**: ocorre quando os dois veículos se encontram próximos, mas sem nenhum conflito, como pode ser visto

Figura 11 – Cenário 4



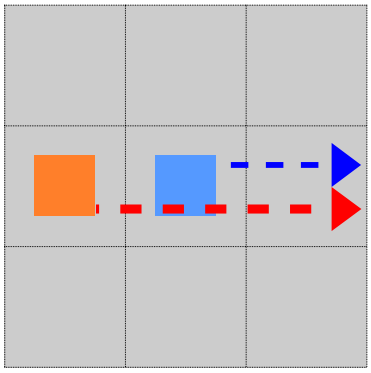
Fonte: Elaborado pelo autor

Figura 12 – Cenário 5 a



Fonte: Elaborado pelo autor

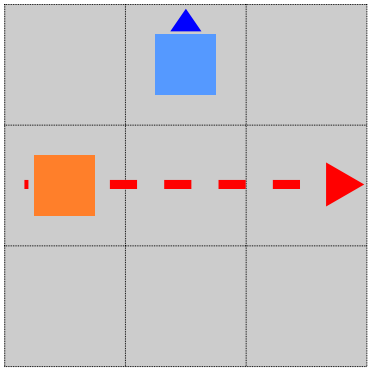
Figura 13 – Cenário 5 b



Fonte: Elaborado pelo autor

na Figura 14, cada veículo possui uma direção diferente. Veículos se encontram, mas não há possível colisão, pode ocorrer quando um veículo já está saindo do cruzamento e o outro faz um caminho por uma saída diferente;

Figura 14 – Cenário 6



Fonte: Elaborado pelo autor

3.6 PROPOSTAS DE SOLUÇÕES

Na Subseção 3.4.4 é descrito que os veículos realizam propostas para chegar a solução dos conflitos. Essas propostas devem possuir informações necessárias para que cada veículo possa agir para evitar uma colisão e garantir que todos atinjam seus objetivos. Enquanto os agentes estão realizando ou respondendo propostas, devem reduzir a sua velocidade.

As propostas realizadas devem tomar as seguintes ações:

- **Não agir**: O veículo que recebe a proposta não precisa tomar nenhuma ação, recuperando sua velocidade de operação. A ação resolutiva do conflito é tomada pelo veículo que envia a proposta.
- **Reduzir velocidade** : O veículo que recebe a proposta deve reduzir sua velocidade, recuperando-a após a situação de conflito deixar de existir, ou seja, quando o outro agente não for mais alcançado pelo sensor.
- **Parar**: O veículo que recebe a proposta deve parar a sua movimentação, voltando a se movimentar após a situação de conflito deixar de existir, ou seja, quando o outro agente não for mais alcançado pelo sensor.
- **Encontrar nova rota**: O veículo recebe posições bloqueadas e deve encontrar um novo caminho para percorrer com velocidade reduzida até que o outro agente não seja mais alcançado pelo sensor.

Contudo, a proposta recebida pode ser aceita como verdade pelo agente ou rejeitada. Pode ser recusada pelos seguintes motivos:

- **Prioridade**: o veículo pode acreditar que a proposta é injusta, causando aumento do tempo de execução ou de seu percurso, prejudicando seu desempenho. Ao rejeitar uma proposta por prioridade, o agente deve responder junto à performativa *reject* qual informação ele acredita que tenha prioridade sobre o outro veículo, como o tempo de percurso restante ou considerar como custoso encontrar uma rota nova. Assim, o agente aguarda uma nova proposta, procurando solucionar o

conflito de uma forma vantajosa para ambos.

- **Impasse**: quando o veículo deve encontrar um novo percurso, mas a busca A* não retorna nenhum caminho possível. A proposta recebida é impossível de ser executada e a informação acompanhada pelo *reject* deve indicar que o veículo não consegue de encontrar uma rota nova, esperando que o outro veículo realize a mudança de rota.

