UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VINICIUS CUSTODIO

IMPLEMENTAÇÃO DE AGENTE INTELIGENTE, MODELADO COMO UM VEÍCULO AUTÔNOMO, PARA DESVIO DE OBSTÁCULOS ESTÁTICOS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2017

VINICIUS CUSTODIO

IMPLEMENTAÇÃO DE AGENTE INTELIGENTE, MODELADO COMO UM VEÍCULO AUTÔNOMO, PARA DESVIO DE OBSTÁCULOS ESTÁTICOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel, em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Gleifer Vaz Alves

PONTA GROSSA
2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Ponta Grossa
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO DE AGENTE INTELIGENTE, MODELADO COMO UM VEÍCULO AUTÔNOMO, PARA DESVIO DE OBSTÁCULOS ESTÁTICOS

por

VINICIUS CUSTODIO

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 08 de junho de 2017 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr Gleifer Vaz Alves
Orientador

_____
Prof. Dr. André Pinz Borges
Membro titular

_____
Prof. Dr. Erikson Freitas de Morais
Membro titular

_____
Prof. Dr. Ionildo José Sanches
Responsável pelo Trabalho de Conclusão de Curso

_____
Prof. Dr. Erikson Freitas de Morais
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

RESUMO

CUSTODIO, Vinicius. **Implementação de Agente Inteligente, Modelado como um Veículo Autônomo, para Desvio de Obstáculos Estáticos**. 2017. 65f. Trabalho de Conclusão de Curso em Ciência da Computação, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

Os veículos autônomos são uma revolução para a indústria automotiva e estão cada vez mais próximos da realidade, sendo já desenvolvidos por grandes empresas como Google e Tesla. A área de veículos autônomos só tende a crescer nos próximos anos e vem para modificar e ampliar a estrutura da sociedade em termos de como os veículos são utilizados e como são as interações com os usuários. O desenvolvimento requer a junção de diversas áreas, uma delas é a de agentes inteligentes. Dentro de agentes inteligentes é possível incorporar diversas técnicas, uma dessas é para o desvio de obstáculos. Existem diversas formas de se implementar um agente inteligente, dentre elas a a arquitetura BDI (Belief, Desire e Intetion), onde o agente possui crenças, desejos e intenções, implementada em um framework chamado JaCaMo. Para que o agente consiga atuar é necessário um ambiente, para esse fim o Simbad é utilizado, correspondendo esse a simulador que permite a construção de ambientes com objetos estáticos e dinâmicos (robôs). O ambiente definido para esse trabalho possui somente obstáculos estáticos, que estão definidos dentro de faixas, e o agente transita entre as faixas para desviar desses obstáculos. O simulador também é responsável por fornecer as informações ao agente, de forma que esse seja capaz de controlar a sua representação dentro do simulador. Para que as informações fossem trocadas, um protocolo de comunicação foi definido, e permite que o agente receba informações sobre o ambiente e enviei as ações que devem ser realizadas para sua representação dentro do simulador. Após a integração do simulador testes foram realizados com o agente para obter informações de suas capacidades, sendo possível notar que dentro do cenário definido o agente consegue realizar as manobras propostas.

Palavras-chaves: Agentes Inteligentes. Simulação. Framework JaCaMo. Veículos Autônomos. Desvio de obstáculos

ABSTRACT

CUSTODIO, Vinicius. **Implementation of Intelligent Agent, Modeled as an Autonomous Vehicle, for Obstacle Avoidance**. 2017. 65f. Work of Conclusion Course Graduation in Computer Science, Federal Technological University - Paraná. Ponta Grossa, 2017.

Autonomous cars are a revolution for the automotive industry and are becoming closer to reality, already being developed by big companies like Google, Uber and Tesla. The area of autonomous vehicles only tends to grown in the coming years and comes to modify and broaden the structure of society in terms of how cars are used and how are the interactions with users.The development requires the joining of several areas, one of which is the area of intelligent agents, within intelligent agents it is possible to incorporate several techniques to the agents, one of these techniques is so that it can avoid obstacles. There are several ways to implement an intelligent agent, this work uses the BDI (Belief, Desire and Intention) architecture, where the agent has beliefs, desires and intentions, a framework that implements this architecture is JaCaMo. In order for the agent to be able to act, an environment is necessary. For this purpose, Simbad is used, which corresponds to a simulator that allows the construction of an environment with static and dynamic objects (robots). The environment defined for this work has only static obstacles, which are defined within tracks, and the agent transits between them to avoid the obstacles. The simulator is also responsible for providing the information to the agent, so that the agent is able to control its representation within the simulator. In order for the information to be exchanged a communication protocol was defined, and allows the agent to receive information about the environment and sent the actions that must be performed for its representation within the simulator. After the integration of the simulator tests were performed with the agent to obtain information of their capabilities, being possible to note that within the defined scenario the agent can perform the proposed maneuvers.

Key-words: Intelligent Agents. Simulation. Framework JaCaMo. Autonomous Vehicles. Obstacle avoidance

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

Os veículos autônomos são um grande avanço tecnológico para a indústria automotiva. Nos últimos anos a indústria parece estar à beira de uma mudança revolucionária, que mudará não somente o mercado competitivo, mas também a forma das cidades e como se utilizam os veículos (WALLACE; SILBER, 2012).

Considerando essa tecnologia emergente o seguinte cenário é possível: são 6 horas da noite, imagine que você está saindo do trabalho. Porém, ainda tem que fazer diversas coisas antes de dormir. Você retira o celular do bolso e requisita um carro. Alguns minutos depois um carro sem motorista estaciona a sua frente. Você entra e diz "Casa". No percurso, você faz suas atividades restantes. Quando chega em casa pode usar o tempo para relaxar e curtir a família. Quando você sai do carro ele se move para o próximo cliente (WALLACE; SILBER, 2012).

O exemplo citado por Wallace e Silber (2012), já é uma realidade próxima. Apesar de ainda não possuir uma regulação que permita que esses veículos navegam por conta própria, diversos veículos autônomos já estão em fase de teste assistida por motorista. Duas empresas na área são a Google e a Tesla. A Google possui diversos carros autônomos em fase de teste, circulando por cidades nos Estados Unidos (WAYMO, 2016). A Tesla por sua vez, já tem um modelo semiautônomo em comercialização, o Tesla Model S (TESLA, 2016).

As empresas citadas estão em busca do desenvolvimento de um veículo autônomo de nível 5. Essa classificação é feita pela agência NHTSA (*National Highway Traffic Safety Administration*) dos Estados Unidos e possui 5 níveis: Nível 1, o motorista controla o veículo; Nível 2, o veículo possui alguma função automatizada, como freio automático ou controle de estabilidade eletrônico; Nível 3, duas funções são automatizadas ao mesmo tempo, como controle de distância e se manter na faixa; Nível 4, em certas condições o motorista não é necessário e o veículo possui controle sobre todas as funções, inclusive as críticas; Nível 5, o veículo possui total controle sobre todas as funções e não é esperado que o motorista assuma o veículo em nenhum momento, podendo o veículo estar desocupado (NHTSA, 2012).

A navegação de um veículo nível 5, requer que o carro esteja preparado para todas as possíveis situações. Tal nível de autonomia requer que o carro tenha controle sobre todas as funções vitais como: acelerar, frear, respeitar sinais de trânsito, detectar e desviar de obstáculos (LEVISON, 2011).

Considerando a função de desvio de obstáculos, é possível constatar que os obstáculos podem ser de diferentes formas, tamanhos, estáticos ou dinâmicos, podendo assumir a forma de um pedestre, uma obra, ou mesmo de outro carro. A detecção de obstáculos é a responsável pela diferenciação de tais, e as técnicas de desvio pela tomada de decisão do agente. Nesse contexto, um carro diariamente se depara com diversos obstáculos no trânsito, necessitando que aconteça a detecção e desvio com segurança.

Distintas técnicas na área de agentes e sistemas multiagentes são utilizadas para realizar

o desvio de obstáculos. A técnica *Reciprocal Velocity Obstacle* (RVO) (BERG; LIN; MANOCHA, 2008) e o *Elliptic Limit-cycles* (ELC) (DAFFLON *et al.*, 2015) são dois exemplos.

O método ELC tem como princípio as leis newtonianas, que por meio de interações de atração e repulsão o agente consegue desviar de obstáculos. O método RVO garante o desvio de forma reativa de obstáculos estáticos e dinâmicos. Este método é uma extensão do *Velocity Obstacle* (VO), o qual garante que é possível evitar uma colisão por meio da análise de todas as possíveis velocidades no qual o agente pode colidir. Ambas as técnicas, quando embarcadas em um agente viabilizam o desvio de obstáculos.

Um agente é definido como um sistema de computador localizado em um ambiente, e que possui a capacidade para ações autônomas para alcançar seu objetivo a qual ele foi projetado (WOOLDRIDGE; JENNINGS, 1995). Assim um agente (neste caso, um veículo autônomo) consegue, através da análise do ambiente, reagir às modificações que possam ocorrer ao seu redor de modo a realizar seu objetivo.

O comportamento reativo é uma característica pertinente a agentes, existindo algumas classificações para definir seus tipos, uma delas é segundo a tomada de decisão, sendo eles reativos, racionais e híbridos. Os agentes reativos, são capazes de manter uma interação contínua com o ambiente e responder rapidamente a possíveis mudanças (WOOLDRIDGE; JENNINGS, 1994). Os agentes racionais são capazes de tomar a iniciativa, planejar e agir racionalmente e não baseados em eventos (WOOLDRIDGE; JENNINGS, 1994). Por sua vez os híbridos correspondem a uma combinação entre os reativos e racionais, aonde esses podem reagir a eventos e tomar iniciativa.

Nesse trabalho o agente será inserido em um ambiente simulado, sendo o agente um mecanismo de alto nível, responsável pela tomada de decisão. O simulador será responsável por fornecer as informações sobre o ambiente, além de permitir a visualização gráfica das ações.

O Simbad (2017) foi o simulador escolhido para esse trabalho. Ele permite a criação de ambientes e a adição de recursos como sensores e câmeras que captam informações dos mesmos. Outra opção seria o V-REP (COPPELIA, 2017), simulador com diversas funcionalidades e componentes com alto grau de detalhamento e realidade, porém seus componentes e funcionalidades dificultam a aprendizagem, não sendo interessante ao projeto neste estágio de desenvolvimento pois, esse se encontra nas primeiras fases de desenvolvimento.

Os principais motivos para a escolha do Simbad é a facilidade para a criação de ambiente, que é implementado em Java e que também viabiliza a conexão com o framework JaCaMo, e por ser um simulador simples, que possui funcionalidades que satisfazem esse trabalho. Algumas dessas funcionalidades são: controle sobre a velocidade, sensores para localização de obstáculos e câmeras para visualização do ambiente do ponto de vista do agente.

As implementações do simulador e do agente serão evolutivas, partindo de ambientes e decisões simples para ambientes mais complexos e que requerem decisões mais difíceis. Para a criação do ambiente serão definidas regras de como deve ser realizado a construção dos obs-

táculos dentro do ambiente, regras como: tamanho dos obstáculos e disposição entre eles. Para o agente serão definidas regras de movimentação como: as manobras disponíveis e a velocidade de movimentação.

A proposta desse trabalho é implementar um agente capaz de desviar de obstáculos estáticos, dentro do ambiente definido, com um conjunto de manobras. Para o ambiente serão considerados três faixas, sendo que o agente alterna entre essas utilizando as manobras disponíveis para desviar dos obstáculos.

1.1 OBJETIVOS

A seguir são descritos o objetivo geral e os objetivos específicos para a realização desse trabalho.

- Objetivo Geral
  - Implementar um agente inteligente como um veículo, capaz de desviar de obstáculos estáticos.
- Objetivos Específicos
  - Levantar os fundamentos de agentes e veículos autônomos;
  - Investigar técnicas de desvio de obstáculos que utilizam agentes (em especial para agentes autônomos);
  - Definir e representar os tipos de obstáculos estáticos que podem ser encontrados pelo veículo autônomo;
  - Elaborar planos para o agente desviar de obstáculos;
  - Implementar o agente, seus respectivos planos e objetivos por meio da linguagem Jason;
  - Implementar artefatos usando o framework Cartago;
  - Construir a integração entre o JaCa (Jason e Cartago) com o simulador Simbad;
  - Implementar os ambientes (cenários), no simulador Simbad;
  - Analisar os planos implementados para o agente.

1.2 JUSTIFICATIVA

Os veículos autônomos já são o próximo avanço tecnológico (WALLACE; SILBER, 2012) e apesar de não estarem totalmente desenvolvidos, estima-se que em 2020 os primeiros

veículos autônomos já estarão sendo comercializados. Eles irão trazer diversos benefícios a sociedade, como a diminuição de emissão de dióxido de carbono (CO2) e a redução de acidentes. Esta redução é prevista em até 90% dos acidentes, quando os veículos autônomos se tornarem o principal meio de transporte, além do aumento em 50 minutos do tempo livre dos motoristas (BERTONCELLO; WEE, 2015).

Por meio das técnicas de desvio de obstáculos, espera-se que os veículos autônomos tenham a capacidade de diminuir o número de acidentes no trânsito. As técnicas procuram garantir as ações necessárias à segurança dos ocupantes e daqueles que estão próximos do perímetro.

Portanto, este trabalho visa contribuir com a implementação de um agente com a autonomia para desviar de obstáculos para o projeto AVIA (*Autonomous Vehicles with Intelligent Agents*) do Grupo de Pesquisa em Agentes de Software (GPAS). Essa pesquisa é um dos primeiros trabalhos pertencentes ao projeto AVIA, que busca a criação de veículos autônomos, utilizando agentes inteligentes como mecanismo de decisão.

As implementações aqui sugeridas e desenvolvidas contribuíram para os próximos trabalhos que serão desenvolvidos pelo projeto. Note que o foco deste trabalho é a implementação do agente com capacidade de desviar de obstáculos, o problema da detecção dos mesmos não será tratado, podendo ser objeto de estudo em pesquisas futuras.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está divido em cinco capítulos. O segundo capítulo possui uma visão geral sobre o atual estado de desenvolvimento dos veículos autônomos tanto em relação a software como hardware. O terceiro capítulo possui as definições sobre agentes, seus tipos, uma visão geral sobre a arquitetura *Belief, Desire and Intetion* e duas técnicas para desvio de obstáculos (RVO e ELC). Além disso o capitulo três possui as ferramentas que são utilizados nesse trabalho: o framework Jacamo, plataforma para implementação de sistemas multiagentes e o simulador Simbad. O quarto capítulo explica o desenvolvido desse trabalho, entrando em detalhes principalmente na implementação do agente. O quinto capítulo, os resultados dos testes realizados em ambientes definidos que verificam a capacidade do agente. O último capítulo possui a conclusão e os trabalhos futuros.

2 VEÍCULOS AUTÔNOMOS

Os veículos autônomos trazem consigo não somente mudanças tecnológicas, mas também mudanças que permitirão criar um trânsito mais seguro e acessível a todos, pois esses são desenvolvidos para ultrapassarem a capacidade humana. Novas oportunidades e empreendimentos irão surgir juntamente com os carros autônomos, como carros autônomos para corridas compartilhadas e serviços de taxis. Além disso, as cidades serão reestruturadas para melhorar o design de forma que seja possível a integração de todos os elementos (WALLACE; SILBER, 2012).

Atualmente a mobilidade não atinge seu nível máximo de aproveitamento, por ser ineficiente e cara. Primeiro, o valor dos veículos, valor do veículo acrescido dos custos de circulação, é em média de US$21.000,00 para cada 15.000 milhas percorridas por ano, somando mais de US$40.000,00 dólares ao longo de cinco anos, para uma máquina que é utilizada em média duas horas por dia (WALLACE; SILBER, 2012). A média de horas utilizadas do veículo corresponde a somente 5% de sua capacidade, esperasse que com os veículos autônomos e o compartilhamento desses, por sistemas como Uber e Lift, a utilização dos veículos aumente para pelo menos 75% do tempo, diminuindo o custo médio de cada carro (CLIFFE, 2012).

Cada cidadão Americano gasta em média 250 horas por ano em um veículo. Os motoristas perdem tempo em congestionamentos e buscando por estacionamento. De acordo com o (MIT, 2016), cada motorista gasta aproximadamente 40% da gasolina do veículo, procurando por estacionamento. Além disso estacionamentos e garagens formam zonas mortas, chegando a mais de um terço de toda a área de algumas cidades dos Estados Unidos (WALLACE et al., 2012).

Estima-se que em 2010 nos Estados Unidos, houveram 6 milhões de acidentes que resultaram em 32.788 mortes no trânsito. Dentre esses, 93% deles são atribuídos a falhas humanas, e o restante a falhas mecânicas. Distrações correspondem a 18% dos acidentes com feridos, e 2.3 milhões tiveram que recorrer ao sistema de tratamento de emergência (WALLACE et al., 2012). Com o desenvolvimento dos veículos autônomos, esperasse que esse número seja reduzido e possivelmente chegue a zero.

Segundo a (NHTSA, 2012) o objetivo dos carros autônomos não deve ser de alcançar a capacidade humana, mas de superar, tornando-os totalmente seguros e livre de falhas. Nos últimos anos os carros autônomos veem evoluindo rapidamente tanto no quesito software como hardware, nas seções a seguir veremos componentes de hardware e software que compõem os veículos autônomos.

2.1 HARDWARE

Um grande passo para o desenvolvimento de carros autônomos foi dado pela DARPA (*Defense Advanced Research Projects Agency*). Ela é responsável pelo *Grand Challenge*, uma competição com o objetivo de impulsionar o desenvolvimento de tecnologias voltadas a carros autônomos. A primeira competição foi realizada em 2004, e não possuiu nenhum vencedor. A partir de 2005 todas as competições possuíram finalistas sendo a edição de 2007 voltada para ambientes urbanos e possuindo como um dos finalistas o Passat Volkswagen, mostrado na Figura 1 (LEVISON, 2011).

Figura 1 – Veículo Autônomo da Universidade de Stanford, finalistas do Grand Challenge de 2007.

Fonte: Levison (2011).

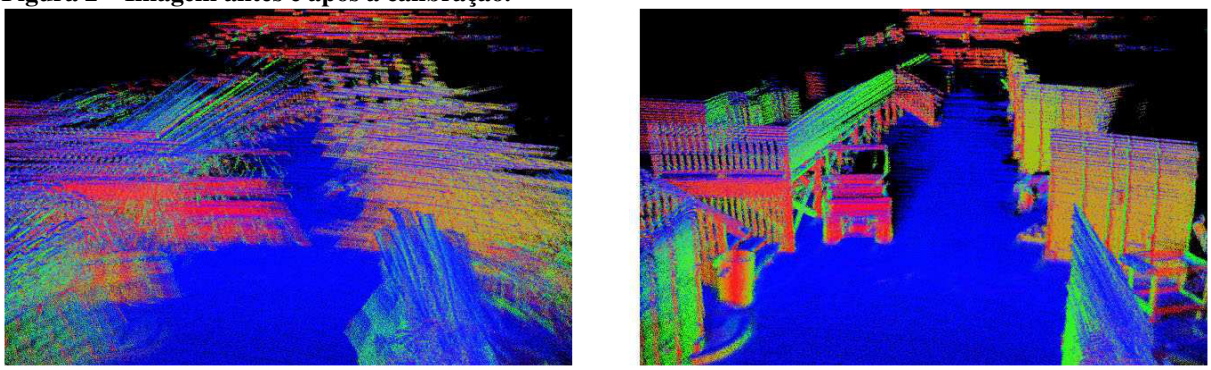
Na Figura 1, é possível observar diversos sensores acoplados ao carro que auxiliam na visualização das ruas, pedestres, carros e possíveis obstáculos que podem ser encontrados. Nele estão embutidos um sensor Velodyne HDL-64E S2 LiDAR (*Light Detection and Ranging*), que garante a visualização do ambiente em 360 graus; Quatro Câmeras (uma esférica, duas câmeras para visão frontal e uma câmera de alta resolução para visão frontal) para uma visão completa em todas as direções; Seis radares, que fazem a verificação da traseira, frente e laterais; Dois SICK LD-LRS LiDAR scanners em pontos não cobertos pelos outros sensores; Um sistema GPS. Além desses sensores o veículo possui diversos componentes de software que fazem a utilização das informações recebidas dos sensores para planejar as ações do carro (LEVISON, 2011).

