UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE ENGENHARIA ELETRÔNICA

ENGENHARIA ELETRÔNICA

BETINA CAROL ZANCHIN

ANÁLISE DO ALGORITMO A* (A ESTRELA) NO PLANEJAMENTO DE ROTAS DE VEÍCULOS AUTÔNOMOS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2018

BETINA CAROL ZANCHIN

ANÁLISE DO ALGORITMO A* (A ESTRELA) NO PLANEJAMENTO DE ROTAS DE VEÍCULOS AUTÔNOMOS

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

Orientador: Prof. Dr. Max Mauro Dias Santos

Coorientador: Prof. Me. Eng. Rodrigo Adamshuk Silva

PONTA GROSSA

2018

TERMO DE APROVAÇÃO

ANÁLISE DO ALGORITMO A* (A ESTRELA) NO PLANEJAMENTO DE ROTAS DE VEÍCULOS AUTÔNOMOS

por

BETINA CAROL ZANCHIN

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 08 de junho de 2018 como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
MAX MAURO DIAS SANTOS
Prof. Orientador

_____
RODRIGO ADAMSHUK SILVA
Coorientador

_____
ANGELO MARCELO TUSSET
Membro titular

_____
HUGO VALADARES SIQUEIRA
Membro titular

- A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica -

Dedico o presente trabalho em primeiro lugar a Deus por me manter resiliente e forte, e a minha família, por serem meu bálsamo em todos os momentos.

AGRADECIMENTOS

Chego ao final desta caminhada com imenso orgulho e saudade, orgulho de tudo que aprendi, e saudade de todas as experiências que fazem parte de quem eu sou, tanto as positivas quanto as negativas, afinal foi com cada uma delas que hoje chego ao final desta longa e sim, penosa caminhada. Digo penosa, não porque não gostei da estrada que percorri, mas sim por que os caminhos não foram repletos de flores, mas convenhamos nem deveriam, de que teria adiantado chegar até o fim sem sentir um amargo durante o caminho. Durante a engenharia aprendemos desde o primeiro semestre que é preciso muito mais do que só amor pelos números para chegar até o final, aprendemos a lutar com o nosso psicológico, porque a maior batalha acontece dentro de nós mesmos, e assim nos preparamos para a vida.

Eu agradeço aos professores que tive o privilégio de conhecer, aqueles que ensinam com um sorriso no rosto e alegria no coração, que fazem o que amam apesar de nem sempre serem apreciados como deveriam, o meu muito obrigada. Agradeço imensamente aos meus orientadores prof. Max e prof. Rodrigo, pela paciência, pelas orientações e pelo apoio incondicional, sem dúvida sem vocês este trabalho não seria possível.

Agradeço aos meus colegas e amigos, guerreiros e sobreviventes, vocês foram os que mais de perto presenciaram e testemunharam o meu crescimento durante toda a graduação, e eu a evolução de vocês também. A turma do Grupo de Sistemas Automotivos e a cada colega que eu encontrei nas disciplinas do curso, agradeço pela colaboração e parceria.

Agradeço a Deus por estar sempre presente, não me fazendo esquecer da minha fé para nunca desistir ou abaixar a cabeça para cada novo desafio. Agradeço a ele também pelas oportunidades, cada pessoa que passou na minha jornada me fazendo lembrar de sempre ser gentil, educada, prestativa, paciente e sonhadora. E agradeço pôr fim à minha família, que é meu alicerce e minha bússola, é pensando em cada um de vocês que eu dirijo a minha vida, meu jeito perfeccionista de ser é sempre buscando nunca os desapontar. Obrigada mãe pelo carinho infinito, obrigada pai pela força imensurável, obrigada minha irmã pelo apoio incondicional, obrigada meu irmão pela paciência de cada dia e obrigada tia e tio pela dose de amor extra.

RESUMO

ZANCHIN, Betina C. **Análise do algoritmo A* (a estrela) no planejamento de rotas de veículos autônomos.** 2018. 62 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2018.

Os veículos autônomos são em conjunto com a crescente onda da inteligência artificial o grande assunto do momento se falando de avanço tecnológico, transporte e robótica. O presente trabalho visa elucidar como um carro autônomo consegue se orientar e navegar pelo ambiente em que está inserido. Neste documento são apresentados os detalhes uma das várias metodologias de navegação autônoma existentes. Utilizando o algoritmo A* um agente inteligente deve ser capaz de encontrar o trajeto ótimo entre um ponto A e um ponto B, de maneira eficiente, visando a menor expansão de nós possível. O mesmo é então comparado ao algoritmo de Dijkstra que é amplamente conhecido no meio, e que serviu de inspiração na criação do A*. Através da análise dos resultados é possível perceber a melhora considerável que o algoritmo A* têm em relação ao de Dijkstra, isso sem necessitarmos de uma implementação muito mais complexa, como é explanado. Com a função heurística correta é possível ter resultados melhores nos mais diversos cenários a que o algoritmo está exposto.

Palavras-chave: Veículos Autônomos. Inteligência Computacional. Algoritmo Inteligente. Algoritmo A*. Algoritmos de Busca.

ABSTRACT

ZANCHIN, Betina C. **Analysis of the algorithm A* (A star) in the planning of autonomous vehicle routes.** 2018. 62 p. Work of Conclusion Course (Graduation in Electronic Engineering) - Federal Technology University - Paraná. Ponta Grossa, 2018.

The autonomous vehicles are in conjunction with the growing wave of artificial intelligence the great subject of the moment in speaking of technological advancement, transportation and robotics. This paper aims to elucidate how an autonomous car manages to navigate in the environment in which it is inserted. In this document is presented in detail one of several existing autonomous navigation methodologies. Using the A* algorithm a smart agent must be able to find the optimal path between a point A to a point B, in a efficiently way, aiming to expand smallest number of nodes as possible. The algorithm is then compared to the Dijkstra algorithm that is widely known, and which served as inspiration in the creation of the algorithm A*. By analyzing the results it is possible to perceive the considerable improvement that the algorithm A* has in relation to the Dijkstra's, without requiring a much more complex implementation, as explained. With the correct heuristic function it is possible to have better results in the most diverse scenarios to which the algorithm is exposed.

Keywords: Autonomous Vehicles. Computational intelligence. Intelligent Algorithm. Algorithm A *. Search Algorithms. Path Planning.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS

VA	Veículo Autônomo
EUA	Estados Unidos da América
V2X	Vehicle-To-Everything
V2V	Vehicle-To-Vehicle
V2P	Vehicle-To-Pedestrian
V2I	Vehicle-To-Infrastructure
V2N	Vehicle-To-Network
GPS	Global Positiong System
ACC	Active Cruise Control
LDWS	Lane Departure Warning System
LKA	Lane Keep Assist
PA	Park Assist
AEB	Automatic Emergency Braking
DM	Driver Monitoring
TJA	Traffic Jam Assist
AP	Automatic Pilot
IMU	Inertial Measurement Unit

LISTA DE SIGLAS

GM	General Motors
SRI	Stanford Research Institute
NHTSA	National Highway Traffic Safety Administration
IA	Inteligência Artificial
IC	Inteligência Computacional

LISTA DE ACRÔNIMOS

DARPA	Defense Advanced Research Projects Agency
SAE	Society of Automotive Engineers
OICA	Organização Internacional dos Fabricantes de Veículos a Motor
BASt	Instituto Federal de Pesquisa de Estradas da Alemanha
ADAS	Advanced Driver-Assistance Systems
LIDAR	Light Detection And Ranging

SUMÁRIO

1 INTRODUÇÃO

A busca pelo carro que se auto-guia está acontecendo, e ele está mais perto de se tornar uma realidade moderna, na próxima década é muito provável que se veja um veículo autônomo navegando pelas ruas já que as pesquisas avançam a passos largos. Apesar das desconfianças depositadas nos carros autônomos, na prática ele é muito mais seguro que um condutor humano, já que a direção controlada pelas máquinas e pelos algoritmos são milimetricamente calculadas para serem executadas da eficientemente.

Mas como ele de fato pode ser mais seguro? Um veículo sem motorista conta com os mais diversos sistemas de monitoramento lançados no mercado, por exemplo, uma série de sensores, como os ultrassônicos, as câmeras, os radares e também o sistema de GPS, para a sua orientação. As tecnologias dos carros passam por testes exaustivos. Não seria nada diferente nos testes e no desenvolvimento dos veículos autônomos, mas só que, afim de comprovar a sua eficácia e segurança, as rotinas de testes e os cenários simulados são ainda mais exigentes e desafiadores.

Somados a toda infraestrutura presente em um carro autônomo, temos ainda a inteligência artificial, que é talvez a fronteira mais interessante em termos de conhecimento e tecnologia. Não importa onde se observa, os sistemas inteligentes que visam melhorar e trazer conforto a vida das pessoas. A inteligência artificial está presente em várias atividades da atuação de um veículo autônomo, desde a visão computacional que analisa o ambiente em que o carro está inserido até no sistema de roteamento e de planejamento da trajetória do veículo, que é o tópico abordado no presente documento.

As principais montadoras de veículos, estão engajadas a encontrar a solução e a trabalhar juntas para que os desafios dos veículos autônomos sejam transpassados. Algumas até juntaram os esforços e participam de projetos colaborativos para desbravar este novo caminho aberto. Gigantes como Tesla, Audi, Mercedes e Volvo apoiam a inserção gradual das funcionalidades em seus veículos de série, já as novas marcas inseridas no mercado recentemente, como Waymo (Google), Baidu e até Apple, estão partindo para “batalhar” ou unir os esforços com as já bem estabelecidas e tradicionais concorrentes no mercado automobilístico.

1.1 DELIMITAÇÃO DO TEMA

Este projeto de pesquisa, delimitou-se em um estudo sobre como o algoritmo inteligente A* (estrela) pode ser implementado, visando suprir a falta de capacidade de veículos autônomos terrestres em planejar possíveis rotas de navegação, tendo como referência tecnologias e metodologias já existentes.

1.2 PROBLEMA E HIPÓTESE

Veículos autônomos já são uma realidade tangível e não mais um sonho longínquo de filme de ficção cientifica. Com essa nova era da indústria automobilística fomos expostos a desafios que antes nem sequer teriam sido cogitados, e com o passar do tempo, através de muito estudo e dedicação de pesquisadores do mundo todo que ultrapassamos as barreiras encontradas nessa jornada.

Um desses desafios abordados no presente trabalho como sendo a sua problemática é a barreira de navegação, mais especificamente a falta de capacidade de um veículo autônomo terrestre em planejar rotas de navegação. Ter a capacidade de se orientar, planejar rotas, buscar caminhos é uma habilidade fundamental para o funcionamento conceitual bem-sucedido de um veículo autônomo.

O estudo será construído partindo da hipótese de que utilizando um algoritmo inteligente, mais especificamente o algoritmo A*, é possível cobrir grande parte dos casos de roteamento de caminhos em que um veículo autônomo possa ser submetido, de maneira a explanar os passos do desenvolvimento e posteriormente da avaliação de desempenho do mesmo.

1.3 OBJETIVO GERAL

O objetivo geral do trabalho é: desenvolver e avaliar o algoritmo A* para o planejamento de rotas de veículos autônomos terrestres, visando resolver a problemática existente relacionada a navegação dos mesmos.

1.4 OBJETIVOS ESPECÍFICOS

Com o desmembramento do objetivo geral do projeto, os objetivos específicos pretendem:

- Aprofundar o leitor acerca dos fundamentos e tecnologias dos VAs;
- Explanar as diferentes abordagens e metodologias dos sistemas inteligentes;
- Explicar o algoritmo A* (estrela) e sua abordagem;
- Apresentar e detalhar o desenvolvimento do algoritmo escolhido;
- Avaliar o desempenho do algoritmo desenvolvido;
- Expor os resultados e as considerações do estudo elaborado.

1.5 JUSTIFICATIVA

A pesquisa em veículos autônomos pode ser dividida em duas grandes áreas, a área dos sistemas inteligentes de transporte e da robótica móvel. A área dos sistemas inteligentes de transporte diz respeito ao uso de equipamentos eletrônicos, métodos de comunicação e de processamento de dados ligadas ao segmento dos transportes. Já a ramificação robótica móvel é principalmente a área que estuda, desenvolve e constrói maquinas automáticas capazes de se locomover (PISSARDINI et al., 2013).

Dito isso, percebe-se que a área dos veículos autônomos possui riqueza e vastidão a problemática em questão foi escolhida não somente pela sua já evidenciada relevância técnica, mas também pela sua relevância social. Os veículos autônomos representam um marco na nova era da indústria automotiva assim como os veículos elétricos. Os veículos autônomos podem também reduzir o número de mortes ocasionados por acidentes, causados pelo julgamento menos assertivo de um humano.

Em comparação ao de um sistema inteligente, como também pode vir a diminuir engarrafamentos, diminuir emissão de gases poluentes na atmosfera terrestre devido a eficiência na sua condução e também, podem vir aumentar a mobilidade de pessoas que antes não possuíam habilitação para conduzir, ou ainda, que tenham algum tipo de dificuldade ou deficiência.

1.6 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está dividido em 6 capítulos. O capítulo 1 apresenta a introdução do mesmo, a delimitação do tema, o problema abordado, os objetivos tanto gerais como específicos e a justificativa para a elaboração do trabalho.

No capítulo 2 a fundamentação teórica acerca dos veículos autônomos é apresentada. Neste capítulo é explanado as características básicas e fundamentais para o entendimento do assunto, para o entendimento da área de veículos autônomos, é esclarecido o seu histórico, classificação, tecnologias, sensores e atuadores presentes e que estão em estudo.

O capítulo 3 conceitos básicos para o entendimento dos sistemas inteligentes são explanados, como as definições, as diferentes metodologias existentes e o algoritmo utilizado no presente trabalho.

No capítulo 4 e 5 são esclarecidos os passos utilizados para a elaboração do algoritmo, bem como a explicação de todas as escolhas tomadas como por exemplo a linguagem utilizada, a definição do problema a ser resolvido e também as metodologias utilizadas para avaliar a performance do mesmo. Finalizando com o capítulo da análise dos resultados obtidos.

O capítulo 6 conta com as considerações finais do trabalho, em que constam; um apanhado geral, uma breve discussão sobre os resultados obtidos, os objetivos alcançados e também uma avaliação sobre os trabalhos futuros a serem desenvolvidos a partir do estudo presente.

2 VEÍCULOS AUTÔNOMOS

Neste capítulo são apresentados conceitos referentes aos veículos autônomos, abordando as definições encontradas atualmente, sua história, pontos chaves, tecnologias e funcionalidades.