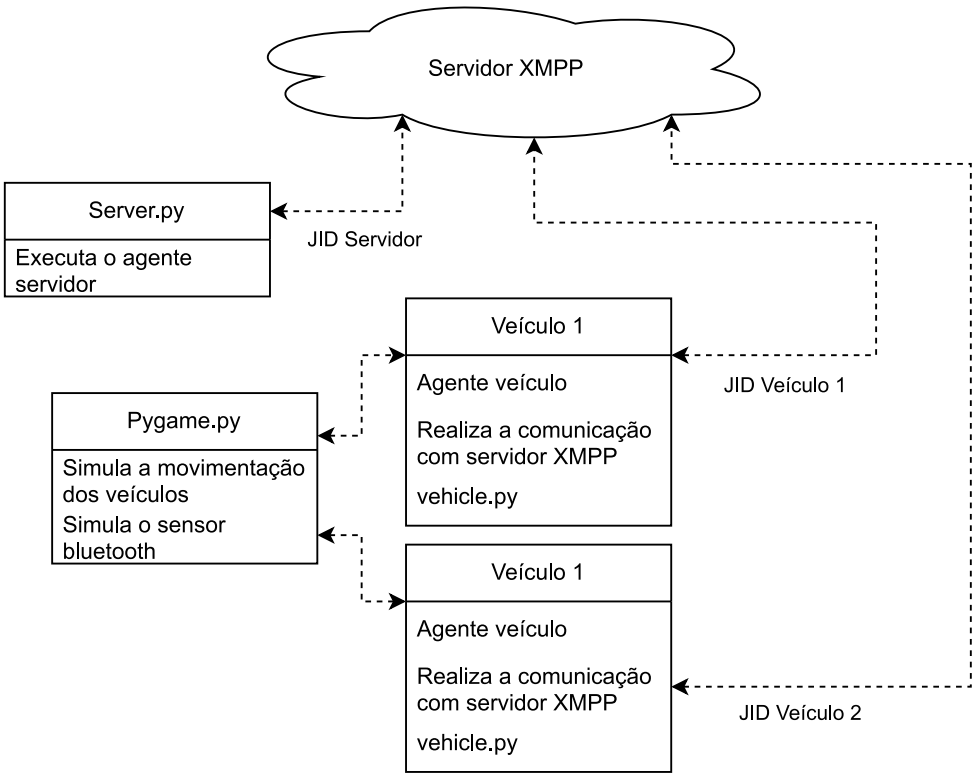
4 ANÁLISE DE DADOS

Como resultado desse trabalho, três resultados de simulações foram obtidos, detalhados nesse capítulo, com uma comparação entre eles ao final.

4.1 ARQUITETURA DA SIMULAÇÃO

A simulação é realizada em Python, com o uso da biblioteca SPADE e Pygame e pode ser observada na Figura 15. O agente servidor é executado pelo *script Server.py*, realizando a comunicação direta com o servidor XMPP por meio do endereço JID definido para o servidor, ele deve conhecer os endereços de todos os agentes veículos que podem ser executados dentro do ambiente.

Figura 15 – Arquitetura da Simulação



Fonte: Elaborado pelo autor

Cada agente veículo é instanciado pela execução do *script Pygame.py*, criando duas instâncias da classe disponível em *vehicle.py*, uma para o veículo 1 e outra para o veículo 2. O script vehicle.py realiza a comunicação do agente com o servidor XMPP por meio do seu endereço JID. Cada agente veículo só tem disponível o endereço do servidor para comunicação.

O *script Pygame.py* realiza a simulação do deslocamento dos veículos no

ambiente e o sensoriamento *Bluetooth*

4.2 CONSIDERAÇÕES SOBRE SIMULAÇÕES

Preliminarmente, o servidor realizava a análise do trajeto integral de cada veículo considerando possíveis colisões e manejava as posições para evitar isto. Contudo, esse artifício centraliza o sistema, uma vez que o servidor deveria resolver os conflitos singularmente, ocasionando na perda de autonomia de um dos veículos e sobrecarregando o servidor. Deste modo, no decorrer da simulação, o servidor centralizado pode gerar mais conflitos. Já com a estratégia descentralizada, o servidor se restringe a simular simplificadamente as rotas de cada veículo, considerando a velocidade padrão de cada um e verificando a possibilidade de colisão ou bloqueios. Assim, o veículo mantém a autonomia de escolha de trajeto e não sobrecarrega o servidor. Dessa forma, a detecção de colisão no servidor foi simplificada a fim de garantir que a comunicação entre os veículos seja efetuada e de descentralizar a solução dos conflitos.

Para se estipular o desempenho das simulações, deve-se considerar o tempo de execução do trajeto. Tal parâmetro é atrelado a velocidade do veículo, a escolha da melhor rota e a quantidade de mensagens trocadas e sua resolutividade no possível conflito, dependente do servidor XMPP e da negociação entre agentes.

4.3 RESULTADOS DAS SIMULAÇÕES

O servidor sempre irá escolher por ordem de chegada dos pedidos de comunicação entre veículos, quem irá realizar o pedido pela proposta e quem irá enviar a proposta.

Para cada situação testada nas Subseções 4.3.2 e 4.3.3, foram executadas três simulações para obter os tempos.

4.3.1 Simulacão 0: sem colisões

Em simulações com apenas um veículo percorrendo o ambiente, os resultados serão utilizados apenas como métricas para analisar o desempenho das outras simulações. Para garantir que a simulação possua resultados que possam ser comparados, são utilizados os mesmos percursos que geram cada tipo de cenário, utilizado nas demais simulações.

Os resultados de tempo para os caminhos realizados serão discutidos junto aos resultados das simulações nas Subseções 4.3.2 e 4.3.3.

4.3.2 Simulação 1: negociação simplificada

A premissa da simulação apresentada nesse tópico é que todas as propostas fornecidas por um veículo durante a negociação sempre serão absolutas, independente de prioridade, tempo de rota, tamanho da rota, distância que falta para chegar ao destino. Isso pode prejudicar o agente que recebe a proposta. Dessa forma, o agente que recebe a proposta, não pode responder com a performativa *reject*, conforme é apresentado na Seção 3.6.

Esse caso pode ser descrito em uma condição na qual o servidor escolhe a ordem dos veículos com base em prioridade, é exigido que o veículo com menor prioridade aceite a proposição. Resultados para cada cenário descritos em 3.5.2.