2.2 SOFTWARE

Os componentes de software são peças fundamentais dentro dos veículos autônomos, pois são eles que permitem que o veículo navegue com segurança. No veículo da Figura 1, os seguintes algoritmos são utilizados para navegação: calibração automática dos lasers, mapeamento e localização, reconhecimento de objetos, planejamento da trajetória, controle e modelagem dinâmica, detecção de semáforos, detecção de sinais de trânsito e classificação de sinais de parada independente da direção (LEVISON, 2011). Cada algoritmo possui um objetivo, a seguir tem-se uma breve descrição de cada algoritmo citado:

- Algoritmo de calibração automática: esse algoritmo realiza a calibração dos lasers, sendo necessário para garantir que as imagens captadas pelos lasers correspondam ao ambiente, para que seja possível extrair informações das imagens. Na Figura 2 são ilustradas duas imagens, à esquerda a imagem captada pelos sensores sem calibração e à direita com os sensores calibrados, para calibrar a imagem foram necessárias 300 iterações do algoritmo de calibração, sobre os sensores que captam a imagem.

Figura 2 – Imagem antes e após a calibração.



Fonte: Levison (2011).

- Algoritmo de mapeamento: este algoritmo consome as informações geradas pelo GPS e pelo Velodyne LiDAR (*Light Imaging, Detection And Ranging*) (LEVISON, 2011). O Velodyne LiDAR utiliza um laser para iluminar os objetos em sua mira e então medir a distância para o objeto. Com isso temos um mapa do terreno, sendo possível identificar a localização do veículo.

- Algoritmo para reconhecimento de objetos: é utilizado para que o carro consiga planejar suas ações corretamente. Saber diferenciar a localização de pedestres e ciclistas em intersecções permite ao carro realizar manobras sofisticadas e prever com antecedência o possível movimento de ciclistas e/ou pedestres. O algoritmo de reconhecimento, no veículo de Levison (2011) é capaz de reconhecer pedestres, ciclistas e carros e é feito pela segmentação das imagens geradas pelo Velodyne, a partir de cada objeto reconhecido e

feito o *tracking*, processo para identificar a trajetória do objeto. As falhas na segmentação e no *tracking* correspondem a principal limitação do reconhecimento de objetos.

- Algoritmo para planejamento de trajetórias: os veículos autônomos necessitam compreender a dinâmica do tráfego para a realização de manobras como: trocar de faixa, entrar em outra faixa com fluxo de carros ou desviar de outros veículos. O planejamento de trajetórias é responsável por definir as manobras que serão realizadas. Sendo que esse módulo gera as informações necessárias para o módulo de controle e modelagem dinâmica.
- O algoritmo de controle e modelagem dinâmica: este algoritmo recebe as informações geradas pelo planejamento e define os passos para que a manobra definida ocorra, os passos aqui são as ações de acelerar, frear e definir a direção do veículo.
- O algoritmo de detecção de sinais de trânsito e classificação de sinais de parada: responsável por localizar placas ao redor do veículo, sendo capaz de distinguir placas de paradas, mesmo quando vistas da parte traseira, permitindo ao veículo diferenciar cruzamento simples (dois pontos de parada) de cruzamentos duplos (quatro pontos de parada).
- Algoritmo para detecção de semáforos: além do algoritmo para detecção de placas de parada o veículo em Levison (2011) possui um módulo para a detecção de semáforos, que é realizada por uma câmera que detecta o estado do semáforo (vermelho, amarelo ou verde), sendo que este módulo não identifica a localização do semáforo e somente o estado.

2.3 NÍVEIS DE AUTONOMIA

Um veículo para ser totalmente autônomo depende do seu software, necessitando que esse consiga controlar todas as funções corretamente. Dependendo de suas capacidades, os veículos autônomos podem ser classificados em níveis de autonomia. A *National Highway Traffic Safety Administration* (NHTSA, 2012), é responsável pela regulamentação dos níveis de autonomia, dividindo os em cinco níveis:

- Nível 1: Veículo sem autonomia. O motorista possui total controle sobre o veículo. E é responsável por todas as manobras e operações.
- Nível 2: Funções específicas. Consiste na automação de uma ou mais funções específicas (exemplo: câmbio automático), podendo possuir múltiplas funções que operam separadamente. O motorista é responsável por todas as operações que possam gerar riscos.
- Nível 3: Funções combinadas. Ao menos duas funções estão automatizadas e trabalham em conjunto para aliviar o número de funções do motorista (controle de faixa e aceleração). Podendo executar funções primárias como acelerar e frear.

- Nível 4: Direção autônoma limitada. O veículo é capaz de ter controle sobre todas as funções críticas, desde que esteja em um ambiente amigável, ou seja, possui o conhecimento de quais ações devem ser tomadas. Porém o motorista deve estar atento, pois caso o veículo encontre uma situação desconhecida este devolverá a direção ao motorista.
- Nível 5: Direção autônoma, o veículo está preparado para realizar todas as funções críticas e monitorar todo o ambiente ao seu redor. O motorista passa a ser um ocupante, não sendo esperado que este realize ações para que o veículo alcance seu destino.

2.4 EMPRESAS

Diversas empresas já estão trabalhando no desenvolvimento de veículos autônomos. Algumas dessas empresas são a Google e a Tesla, que buscam a produção de um veículo autônomo nível 5. Estas já possuem projetos com alto nível de desenvolvimento e estão realizando testes em vias públicas para analisar e aprimorar seus veículos.

2.4.1 Google

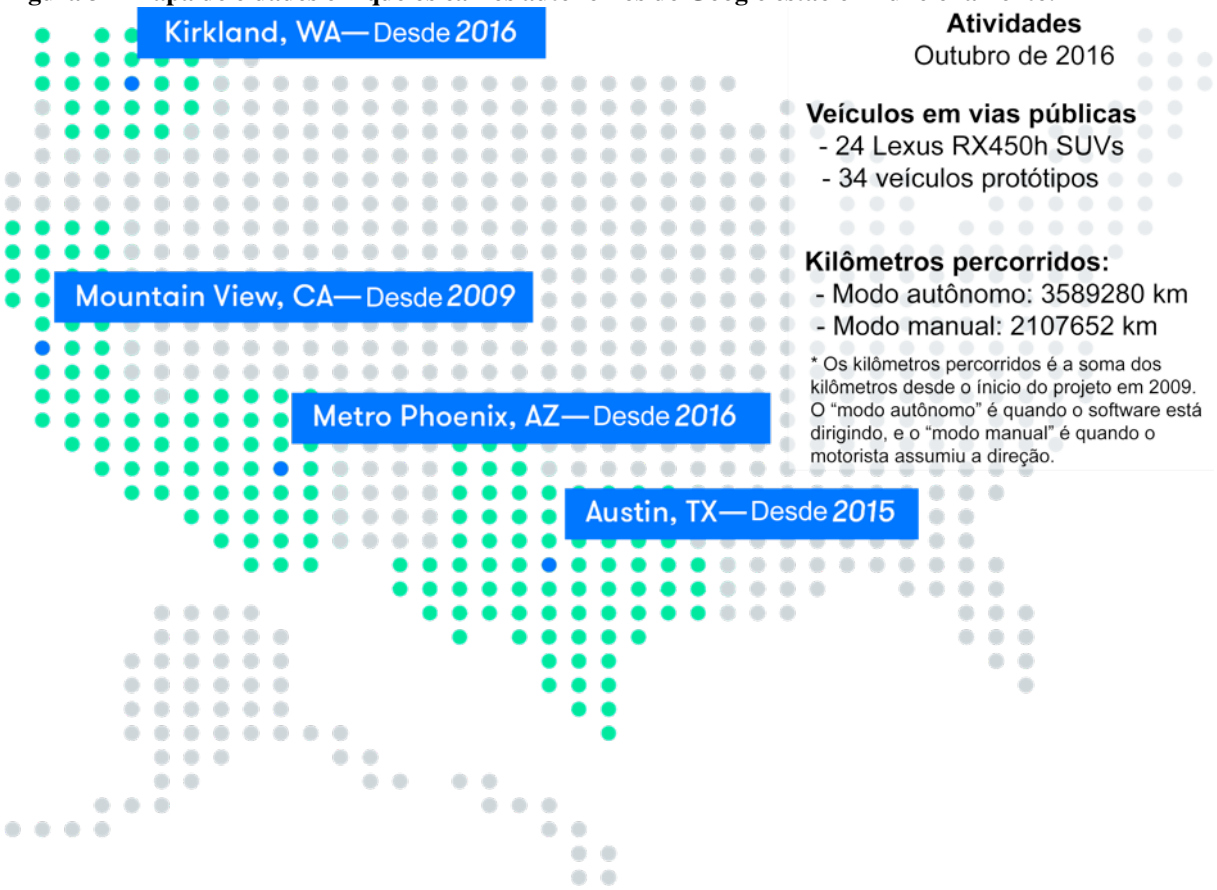
O Waymo, carro autônomo em desenvolvimento pela Google, teve o início em 2009, e já possui diversos veículos de teste em cidades dos Estados Unidos. Na Figura 3, o mapa mostra as cidades de atuação do Waymo, possuindo atualmente 58 veículos, (24 Lexus SUVs, e 34 protótipos de veículos, ambos buscando o nível 5 de autonomia) circulando por vias públicas, com mais de 3,5 milhões de quilômetros no modo autônomo (software dirigindo o veículo), e 2,1 milhões no modo manual (motorista assume a direção) (WAYMO, 2016).

O veículo projetado pelo Google, é baseado em quatro perguntas: "Aonde estou?", "O que está ao meu redor?", "O que vai acontecer?" e "O que devo fazer?", que definem os planos de ações do veículo. Assim como no Passat (LEVISON, 2011), o carro possui diversos sensores que determinam a posição atual, rua e faixa em que ele está localizado, além de detectar os objetos ao redor. Após detecção o software classifica-os baseados em tamanho, forma e padrão de movimento. Com os objetos definidos o software prevê o que o objeto provavelmente fará, e então decide, como atuar no cenário atual (WAYMO, 2016).

2.4.2 Tesla

A Tesla é uma das empresas que já faz a comercialização de carros autônomos. A partir de setembro de 2014, todos os carros da linha Model S eram equipados com uma câmera, um

Figura 3 – Mapa de cidades em que os carros autônomos do Google estão em funcionamento.

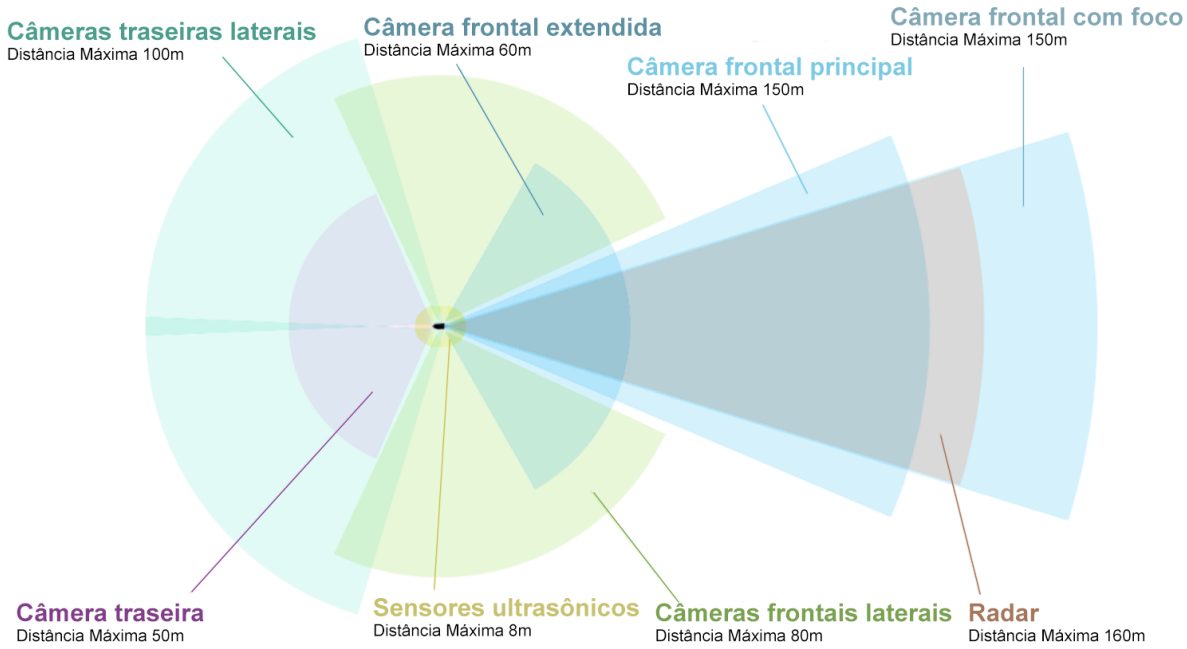


Fonte: Waymo (2016)

radar na dianteira, sensores acústicos ultrassônicos dianteiros e traseiros o que permite ao veículo detectar sinais de trânsito, faixas na rua, obstáculos e outros veículos. O Model S, é categorizado entre os níveis 3 e 4 pela NHTSA, ou seja, o sistema de navegação pode ser ativado quando o veículo estiver em um ambiente familiar a qual ele consegue reagir devidamente (TESLA, 2016).

Em outubro de 2016 a Tesla anunciou um novo sistema para navegação autônoma, além da atualização no hardware para que o sistema funcione corretamente, sendo colocado imediatamente em comercialização. Apesar do sistema ainda depender da regulamentação da NHTSA para entrar em funcionamento, Elon Musk, dono da Tesla Motors, afirma que a partir da regulamentação todos os carros adquiridos com o hardware para navegação autônoma, receberão a atualização de software automaticamente. O hardware é composto por: câmeras laterais com sentido frontal, câmera frontal com visão expandida, câmera frontal principal, câmera frontal com foco a distância, sensores ultrassônicos, câmeras laterais com sentido traseiro, uma câmera traseira e um radar (Figura 4). Esses sensores combinados garante uma visão de 360 graus em torno do carro e alcance de até 250m (TESLA, 2016).

Figura 4 – Sensores para navegação autônoma, Tesla Model S.



Fonte: Tesla (2016)

2.5 VEÍCULOS AUTÔNOMOS NO BRASIL

No Brasil existem alguns projetos em desenvolvimento dentro de universidades como Universidade de São Paulo (USP, 2017), Universidade Federal de Minas Gerais (UFMG, 2017) e Universidade Federal de Itajubá (UNIFEI, 2017). Os projetos mais avançados são os da (USP, 2017), um desses é o projeto desenvolvido em parceria com a empresa de caminhões Scania, e o outro projeto, o Carro Robótico Inteligente para Navegação Autônoma (CaRINA).

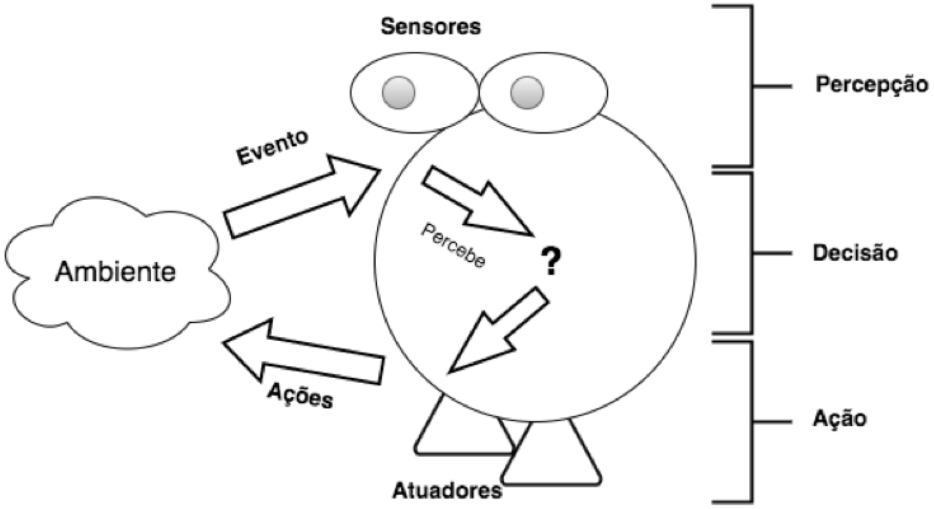
O protótipo de caminhão desenvolvido é capaz de navegar em estradas com fluxo constante, identificar obstáculos e realizar manobras. Mesmo com essas habilidades, o veículo não possui uma estratégia de teste como o Waymo que transita em ruas públicas para coletar informações e então o aperfeiçoamento a navegação, sendo testado somente em ambiente fechados.

O CaRINA, por sua vez é um carro autônomo em desenvolvimento. Seu protótipo começou a ser concebido em 2011, e em 2013 foi realizado o primeiro teste em vias públicas na cidade de São Carlos-SP.

3 AGENTES INGELIGENTES

Wooldridge (2009) define um agente como um sistema de computador, localizado em um ambiente e com a capacidade de interagir autonomamente com este. O agente ele possui a capacidade de perceber o ambiente e a partir dos eventos que ocorram, ele possui a autonomia e controle sobre suas ações de forma que por meio dos atuadores, consegue interferir no meio.

Figura 5 – O agente captura as mudanças no ambiente e produz uma saída de acordo com os acontecimentos.



Fonte: Wooldridge (2009)

Na Figura 5, Wooldridge divide o agente em três partes. A percepção é a parte responsável por reconhecer os eventos do ambiente, por meio dos sensores. A decisão é a fase realizada após o reconhecimento de um evento, pelos sensores e dispara uma ação de acordo com o evento percebido. A ação, parte final, é onde os aturadores interagirão com o ambiente, de acordo com as informações dos módulos anteriores.

Wooldridge (2009) afirma que um dos maiores problemas para os agentes é definir que ações deverão ser executadas e em qual ordem elas serão executadas para que o agente consiga realizar seu propósito. Para isso Wooldridge elenca que todo agente deve possuir reatividade, proatividade, habilidades sociais e autonomia.

A autonomia para Wooldridge (2009) assume que o agente é planejado para um objetivo específico, e não deixa de agir mesmo que não esteja sobre supervisão de uma pessoa, além de possuir controle sobre seus estados internos. O controle sobre os estados internos permite ao agente tomar decisões que o aproximam de seu objetivo sem a intervenção humana.

A Reatividade é a capacidade que o agente tem de perceber e responder rapidamente a mudanças que ocorram no ambiente. Proatividade é a capacidade de tomar a iniciativa em comportamentos direcionados ao objetivo. Habilidades sociais como a capacidade de interagir com outros agentes para alcançar seus objetivos. Utilizando essas características, Wooldridge (2009) define três tipos de agentes: reativos, racionais e híbridos.

3.1 AGENTE REATIVO

O Agente reativo possui duas características principais. A primeira é que as decisões são feitas a partir de um grupo eventos que o agente possui conhecimento. Sendo que cada evento é mapeado a uma decisão. A decisão escolhida tem relacionamento com objetivo do agente, buscando suprir o objetivo. É importante notar que esse tipo de agente não reflete sobre suas ações a longo prazo, sempre que um evento ocorrer a ação será executada. O conjunto de eventos e ações corresponde a um mapeamento do tipo: evento ação (WOOLDRIDGE, 2009).

A segunda característica que define um agente reativo, é que um evento pode disparar mais de uma ação, devendo existir um mecanismo de escolha entre as ações. Wooldridge (2009), propõe que cada ação possua uma prioridade, e que ações com maior prioridade são as que devem ser executadas.

As ações são realizadas a partir do conjunto de comportamentos que o agente possui. Considere P como o conjunto de percepções (condições), A o conjunto de ações e a função $see(s)$ como a percepção do agente. Cada comportamento corresponde a um par ordenado (c, a), onde $c \subseteq P$, um conjunto de percepções (condições), e $a \in A$, é uma ação. O comportamento (c, a) vai ser realizado quando o ambiente estiver no estado $s \in A$, se e somente se $see(s) \in c$. Considerando que $Beh = (c, a) \mid c \subseteq P \wedge a \in A$, como conjunto de todos os comportamento (WOOLDRIDGE, 2009).