2.1 DEFINIÇÃO

Autônomo significa ser independente ou ter o poder de se auto governar. Para a Administração Nacional de Segurança Rodoviária dos EUA (National Highway Traffic Safety Administration - NHTSA) um veículo autônomo ou ainda veículo totalmente automatizado é aquele que; “a operação ocorra sem intervenção direta de um motorista, e ainda que seja projetado a não esperar que o condutor monitore o modo de condução”.

A pesquisa em VAs se origina em dois diferentes campos de pesquisa: Os sistemas inteligentes de transporte e a robótica móvel. Os sistemas inteligentes de transporte usam de equipamentos eletrônicos, métodos de comunicação e processamento de dados integrados com o transporte público e os materiais para atingir o objetivo de trazer conforto aos usuários, ao mesmo tempo em que facilitam o gerenciamento operacional pelas empresas. Já a robótica móvel é basicamente a área de pesquisa que estuda, desenvolve e constrói maquinas, seja elas robôs ou veículos de transporte capazes de se locomover (PISSARDINI et al., 2013).

Para a implementação de um veículo autonômo existem alguns pontos chaves que precisam de mais atenção, como a conectividade, os atuadores, os sensores, a navegação, os processadores e ainda os algoritmos de software.

A navegação é de suma importância para o veículo saber se orientar sobre onde está e para onde vai. Hoje em dia o sistema mais utilizado para a localização é o *Global Positiong System* (GPS). Ele é baseado na triangulação de 3 ou mais satélites usando o intervalo de tempo entre a emissão e a recepção de ondas de rádio para calcular então a distância aproximada do mesmo. Por isso são necessário 3 ou mais satélites para uma resposta no mínimo aceitável, pois sem essa triangulação poderia haver divergências muito grandes quanto a localização do mesmo (KAPLAN, 1996).

Para aumentar a performance dos sistemas utilizando o GPS e a qualidade dos cálculos do posicionamento a Unidade de Medida Inercial (*Inertial Measurement Unit* - IMU) tem sido cada dia mais utilizado. O IMU utiliza-se de um dispositivo eletrônico capaz de coletar velocidade e aceleração. Para conseguir captar tais informações os sistemas devem possuir giroscópios e acelerômetros e ainda algumas vezes magnetômetros capazaes de medir o campo magnetico envolta do equipamento. O IMU faz parte do Sistemas de Navegação Inercial (*Inertial Navigation System* - INS) que é uma plataforma inercial que utiliza a técnica de *dead reckoning* para calcular a posição com mais precisão. *Dead reckoning* é o processo de calcular a posição atual de um objeto utilizando os dados da sua posição anterior, é muito utilizado atualmente para melhorar a performance do GPS quando este por algum motivo acaba por perder o sinal, como em túneis ou em centros das grandes cidades (KING, 1998).

2.2 HISTÓRIA

O marco inicial para as pesquisas dos VAs ocorreu no ano de 1939 com a Feira Mundial de Nova Iorque nos Estados Unidos da América (EUA), a feira Futurama como ficou conhecida, foi uma expressão futurística do projetista Norman Melancton Bel Geddes que patrocinado pela empresa General Motors (GM) apresentou uma ideia conceitual de como seria o mundo em vinte anos, ou seja, próximo a década de 1960. A Figura 1 ilustra um dos modelos criados pelo projetista. O principal destaque da exibição foi um protótipo de sistema de rodovias automatizado onde as estradas poderiam corrigir as falhas de condução humana (WETMORE, 2003).

Após o término da segunda guerra mundial diversos equipamentos e tecnologias utilizados para fins militares foram adaptados para automatizar algumas funcionalidades dos veículos. Não seria diferente com o automóvel como exemplo temos a tecnologia do radar que foi inicialmente implementada para fins militares, mas que foi adaptada e é muito utilizada nos carros mais modernos para alertar e auxiliar na condução dos veículos. Em 1964 a GM apresentou uma nova Feira Mundial em Nova Iorque em que foi atualizado a sua visão de futuro para os sistemas de transporte. Na nova visão conceitual apresentada uma torre de controle operaria a

direção, freios e velocidade de cada veículo em uma pista automática (WETMORE, 2003).

Figura 1 - Exposição da feira futurama

Fonte: General Motors

No fim da década de 1960 apareceram os primeiros esforços relacionado a robótica móvel. Entre os anos de 1966 e 1972 um robô chamado Shakey, ilustrado na Figura 2, foi desenvolvido pelo *Stanford Research Institute* (SRI) financiado pela DARPA. Com o robô Shakey o instituto buscava prover a autonomia de movimentos sobre superfícies, o robô era constituído por uma plataforma sobre roda, equipada com uma câmera de TV estéreo, sensores ultrassônicos e sensores de toque. Todos esses dados coletados pelos sensores eram então enviados por radiofrequência para um computador *mainframe* que era responsável por realizar todos os cálculos, planejamento de rotas e decidir as ações a serem realizadas, essas informações então eram retransmitidas para a navegação de Shakey (GAGE, 1995).

Com o intuito de tornar mais eficiente o trafego urbano diversos grupos de pesquisa da Europa se uniram e criaram o projeto EUREKA *Prometheus* (PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety) entre os anos de 1987 a 1995. Ernest Dickmanns, da BMW, da Daimler-Benz e da Jaguar apresentou um Mercedes-Benz Classe–S modificado, que foi capaz de se auto dirigir por 1000 quilômetros, com velocidades de até 130km/h, seu sucessor o veículo

VaMoRs-P foi ainda mais longe sendo capaz de executar manobras de ultrapassagem na Autoban alemã e atingido velocidades de até 160km/h.

Figura 2 - Robô Shakey

Fonte: SRI International

Com o intuito de tornar mais eficiente o trafego urbano diversos grupos de pesquisa da Europa se uniram e criaram o projeto EUREKA *Prometheus* (PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety) entre os anos de 1987 a 1995. Ernest Dickmanns, da BMW, da Daimler-Benz e da Jaguar apresentou um Mercedes-Benz Classe–S modificado, que foi capaz de se auto dirigir por 1000 quilômetros, com velocidades de até 130km/h, seu sucessor o veículo VaMoRs-P foi ainda mais longe sendo capaz de executar manobras de ultrapassagem na Autoban alemã e atingido velocidades de até 160km/h.

Entre esse meio tempo vários foram os esforços para se desenvolver a tecnologia dos veículos autônomos, fomentados principalmente no meio acadêmico financiados ou não por agências governamentais ou instituições privadas de modo que pequenos avanços foram de fato alcançados, mas foi em 2002 que se iniciou uma das mais conhecidas competições do meio. Neste ano o DARPA lançou o evento *Grand Challenge,* que nada mais era do que uma competição visando estimular as pesquisas na área da navegação veicular, a primeira “corrida” ocorreu de fato no ano de 2004 com um prêmio no valor de um milhão de dólares americanos. Na competição os veículos deveriam navegar e percorrer um trajeto de cerca de 228 km sem

intervenção alguma de humanos em no máximo 10 horas, o cenário escolhido não poderia ser mais desafiador, contendo variações no terreno, alta quantia de sujeira e curvas acentuadas (THRUN et al., 2006; DARPA, 2004).

Na primeira edição da competição tiveram 107 equipes inscritas, sendo que 15 efetivamente correram, mas nenhum dos participantes navegou mais do que 5% de todo o percurso. Na segunda edição realizada no ano de 2005 o prêmio para o vencedor era de não mais e US$ 1 milhão, mas sim US$ 2 milhões e nesta edição houve um total de 195 equipes inscritas, mas somente 23 efetivamente participaram. Nesta edição o robô da Universidade de Stanford apelidado de Stanley (é ilustrado na Figura 3) foi o grande vencedor, Stanley conseguiu finalizar a prova com 6 horas, 53 minutos e 08 segundos, o robô foi produzido pela equipe de pesquisadores da universidade, auxiliados e patrocinados majoritariamente pelas empresas Intel Research e pela Volkswagen. Era um Volkswagen Touareg R5 TDI com uma plataforma contendo 6 processadores Intel e um conjunto de sensores e atuadores que auxiliavam na navegação autônoma, para reduzir o risco de atraso de processamento o software não era centralizado, sendo que os módulos eram executados paralelamente e sem sincronismo, sendo integrados pela utilização de marcas temporais sobre os dados (THRUN et al., 2006; DARPA, 2004).

Figura 3 - Carro Stanley

Fonte: Thrun, 2006

A DARPA acabou por realizar uma terceira edição da competição no ano de 2007, mas com algumas mudanças e variações, foi então realizado o DARPA Urban Challenge. O cenário de provas foi modificado dando lugar a um ambiente urbano simulado onde os competidores deveriam, obedecer as leis de trânsito e ainda ocasionalmente interagir entre si, nesta competição participaram onze veículos sendo que o grande vencedor foi o veículo Boss da Universidade Carnegie Mellon, Boss possuía o sistema de controle integrado com sensores *laser*, radares e câmeras, com

as quais tinha a capacidade de reconhecer regras de trânsito detectar outros veículos e interagir de modo seguro com os mesmos.

É possível perceber que a história e os avanços dos veículos autônomos terrestres estão intimamente ligados ao avanço tecnológico também dos computadores e dos processadores, com a vinda de novas tecnologias podemos afirmar que os VAs já são o nosso presente e não mais um futuro distante.

2.3 CLASSIFICAÇÃO DOS NÍVEIS DE AUTONOMIA

Para diferenciar os tipos e os níveis de autonomia de um sistema inteligente existem classificações que ajudam a determinar se o nível de autonomia desse sistema, ou ainda o nível de “inteligência” do mesmo. Um sistema totalmente autônomo deveria talvez ter a habilidade de se auto reparar em caso de falha nos sistemas ou dos seus componentes. Para alcançar um elevado nível de autonomia o sistema deve ser capaz de realizar funções adicionais ao que os sistemas convencionais têm capacidade de fazer, como por exemplo o seu rastreamento e o entendimento do meio ambiente que está inserido.

Para os carros autônomos isso não é diferente, atualmente existem alguns sistemas de classificação de autonomia coexistindo, apesar de serem diferentes eles não se distanciam muito no que diz respeito aos padrões adotados. Um dos sistemas mais difundidos e adotados hoje na indústria é o padrão adotado e sugerido pela Sociedade de Engenheiros Automotivos (SAE), eles desenvolveram um sistema harmonioso para descrever os seis níveis de uma direção autônoma. Adicionados a SAE existem também outras organizações mundiais que criaram seu próprio padrão de classificação como a Organização Internacional dos Fabricantes de Veículos a Motor (OICA), o Instituto Federal de Pesquisa de Estradas da Alemanha (BASt) e a instituição norte americana de Administração Nacional de Segurança no Tráfego Rodoviário (NHTSA).

Em setembro de 2016 a NHTSA adotou o padrão sugerido pela SAE para ser usado na sua política federal de veículos automatizados e com isso a SAE agora disponibiliza gratuitamente para consulta e utilização o documento “J3016: Taxonomia e Definições para Termos Relacionados aos Veículos Motorizados e Sistemas de condução automatizados” (SAE, 2016; NHTSA, 2016).

No documento da SAE taxonomias e definições são elaboradas para darem embasamento no sistema de classificação criado e nele são descritos seis níveis de direção autônoma, o padrão adotado é consistente com a indústria automotiva atual, pode ser usado por diversas áreas que vierem a usar suas definições desde a engenharia até o judiciário pois com o padrão elaborado se eliminam pontos de confusão ou que poderiam gerar lacunas nas duplas interpretações.

Para um melhor entendimento sobre as diferenças e sobre a classificação adotada é preciso discorrer sobre os termos moldados pelo documento J3016 (SAE, 2016) afim de explanar a classificação propriamente dita, como:

- *Dynamic Driving Task* (DDT) ou em português Tarefa de Condução Dinâmica são todas as funções operacionais ou táticas em tempo real necessárias para operar um veículo no tráfego rodoviário;
- *Operational Design Domain* (ODD) são as condições especificas que um dado sistema de direção autônoma ou um recurso dele são projetados para funcionar incluindo, mas não limitado aos modos de condução;
- *Automated Driving System* (ADS) é o conjunto de software e hardware que coletivamente são capazes de realizar todo o DDT de forma completa independentemente se está limitada a um ODD ou não. Este termo é usado especificamente para descrever sistemas de nível 3, 4 e 5.
- *Driving Automation System or Technology* é o conjunto do software e hardware que coletivamente tem a capacidade de realizar parte ou totalmente o DDT. Este termo é usado genericamente para descrever qualquer sistema desde o nível 1 até o nível 5.
- *DDT Fallback,* tarefa de condução dinâmica de retomada é a resposta do usuário ou por uma ADS para executar o DDT ou para alcançar um nível de risco mínimo após a execução do DDT, após alguma detecção de falha ou ainda na saída de um ODD.
- *Object And Event Detection and Response* (OEDR) em português Detecção de Objetos e Eventos e Resposta são as sub tarefas do DDT que incluem monitoramento do ambiente (detectar, reconhecer e classificar objetos e eventos, preparando-se para responder conforme

necessário) e execução de uma resposta adequada a tais objetos e eventos.

Depois de esclarecidos os termos utilizados para se definir sistemas de automação, ambientes e tarefas de sua responsabilidade então pode-se definir a que níveis se encaixam cada sistema. O nível de autonomia de um sistema é determinado nesse caso de acordo com a SAE pelas funcionalidades do sistema de direção autônoma e a performance e integração entre o sistema e o usuário (mesmo se não houver um) (SAE, 2016)

Os níveis inferiores da classificação (1 e 2) se referem aos casos em que o condutor humano continua a executar parte do DDT enquanto o sistema de automação esteja ativado. Os três níveis superiores (3, 4 e 5) se referem aos casos em que o ADS realiza todo o DDT enquanto ativado.

- Nível 0 – Sem Automação; todas as manobras são executadas por um condutor humano;
- Nível 1 – Assistência ao condutor; durante um ODD específico o sistema de condução realiza o controle do movimento lateral ou longitudinal do veículo, nunca os dois simultaneamente, o condutor executa todas as outras tarefas restantes;
- Nível 2 – Direção autônoma parcial; durante um ODD específico o sistema de condução faz ambos os controles de movimento lateral e longitudinal do veículo, enquanto o motorista monitora e supervisiona os sistemas de automação realizando a OEDR;
- Nível 3 – Direção autônoma condicional; durante um ODD específico um ADS executa o DDT inteiramente, mas o usuário é o responsável por assumir o controle caso ocorra algum imprevisto, falha do sistema ou ainda se o próprio sistema requisitar;
- Nível 4 – Direção autônoma elevada; durante um ODD específico um ADS executa todo o DDT e o *DDT fallback*, ou seja, mesmo as manobras de segurança após alguma eventual falha são realizadas pelo sistema, sem esperar que o usuário responda a um pedido de intervenção;
- Nível 5 – Direção autônoma total; sem necessitar uma ODD específica o ADS executa todo o DDT e o *DDT Fallback* sem esperar que o

> usuário responda a um eventual pedido de intervenção. Por exemplo um sistema de nível 5 é capaz de uma vez setado seu destino, operar o veículo por viagens inteiras independentemente de rodovias, trafego, condições ambientais entre outros.