A solução para os cenários quando a saída é igual e cruzamento com bloqueio é a redução da velocidade do veículo que possui mais posições bloqueantes, podendo ser executadas sem dificuldades, desde que, quando os veículos compartilham a posição central da matriz, um deles pare por completo. O veículo que realiza a proposta sempre considera quem tem um possível bloqueio, realizando a proposta que evita que ocorra um impasse.

Podem existir condições nas quais o cenário de bloqueio total gere um impasse sem solução. Ou seja, quando o veículo que está propondo uma mudança de rota está bloqueando o único caminho possível para que o outro veículo consiga realizar uma rota. Caso esse cenário ocorra, ambos os veículos ficam parados por não conseguirem encontrar uma solução.

Os tempos, em segundos, obtidos para um cenário quando ambos os veículos possuem a mesma saída podem ser observados no Quadro 3.

Quadro 3 – Resultados para cenário com saídas iguais

Saída Igual	1	2	3	Simulação 0
Veículo 1 (*await*)	32,73	33,47	32,78	29,44
Veículo 2 (cfp)	28,61	29,63	29,18	22,02

Fonte: elaborado pelo autor.

Os tempos, em segundos, obtidos para um cenário quando ambos os veículos causam um cruzamento com bloqueio podem ser observados no Quadro 4.

Quadro 4 – Resultados para cenário com cruzamento com bloqueio

Cruzamento com bloqueio	1	2	3	Simulação 0
Veículo 1 (*await*)	33,22	34,23	34,85	29, 44
Veículo 2 (cfp)	30,28	32,68	31,53	23,05

Fonte: elaborado pelo autor.

Os tempos, em segundos, obtidos para um cenário quando ambos os veículos causam um bloqueio total podem ser observados no Quadro 5.

Quadro 5 – Resultados para cenário com bloqueio total

Bloqueio total	1	2	3	Simulação 0
Veículo 1 (*await*)	28,44	27,60	26,75	24,57
Veículo 2 (cfp)	15,39	15,22	15,45	15,06

Fonte: elaborado pelo autor.

Os tempos, em segundos, obtidos para um cenário quando ambos os veículos possuem percursos com a mesma direção podem ser observados no Quadro 6.

Quadro 6 – Resultados para cenário com veículos com mesma direção

Mesma direção	1	2	3	Simulação 0
Veículo 1 (*await*)	32,70	29,78	30,72	29,44
Veículo 2 (cfp)	27,36	26,14	27,26	25,44

Fonte: elaborado pelo autor.

Nos cenários nos quais os veículos têm a mesma direção e estão muito próximos, é requisitado ao veículo que está atrás que pare até não encontrar mais o veículo da frente, causando um atraso maior para evitar que ambos possam entrar em alguma condição de colisão futura. Porém, em um cenário com distância maior, apenas reduzem a velocidade enquanto estão negociando.

O cenário de cruzamento não é possível de ser atingido nas simulações apresentadas, pois o raio de ação do sensor utilizado para conflitos sempre acionará quando os dois veículos tiverem pelo menos 2 posições de diferença. Nas tentativas de simular, outros cenários são atingidos.

Por fim, no cenário quando não há conflitos, os veículos apenas reduzem a sua velocidade para verificar a situação e prontamente voltam a operar normalmente.

Em todos os casos descritos, os veículos trocam apenas uma mensagem com o servidor e uma mensagem para negociação.

4.3.3 Simulação 2: negociação com prioridades

Nas simulações apresentadas nesse tópico, os agentes utilizarão condições para aprovação de propostas.

Com essa modificação, o veículo que considerar a proposição recebida como prejudicial para seu desempenho, poderá rejeitá-la e iniciar uma nova negociação buscando uma solução benéfica para ambos.

Na Subseção 4.3.2 foi definido que o cenário de cruzamento não é possível ser simulado devido às características do sensor de distância.

Todos os cenários que a proposta recebida exigem uma mudança de rota, ou redução de velocidade sem que existam bloqueios serão rejeitados pelo veículo, isso pode ocorrer nos cenários de saída, cruzamento com bloqueio e bloqueio total.

Nos outros cenários, o comportamento será igual aos descritos na Subseção 4.3.2.

Os tempos, em segundos, obtidos para um cenário quando ambos os veículos possuem percursos com a mesma direção podem ser observados no Quadro 6.

Quadro 7 – Resultados para cenário com bloqueio total

Bloqueio total	1	2	3	Simulação 0
Veículo 1 (*await*)	27,70	27,48	28,29	24,57
Veículo 2 (cfp)	18,54	18,07	18,29	15,06

Fonte: elaborado pelo autor.

Em todos os casos, exceto o citado no Quadro 7, os veículos trocam apenas uma mensagem com o servidor e uma mensagem para negociação. No Quadro 7 são necessárias cada agente realiza duas mensagens para a negociação.

4.3.4 Análise dos resultados das simulações

Com os resultados obtidos nas Subseções 4.3.2 e 4.3.3, observa-se que o principal fator para gerar atrasos de tempo é decorrente da troca de mensagens, que os veículos reduzem a velocidade para evitar colisões durante esse período.