3.2 AGENTE RACIONAL

Um agente racional, é aquele que possui a inteligência para realizar escolhas, que o aproximam de seu objetivo. O agente possui a representação do ambiente, como um banco de dados. O banco de dados possui as informações sobre o ambiente, e são mapeadas de acordo com as percepções. A função $see(s)$, funciona da mesma forma que no agente reativo, sendo responsável pela percepção do ambiente. A nova função $next : D \times Per \rightarrow D$, é responsável por mapear uma informação D e uma percepção Per a uma nova informação D, e a função $action : D \rightarrow Ac$, mapea uma informação recebida do ambiente D em uma ação Ac Wooldridge (2009).

Usando das definições de Wooldridge (2009), para Russel e Norvig (2010) todos agentes são racionais, e são divididos em subclasses baseadas nas suas capacidades, ações e objetivos.

> "Um agente é definido pela capacidade de analisar o ambiente através de sensores e atuar no ambiente com seus atuadores. Assim como um "agente humano" possui olhos, orelhas e outros orgãos como sensores e mãos, pernas, bocas e outras partes como atuadores. Um agente robô possui câmeras, infravermelhos e outros como sensores e diversos motores como atuadores." (RUSSEL; NORVIG, 2010)

O agente definido por Russel e Norvig (2010) possui a capacidade de analisar e atuar sobre o ambiente, sendo considerado racional, pois realiza aquilo que é esperado e correto. As ações corretas são aquelas que aproximam o agente do sucesso. Sendo necessário a análise das ações e do ambiente para definir o grau de sucesso (RUSSEL; NORVIG, 2010).

Uma forma de definir o grau de sucesso de um agente é de acordo com sua performance. Para a analisar o agente é necessário estabelecer padrões de acordo com seu o ambiente e o objetivo. Supondo um agente aspirador de pó, uma forma de verificar a performance é de acordo com o nível de sujeira em um ambiente após a limpeza. Os padrões para avaliação definirão se o agente realizou com êxito seu objetivo, uma escolha ineficiente de padrões pode resultar em uma avaliação inconsistente. Considerando o cenário do aspirador, uma avaliação da primeira hora de trabalho do agente prejudica aqueles que não possuem um ciclo rápido.

Segundo Russel e Norvig (2010) um agente racional em qualquer momento depende de quatro características.

- Ser capaz de definir um grau de sucesso;
- Possuir um estado interno que controle os acontecimentos;
- Informações prévias sobre o ambiente;
- E as ações que devem ser tomadas no ambiente.

Um agente racional deve ser capaz de responder a cada sequência de acontecimentos, com informações dos acontecimentos anteriores (experiência) e com os estados internos que esse possui, que representam o ambiente. O conhecimento do ambiente permite ao agente reagir aos acontecimentos, porém se ações forem somente realizadas de acordo com esses conhecimentos o agente não possuirá autonomia (RUSSEL; NORVIG, 2010).

Russel e Norvig (2010) definem que o comportamento do agente racional deve ser baseado em suas experiências e conhecimentos. A autonomia do sistema é definida pela capacidade de determinar seu comportamento, baseado em suas próprias experiências. Assim como a evolução em animais garante o conhecimento suficiente para a sobrevivência ao período inicial, os agentes devem também possuir o mesmo mecanismo. Devem ser construídos com conhecimentos iniciais e com a habilidade de aprendizagem através de suas experiências.

3.3 AGENTE HÍBRIDO

Segundo Wooldridge (2009), um terceiro tipo de agente pode ser definido unindo os agentes racionais e reativos, os chamados agentes híbridos, que tentam resolver problemas tanto de agentes reativos como agentes racionais.

Um problema que pode ser citado para os agentes reativos é que eles somente consideram as informações locais (estado atual), gerando comportamentos errôneos quando informações de outros estados são necessárias. Uma possível limitação aos agentes racionais é a demora para escolha e execução dos plano (WOOLDRIDGE, 2009).

O agente híbrido atua tanto de forma rápida frente a situações que possam surgir no ambiente, como também possui capacidades de agentes racionais para armazenar estados prévios, e utilizá-los em planos em que informações prévias são necessárias (WOOLDRIDGE, 2009).

3.4 SISTEMAS MULTIAGENTES

Segundo Jennings, Sycara e Wooldridge (1998), existem vários casos em que um único agente é o suficiente para a solução de um problema. Contudo, para outros casos se faz necessário a utilização de sistemas multiagentes.

Sistemas multiagentes são sistemas compostos de múltiplos agentes que relacionam-se mutuamente, em sua maioria, através da cooperação (ações conjuntas em busca de um objetivo em comum), coordenação (ações de diferentes agentes organizadas para que as interações sejam benéficas e os conflitos, evitados), negociação (busca de um acordo em que todas as partes envolvidas na ação estejam satisfeitas) (WOOLDRIDGE, 2009) e navegação (encontrar um caminho livre de colisão para um agente se locomover de uma posição a outra) (DUCATELLE *et al.*, 2014).

Quadro 1 – Exemplos de agentes.

Agentes	Percepção	Ações	Objetivo	Ambiente
Sistema de diagnóstico médico	Sintomas, resposta do paciente	Perguntas, testes, tratamentos	Curar o paciente, diminuir custos	Paciente, Hospital
Sistema de análise de imagens de satélite	Diferenciação dos pixels, cores	Categorizar diferentes cenários	Categorização correta	Imagens de satélites em órbita
Robô coletor de partes	Diferenciação dos pixels	Coletar as partes e agrupar em categorias	Correto agrupamento das partes	Linhas de montagem com esteira
Controle de uma refinaria	Leitura de temperatura e pressão	Abrir e fechar válvulas; Ajustar a temperatura	Aumentar a pureza, produção e segurança	Refinaria
Instrutor interativo de inglês	Palavras digitadas	Exercícios, sugestões e correções	Aumentar as notas do estudante	Grupo de estudantes

Fonte: Russel e Norvig (2010).

No Quadro 1, é possível notar que os agentes possuem quatro camadas: percepções,

ações, objetivos e ambiente. De acordo com seus objetivos o agente possuirá diferentes habilidades, por exemplo, o agente robô coletor de partes, tem como objetivo o agrupamento das partes coletadas. Para isso ele precisa reconhecer as partes da linha de montagem, e possuir a habilidade para pegar a peça da esteira e organizá-las. Para que os agentes possuem essa habilidade, é necessário a utilização de alguma arquitetura, umas delas é a arquitetura de Crenças, Desejo e Intenções. Onde as crenças serão as percepções, os desejos os objetivos do agente e as intenções as ações

3.5 CRENÇAS, DESEJOS E INTENÇÕES

A arquitetura mais comum para agentes racionais é a BDI (*Belief, Desire and Intention*), onde agente que possui crenças, desejos e intenções, que são utilizados para controlar as ações com o ambiente (WOOLDRIDGE, 2009). Segundo Bordini, Hübner e Wooldridge (2007) crenças, desejos e intenções podem ser definidos como:

- Crenças: informações que o agente tem o sobre o ambiente.
- Desejos: objetivos que o agente quer alcançar.
- Intenções: como o agente vai alcançar seus desejos.

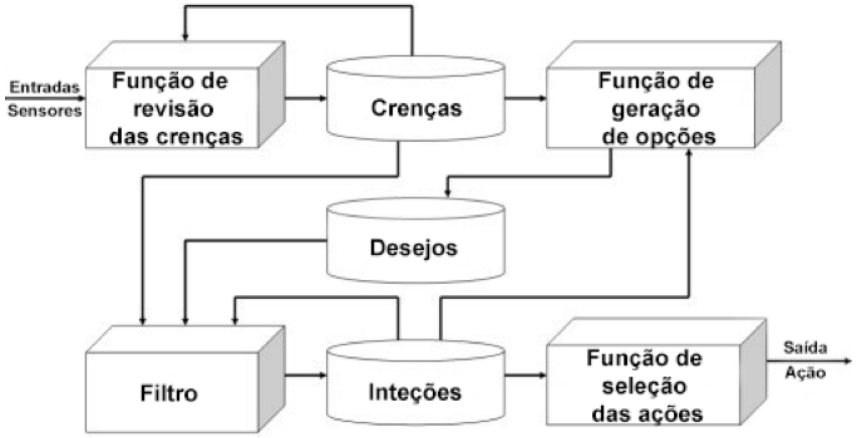
Esse modelo utiliza noções mentais para representar o raciocínio prático. O raciocínio prático é uma questão de pesar as diferentes considerações a favor e contra das possíveis alternativas disponíveis, aonde as considerações relevantes são fornecidas pelos desejos que o agente acredita (BRATMAN, 1990, p. 17).

O raciocínio prático utiliza os desejos, crenças e intenções para definir quais serão os objetivos a serem realizados. A definição dos objetivos é conhecida como deliberação. Além disso, a partir dessas noções são definidos os planos para atingir os objetivos escolhidos, sendo esse processo chamado de raciocínio meio e fim.

O exemplo a seguir é dado por Wooldridge (2009) ajuda a compreender as definições de deliberação e raciocínio meio e fim.

> "Considerando um aluno no final de sua graduação, ele precisa tomar diversas escolhas. A primeira qual carreira ele deve seguir na academia ou indústria. Esse processo decisório de escolher uma carreira é a deliberação. Depois de selecionado uma carreira, outras escolhas precisam ser realizadas em particular, como alcançar essa carreira. Supondo que após a deliberação, o aluno tenha escolhido a carreira acadêmica. O próximo passo é planejar como alcançar esse objetivo, esse planejamento é o raciocínio meio e fim. O resultado do raciocínio é um plano para atingir o objetivo escolhido. Para o exemplo da carreira acadêmica, um plano pode envolver se inscrever para o mestrado em uma universidade." (WOOLDRIDGE, 2009)

Figura 6 – Raciocínio prático em um agente com arquitetura BDI.



Fonte: Russel e Norvig (2010)

O processo de raciocínio prático é apresentado na Figura 6. Com o acontecimento de um evento, as crenças do agente são revisadas, podendo haver alterações no conjunto de crenças do agente, sendo a função de revisão de crenças responsável por essas alterações. A função de geração de opções define os desejos baseados nas crenças e intenções. A função filtro é a deliberação e escolhe quais serão as intenções do agente. Por fim a função de seleção das ações é responsável pelo raciocínio meio fim, ou seja, como as intenções serão cumpridas. Uma forma de implementar um agente BDI é através do framework Jacamo, especificamente por meio da linguagem Jason.

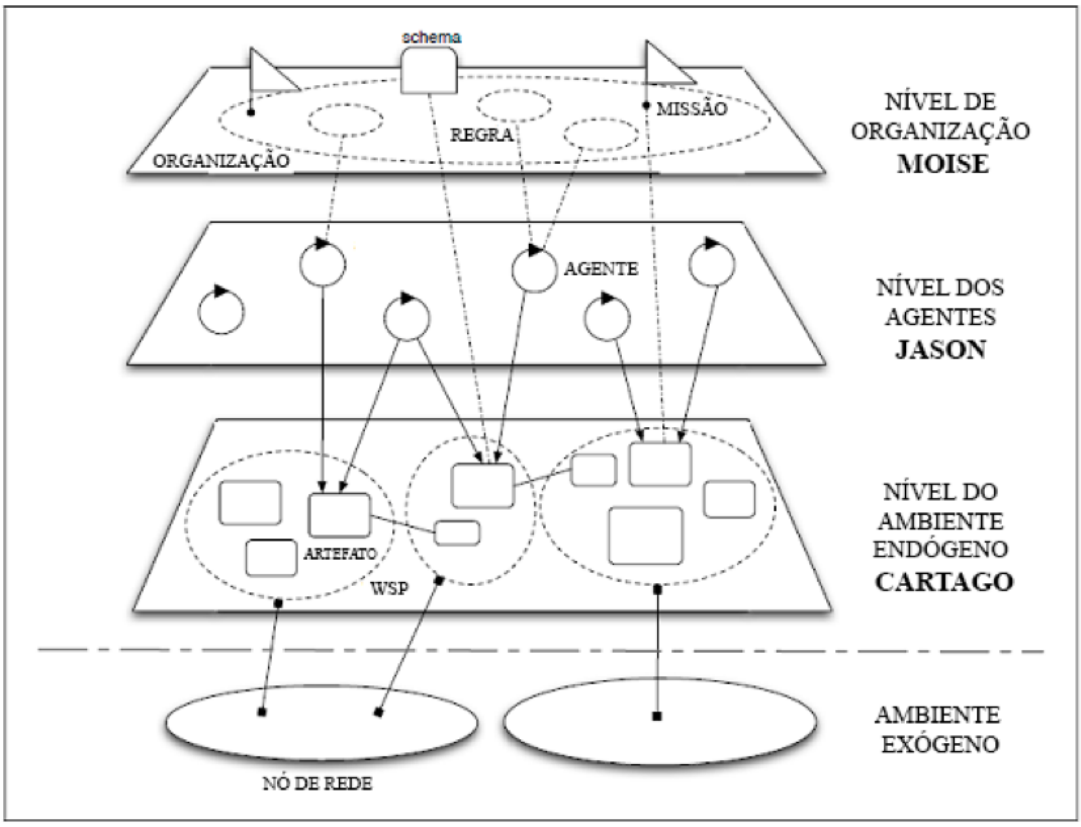
3.6 FRAMEWORK JACAMO

O JaCaMo é um framework que integra três ferramentas (Jason, Cartago e Moise), cada ferramenta corresponde a um nível de um sistema multiagente, Figura 7.

Cada um dos níveis apresentados na Figura 7 corresponde a uma plataforma independente dentro do framework JaCaMo: (i) nível de organização correspondente ao programa Moise, responsável pela organização de agentes BDI; (ii) nível dos agentes, o programa responsável por esse nível é o Jason e é uma ferramenta para implementação dos agentes; (iii) nível de ambiente, é o ambiente em que os agentes atuarão e que é implementado pelo Cartago.

Devido a natureza desse trabalho, a organização dos agentes não é um ponto a ser abordado. Nas próximas seções serão vistos somente as linguagens Jason e Cartago (JACAMO, 2017).

Figura 7 – Níveis dos sistemas multiagentes no JaCaMo.



Fonte: JaCaMo (2017)

3.6.1 Jason

A linguagem Jason é uma extensão do AgentSpeak, uma linguagem que utiliza a arquitetura BDI. A seguir serão descritas as funções que a linguagem possui, a partir de três categorias (crenças, objetivos e planos) (BORDINI; HüBNER; WOOLDRIDGE, 2007).

3.6.1.1 Crenças

As crenças dentro do Jason correspondem a uma coleção de informações, e são representadas na forma de predicados, sendo possível criar crenças através do relacionamento de dois objetos: `tall(john) ou likes(john,music)`.

O predicado, `tall(john)` indica que John possui uma característica, no caso que ele é alto, o segundo que John gosta de música. O agente possuir a crença "John gosta de música" significa que ele acredita que isso seja verdade.

Jason possui duas formas de negação, utilizando o operador `not`, através da negação por falha onde se `not p` for avaliado e falhar então será verdadeiro. Outro tipo de negação é pelo operador $\sim$, conhecido também como negação forte e é usada para quando o agente acre-

dita que uma crença é falsa. Por exemplo `colour(box1,white)`, significa que a é branca e ∼colour(box1,white) é quando o agente acredita que a `box` não é branca.

As crenças representam estados os quais o agente acredita que esses sejam verdades, dentro da arquitetura BDI quando o agente possui um estado de incerteza ou falsidade, este é um estado chamado objetivo.

3.6.1.2 Objetivos

Objetivos correspondem à estados do ambiente, onde agente atua para que estes eles tornem verdade, sendo conhecido como objetivo por realização. Um outro tipo de objetivos é o de teste, que é utilizado para verificar o estado de alguma crença que o agente possui.

O objetivo de teste é denotado por "?". Por exemplo: `?bank_balance(BB)`, significa que o agente vai verificar em suas crenças e buscar pela informação do saldo atual.

Na linguagem Jason objetivos por realização são definidos pelo operador "!", ele é declarado por exemplo como `!own(house)`, fazendo assim com que o agente queria ser dono da casa. O agente a partir da definição do objetivo executará um plano para que esse objetivo seja confirmado como verdade. A execução do plano deve resultar com que `own(house)` passe a ser verdade, ou seja, ao fim do plano o agente será dono da casa, o plano é um conjunto de passos que faz com que o agente se torne dono da casa no exemplo acima. Na seção a seguir veremos o funcionamento dos planos.

3.6.1.3 Planos

Um plano é constituído por três partes: evento de disparo, contexto e corpo:

$$\texttt{evento\_disparo} : \texttt{contexto} \leftarrow \texttt{corpo}$$

Onde:

- Evento de disparo: corresponde à mudanças no ambiente que afetam diretamente os objetivos e as crenças do agente, podendo ser a exclusão ou a criação de uma crença ou objetivo;
- Contexto: o contexto é utilizado como critério de decisão a execução de diferentes planos. O contexto é um subconjunto de crenças e regras que definem qual o plano será executado, quando essas crenças e regras definidas no contexto possuírem o mesmo valor que as

crenças do agente o plano passa a ser considerado como um candidato a execução;

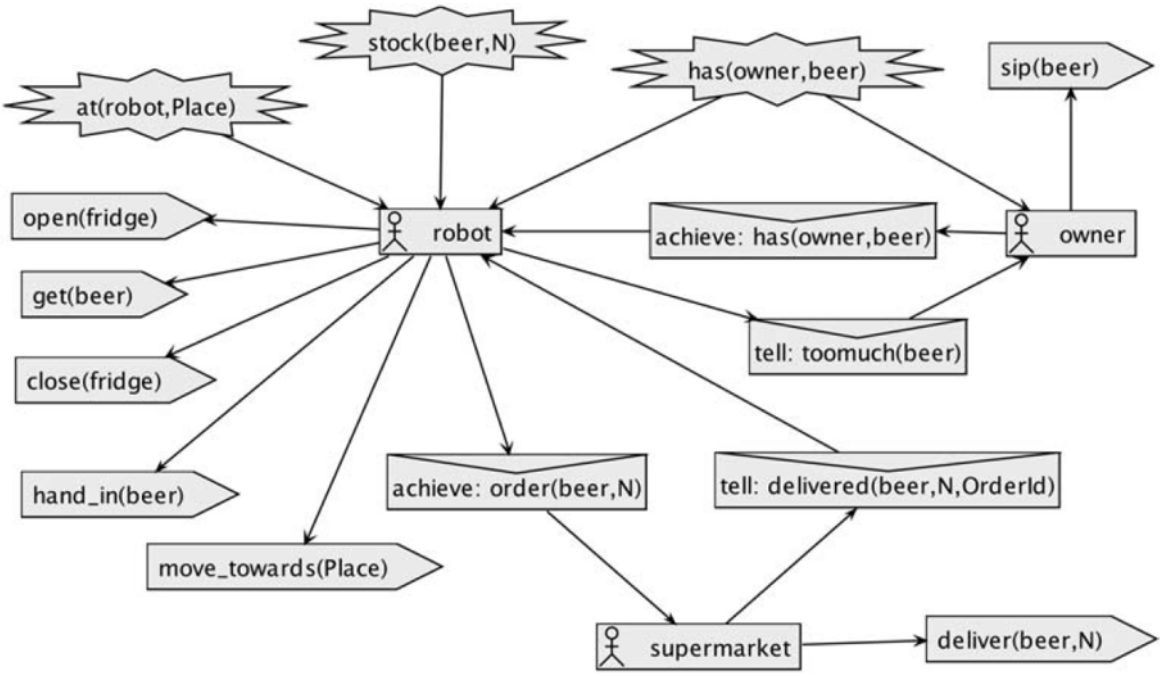
- Corpo: corresponde aos passos do agente a serem executados pelo plano, podendo haver outros objetivos que serão referidos aqui como subobjetivos, e consequentemente subplanos, que serão necessários para que o objetivo final seja cumprido.

3.6.1.4 Exemplo - Robô doméstico

O exemplo apresentado nesta seção é descrito em Bordini, Hübner e Wooldridge (2007).

> "Um robô doméstico possui o objetivo de servir cervejas a seu dono. Sua missão é simples, ele recebe uma requisição de seu dono, se dirige a geladeira, pega a cerveja e a leva a seu dono. Ele também toma conta do estoque de cervejas (eventualmente ele faz requisições de cervejas para o serviço de entrega do supermercado), e possui regras definidas sobre o consumo máximo diário, de acordo com o Departamento de Saúde." (BORDINI; HüBNER; WOOLDRIDGE, 2007)

Figura 8 – Diagrama do robô doméstico através da metodologia Prometheus.