Ainda de acordo com o documento da SAE é de responsabilidade das empresas responsáveis pelos sistemas de automação definirem os requisitos, o ODD, as características de operação, as maneiras de uso e ainda o nível de automação da sua aplicação.

2.4 SISTEMAS SENSORIAIS E DE ATUAÇÃO

Para que o carro possa ter a capacidade de interagir com o meio em que está inserido é necessário que ele possa “senti-lo” como os humanos possuem sentidos sensoriais através do olfato, da visão, audição e do tato é preciso fornecer ao veículo equipamentos que sejam capazes de exprimir esses sentidos. Com estes “sentidos” captados é preciso então traduzir todas estas mensagens e convertê-las em uma linguagem em que o sistema possa entender e usar as mesmas para construir estratégias e lógicas afim de interagir com o ambiente por meio dos atuadores. Todo esse caminho ocorre por que o veículo como um todo está equipado com hardware e software necessário.

Os atuadores são os componentes do veículo responsáveis por mover e controlar os mecanismos, é com eles que os sistemas de controle e efetuam ações com o meio propriamente dito, para acionar os freios, a direção e ou aceleração por exemplo. Há alguns tipos diferentes de atuadores indicados para sistemas e finalidades distintas: atuadores hidráulicos atuam em fluidos, atuadores pneumáticos convertem a energia formada pelo vácuo ou ar comprimido, atuadores elétricos formados por um motor elétrico converte a energia elétrica em energia mecânica e assim por diante cada atuador possui um meio de acionamento específico. Nos veículos, os atuadores são utilizados para responder a um sinal enviado de alguma Unidade de Controle Eletrônica (ECU).

ECUs são processadores presentes em todos os veículos produzidos atualmente, eles são essenciais para a maioria das aplicações ADAS, uma ECU é um

dispositivo eletrônico que controla as operações do sistema que para ele foi programado. Nos modelos mais modernos há inúmeras ECUs sendo usadas em um único veículo, como por exemplo no controle do motor, no controle dos freios, nos sistemas de monitoramento, entre outros, alguns veículos chegam a conter mais de 80 ECUs e esse número continua a crescer, pois há mais e mais sensores e atuadores que precisam ser são controlados eletronicamente, na Figura 4 observamos o exemplo de uma ECU.

Figura 4 - Unidade de controle eletrônica (ECU)

Fonte: Denso Corporation

Os sensores são a parte principal para um veículo totalmente autônomo, pois como dito anteriormente é com eles que se é possível “enxergar” o ambiente e traduzir as suas intenções, da mesma maneira ou até melhor que um humano faria. Mas para isso ele precisa de uma diversidade desses dispositivos e cada um tem uma finalidade específica, alguns dos tipos mais utilizados são os *LIDAR,* RADAR, ultrassônicos e as câmeras por exemplo.

Light Detection And Ranging (LIDAR), o sensor LIDAR pode ser aplicado em diferentes áreas de estudo, através do seu mapeamento em três dimensões pode ser utilizado para determinar a geografia de um local ou ainda estudar a vegetação de alguma floresta por exemplo. Nos veículos autônomos o LIDAR tem ajudado no reconhecimento do ambiente em que o veículo está inserido, o LIDAR utiliza luzes laser pulsadas infravermelha para medir a distância de um objeto contando o tempo necessário para a luz emitida retornar ao emissor. Usando o LIDAR é possível criar mapas tridimensionais dando então dados mais robustos para que o sistema possa efetuar os cálculos necessários, além disso o LIDAR não necessita de luz para

funcionar, sendo então muito útil para casos com pouca ou nenhuma visibilidade da câmera de vídeo por exemplo, na Figura 5 pode-se ver como é um modelo de LIDAR.

Figura 5 - Modelos de LIDAR

Fonte: Velodyne LiDAR

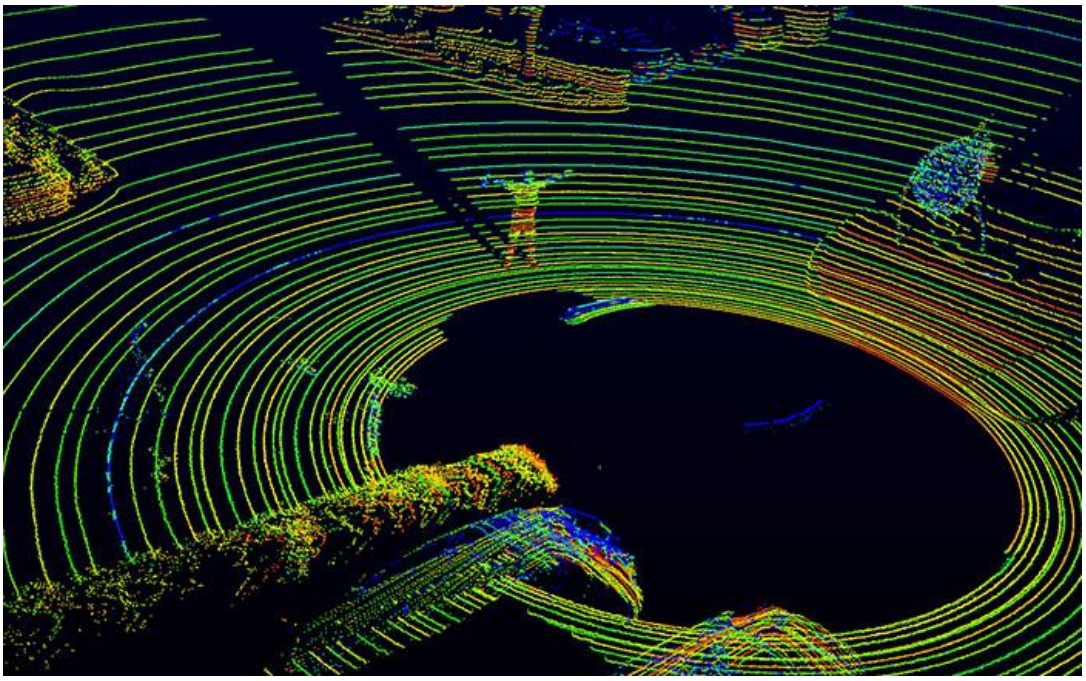
Na Figura 6 observa-se o exemplo de um mapa tridimensional criado através dos dados captados e traduzidos pelo LIDAR, onde é possível perceber de modo apurado a distância e os objetos ao redor do sensor, como os veículos, a vegetação e o pedestre que se encontra a frente.

Radio Detection and Ranging (RADAR) este sensor já muito utilizado nos automóveis modernos é muito similar ao LIDAR, mas ao invés de utilizar as luzes infravermelhas ele usa ondas eletromagnéticas, que são emitidas por uma antena e refletida pelos objetos e novamente pelo cálculo do intervalo de tempo entre a emissão e a captação do sinal refletido estimasse a distância que o objeto se encontra. Com o avanço da tecnologia já e possível determinar a velocidade do objeto sendo rastreado, também pode ser utilizado com precisão em condições adversas do ambiente como chuva, nevoa ou neve, o LIDAR ainda não responde muito bem a estas condições.

Sensores ultrassônicos já há muito tempo utilizados no mercado em várias áreas não só as dos automóveis. Ele é particularmente muito utilizado para detectar objetos muito próximos, seu princípio de funcionamento é muito parecido ao utilizado

pelos morcegos que utilizam para detectar os objetos e medir suas distâncias aproximadas, utilizando da propagação de ondas sonoras.

Figura 6 - Demonstração dos dados adquiridos por um LIDAR



Fonte: Velodyne LiDAR

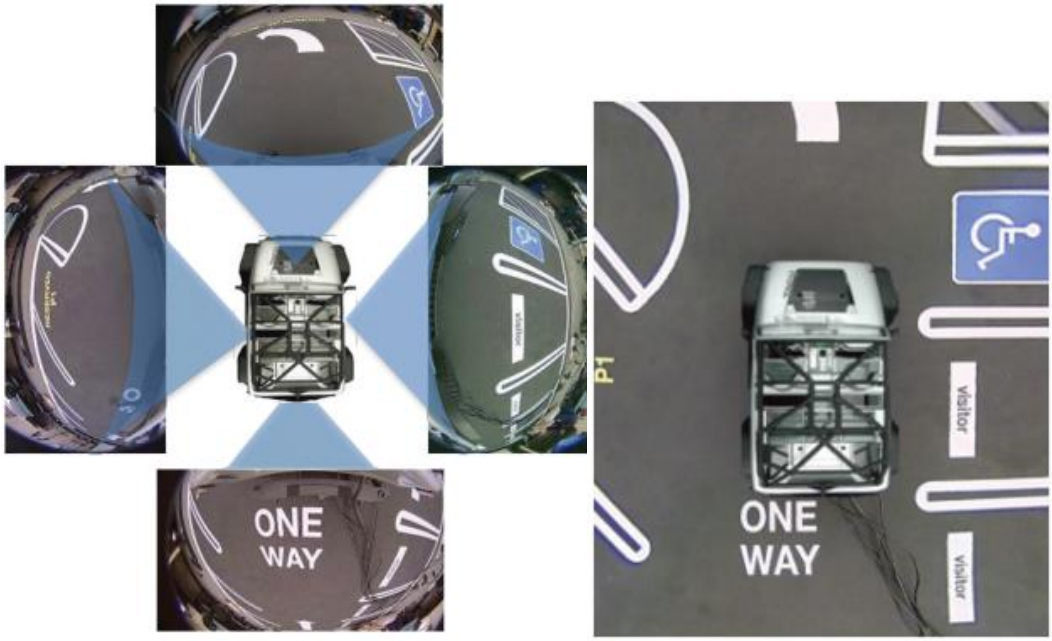
Sensores ultrassônicos já há muito tempo utilizados no mercado em várias áreas não só as dos automóveis. Ele é particularmente muito utilizado para detectar objetos muito próximos, seu princípio de funcionamento é muito parecido ao utilizado pelos morcegos que utilizam para detectar os objetos e medir suas distâncias aproximadas, utilizando da propagação de ondas sonoras.

E temos ainda na categoria de sensores as câmeras de vídeo, muitos modelos são usados para diferentes tipos de aplicações assim também não é diferente para os veículos, o avanço da tecnologia das câmeras em si, como suas resoluções, qualidade de lentes e captação trouxe um leque de possibilidades de aplicações para as mesmas, aliado a isso temos o constante avanço nos estudos sobre processamento de imagens, que se utilizam de artifícios de programação e software para melhorar a performance alcançada.

Há alguns tipos específicos de câmeras utilizadas no meio automotivo como as *Surround Cameras* que são utilizadas nos veículos para auxiliar o motorista nas manobras de estacionamento ou quando é preciso ter uma visão mais ampla do automóvel até para se evitarem possíveis acidentes com os ditos “pontos cego”. Na

Figura 7 é ilustrado esse exemplo, em que temos a vista das quatro câmeras separadas que foram tratadas com algoritmos de processamento de imagens e assim alinhadas para exibirem para o usuário uma vista superior do veículo.

Figura 7 - Visão dos arredores



Fonte: Texas Instruments

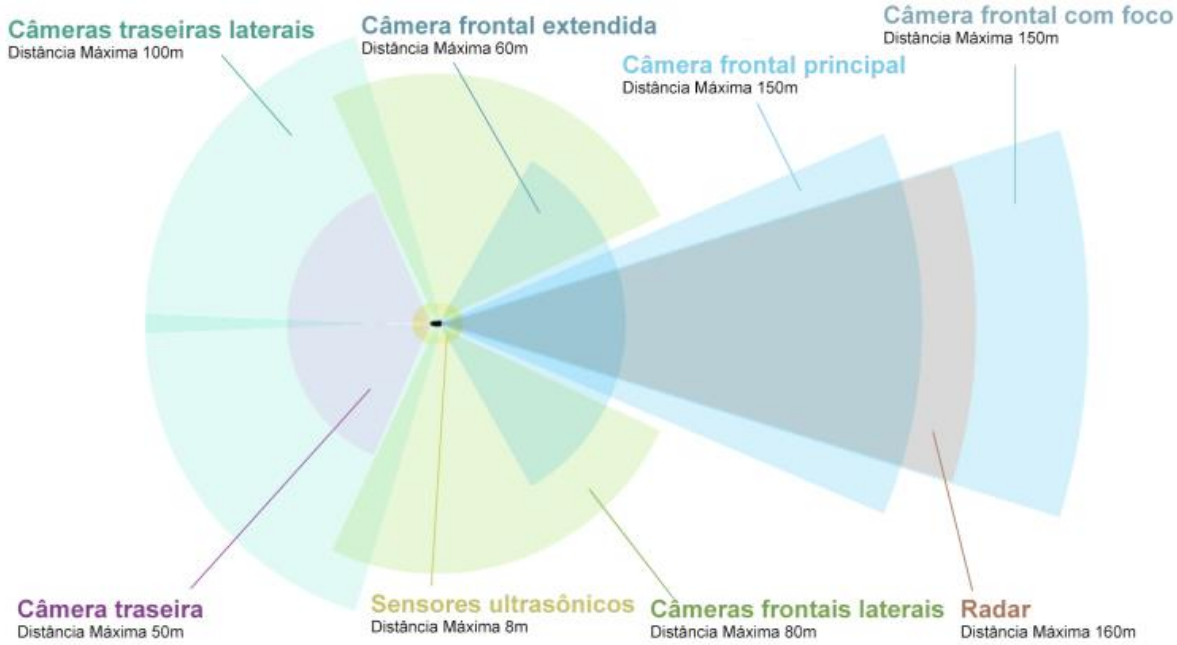
Têm-se ainda as câmeras com visão monocular e com visão estéreo. Visão monocular é aquela em que as imagens são adquiridas separadamente, o campo de visão é maior, mas a percepção de profundidade acaba por ficar limitada. Nos humanos este tipo de visão acaba ocasionando inúmeras limitações em várias atividades consideradas cotidianas, como a prática de esportes, exercício de algumas profissões ou ainda algumas atividades de recreação como por exemplo assistir a filmes em tecnologia 3D. Na maioria dos casos em que são utilizadas estas câmeras com visão monocular elas são instaladas no para-brisas dos carros, para poder coletar imagens da rodovia, detectar as faixas e ainda observar a frente do para evitar colisões dianteiras.