Como o agente que realiza a proposta, considera quaisquer conflitos que podem acontecer dentro do espaço em que pode ocorrer um impasse e já propor a melhor solução, nos casos em que os veículos não estão bloqueando por completo seus percursos, nunca será necessária uma renegociação devido à prioridade de um veículo. Inclusive, a proposta já considera quem é o melhor agente para encontrar uma nova rota, considerando o possível bloqueio, garantindo que sempre existirá uma rota possível para ser encontrada.

Quando há um bloqueio total dos percursos e renegociação ocorre, é notado um aumento significativo do tempo de execução dos dois veículo, devido as novas mensagens que precisam ser trocadas para solucionar o conflito. Como nessa situação, os veículos precisam aguardar enquanto o outro veículo procura pela nova rota e começar a percorrê-la, ambos os veículos ficam em um estado com a velocidade reduzida, incrementando o tempo de execução de cada um.

Contudo, apesar dos diálogos realizados entre os agentes incrementar o tempo de execução das rotas de cada veículo, isso possibilita todos os veículos a alcançarem seus objetivos finais, ocorrendo a cooperação entre todos os agentes presentes no ambiente.

A proposta não resolveria impasses entre três ou mais agentes simultaneamente. Caso isso ocorresse, o servidor escolheria um par de veículos para resolverem o conflito, ignorando o terceiro veículo. Contudo, o veículo ignorado pode causar impasses sobre a solução encontrada pelos outros veículos, pois eles não

terão disponíveis as posições na matriz do terceiro veículo.

Ademais, esse trabalho não considera a perda de tempo durante a comunicação realizada com o servidor XMPP ou o atraso de propagação do sinal *Bluetooth*. As características de tempo demonstradas são utilizadas para comprovar que a solução de conflitos é realizada e que pode demandar mais tempo que a execução sem a detecção de conflitos.

4.4 DIFICULDADES ENCONTRADAS

Com o desenvolvimento da simulação, foram encontradas dificuldades com o servidor XMPP utilizado para a troca de mensagens entre agentes. O primeiro servidor utilizado passou por instabilidade durante o desenvolvimento, sendo necessária a troca por outro, existiu dificuldade para encontrar outro servidor com desempenho igual. Outra opção seria criar um servidor XMPP local, que considerando uma aplicação real, torna-se a melhor opção, pois o controle do servidor fica nas mãos do proprietário.

A documentação da biblioteca SPADE para Python dificultou o desenvolvimento da simulação, considerando que agentes sempre possuem comportamentos únicos, não tendo documentada a troca de performativas para o mesmo agente, contudo, durante o desenvolvimento, foi possível controlar o tipo de performativa esperado para cada estado da máquina de estado do agente.

5 CONCLUSÕES

Este trabalho descreveu os conceitos necessários para propor uma estratégia de coordenação de veículos seguidores de linha em um sistema multiagentes. Definindo os conceitos de veículos auto guiados, algoritmos de busca, agentes inteligentes, sistemas multiagentes e linguagens de comunicação, necessários para o desenvolvimento.

Seu desenvolvimento foi dividido em quatro partes, a viabilização técnica do *Bluetooth* como estratégia para encontrar outros agentes, a definição do melhor algoritmo de busca para o planejamento de rotas, a definição da máquina de estados de cada agente e os diálogos entre eles utilizando protocolos da indústria e a descrição dos cenários que podem ser encontrados durante a execução das tarefas dos veículos dentro do ambiente.

A demonstração da proposta foi realizada por uma simulação, refletindo as características reais da movimentação dos veículos seguidores de linha, sensores de proximidade e diálogo entre agentes. Provando que os cenários de conflito descritos podem ser solucionados a partir da proposta de coordenação. Também demonstra a capacidade do algoritmo de busca A* definir o planejamento da rota, mesmo quando é necessário resolver o conflito.

Do trabalho foi possível compreender que a colaboração entre agentes é necessária para atingir os diversos objetivos individuais, garantindo a coexistência e cooperação entre eles.

Além disso, foi possível demonstrar como o uso da linguagem FIPA ACL padroniza e estrutura a comunicação. Como a plataforma SPADE, permite o uso de um servidor de mensagens XMPP para os diálogos entre agentes. Também, que o uso do BLE como método para detecção de outros veículos é viável.

Com todas as ferramentas utilizadas, é possível inserir a proposta em uma aplicação real, inclusive o desenvolvimento de um protótipo.

5.1 TRABALHOS FUTUROS

A partir do que foi desenvolvido nesse trabalho, obteve-se uma estratégia de coordenação, baseada em regras e condições definidas para evitar impasses durante a execução de cada agente. Porém, seguindo o conceito de agentes inteligentes, pode-se implementar conceitos de aprendizado e uso de redes neurais para a formulação dos agentes e suas decisões sobre o ambiente, visando a solução do problema de coordenação e tomadas de decisões.

Outrossim, é possível realizar a simulação considerando todos os atrasos

adicionados durante o processo, seja pela propagação do sinal *Bluetooth*, ou pelo atraso adicionado pela comunicação com o servidor XMPP, sendo possível até a implementação local do servidor para troca de mensagens entre agentes.