Fonte: Bordini, Hübner e Wooldridge (2007)

A Figura 8 é feita através da metodologia Prometheus, possuindo representações para as percepções, ações, mensagens e agentes (Figura 9).

As percepções correspondem a mudanças no ambiente, as ações são planos executados para alcançar um objetivo, as mensagens são informações enviadas entre os agentes e o agente é o responsável por coordenar as percepções, ações e mensagens.

Figura 9 – Percepção, mensagem, ação e agente respectivamente na metodologia Prometheus.

Fonte: Bordini, Hübner e Wooldridge (2007)

No exemplo temos, o agente `robot` com as percepções `at`, `stock` e `has` e o agente `owner` com a percepção `has`:

- `at(robot, Place)`: percebe aonde o agente `robot` está, neste exemplo os lugares foram simplificados para `fridge` (em frente a geladeira) e `owner` (próximo ao dono).
- `stock(beer, N)`: quando o agente `robot` estiver com a geladeira aberta ele perceberá quantas cervejas ainda estão em estoque.
- `has(owner,beer)`: os agentes (`robot` e `owner`) perceberão quando o `owner` possuir uma garrafa vazia.

Além disso, os agentes `robot`, `supermarket` e `owner` possuem as seguintes ações:

- Robot
  - `open(fridge)`: abre a geladeira;
  - `get(beer)`: pega a cerveja;
  - `close(fridge)`: fecha a geladeira;
  - `hand_in(beer)`: entrega a cerveja ao dono;
  - `move_towards(Place)`: se dirige ao lugar especificado podendo ser `fridge` ou `owner`;
- Supermarket
  - `deliver(beer, N)`: faz a entrega das cervejas;
- Owner
  - `sip(beer)`: enquanto o agente `owner` possuir uma cerveja ele deve continuar bebendo.

E as seguintes mensagens, `achieve` para requisição e `tell` para resposta, são trocadas entre os agentes `robot`, `supermarket` e `owner`:

- `achieve: order(beer,N)`, mensagem em que o agente `robot` requisita a reposição do estoque ao agente `supermarket`;

- `tell: delivered(beer,N,OrderId)`, mensagem onde o agente `supermarket` repassa as informações sobre o pedido ao agente `robot`;
- `achieve: has (owner,beer)`, mensagem em que o agente `owner` requisita cervejas ao agente `robot`;
- `tell: toomuch(beer)`, mensagem que o agente `robot` informa ao agente `owner` que esse já atingiu o consumo máximo de cervejas.

Nesse trabalho somente o Código 1 pertencente ao agente robô de Bordini, Hübner e Wooldridge (2007), sendo somente um exemplo para especificar a linguagem Jason.

Código 1 – Agente Robot

```
1  // Crencas iniciais
2  // Inicialmente, o agente acredita que existem algumas cervejas na
       geladeira
3  available(beer,fridge).
4
5  // o dono nao pode consumir mais de 10 cervejas por dia
6  limit(beer, 10)
7
8  /* Regras */
9  too_much(B) :-
10     .date(YY, MM, DD) &
11     .count(consumed(YY, MM, DD, \_, \_, \_, B), QtdB) &
12     limit(B, Limit) &
13     QtdB > Limit
14
15 /* Planos */
16 +!has(owner, beer)
17     : available(beer,fridge) & not too_much(beer)
18     <- !at(robot, fridge);
19         open(fridge);
20         get(beer);
21         close(fridge);
22         !at(robot,owner);
23         hand_in(beer);
24         // salvar que outra cerveja sera consumida
25         .date(YY, MM, DD); .time(HH, NN, SS);
26         +consumed(YY, MM, DD, HH, NN, SS, beer).
27
28 +!has(owner, beer)
29     : not available (beer, fridge)
30     <- .send(supermarket, achieve, order (beer, 5));
31         !at(robot, fridge).
32         // ir ate a geladeira e esperar la
33
```

```
34 +!has(owner,beer)
35     : too_much(beer) & limit(beer,L)
36     <- .concat("The Department of Health does not allow me ",
37         "to give you more than ", L,
38         " beers a day! I am very sorry about that!",M);
39         .send(owner,tell,msg(M)).
40
41 +!at(robot,P) : at(robot,P) <- true.
42
43
44 +!at(robot,P) : not at(robot,P)
45     <- move_towards(P);
46         !at(robot,P).
47
48 // quando receber a informacoes de entrega tentar
49 // executar o 'has' novamente
50 +delivered(beer,Qtd,OrderId)[source(supermarket)] : true
51     <- +available(beer,fridge);
52         !has(owner,beer).
53
54 // quando a geladeira e aberta o agente percebe o estoque
55 // e atualiza a crenca de cervejas disponiveis
56 +stock(beer,0)
57     : available(beer,fridge)
58     <- -available(beer,fridge).
59
60 +stock(beer,N)
61     : N > 0 & not available(beer,fridge)
62     <- +available(beer,fridge).
```

Fonte – Bordini, Hübner e Wooldridge (2007)

Inicialmente é definida a crença de que o robô acredita que existe alguma cerveja dentro da geladeira, linha 3.

Na linha 6, temos a definição de uma crença que limita o consumo diário a 10 cervejas por dia e uma regra chamada `too_much(B)` (linhas 9-13), que pode ser invocada quando é necessário verificar se o consumo máximo diário já foi atingido. Com as crenças estabelecidas, três planos são definidos para os cenários relacionado a entrega da cerveja.

O primeiro plano, linhas 16 à 26, é o caso quando a crença de cervejas disponíveis é verdade e o consumo diário ainda não foi alcançado. O plano é executado da seguinte forma: o robô se dirige a geladeira, abre ela, pega cerveja, fecha a geladeira, se dirige ao dono, entrega a cerveja e registra que na data de atual foi consumido uma cerveja.

O segundo plano, linhas 28 à 31, quando não existem cervejas disponíveis, o contexto para a execução é que o agente acredite que não existam cervejas na geladeira, a execução plano consiste na requisição de compra ao agente mercado e locomoção até a geladeira.

E o último plano desse cenário, linhas 34 à 39, quando não existem cervejas disponíveis, o contexto para a execução é que o agente acredite que não existam cervejas na geladeira, a execução plano consiste na requisição de compra ao agente mercado e locomoção até a geladeira.

Com a definição dos planos que realizam a entrega da cerveja, é necessário agora definir planos que verificam se o agente `robot` está no lugar correto para entregar a cerveja, nesse caso dois planos foram definidos.

O primeiro plano, linha 41, verifica se o agente está em uma determinada posição `T`. Caso ele não esteja na posição `P`, o segundo plano (linhas 44-46) será invocado e realizará a movimentação do agente através da função `move_towards(P)` (linha 45), para alcançar a posição desejada.

Com a capacidade de movimentação e de entrega das cervejas, o agente precisa verificar se existem ou não cervejas disponíveis, e quando receber uma entrega do mercado essas informações devem ser atualizadas.

O plano definido nas linhas 50 à 52, atualiza o número de cervejas disponíveis quando esse recebe a entrega do agente `supermarket`. Quando a entrega é feita, uma mensagem do agente `supermarket` informa a quantidade de cervejas entregue e o número do pedido, a partir disso a crença de cervejas na geladeira é atualizada e tenta executar o plano `!has(owner, beer)` novamente.

O plano das linhas 56 à 58, é executado quando a geladeira estiver aberta e não possuir mais cervejas disponíveis, nesse plano o agente remove a crença que indica que cervejas estão disponíveis (linha 58). E o último plano, linhas 60 à 62, é o caso contrário onde o agente não possui a crença que existem cervejas disponíveis, mas ele percebe que existe estoque, nesse caso a crença de cervejas disponíveis será adicionada com a quantidade percebida em estoque.

Durante o Código 1, as ações `.concat` (linha 36, para concatenar a *string*), `.send` (linha 30, enviar uma mensagem) e `.date` (linha 25). Essas são ações internas e nativas da linguagem Jason e podem ser realizadas por qualquer agente implementado.

3.6.2 Cartago

O CArtAgO (Common Artifact infrastructure for Agentes Open environments), é uma ferramenta para criação de ambiente virtuais através de artefatos (programados em Java) e possui integração com a ferramenta Jason. As principais funções do Cartago são descritas a seguir (RICCI; VIROLI; OMICINI, 2006).

- *Workspace*: É o ambiente, que pode ser composto por um ou mais *workspaces*, um agente implementado em Jason pode participar de um ou mais *workspaces* simultaneamente, sendo inicialmente designado a um *workspace* padrão necessitando entrar em outro ambiente caso queria atuar neles.

- Ações disponíveis: é dado por um conjunto de ações definidas pelos artefatos do ambiente, sendo um mapeamento um-para-um entre ações e operações, implicando também que se uma operação tem sucesso ou falha sua ação possuirá o resultado respectivo
- Artefatos padrões: estão disponíveis em cada um dos *workspaces*, permitindo funções básicas como criar, verificar, deletar, conectar e focar artefatos do ambiente assim como criar e unir *workspaces*.
- Mapeamento de propriedades e eventos em crenças: com o foco em um artefato suas propriedades são mapeadas em crenças do agente, sendo um processo automático, por sua vez seus eventos também podem ser mapeados em crenças, porém esse não corresponde a processo automático.
- Java *data-biding*: as estruturas de Java são como os artefatos são representados e manipulados.

A criação de artefatos é feita pelo agente, com a função `makeArtifact` (Código 2, linha 2), para a criação do artefato os parâmetros necessários são: i) nome do artefato; ii) classe do artefato; iii) parâmetros da função `init` (função dentro do artefato, herdada do java); iv) variável para salvar o identificador do artefato.

Código 2 – Criação e utilização do artefato através do Jason.

```
1 +!setupSimulator <-
2                     makeArtifact("a_Simulator",
3                                  "autonomous_vehicles.Simulator_artifact",
4                                  ["Starting"],
5                                  ArtId);
6                     focus(ArtId);
7                     startSimulator[artifact_id(ArtId)].
```

Fonte – Autoria Própria.

A partir do artefato é possível acessar funções internas a esse para realizar procedimentos que sejam necessários. A função `focus` na linha 6, garante que o agente esteja observando o artefato desejado, e assim seja possível chamar a função do artefato `startSimulator`(linha 7), o parâmetro `artifact_id(ArtId)` identifica o artefato a qual a função `startSimulator` pertence, não aparecendo na declaração da função dentro do artefato (Código 3).

Código 3 – Operação para iniciar o simulador.

```
1 @OPERATION void startSimulator () {
2     clientCommunication.sendMessage("0");
3 }
```

Fonte – Autoria Própria.

O Código 3, é a operação que pode ser invocada pelo agente. Para que o método seja invocado é necessário a utilização da anotação `@OPERATION` (linha 1). Outras anotações possíveis

são `@OPSTEP` e `@GUARD`. `@OPSTEP` define um método que é executado antes do método invocado e `@GUARD` define uma condição para que um método seja invocado.

Com a utilização do Cartago é possível implementar ambientes através de seus artefatos, *workspaces*. Apesar de não possuir nenhum modelo de agentes ou plataforma, ele está relacionado a noção de agentes inteligentes, em especial com a arquitetura BDI.

3.7 SIMBAD SIMULADOR

Simbad é um simulador *open source* com fins científicos e educacionais. Focado principalmente para o desenvolvimento e estudo de Inteligência Artificial, Aprendizado de Máquina, Robôs Autônomos e Agentes Autônomos (SIMBAD, 2017).

O simulador é composto por um ambiente, que possui objetos estáticos e o robô, que é um agente, e ambos são customizáveis, dentro do simulador o ambiente é criado a partir da classe `EnvironmentDescription` Código 4, através do método construtor da classe do ambiente é possível construir os obstáculos, e criar o agente (robô).

Código 4 – Criação do ambiente no simbad, Java.

```
1  static public class MyEnv extends EnvironmentDescription {
2      public MyEnv() {
3          light1IsOn = true;
4          light2IsOn = false;
5          // Top
6          Wall w1 = new Wall(new Vector3d(9, 0, 0), 6, 1, this);
7          w1.rotate90(1);
8          add(w1);
9          //Bottom
10         Wall w2 = new Wall(new Vector3d(-9, 0, 0), 6, 1, this);
11         w2.rotate90(1);
12         add(w2);
13         //Right
14         Wall w3 = new Wall(new Vector3d(0, 0, 3), 19, 1, this);
15         add(w3);
16         //Left
17         Wall w4 = new Wall(new Vector3d(0, 0, -3), 19, 1, this);
18         add(w4);
19
20         Box b1 = new Box(new Vector3d(-3, 0, 0),
21                          new Vector3f(1, 1, 1), this);
22         add(b1);
23
24         Arch arch = new Arch(new Vector3d(-6, 0, 0), this);
25         arch.rotate90(3);
26         add(arch);
```

```
27
28          Robot robot = new Robot(new Vector3d(0, 0, 0), "robot 1");
29          add(robot);
30      }
31  }
```

Fonte – Autoria Própria.

O Código 4 posiciona os objetos e o agente dentro do ambiente. As variáveis `light1IsOn` e `light2IsOn` (respectivamente linhas 3 e 4), correspondem as variáveis do simulador que controlam a luz do ambiente. No Código 4 somente a luz 1 está ativa.

Os objetos do tipo `Wall` pertencem ao simulador, na criação deles é necessário informar: (i) a posição, que é dada pelas coordenadas x, y e z sendo necessário a criação de um objeto `Vector3d`, (ii) o comprimento, (iii) a altura do objeto, (iv) e o ambiente que está criando o objeto. Após a criação do elemento `Wall` (exemplo, linha 6), a função `rotate90(1)` (linha 7) é invocada e recebe como parâmetro um inteiro que indica quantas vezes a rotação de 90º graus em torno do eixo Y deve ser realizada. Com isso o objeto alcança a posição desejada e então através da função `add()`, que pertencem ao simulador, o objeto é adicionado ao ambiente.

Na linha 20 do Código 4, um objeto do tipo `Box` é criado, sendo que as informações que são passadas como parâmetro são: (i) a posição do objeto através de um `Vector3d`, com as posições x, y e z; (ii) as dimensões para o objeto através de um `Vector3f`, que recebe as dimensões para os eixos x, y e z respectivamente.

O objeto `Arch` (linha 24), recebe somente a posição que esse objeto deve ser posicionado e o ambiente que é responsável por esse. Na linha 335, o objeto `Arch` criado e rotacionado em 90º graus em torno do eixo Y, três vezes para que alcance a posição desejada.

O último elemento que é adicionado a esse ambiente é o `Robot` (linha 28), que é uma extensão da classe `Agent`, que é a classe do simulador que controla o objeto `Robot` dentro do simulador. Para a criação do objeto `Robot` é necessário informar a posição através de um `Vector3d` que recebe as coordenadas x, y e z.

O Código 5 é responsável por definir quais elementos o `Robot` possuirá, nesse caso serão adicionados dois elementos: uma câmera (linha 4) e colar de sensores contendo um 8 sensores (linha 6), quantidade de sensores padrão do simulador.

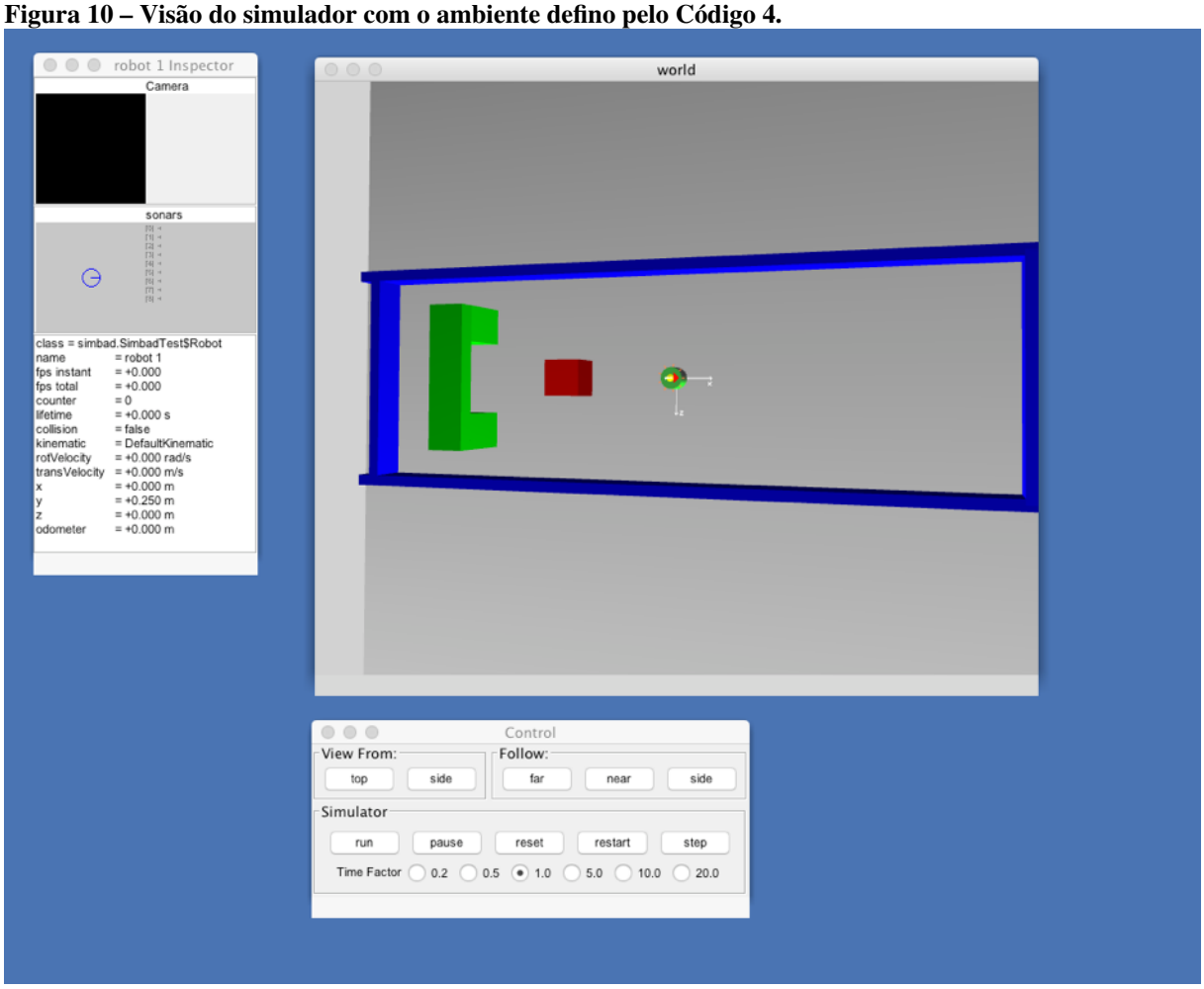
Código 5 – Criação do agente no simulador.

```
1  public Robot(Vector3d position, String name) {
2      super(position, name);
3      // Add camera
4      camera = RobotFactory.addCameraSensor(this);
5      // Add sonars
6      sonars = RobotFactory.addSonarBeltSensor(this);
7  }
```

Fonte – Autoria Própria.

A interação do agente acontece pela função `performBehavior()`, que pertence a classe `Agent`. Essa função é chamada ciclicamente, aproximadamente 20 vezes por segundo, sendo possível definir propriedades como a velocidade do agente e a direção, além de permitir a análise dos sensores para possíveis comportamentos que sejam pertinentes ao agente, como por exemplo desviar de um obstáculo.

Figura 10 – Visão do simulador com o ambiente defino pelo Código 4.



Fonte: Autoria Própria.

A Figura 10, é a visualização do ambiente, após a execução do código definido no Código 4. A janela "world" corresponde a representação gráfica do ambiente, é através dela que é possível visualizar o ambiente com seus elementos estáticos (Wall, Box, Arch) e dinâmicos (Robot).

A janela "robot 1 inspector", mostra informações referentes ao Robot que foi adicionado ao ambiente. Informações como: tempo de vida, colisão, velocidade de translação, velocidade de rotação, a posição (x, y, z) e a distância percorrida. Além disso através da "Camera", é possível visualizar o que o Robot visualiza e pelo do "sonars" é possivel visualizar quais sonares estão detectando elementos.