A visão estéreo é a visão baseada na natureza já que seres humanos e outros animais possuem este tipo de visão, com está visão o senso de profundidade é mais apurado sendo assim é possível mapear os ambientes tridimensionalmente. Câmeras com visão estéreo são compostas basicamente de duas câmeras colocadas a uma distância fixa entre elas, para tratar e analisar melhor as imagens obtidas pelas duas câmeras, é necessário que as imagens passem por um “tratamento”, utilizando o

processamento de imagens que irá sintetizar as informações adquiridas pelas câmeras. Com a visão estéreo é possível implementar inúmeras funcionalidades nos automóveis, como o sistema de frenagem emergencial, detecção de pedestres, reconhecimento de sinais de trânsito, alerta de saída de faixa e muitas outras funções, é claro que muitas dessas funções seriam mais eficientes se fossem integradas com outros sensores também, especialmente para a leitura de cenários mais complexos.

A Figura 8 exemplifica ilustrando como os veículos da empresa Tesla estão sendo equipados e como os seus sensores realizam a leitura em torno do veículo, pode-se observar que os sensores ultrassônicos são utilizados para distâncias menores e com maior angulação enquanto que o radar é utilizado para distâncias longínquas e perceptível, com uma menor angulação. Pode-se perceber também que a empresa utiliza muitas câmeras ao redor de todo o veículo para poder fazer o sensoriamento e acaba por não usar o LIDAR por exemplo, talvez pelo fato de ser uma tecnologia ainda muito cara para ser utilizada em fins comercias.

Figura 8 - Ilustração dos sensores aplicados em um VA



Fonte: Tesla

2.5 FUNCIONALIDADES PRESENTES EM VEÍCULOS AUTÔNOMOS

Para executar de maneira segura e ótima as complexas funcionalidades, VAs necessitam que vários sensores e sistemas, funcionem de forma integrada se

comunicando internamente e trocando as informações coletadas do ambiente. Para isso existem hoje algumas funcionalidades ADAS e muitas ainda estão por vir, entre elas podemos citar e discorrer das mais comuns:

Controle de Cruzeiro Ativo (ACC - Active Cruise Control), essa funcionalidade tem a capacidade de se manter a uma velocidade pré-determinada, assim como manter uma distância segura de eventuais veículos a frente. Sensores do tipo radares são os mais comumente utilizados para implementar tal funcionalidade.

Sistema de Aviso de Saída de Faixa (LDWS – Lane Departure Warning System) a principal funcionalidade deste sistema é detectar as marcações das faixas de sinalização na rodovia e alertar o motorista para o caso de haver uma saída de faixa não intencional. Se o sistema detectar que a manobra é intencional acionando as setas de direção ou ainda acelerando o veículo o sistema não emite nenhum alerta. Para implementar o sistema há integração de sensores de vídeo, *laser*, e infravermelhos dependendo muito do fabricante.

Assistente de Faixa (LKA – Lane Keep Assist) em alguns casos o LKA pode ser visto como um sistema complementar do LDWS, já que o LKA efetua manobras em caso de o motorista não tomar nenhuma atitude, para prevenir uma mudança não intencional de faixa. Em outros casos o LKA ajuda o carro a se manter no curso da rodovia, próximo ao centro da faixa efetuando manobra na direção evitando saída não intencional da faixa ou ainda a colisão com objetos ao lado do veículo. Para implementar esta tecnologia uma integração entre atuadores e sensores é necessária, pois o veículo precisa efetuar manobras.

Assistente de Estacionamento (PK – Park Assist) há diferentes modelos e tipos de assistentes para estacionamento no mercado, os mais comuns no mercado atualmente são os que emitem algum tipo de alerta para avisar o motorista sobre a manobra que ele esteja efetuando. Os mais recentes, mas ainda não tão comuns são os que de fato podem efetuar a manobra de baliza de maneira autônoma, e estacionar sozinhos ou ainda que requisitem poucas ações do motorista a fim de orientá-lo na manobra. Para implementar estas funcionalidades integra-se atuadores e sensores, por exemplo podem-se usar sensores ultrassônicos para medir a distância do veículo com o objeto ou usando câmeras para que o motorista possa ver a traseira do mesmo.

Freio Automático de Emergência (AEB – Automatic Emergency Braking) é uma funcionalidade que colabora na prevenção de colisão entre veículos, o sistema emite um alerta ao motorista sobre o risco de colisão eminente e o ajuda na tarefa de

frear o carro de maneira ótima, caso a situação seja crítica em alguns casos ele pode ainda frear de maneira independe sem esperar uma resposta do motorista.

Monitoramento do Motorista (DM – Driver Monitoring) o DM é um sistema capaz de monitorar o motorista enquanto ele conduz o veículo usando sensores infravermelhos ou câmeras de vídeo que visam captar e interpretar o nível de atenção do mesmo na condução do carro. Caso o motorista não esteja atento na sua tarefa e uma emergência for detectada então o sistema alerta o motorista seja por meio de luzes piscando ou ainda emitindo avisos sonoros, se ainda assim nenhuma ação for tomada o sistema toma a ação do mesmo efetuando algum tipo de manobra evasiva com o intuito de colocá-lo em uma situação de risco mínimo.

Assistente de Engarrafamento (TJA – Traffic Jam Assist) engarrafamentos são estressantes e aumentam o risco de acidentes ocasionados por colisão traseira, TJA é uma funcionalidade integrada dos sistemas ADAS que ajudam o motorista a conduzir o carro. Com está funcionalidade o carro pode frear, acelerar, manter o veículo a uma distância segura do veículo a frente e ainda manter o mesmo dentro e ao centro da faixa, tudo isso sem a intervenção do motorista. As funcionalidades e como elas são integradas pode variar de um fabricante para o outro, mas de maneira genérica elas performam a mesma atividade.

Piloto Automático (AP – Automatic Pilot) essencialmente o piloto automático é a função de direção autônoma, em que o veículo é capaz de realizar as manobras corretamente e conduzir de maneira segura sem a intervenção humana pela trajetória pré-determinada. Para realizar muitas destas ações e funcionalidades os sistemas precisam estar integrados e se comunicar entre si realizando as operações e as lógicas necessárias para performar os movimentos de maneira ótima e segura.

3 SISTEMAS INTELIGENTES

No presente capítulo são abordados conceitos referentes aos Sistemas Inteligentes. Aprofunda-se no conceito de sistemas inteligentes, inteligência artificial e inteligência computacional que são encontradas nos livros referência sobre o assunto. Posteriormente a explanação das categorias de sistemas existentes é então aprofundado o assunto referente aos algoritmos de buscas e suas vertentes chegando ao algoritmo A*, que é o principal algoritmo utilizado na elaboração do trabalho.

3.1 DEFINIÇÃO

Sistemas inteligentes podem ser definidos como sistemas que incorporam inteligência em aplicações gerenciadas por máquinas, capazes de realizar pesquisa e otimização em conjunto com habilidades de aprendizado e também podem executar complexas tarefas de automação que não seriam executadas por meio da computação tradicional. São sistemas que se baseiam em métodos e técnicas do campo da Inteligência Artificial (IA) para executar operações eficientes e precisas na resolução dos problemas (MANKAD, 2014; KÖSE, ARSLAN, 2014).

Atualmente, a IA abrange uma variedade de campos diferentes e é potencialmente relevante para qualquer esfera da atividade intelectual, seguindo esse preceito não é difícil encontrar várias definições sobre o que vem a ser a IA. Russel e Norvig (2009) usando oito livros didáticos definiram que as definições variam em duas grandes dimensões, como ilustrado na Figura 9. As que estão na parte superior da tabela de relacionam a processos de pensamento e raciocínio, enquanto as definições da parte inferior se referem ao desempenho humano. As definições do lado direito medem o sucesso comparando-o a um conceito ideal de inteligência que Russel e Norvig (2009) chamam de racionalidade, segundo os autores um sistema é racional se “faz tudo certo”, com os dados que tem.

Segundo os autores existe historicamente uma tensão entre as abordagens centradas em torno de seres humanos e abordagens centradas em torno da racionalidade. “Uma abordagem centrada nos seres humanos deve ser uma ciência empírica, envolvendo hipóteses e confirmação experimental. Uma abordagem

racionalista envolve uma combinação de matemática e engenharia". (RUSSEL, NORVIG, 2009).

Figura 9 - Definições de inteligência artificial, em quatro categorias.

Sistemas que pensam como seres humanos	Sistemas que pensam racionalmente
"O novo e interessante esforço para fazer os computadores pensarem... Máquinas com mentes, no sentido total e literal." (Haugeland, 1985)	"O estudo das faculdades mentais pelo uso de modelos computacionais." (Charniak e McDermott, 1985)
"[Automatização de] atividades que associamos ao pensamento humano, atividades como tomada de decisões, a resolução de problemas, o aprendizado..." (Bellman, 1978)	"O estudo das computações que tornam possível perceber, racionar e agir." (Winston, 1992)
Sistemas que atuam como seres humanos	**Sistemas que atuam racionalmente**
"A arte de criar máquinas que executam funções que exigem inteligência quando executadas por pessoas." (Kurzweil, 1990)	"A Inteligência Computacional é o estudo do projeto de agentes inteligentes." (Poole *et al*., 1998)

Fonte: (Russel e Norvig, 2009) - adaptado

Seguindo a linha de sistemas que atuam racionalmente sugerida anteriormente chegamos ao campo da Inteligência Computacional (IC) que segundo a definição de (POOLE et al., 1998) "é o estudo do projeto de agentes inteligentes". Agente é algo que age, mas para distingui-lo de um programa normal ele deve ter a capacidade de operar sob controle autônomo, perceber o ambiente a sua volta, se adaptar as mudanças e ser capaz de assumir metas, em resumo um "agente racional é aquele que age para alcançar o melhor resultado, ou quando há incerteza, o melhor resultado esperado" (RUSSEL, NORVIG, 2009).

A IC busca desenvolver sistemas inteligentes por meio de técnicas inspiradas na natureza, visando imitar aspectos e comportamentos "inteligentes" de qualquer forma de vida ou de sistemas complexos, nesse contexto a inteligência está intimamente ligada à tomada de decisão e ao raciocínio.

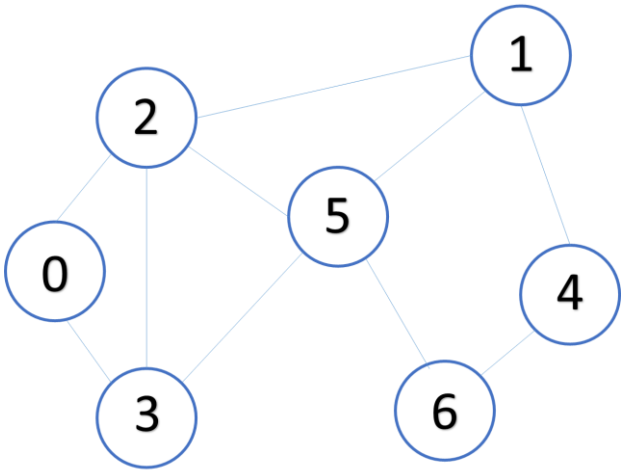
3.2 ESTRATÉGIAS E ALGORITMOS DE BUSCA

Em problemas de busca o espaço de busca e as suas propriedades são definidos pela técnica utilizada para a codificação da sua solução, por exemplo, em

espaços discretos deve-se percorrer um grafo em busca de um nó, já em espaços contínuos deve-se varrer o espaço em busca de um ponto (RUSSEL, NORVIG, 2009).

Um grafo é uma estrutura matemática usada para representar as relações entre objetos, eventos, ocorrências, etc. É essencialmente um conjunto finito de pontos (vértices) e um conjunto finito de arestas, e cada uma destas arestas conecta dois vértices (não necessariamente distintos), como ilustrado na Figura 10.

Figura 10 - Representação da estrutura de um grafo.



Fonte: Autoria própria

Existem algumas categorias diferentes de buscas que visam solucionar problemas ainda mais específicos, algumas serão citadas visando tornar-se conhecidas ao leitor, mas não serão explanadas a fundo, pois não são atribuídas ao caso do algoritmo desenvolvido no presente trabalho, são os casos das buscas populacionais e não populacionais; buscas com decisões determinísticas ou estocásticas; com ou sem garantia de convergência para uma solução ótima; com ou sem busca local. Para qualquer metodologia utilizada é desejável estruturá-las de modo a evitar ciclos.

A busca não-informada é também conhecida como busca cega, ela se utiliza somente da definição do problema a ser solucionado. Dentro da categoria da busca cega podemos ter diferentes algoritmos como; a busca em largura que expande o nó mais próximo da raiz que não tenha sido expandido; a busca com custo uniforme, que expande o nó de menor custo ainda não expandido; a busca em profundidade que expande o nó mais profundo que não tenha sido expandido, entre outras. Pode-se perceber que ao definir o problema a ser solucionado por meio de busca cega, o nível de detalhes na definição do problema deve ser extremamente elevado pois caso

contrário a solução pelos algoritmos citados torna-se inviável (RUSSEL, NORVIG, 2009).

Já a busca informada se utiliza de informações adicionais além de somente a definição do problema como na busca cega. A busca informada também é conhecida como busca heurística (A palavra heurística vem do grego e significa “descobrir”. Tem a ver com o uso do “senso comum” na solução de problemas), elas são parecidas com a busca em amplitude, tirando o fato de que a busca não se expande de forma uniforme a partir do nó-raiz, mas sim, obedecendo heurísticas diferentes para cada problema, ou seja, com informações distintas e específicas de cada problema, é possível definir uma ordem de preferência entre os ramos possíveis a serem expandidos, logo com mais informações a metodologia se torna mais eficiente (RUSSEL, NORVIG, 2009).

A melhor rota até a solução é obtida pela adoção de decisões do tipo *best-first* a cada nó expandido, para isso, estimar e calcular o custo de chegar em um nó e o quão distante ele está da solução é de suma importância para o desempenho do algoritmo. Uma heurística não garante a solução ótima, no entanto, permite em geral uma grande redução de custo e tempo. Heurísticas “bem-formadas” fornecem soluções que estão muito próximas da solução ótima. A decisão para que nó deve-se seguir é determinada por uma função heurística (RUSSEL, NORVIG, 2009).

Funções heurísticas são especificas para cada problema, elas estimam custo do caminho mais barato do estado atual para o estado final. Matematicamente a função heurística *h(n)* é o custo estimado do caminho mais econômico do nó *n* até o nó objetivo. Se *n* é o objetivo, então *h(n) = 0* (RUSSEL, NORVIG, 2009).