REFERÊNCIAS

J. AGUILAR A. RIOS, F. H.; CERRADA, M. Sistemas multiagentes y sus aplicaciones en automatizaci on industrial. 01 2013.

ALMEIDA, L. F. A. **VEÍCULO AUTO GUIADO (AGV - AUTOMATED GUIDED VEHICLE) - PROTÓTIPO SEGUIDOR DE LINHA**. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) — INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO CAMPUS SÃO JOSÉ DOS CAMPOS, 2016.

ARDUINOBLE. 2018. Disponível em: https://www.arduino.cc/reference/en/libraries/arduinoble/. Acesso em: acesso em 12 de dezembro de 2022.

BITTENCOURT, G. **Inteligência artificial : ferramentas e teorias** / **Guilherme Bittencourt.** [s.n.], 2006. ISBN 8532801382. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=cat07205a&AN=uls.227065&site=eds-live.

BROWN, D. **Fortaleza digital**. Ediciones Urano S. A., 2009. (Books4pocket narrativa). ISBN 9788492516209. Disponível em: https://books.google.com.br/books?id=Em8dSQAACAAJ.

DAS, S. K.; PASAN, M. K. Design and methodology of automated guided vehicle-a review. **IOSR Journal of Mechanical and Civil Engineering**, Special Issue, p. 29–35, apr 2016.

DONANCIO, H.; CASALS, A.; BRANDãO, A. **Exposing agents as web services: a case study using JADE and SPADE**. [S.l.: s.n.], 2019.

DRAGANJAC, I. et al. Decentralized control of multi-agv systems in autonomous warehousing applications. **IEEE Transactions on Automation Science and Engineering**, PP, p. 1–15, 09 2016.

FERBER, J. **Multi-agent systems : an introduction to distributed artificial intelligence** / **Jacques Ferber.** [s.n.], 1999. ISBN 0201360489. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=cat07205a&AN=uls.197386&site=eds-live.

FIPA. **Agent Communication Language Specifications**. 2002. Disponível em: http://www.fipa.org/repository/aclspecs.html. Acesso em: acesso em 12 de dezembro de 2022.

FIPA. **FIPA Request Interaction Protocol Specification**. 2002. Disponível em: http://www.fipa.org/specs/fipa00026/SC00026H.html. Acesso em: acesso em 12 de dezembro de 2022.

FLIPSE, M. **Altering and Improving Kiva Some suggestions for improvement of the current Kiva system**. Paper — Vrije Universiteit, 2011.

FLYNN, M.; FLYNN, M. J. **Computer Architecture: Pipelined and Parallel Processor Design**. Jones and Bartlett, 1995. (Computer Science Series). ISBN 9780867202045. Disponível em: https://books.google.com.br/books?id=JS-01OTl9dsC.

GLUZ, J. C.; VICCARI, R. M. **Linguagens de Comunicação entre Agentes: Fundamentos Padrões e Perspectivas**. 2003. Disponível em: http://professor.unisinos.br/jcgluz/disciplina-aose/apostila-jaia03-acl-formato-sbc.pdf.

GOYAL, A. et al. Path finding: A* or dijkstra's? **International Journal in IT and Engineering**, v. 2, p. 1–15, 2014.

HUHNS, M.; STEPHENS, L. Multiagent systems and societies of agents. 03 1998.

JENNINGS, N. R.; BUSSMANN, S. Agent-based control systems: Why are they suited to engineering complex systems? **IEEE Control Systems Magazine**, v. 23, n. 3, p. 61–73, 2003.

Chapter 1 - introducing bluetooth applications. In: KAMMER, D. et al. (Ed.). **Bluetooth Application Developer's Guide**. Burlington: Syngress, 2002. p. 1–68. ISBN 978-1-928994-42-8. Disponível em: https://www.sciencedirect.com/science/article/pii/B9781928994428500045.

Q. LI A.C. ADRIAANSEN, J. U.; POGROMSKY, A. Design and control of automated guided vehicle systems: A case study. **IFAC Proceedings Volumes**, v. 44, n. 1, p. 13852 – 13857, 2011. ISSN 1474-6670. 18th IFAC World Congress. Disponível em: http://www.sciencedirect.com/science/article/pii/S1474667016458507.

NORVIG, P.; RUSSELL, S. **Inteligência Artificial**. 3. ed. Rio de Janeiro: Elsevier, 2013.

JAMILI OSKOUEI, D. R.; VARZEGHANI, H.; SAMADYAR, Z. Intelligent agents: A comprehensive survey. **International Journal of Electronics Communication and Computer Engineering (2278–4209)**, v. 5, p. 790–798, 06 2014.

PAKDAMAN, M.; SANAATIYAN, M. M.; GHAHROUDI, M. R. A line follower robot from design to implementation: Technical issues and problems. In: **2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)**. [S.l.: s.n.], 2010. v. 1, p. 5–9.

PALANCA, J. **SPADE**. 2020. Disponível em: https://spade-mas.readthedocs.io/en/latest/readme.html/. Acesso em: acesso em 12 de dezembro de 2022.

PARAMESWARAN, A.; IFTEKHAR HUSAIN, M.; UPADHYAYA, S. Is rssi a reliable parameter in sensor localization algorithms: an experimental study. 01 2009.

PIRES, J. N. Robótica das máquinas gregas à moderna robótica industrial. **Jornal Público, caderno de Computadores**, n. 1, jul. 2002.