A janela Control, permite o controle da aplicação, os seguintes itens podem ser ajustados:

- Posição de visualização (*View From*): *Top* (visão superior) ou *side* (visão lateral);
- Modo para acompanhar o *Robot* (*Follow*): *Far* (longe), *near* (próximo) ou *side* (lateral);
- Controle da simulação (*Simulator*): *Run* (iniciar), *pause* (pausar), *reset* (voltar ao inicio), *restart* (reiniciar) ou *stop* (parar)
- Fator tempo (*Time Factor*): quando definido como 1, cada segundo na simulação corresponde a um segundo fora dela. A alteração desse afeta o tempo dentro do simulador, quando alterado para 5, por exemplo: cada 5 segundos dentro do simulador correspondem a um segundo fora dele.

Apesar de possuir uma interface simples e ser construído como uma biblioteca de Java, o simulador possuir recursos como: velocidades de translação e rotação, câmeras e sensores, que servem ao propósito desse trabalho.

3.8 DESVIO DE OBSTÁCULOS

Dentre as habilidades que os agentes podem possuir, esse trabalho estará focado na navegação, mais especificamente na ação de desviar de um obstáculo. Considerando o agente definido no Quadro 2, este trabalho se preocupa em desenvolver ações para que o agente consiga reagir a obstáculos encontrados no ambiente.

Quadro 2 – Percepções, ações objetivos e ambiente de um veículo autônomo.

Agentes	Percepção	Ações	Objetivo	Ambiente
Agente veículos autônomo	Posição e distância	Direita ou esquerda	Ir de um ponto A até B	Rua de mão única

Fonte: Autoria Própria.

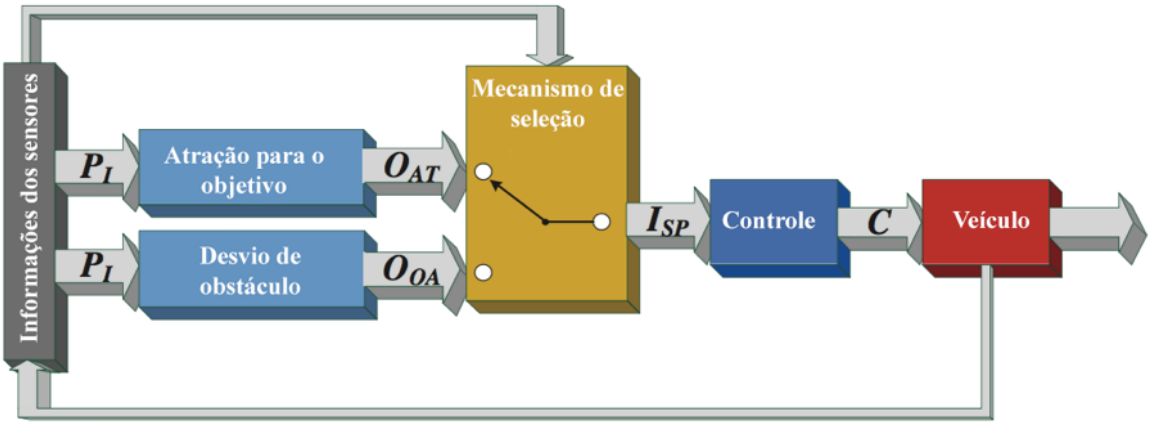
O agente definido no Quadro 2, corresponde a um veículo autônomo. Para o seu funcionamento é necessário que o agente seja capaz de perceber o ambiente através de câmeras, sensores e/ou gps. Além disso ele deverá possuir um objetivo, que é conseguir se locomover de um ponto A a um ponto B através das ruas. Por fim, para se locomover, será preciso que ele seja capaz de raciocinar sobre o ambiente, percepções e objetivos, e criar uma estratégia para que satisfaça o objetivo.

Durante a locomoção, o agente deverá ser capaz de desviar de obstáculos para que a locomoção seja um sucesso. Sendo assim habilidade importante para um agente. Dentro de ambientes urbanos desviar de obstáculos é uma tarefa complexa pelos diferentes tipos de obstáculos que podem ser encontrados. Eles podem ser estáticos ou dinâmicos com diferentes formas e tamanhos. Nas seções a seguir serão abordadas duas técnicas de desvio de obstáculos a *Elliptic Limit Cyles* (ELC) e a *Reciprocal Velocity Obstacles* (RVO).

3.8.1 *Elliptic Limit-cycles*

O trabalho desenvolvido por Dafflon *et al.* (2015) apresenta a técnica de trajetórias elípticas de Adouane, Benzerrouk e Martinet (2011), com modificações que a tornam mais eficiente e permitem o desvio de obstáculos com diferentes formas e tamanhos.

Figura 11 – Arquitetura embarcada no veículo terreste não-tripulado.



Fonte: Autoria Própria.

A arquitetura na Figura 11, é utilizada por Dafflon *et al.* (2015), os sensores captam as informações e as repassam aos módulos de atração para o objetivo e de desvio de obstáculo, além de fornecer essas informações ao mecanismo de seleção que possui uma alavanca que define qual módulo deverá ser usado (atração para o objetivo ou desvio de obstáculo). O Mecanismo de seleção escolhe o módulo de desvio de obstáculos assim que um obstáculo se encontra dentro da rota desejada. Após a escolha do módulo, as informações geradas pelo modulo selecionado serão passadas ao modulo de controle que definirá as saídas para que o veículo saiba as manobras que deverão ser realizadas.

Os agentes são peças importantes nesse modelo, pois são eles que fazem a conexão entre os veículos (mundo real) e o "mundo dos agentes". Através das combinações das informações adquiridas a partir dos controladores de aproximação do objetivo, e de desvio de obstáculos, tem-se um local de atração para o agente e locais de repulsão para obstáculos.

O agente possui interações baseadas na relação entre agentes, obstáculos e objetivos, sendo interações de atração ou repulsão (DAFFLON *et al.*, 2015):

- Entre agente e objeto: é criado um campo de força que atrai o agente ao objetivo, aonde a força é definida pela fórmula a seguir, sendo $A\overrightarrow{_i}T$, a posição do veículo em relação ao objetivo.

- Entre agente e agente: é criado um campo de repulsão para cada agente, para assegurar uma exploração homogênea do ambiente, e para que esses não colidam. De forma que se

Equação 1 – Equação que define a força de atração ao objetivo.

$$\overrightarrow{F} = \beta_g m A\overrightarrow{_i}T$$

A_i e A_j, dois agentes, nas posições P_i e P_j, respectivamente a força de repulsão é definida por:

Equação 2 – Equação que define a força de repulsão entre dois agentes.

$$F\overrightarrow{r}_{ij} = \alpha m_i m_j \frac{P\overrightarrow{_i}.P\overrightarrow{_j}}{\|P\overrightarrow{_i}.P\overrightarrow{_j}\|^3}$$

- Entre agente e obstáculos: os obstáculos identificados, assim como os agentes, possuem um campo de repulsão que garantem que o desvio desses aconteça, através da Equação 2, generalizada para 2 dimensões:

Equação 3 – Equação que define a força de repulsão entre agente e obstáculo.

$$\begin{cases} Fo_i^{\mathrm{X}} = \sum_o (\Delta_o . m . m_o \frac{(x_i - x_o)}{((y_i - y_o)^2 + (x_i - x_o)^2)^{3/2}} \\ Fo_i^{\mathrm{Y}} = \sum_o (\Delta_o . m . m_o \frac{(x_i - x_o)}{((y_i - y_o)^2 + (y_i - y_o)^2)^{3/2}} \end{cases}$$

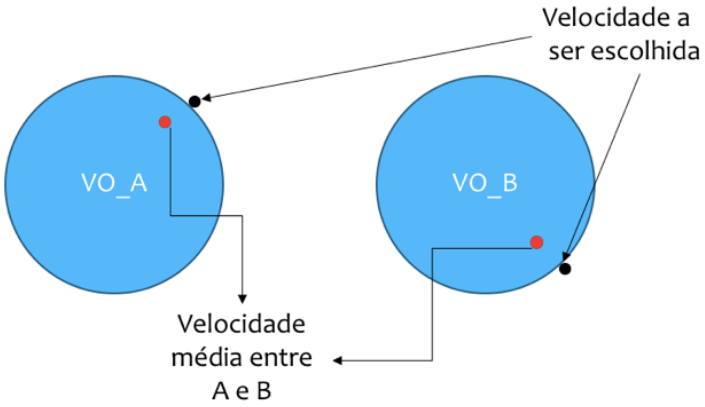
O desvio de obstáculos nesse modelo ocorre através das interações acima, a partir delas o agente procura alcançar seu objetivo com sucesso, desviando dos obstáculos e se locomovendo de uma origem até um destino.

3.8.2 *Reciprocal Velocity Obstacles*

O método *Reciprocal Velocity Obstacles* RVO é uma expansão do *Velocity Obstacles* (VO). Segundo Fiorini e Shiller (1998), VO é o conjunto de velocidades aonde dois agentes colidiriam em algum momento da rota, sendo assim qualquer velocidade que esteja fora desse conjunto são, velocidades seguras, e que permitem a navegação segura, ou seja, sem colisões.

Um problema do VO é a oscilação das velocidades. Suponha o cenário aonde dois agentes colidiriam, cada agente escolhe aleatoriamente uma velocidade que esteja fora do conjunto VO, porém a escolha da velocidade altera o conjunto VO de forma que a nova velocidade agora faz parte do conjunto VO, fazendo necessário com que os agentes tenham que realizar outra escolha. Esse ciclo pode se repetir até que ambos agentes escolham velocidades que fiquem fora do conjunto VO após a alteração da velocidade. Na Figura 12, temos a expansão da técnica VO.

Figura 12 – Seleção da nova velocidade através da técnica RVO.



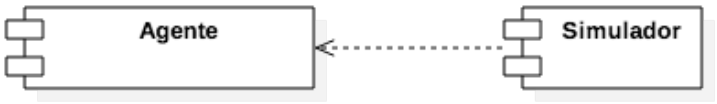
Fonte: Autoria Própria.

4 DESENVOLVIMENTO

Neste capítulo estão as atividades que desenvolvidas, a implementação do agente autônomo, do ambiente no simulador e a integração entres esses. O objetivo do agente implementado é desviar de obstáculos. O agente é implementado em alto nível, ou seja, ele é responsável somente pela tomada de decisões. O simulador por sua vez possui uma representação do agente, que será referenciada como veículo, e esse possui as manobras que podem ser executadas, além disso o simulador é responsável por todo o ambiente que o agente navega. A comunicação entre esses componentes é feita via rede utilizando o protocolo TCP/IP através de chamadas assíncronas.

O projeto é constituído por dois componentes, um que corresponde ao agente e outro ao simulador (Figura 13).

Figura 13 – Diagrama de Componentes, relação entre o agente e o simulador.



Fonte: Autoria Própria.

Na Figura 13, o simulador é dependente do agente e é responsável por criar e gerenciar o ambiente, e por controlar o veiculo dentro do ambiente através de acordo com as manobras definidas pelo agente. Dessa forma o agente, o agente fica encarregado de receber as percepções do ambiente e definir as manobras que devem ser realizadas.

Nas subseções a seguir serão vistos: o simulador, a comunicação para com o agente, o agente e os resultados obtidos através dos testes.

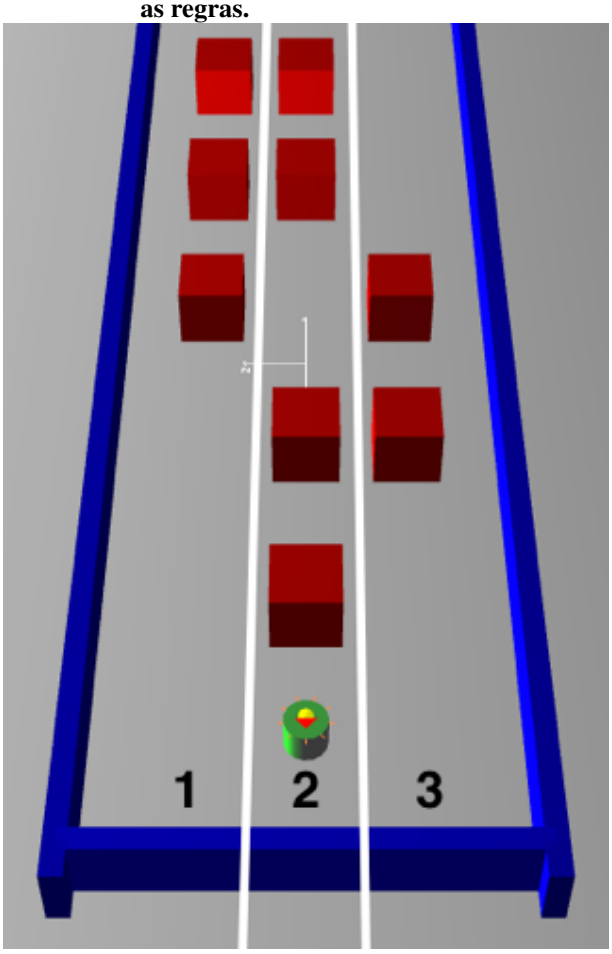
4.1 IMPLEMENTAÇÃO DO SIMULADOR

O simulador é responsável por construir o ambiente e controlar o veículo. O simulador escolhido para esse trabalho foi o Simbad. Os principais motivos pelo qual esse foi selecionado é por ele ser uma biblioteca de Java, por corresponder a um simulador menos complexo, em relação a simuladores mais realistas como o V-REP, e por ser de fácil aprendizagem, atendendo os requisitos para esse trabalho, que é a locomoção de um veículo dentro de um ambiente que possua obstáculos.

Para que fosse possível definir um grupo de manobras a qual o veículo pode executar, as seguintes regras foram definidas para delimitar como o ambiente deve ser construído:

- Todos os obstáculos devem ser cubos de 1 metro;
- O ambiente é organizado em três faixas (1, 2 ,3);
- Os obstáculos se encontram somente em uma das faixas, ou seja, não é possível que um mesmo obstáculo ocupe espaço em duas faixas.

Figura 14 – Exemplo de um ambiente de acordo com as regras.



Fonte: Autoria Própria.

Na Figura 14, um exemplo de um ambiente que segue as regras que foram definidas. Essas regras são necessárias para que fosse possível definir um possível conjuntos de manobras, que o veículo consegue realizar no ambiente.

Além das regras para o ambiente, o veículo também possui um conjunto de normas de como ele pode navegar por esse ambiente:

- O veículo sempre se locomove pela segunda faixa (ou faixa central) quando não estiver realizando uma manobra;
- O veículo se locomove em velocidade continua de 1.5 m/s;
- Quando uma alteração na direção é necessária, o veículo sempre muda a direção em 90 graus, e sempre estará apontando a um ponto cardeal (Norte, Sul, Oeste ou Leste);

- O veículo possui quatro manobras: i) desvio pela direita, ii) desvio pela esquerda, iii) parar o agente e iv) desfazer manobra (sendo que uma manobra pode ser desfeita parcialmente ou integralmente).

Cada manobra implementada é executada quando o agente define que essa é necessária para evitar algum obstáculo. No Código 6 tem-se o controle principal do veículo, onde as informações são captadas do ambiente e também onde os passos das manobras são realizados.

Código 6 – Controle do simulador, código Java.

```
1  public void performBehavior() {
2      setTranslationalVelocity(1.5);
3      setNorthDirection();
4
5      if (sonars.oneHasHit() &&
6          sonars.getMeasurement(0) < 5.0 &&
7          maneuver == ManeuverType.STEADY &&
8          decision == DecisionProcess.DONE) {
9
10         obstacleDistance = sonars.getMeasurement(0);
11         obstacleDetected();
12     }
13
14     if (sonars.oneHasHit() &&
15         sonars.getMeasurement(0) < 5.0 &&
16         (maneuver == ManeuverType.LEFT ||
17          maneuver == ManeuverType.RIGHT)) {
18         if (step == StepNumber.FIFTH_STEP) {
19             Double turnPointValue = turnPosition.getX() -
20                                     obstacleDistance -
21                                     2.0;
22
23             obstacleDetectedInManeuver(sonars.getMeasurement(0),
24                                        turnPointValue);
25         } else {
26             obstacleDetectedInManeuver(sonars.getMeasurement(0));
27         }
28
29     }
30
31     if (sonars.oneHasHit() && first > 5) {
32         switch (maneuver) {
33             case LEFT:
34                 if (DecisionProcess.DONE == decision) {
35                     performLeftManeuver();
36                 }
37                 break;
38
```

```
39                case RIGHT:
40                    if (DecisionProcess.DONE == decision) {
41                        performRightManeuver();
42                    }
43                    break;
44
45                case STEADY:
46                    stopRotation();
47                    break;
48
49                case WALKBACK:
50                    if (DecisionProcess.DONE == decision) {
51                        performWalkback();
52                    }
53                    break;
54
55                case STOP:
56                    stopRotation();
57                    stopTranslation();
58                    break;
59            }
60        }
61    }
```

Fonte – Autoria Própria.

O Código 6, é o controle principal do veiculo, sendo invocado 20 vezes por segundo enquanto o simulador está em funcionamento (SIMBAD, 2017). No início do algoritmo tem-se a função `setTranslationalVelocity` (linha 2), é ela que define a velocidade de locomoção do veículo, em seguida a função `setNorthDirection` (linha 3) é invocada para garantir que no inicio da simulação o agente esteja apontado para o Norte.

Cada `if` definido no Código 6, possui um objetivo. O primeiro `if` (linhas 5-8), define as percepções que serão passadas ao agente quando o veículo não estiver realizando manobras e o segundo (linhas 14-17) quando o veículo estiver realizando uma manobra. As percepções correspondem a detecção de algum obstáculo a frente do veículo.

Uma manobra pode ser divida em passos, cada passo corresponde a uma ação que o veículo deve realizar para desviar do obstáculo, caso o veículo esteja no quinto passo da manobra `if` linha 18, então o veículo precisa informar ao agente qual a posição que ele pretende realizar uma curva, pois durante esse passo o ponto é variável, diferente dos outros passos aonde os pontos são fixos. Tal posição é dada pela posição em que o agente realizou a curva, acrescido da distância que o veículo estava quando detectou o obstáculo, mais um metro para considerar o tamanho do obstáculo e um para adicionar uma margem de segurança em relação ao obstáculo.

A percepção do veículo é realizada através de um sonar que está acoplado a ele, através do sonar o veículo possui a capacidade de verificar oitos pontos distribuídos ao seu redor. Para

esse trabalho somente um deles será utilizado, o sonar que verifica o ponto diretamente a frente do veículo (linhas 5-6 e 14-15). Para que as percepções sejam consideradas a distância do veículo para o obstáculo deve ser menor que 5 metros, para garantir que uma manobra não seja iniciada a mais de 5 metros do obstáculo.

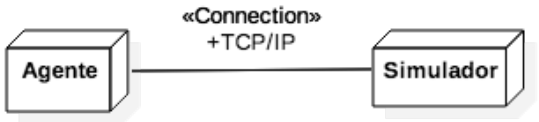
O último `if` (linha 31) é responsável por controlar a manobra que será executada. Ele verifica a manobra definida (linha 32) e invoca o método responsável para realizar os passos da manobra. Devido as percepções do ambiente a manobra que está sendo executada pode ser alterada para prevenir que o veículo colida com algum obstáculo.

As manobras que serão realizadas são definidas dentro do simulador, porém é o agente que realiza a decisão de qual será executada. Sendo assim informações precisando ser trocados entre o simulador e o agente, as informações de percepção adquiridas pelo Código 6, devem ser enviadas ao agente, utilizando um meio de comunicação definido na seção a seguir.

4.2 IMPLEMENTAÇÃO DA COMUNICAÇÃO

A comunicação permite que o agente tenha conhecimento que uma decisão precisa ser realizada. Para esse trabalho a comunicação será feita pela rede, através do protocolo TCP (Figura 15).

Figura 15 – Diagrama de implantação, conexão entre os pacotes.