3.3 O ALGORITMO A*

Um dos algoritmos *best-first* mais conhecidos é chamado de A* (lê-se “A-estrela”) esse algoritmo é a combinação de aproximações heurísticas, como do algoritmo de busca em largura e do algoritmo de Dijkstra (1959).

O algoritmo de Dijkstra foi concebido pelo holandês Edsger Dijkstra em 1959, basicamente este algoritmo visava solucionar os problemas de caminho mais curto. A principal diferença entre ele e o algoritmo A* é a ausência de uma função heurística que facilite e diminua o número de nós expandidos, pois a cada passo o algoritmo de

Dijkstra verificaria os nós adjacentes para efetuar a avaliação, sem se importar com uma ordem ou priorização dos ramos, o que acontece no algoritmo A* devido a utilização da função heurística determinada pelo problema.

O algoritmo A* Foi descrito primeiramente por Hart, Nilsson e Raphael (1968) mas hoje em dia já existem muitas outras variantes do algoritmo proposto originalmente. Ele avalia os nós através da combinação de *g(n)* que é o custo para alcançar cada nó com a função *h(n)* que é o menor custo partindo da origem para se chegar ao destino, matematicamente dado na equação 1:

$$f(n) = g(n) + h(n) \quad (1)$$

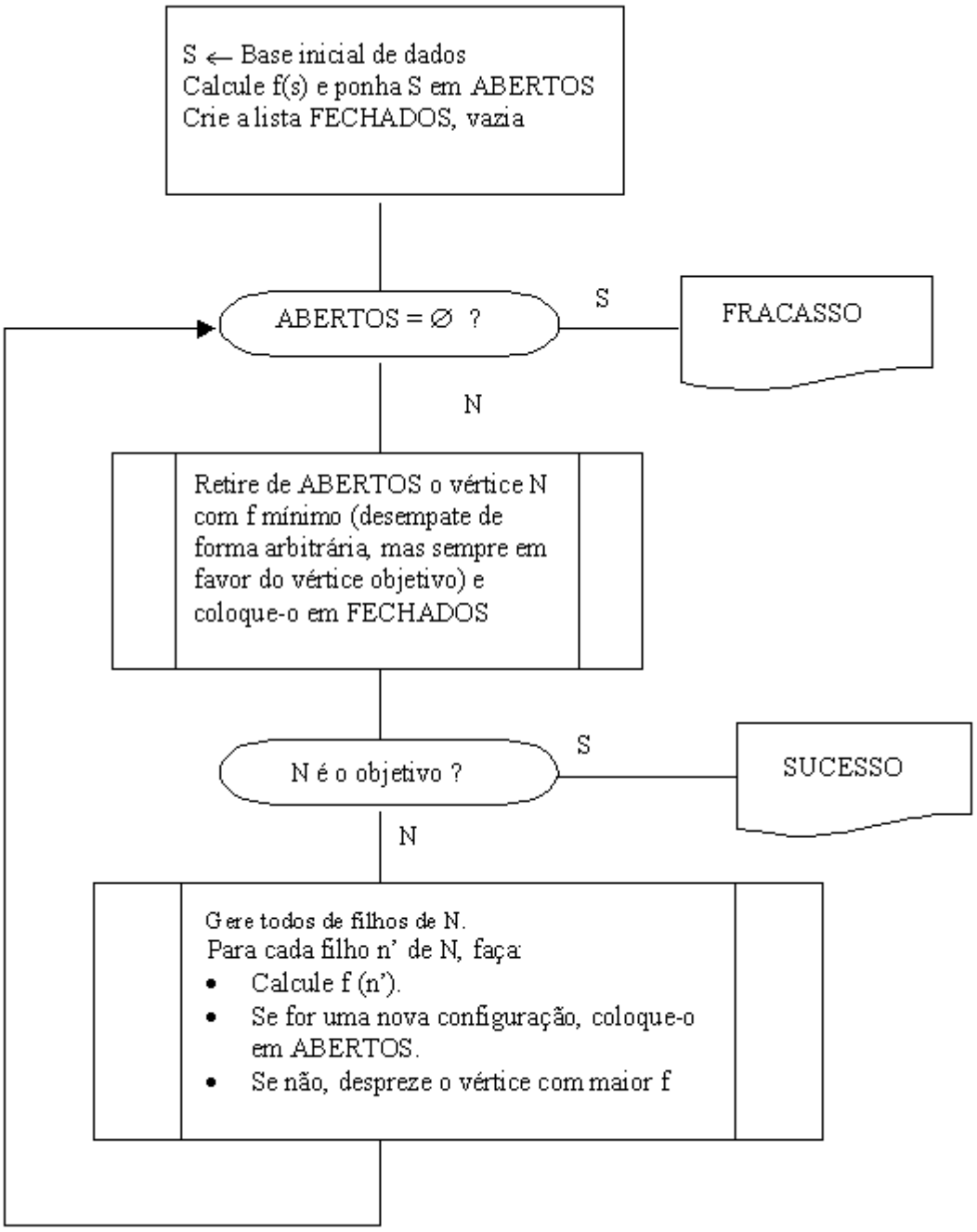
Desse modo f(n) é, portanto, o custo estimado da solução de custo mais baixo passando por n. Esta técnica requer que a estimação do custo restante no próximo nó não seja nunca maior que o custo restante do nó anterior. Diferente das duas técnicas anteriores, sob esta hipótese, sempre é possível encontrar a solução ótima com a busca A*.

Na Figura 11 é ilustrado um fluxograma do algoritmo A*, em que são ilustrados as tomadas de decisões e os passos para o desenvolvimento do algoritmo. O algoritmo se inicia com a declaração das variáveis e dos vetores, em que S é a posição de partida do agente. *ABERTOS* é o nome da lista de nós ou posições que já foram abertas. *FECHADOS* é o nome da lista de nós ou posições que já foram fechadas, ou seja, já foram avaliadas e ordenadas. N é a posição ou o nó atual do agente, é com ele que se determina se o objetivo foi alcançado ou se o algoritmo precisa expandir mais nós e avalia-los.

Os algoritmos de busca podem ser avaliados sob quatro aspectos; Completeza, se ele encontra a solução se ela existir; Otimização, se ele encontra a solução de menor custo; Tempo, o tempo que ele leva para encontrar a solução e Espaço, que é a quantidade de memória consumida para executar a busca. (RUSSEL, NORVIG, 2009).

A busca A* é completa, ótima e eficiente (HART et al., 1968). É completa a não ser que exista uma quantidade infinita de nós. Ótima, pois, nenhum outro algoritmo tem garantia de expandir um número de nós menor que o A*. Isso porque qualquer algoritmo que não expande todos os nós com $f(n) < C^*$ corre o risco de omitir uma solução ótima. Sua complexidade ainda é exponencial e o seu uso de memória é intenso.

Figura 11 - Fluxograma do algoritmo A*

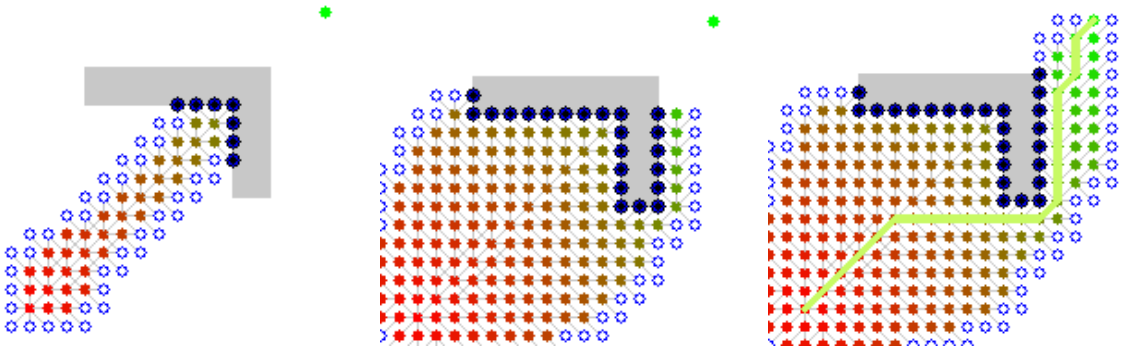


Fonte: Adaptado de Hart, Nilsson e Raphael (1968)

A busca A* é completa, ótima e eficiente (HART et al., 1968). É completa a não ser que exista uma quantidade infinita de nós. Ótima, pois, nenhum outro algoritmo tem garantia de expandir um número de nós menor que o A*. Isso porque qualquer algoritmo que não expande todos os nós com f(n) < C* corre o risco de omitir uma solução ótima. Sua complexidade ainda é exponencial e o seu uso de memória é intenso.

Na Figura 12 observa-se como os nós são expandidos até ser encontrado trajeto ótimo entre o ponto inicial e o ponto final do exemplo em questão.

Figura 12 - Exemplo da expansão de nós de um algoritmo A*.

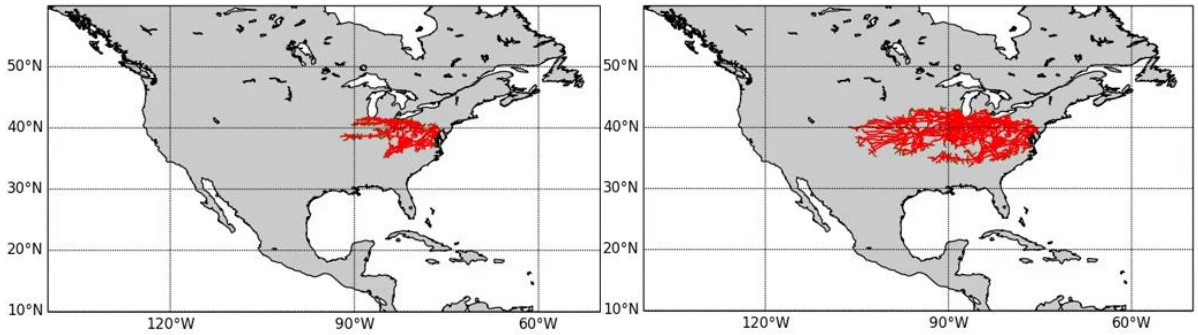


Fonte: (Bhattacharya, 2011)
<https://commons.wikimedia.org/wiki/File:Astar_progress_animation.gif>

3.4 EXEMPLO DE USO DE CASOS DO ALGORITMO A*

As aplicações, a complexidade e o uso do algoritmo de busca A*, varia de acordo com a sua finalidade, mas ele busca visa ser enxuto e simples, de maneira que possa ser implementado nos mais diversos tipos de aplicações. Na sequência de imagens (Figura 13 e Figura 14) é possível ver como o algoritmo A* evolui e como ele resolve o problema de busca de rotas de linhas ferroviárias para ir de *Washington* a *Los Angeles* nos Estados Unidos. Na Figura 13 nos atentamos para como o algoritmo inicia o roteamento, checando os nós iniciais possíveis.

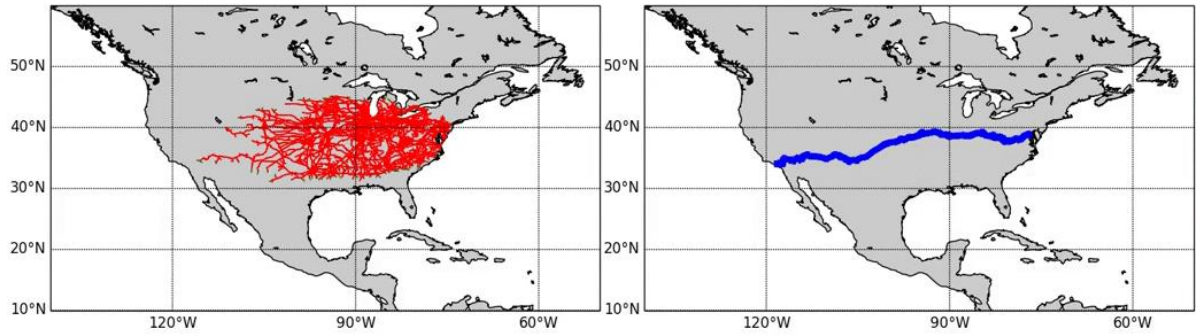
Figura 13 - Um exemplo de aplicação real do algoritmo A* (a).



Fonte: (Peterson, 2014)
<https://commons.wikimedia.org/wiki/File:A*_Search_Example_on_North_American_Freight_Train_Network.gif>

Na Figura 14 o algoritmo já encontrou a rota ótima, após expandir um número incontável de nós por todo o pais e averiguar que a rota exibida em azul é o melhor caminho.

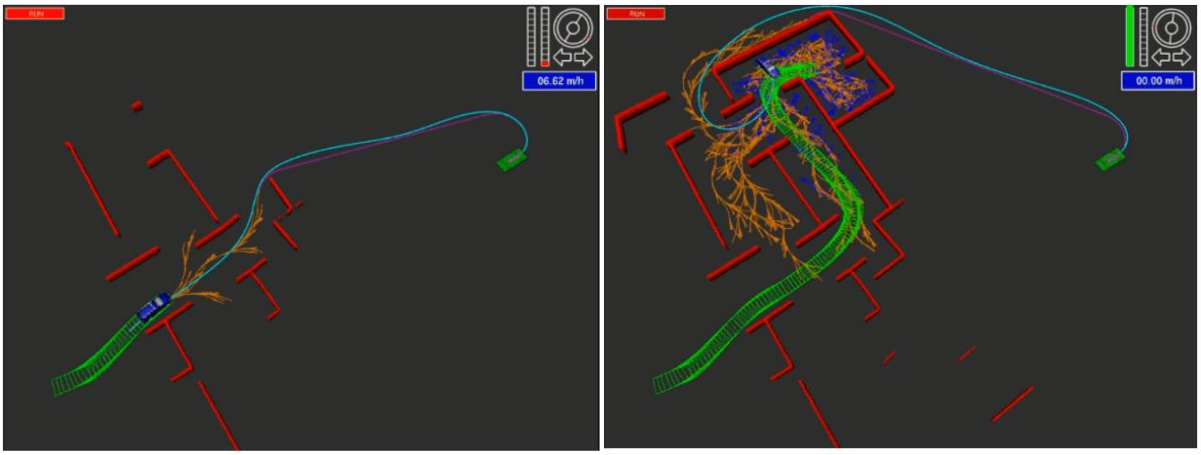
Figura 14 - Um exemplo de aplicação real do algoritmo A* (b).



Fonte: (Peterson, 2014)
<https://commons.wikimedia.org/wiki/File:A*_Search_Example_on_North_American_Freight_Train_Network.gif>

Já na Figura 15 têm-se o algoritmo em ação em um ambiente simulado, onde temos um agente (mais especificamente um carro autônomo), equipado com “sensores” para poder detectar os obstáculos (ou as paredes do labirinto), o objetivo deste veículo é poder chegar até o ponto final ilustrado em verde nas imagens.