PSCHEIDT, E. R. **Robô Autônomo - Modelo de Chão de Fábrica**. Monografia — Centro Universitário Positivo, Núcleo de Ciências Exatas e Tecnológicas, Engenharia da Computação, 2007.

PYGAME. 2009. Disponível em: https://www.pygame.org/wiki/about. Acesso em: acesso em 12 de dezembro de 2022.

RAMADANI, E.; HALILI, F.; IDRIZI, F. Tailored dijkstra and astar algorithms for shortest path softbot roadmap in 2d grid in a sequence of tuples. v. 8, p. 56, 09 2019.

REIS, L. P. **Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico (Coordination in Multi-Agent Systems: Applications in University Management and Robotic Soccer)**. PhD Thesis — FEUP, julho 2003.

ROBOTIC AUTOMATION. **Hospital AGVs**. 2020. Disponível em: https://www.roboticautomation.com.au/ra-solutions/hospital-agvs-automated-guided-vehicles. Acesso em: 8 nov. 2022.

SEER. **AMB-J Product Parameters**. 2022. Disponível em: https://www.seer-group.com/agvs/amr-robots/AMB-J/Parameter. Acesso em: acesso em 15 de abril de 2022.

SIEGWART, R.; SCARAMUZZA, D.; NOURBAKHSH, I. R. **Introduction to autonomous mobile robots / Roland Siegwart, Illah R. Nourbakhsh, Davide Scaramuzza**. [s.n.], 2011. ISBN 9780262015356. Disponível em: http://search.ebscohost.com/login.aspx?direct=true&db=cat07205a&AN=uls.332398&site=eds-live.

SIG, B. **Understanding Bluetooth Range**. 2021. Disponível em: https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/. Acesso em: acesso em 19 de dezembro de 2022.

SILVA, B. C. d. **Utilização de um sistema multi-agentes em redes de comunicação para a proteção digital de distância adaptativa**. Dissertação (Mestrado em Sistemas Elétricos de Potência) — Escola de Engenharia de São Carlos, Universidade de São Paulo, 2009.

SILVA, J. M. C. da; BAVARESCO, N.; SILVEIRA, R. A. Projeto e desenvolvimento de um sistema multi-agentes para objetos inteligentes de aprendizagem baseado no padrão scorm. **Revista Brasileira de Informática na Educação**, v. 16, n. 01, 2008.

SILVA, L. R. d.; BITTENCOURT, G. **Análise e programação de robôs móveis autônomos da plataforma Eyebot Luciano Rottava da Silva ; orientador, Guilherme Bittencourt. [dissertação]** /. 2003. Disponível em: http://www.tede.ufsc.br/teses/PEEL0815.pdf.

TOBE, F. **The technology gap left by Amazon's acquisition of Kiva Systems**. 2016. Disponível em: https://www.therobotreport.com/the-technology-gap-left-by-amazons-acquisition-of-kiva-systems/. Acesso em: acesso em 15 de novembro de 2020.

SILVA VIANNA, A. J. G. da; SOUSA FERREIRA, M. de; NUNES, T. C. A. **Veículo seguidor de trilha**. Trabalho de Conclusão de Curso (Técnico em Automação Industrial) — Centro Federal de Educação Tecnológica de Minas Gerais, 2014.

VIVALDINI, K. et al. Robotic forklifts for intelligent warehouses: Routing, path planning, and auto-localization. In: . [S.l.: s.n.], 2010. p. 1463 – 1468.

VOLKSWAGEN. **AGV Estágio**. 2014. Disponível em: http://www.vw.com.br/blogvolkswagen/post/2012/09/03/AGVEstagio.aspx. Acesso em: acesso em 11 de abril de 2020.

WEISS, G. Multiagent systems: A modern approach to distributed artificial intelligence. In: ____. [S.l.: s.n.], 2000. v. 1, p. –648. ISBN 0262731312.

WIRED. **How the Tesla Model S is Made | Tesla Motors Part 1**. 2013. Disponível em: https://www.youtube.com/watch?v=8_lfxPI5ObM. Acesso em: 9 nov. 2020.

WURMAN, P. R.; D ANDREA, R.; MOUNTZ, M. **Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses**. 2008. 9-9-19 p. Disponível em: http://explore.bl.uk/primo_library/libweb/action/display.do?tabs=detailsTab&gathStatTab=true&ct=display&fn=search&doc=ETOCRN226719396&indx=1&recIds=ETOCRN226719396.

XMPP. 2006. Disponível em: https://xmpp.org/about/. Acesso em: acesso em 19 de novembro de 2020.

YAGHOUB, S. et al. Designing and methodology of automated guided vehicle robots/ self guided vehicles systems, future trends. In: . [S.l.: s.n.], 2012. p. 340–345.

APÊNDICE A - MÁQUINAS DE ESTADOS DO SERVIDOR

Máquina de estados representando o funcionamento do servidor, descrito na Seção 3.3.