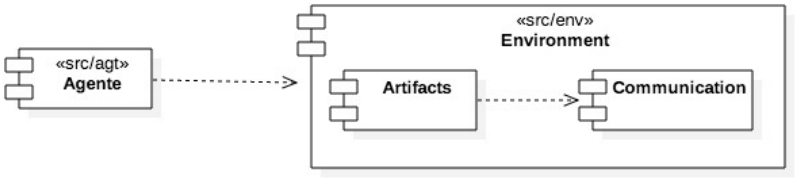
Fonte: Autoria Própria.

O protocolo TCP foi selecionado para a comunicação devido à sua propriedade de garantir a entrega dos pacotes. Outra opção seria a utilização do protocolo UDP, porém esse não garante a entrega dos pacotes, podendo ocasionar em uma falha no software para cenários aonde a mensagem não alcance o destino.

Cada componente presente na Figura 13, possui um subcomponente responsável pela comunicação, esse componente de comunicação é responsável por enviar, receber e processar mensagens (Figuras 16 e 17)

Os componentes presentes nas Figuras 16 e 17, dependem do componente de comunicação, as setas indicam a dependência dos componentes no diagrama. O agente é dependente do ambiente, para que ele consiga realizar decisões e o Simbad é dependente da comunicação para receber informações de manobras que devem ser realizadas. É possível perceber por meio da Figura 18, que os componentes funcionam de forma independente e não influenciam no fun-

Figura 16 – Diagrama de componentes, componentes do agente.



Fonte: Autoria Própria.

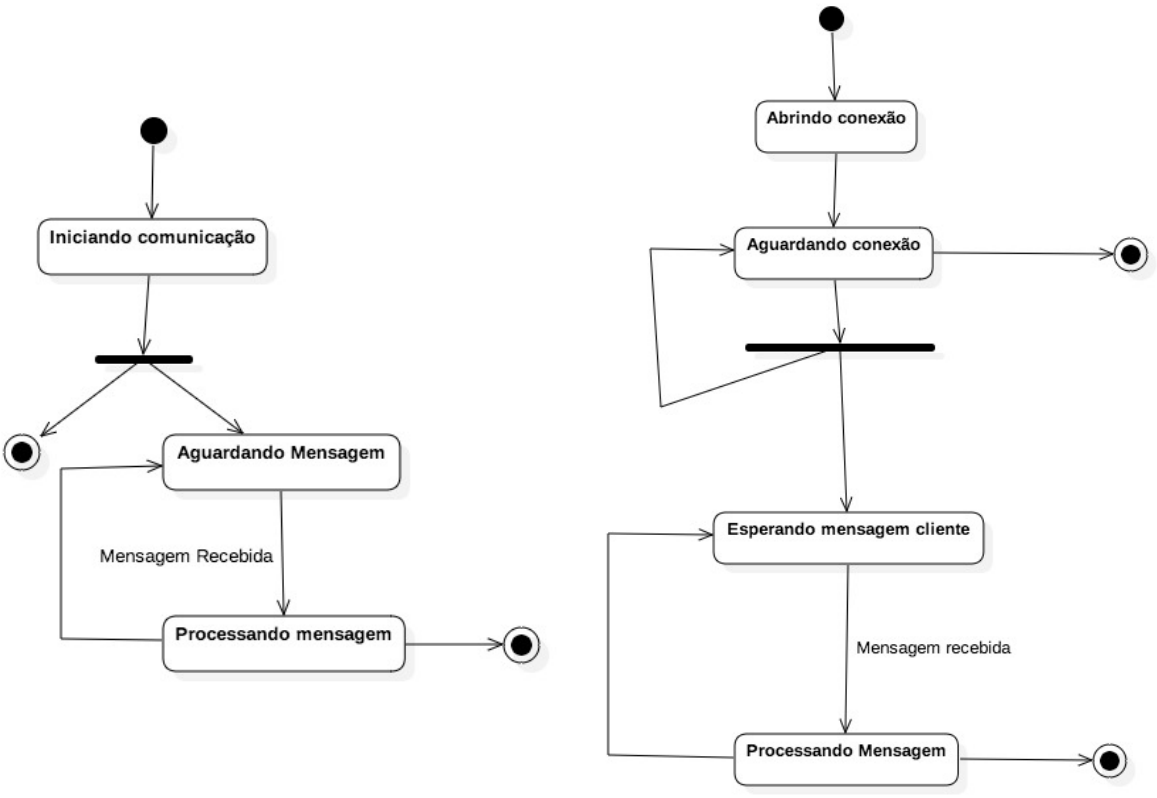
Figura 17 – Diagrama de componentes, componentes do simulador.

Fonte: Autoria Própria.

cionamento da aplicação. As mensagens são enviadas e recebidas de forma assíncrona.

As classes de comunicação, elas funcionam em um loop e sempre que um mensagem é recebida essa é repassada ao componente de destino. É importante notar que a aplicação funciona localmente erros de conexão não aconteceram e serão omitidos no diagrama a seguir não sendo o foco do trabalho. A Figura 18, possui a representação das classes de comunicação.

Figura 18 – Diagrama de estados, classes de comunicação do agente (direita) e do simulador (esquerda).



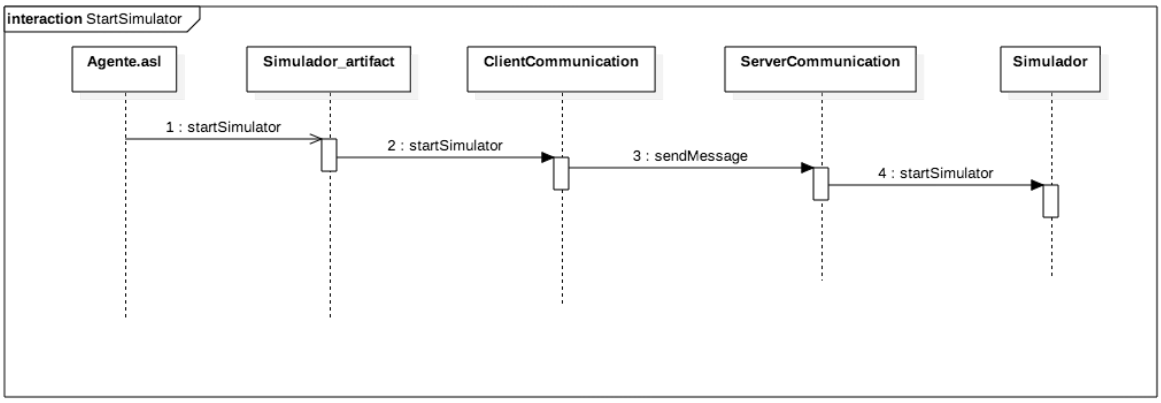
Fonte: Autoria Própria.

O componente de comunicação do Simulador foi definido como sendo o servidor e o componente de comunicação do Agente como o cliente. Para que a comunicação através do meio TCP fosse possível, o protocolo definido no 3 foi estabelecido entre o cliente e o servidor. Todas as mensagens começam por um número indicando qual a mensagem e as informações são divididas através do delimitador "#".

As mensagens do protocolo definido no Quadro 3, estabelecem a comunicação entre o agente e o simulador. Elas definem como as informações devem ser trocadas, para que o agente tenha conhecimento do ambiente e o simulador das decisões. As mensagens enviadas do agente ao simulador sempre serão iniciadas por um número par, sendo que o agente possui mais mensagens pois essas estão relacionadas ao número de manobras que o agente pode realizar. Por sua vez as mensagens enviadas do simulador ao agente sempre iniciaram por um numero ímpar, seguida das informações do ambiente. Note que as mensagens não estão organizadas como requisição e resposta, ou seja, a mensagem 5 não necessariamente possuirá como resposta a mensagem 6.

A única mensagem que não possui uma resposta é a de inicialização do simulador, sendo enviada do agente para o simulador. É possível visualizar o fluxo dessa através da Figura 19.

Figura 19 – Diagrama de sequência, inicialização do simulador.



Fonte: Autoria Própria.

Na Figura 19, a requisição de inicialização do simulador parte do agente, que informa o artefato que o simulador deve ser inicializado, então o artefato faz uma requisição ao servidor através da classe de comunicação. Ao receber a requisição o servidor repassa a mensagem para o simulador informando que esse deve ser inicializado.

As mensagens de requisição para quando o simulador detecta um obstáculo possuirão uma resposta do agente quando uma ação for necessária. Para esse caso temos o seguinte diagrama na Figura 20.

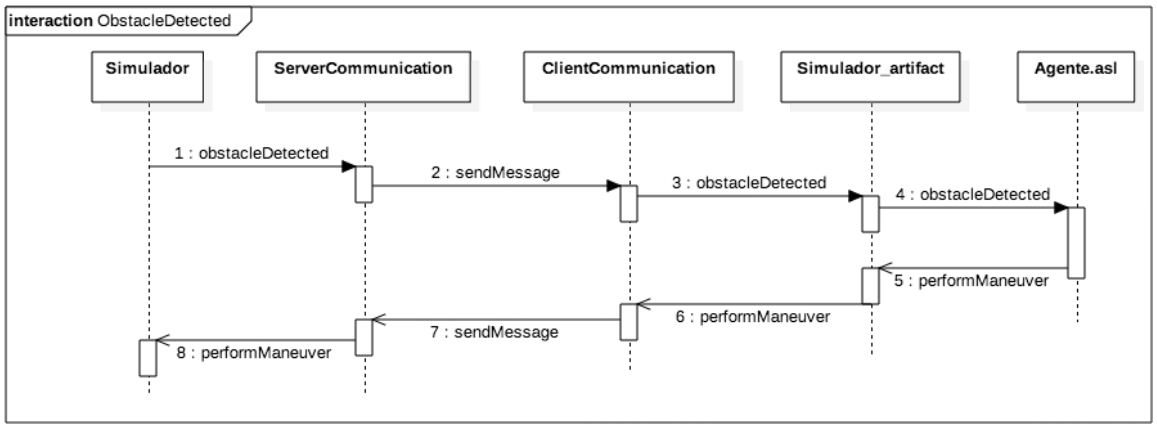
As mensagens para quando o veículo percebe um obstáculo partem do simulador, que informa o servidor que uma mensagem precisa ser enviada ao agente. Ao receber essa a classe de comunicação do agente, processa ela atualiza as crenças de acordo com as informações recebidas e declara a intenção de desviar do obstáculo. Após a decisão, o plano invocado, passa a

Quadro 3 – Protocolo para comunicação entre o agente e o simulador.

Mensagem	Agente para o simulador	Simulador para o agente
0	Requisição para iniciar o simulador	
1#x_pox#y_pos#z_pos		Informa a posição do veiculo na simulação
2	Informa que uma manobra a esquerda é necessária	
3#x_pos#y_pos#z_pos		Informa a posição do obstáculo
4	Informa que uma manobra a direita é necessária.	
5#obstacle_left #obstacle_center #obstacle_right		Informa em quais posições existem obstáculos (três posições possíveis: esquerda, centro e direita)
6	Informa que o veículo deve parar de se movimentar	
7#maneuver #direction #distance_to_hit #actual_x #actual_y #actual_z #last_x #last_y #last_z		Informa a manobra atual, direção do veículo, posição atual (coordenadas x, y, z) e a ultima posição que o agente fez uma curva (x, y, z)
8	Informa que o veiculo deve desfazer os passos atuais e tentar realizar outra manobra (caso 1 - esteja realizando a manobra pela direita, deve tentar proceder pela esquerda; caso 2 - quando está realizando a manobra pela esquerda, tentar proceder pela direta).	
9#maneuver #direction #distance_to_hit #turn_point #actual_x #actual_y #actual_z #last_x #last_y #last_z		Informa a manobra atual, direção do veiculo, próxima posição que o veículo pretende fazer uma curva, posição atual (coordenadas x, y, z) e a ultima posição que o veículo fez uma curva (x, y, z)
10#Walkback#Right ou 10#Walkback#Left	Informa que veículo deve desfazer os passos da manobra atual e tentar realizar as manobras informadas.	
12#Walkback#Keep_Right ou 12#Walkback#Keep_Left	Informa que veículo deve desfazer o ultimo passo realizado e continuar com a manobra.	

Fonte: Autoria Própria.

Figura 20 – Diagrama de sequência, obstáculo detectado.



Fonte: Autoria Própria.

mensagem a classe de comunicação que envia a requisição ao servidor que informa o simulador qual a ação foi selecionada.

4.3 IMPLEMENTAÇÃO DO AGENTE

Com a comunicação estabelecida entre o agente e o simulador, o agente agora possui um meio de receber as percepções realizadas pelo simulador. A implementação do agente é realizada através do framework JaCaMo, com a criação de um agente e um artefato para comunicação.

O plano inicial do agente é configurar a conexão entre o agente e o simulador através da criação do artefato que faz o gerenciamento do simulador, após o estabelecimento da conexão o agente envia uma requisição `startSimulator` (Código 7) para que o simulador seja iniciado.

Código 7 – Plano inicial, iniciar o simulador.

```
1  /* Initial goals */
2  !setupSimulator.
3
4  /* Plans */
5  +!setupSimulator <- .print("Setup Simulator");
6                      makeArtifact("a_Simulator",
7                                   "autonomous_vehicles.Simulator_artifact",
8                                   ["Starting"],
9                                   ArtId);
10                     focus(ArtId);
11                     startSimulator[artifact_id(ArtId)].
```

Fonte – Autoria Própria.

O agente será responsável pela tomada de decisão, o artefato será o responsável por organizar as percepções recebidas do simulador e as respostas que serão enviadas. Para permitir

que as crenças do agente, fossem atualizáveis pelo artefato, essas foram definidas como propriedades observáveis, que são crenças manipuláveis pelo artefato (Código 8), sendo necessário que o agente escolha observar as propriedades do artefato utilizando a função `focus` (Código 7, linha 10).

Código 8 – Propriedades observáveis, crenças.

```
1  // Agent beliefs
2  defineObsProperty("left", false);
3  defineObsProperty("center", false);
4  defineObsProperty("right", false);
5  defineObsProperty("actualPosition", 0.0, 0.0, 0.0);
6  defineObsProperty("lastTurnPosition", 0.0, 0.0, 0.0);
7  defineObsProperty("maneuver", "STEADY");
8  defineObsProperty("direction", "NORTH");
9  defineObsProperty("distanceToHit", 0.0);
10 defineObsProperty("turnPoint", 999);
```

Fonte – Autoria Própria.

Cada linha pertencente ao Código 8 corresponde a uma crença. As três primeiras crenças indicam quais são os lugares, imediatamente a frente do veiculo que possuem obstáculos, como o ambiente está divido em três faixas, cada faixa corresponde a uma crença linha 2: left, linha 3: center, linha 4: right. Inicialmente o agente não possui conhecimento de nenhum obstáculo, por esse motivo o valor dessas crenças é false.

As linhas 5 à 10, corresponde a informações do ambiente. A linha 5 corresponde a posição atual do agente através de coordenadas x, y e z, linha 6 a última posição que o agente realizou uma curva também dada por coordenadas x, y e z. Linha 7 indica a manobra que está sendo realizada, sendo valor "Steady" para nenhuma manobra, a linha 8 indica a direção que o agente está se locomovendo. A linha 9 a distância para o agente colidir, com zero indicando que nenhum obstáculo foi detectado e a linha 10 a posição da próxima curva, esse ponto é referente a somente uma coordenada x, y ou z, com 999 indicando que nenhum ponto foi informado.

Com essas três crenças, o primeiro conjunto de planos que o agente pode realizar é definido, esses planos somente são invocados quando o veículo não está realizando nenhuma manobra (Código 9).

Código 9 – Método para atualização das crenças e declaração do objetivo.

```
1 +collisionDetected(ArtId)
2 <- .print("Collision");
3     ?left(LEFTVALUE);
4     ?center(CENTERVALUE);
5     ?right(RIGHTVALUE);
6     !collision(ArtId, LEFTVALUE, CENTERVALUE, RIGHTVALUE).
```

Fonte – Autoria Própria.

O método `collisionDetected` (Linha 1) , é invocado quando o agente recebe percep-

ções. Note que cada um dos planos a seguir possui um .print, que é utilizado para identificar os planos invocados durante a execução do agente. As linhas 3 à 5, são necessárias para armazenar o valor atual das crenças nas variáveis designadas.

Com as percepções atualizadas a intenção de desviar um obstáculo é declarada, linha 6. Baseado nas crenças, um plano será invocado. Considerando o cenário onde crenças LEFT, CENTER e RIGHT forem verdadeiras então o plano definido no Código **??** é executado.

Código 10 – Plano para o agente para de se mover.

```
1 +!collision(ArtId, LEFT, CENTER, RIGHT)   : LEFT & CENTER & RIGHT
2                           <- .print("stop", centerValue);
3                              performStop[artifact_id(ArtId)].
```

Fonte – Autoria Própria.

O plano do Código 10, é somente executado quando todos os caminhos possíveis em que o veículo pode transitar estão bloqueados, necessitando que esse pare de se locomover.

Os planos no Código 11, correspondem a planos para quando LEFT for falso ou RIGHT for falso. Sendo assim existe uma manobra disponível, que pode ser realizada.

Código 11 – Planos para quando quando não existem obstáculos nas laterais.

```
1 +!collision(ArtId, LEFT, CENTER, RIGHT)   : LEFT & CENTER & not RIGHT
2                           <- .print("right");
3                              performRightManeuver[artifact_id(ArtId)].
4
5 +!collision(ArtId, LEFT, CENTER, RIGHT)   : not LEFT & CENTER & RIGHT
6                          <- .print("left");
7                              performLeftManeuver[artifact_id(ArtId)].
```

Fonte – Autoria Própria.

No Código 11, o primeiro plano é executado quando não existe obstáculo em RIGHT, nesse caso o simulador receberá a notificação pela ação, que pertence ao artefato *Simulator_artifact*, significando que a manobra pela direita deve ser executada para evitar os obstáculos LEFT e CENTER. O segundo plano é similar ao anterior, porém para quando LEFT está livre de obstáculos, o que possibilita que a manobra pela esquerda seja efetuada, para esse caso a ação será executada para informar o Simulador.

O último plano, Algoritmo 21, é para quando o agente não possui obstáculos em LEFT e RIGHT.

Código 12 – Plano para quando não existem obstáculos em LEFT e RIGHT.

```
1 +!collision(ArtId, LEFT, CENTER, RIGHT) : not LEFT & CENTER & not RIGHT
2                          <- .print("center"); focus(ArtId);
3                              performLeftManeuver[artifact_id(ArtId)].
```

Fonte – Autoria Própria.

O Código 12, é ativado quando existe um obstáculo, e esse está em CENTER. Nesse caso ambas manobras podem ser realizadas, porém como padrão a manobra pela esquerda sempre será executada nesse caso aonde não existem obstáculos em LEFT e RIGHT.

Os planos vistos até o momento consideram somente grupos de obstáculos diretamente a frente do veículo, o que pode acontecer é, enquanto realizando uma manobra o agente pode se deparar com outros obstáculos. Para isso foram definidos, planos de recuperação, que são invocados pelo Código 13, quando o veículo gerar percepções durante uma manobra, identificando um obstáculo durante a manobra que está sendo realizada.

Código 13 – Método para atualização das crenças quando o agente está realizando uma manobra.

```
1  +collisionWhileManeuver(ArtId)
2  <- ?maneuver(MANEUVER);
3       ?direction(DIRECTION);
4       ?distanceToHit(DISTANCETOHIT);
5       ?actualPosition(ACTUALX, ACTUALY, ACTUALZ);
6       ?lastTurnPosition(LASTX, LASTY, LASTZ);
7       ?turnPoint(TURNPOINT);
8       !collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT,
      ACTUALX, ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ).
```

Fonte – Autoria Própria.

Após a atualização das crenças, as linhas 3 à 7 do Código 13, são executadas para armazenar o valor das crenças nas variáveis entre parênteses. A intenção para desviar de um obstáculo é declarada na linha 8, sendo que o respectivo plano será invocado dependendo do contexto atual das crenças. Para esses planos, o contexto leva em consideração a manobra que está sendo realizada, a direção do agente (norte, sul, leste ou oeste) e a posição que o agente pretende realizar uma curva.

Considerando a direção norte, os seguintes planos podem ser invocados Código 14.

Código 14 – Planos para quando o agente está em direção ao norte.

```
1  +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
2  : DIRECTION == "NORTH" &
3    (TURNPOINT > (ACTUALX - DISTANCETOHIT + 0.5))
4      <- print("Keep going 0").
5
6  +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
7  : DIRECTION == "NORTH" &
8    (TURNPOINT < (ACTUALX - DISTANCETOHIT))
9      <- print("Driveback");
10        changeManeuver("10")[artifact_id(ArtId)].
```

Fonte – Autoria Própria.