Figura 15 - Algoritmo A* em ambiente simulado (a)

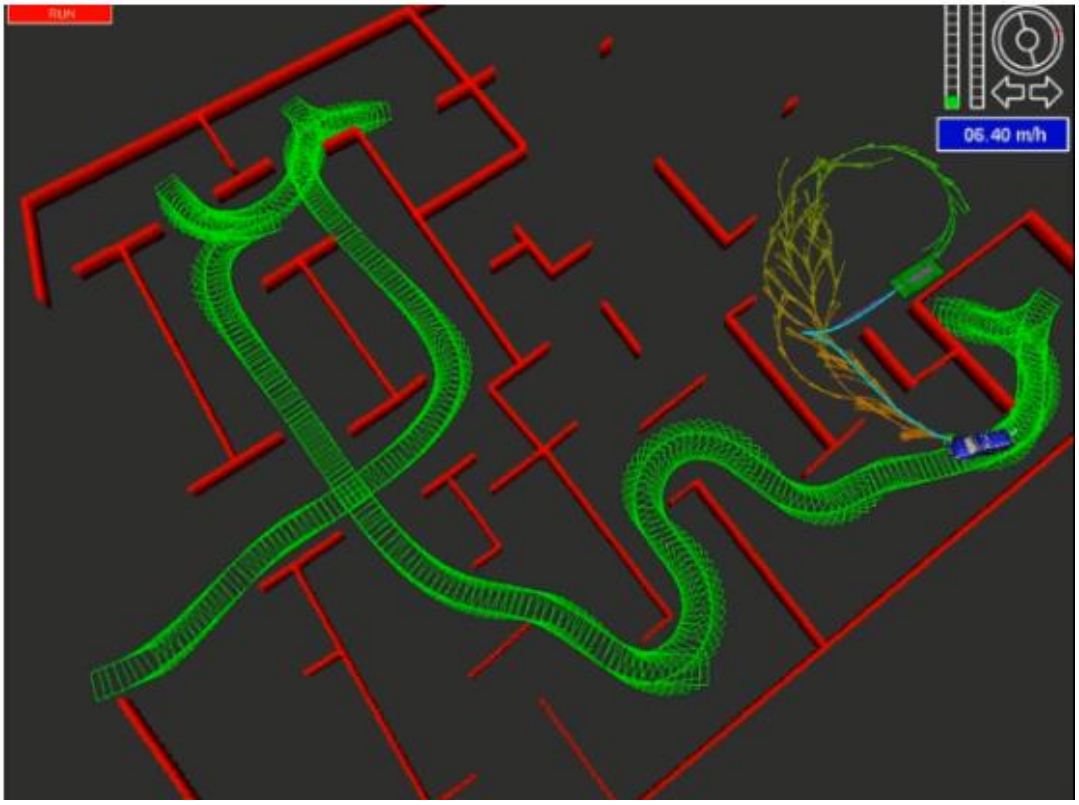


Fonte: Udacity, 2012 (Curso aberto de Inteligência Artificial para Robótica)

Segundo Thrun a heurística utilizada para detectar o menor caminho foi a distância euclidiana entre a posição atual do veículo e o ponto final de chegada. Uma rota em azul é construída pelo algoritmo, mas esta rota se altera à medida que o veículo navega pelo ambiente e acaba por encontrar os obstáculos, precisando desta forma realizar manobras de retorno afim de continuar a explorar o labirinto e encontrar o seu ponto final.

Na a Figura 16 É possível ainda ver em verde o caminho já percorrido e todas as mudanças e empecilhos que foram responsáveis por essa mudança de trajetória, em laranja a arvore de “nós” expandidos pelo algoritmo enquanto ele busca o melhor caminho.

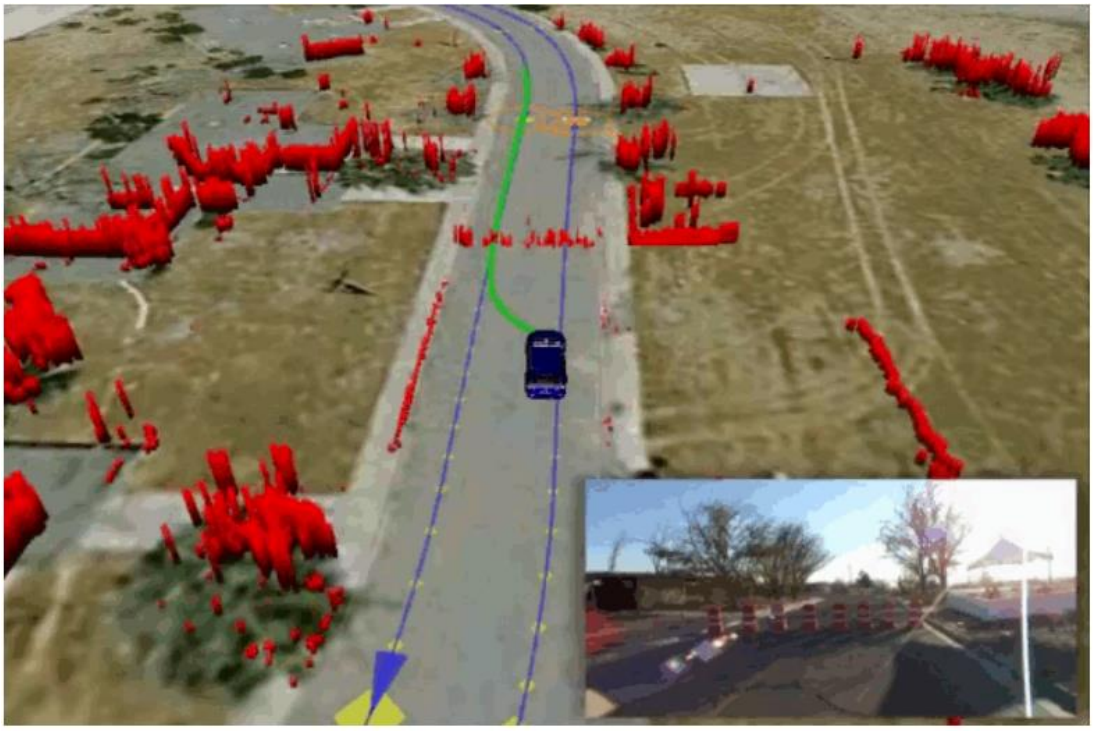
Figura 16 - Algoritmo A* em ambiente simulado (c)



Fonte: Udacity, 2012 (Curso aberto de Inteligência Artificial para Robótica)

Um exemplo da aplicação de um carro autônomo em ambiente real é ilustrado na Figura 17. Neste cenário o carro navega pela via e acaba por encontrá-la bloqueada. Estudando que não haveria como transpassar o bloqueio, o veículo então decide realizar uma manobra em “U” retornando pelo caminho anterior de onde havia vindo, para então poder calcular e seguir em outro caminho que o leve até o seu destino. Segundo Thrun o algoritmo A* estava em ação nesta situação, guiando o veículo para realizar a busca pela trajetória ótima.

Figura 17 - Exemplo de aplicação do algoritmo A* em um carro autônomo



Fonte: Udacity, 2012 (Curso aberto de Inteligência Artificial para Robótica)

4 O ALGORTIMO A* IMPLEMENTADO

No presente capítulo é explicado a motivação por utilizar a linguagem *Python* na implementação do algoritmo, e também a explanação do problema base a ser solucionado pelo mesmo. Será explicado em detalhes os passos utilizados no desenvolvimento do código assim como a explanação dos resultados obtidos na resposta do algoritmo.

4.1 LINGUAGEM UTILIZADA

A linguagem escolhida para a implementação do algoritmo é *Python*, ela tem ganho popularidade, baseada nos dados do índice *Tiobe* que todo ano aponta as linguagens mais usadas no mundo. O índice *Tiobe* é uma lista de linguagens de programação, classificada pela frequência de pesquisa na web usando o nome da linguagem como a palavra-chave. De acordo com a classificação fornecida, Python é a 4ª linguagem mais usada em 2018 ultrapassando C#, PHP e até Javascript, ficando atrás somente de Java, C e C++. Vale ressaltar que a linguagem vem crescendo e subindo no ranking a cada ano, por exemplo no ano de 1998 a linguagem ficava somente em 24º lugar. (TIOBE, 2018).

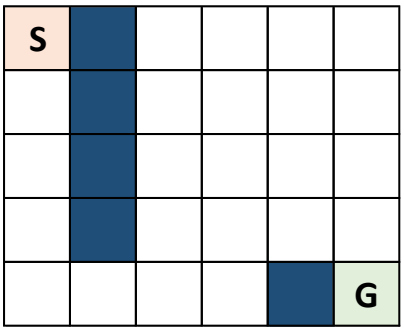
Python vem ganhando espaço e adeptos também por causa dos avanços da IA, já que ela é utilizada em grande escala por desenvolvedores de algoritmos inteligentes, como; *Tensor Flow,* que é uma biblioteca de aprendizado de máquina disponibilizada pela Google; *Blender,* uma plataforma de desenvolvimento para animações e jogos; *NumPy* e *Scikit-Learn* que são bibliotecas utilizadas para aprendizado de máquina; Além de automatização de tarefas e no processamento de dados coletados da rede mundial de computadores. Além dos já citados motivos, a simplicidade de implementação da linguagem também foi um dos fatores determinante para a sua escolha.

4.2 O PROBLEMA

Para desenvolver o algoritmo tomou-se como base uma problemática simples; o agente inteligente, deve planejar uma rota em um labirinto bidimensional em forma

de *grid* (como o ilustrado na Figura 18), navegando entre um ponto de partida (S) até um ponto de chegada (G), definindo assim a sequência de movimentos a serem efetuados.

Figura 18 - Exemplo de labirinto



Fonte: Autoria própria

Tomando como base o uso do algoritmo, buscou-se implementa-lo de maneira menos rebuscada, excluindo-se das variáveis dinâmicas e não discretas que é encontrado no mundo real e retendo-se aos movimentos básicos para a circulação do agente pelo labirinto, (para a direita, para a esquerda, para cima e para baixo) dessa maneira é possível entender de forma simples como um algoritmo que pode ser usado em situações ainda mais complexas pode ser implementado. No problema em questão o algoritmo é exposto a diferentes tipos de labirintos em que é possível avaliar o seu desempenho tomando-se como base os nós expandidos e os movimentos realizados pelo agente.

4.3 IMPLEMENTAÇÃO

No presente subcapitulo será explicado como foi elaborado e implementado o algoritmo. Para implementar a busca A*, foi utilizado um labirinto base para que pudessem ser efetuados os testes e o próprio desenvolvimento do algoritmo, sendo assim foi criado então uma variável de nome *grid,* contendo a informação das barreiras e o tamanho do próprio “labirinto”. Os 0’s significam espaços vazios, em que o agente pode circular, já os 1’s significam obstáculos que o agente, portanto deve desviar, como é ilustrado pelo Código 1.

O Código 2, ilustra os pontos de partida de chegada setados para o exemplo elaborado, e também o custo para efetuar cada movimento, sendo eles respectivamente as variáveis *init, goal* e *cost*.

Código 1 - grid

```
1   grid = [[0, 1, 0, 0, 0, 0],
2           [0, 1, 0, 0, 0, 0],
3           [0, 1, 0, 0, 0, 0],
4           [0, 1, 0, 0, 0, 0],
5           [0, 0, 0, 0, 1, 0]]
```

Código 2 - *init, goal e cost*

```
14  init = [0, 0]                                # ponto de partida do agente
15  goal = [len(grid)-1, len(grid[0])-1]         # ponto de chegada do agente
16  cost = 1                                     # custo para cada movimento
```

Na Código 3, está a função heurística nomeada de *heuristic* no código do algoritmo, que para este caso elaborado é a distância ou o número de casas distantes do ponto de chegada estabelecido.

Código 3 - *heuristic*

```
8   heuristic = [[9, 8, 7, 6, 5, 4],
9               [8, 7, 6, 5, 4, 3],
10              [7, 6, 5, 4, 3, 2],
11              [6, 5, 4, 3, 2, 1],
12              [5, 4, 3, 2, 1, 0]]
```

Foi criado também uma variável chamada *delta* que nada mais é do que a tradução dos movimentos permitidos pelo agente, como se mover para cima, baixo, direita e esquerda. Para traduzir essa informação é usado do princípio de vetores e matrizes, de onde se obtém a localização relativa do agente no “mapa” criado, pois por exemplo se o agente estiver na posição de 0,0 (linha 1 e coluna 1) e ele se movimentar para baixo, sua nova posição seria de 1,0 (pois é acrescido 1 da posição em linha). No Código 4 está ilustrada a variável *delta* e *delta_name*, em que *delta _name* é a representação “simbólica” do movimento literal traduzido de *delta.*

Código 4 - *delta e delta_name*

```
18  delta = [[-1, 0 ],  # para cima
19           [ 0, -1],  # para a esquerda
20           [ 1, 0 ],  # para baixo
21           [ 0, 1 ]]  # para a direita
22
23  delta_name = ['^', '<', 'v', '>']
```

O próximo passo é definir dentro da função que será chamada (de nome *search*), os vetores (com o código ilustrado no Código 5); *closed* é uma matriz que contém a informação dos nós ou posições que foram expandidos ou não. A matriz *expand* contém a ordem crescente com que os nós foram expandidos. A matriz *action* é a variável que irá receber os movimentos realizados até se chegar ao ponto de chegada. E finalmente a matriz *path* que irá exibir a trajetória efetivamente ótima encontrada pelo algoritmo.

Código 5 - *closed, expand, action* e *path*

```
27      closed = [[0 for row in range(len(grid[0]))] for col in range(len(grid))]
28      closed[init[0]][init[1]] = 1
29
30      expand = [[-1 for row in range(len(grid[0]))] for col in range(len(grid))]
31      action = [[-1 for row in range(len(grid[0]))] for col in range(len(grid))]
32
33      path = [[' ' for row in range(len(grid[0]))] for col in range(len(grid))]
```

Para setar e salvar a posição do agente são utilizadas as variáveis *x* e *y,* como pode ser visto em Código 6, em que são inseridos os valores iniciais das variáveis. Podemos ver também que *g* e *f* são declarados e inicializados e usando a informação de todos estes “parâmetros” cria-se um vetor, neste caso o chamaremos de *open.*

Código 6 - *x, y, g, f* e *open*

```
35      x = init[0]             # seta a posicao inicial do agente em x
36      y = init[1]             # seta a posicao inicial do agente em y
37      g = 0                   # seta o valor inicial da funcao g
38      f = g + heuristic[x][y] # seta o valor inicial de f
39
40      open = [[f, g, x, y]]   # cria o vetor para guardar as componentes
```

Para rodar o algoritmo precisamos também criar algumas *flags*, que terão a função de “monitorar” se a busca chegou ao fim, e ainda se há nós a serem expandidos, mesmo que não se chegue ao ponto final, *found* e *resign* são respectivamente as variáveis ou *flags* que realizarão esta função. Para manter a contagem de nós expandidos e ainda atualizar a matriz *expand* será usado a variável com nome de *count,* como é possível observar em Código 7.