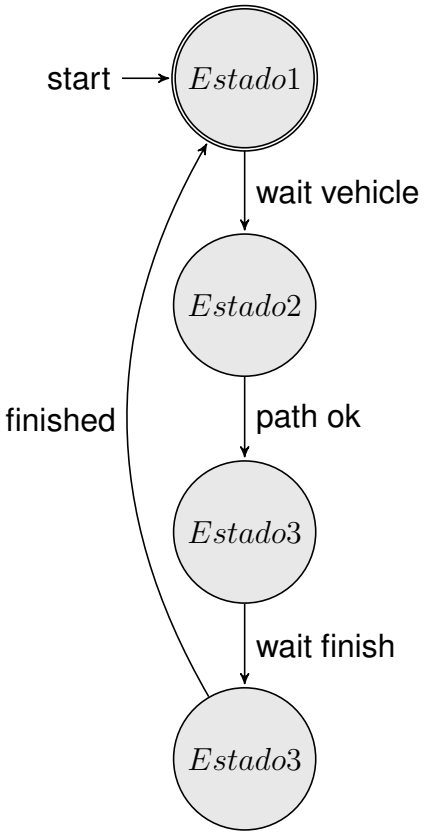


Figura 16 – Máquina de Estados do Servidor

Fonte: Elaborado pelo autor

APÊNDICE B - MÁQUINAS DE ESTADOS DO VEÍCULO

Máquina de estados representando o funcionamento do veículo, descrito na Seção 3.4.