Os planos definidos em Código 14, baseiam-se na localização do TURNPOINT, que

é o ponto onde o veículo pretende fazer uma curva, através da seguinte comparação: `(ACTUALX - DISTANCETOTHIT + 0,5) < TURNPOINT < (ACTUALX - DISTANCETOHIT + 0.5)`, o agente verifica se o obstáculo se encontra antes ou depois da posição que o veículo pretende realizar uma curva. Nesses planos o cálculo da posição do obstáculo é realizado através de uma subtração, pois quando o veículo está se locomovendo em direção ao norte os valores de X decaem. Com isso se é feito a subtração do valor atual de X com a distância para o obstáculo, sendo o valor gerado a posição do obstáculo. Finalmente o valor de 0,5 é adicionado para considerar o corpo do veículo, pois como o `TURNPOINT` é aonde o veiculo vai realizar a curva, então metade de seu corpo ficará à direita desse ponto e a outra metade a esquerda, necessitando espaço extra entre o ponto de curva e o obstáculo. Note que todos os planos definidos nos algoritmos a seguir possuirão valor (0,5) para assegurar que o veículo não colidirá no momento de realizar a curva.

O primeiro plano do Código 14 (linha 1-4), é executado quando a posição marcada para a realização da curva estiver mais próximo do que a posição do obstáculo, permitindo que ele continue com o caminho planejado. O segundo plano (linha 6-10), é o caso contrário, quando a posição que o veículo pretende virar está após a posição do obstáculo, nesse caso o veículo deve retornar pelo caminho percorrido até o ponto onde a manobra se iniciou e então realizar a manobra oposta, se possível.

Por sua vez considerando a direção oeste, quatro planos podem ser invocados Código 15.

Código 15 – Planos para quando o agente está em direção ao oeste.

```
1  +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
2  : DIRECTION == "WEST" &
3    MANEUVER == "LEFT" &
4    (1.5 >= (ACTUALZ + DISTANCETOHIT - 0.5 ))
5      <- print("Driveback + Right");
6          changeManeuver("12#WALKBACK#RIGHT")[artifact_id(ArtId)].
7
8  +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
9  : DIRECTION == "WEST" &
10   MANEUVER == "LEFT" &
11   (1.5 < (ACTUALZ + DISTANCETOHIT - 0.5))
12     <- print("Keeg Going 1").
13
14 +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
15 : DIRECTION == "WEST" &
16   MANEUVER == "RIGHT" &
17   (0.0 >= (ACTUALZ + DISTANCETOHIT - 0.5))
18     <- print("Driveback + Keep going right.");
19         changeManeuver("14#WALKBACK#KEEP_RIGHT")[artifact_id(ArtId)].
20
```

```
21 +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
22 : DIRECTION == "WEST" &
23   MANEUVER == "RIGHT" &
24   (0.0 < (ACTUALZ + DISTANCETOHIT - 0.5))
25     <- print("Keep going 3").
```

Fonte – Autoria Própria.

Os planos no Código 15, são invocados quando a direção em que o veículo está voltado é a oeste, nesse caso a manobra pela esquerda ou pela direita pode ser realizada. O raciocínio para os cálculos são os mesmos utilizados em Código 14. Quando o veículo estiver realizando, ou manobra pela esquerda ou a manobra pela direita, o agente possuirá dois planos para cada manobra, um que permite que o veículo prossiga com a manobra e outro para quando um desvio é necessário.

Assim como no Código 14, os planos são ativados de acordo com a posição do veículo, do obstáculos e posição aproximada de curva. A diferença nos planos do Código 15, é a posição aproximada de curva do veículo que é definida de acordo com um ponto fixo dependente da manobra que está sendo executada. Esses pontos são definidos pelas faixas do simulador, sendo o valor 1.5 para faixa 1, 0.0 para a faixa 2 e -1.5 para a faixa 3.

Nos dois primeiros planos (linhas 1-6 e 8-12 respectivamente do Código 15), o agente está tentando se mover para a faixa 1, através da manobra a esquerda. Caso ele encontre algum obstáculo que esteja mais próximo ao veículo do que a posição para realizar a curva, então o primeiro plano (linhas 1-6) é ativado e o veículo retornará pelo caminho até o ponto inicial e tentará realizar a manobra pela esquerda, caso o obstáculo esteja mais distante então o segundo plano (linhas 8-12) é ativado e o veículo continua com a manobra.

O terceiro (linhas 14-19) e o quarto (linhas 21-25) planos definidos no Código 15, são para a manobra a esquerda. Funcionando de forma similar aos planos anteriores, um para quando o obstáculo detectado está fora da trajetória do veículo, e outro para quando o obstáculo estiver dentro da trajetória. Nesse caso o veículo deve retornar pelo trajeto e tentar a manobra na direção oposta. A diferença em relação aos dois primeiros planos, é que o veículo está se locomovendo da faixa 1 para 2, por isso o ponto aproximado de curva muda de 1.5 para 0.0.

Com a direção leste, temos os seguintes plano definidos no Código 16.

Código 16 – Planos para quando o agente está em direção ao leste.

```
1 +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
      ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
2 : DIRECTION == "EAST" &
3   MANEUVER == "RIGHT" &
4   (-1.5 <= (ACTUALZ - DISTANCETOHIT + 0.5))
5     <- print("Driveback + Left");
6        changeManeuver("12#WALKBACK#LEFT")[artifact_id(ArtId)].
7
```

```
8  +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
9  : DIRECTION == "EAST" &
10   MANEUVER == "RIGHT" &
11   (-1.5 > (ACTUALZ - DISTANCETOHIT + 0.5))
12     <- print("Keep Going 2").
13
14 +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
15 : DIRECTION == "EAST" &
16   MANEUVER == "LEFT" &
17   (0.0 <= (ACTUALZ - DISTANCETOHIT + 0.5))
18     <- print("Driveback + Keep going left");
19     changeManeuver("14#WALKBACK#KEEP_LEFT")[artifact_id(ArtId)].
20
21 +!collision(ArtId, MANEUVER, DIRECTION, DISTANCETOHIT, TURNPOINT, ACTUALX,
       ACTUALY, ACTUALZ, LASTX, LASTY, LASTZ)
22 : DIRECTION == "EAST" &
23   MANEUVER == "LEFT" &
24   (0.0 > (ACTUALZ - DISTANCETOHIT + 0.5))
25     <- print("Keep going 4").
```

Fonte – Autoria Própria.

Os planos acima, Código 16, funcionam da mesma forma aos anteriores, baseados na posição em que o veículo pretende realizar uma curva e na posição do obstáculo enquanto o veículo realizava uma manobra, o agente procura distinguir os casos aonde é possível prosseguir com a manobra e os casos aonde é necessário, uma nova rota.

O primeiro (linhas 1-6) e o segundo (8-12) planos (Código 16), são executados quando a direção está definida como leste e manobra que está sendo executada é pela direita. Nesse caso quando o agente não possuir espaço suficiente para finalizar a manobra, significa que existe algum obstáculo entre o agente e a posição -1.5, sendo o primeiro plano será executado e o agente irá informar o simulador que o veículo deve retornar pelo caminho realizado e tentar executar uma nova manobra. Caso não exista nenhum obstáculo entre o veiculo e posição -1.5 (faixa 3), então o agente pode prosseguir com a manobra.

O terceiro (14-19) e o quarto (21-25) planos definidos no Código 16, realizam a verificação para quando existe ou não algum obstáculo entre o veiculo e a posição que ele deseja se locomover, nesse caso ele estará realizando uma movimentação da faixa 3 para a faixa 2.

Os planos dos códigos, formam um conjunto de conhecimentos que permite ao agente tomar uma decisão de como o veículo pode proceder para desviar de obstáculos, note que as decisões necessitam que o ambiente esteja de acordo com as regras definidas, mas além disso um fator decisivo para que o agente consiga realizar uma decisão é a percepção do veículo.

O conjunto de planos, para quando o veículo não está realizando uma manobra, é ativado pela percepção do sonar quando ele estiver na faixa 2 e existir um obstáculo a sua frente,

nesse caso o agente será notificado com o obstáculo a frente do veículo e se existem obstáculos a direita ou esquerda do obstáculo, a definição se existe alguém obstáculo a direita ou esquerda não é definida pelo sonar e sim por uma lista de obstáculos que o simulador possui e gerencia para identificar qual conjunto de obstáculos o veículo está percebendo. Por sua vez quando o veículo estiver realizando uma manobra as percepções geradas serão informações que podem ser captadas do sonar e são relacionadas somente obstáculo detectado a frente do veículo. Com essas percepções o agente então possuirá as informações para que os planos sejam executados.

5 RESULTADOS

Com a implementação de todos os componentes (simulador, agente e comunicação), é possível então realizar testes para verificar a capacidade de funcionamento. Neste trabalho três cenários foram planejados, o Quadro 4 propicia um visão geral dos resultados obtidos.

Quadro 4 – Cenários avaliados.

Cenários	Objetivo	Resultado Obtido
Cenário 1: Teste das manobras simples	Verificar a realização de manobras simples	Manobras simples conseguem ser executadas com sucesso
Cenário 2: Teste da manobras complexas	Verificar a capacidade do agente de realizar manobras complexas	Manobras complexas conseguem ser executadas com sucesso
Cenário 3: Teste do limite de velocidade	Verificar a velocidade máxima possível no cenário proposto	Velocidade máxima 4,8 m/s

Fonte: Autoria Própria.

Os cenários propostos têm o objetivo de avaliar a capacidade do agente, apesar de utilizar tempo como um parâmetro de medida esse não é utilizado para verificar a capacidade da rede e sim para indicar que uma manobra foi realizada com sucesso, que as manobras complexas demoram mais tempo que que a velocidade influência diretamente na capacidade do agente.

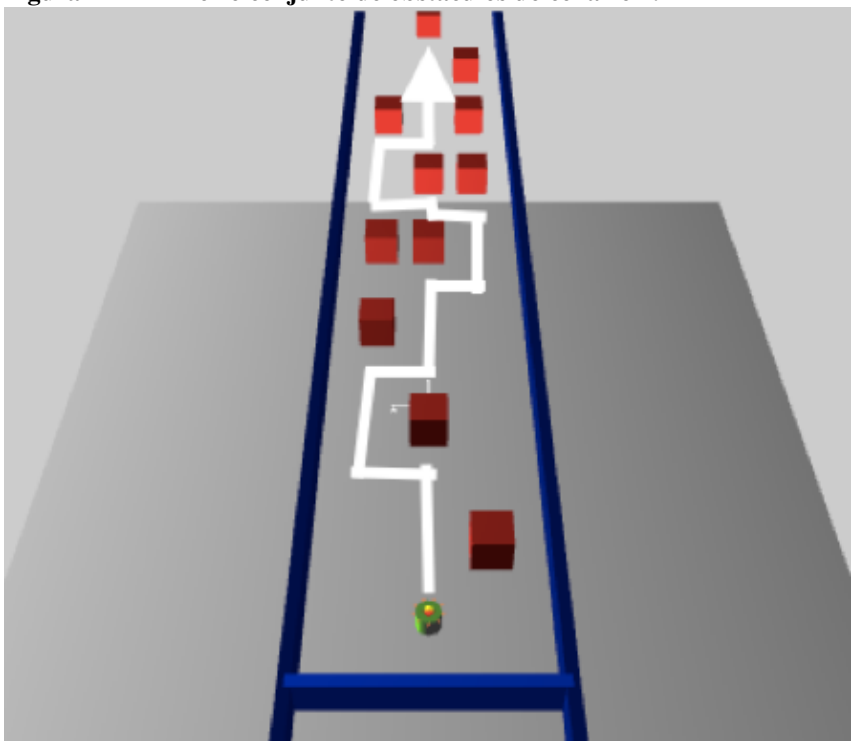
5.1 CENÁRIO 1

No primeiro cenário de teste, foram definidos obstáculos que podem ser desviados com manobras simples, que conseguem ser realizadas sem que o agente tenha que tomar uma decisão durante a manobra. Elas de dois tipos: a manobra pela esquerda e a manobra pela direita. Na Figura 21, é possível observar os obstáculos e o caminho que o veículo irá percorrer, note que para todos obstáculos existe espaço suficiente para a execução da manobra, sem a necessidade de uma intervenção durante a manobra.

O Gráfico 1, mostra as manobras realizadas e o tempo de cada manobra. O eixo das abcissas mostra a ordem de execução das manobras juntamente com as manobras que estão sendo executadas, e o eixos das ordenadas o tempo para a realização de cada manobra em segundos. Pela análise das informações encontradas no gráfico, temos que o tempo de execução da manobra pela esquerda leva em média de 4.5625 segundos, a manobra à direita a média é de 4.55 segundos e a média geral para as manobras simples é de 4.5583 segundos.

Esse cenário foi definido para testar tods os planos de decisão do agente, o motivo pelo qual existe uma maior amostra de manobras à esquerda, se deve ao fato que o agente opta por manobras à esquerda quando ambas as manobras podem ser realizadas. Note que esse cenário

Figura 21 – Primeiro conjunto de obstáculos do cenário 1.



Fonte: Autoria Própria.

segue todas as regras definidas para o ambiente e para o veículo (seção 4.1).

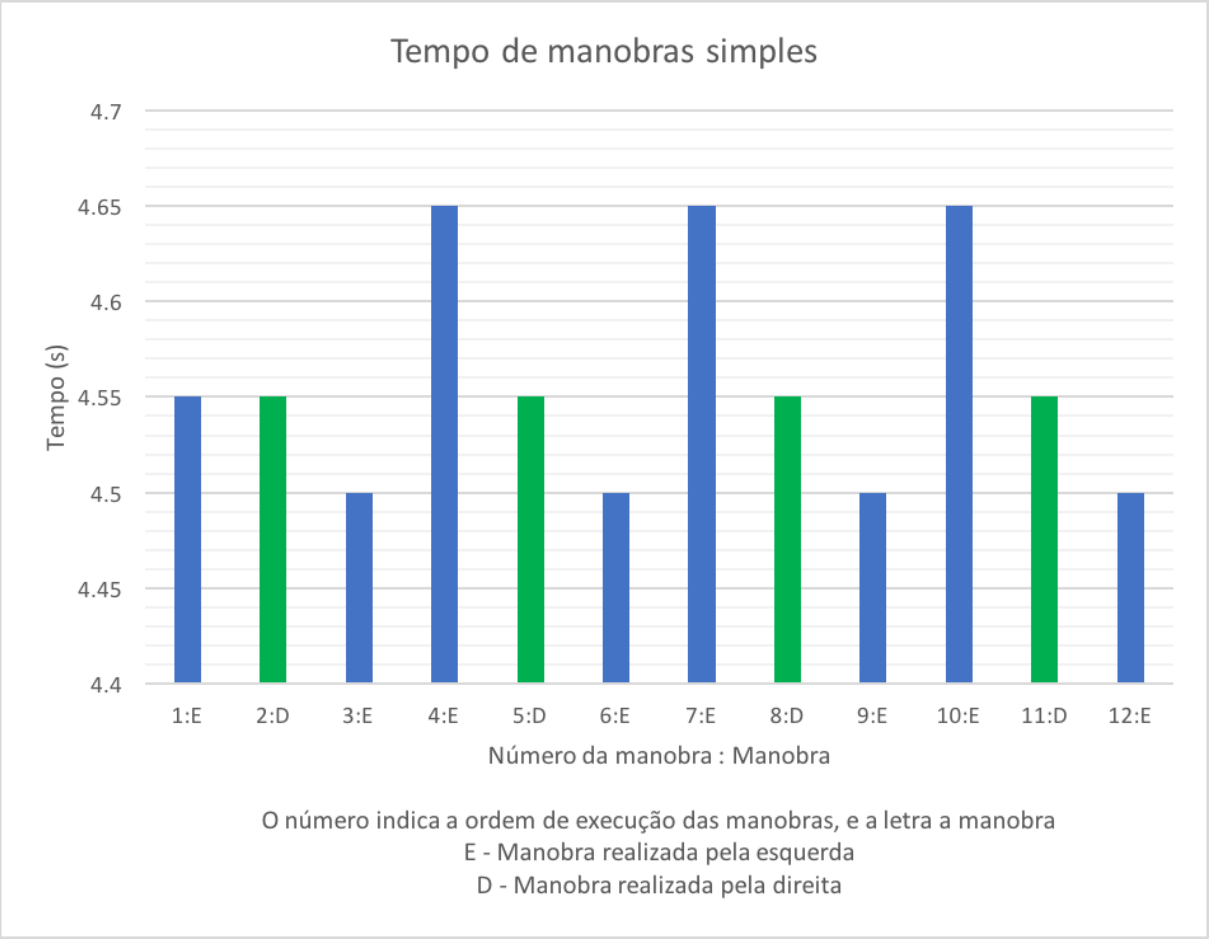
5.2 CENÁRIO 2

Nesse cenário, assim como no anterior o ambiente do simulador segue todas as regras. Esse cenário de teste buscar avaliar todas as possíveis decisões que o agente pode considerar, dessa forma as manobras disponíveis no simulador foram divididas em manobras simples, sendo elas à direita ou à esquerda e manobras complexas. Uma manobra é definida como complexa, quando uma manobra simples não pode ser completada com sucesso, necessitando que a manobra ou o trajeto seja alterado.

A Figura 22, ilustra uma parte do cenário de teste e o trajeto do agente para desviar do obstáculo, os obstáculos são posicionados de forma que existam manobras simples, porém a maior parte das manobras definidas para o cenário são complexas, sendo o principal objetivo de teste desse cenário.

O Gráfico 2, mostra o tempo de execução de manobras simples e complexas. Possui a mesma organização do Gráfico 1, ou seja, eixo das abcissas possui a ordem de execução das manobras e identificação da manobra e o das ordenadas o tempo para que cada manobra fosse completada. Nesse cenário de teste, o agente obteve um taxa de 100% de sucesso ao desviar dos obstáculos e o tempo médio para a execução de manobras à esquerda ficou em 4.5312 segundos, para manobras à direita em 4.6875 segundos e as manobras complexas em 6.175 segundos.

Gráfico 1 – Tempo de manobras simples.

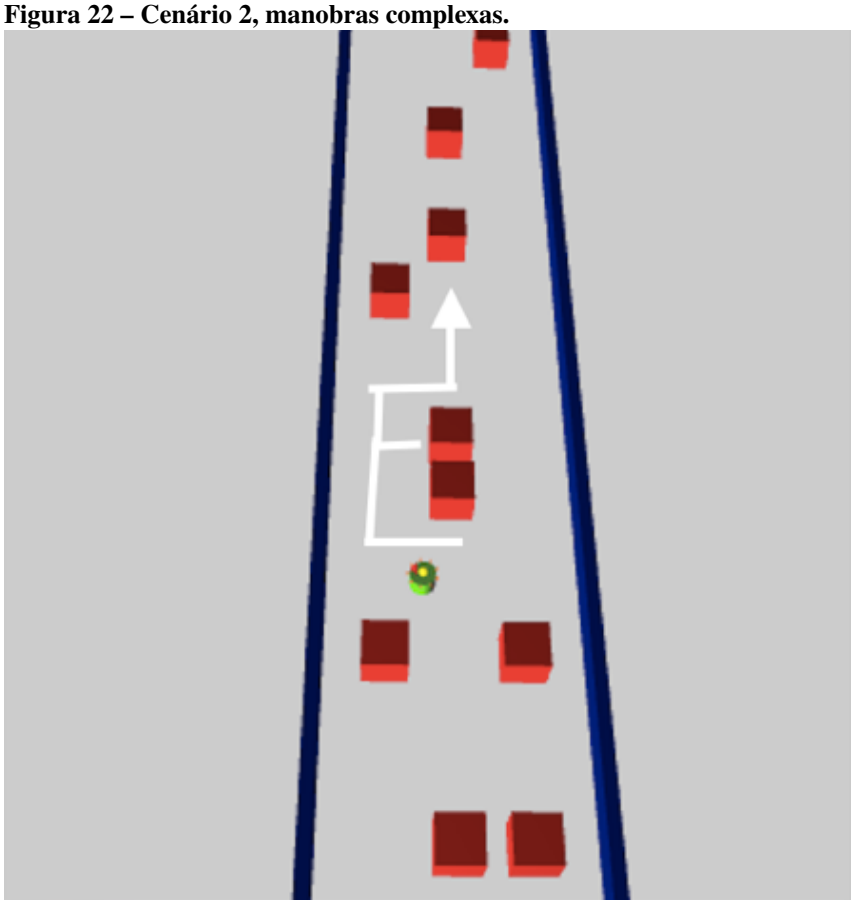


Fonte: Autoria Própria.