Para realizar a checagem se a busca já está completa ou se ainda há nós a serem expandidos é preciso inserir as flags em um loop (ilustrado em Código 8), neste loop o código se repete até que uma das flags seja alterada, fazendo assim com que

o código saia de dentro do mesmo. Logo após “entrar” no loop o código começa por executar uma verificação, em que checa se o tamanho do vetor de open é equivalente a zero, caso isso ocorra então conclui-se que não há mais nós a serem expandidos e não foi encontrado o ponto de chegada, fazendo assim com que uma mensagem de “*faill*” seja retornada na função chamada pelo programa.

Código 7 - *found, resign* e *count*

```
42      found = False   # flag que e setada quando a busca esta completa
43      resign = False  # flag que e setada caso nao tenha mais no a expandir
44      count = 0       # variavel de contagem de nos expandidos
```

Código 8 - Loop de checagem se ponto final

```
46      while not found and not resign: # loop para caso a busca nao
47                                      #  esteja completa e ainda tenha
48                                      #  nos para expandir
49
50          if len(open) == 0:  # checa o vetor open
51              resign = True   # seta a flag resign
52              return 'fail'   # retorna mensagem de 'fail'
```

Caso ainda tenham nós a serem expandidos e o agente não tenha encontrado o ponto de chegada o programa continua, e o próximo passo a ser realizado é ordenar de forma crescente os nós já expandidos e avaliar qual deles é o próximo ponto de localização para o agente. É preciso também lembrar de retirar da lista de nós o local escolhido com o melhor resultado em *f* (em Código 9). Após isso o nó expandido deve receber seu número respectivo na contagem progressiva de nós expandidos e a variável *count* deve ser acrescida de 1, para então continuar sua contagem.

Após atualizar a posição e os seus parâmetros, é preciso verificar se o ponto de chegada foi atingido (em Código 10), caso tenha se chegado no ponto final então a flag *found* é setada como sendo verdadeira.

Se ainda não foi atingido o objetivo então o código continua a ser executado, entrando em um loop em formato de *for*, em que será verificado os movimentos possíveis a serem feitos, ou seja, esquerda, direita, cima e baixo. Neste momento varre-se as 4 possíveis movimentações, verifica-se se elas são plausíveis, como por exemplo checando se ao movimentar o agente ele não “sairá” do labirinto inicialmente configurado para ele se mover (em Código 11).

Código 9 - Caso não tenha encontrado o ponto objetivo

```
54          else:               # caso nao tenha erro no vetor, continua a rodar
55
56              open.sort()         # ordena o vetor open
57              open.reverse()      # inverte a ordenacao
58              next = open.pop()   # next recebe o valor extraido de open
59              f = next[0]         # f recebe valor de next
60              g = next[1]         # g recebe valor de next
61              x = next[2]         # x recebe valor de next
62              y = next[3]         # y recebe valor de next
63
64              expand[x][y] = count    # o no expandido recebe sua contagem
65              count += 1              # e acresido em 1 o valor de nos expandidos
```

Código 10 - Caso o objetivo seja atingido

```
67              if x == goal[0] and y == goal[1]:   # condicao para caso o
68                                                  # objetivo seja atingido
69                  found = True    # seta a flag found
```

Código 11 - Caso ainda não tenha encontrado o objetivo

```
71              else:   # caso ainda nao tenha encontrado o objetivo, continuar
72
73                  for i in range(len(delta)): # loop para "movimentar" o agente
74                      x2 = x + delta[i][0]    # x2 recebe o valor de x adicionado
75                                               # do movimento especifico
76                      y2 = y + delta[i][1]    # y2 recebe o valor de x adicionado
77                                               # do movimento especifico
78
79                      if x2>=0 and x2<len(grid) and y2>=0 and y2<len(grid[0]):
80                          # condicao para identificar se a proxima posicao
81                          # esta dentro do grid
```

Após verificar que o movimento não burlará as fronteiras do grid, ainda é necessário averiguar se a possível futura localização já não foi expandida ou se ainda não é um obstáculo. Caso não seja nenhum desses casos e a ação seja válida então deve-se atualizar o custo dessa movimentação, atualizando então a função *f* e a função *g*, deve-se ainda adicionar essa nova posição no vetor *open*, marcar a posição como um nó já expandido e também salvar a ação realizada para se chegar no mesmo, como ilustrado em Código 12.

Chegando ao final do código ainda temos que reconstruir a trajetória ótima encontrada pelo algoritmo, para fazer isso setamos o agente para a posição inicial, fazendo com que *x* e *y* sejam equivalentes a 0. O próximo passo deve ser um loop que “percorra” as posições e refaça as ações realizadas pelo agente e que ainda salve este trajeto em um vetor, que neste caso é a matriz chamada de *path*.

Código 12 - Realiza checagem e salva componentes

```
                    if closed[x2][y2] == 0 and grid[x2][y2] == 0:
                        # condicao para verificar se o no relativo
                        # a posicao ainda nao foi expandido

                        g2 = g + cost                  # g recebe o g atual somado do
                                                        # custo da operacao
                        f2 = g2 + heuristic[x2][y2]    # g recebe o g atual somado do
                                                        # custo da operacao
                        open.append([f2, g2, x2, y2]) # adiciona mais uma posicao no vetor,
                                                        # com os novos valores calculados
                        closed[x2][y2] = 1  # sinaliza que o no ja foi expandido
                        action[x][y] = i    # sinaliza qual foi a acao tomada na posicao
```

Código 13 - Finalização, reconstrução do caminho ótimo

```
    x = 0    # seta a posicao x para 0
    y = 0    # seta a posicao y para 0

    while x!=goal[0] or y!=goal[1]: # loop para construir o caminho
                                    # percorrido com as acaes salvas
        x2 = x + delta[action[x][y]][0] # x2 recebe posicao + movimento
        y2 = y + delta[action[x][y]][1] # y2 recebe posicao + movimento
        path[x][y] = delta_name[action[x][y]] # variavel path salva as acoes
        x = x2 # x recebe x2
        y = y2 # y recebe y2

    path[goal[0]][goal[1]] = '*' # no ponto de chegada inserir caractere '*'

    return expand # retorna o caminho otimo
```

Ao ser executado este código com o labirinto dado, têm-se a matriz contendo o trajeto realizado e também uma matriz que exibirá os nós expandidos, desta forma pode-se realizar a análise de performance do mesmo. A Figura 19 ilustra como é a matriz resultado *path* na saída do algoritmo, onde temos o caminho que o algoritmo escolheu como sendo ótimo (ilustrado pelas setas).

Figura 19 - Saída do código com o caminho encontrado

```
[['v', ' ', ' ', ' ', ' ', ' '],
 ['v', ' ', ' ', ' ', ' ', ' '],
 ['v', ' ', ' ', ' ', ' ', ' '],
 ['v', ' ', ' ', '>', '>', 'v'],
 ['>', '>', '>', '^', ' ', '*']]
```

Fonte: Autoria Própria

5 RESULTADOS E DISCUSSÕES

Para avaliar a performance e comprovar a eficácia do algoritmo em termos de encontrar a trajetória ótima, pôde-se comparar as respostas encontradas pelo algoritmo A* em relação as respostas encontradas pelo algoritmo de Dijkstra.

As situações exemplos ou os “labirintos” usados são básicos, mas conseguem exemplificar de forma fiel o resultado encontrado para problemas mais complexos. Na Figura 20 é ilustrado a matriz de saída, com a resposta dos nós expandidos pelo algoritmo A* para um grid exemplo, observamos que ao total foram expandidos 12 nós até que o algoritmo encontrasse o trajeto ideal.

Figura 20 - Nós expandidos com A* no labirinto 1

0					
1					
2					
3		8	9	10	11
4	5	6	7		12

Fonte: Autoria Própria

Já na Figura 21 observamos o trajeto ótimo encontrado pelo algoritmo, com as “setas” indicando o sentido a ser seguido na próxima ação executada, até se encontrar o ponto final, representado pelo “ * ”, nesta implementação.

Figura 21 - Trajetória ótima encontrada com A* no labirinto 1

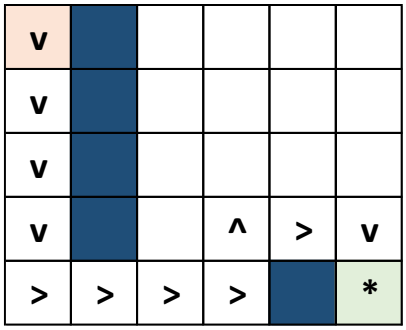
v					
v					
v					
v			^	>	v
>	>	>	>		*

Fonte: Autoria Própria

Nos itens anteriores tivemos as matrizes resultantes para o algoritmo A*, mas a resposta é alterada se analisarmos o algoritmo de Dijkstra. na Figura 22 podemos

ver que o caminho encontrado é o mesmo que o descoberto pelo algoritmo anterior A*.

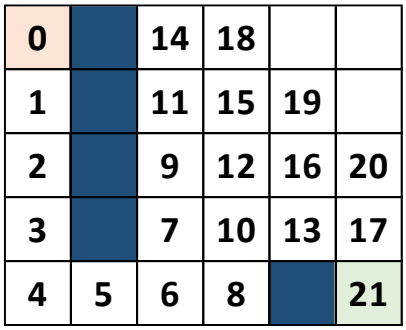
Figura 22 - Rota encontrado pelo algoritmo de Dijkstra no labirinto 1



Fonte: Autoria Própria

Mas ao analisarmos a quantidade de nós que foram expandidos até se encontrar a solução, ilustrada na Figura 23, percebemos que foi maior, ou seja não havia nenhuma regra para o agente seguir e por isso ao expandir os nós ele não consegue identificar se está ou não próximo do ponto desejado, podemos então aferir que neste quesito o algoritmo A* é muito mais eficiente que o seu “antecessor”.

Figura 23 - Nós expandidos para o algoritmo de Dijkstra no labirinto 1

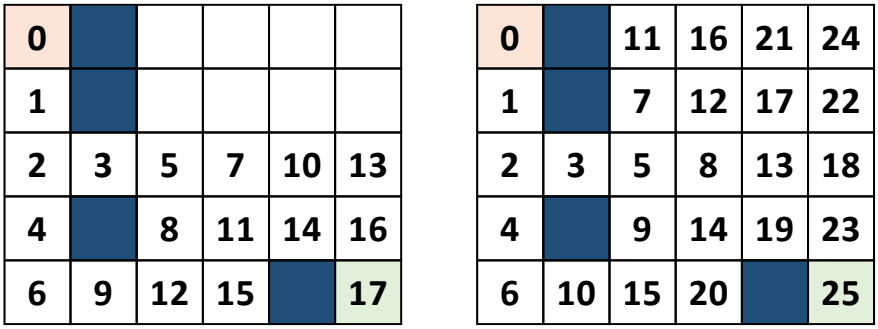


Fonte: Autoria Própria

Analisando outro exemplo ilustrado na Figura 24, pode-se observar que a quantidade de nós expandidas pelo algoritmo A* é significativamente menor que a quantidade de nós expandidos pelo algoritmo de Dijkstra (apresentado na figura pelo labirinto à direita).

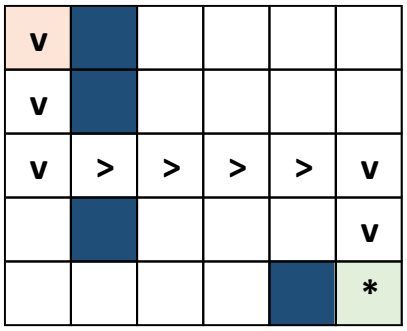
Na Figura 25 temos ilustrados pelas setas o caminho ótimo encontrado pelos algoritmos, podendo observar que os dois encontraram o mesmo caminho, mas como observado anteriormente, o custo para se encontrar o caminho no algoritmo de Dijkstra foi maior devido ao maior número de nós expandidos.

Figura 24 - Nós expandidos respectivamente para o A* e Dijkstra no labirinto 2



Fonte: Autoria Própria

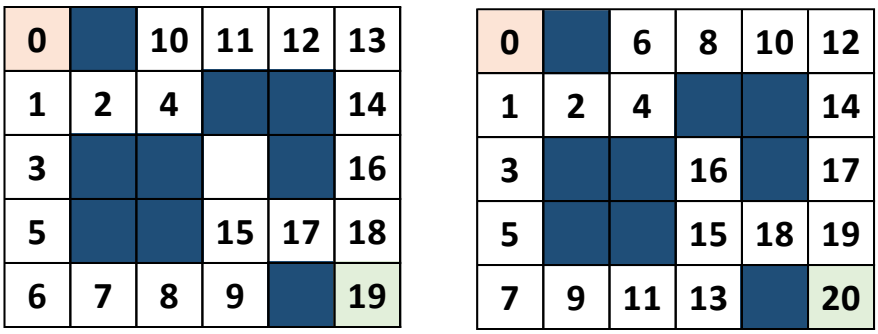
Figura 25 - Caminho ótimo encontrado para o labirinto 2



Fonte: Autoria Própria

Outro exemplo, com um labirinto mais populoso e “complexo” ilustrado na Figura 26, observa-se que o algoritmo A* têm uma pequena vantagem em relação ao de Dijkstra se tratando ao número de nós expandidos, em que foram expandidos 20 nós, contra os 19 abertos pelo A*.

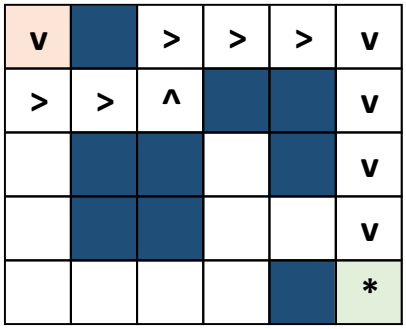
Figura 26 - Nós expandidos respectivamente para o A* e Dijkstra no labirinto 3



Fonte: Autoria Própria

E mesmo com um labirinto mais complexo obtém-se a mesma resposta se tratando da trajetória ótima que o agente percorre para chegar ao ponto de destino, como ilustrado na Figura 27.