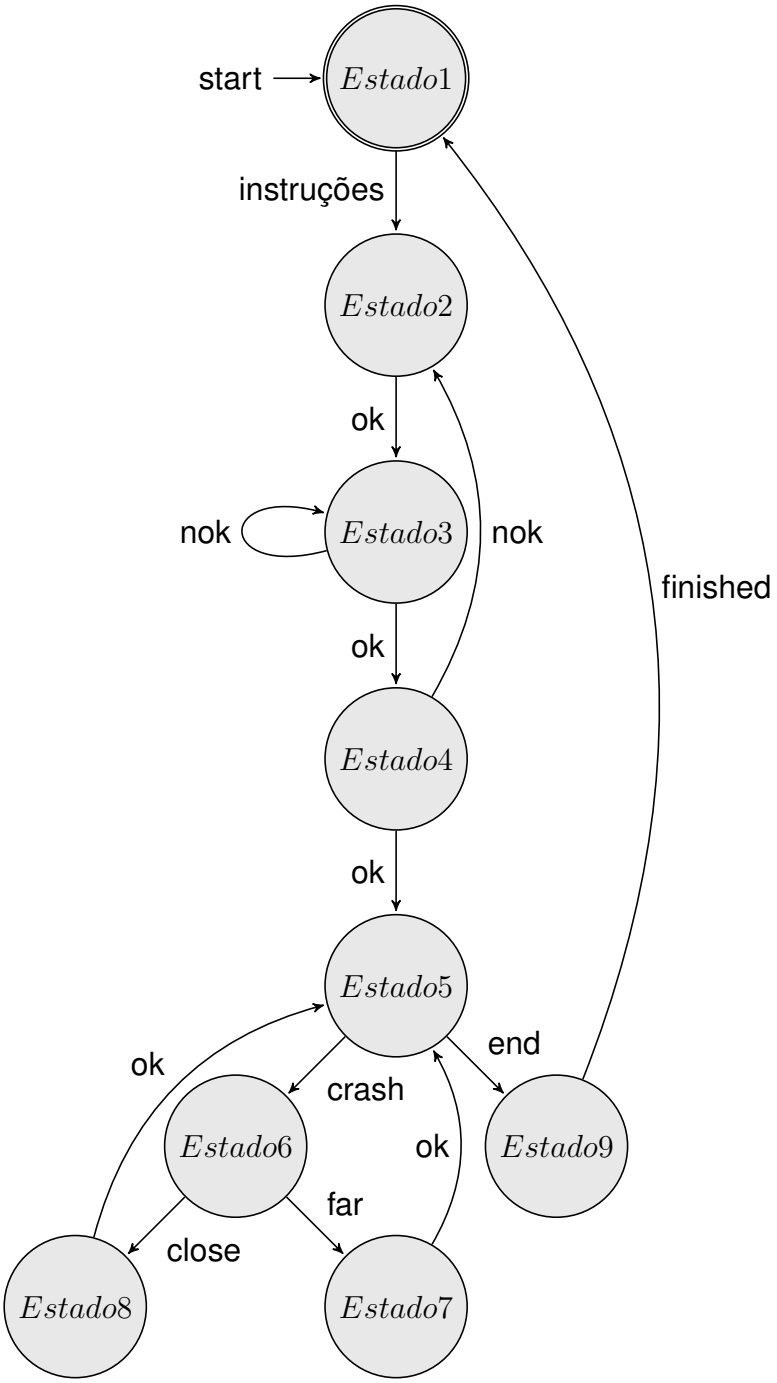


Figura 17 – Máquina de Estados do Veículo

Fonte: Elaborado pelo autor

APÊNDICE C - IMPLEMENTAÇÃO DA MÁQUINA DE ESTADOS DOS DIÁLOGOS NO SPADE

Implementação da máquina de estados utilizando o SPADE em Python, para descrever os diálogos de inscrição do veículo no ambiente e envio do caminho para o servidor.

```
import time

from spade.agent import Agent
from spade.behaviour import FSMBehaviour, State
from spade.message import Message

STATE_ONE = "STATE_ONE"
STATE_TWO = "STATE_TWO"
STATE_THREE = "STATE_THREE"
STATE_FOUR = "STATE_FOUR"



class VehicleFSMBehaviour(FSMBehaviour):
    async def on_start(self):
        print(f"FSM starting at initial state {self.current_state}")

    async def on_end(self):
        print(f"FSM finished at state {self.current_state}")
        await self.agent.stop()


class StateOne(State):
    async def run(self):
        print("Vehicle sending subscribe to server")
        msg = Message(to=str(self.agent.jid))
        msg.set_metadata("performative", "subscribe")
        msg.body = "vehicle subscribe"
        await self.send(msg)
        self.set_next_state(STATE_TWO)


class StateTwo(State):
    async def run(self):
        ans = await self.receive(timeout=3)
        if ans and 'accept' in ans.metadata.values():
```

```
37              self.set_next_state(STATE_THREE)
38          elif ans and 'reject' in ans.metadata.values():
39              time.sleep(10)
40              self.set_next_state(STATE_ONE)
41
42
43  class StateThree(State):
44      async def run(self):
45          print("Vehicle sending path to server")
46          msg = Message(to=str(self.agent.jid))
47          msg.set_metadata("performative", "inform")
48          msg.body = self.path
49          await self.send(msg)
50          self.set_next_state(STATE_FOUR)
51
52  class StateFour(State):
53      async def run(self):
54              print("Vehicle sending path to server")
55
56  class FSMAgent(Agent):
57      async def setup(self):
58          fsm = VehicleFSMBehaviour()
59          fsm.add_state(name=STATE_ONE, state=StateOne(), initial=True)
60          fsm.add_state(name=STATE_TWO, state=StateTwo())
61          fsm.add_state(name=STATE_THREE, state=StateThree())
62          fsm.add_state(name=STATE_FOUR, state=StateFour())
63          fsm.add_transition(source=STATE_ONE, dest=STATE_TWO)
64          fsm.add_transition(source=STATE_TWO, dest=STATE_THREE)
65          fsm.add_transition(source=STATE_TWO, dest=STATE_ONE)
66          fsm.add_transition(source=STATE_THREE, dest=STATE_FOUR)
67          self.add_behaviour(fsm)
68
69
```

Listing 1 – Código dispositivo BLE

APÊNDICE D - IMPLEMENTAÇÃO DO ALGORITMO DE BUSCA A*

Implementação do algoritmo de busca A* em python, utilizado na Seção 3.4.2. Possui uma classe caracterizando o nó de cada ponto da busca e uma função que realiza a busca dentro de um labirinto definido, recebendo a posição inicial definida e final desejada.

```
1
2     class Node():
3     """A node class for A* Pathfinding"""
4
5     def __init__(self, parent=None, position=None):
6         self.parent = parent
7         self.position = position
8
9         self.g = 0
10        self.h = 0
11        self.f = 0
12
13    def __eq__(self, other):
14        return self.position == other.position
15
16 def astar(maze, start, end):
17    """Returns a list of tuples as a path from the given start to the
      given end in the given maze"""
18
19    # Create start and end node
20    start_node = Node(None, start)
21    start_node.g = start_node.h = start_node.f = 0
22    end_node = Node(None, end)
23    end_node.g = end_node.h = end_node.f = 0
24
25    # Initialize both open and closed list
26    open_list = []
27    closed_list = []
28
29    # Add the start node
30    open_list.append(start_node)
31
32    # Loop until you find the end
33    while len(open_list) > 0:
34
35        # Get the current node
36        current_node = open_list[0]
37        current_index = 0
```

```
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        # Pop current off open list, add to closed list
        open_list.pop(current_index)
        closed_list.append(current_node)

        # Found the goal
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1]  # Return reversed path

        # Generate children
        children = []
        #for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1)
    , (-1, 1), (1, -1), (1, 1)]:  # Adjacent squares
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1)
    ]:

            # Get node position
            node_position = (current_node.position[0] + new_position[0],
     current_node.position[1] + new_position[1])

            # Make sure within range
            if node_position[0] > (len(maze) - 1) or node_position[0] <
    0 or node_position[1] > (
                    len(maze[len(maze) - 1]) - 1) or node_position[1] <
    0:
                continue

            # Make sure walkable terrain
            if maze[node_position[0]][node_position[1]] != 0:
                continue

            # Create new node
            new_node = Node(current_node, node_position)

            # Append
            children.append(new_node)

        # Loop through children
```

```
80          for child in children:
81
82              # Child is on the closed list
83              for closed_child in closed_list:
84                  if child == closed_child:
85                      continue
86
87              # Create the f, g, and h values
88              child.g = current_node.g + 1
89              child.h = ((child.position[0] - end_node.position[0]) ** 2)
    + (
90                      (child.position[1] - end_node.position[1]) ** 2)
91              child.f = child.g + child.h
92
93              # Child is already in the open list
94              for open_node in open_list:
95                  if child == open_node and child.g > open_node.g:
96                      continue
97
98              # Add the child to the open list
99              open_list.append(child)
100
```

Listing 2 – Código Busca A*