As manobras à esquerda e à direita não possuem uma diferença significativa em relação ao cenário 1, apesar de serem as mesmas manobras a diferença entre as médias se dá ao fato que as manobras levam em contam o momento em que um obstáculo é identificado. Casos onde o obstáculo é identificado a uma maior distância, aumentam o tempo de realização da manobra.

As manobras complexas por sua vez possuem um maior tempo de execução pois, durante uma manobra à esquerda ou à direita o veículo não será capaz de completar a manobra, necessitando assim alterar a manobra em execução para que o obstáculo seja evitado. Além disso, existe uma maior variação de tempo entre as manobras complexas, sendo relativo a decisão e o momento da decisão. Uma manobra alterada logo em sua fase inicial possui um menor tempo de execução, e a alterada em suas fases finais causa uma elevação no tempo de execução. A substituição da manobra (mudar de esquerda para direita, ou direita para esquerda), também eleva o tempo de execução.

Figura 22 – Cenário 2, manobras complexas.



Fonte: Autoria Própria.

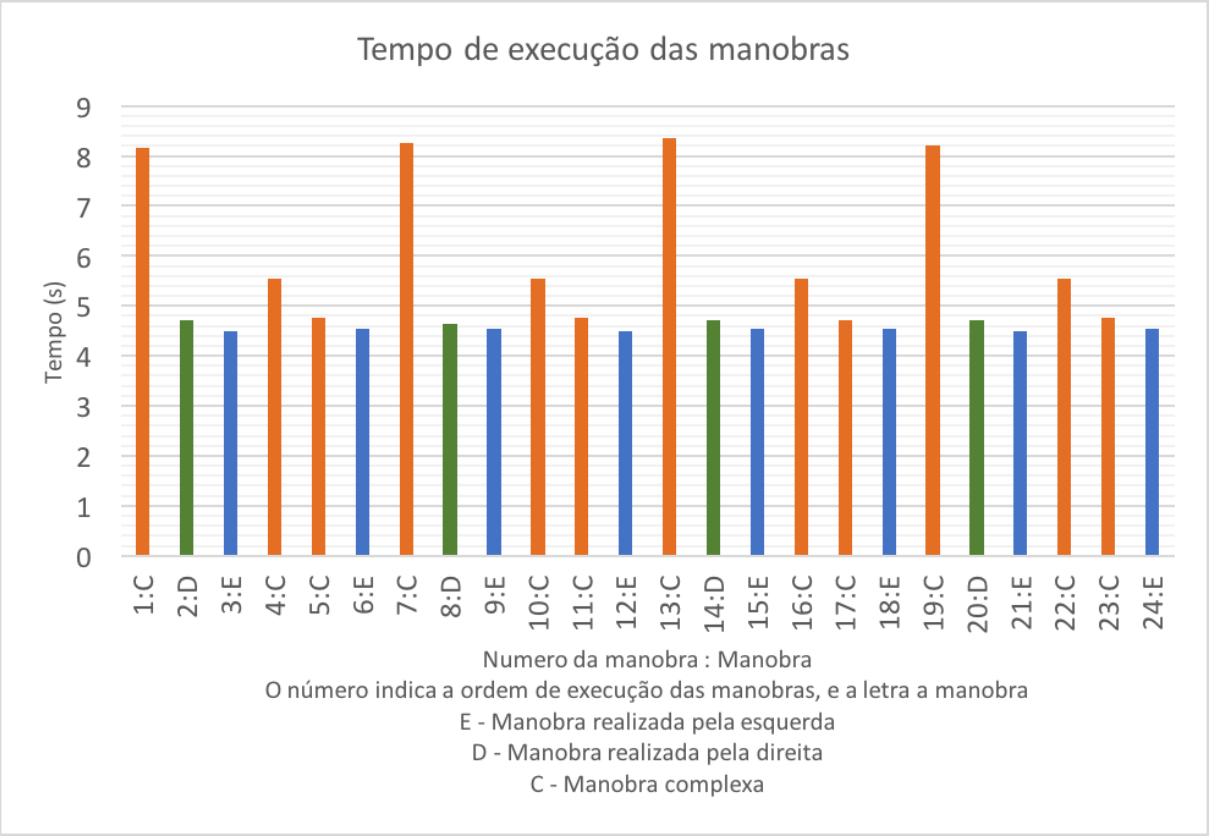
5.3 CENÁRIO 3

O cenário 3, diferente dos anteriores infringe uma das regras definidas para o simulador, nesse cenário o veículo passa a não ter uma velocidade constante e sim uma velocidade incremental. A velocidade inicial para esse cenário é de 1,6 m/s, sendo incrementada em 0,1 para cada obstáculo que o agente consiga desviar.

Para esse cenário, como o objetivo era encontrar a velocidade máxima que o veículo conseguiria alcançar, o ambiente do cenário 2 foi expandido até o momento em que o veículo colidiu.

A Figura 3, mostra as velocidades e o tempo para execução das manobras. A velocidade máxima que o agente conseguiu alcançar sem colidir com os obstáculos foi de 4.8 m/s, sendo que a 4.9 m/s o veículo colidiu e a manobra não foi completada. A média para as manobras a esquerda ficou em 2.3318 segundos, a direita 2.7416 e a média das manobras complexas ficou em 3.625 e geral de todas as manobras em 3.033. Através do Gráfico 3 e das médias, podemos perceber a relação entre o tempo de execução da manobra e a velocidade do veículo, quanto maior a velocidade menor o tempo de execução da manobra. Sendo que todas médias foram afetadas em relação aos cenários anteriores, devido a aceleração que o veículo sofre nesse cenário. Com os cenários de teste é possível perceber que o agente consegue executar tanto às manobras simples,

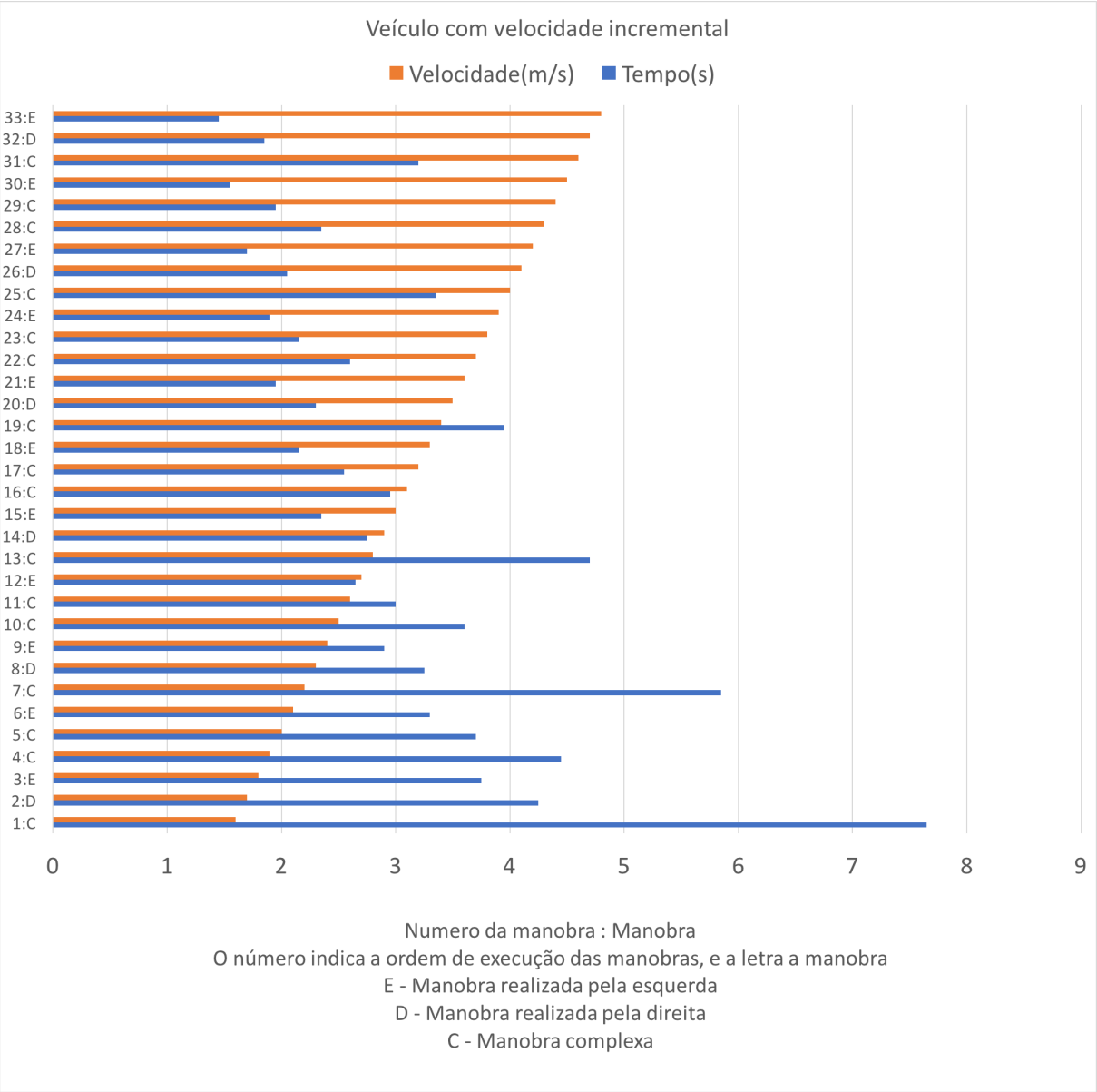
Gráfico 2 – Tempo de execução das manobras.



Fonte: Autoria Própria.

que foram propostas, mas também as manobras complexas para desviar de obstáculos que se encontrem no caminho da manobra que está sendo executada.

Gráfico 3 – Veículo com velocidade incremental.



Fonte: Autoria Própria.

6 CONCLUSÃO

Os veículos autônomos vêm revolucionando o mundo, desde o primeiro DARPA Grand Challenge, eles estão cada vez mais evoluídos, aproximando-se do nível 5 de autonomia. As empresas estão trabalhando diariamente para aprimorar seus veículos e com os avanços espera-se que nos próximos anos, à partir de 2017, já existam os primeiros veículos autônomos.

Assim como as empresas, que estão em busca do desenvolvimento de veículos autônomos, esse trabalho busca contribuir com o projeto AVIA (*Autonomous Vehicles with Intelligent Agents*) do Grupo de Pesquisa em Agentes de Software (GPAS), que busca o desenvolvimento de um veículo autônomo que utiliza agentes inteligentes como mecanismo que irá guiar o veículo.

As principais contribuições desse trabalho são:

- Construção de um agente capaz de guiar um veículo por entre obstáculos estáticos posicionados em três faixas, de forma que o agente preferencialmente irá percorrer o caminho pela faixa central, alterando entre as faixas somente quando necessário;
- Construção de um ambiente simulado;
- Integração entre o agente e o veículo (simulador), que permite a interação entre os componentes do agente (Jason e Cartago) com a simulação, obedecendo o protocolo de comunicação definido neste trabalho.

Esse trabalho, sendo um dos primeiros do projeto AVIA, será utilizado como base para trabalhos futuros. Os esforços aqui empregados permitirão que outros trabalhos sejam desenvolvidos a partir desse.

Utilizando esse trabalho como prólogo, alguns pontos relevantes podem ser considerados para as próximas pesquisas:

- Utilização de mais sensores para a percepção do ambiente, o que permitiria também a expansão do ambiente, permitindo ambientes mais complexos e com menos regras.
- A ampliação e evolução dos planos do agente de forma que esse consiga resolver os problemas que ficaram fora do escopo desse trabalho, como por exemplo o problema com altas velocidades.
- A criação de manobras mais elaboradas, para que o agente tenha mais opções de como desviar de obstáculos.
- Substituição do simulador, para um que consiga criar ambientes mais complexos e próximos da realidade de veículos autônomos. Sendo possível a reutilização da classe de comunicação e do agente, necessitando somente adaptar o simulador para que ele repasse as informações através do protocolo definido na seção 4.2.

- Substituição do agente. Criação de um agente em outra linguagem, como por exemplo através do Gwendolen (DENNIS; FARWER, 2008), que é uma linguagem para criação de agentes com a arquitetura BDI e que permite a verificação formal dos planos do agente. Uma possível alteração da arquitetura, como por exemplo a de agente baseados em lógica, onde a ações são dadas por deduções lógicas ou pela arquitetura de camadas, onde as decisões são feitas através de camadas de software onde cada uma delas é responsável por um nível de abstração (WEISS, 2013). Note que essas alterações, podem ser feitas sem a utilização de outro simulador, necessitando somente que o novo agente utilize o protocolo de comunicação.

O simulador e o agente implementados nesse trabalho correspondem a um pequeno passo na criação de um sistema maior que poderá ser utilizado para controlar um veículo. O simulador (Simbad) fornece uma solução simples que permite a visualização e testes dos planos do agente, e que apesar da sua simplicidade o simulador possui elementos que podem ser estudados mais a fundo em trabalhos futuros, como a câmera que permite a captura de imagens externas do que veículo está em contato e a utilização de mais sensores. O Simbad é uma ferramenta poderosa, um dos fatores é pelo fato de ser programado em Java, logo sua curva de aprendizagem é menor, permitindo um foco maior na implementação do agente.

Nas primeiras iterações do projeto AVIA, o agente deve ser o foco dos trabalhos, pois como o simulador funciona somente como uma ferramenta auxiliar ao agente, o projeto deve focar em desenvolver a inteligência do agente. Porém com um agente mais robusto, a substituição do Simbad seria necessária, a fim de simular componentes reais presentes nos carros autônomos, permitindo assim testes mais próximos a realidade. Entretanto para os próximos trabalhos, o Simbad cumpre o seu papel, possuindo opções que podem ser exploradas para deixar o ambiente variado e testar a inteligência do agente.

REFERÊNCIAS

ADOUANE, Lounis; BENZERROUK, Ahmed; MARTINET, Philippe. Mobile robot navigation in cluttered environment using reactive elliptic trajectories. **18th IFAC World Congress**, v. 18, n. 7, Agosto 2011.

BERG, Van Den; LIN, Ming C.; MANOCHA, Dinesh. Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation. **IEEE Conference Robotics and Automation**, IEEE, p. 1928–1935, 2008.

BERTONCELLO, Michele; WEE, Dominik. **Ten ways autonomous driving could redefine the automotive world**. [S.l.]: McKinsey and Company, 2015. Disponível em: <http://www.mckinsey.com/industries/automotive-and-assembly/our-insights/ten-ways-autonomous-driving-could-redefine-the-automotive-world>. Acesso em: 27 ago. 2016.

BORDINI, Rafael H.; HüBNER, Jomi Fred; WOOLDRIDGE, Michael. **Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)**. [S.l.]: John Wiley & Sons, 2007. ISBN 0470029005.

BRATMAN, Michael. What is intention? The MIT Press: Cambridge, MA, USA, p. 15–32, 1990.

CLIFFE, M. **Driverless cars – the route to more than smart cities**. 2012. Disponível em: <https://ingworld.ing.com/en/2016-IQ/12-collum-m-cliffe>. Acesso em: 28 ago. 2016.

COPPELIA, Robotics. **Coppelia Robotics**. 2017. Disponível em: <http://www.coppeliarobotics.com>. Acesso em: 02 nov. 2016.

DAFFLON, Baudouin *et al.* Adaptive autonomous navigation using reactive multi-agent system for control law merging. In: KOZIEL, Slawomir *et al.* (Ed.). **Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014**. Elsevier, 2015. (Procedia Computer Science, v. 51), p. 423–432. Disponível em: <http://dx.doi.org/10.1016/j.procs.2015.05.262>.

DENNIS, Louise A.; FARWER, Berndt. **Gwendolen: A BDI Language for Verifiable Agents**. [S.l.]: University of Aberdeen, 2008.

DUCATELLE, Frederick *et al.* Cooperative navigation in robotic swarms. **Swarm Intelligence**, v. 8, n. 1, p. 1–33, 2014. Disponível em: <http://dx.doi.org/10.1007/s11721-013-0089-4>.

FIORINI, Paolo; SHILLER, Zvi. Motion planning in dynamic environments using velocity obstacles. **I. J. Robotics Res.**, v. 17, n. 7, p. 760–772, 1998. Disponível em: <https://doi.org/10.1177/027836499801700706>.

JACAMO. **JaCaMo**. 2017. Disponível em: <http://jacamo.sourceforge.net>. Acesso em: 3 nov. 2016.

JENNINGS, Nicholas R.; SYCARA, Katia P.; WOOLDRIDGE, Michael. A roadmap of agent research and development. **Autonomous Agents and Multi-Agent Systems**, v. 1, n. 1, p. 7–38, 1998. Disponível em: <http://dx.doi.org/10.1023/A:1010090405266>.

LEVISON, Jesse. Towards fully autonomous driving: System and algorithms. **Intelligent Vehicles Symposium**, v. 4, p. 164–168, 2011.

MIT, Media Lab. **MIT Media Lab**. 2016. Disponível em: <https://www.media.mit.edu/>. Acesso em: 08 nov. 2016.

NHTSA. **Preliminary Statement of Policy Concerning Automated Vehicles**. 2012. Disponível em: <http://www.autoalliance.org/index.cfm?objectid=CC9678B0-A415-11E5-997E000C296BA163>. Acesso em: 22 ago. 2016.

RICCI, Alessandro; VIROLI, Mirko; OMICINI, Andrea. Carta go : A framework for prototyping artifact-based environments in MAS. In: WEYNS, Danny; PARUNAK, H. Van Dyke; MICHEL, Fabien (Ed.). **Environments for Multi-Agent Systems III, Third International Workshop, E4MAS 2006, Hakodate, Japan, May 8, 2006, Selected Revised and Invited Papers**. Springer, 2006. (Lecture Notes in Computer Science, v. 4389), p. 67–86. ISBN 978-3-540-71102-5. Disponível em: <https://doi.org/10.1007/978-3-540-71103-2_4>.

RUSSEL, Stuart J; NORVIG, Peter. **Artificial Intelligence - A Modern Approach**. New Jersey: Pearson Education, 2010.

SIMBAD. 2017. Dispoível em: <http://simbad.sourceforge.net>. Acesso em: 22 jan. 2017.

TESLA. 2016. Dispoível em: <https://www.tesla.com/models>. Acesso em: 26 ago. 2016.

UFMG, Computação Robótica. **Computação Robótica**. 2017. Disponível em: <http://coro.cpdee.ufmg.br>. Acesso em: 10 abril 2017.

UNIFEI. **Universidade Federal do Itajubá**. 2017. Disponível em: <https://www.unifei.edu.br/noticia/unifei_é_not%C3%ADcia_no_bom_dia_brasil_da_globo>. Acesso em: 13 jan. 2017.

USP, Laboratório de Robótica Móvel. **Laboratório de Robótica Móvel**. 2017. Disponível em: <http://www.lrm.icmc.usp.br>. Acesso em: 10 abril 2017.

WALLACE, Richard; SILBER, Gary. Self-driving cars: The next revolution. KPMG Center for Automotive Research, 2012. Acesso em: 23 ago, 2016.

WAYMO. 2016. Disponível em: <https://www.google.com/selfdrivingcar/>. Acesso em: 20 ago. 2016.

WEISS, Gerhard. **Multiagent Systems**. [S.l.]: The MIT Press, 2013.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R. **Agent Theories, Architectures, and Languages: A survey**. [S.l.: s.n.], 1994. 1-39 p.

____. **Intelligent agentes: theory and practice**. [S.l.: s.n.], 1995. v. 10. 115-152 p.

WOOLDRIDGE, Michael J. **An Introduction to MultiAgent Systems (2. ed.)**. [S.l.]: Wiley, 2009. ISBN 978-0-470-51946-2.