Figura 27 - Caminho ótimo encontrado para o labirinto 3



Fonte: Autoria Própria

Levando em consideração que não são necessárias alterações significativas na implementação e codificação do mesmo, e nem que o código fique mais complexo após efetuar as mudanças, é possível afirmar que a aplicação do mesmo é mais indicada, tudo a depender da função heurística utilizada. No caso de veículos autônomos a heurística base utilizada é a distância euclidiana, o que acaba por não necessitar elevado poder computacional fazendo assim o algoritmo A* um grande candidato na navegação de veicular.

6 CONSIDERAÇÕES FINAIS

O desenvolvimento do presente estudo possibilitou uma análise da implementação de um agente inteligente de busca e também de como o desenvolvimento de um algoritmo inteligente pode auxilia-los no planejamento de rotas dos meios de transportes do futuro. É perceptível a relevância técnica dos veículos autônomos nas pesquisas desenvolvidas, pois os sistemas e as tecnologias utilizadas para viabilizá-los podem ser utilizadas para aprimorar diferentes sistemas em outras áreas de estudo.

De modo geral pôde-se perceber que a implementação do algoritmo não é rebuscada ou de alguma forma complexa para o caso exemplificado, mas que em linhas gerais apresenta elevada performance na conclusão dos objetivos. Com a comparação dos resultados dos algoritmos A* e de Dijkstra conseguiu-se perceber a eficácia de se usar de uma função heurística, que traduza o sistema de forma sucinta e fiel, permitindo assim, que os objetivos propostos fossem alcançados.

Dada à importância do assunto, torna-se necessário o desenvolvimento de novas metodologias e novas propostas de abordagem na navegabilidade dos veículos, acompanhando de forma gradual o desenvolvimento das tecnologias e ferramentas já utilizadas. Com o avanço dos sistemas inteligentes e da infraestrutura responsável por suportar os softwares, a realidade de encontrarmos os carros autônomos em operação é palpável, pois são estes sistemas que irão proporcionar o avanço científico da área.

Nesse sentido, é importante ressaltar a possibilidade de que trabalhos futuros possam ser desenvolvidos, visando a implementação do algoritmo em um ambiente simulado, utilizando Unity por exemplo. Para que então as variáveis de movimento, e do próprio ambiente fossem levadas em consideração, de modo a representar a nossa realidade, onde o mundo a nossa volta não é digital, e sim analógico e cheio de pequenas surpresas e detalhes, que precisam ser observados, analisados e então traduzidos para o espaço digital em que as máquinas, e os sistemas eletrônicos funcionam.

REFERÊNCIAS

MOTAVALLI, J. **Self-Driving Cars Will Take Over By 2040.** Forbes Magazine. 25 set. 2012. Disponível em: < https://www.forbes.com/sites/economics/2012/09/25/self-driving-cars-will-take-over-by-2040/#36051e341c92 >. Acesso em: 26 fev. 2018.

WETMORE, J. **Driving the Dream: The History and Motivations behind Sixty Years of Automated Highway Systems in America**. Automotive History Review. Summer, 2003. pp. 4- 19.

GAGE, D.W. **UGV History 101: A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts**. Unmanned Systems Magazine. v. 13, n. 3. Disponível em: <http://www.fastattackvehicle.com/automated%20robot%20vehicles%20highlited.pdf >. Acesso em: 12 mar. 2018.

DARPA. **DARPA Grand Challenge rulebook**. 2004. Disponível em <http://www.darpa.mil/ grandchallenge05/Rules_8oct04.pdf>. Acesso em: 12 mar. 2018.

THRUN, S.; MONTEMERLO, M.; DAHLKAMP, H.; STAVENS, D.; ARON, A.; DIEBEL, J.; FONG, P.; GALE, J.; HALPENNY, M.; HOFFMANN, G.; LAU, K.; OAKLEY, C.; PALATUCCI, M.; PRATT, V.; STANG, P.; STROHBAND, S.; DUPONT, C.; JENDROSSEK, L. E.; KOELEN, C.; MARKEY, C.; RUMMEL ,C.; VAN NIEKERK, J.; JENSEN, E.; ALESSANDRINI, P.; BRADSKI, G.; DAVIES, B.; ETTINGER, S.; KAEHLER, A.; NEFIAN, A. E MAHONEY, P. **Stanley: The Robot that Won the DARPA Challenge**. Journal of Field Robotics. 2006. n. 23(9), p.661–692. Disponível em: < https://www-cs.stanford.edu/people/dstavens/jfr06/thrun_etal_jfr06.pdf > Acesso em: 12 mar. 2018

SAE. **Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.** 2016. Disponível em: https://www.sae.org/standards/content/j3016_201609/. Acesso em: 14 mar. 2018

NHTSA. **Federal Automated Vehicles Policy.** 2016. Disponível em: < https://www.transportation.gov/AV/federal-automated-vehicles-policy-september-2016>. Aceso em: 14 mar. 2018

PISSARDINI, R. S.; WEI, D. C. M. ; FONSECA JR., E. S. **Veículos Autônomos: Conceitos, Histórico e Estado-da-Arte**. Anais do XXVII Congresso de Pesquisa e Ensino em Transportes, 2013.

KAPLAN, E. D. **Understanding GPS: principles and applications**. Boston, Artech House, pp 554. 1996.

KING, A. D. **Inertial Navigation – Forty Years of Evolution**. 1998. Disponível em: < http://www.imar-navigation.de/downloads/papers/inertial_navigation_introduction.pdf > Acesso em: 07 abr. 2018

YOLE Development. **Sensors and Data Management for Autonomous Vehicles report.** 2015. Disponível em: < https://www.i-micronews.com/report/product/sensors-and-data-management-for-autonomous-vehicles-report-2015.html?Itemid=0 >. Acesso em: 24 mar. 2018

MANKAD, K.B. **An Intelligent Process Development Using Fusion of Genetic Algorithm with Fuzzy Logic.** Handbook of Research on Artificial Intelligence Techniques and Algorithms. Universiti Teknologi Petronas, 2014. pp 44-81.

KÖSE, U.; ARSLAN, A. **Chaotic Systems and Their Recent Implementations on Improving Intelligent Systems.** Handbook of Research on Novel Soft Computing Intelligent Algorithms: Theory and Practical Applications. Petronas University of Technology, 2014. pp 64-101.

RUSSEL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3ª edição. Pearson, 2009.

HAUGELAND, J. **Artificial Intelligence: The Very Idea**. MIT Press, Cambridge, Massachusetts. Ed. 1985.

BELLMAN, R. E. **An Inrrocluction to Artificial Intelligence: Can Computers Think?** Boyd & Fraser Publishing Company, San Francisco, 1978.

CHARNIAK, E; MCDERMOTT, D. **Introduction to Artijicial Intelligence**. Addison-Wesley, Reading, Massachusetts, 1985.

KURZWEIL, R. **The Age of Intelligent Machines**. MIT Press, Cambridge, Massachusetts, 1990.

POOLE, D.; MACKWORTH, A. K.; GOEBEL, R. **Computational intelligence: A logical approach**. Oxford University Press, Oxford, UK, 1998.

DIJKSTRA, E. W. **A note on two problems in connexion with graphs**. Numerische Mathematik, 1959. pp 269-271.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B**. A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. IEEE Transactions on Systems Science and Cybernetics SSC4, 1968. pp 100–107.

TIOBE. **TIOBE Index for April 2018**. 2018. Disponível em: < https://www.tiobe.com/tiobe-index/ >. Aceso em: 10 abr. 2018

UDACITY. **Inteligência Artificial para Robótica.** 2012. Disponível em: <https://br.udacity.com/course/artificial-intelligence-for-robotics--cs373>. Aceso em: 9 mai. 2018

APÊNDICE A - Código Completo A* Implementado

```
grid = [[0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0]]


heuristic = [[9, 8, 7, 6, 5, 4],
             [8, 7, 6, 5, 4, 3],
             [7, 6, 5, 4, 3, 2],
             [6, 5, 4, 3, 2, 1],
             [5, 4, 3, 2, 1, 0]]

init = [0, 0]                              # ponto de partida do agente
goal = [len(grid)-1, len(grid[0])-1]       # ponto de chegada do agente
cost = 1                                   # custo para cada movimento

delta = [[-1, 0 ],  # para cima
         [ 0, -1],  # para a esquerda
         [ 1, 0 ],  # para baixo
         [ 0, 1 ]]  # para a direita

delta_name = ['^', '<', 'v', '>']

def search(grid,init,goal,cost):    # definicao de uma nova funcao

    closed = [[0 for row in range(len(grid[0]))] for col in range(len(grid))]
    closed[init[0]][init[1]] = 1

    expand = [[-1 for row in range(len(grid[0]))] for col in range(len(grid))]
    action = [[-1 for row in range(len(grid[0]))] for col in range(len(grid))]

    path = [[' ' for row in range(len(grid[0]))] for col in range(len(grid))]

    x = init[0]             # seta a posicao inicial do agente em x
    y = init[1]             # seta a posicao inicial do agente em y

    g = 0                   # seta o valor inicial da funcao g
    f = g + heuristic[x][y] # seta o valor inicial de f

    open = [[f, g, x, y]]   # cria o vetor para guardar as componentes

    found = False   # flag que e setada quando a busca esta completa
    resign = False  # flag que e setada caso nao tenha mais no a expandir
    count = 0       # variavel de contagem de nos expandidos

    while not found and not resign: # loop para caso a busca nao
                                    #  esteja completa e ainda tenha
                                    #  nos para expandir

        if len(open) == 0:  # checa o vetor open
            resign = True   # seta a flag resign
            return 'fail'   # retorna mensagem de 'fail'
```

```
54          else:               # caso nao tenha erro no vetor, continua a rodar
55
56              open.sort()         # ordena o vetor open
57              open.reverse()      # inverte a ordenacao
58              next = open.pop()   # next recebe o valor extraido de open
59              f = next[0]         # f recebe valor de next
60              g = next[1]         # g recebe valor de next
61              x = next[2]         # x recebe valor de next
62              y = next[3]         # y recebe valor de next
63
64              expand[x][y] = count    # o no expandido recebe sua contagem
65              count += 1              # e acresido em 1 o valor de nos expandidos
66
67              if x == goal[0] and y == goal[1]:   # condicao para caso o
68                                                  # objetivo seja atingido
69                  found = True    # seta a flag found
70
71              else:   # caso ainda nao tenha encontrado o objetivo, continuar
72
73                  for i in range(len(delta)): # loop para "movimentar" o agente
74                      x2 = x + delta[i][0]    # x2 recebe o valor de x adicionado
75                                               # do movimento especifico
76                      y2 = y + delta[i][1]    # y2 recebe o valor de x adicionado
77                                               # do movimento especifico
78
79                      if x2>=0 and x2<len(grid) and y2>=0 and y2<len(grid[0]):
80                          # condicao para identificar se a proxima posicao
81                          # esta dentro do grid
82
83                          if closed[x2][y2] == 0 and grid[x2][y2] == 0:
84                              # condicao para verificar se o no relativo
85                              # a posicao ainda nao foi expandido
86
87                              g2 = g + cost                # g recebe o g atual somado do
88                                                           # custo da operacao
89                              f2 = g2 + heuristic[x2][y2]  # g recebe o g atual somado do
90                                                           # custo da operacao
91                              open.append([f2, g2, x2, y2]) # adiciona mais uma posicao no vetor,
92                                                           # com os novos valores calculados
93                              closed[x2][y2] = 1  # sinaliza que o no ja foi expandido
94                              action[x][y] = i    # sinaliza qual foi a acao tomada na posicao
95
96      x = 0   # seta a posicao x para 0
97      y = 0   # seta a posicao y para 0
98
99      while x!=goal[0] or y!=goal[1]: # loop para construir o caminho
100                                     # percorrido com as acaes salvas
101         x2 = x + delta[action[x][y]][0] # x2 recebe posicao + movimento
102         y2 = y + delta[action[x][y]][1] # y2 recebe posicao + movimento
103         path[x][y] = delta_name[action[x][y]] # variavel path salva as acoes
104         x = x2 # x recebe x2
105         y = y2 # y recebe y2
106
107     path[goal[0]][goal[1]] = '*' # no ponto de chegada inserir caractere '*'

107     path[goal[0]][goal[1]] = '*' # no ponto de chegada inserir caractere '*'
108
109     return expand # retorna o caminho otimo
110
111 print search(grid,init,goal,cost) # chama a funcao definida como search
```

APÊNDICE B - Algoritmo de Dijkstra Implementado Para Comparativo

```
grid = [[0, 1, 0, 0, 0],
        [0, 1, 0, 0, 0],
        [0, 1, 0, 0, 0],
        [0, 1, 0, 0, 0],
        [0, 0, 0, 1, 0]]
init = [0, 0]
goal = [len(grid)-1, len(grid[0])-1]
cost = 1

delta = [[-1, 0 ],
         [ 0, -1],
         [ 1, 0 ],
         [ 0, 1 ]]

delta_name = ['^', '<', 'v', '>']

def search(grid,init,goal,cost):
    closed = [[0 for row in range(len(grid[0]))] for col in range(len(grid))]
    closed[init[0]][init[1]] = 1

    path = [[' ' for row in range(len(grid[0]))] for col in range(len(grid))]

    action = [[-1 for row in range(len(grid[0]))] for col in range(len(grid))]

    x = init[0]
    y = init[1]
    g = 0

    k = 0
    expand = [[-1 for row in range(len(grid[0]))] for col in range(len(grid))]

    open = [[g, x, y]]

    found = False
    resign = False

    while not found and not resign:
        if len(open) == 0:
            resign = True
            return 'fail'
        else:
            open.sort()
            open.reverse()
            next = open.pop()
            x = next[1]
            y = next[2]
            g = next[0]

            expand[x][y] = k
            k += 1

            if x == goal[0] and y == goal[1]:
                found = True
            else:
                for i in range(len(delta)):
```

```
                    x2 = x + delta[i][0]
                    y2 = y + delta[i][1]
                    if x2 >= 0 and x2 < len(grid) and y2 >=0 and y2 < len(grid[0]):
                        if closed[x2][y2] == 0 and grid[x2][y2] == 0:
                            g2 = g + cost
                            open.append([g2, x2, y2])
                            closed[x2][y2] = 1
                            action[x][y] = i
    x = 0
    y = 0
    while x!=goal[0] or y!=goal[1]:
        x2 = x + delta[action[x][y]][0]
        y2 = y + delta[action[x][y]][1]
        path[x][y] = delta_name[action[x][y]]
        x = x2
        y = y2
    path[goal[0]][goal[1]] = '*'
     return expand

print search(grid,init,goal,cost)
```